Building Reliable Metaclassifiers for Text Learning

Paul N. Bennett

May 2006

CMU-CS-06-121

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by the Defense Advanced Research Agency (DARPA) under Contract No. NBCHC030029, SRI International under Subcontract No. 55000691, the National Imagery & Mapping Agency (NIMA) under Contract No. NMA401-02-C0033, and the National Science Foundation under Grant Nos. IIS-9982226, IIS-9873009, and IIS-9988084. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either express or implied, of any sponsoring institution, the U.S. government, or any other entity.

Keywords: text classification, classifier combination, metaclassifiers, reliability indicators, reliability, multiple models, locality, combining estimates, calibration

This is dedicated first to my parents whose pride in my work has been as uplifting as their supporting words, but most importantly, this is dedicated to my loving wife, Tina. Your support made my long nights bearable; I hope my absence didn't make yours too unbearable.

Abstract

Appropriately combining information sources to form a more effective output than any of the individual sources is a broad topic that has been researched in many forms. It can be considered to contain sensor fusion, distributed data-mining, regression combination, classifier combination, and even the basic classification problem. After all, the hypothesis a classifier emits is just a specification of how the information in the basic features should be combined.

This dissertation addresses one subfield of this domain: leveraging locality when combining classifiers for text classification. Classifier combination is useful, in part, as an engineering aid that enables machine learning scientists to understand the difference in base classifiers in terms of their local reliability, dependence, and variance — much as higher-level languages are an abstraction that improves upon assembly language without extending its computational power. Additionally, using such abstraction, we introduce a combination model that uses inductive transfer to extend the amount of labeled data that can be brought to bear when building a text classifier combination model.

We begin by discussing the role calibrated probabilities play when combining classifiers. After reviewing calibration, we present arguments and empirical evidence that the distribution of posterior probabilities from a classifier will give rise to asymmetry. Since the standard methods for recalibrating classifiers have an underlying assumption of symmetry, we present asymmetrical distributions that can be fit efficiently and produce recalibrated probabilities of higher quality than the symmetrical methods. The resulting improved probabilities can either be used directly for a single base classifier or used as part of a classifier combination model.

Reflecting on the lessons learned from the study of calibration, we go on to define local calibration, dependence, and variance and discuss the roles they play in classifier combination. Using these insights as motivation, we introduce a series of reliability-indicator variables which serve as an intuitive abstraction of the input domain to capture the local context related to a classifier's reliability.

We then introduce the main methodology of our work, *STRIVE*, which uses metaclassifiers and reliability indicators to produce improved classification performance. A key difference from standard metaclassification approaches is that reliability indicators enable the metaclassifier to weigh each classifier according to its local reliability in the neighborhood of the current prediction point. Furthermore, this approach empirically outperforms state-of-the-art metaclassification approaches that do not use locality. We then analyze the contributions of the various reliability indicators to the combination model and suggest promising features to consider when redesigning the base classifiers or new combination approaches. Additionally, we show how inductive transfer methods can be extended to increase the amount of labeled training data available for learning a combination model by collapsing data traditionally viewed as coming from different learning tasks.

Next, we briefly review online-learning classifier combination algorithms that have theoretical performance guarantees in the online setting and consider adaptations of these to the batch settings as alternative metaclassifiers. We then present empirical evidence that they are weaker in the offline setting than methods which employ standard classification algorithms as metaclassifiers, and we suggest future improvements likely to yield more competitive algorithms.

Finally, the combination approaches discussed are broadly applicable to classification problems other than topic classification, and we emphasize this with experiments that demonstrate *STRIVE* improves performance of action-item detectors in e-mail — a task where both the semantics and base classifier performance are significantly different than topic classification.

Acknowledgments

First, I would like to thank the five most influential forces in my academic life — Drs. Ron Richards, Bob Causey, Ray Mooney, Sue Dumais, and Jaime Carbonell — for crafting my view of scientific inquiry, instilling me with computational curiosity, and equipping me with the tools to apply the former to the latter. Additionally, I wish to acknowledge the efforts of each of my committee members, Drs. Jaime Carbonell, John Lafferty, Tom Mitchell, Sue Dumais, and Eric Horvitz, for their helpful feedback and corrections. Their contributions have been invaluable, and any remaining errors are mine alone. Finally, I most heartily thank Francisco Pereira and Kevyn Collins-Thompson whose ability to find spare CPU cycles is only surpassed by their friendship.

Contents

List of Figures

List of Tables

Notation

$\mathcal{F}$	The set of all features in a classification problem.
$\mathcal{X}$	The input domain of the training examples. Typically, $\mathcal{X} \subseteq \mathbb{R}^{\lvert\mathcal{F}\rvert}$
$\mathcal{Y}$	The set of classes for a classification problem. Generally, we will refer to binary classification tasks where $\mathcal{Y} = \{0, 1\}$ or $\mathcal{Y} = \{-1, 1\}$.
$\mathcal{C}$	The set of all base classifiers.
$\mathcal{C}_i$	A particular classifier i.
$\langle x_i, y_i \rangle$	A labeled example where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$
$c(x)$	The class of example x where $x \in \mathcal{X}$.
$\mathcal{D}$	The true distribution of examples.
$P(c(x) = y \mid x)$	The "true" posterior or conditional distribution of the class of the example, $c(x)$, given x. This will often be abbreviated $P(y \mid x)$
$p(x)$	The "true" density of unlabeled examples.
$p(x, y)$	The "true" joint density of labeled examples.
$\hat{P}_A, \hat{p}_A$	Model A's estimate of a probability distributioin P or density p, respectively.
$\hat{y}_i$	Classifier $\mathcal{C}_i$'s prediction about the class of an example where the particular example is clear in context.
π_i	Classifier $\mathcal{C}_i$'s posterior probability distribution over the classes of an example. That is, $\pi_i(c, x) = \hat{P}_{\mathcal{C}_i}(c(x) = y \mid x)$. In the context of a binary classification problem, we will often use π_i to denote the posterior of the positive class, *i.e.*, $\hat{P}_{\mathcal{C}_i}(c(x) = 1 \mid x)$
λ_i	Classifier $\mathcal{C}_i$'s posterior log odds of the class of an example. That is, $\lambda_i(c, x) = \log \frac{\hat{P}_{\mathcal{C}_i}(c(x)=y\mid x)}{1-\hat{P}_{\mathcal{C}_i}(c(x)=y\mid x)}$. In the context of a binary classification problem, we will often use $\lambda_i(x)$ or simply λ_i to denote the log odds of the positive class, *i.e.*, $\log \frac{\hat{P}_{\mathcal{C}_i}(c(x)=1\mid x)}{\hat{P}_{\mathcal{C}_i}(c(x)\neq 1\mid x)}$

Other Definitions

Logit	The (natural) log of a probability divided by its complement. $\log \frac{p}{1-p}$
Odds	The quotient of a probability of an event and its complement. $\frac{p}{1-p}$
Log Odds	The log of the odds of an event. This is equivalent to the logit for a two-class problem. $\log \frac{p}{1-p}$. Also, the log of the probabilities for two mutually exclusive events, $\log \frac{p_i}{p_j}$ (see note below).
Odds Ratio	The quotient of the odds ratios of two different events. $\frac{p(1-r)}{(1-p)r}$
Log Odds Ratio	The log of an odds ratio. Sometimes called log odds when no risk of confusion with the above definition. $\log \frac{p(1-r)}{(1-p)r}$

A note on *Log Odds* Given n probabilities p_i for n mutually exclusive and exhaustive events such that $\sum_{i=1}^{n} p_i = 1$, it is unclear what the established terminology is for the quantity $\log \frac{p_i}{p_j}$. When $n = 2$ this is just the log of the odds of event i, but for $n > 2$ this does not reduce. Some (p. 96 [HTF01]) refer to this quantity as log odds, log odds ratios, and logits even though it does not reduce to any of the above forms. Furtherermore, when $n = 2$ many parts of the literature say “Log Odds Ratio” when meaning “Log Odds” as defined above. We keep with the looser terminology which is more prevalent in the literature. When clarity is necessary, we specify the intended meaning.

Chapter 1

Overview

1.1 Introduction

A text classification algorithm uses example documents that have been tagged with classes by an authority[1] to learn a model that, with high accuracy, can automatically predict the class the authority would have assigned to future documents. In cases like topic classification (Figure 1.1) where each example can belong to multiple topics, the problem is usually reduced to a series of binary classification tasks, *Corporate Acquisitions* vs. *not Corporate Acquisitions, Earnings* vs. *not Earnings, etc.*

With the surge in digital text media, text classification has become increasingly important. Text classification techniques can assist in junk e-mail detection [SDHH98], allow medical doctors to more rapidly find relevant research [HBLH94], aid in patent searches [Lar99], improve web searches [CD00], and serve as a backend in a multitude of other applications. The interested reader should see Sebastiani [Seb02] for a survey of recent applications of machine learning to text classification.

Decision Trees, kNN, SVMs, language models, and naïve Bayes are a few of the classification algorithms that have been developed [HMS66, Fri77, BFOS84, CH67, Vap00, CV95, MK60, Abr63] and later used by researchers to address the problem of text classification [LR94, ADW94b, MLW92, Joa02, MN98, Lew92b]. Each of these models generally are designed using a different set of assumptions regarding the data. However, none of these algorithms dominate all text classification problems.[2] Furthermore, even when one class-

[1]It doesn't matter for our purposes whether this "authority" is a single person, a committee, or any other entity. The only stipulation is that the labeling is consistent in the sense that it is the same authority labeling the training documents and future documents.

[2]Although SVMs show perhaps the most robust behavior across a span of text classification problems.

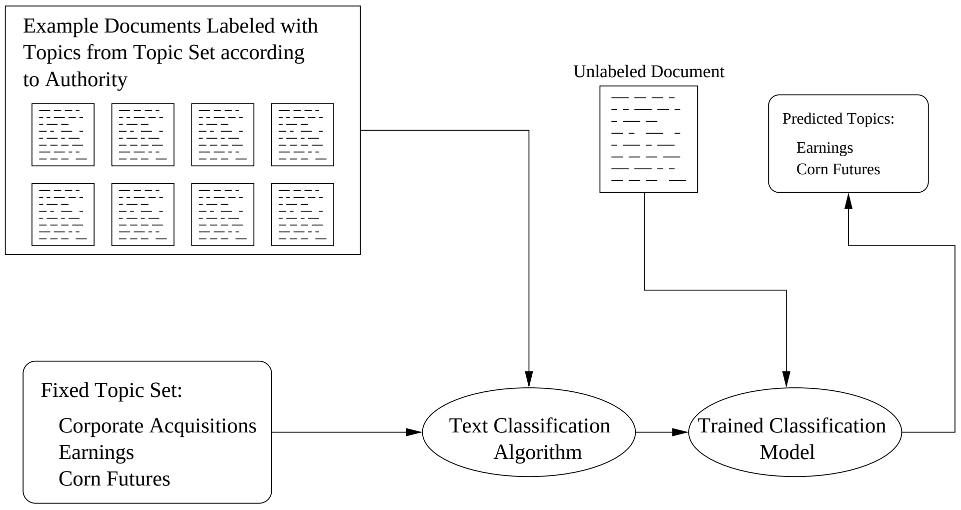

Example Documents Labeled with Topics from Topic Set according to Authority
Unlabeled Document
Predicted Topics:
Earnings
Corn Futures
Fixed Topic Set:
Corporate Acquisitions
Earnings
Corn Futures
Text Classification Algorithm
Trained Classification Model

Figure 1.1: A typical text classifi cation problem. A text classifi cation algorithm takes as input a set of example documents. Each document is labeled by an authority with a set of classes (here the *topics*). The algorithm uses these examples to construct a model that with high accuracy can predict the topics the authority would have assigned to future documents. This particular type of text classifi cation problem is called *topic classifi cation*.

ification algorithm significantly outperforms another for a given classification problem, it is rarely the case that the worse classifier's errors are a superset of the better. This fact has long motivated the desire to combine models in order to obtain better, or more robust, overall performance. Schemes to do this have varied widely, from simple voting to methods for including unlabeled examples.

Appropriately combining information sources to form a more effective output than any of the individual sources is a broad field that has been researched in many forms. It can be considered to contain sensor fusion, distributed data-mining, regression combination, classifier combination, and even the basic classification problem; after all, the hypothesis a classifier emits is just a specification of how the information in the basic features should be combined.

Problems that arise in several situations motivate combining multiple learners. For example, it may not be possible to train using all the data because data privacy and security concerns prevent sharing the data. However, a classifier can be trained over different data subsets and the predictions they issue may be shared. In other cases, the computation burden of the base classifier may motivate classifier combination. When a classifier with a nonlinear training or prediction cost is used, computational gains can be realized by partitioning the data and applying an instance of the classifier to each subset. In other situations, combining classifiers can be seen as a way of extending the hypothesis space or relaxing

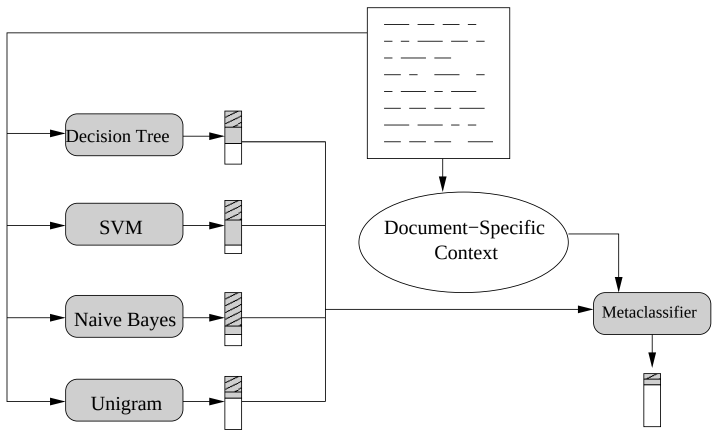


Figure 1.2: Schematic characterization of reliability-indicator methodology. The output of the classifi ers is a graphical representation of a distribution over possible class labels.

the bias of the original base classifier. Boosting decision stumps [SFBL98, BK99], cascade generalization [Gam98a, Gam98b], and stacking [Wol92] can all be seen as methods where the hypothesis space of the combiner can fit a more general class of functions than the input base classifiers. Finally, in many different situations, classifier combination can be used as a way to balance the strengths and weaknesses of a set of classifiers in order to achieve increased generalization performance.

This work approaches classifier combination with increased performance as the primary motivation, but the same methods are applicable for any of the above purposes. In order to do so, we attempt to exploit the fact that the models learned by different classification algorithms have different error profiles. This is done by defining data-dependent characteristics that can be tied to the likelihood a model predicts well in the context established by the current example. We focus on the local reliability, variance, and dependence of the classifiers as the key data-dependent characteristics for classifier combination. Therefore, this work is similar in flavor to Kahn's [Kah04] without making his distributional assumptions for classifier outputs which often do not hold in practice. In order to capture these characteristics, we define a set of *reliability indicators* that we argue are tied to these characteristics for text classification problems. The general approach is depicted in Figure 1.2.

A secondary goal of this work is to provide a basis for understanding the interactions of a set of classifiers. We argue this enables machine learning practitioners to more readily choose between the trade-offs when choosing which methods to apply to a problem. This abstraction is key when using classifier combination to mitigate data privacy and security problems. Concerned parties need only to verify that they can share both the predictions and indicators rather than the data itself.

Classifier combination is ultimately a *hard* problem where, as in other problems, the optimal combination is rarely computable. Even weaker results, such as a combination scheme that *always* outperforms its base input classifiers, have been shown to be theoretically unattainable [DHS01, Wol95]. Therefore, our goal will be to make substantial gains when possible and to be within a small performance difference at other times.

Finally, our work shows that not only can these data-dependent characteristics be used to construct a more effective classifier, but since they behave similarly across a set of related problems, the data-scarcity problem can be somewhat alleviated during meta-learning by sharing data via inductive transfer.

In order to situate the reader, the remainder of this chapter gives key examples to motivate the gains and elucidate the challenges of classifier combination before finally making a more formal thesis statement and stating our evaluation criteria.

1.2 Worse Performance Does Not Mean Obsolete

Although theoretical results [DHS01, Wol95] indicate there is no a priori choice of algorithm which will perform the best over all problems, experience has shown that some algorithms can dominate large classes of problems. For example, most acknowledge that SVMs with a linear kernel will perform at least as well as and usually outperform most known methods in topic classification. As a result, Machine Learning researchers often attempt to understand how well an algorithm (*e.g.* SVM) fits a set of problems (*e.g.* topic classification) by empirically evaluating algorithms.

In contrast, even when an algorithm outperforms another algorithm across a problem set, combining the algorithms can lead to better results than either alone. Thus, while good empirical performance gives evidence that an algorithm's assumptions match the underlying domain structure well, improvement from combination methods provides weak evidence that the base classifiers are only capturing subdomain structure and failing to entirely capture the learnable structure of the problem. There are many concrete situations where weak classifiers can help improve the performance of a strong classifier. In the following, we construct several examples which illustrate the potential gains from classifier combination.

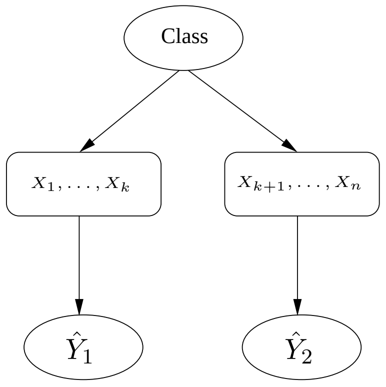


Figure 1.3: Influence diagram for classifi ers built using two conditionally independent views of the data.

1.2.1 Examples from Synthetic Data

First, consider the case where there are two views of the data. For example, when classifying a web page, we might represent it as the text contained on the web page itself, or we could view it as the text on the pages linked to by the web page. In the ideal case, these two views would be independent given the class label. Figure 1.3 shows an influence diagram for this simplified case. The generative process for this diagram can be thought of as follows. First, the class, c, is chosen according to some prior, $P(C)$. Then a distribution, $V_{1,c}$, which governs the first view generates the features $X_1, \ldots, X_k$. A second distribution, $V_{2,c}$, then generates the features $X_{k+1}, \ldots, X_n$. The first classifier, $\hat{Y}_1$, uses only the first feature set to make its prediction while the second classifier $\hat{Y}_2$ uses the second feature set.

To make this concrete, consider a binary classification task where the classes are $\{-1, 1\}$ and we have only one feature per view. Each feature will be generated by a normal distribution. Let $V_{1,c} = N(5c, 3)$, $V_{2,c} = N(c, 1)$, $\hat{Y}_1 = \mathrm{argmax}_{c \in C} P(c|X_1)$, and $\hat{Y}_2 = \mathrm{argmax}_{c \in C} P(c|X_2)$. Assuming that each class is equally likely, $P(-1) = P(1) = 0.5$. Then the first classifier will have an error of 4.78% and the second an error of 15.87%. Even though the first classifier far outperforms the second, it is obvious in this case that each classifier has information which can lead to a better combined decision. Now, how good can the combination of these classifiers be?

At this point, the astute reader will have already taken note that $\hat{Y}_1$ and $\hat{Y}_2$ are making predictions in accordance with the correct posterior over their respective feature sets, but this does not mean the classifiers have access to the actual probability functions, $P(c|X_1)$ and $P(c|X_2)$, just that they are correct with respect to the decision threshold. Assuming that we have two such classifiers and the combination function only has access to $\hat{Y}_1$ and $\hat{Y}_2$, then it is well-known that the best we can do can be expressed as a linear combination

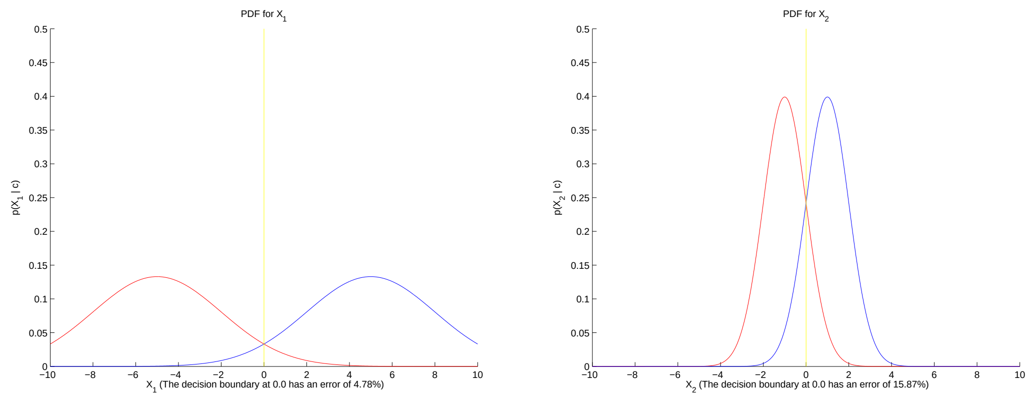


Figure 1.4: (a) The pdf for X_1 as well as the decision boundary used by optimal classifi er $\hat{Y}_1$; (b) The pdf for X_2 as well as the decision boundary used by optimal classifi er $\hat{Y}_2$.

of the classifier predictions[3], specifically $sign\left(\log \frac{P(+)}{P(-)} + \sum w_{i,\hat{y}_i}\hat{y}_i\right)$, where the weights are a function of the classifiers' prediction accuracies:

$$w_{i,\hat{y}_i} = \begin{cases} \log \frac{P(\textit{Correct Positive}_i)}{P(\textit{False Positive}_i)} + \log \frac{P(-)}{P(+)} & \text{if } \hat{y}_i = +1 \\ \\ \log \frac{P(\textit{Correct Negative}_i)}{P(\textit{False Negative}_i)} + \log \frac{P(+)}{P(-)} & \text{if } \hat{y}_i = -1. \end{cases} \tag{1.1}$$

With a class prior of 0.5, our example classifiers have a symmetric distribution of false positives and false negatives. Therefore, the best combination of these two classifiers would still have an error rate of 4.78%, equal to that of the best classifier. While the error rate would improve if we had more conditionally-independent classifiers, the problem here is that the weaker classifier cannot overpower the stronger classifier because there is no information about whether an example lies near the stronger classifier's decision threshold.

On the other hand, if the classifiers actually issue the true probability estimates, $P(c|X_1)$ and $P(c|X_2)$, along with their class predictions, then the optimal combination now has an error rate of 2.63%. This gives nearly a 50% reduction in error over the best classifier by combining it with a classifier that's "3 times worse" according to error! The optimal combination gains for this example in other contexts fall between these two extremes. For example, a more reasonable case is when the classifiers produce probability estimates that lie on the correct side of the decision threshold with respect to their feature set, but the probabilities themselves are not correct. Of course, in the most common case in practice,

[3]For the two class case, it is possible to write a single weight for each classifier rather than making the weight a function of the classifier's prediction. The form is equivalent but more tedious to write out, and the interpretation is less obvious.

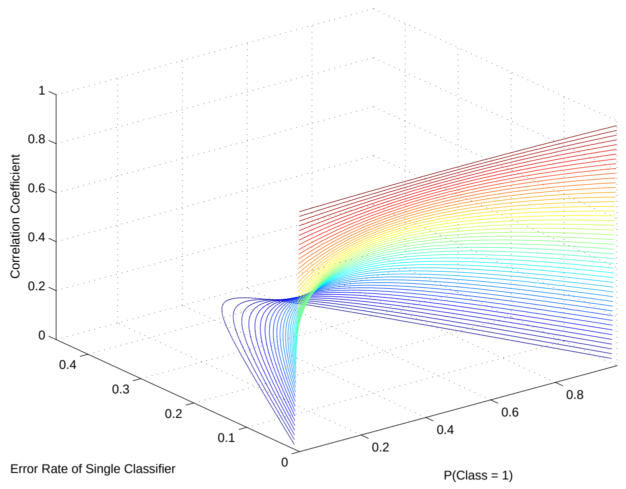


Figure 1.5: The correlation of conditionally-independent classifi ers is largely determined by their error rates and the class prior. This graph is generated with two classifi ers whose error rates are equal.

the classifiers produce probabilities that are not accurate nor do they always produce a consistent decision.

Taken together, this discussion highlights that one of the challenges in classifier combination lies in dealing with the output type of the classifier — whether they are probabilities, scores, or predictions — as well as their quality.

Finally, it is worth noting the subtle point that analyzing the correlation alone will not indicate whether classifiers have conditionally-independent predictions. In our particular example, the Pearson correlation coefficient between the two classifiers' predictions shows a borderline strong correlation of 0.6174 simply because both classifiers are relatively accurate and therefore tend to agree. If we change the accuracy of $\hat{Y}_2$ by changing the standard deviation of the second feature to 0.6, the correlation coefficient jumps to 0.8180. However, regardless of the correlation, in both cases the classifiers provide distinctly different information because they are conditionally-independent given the class.

In these examples, it is the class prior emission probability and the error rate of the classifiers that largely determine how strongly correlated the predictions are. To remind the reader, the Pearson correlation coefficient is $\rho(\hat{Y}_1, \hat{Y}_2) = \frac{Cov(\hat{Y}_1, \hat{Y}_2)}{\sigma_{\hat{Y}_1}\sigma_{\hat{Y}_2}}$. Assuming the classifiers are relatively accurate, a class prior close to 0.5 will result in the predictions having a large standard deviation (close to the max of 1) which will require a large number of disagreements in the classifiers' predictions to achieve a low correlation. This is unlikely by chance if they are both accurate. As the class prior goes to zero or one, the number of disagreements needed to drive the correlation to zero shrinks rather rapidly. Figure 1.5 will help the reader visualize this for the case where $\hat{Y}_1$ and $\hat{Y}_2$ are constrained to have the same accuracy.

At this point, the reader should be convinced that even strongly correlated predictions between classifiers are not sufficient to dismiss the hypothesis that a combination will outperform either classifier.

1.2.2 Diversity and Accuracy in Real Datasets

The idealized examples presented in section 1.2.1 highlight two necessary conditions for benefitting from classifier combination. First, the base classifiers must be diverse in that they give different predictions. In the idealized example, this diversity came because the classifier predictions were conditionally-independent given the class. Second, the classifiers must be fairly accurate in addition to being diverse. Clearly, diversity can be easily obtained by random predictions unless this second constraint is enforced.

In the context of this dissertation, we use standard classification algorithms to obtain both diversity and accuracy. These algorithms include Decision Trees, kNN, SVMs, simple language models, and naïve Bayes. These algorithms have been empirically evaluated in a variety of text classification problems, and it is known that they are anywhere from reasonably accurate to state-of-the-art. It is less clear that the predictions they issue should be diverse especially when trained over the same data.

We argue that the fundamentally different assumptions underlying the algorithms give rise to models that fit different parts of the data more or less well. Finding the conditions under which these classifiers can be combined to obtain better performance is therefore a precursory step to determining how and why any single classifier's assumptions are not sufficient to fully model the classification task at hand. We have chosen these algorithms not only because they are known to perform well but also because they are different types of classifiers along several different dimensions. The SVM, language model, and naïve Bayes algorithms we investigate produce a linear decision boundary where the kNN and Decision Tree classifiers are non-linear classifiers. In contrast, the language model and naïve Bayes algorithms we employ are generative classifiers where the SVM, kNN, and Decision Tree classifiers are discriminative.

In addition to these foundational motivations for our choice in base classifiers, empirical evidence bears out the fact that their error profiles differ in a number of ways. When viewed using ROC curves, there are typically cross-overs in the curve indicating that no single model is appropriate for all cost functions. Finally, an optimistic estimate of the performance ceiling for combination, such as reducing error to only those examples where all of the base classifiers are wrong, results in an error rate far better than the best classifier.

1.3 Thesis Statement

While classifier combination techniques comprise a broad field, this dissertation will focus solely on how context-sensitive metaclassification techniques can be used to improve generalization performance.

An initial attempt to address this issue might start by requiring the classifiers to output a probability distribution over possible class labels then combine them via a simple strategy, such as a constant-weighted linear sum. While this seems like an inviting avenue to pursue, obtaining "good" probability estimates can be problematic. We review our early investigations [Ben00, Ben02] that demonstrated how to convert a classifier's scores to probability estimates that are of quality better than or comparable to other methods for calibrating classifier scores. While this is a step in the right direction, it does not address one of the primary challenges of classifier combination — namely estimating the dependencies of the classifier outputs. Furthermore, it fails to take advantage of the fact that the reliability of a classifier's predictions can vary across the input space. Additionally, the correlation between two classifiers' predictions may vary locally as well — in some areas, showing high dependence and in others being largely independent.

Of course, the characterization of context is the operative factor in the distinctions the metalevel classifier can make. Other methods that have tried to leverage context have typically used only the classifier outputs or the outputs and all of the base features as context [TW99, Gam98a, Gam98b, TT95, MP99]. In order to make finer local discriminations, we wish to use more context than the classifier scores, but because of the high dimensionality of text, using all the base features as a representation of context is a poor choice because of the amount of data needed to accurately learn such a model. Following the work that inspired this model [TH00], we introduce a set of *reliability-indicator variables* that are a low-dimensional rich abstraction of the discriminatory context provided by a document for learning.

In contrast to Toyama & Horvitz's work [TH00], we give formal definitions for the local *dependence*, *reliability*, and *variance* of a set of classifiers, and then we define indicator variables that are either direct or indirect approximations of these statistics. Other combination work [Kah04] relates the combination weights to these quantities when the classifier's log-odds predictions are assumed to follow a certain generative form. Our work can be seen as generalizing this framework. Even when a generative form is not assumed, these quantities are at least necessary. Although, in practice, we can rarely compute them directly and accurately. Therefore, the fundamental assumptions underlying the metaclassifier approach are: (1) Since we know these quantities are necessary for classifier combination, we will gain by reducing the dimension of the metaclassifier space from the original input space to approximations of these quantities; (2) Assuming these quantities are sufficient for classifier combination, it is not necessary to explicitly know the "locality"[4] of an example — documents that are similar in the metaclassifier space will have similar combination rules since these statistics are assumed sufficient.

Furthermore, we generalize the reliability-indicator characterization of context in a way that enables using labeled data from separate learning tasks to learn an improved combination policy across all tasks. This can be done if we treat the metaclassifier as an abstraction from discriminating a specific topic (*e.g., Corporate Acquisitions* vs. *not Corporate Acquisitions*) to the problem of discriminating topic membership in general (*i.e., In-Topic* vs. *Out-of-Topic*). The base-level classifiers that are trained on a particular topic are used as the representation of topic-specific knowledge, while the metaclassifier provides information about how to leverage context across topic-classification in general. Such an extension is only possible if we generalize the reliability indicators away from linkages to the precise words in a document. Consider when "shares" occurs in a document in the *Corporate Acquisitions* discrimination task and "corn" occurs in a document in the *Corn Futures* discrimination task; one simple task invariant representation of context at the metalevel might transform both of these to: *Is the word with maximum mutual information for the current task present in this document?* This representation enables the metaclassifier to use information about how document-specific context influences topic discrimination across a wide variety of text classification tasks. The success of this abstracted model depends critically upon our ability to find a "normalized" representation that captures how topic classification decision boundaries co-vary with the statistical properties of language. We present empirical evidence that this approach can, in fact, succeed — increasing the amount of labeled data available to build the metaclassifier by pooling it. We discuss possible issues that might arise when conglomerating the data and offer solutions to these practical problems. Finally, since the empirical evaluations of this model have focused primarily upon how *topic* classification tasks relate to language-use distributions, we demonstrate how it can be

[4]We refer to a combination method as using "locality" if the algorithm can induce a model that cannot be expressed as a constant-weighted linear combination of a classifier's predictions, probabilities, scores, or log odds. Thus both the representation and the algorithm determine whether an approach uses locality. For example, a *linear* SVM metaclassifier applied to the log-odds of a classifier's predictions does not use locality, but a decision tree algorithm, which can learn a non-linear function of the classifier's outputs, uses locality.

extended to other domains of text classification as well, specifically *action-item detection* in e-mail documents.

Throughout all of the work, we argue the following thesis: **Context-dependent combination procedures provide an effective way of combining classifiers that are generally superior to constant-weighted linear combinations of the classifier's estimates of the posterior or log-odds. Furthermore, context can be leveraged in text classifier combination via an abstraction of the local reliability, dependence, and variance of the base classifier outputs. Finally, these abstractions help identify opportunities for data re-use that can be employed to significantly improve classification performance.**

1.4 Criteria for Evaluation

As discussed earlier, it is not possible to construct a metaclassifier that always outperforms its base classifiers. As a demonstration of the suitability of these methods for text classification though, we set the goal of statistically significantly outperforming the current state-of-the-art base classification methods over several standard text classification corpora. In addition, to prevent overtuning to specific corpora, we have chosen the corpora for their breadth and have completely withheld performing experiments on one corpus until the final stages of this dissertation. Furthermore, since we argue that our representation of context is key, we will empirically demonstrate that these methods outperform simple constant-weighted combinations of the classifier outputs in some corpora and, in the remaining ones, achieve a statistically negligible difference.

1.5 Roadmap to This Document

The remainder of the document is laid out as follows. First, we describe related work. From there, we discuss why obtaining quality probability estimates from a classifier based solely on its output is problematic and introduce improved methods that rely on asymmetry. Using the same framework of analysis, we motivate and define the local reliability, dependence, and variance of a classifier. Based on these insights, we lay out the set of reliability indicators we use, show situations where they can be helpful, and discuss the computational and practical implications for computing them. Before delving into the key contributions of this dissertation, we briefly review our evaluation methodology, the implementation details of the base classifiers, and a variety of performance measures that can be used to evaluate the effectiveness of text classifiers. Then, we describe the STRIVE framework, which uses a standard classifier as the metaclassifier and the reliability-indicator representation to build a more effective classifier. We follow this with a description of how the characterization of context used in STRIVE can be generalized to build a domain-level metaclassifier that increases the pool of data that can be used for building a single model. In the next chapter we consider alternative metaclassifiers based on online learning with performance guarantees, conduct an empirical analysis of them, and discuss why these methods do not currently yield performance comparable to offline metaclassification approaches. In the following chapter, we demonstrate that these variables and representations are applicable to other text classification problems such as e-mail action-item detection. Finally, we summarize our contributions and highlight important directions for future work.

Chapter 2

Related Work

Appropriately combining information sources to form a more effective output than that of any of the individual sources is a broad topic that has been researched in many forms. The challenges of integrating information have gone under the labels of diagnosis [HBH88], pattern recognition [DHS01], sensor fusion [Kle99], multistrategy learning [MT94], distributed data mining [KC00], and a variety of ensemble methods [Die00]. Diagnosis centers on identifying disorders from multiple pieces of evidence, such as reasoning about probability distributions over a patient's diseases from a set of symptoms and test results. Pattern recognition and sensor fusion typically address challenges with integrating information from multiple modalities (*e.g.*, auditory and visual) while distributed data mining addresses how results retrieved from distinct training data sets can be unified to provide one coherent view to the user. Multistrategy learning methods have focused primarily on combining methods from different paradigms (*e.g.*, abductive and inductive methods). Ensemble methods are methods that first solve a classification or regression problem by creating multiple learners that each attempt to solve the task independently, then use a procedure specified by the particular ensemble method for selecting or weighting the individual learners. Ensemble methods include such examples as Bayesian averaging [Lea78, HMRV98], bagging [Bre96], boosting [Sch90, Fre95, FS97], stacking [Wol92], cascade generalization [Gam98a, Gam98b], hierarchical mixture of experts [JJ94], and this dissertation.

This chapter presents a sampling of the key works in the literature that are related to this work.[1] First, we review the two primary types of ensemble methods: those that combine different models obtained from the same classification algorithm (homogeneous ensembles) and those that combine models obtained from different classification algorithms (het-

[1]Additional work that is relevant to particular sections but not to the core focus of this work is discussed in the appropriate chapter.

erogeneous ensembles). Our proposed work focuses on heterogeneous ensembles. We then specifically highlight combination methods that have employed some notion of locality as well as specific applications of combination methods to text problems. Since calibration is central to our exposition of the roles of local reliability, dependence, and variance in classifier combination, we also conduct a brief survey of previous attempts to improve probability estimates or obtain calibrated estimates from a single classifier. Finally, we conclude with a discussion of the *No Free Lunch* theorem — a well-known negative theoretical result regarding classification and classifier combination.

2.1 Homogeneous Ensembles

Homogeneous ensembles are combination methods that combine different models obtained from multiple runs of the same classification algorithm. The models may differ for a variety of reasons. For example, when combining neural networks, each model instance in the ensemble could be the result of training after a random initialization of the model weights [HS90][2]; thus, each training run might settle on a different local error minimum. Alternatively, for decision trees, each tree could be obtained by training over a different sample obtained by sampling randomly with replacement from the training data [Bre96]. Homogeneous ensemble methods include Bayesian averaging, bagging, boosting, and hierarchical mixture of experts.

The primary focus of many homogeneous ensembles is to account for model parameter uncertainty that results from noise in the data and having estimated the model parameters with finite data. Bayesian averaging works directly with this concept and weights each hypothesis (possible model) by its probability of being the correct hypothesis according to a chosen prior distribution. While Bayesian averaging is one of the oldest developed combination methodologies [Lea78], it has only recently become computationally feasible to deal with large or possibly infinite hypothesis spaces via sampling or other techniques. Hoeting *et al.* [HMRV98] study and give a survey of recent approaches to Bayesian averaging.

Model uncertainty can also be thought of as the variance in the model parameters if different training sets of the same size were drawn from the same underlying distribution an infinite number of times. Breiman [Bre96] introduces bagging (bootstrap aggregation) from this approach to model uncertainty. A *bootstrap sample* of the same size as the training set

[2]Merz's [Mer98] definition of "homogeneous" and "heterogeneous" differs slightly. In his terminology "heterogenous" refers to any difference in the learning algorithms. Thus different neural network models obtained by different random initializations of network weights would be termed "heterogenous" by his approach.

is drawn by sampling the training set with replacement. Then, the bootstrap sample is given to the classification algorithm to obtain a model. This process is repeated N times (for some large N) and the resulting ensemble is combined via majority vote.

Boosting, as it has typically been applied, also constructs an ensemble of models by obtaining each model from a training run over a different distribution on the training set. However, each model is produced sequentially and the weight each training example is given is a function of the number of previously trained models that predicted the example's class incorrectly. In this way, each successive model focuses more on the "hard examples" that earlier models mispredicted. Boosting can often be better viewed as a feature selection method rather than a variance reduction method such as the previous two methods. The reason for this is that the primary empirical success of boosting has been to classification algorithms where each model does not attempt to fully solve the problem. One such example is boosting decision stumps [SFBL98]; a decision stump is a decision tree of depth one. In this sense, they can be seen as more of an attempt to avoid "data-fracturing" by choosing a set of features where every example had at least some weight on the predictor chosen. For those interested in alternative methods of using "all the data", Domingos [Dom94] uses a heuristically guided hill-climbing search to induce a series of classification rules over all the data and weights the final combination of rules according to a combined measure of coverage and precision.

Hierarchical mixture of experts (HME) [JJNH91, JJ94] has a quite different flavor from the other homogeneous ensembles discussed. This method can be thought of as a tree with classification models at the leaves and weighting functions at the internal nodes. The models at the leaves make a prediction, and the predictions are "blended" by the weighting functions, which act as gates as they propagate up the trees. The weighting functions are functions of the input as well. Therefore, the weighting functions can provide a soft partitioning of the input space. The authors first motivated the use of HME as another method that avoids fracturing the data as divide-and-conquer methods do, and instead use all of the data in combination with simple (high-bias) estimators at the leaves to strike a more favorable bias-variance tradeoff. Because the weighting functions are functions of the input, HME is the first method we have discussed that has a notion of locality. The weighting functions typically used are generalized linear functions (a fixed non-linear function of a linear transform of the weight). Typically, the model parameters and the weighting functions are trained in conjunction. Therefore, depending on the type of experts used at the leaves, estimating the parameters may be computationally intensive. As each example is seen, the gating function gives increased weight to the experts that perform well on it, and when the experts commit errors during training, they are updated according to the amount of weight that the gating network placed on them. Therefore all experts are not generally trained over the same data. Thus, this approach shares similarity with boosting in that it addresses how a complex task can be broken down into the combination of simple classifiers trained over altered data distributions. The authors do not specifically address how fixed models can be combined. Although the scheme could be altered to do so; in this case, it would become an instance of local cascade generalization (discussed later) where the gating function is essentially the metaclassifier being trained.

In contrast, our approach focuses on richer definitions of locality and uses these in combining the outputs of models from *different* inductive algorithms. Thus, as opposed to homogeneous methods where the individual models result from the same algorithm, we seek to also draw insights into how the strengths of the various classification algorithms are effectively employed by the combination algorithm.

2.2 Heterogeneous Ensembles

Heterogeneous ensembles are combination methods that combine different models obtained from different classification algorithms. While technically a heterogeneous ensemble could be applied to different models obtained from the same algorithm, they are different in that they typically stem from one of the following two motivations: (1) if the errors of a set of classifiers are independent,[3] then the error rate of an appropriate combination of those classifiers drops exponentially fast with the number of classifiers; [Die00] (2) each classifier has a *bias*, or a restriction on the set of functions it can learn, and by combining different classification algorithms it is possible to relax the bias and learn more expressive models (as always at a cost in terms of variance) [Gam98a, Gam98b].

Since a classification algorithm often outputs a model that performs well but disagrees on some examples with a model obtained from a different classification algorithm, researchers have often simply assumed they provide independent sources of information and combined them accordingly. Majority vote and constant-weighted sums of outputs have operated under this assumption with varying empirical success; we discuss particular applications of these rules to text problems below.

An alternative to assuming the classifier outputs provide independent information about the class is to take into account the fact that they may be partially dependent. Merz and

[3]At this point, a common misunderstanding most be pointed out. The phrase, often used in literature, "errors of a set of classifiers are independent" is sometimes mistakenly confused with "the output of the classifiers are independent". Obviously, we do not expect the output of the classifiers to be independent — they are learning the same function. Instead a more precise statement would refer to the classifier's outputs being conditionally-independent given the class label as discussed in Section 1.2.1.

Pazzani [Mer98, MP99, Mer99] do this by introducing methods which use an intermediate representation obtained via singular value decomposition. They introduce a method for combining regression estimates (PCR*) and one for combining classifier outputs (SCANN). The first uses principle components analysis to factor the covariance matrix of the regression estimates, followed by a validation step to determine the number of principal vectors to retain, and then performs regression using the retained components to determine how heavily to weight each method. The final form of the learned function uses a constant-weighted linear combination and thus is not a local method in our terminology. For classification, the class predictions of the classifiers are obtained and correspondence analysis is applied to the prediction matrix to determine the canonical variates. Similar to the regression approach, a validation step is used to determine the number of variates to keep and the final set of weights is determined using a nearest neighbor approach in the space of the remaining variates. The final form of the learned function uses a single weight per class. Thus the classification approach has some use of locality. Although since we decompose problems into binary prediction tasks, we can rewrite such a model for binary classification as a model that uses a single set of weights.

Another approach to taking dependency into account is taken by those models that use a metaclassifier (as in Figure 1.2). Two such approaches are stacking and cascade generalization. Wolpert introduced stacking in [Wol92], and more recently, others have studied its effectiveness more thoroughly [TW99]. Stacking trains a metaclassifier over the outputs of the lower level classifiers. Stacking can thus account for dependencies between the classifiers by observing them in the training data for the metaclassifier model. Because the metaclassifier can also generalize according to how close the examples are in terms of the base classifier outputs, then stacking[4] also has a sense of locality but restricted to nearness in base classifier outputs. Cascade generalization [Gam98a] is similar to stacking, but where stacking performs cross-validation to obtain outputs from the base classifiers to train the metaclassifier, cascade generalization simply trains the base classifiers once and gives the metaclassifier the base classifiers' predictions over the training set. Thus, Gama argues that cascade generalization is more appropriately seen as relaxing the bias (or extending the expressiveness) of the base models. *Local* cascade generalization [Gam98b] also includes all of the base features of the example when training the metaclassifier. As mentioned above, using all of the base features is often not feasible for text problems because of the high dimensionality of text. Thus, our work attempts to find a low-dimensional representation of locality. Additionally, the representation aids in the interpretability of the resulting models.

[4]Depending on which metaclassifier model is used.

In recent work, Caruana *et al.* [CNM04] present a method for performing simple globably weighted combinations of classifiers. Their approach builds a large library of classification models (several thousand) by varying the parameters of several agorithms. Then using a simple hillclimbing approach to select with replacement a set of models to vote equally. Correlation is an even greater challenge for their framework since the majority of models result from varying a parameter smoothly for a base classification algorithm. Their empirical investigation includes several oddities, however. They restrict the data available to the metaclassifier by using a small validation set. Because of the highly correlated models, this puts many of the meta-algorithms at a disadvantage. They probably would have performed better if only given a small set of base models using default paramaters. Additionally, they do not compare their methods to methods explicitly targeted at handling correlation such as Merz's work discussed above.

2.2.1 Other Relevant Approaches

Another approach sometimes termed any of "metaselection", "metalearning", or "metaclassification", attempts to pick one single (but potentially different) classifier for each discrimination task. For example, one might learn a rule of the form, "If there are more than 200 examples of each class, use the kNN model, else use the SVM model." We prefer the term "metaselection" for this approach since only one single model is actually used for predictions. In contrast, most of the other heterogeneous methods (and ours) vote or blend the predictions of each in some way. An example of metaselection is [LL01] where the authors define features that they then use to choose the classifier to apply to a particular problem. Our approach is more general than this since we can blend classifiers based on the specific documents. For features that may be relevant to choosing a classifier for a problem but static across all documents within that problem (*e.g.*, number of training examples), our work on Inductive Transfer demonstrates how to extend our combination framework to obtain the benefits of both metaselection and combination.

Inductive Transfer is one of several methods such as multitask learning that attempt to overcome the typical scarcity of labeled data by building predictive models using knowledge transfered from another task. In multitask learning, additional information for building models comes in the form of labels for related functions which can be learned over the same input. Although such additional labels are typically unavailable at prediction time, results have demonstrated that generalization performance can be improved on the primary task by learning to predict the new variables in addition to the output variable of interest.

Caruana [Car97] presents an approach to and analysis of multitask learning when the n function-approximation tasks are over the same input (*i.e.*, a labeled example consists of $x_1, \ldots, x_m$ data attributes and the values for this example of the n functions to be learned $f_1(\vec{x}), \ldots, f_n(\vec{x})$). In this analysis, the main concern is generalization performance for one particular f_i, the primary problem. Likewise, the Curds & Whey approach proposed by Breiman & Friedman [BF95] solves a similarly formulated problem but attempts to minimize the squared error across all of the n functions instead of placing emphasis on one task.

In contrast to multitask learning, we show how to leverage labeled data from related problems over examples in different input spaces to enhance the final model used in prediction. Problems related to this challenge have been termed *classifier re-use* [BG98] or *knowledge transfer* [CK97]. We introduce a new approach to the challenge that hinges on mapping the original feature space, targeted at predicting membership in a specific topic, to a new feature space aimed at modeling the reliability of an ensemble of text classifiers.

Thrun & O'Sullivan [TO96] present methods for identifying related tasks and sequentially transferring knowledge when using a nearest-neighbor classifier. These methods are applicable when the input has the same representation across tasks. Both Thrun & O'Sullivan's and Breiman & Friedman's work could be applied to the inductive transfer problem we lay out *after* transforming the data to our representation.

Cohen & Kudenko [CK97] perform an analysis of classifier re-use and sequential knowledge transfer in information filters for text documents. Their work showed that significant improvements could be introduced when the classifiers were constructed to primarily model features positively correlated with the topic (*i.e.*, word *presence* that is positively correlated with being *In-Topic*). However, the method also relies on the new task and the old task sharing significant overlap in the underlying concept to be learned.

Finally, Bollacker & Ghosh [BG98] present a novel mechanism for classifier re-use where a classifier is constructed for each of a set of support tasks that are later used in predictions for a primary task. The final classification is selected by predicting the same class as the training data item (from the primary task data) that has the most similar prediction pattern using the support classifiers. Since each support classifier is applied to examples from every task, the input representation for each of the related tasks must be the same. Additionally, the scheme, like error-correcting output coding [Die00], relies more on an assumption that the extra-task labels will serve as a natural encoding for the data rather than other re-use mechanisms that specifically bias models or build representations of domain knowledge.

2.3 Related Work Using Locality

We refer to a combination method as using *locality* if the algorithm can induce a model which cannot be expressed as a constant-weighted linear combination of a classifier's predictions, probabilities, scores, or log odds. Thus both the representation and the algorithm determine whether an approach uses locality. For example, a *linear* SVM metaclassifier applied to the log-odds of a classifier's predictions does not use locality, but a decision tree algorithm, which can learn a non-linear function of the classifier's outputs, uses locality.

Locally weighted combinations have received far less attention in the reseach community. As we have mentioned above, some of the more prominent examples that include some notion of locality are stacking, cascaded generalization, and HME. These methods[5] use either only the classifier outputs or the classifier outputs and base features to set a local weight. In contrast, we combine the outputs based on properties of the classifier output that capture their local variance and accuracy.

Merz [Mer95] also uses locality in his approach to dynamic selection and combination by using a nearest-neighbor approach where two examples are considered similar if the base classifiers have a similar prediction pattern for both of them. In the selection case, the classifier with the highest accuracy in the retrieved neighbors is used, and in the combination approach, they are voted according to their accuracies. However, he was not able to obtain promising results in the empirical evaluation. Woods, Kegelmeyer and Bowyer [WJB97] present a similar method but the local accuracy of the classification methods are estimated over a neighborhood found in the original input space. Very minor improvements were demonstrated for several problems that had low-dimensional input spaces. Additionally, while they tried different neighborhood sizes, they do not report how they decided on the size they used in their final results.

Finally, Tresp & Taniguchi [TT95] investigate a variety of locally weighted combination rules and apply them to a low-dimensional data set (13 features). While they do investigate rules that take advantage of local variance, density (number of samples around the point), and reliability, their rules are sometimes not practical for high-dimensional data. More importantly, their approach relies on generative assumptions about the data.

Additionally, our approach allows for richer definitions of locality than simple nearness in the (Euclidean) feature representation. For example, the reliability indicator *Unigram-Variance* that we discussed briefly above and in more detail below is essentially the variance of a unigram model's output with the deletion of a single word occurrence in the document. We could also consider a more expressive generalization such as the variance with deletion

[5]The reader is referred to Sections 2.1 and 2.2 above for more details on each of these methods.

of a sentence or paragraph. Since whole contiguous sections may be deleted, this allows the covariance of the words, as determined by the structure of language, to play a role in how locality is effectively defined for documents.

2.4 Previous Applications to Text Problems

The combination of multiple methodologies or representations has been employed in several text related areas outside of text classification. For example, previous research in information retrieval has demonstrated that retrieval effectiveness can be improved by using multiple, distinct representations [BCB94, KMT$^+$82, RC95], or by using multiple queries or search strategies [BCCC93, SF95]. Freitag [Fre98] presents a study of combining inductive learners for information extraction where several simple rules for combination are employed after the classifier outputs are normalized.

In the realm of text classification, several researchers have achieved improvements in classification accuracy via the combination of different classifiers [HPS96, LC96, LJ98, YAP00]. Other investigators have reported that combined classifiers work well compared to some particular approach [AKTV$^+$01], but they have not reported results that compare the accuracy of the classifier with the accuracies of the individual contributing classifiers. Thus, it is difficult to draw insights from their work about how the reliabilities of the contributing classifiers vary over the input space. Similarly, systems that seek to enhance classification performance by applying many instances of the same classifier, such as in boosting procedures [SS00, WAD$^+$99], typically leverage weaker component learners that would not be directly examined as stand-alone classifiers.

Much of the previous work on combining text classifiers has used relatively simple policies for selecting the best classifier or for combining the output of multiple classifiers. As some examples, Larkey and Croft [LC96] used weighted linear combinations of system ranks or scores; Hull *et al.* [HPS96] used linear combinations of probabilities or log odds scores; Yang *et al.* [YAP00] used a linear combination of normalized scores; and Li and Jain [LJ98] used voting and classifier selection techniques. As discussed in detail in Section 2.2, Lam and Lai [LL01] use category-averaged features to perform metaselection. Ruiz and Srinivasan's [RS02] study on applying HME to hierarchical text classification is an example of a more complicated combination rule that has been applied to text. In order to make the approach computationally feasible, they performed significant dimensionality reduction using feature selection. Despite this, they only obtained performance comparable to a version of the Rocchio algorithm. In contrast, our work has demonstrated a significant improvement over competitive text classification algorithms. This indicates that making the most effective use of locality when combining text classifiers is still an unanswered question.

2.5 *No Free Lunch* and Its Implications

Finally, no discussion of work related to classifier combination would be complete without a discussion of the *No Free Lunch Theorem* [DHS01, Wol95] and its implications for classifier combination. Wolpert, who also introduced stacking [Wol92], derived it as a hardness result, which in short demonstrates that there exists no classifier superior to every other classifier unless we make assumptions about the example distribution. That is, even given substantial training data, for any specific classifier there will always be some example distributions on which it is outperformed by another classifier.[6] Additionally, there will also exist some distributions where random guessing outperforms the classifier.

As a result, the performance of a classifier on a set of problems can be seen to be more an issue of the appropriateness of the fit between the classifier's assumptions and the true underlying distribution. Since classifier combination seeks to create a more effective classifier from the individual input classifiers, some researchers have mistakenly believed that the *No Free Lunch Theorem* implies any attempt at classifier combination is futile. However, this is clearly no more true than saying that classification is futile. Instead it must be understood that the empirical performance of the combination method will be dependent on the fit between the assumptions the combination method makes about the base classifiers, the example distribution, and reality. When these assumptions are a good fit for the problems seen in practice, the combination model will perform well. Thus, just as some classifiers (*e.g.*, SVMs) dominate a large number of problems (*e.g.*, text classification) seen in practice, the challenge is to develop a metaclassification algorithm that captures the common interactions among base classifiers seen in practice.

In the context of this dissertation then, we will highlight the conditions under which a particular metaclassifier will perform well (provide a good fit) based on the interaction of the base classifiers and the training data available (characteristics of the task). For example, when the other base classifiers do not provide additional information to that of the best base classifier, then we might expect some algorithms will use the data more efficiently to perform better. Whereas, when a single classifier dominates in different regions of the input space, then another metaclassifier might outperform the previous algorithm. Likewise, the best choice might vary again when we consider when the optimal cases are linear combinations. Since the No Free Lunch theorem provides us with a proof that there is no

[6]One common assumption that is made to prove results are that the examples are drawn *i.i.d.*

data-independent best choice or even a choice that will always outperform random, then we aim to elucidate through data properties and empirical evaluation the tradeoffs involved.

Chapter 3

Calibration

In this chapter we first review the concept of *calibration* — a measure of how good a set of probability estimates are. If we consider the extreme case of applying a metaclassifier to the outputs of a single base classifier, then the metaclassifier is either implicitly or explicitly recalibrating the base classifier. In fact, given only the base classifier's *probability estimates*, the metaclassifier cannot improve on those estimates if they are well-calibrated.

As a result, understanding how any combination method works in the case of a single base classifier can give important insight into its behavior — especially since it is possible to easily examine the empirical behavior of a base classifier's probabilities. For example, we will show that Kahn's [Kah04] assumption of normally distributed class-conditional log-odds rarely empirically holds for a single base classifier, and we will provide an explanation why it is unlikely to hold for accurate classifiers. Additionally, since not all classifiers directly estimate probabilities, this study also aids in producing probability estimates for combination methods that require them.

The remainder of the chapter is devoted to investigating the empirical behavior of probability estimates obtained from various classifiers, explaining this behavior, and developing new methods to improve the calibration of probability estimates obtained from classifiers based on their observed empirical behavior.

3.1 Calibration and Related Concepts

An obvious way to approach classifier combination is to treat each classifier, $\mathcal{C}_i$, as a black box that outputs an estimate of the probability distribution over class labels for each datapoint, $\hat{P}_{\mathcal{C}_i}(c \mid d)$. Then, perform the combination by simply combining these estimates with a linear or a weighted (normalized) multiplicative combination. A slightly more general approach assumes each classifier outputs an unnormalized score $score_{\mathcal{C}_i}(d, c)$, which is the score assigned to class c for document d. This score is then converted to a probability $\hat{P}_{\mathcal{C}_i}(c \mid d)$ using some other method.[1]

However, there is no guarantee that the estimates from different classifiers adhere to a fixed standard. That is, for one classifier, 0.8 of the items assigned 0.6 probability for the class under consideration may actually belong to the class; while for another classifier, 0.5 of the items assigned 0.6 probability may belong to the class. In some sense then, a prediction of 0.6 from each classifier "means" different things.

DeGroot and Fienberg [DF83] review the concept of *calibration*, a candidate to use as a fixed standard that addresses such inconsistencies. We say a classifier is *well-calibrated* if as the number of predictions goes to infinity, the predicted probability goes to the empirical probability. That is, for all unique π_i, such that $\pi_i = \hat{P}_{\mathcal{C}_i}(c \mid d)$, the empirical relative frequency[2] equals π_i, *i.e.* $\tilde{P}(c \mid \pi_i) = \pi_i$. This can be best envisioned graphically with the aid of a reliability diagram. Consider a two-class discrimination problem where $\mathcal{Y} = \{0, 1\}$. Then, we can plot the classifier's predictions for one of the classes on the x-axis and the empirical relative frequency on the y-axis. Figure 3.1 demonstrates this.

The above described calibration as a frequentist concept, but the reader should note that it can also be viewed from a Bayesian viewpoint [GCSR95, GZ86]. From the Bayesian view, an outside observer is actually stating his belief about a classifier's behavior. The classifier is then well-calibrated if the observer cannot improve upon the classifier's forecasts given only the output of the classifier. Thus, a well-calibrated forecaster has, in a sense, summarized all of its information in the probabilities it is emitting.

When evaluating the usefulness of predictions, we also must consider the frequency with which a classifier outputs a particular prediction. For example, suppose for our problem the actual prior is $P(c = 1) = 0.7$. A classifier could simply predict $\hat{P}_{\mathcal{C}_i}(c = 1 \mid d) = 0.7$ all the time, and it would be well-calibrated. However, it is clearly less useful than another classifier that is also well-calibrated and outputs $\hat{P}_{\mathcal{C}_j}(c = 1 \mid d) = 0.9$ half the time and $\hat{P}_{\mathcal{C}_j}(c = 1 \mid d) = 0.5$ the other half of the time. DeGroot and Fienberg [DF83] introduce the concept of *refinement* to compare two well-calibrated classifiers. Essentially, one well-calibrated classifier $\mathcal{C}_i$ is at least refined as another $\mathcal{C}_j$ if the predictions of $\mathcal{C}_i$ can be passed through a noisy channel to produce a well-calibrated classifier whose characterization in terms of calibration (the discrete set of prediction values it outputs and their

[1] An assumption often made during this conversion is that $\hat{P}_{\mathcal{C}_i}(c \mid d)$ is monotonic in $score_{\mathcal{C}_i}(d, c)$.

[2] There is an assumption that the space of probability estimates has been discretized to form a finite set of possible values, *e.g.* $\{0, 0.1, 0.2, \ldots, 1\}$.

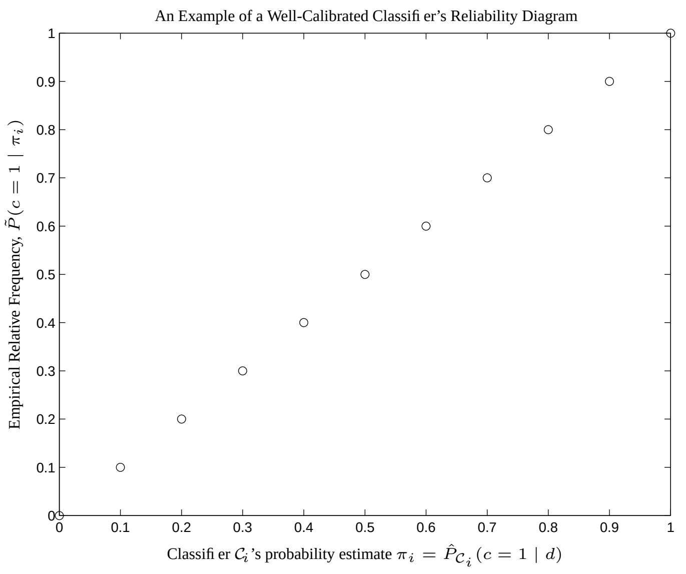


Figure 3.1: For a well-calibrated classifi er, all points in a reliability diagram fall on the diagonal. In the long-run 0.6 (generally π_i) of the items the classifi er predicts to have 0.6 probability (generally probability π_i) of belonging to the class, actually do belong to the class. Additionally, a reliability diagram often has annotations indicating the frequency with which a certain value is predicted.

frequencies) is the same as j. Formally, this holds if there exist stochastic functions (*i.e.*, distributions) h such that the following equations are satisfied for all $\pi_j \in \Pi$:

$$P(\pi_j) = \sum_{\pi_i \in \Pi} h(\pi_j \mid \pi_i) P(\pi_i) \tag{3.1}$$

$$\pi_j P(\pi_j) = \sum_{\pi_i \in \Pi} \pi_i h(\pi_j \mid \pi_i) P(\pi_i) \tag{3.2}$$

Additionally, DeGroot and Fienberg give a simple statistical test that is necessary and sufficient to determine if one classifier is more refined than another. The most refined classifier is the one that only outputs predictions 0 and 1 and is always correct.

An implication for classifier selection is that if we must choose only one of two well-calibrated predictors, then it is always better to use the most refined predictor regardless of how the predictions will be used. Refinement is a partial-ordering, and thus it is possible that neither classifier is more refined than the other. Furthermore, it is unclear what the implication is for predictors that are not calibrated — which is the case far more often than not in practice.

Therefore, DeGroot and Fienberg continue by generalizing the notion of refinement to a related concept they call *sufficiency*, which can be used to compare any two classifiers regardless of whether or not they are well-calibrated. A classifier $\mathcal{C}_i$ is sufficient for $\mathcal{C}_j$ if the distribution of $\mathcal{C}_j$'s predictions can be characterized as a stochastic function of $\mathcal{C}_i$'s.

Formally, $\mathcal{C}_i$ is sufficient for $\mathcal{C}_j$ if there exists a set of distributions h such that the following equalities are satisfied:[3]

$$P(\pi_j \mid y) = \sum_{\pi_i \in \Pi} h(\pi_j \mid \pi_i) P(\pi_i \mid y) \quad \text{for all } \pi_j \in \Pi \text{ and } y \in \mathcal{Y}. \tag{3.3}$$

Like refinement, sufficiency is a partial ordering, and if we must select only one of two classifiers and $\mathcal{C}_i$ is sufficient for classifier $\mathcal{C}_j$, it is always better to choose classifier $\mathcal{C}_i$. However, we must ask whether we can still gain from combining these two classifiers. DeGroot and Fienberg demonstrate that it is possible for classifier i to be sufficient for classifier j but *information can always be gained from their combination* except when $P(c \mid \pi_i, \pi_j) = P(c \mid \pi_i)$ for all π_i, π_j. To rephrase, we can gain by combining the two classifiers unless the class is independent of the output of $\mathcal{C}_j$ given the output of $\mathcal{C}_i$.

Since this has implications for classifier combination, it is worth considering further how even when one classifier can be characterized as a stochastic function of another, it can sometimes be used to gain improvement in combination. The argument proving it is constructive but not exhaustive. That is, we show one set, but not the only set, of conditions where this holds. Suppose, we are given $\mathcal{C}_i$. We can easily choose an arbitrary set of $h(\pi_j \mid \pi_i)$ to probabilistically generate π_j from π_i. Thus h, $P(\pi_j \mid c)$, and $P(\pi_i \mid c)$ satisfy the conditions in Eq. 3.3. Now, one class-conditional valid joint is when the classifiers are conditionally independent: $P(\pi_i, \pi_j \mid c) = P(\pi_i \mid c)P(\pi_j \mid c)$. When that distribution governs the data, then the combination of π_j and π_i can surpass either classifier. In essence, even though knowing π_i gives us information about what values π_j takes, it is only indirectly via the common class variable they are predicting.

Finally, *we may not retain the property of calibration even when we are combining well-calibrated classifiers* with simple rules such as arithmetically averaging their predictions or using a normalized product (naïve Bayes combination). Table 3.1 shows an example where the simple combinations improve their predictive power but fail to maintain calibration. However, if we were to perform perfect post-processing recalibration on π_P and π_A, both would perform as well as the optimal combination π_*. While recalibration plays only a small role in our combination framework, it is sometimes necessary for methods that assume their inputs are calibrated or when gauging if a combination method has been more successful than is apparent (by recalibrating the combination method). The next section discusses the new recalibration techniques we designed and the improvement they lead to in practice.

[3]This equation does not indicate how to find such an h, however.

P(x)	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125		
Class	1	0	1	0	1	0	1	0	MSE	E[ln $\hat{P}$]
π_1	0.25	0.25	0.75	0.25	0.75	0.25	0.75	0.75	0.1875	-0.5623
π_2	0.75	0.75	0.75	0.25	0.75	0.25	0.25	0.25	0.1875	-0.5623
π_P	0.5	0.5	0.9	0.1	0.9	0.1	0.5	0.5	0.1300	-0.3933
π_A	0.5	0.5	0.75	0.25	0.75	0.25	0.5	0.5	0.1562	-0.4904
π_*	0.5	0.5	1	0	1	0	0.5	0.5	0.1250	-0.3466

Table 3.1: Displayed is an example of the output distribution of two well-calibrated classifiers, π_1 and π_2, and some sample combination rules: normalized product (π_P), average (π_A), and the optimal combination given only the predictions $\pi_* = P(c \mid \pi_1, \pi_2)$. Although both π_P and π_A improve over the base classifiers, neither are well-calibrated.

3.2 Recalibrating Classifiers

As mentioned above, calibration can play a crucial role in classifier combination. While the majority of our work has explored combination methods that implicitly account for differing levels of calibration among the classifiers, some combination algorithms rely on the input classifiers to be calibrated in order to perform well. Therefore, this section presents new methods we derived for recalibrating classifiers and mapping scores to probability estimates. The primary contribution is the use of an asymmetric Laplace distribution to achieve as good or better results than competing parametric models.

The connections between recalibration and combining classifiers also runs deeper. If we consider giving a metaclassifier a single classifier as input, the metaclassifier is simply recalibrating the base classifier. Both the empirical methods presented in this section and the theoretically-based arguments show that classifiers will typically demonstrate asymmetric behavior. Therefore, we see that Kahn's [Kah04] assumption when combining classifiers of normally distributed class-conditional log-odds does not even hold for a single classifier.

3.2.1 The Need for Calibrated Probabilities in Other Applications

In addition to the role calibration plays in combination, recalibrating classifiers and mapping scores to probability estimates is an important problem in its own right. This is because text classifiers that give probability estimates are more flexible in practice than those that give only a simple classification or even a ranking. For example, rather than choosing one set decision threshold, they can be used in a Bayesian risk model [DHS01] to issue a run-time decision which minimizes the expected cost of a user-specified cost function dynamically chosen at prediction time. This can be used to minimize a linear utility cost function for filtering tasks where pre-specified costs of relevant/irrelevant are not available during training but are specified at prediction time. Furthermore, the costs can be changed without retraining the model. Additionally, probability estimates are often used as the basis of deciding which document's label to request next during active learning [LG94, STP01]. Effective active learning can be key in many information retrieval tasks where obtaining labeled data can be costly — severely reducing the amount of labeled data needed to reach the same performance as when new labels are requested randomly [LG94]. Finally, they are also amenable to making other types of cost-sensitive decisions [ZE01]. However, in all of these tasks, the quality of the probability estimates is crucial.

Parametric models generally use assumptions that the data conform to the model to trade-off flexibility with the ability to estimate the model parameters accurately with little training data. Since many text classification tasks often have very little training data, we focus on parametric methods. However, most of the existing parametric methods that have been applied to this task have an assumption we find empirically undesirable. While some of these methods allow the distributions of the documents relevant and irrelevant to the topic to have different variances, they typically enforce the unnecessary constraint that the documents are symmetrically distributed around their respective modes. We introduce several asymmetric parametric models that allow us to relax this assumption without significantly increasing the number of parameters and demonstrate how we can efficiently fit the models. Additionally, these models can be interpreted as assuming the scores produced by the text classifier have three basic types of empirical behavior — one corresponding to each of the "extremely irrelevant", "hard to discriminate", and "obviously relevant" items.

First, we discuss in further detail the need for asymmetric models. After this, we describe two specific asymmetric models and, using two standard text classifiers, naïve Bayes and SVMs, demonstrate how they can be efficiently used to recalibrate poor probability estimates or produce high quality probability estimates from raw scores. We then review experiments using previously proposed methods and the asymmetric methods over several text classification corpora to demonstrate the strengths and weaknesses of the various methods. Finally, we review related work on improving probability estimates and summarize our contributions.

3.2.2 Recalibration Problem Definition & Approach

Our work differs from earlier approaches primarily in two points: (1) We provide *asymmetric* parametric models suitable for use when little training data is available; (2) We explicitly analyze the quality of probability estimates these and competing methods produce and provide significance tests for these results.

Problem Definition

The general problem we are concerned with is highlighted in Figure 3.2. A text classifier

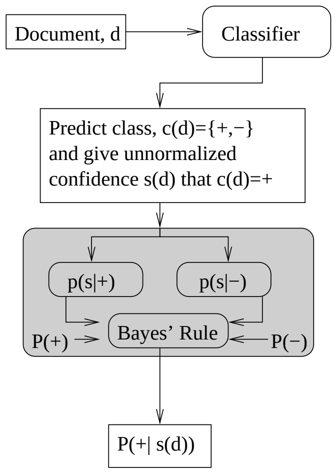


Figure 3.2: We are concerned with how to perform the box highlighted in grey. The internals are for one type of approach.

produces a prediction about a document and gives a score $s(d)$ indicating the strength of its decision that the document belongs to the *positive* class (relevant to the topic). We assume throughout there are only two classes: the positive and the negative (or irrelevant) class ('+' and '-' respectively).

There are two general types of parametric approaches. The first of these tries to fit the posterior function directly, *i.e.* there is one function estimator that performs a direct mapping of the score s to the probability $P(+|s(d))$. The second type of approach breaks the problem down as shown in the grey box of Figure 3.2. An estimator for each of the class-conditional densities (*i.e.* $p(s|+)$ and $p(s|-)$) is produced, then Bayes' rule and the class priors are used to obtain the estimate for $P(+|s(d))$.

Motivation for Asymmetric Distributions

Most of the previous parametric approaches to this problem either directly or indirectly (when fitting only the posterior) correspond to fitting Gaussians to the class-conditional densities; they differ only in the criterion used to estimate the parameters. We can visualize this as depicted in Figure 3.3. Since increasing s usually indicates increased likelihood of belonging to the positive class, then the rightmost distribution usually corresponds to $p(s|+)$.

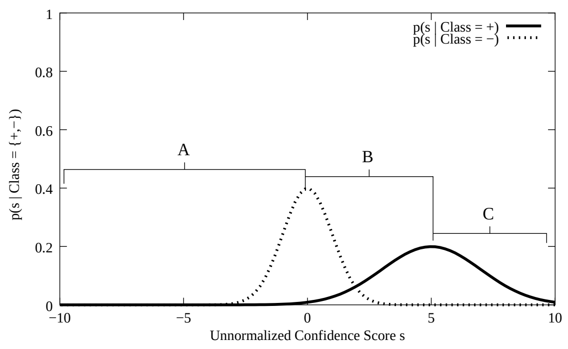


Figure 3.3: Typical View of Discrimination based on Gaussians

However, using standard Gaussians fails to capitalize on a basic characteristic commonly seen. Namely, if we have a raw output score that can be used for discrimination, then the empirical behavior between the modes (label B in Figure 3.3) is often very different than that outside of the modes (labels A and C in Figure 3.3). Intuitively, the area between the modes corresponds to the *hard* examples, which are difficult for this classifier to distinguish, while the areas outside the modes are the extreme examples that are usually easily distinguished. This suggests that we may want to uncouple the scale of the outside and inside segments of the distribution (as depicted by the curve denoted as *A-Gaussian* in Figure 3.4). As a result, an asymmetric distribution may be a more appropriate choice for application to the raw output score of a classifier.

Ideally (*i.e.* perfect classification) there will exist scores θ_- and θ_+ such that all examples with score greater than θ_+ are relevant, and all examples with scores less than θ_- are irrelevant. Furthermore, no examples fall between θ_- and θ_+. The distance $|\theta_- - \theta_+|$ corresponds to the margin in some classifiers, and an attempt is often made to maximize this quantity. Because text classifiers have training data to use to separate the classes, the final behavior of the score distributions is primarily a factor of the amount of training data and the consequent separation in the classes achieved. This is in contrast to search engine retrieval where the distribution of scores is more a factor of language distribution across documents, the similarity function, and the length and type of query.

Perfect classification corresponds to using two very asymmetric distributions, but in this case, the probabilities are actually one and zero, and many methods will work for typical purposes. Practically, some examples will fall between θ_- and θ_+, and it is often important to estimate the probabilities of these examples well (since they correspond to the "hard" examples). Justifications can be given for both why you may find more and less examples between θ_- and θ_+ than outside of them, but there are few empirical reasons to believe that the distributions should be symmetric.

A natural first candidate for an asymmetric distribution is a generalization of a common symmetric distribution, *e.g.* the Laplace or the Gaussian. An asymmetric Laplace distribution can be achieved by placing two exponentials around the mode in the following manner:

$$p(x \mid \theta, \beta, \gamma) = \begin{cases} \frac{\beta\gamma}{\beta+\gamma} \exp\left[-\beta\left(\theta - x\right)\right] & x \leq \theta \qquad \text{(left of mode)} \\ & \qquad (\beta, \gamma > 0) \\ \frac{\beta\gamma}{\beta+\gamma} \exp\left[-\gamma\left(x - \theta\right)\right] & x > \theta \qquad \text{(right of mode)} \end{cases} \tag{3.4}$$

where θ, β, and γ are the model parameters. θ is the mode of the distribution, β is the inverse scale of the exponential to the left of the mode, and γ is the inverse scale of the exponential to the right. We will use the notation $\Lambda(X \mid \theta, \beta, \gamma)$ to refer to this distribution.

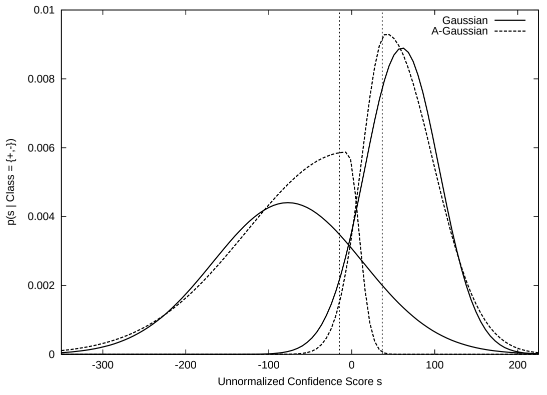


Figure 3.4: Gaussians vs. Asymmetric Gaussians. A Shortcoming of Symmetric Distributions — The vertical lines show the modes as estimated nonparametrically.

We can create an asymmetric Gaussian in the same manner:

$$p(x \mid \theta, \sigma_l, \sigma_r) = \begin{cases} \frac{2}{\sqrt{2\pi}(\sigma_l+\sigma_r)} \exp\left[-\frac{(x-\theta)^2}{2\sigma_l^2}\right] & x \leq \theta \quad \text{(left of mode)} \\ & (\sigma_l, \sigma_r > 0) \\ \frac{2}{\sqrt{2\pi}(\sigma_l+\sigma_r)} \exp\left[-\frac{(x-\theta)^2}{2\sigma_r^2}\right] & x > \theta \quad \text{(right of mode)} \end{cases} \tag{3.5}$$

where θ, σ_l, and σ_r are the model parameters. σ_l and σ_r are the scale parameters to the *left* and to the *right* of the mode, respectively; when $\sigma_l = \sigma_r$, a symmetric Gaussian with standard deviation σ_l is obtained. To refer to this asymmetric Gaussian, we use the notation $\Gamma(X \mid \theta, \sigma_l, \sigma_r)$. While these distributions are composed of “halves”, the resulting function is a single *continuous* distribution.

These distributions allow us to fit our data with much greater flexibility at the cost of only fitting six parameters. We could instead try mixture models for each component or other extensions, but most other extensions require at least as many parameters (and can often be more computationally expensive). In addition, the motivation above should provide significant cause to believe the underlying distributions actually behave in this way. Furthermore, this family of distributions can still fit a symmetric distribution, and finally, in the empirical evaluation, evidence is presented that demonstrates this asymmetric behavior (see Figure 3.5).

To our knowledge, neither family of distributions has been previously used in machine learning or information retrieval. For the interested reader, statistical properties relevant to these distributions are discussed in great detail in [KKP01].

3.2.3 Estimating the Parameters of the Asymmetric Distributions

This section develops the method for finding maximum likelihood estimates (MLE) of the parameters for the above asymmetric distributions. In order to find the MLEs, we have two choices: (1) use numerical estimation to estimate all three parameters at once (2) fix the value of θ, and estimate the other two (β and γ or σ_l and σ_r) given our choice of θ, then consider alternate values of θ. Because of the simplicity of analysis in the latter alternative, we choose this method.

Asymmetric Laplace MLEs

For $\boldsymbol{D} = \{x_1, x_2, \ldots, x_N\}$ where the x_i are *i.i.d.* and $X \sim \Lambda(X \mid \theta, \beta, \gamma)$, the likelihood is:

$$\prod_{i}^{N} \Lambda(X \mid \theta, \beta, \gamma). \tag{3.6}$$

We desire to find the maximum likelihood estimates for β, γ and θ. To do so, we fix θ and compute the maximum likelihood for that choice of θ. Then, we can simply consider all choices of θ and choose the one with the maximum likelihood (or equivalently the loglikelihood) over all choices of θ.

The loglikelihood we must compute then is:

$$\begin{aligned}
\log \prod_{i=1}^{N} \Lambda(x_i \mid \theta, \beta, \gamma) &= \sum_{i=1}^{N} \log \Lambda(x_i \mid \theta, \beta, \gamma) && (3.7)\\
&= \sum_{x \in \boldsymbol{D} \mid x \leq \theta} \log \Lambda\left(x_i \mid \theta, \beta, \gamma\right) + \sum_{x \in \boldsymbol{D} \mid x > \theta} \log \Lambda\left(x_i \mid \theta, \beta, \gamma\right) && (3.8)\\
&= \sum_{x \in \boldsymbol{D} \mid x \leq \theta} \left[\log \frac{\beta\gamma}{\beta + \gamma} - \beta(\theta - x)\right] \\
&\quad + \sum_{x \in \boldsymbol{D} \mid x > \theta} \left[\log \frac{\beta\gamma}{\beta + \gamma} - \gamma(x - \theta)\right] && (3.9)\\
&= N \log \frac{\beta\gamma}{\beta + \gamma} + \sum_{x \in \boldsymbol{D} \mid x \leq \theta} [-\beta(\theta - x)] \\
&\quad + \sum_{x \in \boldsymbol{D} \mid x > \theta} [-\gamma(x - \theta)] && (3.10)
\end{aligned}$$

Let $N_l = \left|\{x \in \boldsymbol{D} \mid x \leq \theta\}\right|, \; N_r = \left|\{x \in \boldsymbol{D} \mid x > \theta\}\right|$
and $S_l = \sum_{x \in \boldsymbol{D} \mid x \leq \theta} x, \; S_r = \sum_{x \in \boldsymbol{D} \mid x > \theta} x$

$$= N \log \frac{\beta\gamma}{\beta + \gamma} - N_l \beta\theta + \beta S_l + N_r \gamma\theta - \gamma S_r \tag{3.11}$$

Let $D_l = N_l\theta - S_l$, $D_r = S_r - N_r\theta$

$$= N \log \frac{\beta\gamma}{\beta + \gamma} - \beta D_l - \gamma D_r \tag{3.12}$$

Note that D_l and D_r are the sum of the absolute differences between the x belonging to the left and right halves of the distribution (respectively) and θ. The partial derivatives are: $\frac{\partial\beta}{\partial l} = \frac{N\gamma}{\beta(\beta+\gamma)} - D_l$ and $\frac{\partial\gamma}{\partial l} = \frac{N\beta}{\gamma(\beta+\gamma)} - D_r$. We can set the derivatives to zero and solve them analytically to find that the MLEs for β and γ for a fixed θ are:

$$\beta_{MLE} = \frac{N}{D_l + \sqrt{D_r D_l}} \quad \gamma_{MLE} = \frac{N}{D_r + \sqrt{D_r D_l}}. \tag{3.13}$$

These estimates are not wholly unexpected since we would obtain $\frac{N_l}{D_l}$ if we were to estimate β independently of γ. The elegance of the formulae is that the estimates will tend to be symmetric only insofar as the data dictate it (*i.e.* the closer D_l and D_r are to being equal, the closer the resulting inverse scales).

By continuity arguments, when $N = 0$, we assign $\beta = \gamma = \epsilon_0$ where ϵ_0 is a small constant that acts to disperse the distribution to a uniform. Similarly, when $N \neq 0$ and $D_l = 0$, we assign $\beta = \epsilon_{\text{inf}}$ where ϵ_{inf} is a very large constant that corresponds to an extremely sharp distribution (*i.e.* almost all mass at θ for that half). $D_r = 0$ is handled similarly.

Assuming that θ falls in some range $[\phi, \psi]$ dependent upon only the observed documents, then this alternative is also easily computable. Given N_l, S_l, N_r, S_r, we can compute the posterior and the MLEs in constant time. In addition, if the scores are sorted, then we can perform the whole process quite efficiently. Starting with the minimum $\theta = \phi$ we would like to try, we loop through the scores once and set N_l, S_l, N_r, S_r appropriately. Then we increase θ and just step past the scores that have shifted from the right side of the distribution to the left. Assuming the number of candidate θs are $O(n)$, this process is $O(n)$, and the overall process is dominated by sorting the scores, $O(n \log n)$ (or expected linear time).

Asymmetric Gaussian MLEs

For $\boldsymbol{D} = \{x_1, x_2, \ldots, x_N\}$ where the x_i are *i.i.d.* and $X \sim \Gamma(X \mid \theta, \sigma_l, \sigma_r)$, the likelihood is:

$$\prod_{i=1}^{N} \Gamma(x_i \mid \theta, \sigma_l, \sigma_r) \tag{3.14}$$

We desire to find the maximum likelihood estimates for σ_l, σ_r and θ. Similar to the above, we fix θ and compute the maximum likelihood for that choice of θ. Then, we can simply consider all choices of θ and choose the one with the maximum likelihood (or equivalently the loglikelihood) over all choices of θ. The derivation is very similar to that for the Asymmetric Laplace.

The loglikelihood we must compute then is:

$$\begin{aligned}
\log \prod_{i=1}^{N} \Gamma(x_i \mid \theta, \sigma_l, \sigma_r) &= \sum_{i=1}^{N} \log \Gamma(x_i \mid \theta, \sigma_l, \sigma_r) && (3.15)\\
&= \sum_{x \in \boldsymbol{D} | x \leq \theta} \log \Gamma(x_i \mid \theta, \sigma_l, \sigma_r) + \sum_{x \in \boldsymbol{D} | x > \theta} \log \Gamma(x_i \mid \theta, \sigma_l, \sigma_r) && (3.16)\\
&= \sum_{x \in \boldsymbol{D} | x \leq \theta} \left[\log \frac{2}{\sqrt{2\pi}(\sigma_l + \sigma_r)} - \frac{(x-\theta)^2}{2\sigma_l^2} \right] \\
&\quad + \sum_{x \in \boldsymbol{D} | x > \theta} \left[\log \frac{2}{\sqrt{2\pi}(\sigma_l + \sigma_r)} - \frac{(x-\theta)^2}{2\sigma_r^2} \right] && (3.17)\\
&= N \log \frac{2}{\sqrt{2\pi}(\sigma_l + \sigma_r)} \\
&\quad - \frac{1}{2\sigma_l^2} \sum_{x \in \boldsymbol{D} | x \leq \theta} (x-\theta)^2 - \frac{1}{2\sigma_r^2} \sum_{x \in \boldsymbol{D} | x > \theta} (x-\theta)^2 && (3.18)
\end{aligned}$$

Let $N_l = \ |\{x \in \boldsymbol{D} \mid x \leq \theta\}|\ ,\ N_r = \ |\{x \in \boldsymbol{D} \mid x > \theta\}|\ ,$

$$S_l = \sum_{x \in \boldsymbol{D} | x \leq \theta} x,\ \ S_r = \sum_{x \in \boldsymbol{D} | x > \theta} x,\ \ S_{l^2} = \sum_{x \in \boldsymbol{D} | x \leq \theta} x^2,\ \text{and}\ \ S_{r^2} = \sum_{x \in \boldsymbol{D} | x > \theta} x^2.$$

$$\begin{aligned}
&= N \log \frac{2}{\sqrt{2\pi}(\sigma_l + \sigma_r)} - \frac{1}{2\sigma_l^2} \left[S_{l^2} - S_l\theta + N_l\theta^2 \right] \\
&\quad - \frac{1}{2\sigma_r^2} \left[S_{r^2} - S_r\theta + N_r\theta^2 \right] && (3.19)
\end{aligned}$$

Let $D_{l^2} = S_{l^2} - S_l\theta + \theta^2 N_l,\ D_{r^2} = S_{r^2} - S_r\theta + \theta^2 N_r$

$$= N \log \frac{2}{\sqrt{2\pi}(\sigma_l + \sigma_r)} - \frac{1}{2\sigma_l^2} D_{l^2} - \frac{1}{2\sigma_r^2} D_{r^2} \qquad (3.20)$$

The partial derivatives are: $\frac{\partial \sigma_l}{\partial l} = \frac{D_{l2}}{\sigma_l^3} - \frac{N}{\sigma_l + \sigma_r}$ and $\frac{\partial \sigma_r}{\partial l} = \frac{D_{l2}}{\sigma_r^3} - \frac{N}{\sigma_l + \sigma_r}$. We can set the derivatives to zero and solve them analytically to find for a fixed θ only one feasible solution:

$$\begin{aligned}
\sigma_{l,MLE} &= \sqrt{\frac{D_{l^2} + D_{l^2}^{2/3} D_{r^2}^{1/3}}{N}} && (3.21)\\
\sigma_{r,MLE} &= \sqrt{\frac{D_{r^2} + D_{r^2}^{2/3} D_{l^2}^{1/3}}{N}}. && (3.22)
\end{aligned}$$

By continuity arguments, when $N = 0$, we assign $\sigma_r = \sigma_l = \epsilon_{\text{inf}}$, and when $N \neq 0$ and $D_{l^2} = 0$ (resp. $D_{r^2} = 0$), we assign $\sigma_l = \epsilon_0$ (resp. $\sigma_r = \epsilon_0$). Again, the same computational complexity analysis applies to estimating these parameters.

3.2.4 Experimental Analysis

Methods

For each of the methods that use a class prior, we use a smoothed add-one estimate, i.e. $P(c) = \frac{|c|+1}{N+2}$ where N is the number of documents. For methods that fit the class-conditional densities, $p(s|+)$ and $p(s|-)$, the resulting densities are inverted using Bayes' rule as described above. All of the methods below are fit using maximum likelihood estimates.

For recalibrating a classifier (*i.e.* correcting poor probability estimates output by the classifier), it is usual to use the log-odds of the classifier's estimate as $s(d)$. The log-odds are defined to be $\log \frac{P(+|d)}{P(-|d)}$. The normal decision threshold (minimizing error) in terms of log-odds is at zero (*i.e.* $P(+|d) = P(-|d) = 0.5$).

Since it scales the outputs to a space $[-\infty, \infty]$, the log-odds make normal (and similar distributions) applicable [LTB79]. Lewis & Gale [LG94] give a more motivating viewpoint that fitting the log-odds has a dampening effect for the inaccurate independence assumption and a bias correction for inaccurate estimates of the priors. In general, fitting the log-odds can serve to boost or dampen the signal from the original classifier as the data dictate.

Gaussians

A Gaussian is fit to each of the class-conditional densities, using the usual maximum likelihood estimates. That is, for the class-conditional mean, we use $\mu_c = \frac{1}{N} \sum_{c(d)=c} s(d)$, and for the class-conditional variance[4] we use $\sigma_c^2 = \frac{1}{N} \sum_{c(d)=c} [s(d) - \mu_c]^2$. This method is denoted in the tables below as *Gauss*.

Asymmetric Gaussians

An asymmetric Gaussian is fit to each of the class-conditional densities using the maximum likelihood estimation procedure described in Section 3.2.3 above. Intervals between adjacent scores are divided into 10 pieces for testing candidate θs, *i.e.* Eight points between actual scores occurring in the data set are tested. This method is denoted as *A. Gauss*.

Laplace Distributions

Even though Laplace distributions are not typically applied to this task, we also tried this method to isolate why benefit is gained from the asymmetric form. The usual MLEs

[4]For the data we evaluated here, N was large enough that there was little difference between using the MLE for variance given here (which is biased) or the unbiased version, which multiplies by $\frac{1}{N-1}$ instead.

were used for estimating the location and scale of a classical symmetric Laplace distribution as described in [KKP01].

That is, let r_i, $i = 1, \ldots, N$ denote the ith score after the scores have been ranked by $s(d)$. The location parameter, θ is essentially the median of the datapoints. For N odd, $\theta = r_{\frac{N+1}{2}}$, and for N even, $\theta = \frac{1}{2}\left[r_{\frac{N}{2}} + r_{\frac{N}{2}+1}\right]$. The inverse scale parameter is given by: $\beta = \frac{N}{\sum |s(d)-\theta|}$. We denote this method as *Laplace* below.

Asymmetric Laplace Distributions

An asymmetric Laplace is fit to each of the class-conditional densities using the maximum likelihood estimation procedure described in Section 3.2.3 above. As with the asymmetric Gaussian, intervals between adjacent scores are divided into 10 pieces for testing candidate θs. This method is denoted as *A. Laplace* below.

Logistic Regression

This method is the first of two methods we evaluated that directly fit the posterior, $P(+|s(d))$. Both methods restrict the set of families to a two-parameter sigmoid family; they differ primarily in their model of class labels. As opposed to the above methods, one can argue that an additional boon of these methods is they completely preserve the ranking given by the classifier. When this is desired, these methods may be more appropriate. The previous methods will mostly preserve the rankings, but they can deviate if the data dictate it. Thus, they may model the data behavior better at the cost of departing from a monotonicity constraint in the output of the classifier.

Lewis & Gale [LG94] use logistic regression to recalibrate naïve Bayes for subsequent use in active learning. The model they use is:

$$P(+|s(d)) = \frac{\exp(a + b\, s(d))}{1 + \exp(a + b\, s(d))}. \tag{3.23}$$

Instead of using the probabilities directly output by the classifier, they use the loglikelihood ratio of the probabilities, $\log \frac{P(d|+)}{P(d|-)}$, as the score $s(d)$. Instead of using this below, we will use the log-odds ratio. This does not affect the model as it simply shifts all of the scores by a constant determined by the priors. We refer to this method as *LogReg* below.

Logistic Regression with Noisy Class Labels

Platt [Pla99] proposes a framework that extends the logistic regression model above to incorporate noisy class labels and uses it to produce probability estimates from the raw output of an SVM.

This model differs from the *LogReg* model only in how the parameters are estimated. The parameters are still fit using maximum likelihood estimation, but a model of noisy class labels is used in addition to allow for the possibility that the class was mislabeled. The noise is modeled by assuming there is a finite probability of mislabeling a positive example and of mislabeling a negative example; these two noise estimates are determined by the number of positive examples and the number of negative examples (using Bayes' rule to infer the probability of incorrect label).

Even though the performance of this model would not be expected to deviate much from *LogReg*, we evaluate it for completeness. We refer to this method below as *LR+Noise*.

Data

We examined several corpora, including the *MSN Web Directory* (13 classes), *Reuters* (10 classes), and *TREC-AP* (20 classes). More details about the data are given in Section 6.2.

Classifiers

We selected two classifiers for evaluation — a linear SVM classifier, which is a discriminative classifier that does not normally output probability values, and a naïve Bayes classifier, whose probability outputs are often poor [Ben00, DP96] but can be improved [Ben00, ZE01, ZE02].

SVM

For linear SVMs, we use the *Smox* toolkit which is based on Platt's **S**equential **M**inimal **O**ptimization algorithm. The features were represented as continuous values. We used the raw output score of the SVM as $s(d)$ since it has been shown to be appropriate before [Pla99]. The normal decision threshold (assuming we are seeking to minimize errors) for this classifier is at zero.

Naïve Bayes

The *naï ve Bayes* classifier model is a multinomial model [MN98]. We smoothed word and class probabilities using a Bayesian estimate (with the word prior) and a Laplace m-estimate, respectively. For more details, see Section 6.3.4. We use the log-odds estimated by the classifier as $s(d)$. The normal decision threshold is at zero.

Performance Measures

We use log-loss [Goo52] and squared error [Bri50, DF86] to evaluate the quality of the probability estimates. Both of these are *proper scoring rules* [DF83, DF86] in the sense that a classifier's view of its expected performance is maximized when the classifier actually issues a probability of $\hat{p}$ when it assesses the probability to be $\hat{p}$, *i.e.* the classifier cannot expect to gain from "hedging its bets".

For a document d with class $c(d) \in \{+, -\}$ (*i.e.* the data have known labels and not probabilities), log-loss is defined as:

$$\delta(c(d), +) \log P(+|d) + \delta(c(d), -) \log P(-|d) \tag{3.24}$$

where $\delta(a, b) \doteq 1$ if $a = b$ and 0 otherwise. The squared error is:

$$\delta(c(d), +)(1 - P(+|d))^2 + \delta(c(d), -)(1 - P(-|d))^2. \tag{3.25}$$

When the class of a document is correctly predicted with a probability of one, log-loss is zero and squared error is zero. When the class of a document is *incorrectly* predicted with a probability of one, log-loss is $-\infty$ and squared error is one. Thus, both measures assess how close an estimate comes to correctly predicting the item's class but vary in how harshly incorrect predictions are penalized.

We report only the sum of these measures and omit the averages for space. Their averages, average log-loss and mean squared error (MSE), can be computed from these totals by dividing by the number of binary decisions in a corpus. Note that the log-loss numbers given in this chapter are given as log base 2.

In addition, we also compare the error of the classifiers at their default thresholds and with the probabilities. This evaluates how the probability estimates have improved with respect to the decision threshold $P(+|d) = 0.5$. Thus, error *only* indicates how the methods would perform if a false positive was penalized the same as a false negative and not the general quality of the probability estimates. It is presented simply to provide the reader with a more complete understanding of the empirical tendencies of the methods.

We use a standard paired micro sign test [YL99] to determine statistical significance in the difference of all measures. Only pairs that the methods disagree on are used in the sign test. This test compares pairs of scores from two systems with the null hypothesis that the number of items they disagree on are binomially distributed. We use a significance level of $p = 0.01$.

Naïve Bayes

	Log-loss	Error^2	Errors
MSN Web			
Gauss	-60656.41	10503.30	10754
A.Gauss	-57262.26	8727.47	9675
Laplace	-45363.84	8617.59	10927
A.Laplace	-36765.88	**6407.84**†	<u>**8350**</u>
LogReg	-36470.99	6525.47	8540
LR+Noise	**-36468.18**	6534.61	8563
naï ve Bayes	-1098900.83	17117.50	17834
Reuters			
Gauss	-5523.14	1124.17	1654
A.Gauss	-4929.12	652.67	888
Laplace	-5677.68	1157.33	1416
A.Laplace	**-3106.95**‡	**554.37**‡	<u>**726**</u>
LogReg	-3375.63	603.20	786
LR+Noise	-3374.15	604.80	785
naï ve Bayes	-52184.52	1969.41	2121
TREC-AP			
Gauss	-57872.57	8431.89	9705
A.Gauss	-66009.43	7826.99	8865
Laplace	-61548.42	9571.29	11442
A.Laplace	-48711.55	**7251.87**‡	<u>**8642**</u>
LogReg	**-48250.81**	7540.60	8797
LR+Noise	-48251.51	7544.84	8801
naï ve Bayes	-1903487.10	41770.21	43661

SVM

	Log-loss	Error^2	Errors
MSN Web			
Gauss	-54463.32	9090.57	10555
A. Gauss	-44363.70	6907.79	8375
Laplace	-42429.25	7669.75	10201
A. Laplace	-31133.83	<u>**5003.32**</u>	<u>**6170**</u>
LogReg	**-30209.36**	5158.74	6480
LR+Noise	-30294.01	5209.80	6551
Linear SVM	N/A	N/A	6602
Reuters			
Gauss	-3955.33	589.25	735
A. Gauss	-4580.46	428.21	532
Laplace	-3569.36	640.19	770
A. Laplace	-2599.28	412.75	**505**
LogReg	-2575.85	**407.48**	509
LR+Noise	**-2567.68**	408.82	516
Linear SVM	N/A	N/A	516
TREC-AP			
Gauss	-54620.94	6525.71	7321
A. Gauss	-77729.49	6062.64	6639
Laplace	-54543.19	7508.37	9033
A. Laplace	-48414.39	**5761.25**‡	**6572**‡
LogReg	-48285.56	5914.04	6791
LR+Noise	**-48214.96**	5919.25	6794
Linear SVM	N/A	N/A	6718

Table 3.2: (a) Results for naï ve Bayes (*left*) and (b) SVM (*right*). The best entry for a corpus is in bold. Entries that are statistically signifi cantly better than all other entries are underlined. A † denotes the method is signifi cantly better than all other methods except for *naïve Bayes*. A ‡ denotes the entry is signifi cantly better than all other methods except for *A. Gauss* (and *naïve Bayes* for the table on the left). The reason for this distinction in signifi cance tests is described in the text.

Experimental Methodology

As the categories under consideration in the experiments are not mutually exclusive, the classification was done by training n binary classifiers, where n is the number of classes.

In order to generate the scores that each method uses to fit its probability estimates, we use five-fold cross-validation on the training data. We note that even though it is computationally efficient to perform leave-one-out cross-validation for the naïve Bayes classifier, this may not be desirable since the distribution of scores can be skewed as a result. Of course, as with any application of n-fold cross-validation, it is also possible to bias the results by holding n too low and underestimating the performance of the final classifier.

Results & Discussion

The results for recalibrating naïve Bayes are given in Table 3.2a. Table 3.2b gives results for producing probabilistic outputs for SVMs.

We start with general observations that result from examining the performance of these methods over the various corpora. The first is that *A. Laplace*, *LR+Noise*, and *LogReg*, quite clearly outperform the other methods. There is usually little difference between the performance of *LR+Noise* and *LogReg* (both as shown here and on a decision by decision basis), but this is unsurprising since *LR+Noise* just adds noisy class labels to the *LogReg* model. With respect to the three different measures, *LR+Noise* and *LogReg* tend to perform slightly better (but never significantly) than *A. Laplace* at some tasks with respect to log-loss and squared error. However, *A. Laplace* always produces the least number of errors for all of the tasks, though at times the degree of improvement is not significant.

In order to give the reader a better sense of the behavior of these methods, Figures 3.5-3.6 show the fits produced by the most competitive of these methods versus the actual data behavior (as estimated nonparametrically by binning) for class *Earn* in Reuters. Figure 3.5 shows the class-conditional densities, and thus only *A. Laplace* is shown since *LogReg* fits the posterior directly. Figure 3.6 shows the estimations of the log-odds, (*i.e.* $\log \frac{P(Earn|s(d))}{P(\neg Earn|s(d))}$). Viewing the log-odds (rather than the posterior) usually enables errors in estimation to be detected by the eye more easily.

We can break things down as the sign test does and just look at wins and losses on the items that the methods disagree on. Looked at in this way only two methods (*naï ve Bayes* and *A. Gauss*) ever have more pairwise wins than *A. Laplace*; those two sometimes have more pairwise wins on log-loss and squared error even though the total never wins (*i.e.* they are dragged down by heavy penalties).

In addition, this comparison of pairwise wins means that for those cases where *LogReg* and *LR+Noise* have better scores than *A. Laplace*, it would not be deemed significant by the sign test at any level since they do not have more wins. For example, of the 130K binary decisions over the MSN Web dataset, *A. Laplace* had approximately 101K pairwise wins versus *LogReg* and *LR+Noise*. No method ever had more pairwise wins than *A. Laplace* for the error comparison nor did any method ever achieve a better total.

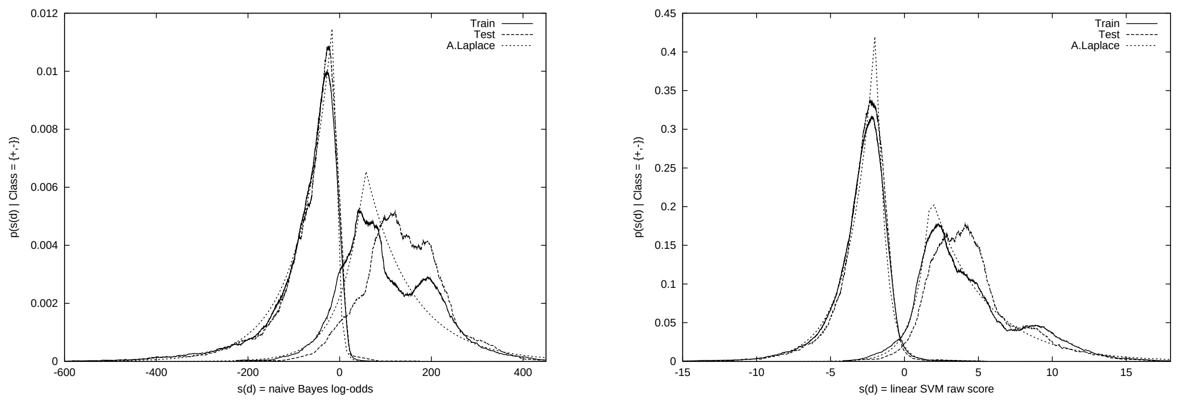


Figure 3.5: The empirical distribution of classifi er scores for documents in the training and the test set for class *Earn* in Reuters. Also shown is the fi t of the asymmetric Laplace distribution to the training score distribution. The *positive* class (*i.e. Earn*) is the distribution on the right in each graph, and the *negative* class (*i.e.* ¬*Earn*) is that on the left in each graph.

The basic observation made about naïve Bayes in previous work is that it tends to produce estimates very close to zero and one [Ben00, LG94]. This means if it tends to be right enough of the time, it will produce results that do not appear significant in a sign test that ignores size of difference (as the one here). The totals of the squared error and log-loss bear out the previous observation that "when it's wrong it's *really* wrong".

There are several interesting points about the performance of the asymmetric distributions as well. First, *A. Gauss* performs poorly because (similar to naïve Bayes) there are some examples where it is penalized a large amount. This behavior results from a general tendency to perform like the picture shown in Figure 3.4 (note the crossover at the tails). While the asymmetric Gaussian tends to place the mode much more accurately than a symmetric Gaussian, its asymmetric flexibility combined with its distance function causes it to distribute too much mass to the outside tails while failing to fit around the mode accurately enough to compensate. Figure 3.4 is actually a result of fitting the two distributions to real data. As a result, at the tails there can be a large discrepancy between the likelihood of belonging to each class. Thus when there are no outliers *A. Gauss* can perform quite

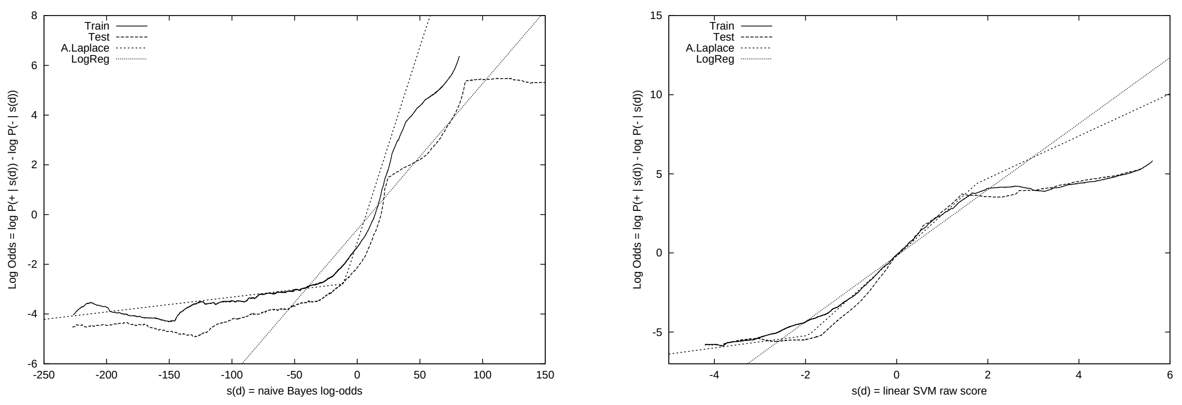


Figure 3.6: The fit produced by various methods compared to the empirical log-odds of the training data for class *Earn* in Reuters.

competitively, but when there is an outlier *A. Gauss* is penalized quite heavily. There are enough such cases overall that it seems clearly inferior to the top three methods.

However, the asymmetric Laplace places much more emphasis around the mode (Figure 3.5) because of the different distance function (think of the "sharp peak" of an exponential). As a result most of the mass stays centered around the mode, while the asymmetric parameters still allow more flexibility than the standard Laplace. Since the standard Laplace also corresponds to a piecewise fit in the log-odds space, this highlights that part of the power of the asymmetric methods is their sensitivity in placing the knots at the actual modes — rather than the symmetric assumption that the means correspond to the modes. Additionally, the asymmetric methods have greater flexibility in fitting the slopes of the line segments as well. Even in cases where the test distribution differs from the training distribution (Figure 3.5), *A. Laplace* still yields a solution that gives a better fit than *LogReg* (Figure 3.6), the next best competitor.

Finally, we can make a few observations about the usefulness of the various performance metrics. First, log-loss only awards a finite amount of credit as the degree to which something is correct improves (*i.e.* there are diminishing returns as it approaches zero), but it can infinitely penalize for a wrong estimate. Thus, it is possible for one outlier to skew the totals, but misclassifying this example may not matter for any but a handful of actual utility functions used in practice. Secondly, squared error has a weakness in the other direction. That is, its penalty and reward are bounded in $[0, 1]$, but if the number of errors is small enough, it is possible for a method to appear better when it is producing what we generally consider unhelpful probability estimates. For example, consider a method that only estimates probabilities as zero or one (which naïve Bayes tends toward but doesn't quite reach if you use smoothing). This method could win according to squared error, but with just one error it would never perform better on log-loss than any method that assigns some non-zero probability to each outcome. For these reasons, we recommend that neither of these are used in isolation as they each give slightly different insights to the quality of the estimates produced. These observations are straightforward from the definitions but are underscored by the evaluation.

3.2.5 Related Work

Parametric models have been employed to obtain probability estimates in several areas relevant to text classification. Lewis & Gale [LG94] use logistic regression to recalibrate naïve Bayes though the quality of the probability estimates are not directly evaluated; it is simply performed as an intermediate step in active learning. Manmatha *et al.* [MRF01] introduced appropriate models for producing probability estimates from relevance scores returned from search engines and demonstrated how the resulting probability estimates could be subsequently employed to combine the outputs of several search engines. They use a different parametric distribution for the relevant and irrelevant classes but do not pursue two-sided asymmetric distributions for a single class, as described here. They also survey the long history of modeling the relevance scores of search engines. Our work is similar in flavor to these previous attempts to model search engine scores, but we target text classifier outputs which we have found demonstrate a different type of score distribution behavior because of the role of training data.

Zadrozny & Elkan [ZE01] provide what can be thought of as a type of pruning targeted at improving the reliability of probability estimates obtained from decision trees (termed *curtailment*) and a non-parametric method for recalibrating naïve Bayes. In more recent work [ZE02], they investigate using a semi-parametric method that uses a monotonic piecewise-constant fit to the data and apply the method to naïve Bayes and a linear SVM. While they compared their methods to other parametric methods based on symmetry, they fail to provide significance test results. Our work provides asymmetric parametric methods that complement the non-parametric and semi-parametric methods they propose when data scarcity is an issue. In addition, their methods reduce the resolution of the scores output by the classifier (the number of distinct values output), but the methods here do not have such a weakness since they are continuous functions.

Just as logistic regression allows the log-odds of the posterior distribution to be fit directly with a line, we could directly fit the log-odds of the posterior with a piecewise linear function (a spline) instead of indirectly doing the same thing by fitting the asymmetric

Laplace. In a follow-up to our work, Zhang and Yang [ZY04] did just that and obtained an approach with even more power while retaining asymmetry.

There is a variety of other work that this section of the dissertation extends. Platt [Pla99] uses a logistic regression framework that models noisy class labels to produce probabilities from the raw output of an SVM. His work showed that this post-processing method not only can produce probability estimates of similar quality to SVMs directly trained to produce probabilities (regularized likelihood kernel methods), but it also tends to produce sparser kernels (which generalize better). Finally, recalibrating poorly calibrated classifiers is not a new problem. Lindley et al. [LTB79] first proposed the idea of recalibrating classifiers, and DeGroot & Fienberg [DF83, DF86] gave the now accepted standard formalization for the problem of assessing calibration initiated by others [Bri50, Win69].

3.2.6 Summary of Recalibration Methods

We have reviewed a wide variety of parametric methods for producing probability estimates from the raw scores of a discriminative classifier and for recalibrating an uncalibrated probabilistic classifier. In addition, we have introduced two new families that attempt to capitalize on the asymmetric behavior that tends to arise from learning a discrimination function. We have given an efficient way to estimate the parameters of these distributions.

While these distributions attempt to strike a balance between the generalization power of parametric distributions and the flexibility that the added asymmetric parameters give, the asymmetric Gaussian appears to have too great of an emphasis away from the modes. In striking contrast, the asymmetric Laplace distribution appears to be preferable over several large text domains and a variety of performance measures to the primary competing parametric methods, though comparable performance is sometimes achieved with one of two varieties of logistic regression.

Given the ease of estimating the parameters of this distribution, it is a good first choice for producing quality probability estimates when training data is scarce. When training data is plentiful, isotonic regression methods [ZE02] may yield better results. These estimates can then be used as input to a classifier combination algorithm. Additionally, the work in this section has demonstrated both empirically and from a theoretical standpoint, that classifiers tend to yield asymmetric score distributions and are unlikely to give rise to normal distributions as assumed in combination work by Kahn [Kah04].

Chapter 4

Locality

We remind the reader that our goal is ultimately to outperform a global linear combination of classifier log-odds, probabilities, or scores. Thus a simple definition of locality is any combination function where the weight on such classifier outputs cannot be expressed as a global weight. However, this section seeks to motivate *why* we may want to vary the weight we place on a classifier, the implications this carries, and, in the ideal case, what statistics should be most influential in determining the weight.

To accomplish this, we approach the problem from a series of simplified views. For example, we can consider when there is a single base classifier, when the base classifiers are both calibrated and conditionally independent given the class, when the base classifiers are uncalibrated but remain conditionally independent, and finally when the base classifiers are neither calibrated nor conditionally independent. Likewise, we can also simplify the formulation by considering the form the solution would take for a particular combination model if we were given not only the class labels for each training example, but the "true" posterior $P(c \mid x)$ as well. Examining the combination from these simplified viewpoints will allow us to separate what terms are hard to estimate, due to sparse data or missing information, and how the information should be used.

To enable clarity in the rest of the chapter, we start by discussing the interpretation of the "true" posterior for classification. Following this, we briefly return to calibration, and demonstrate that while it is a key characteristic as discussed in Chapter 3, it does not address one of the primary challenges of classifier combination — namely estimating the dependencies of the classifier outputs.

Furthermore, it does not fully take advantage of the fact that the reliability of a classifier's predictions can vary across the input space. In this chapter, we argue that considering the locally changing interactions among classifiers is key to improving classifier combination performance. Therefore, we turn our focus on these issues by motivating and then defining the concepts of local dependence, reliability, and variance.

4.1 "True" Posteriors, Log-odds, and Confidences

For clarity, we can use a common situation researchers encounter to discuss concepts key to the rest of this chapter in a simplified manner. Consider the following tasks often faced during peer review[1]:

- "Make a recommendation accept/reject."
- "Rate this paper from 0 to 5, where 0 is definitely reject and 5 is definitely accept."
- "State your confidence on a 0 to 5 scale in your review."

When a reviewer answers the first question, she is making a classification prediction regarding the paper. The answer to the second question is the posterior probability after reading the paper that the reviewer assesses regarding "Accept/Reject". In fact, if we were to center the 0 to 5 scales on 0 so that they run from -2.5 to 2.5, then the score would be similar to log-odds. A negative value would indicate that "Reject" has a higher posterior probability and a positive value would indicate that "Accept" has a higher posterior probability. Presumably, if the reviewer is acting consistently her recommendation will behave similarly.

Next, the reviewer states her confidence — which intuitively is a self-assessment of her expertise and mathematically is a statement about how strongly she believes the posterior she gave is correct. In other words it is a second-order summary of the uncertainty the reviewer has about her classification. Consider if instead of making the reviewer specify her uncertainty, we allowed her to specify an entire distribution expressing her belief $p(P(c \mid x) = z \mid x)$. Then when she is first forced to summarize her uncertainty via the rating, the typical and self-consistent approach is to predict the expected value of the distribution: $\hat{P}(c \mid x) = \int z\, p(P(c \mid x) = z \mid x)\, dz$. However, as the reader is well aware, the mean of a distribution does not fully summarize the distribution. Presumably, as the reviewer receives more information or perceives she has all necessary information because of her expertise, her confidence that the expected value fully summarizes her uncertainty will become quite high. Therefore a reasonable measure for confidence is to treat it as an (inverse) measure

[1]We assume that the reviewer assigns these outcomes conditionally on the paper reviewed but independent of the content of other papers.

of the variance of $p(P(c \mid x) = z \mid x)$ — or the spread of the distribution from its mean value.

While researchers commonly perform peer reviews and understand the intuitive notions it involves, they are often perplexed when trying to mechanically produce such estimates from a classifier. This occurs for many reasons — some of which can be highlighted in the same example. What does it mean to say a paper has a “true class” of “Reject”? Furthermore, what would it mean to say that the probability of reject after reading the paper is 0.8. In part, the confusion results from confusing a Bayesian notion of subjective probabilities with a frequentist notion of empirical probabilities. In the Bayesian scheme (including the preceding paragraph), probability theory is simply a useful tool to convey uncertainty in a manner that follows certain rules of self-coherence.

Whereas, in the frequentist notion, we must have at least an imaginary concept of a repeatable experiment. How would one sample “similar” papers? A slightly more tenable viewpoint would be to sample reject/accept opinions from “similar” people. In our example, “similar” people would amount to something like “people with expertise like those on the editorial board”. In this view the $P(\text{accept} \mid \text{paper})$ is the empirical frequency as we sample more opinions , *i.e.* $\lim_{N\to\infty} \frac{|R_i = \text{accept}|}{N}$ and the belief distribution $p(P(c \mid x) = z \mid x)$ is an estimate of the probability the limit will take on each of these values given the evidence. In other words, a statement that $\hat{P}(\text{accept} \mid \text{paper}) = 0.80$ means that the estimator believes 80% of a “similar” population would decide to accept this paper.

The frequentist explanation is not necessary, but it makes it easier to conceive of the type of uncertainty we may want to capture. For our applications, the task is often to predict topic and the population being sampled can be thought of as all potential users of the application. Some documents are less clearly on one topic, and therefore, there will be higher disagreement on those documents. Thus documents will rarely ever have a true posterior of 1 or 0. It is simply often treated as such because our training data typically only consists of a single class label for each point.

4.2 Calibration & Locality

Similar to other works, we assume we can obtain from each classifier $\mathcal{C}_i$ a conditional probability estimate, $\hat{\pi}_i(c) = \hat{P}_{\mathcal{C}_i}(c|x)$, and a log-odds like score, $\hat{\lambda}_i(c) = \log \frac{\hat{\pi}_i(c)}{1-\hat{\pi}_i(c)}$, either directly or by postprocessing as discussed in Section 3.2. Thus, our approach can either be viewed as a method of combining probability forecasters or as combining classifiers where we make assumptions that allow us to model the "internal probabilities" the classifiers are implicitly utilizing in making their decisions.

Viewing this as a problem of combining log-odds, the problem is equivalent to performing inference in the model depicted in Figure 4.1. Here we have simply replaced the difficult problem of estimating $p(\hat{\lambda}_1, \ldots, \hat{\lambda}_n \mid c)$ by what seems to be the equally hard problem of estimating $p(\hat{\lambda}_1, \ldots, \hat{\lambda}_n \mid \lambda)$.[2] However, when we start to make simplifying assumptions, the difference becomes more obvious. As we noted, Kahn [Kah04] worked with a model where the class-conditional classifier outputs were assumed to be Gaussian, but even in the case of a single classifier this model cannot support the type of asymmetric behavior that we see in practice as shown in Section 3.2.

As mentioned in Chapter 3, if we restrict the number of base classifiers to a single classifier, the problem becomes equivalent to recalibrating that classifier. Let's assume we have a single classifier whose log-odds estimates, $\hat{\lambda}$, are distributed normally around the true log-odds, $\hat{\lambda} \sim N(\lambda, 1)$. As shown in the left of Figure 4.2, even when we assume the prior on the true log-odds is a simple Gaussian, the resulting class-conditional distribution $p(\hat{\lambda} \mid c)$ has asymmetric properties similar to what is seen in practice (see Chapter 3).

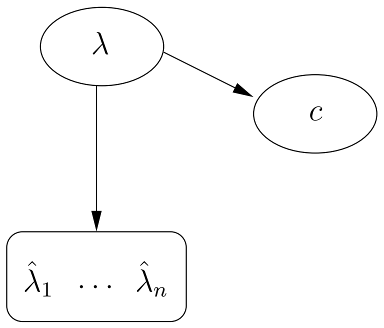


Figure 4.1: Classifi er combination can be thought of as combining estimates of each classifi er's estimate of the log-odds, $\hat{\lambda}_i$, via the latent variable representing the true log-odds, λ, to improve the prediction of the class c. That is via, $p(\hat{\lambda}_1, \ldots, \hat{\lambda}_n \mid c) = \int_{-\infty}^{\infty} p(\hat{\lambda}, \ldots, \hat{\lambda}_n \mid \lambda) p(\lambda \mid c)\, d\lambda$.

If we were to continue to work with this formal model, different formulations would focus on different characteristics. For example, the classifier's predictions may not be centered on the true log-odds, but instead the predictions may show a bias to the left or right. Thus we could easily include a factor that allowed us to model a systematic shift in each classifier indicating overconfidence or underconfidence. Likewise, where we used $\hat{\lambda} \sim N(\lambda, 1)$ in the example above, we might instead choose to use a different standard

[2]Note that by definition $P(c|\lambda) = (1 + \exp\{-c\lambda\})^{-1}$ and we can use Bayes' rule to invert it.

deviation in various parts of the space. For example, areas where we have large amounts of training data could use a small standard deviation as we achieve a tighter confidence around our prediction. Note that this approach implies that the calibration of the classifier varies locally according to the estimated confidence bound on the true log-odds.

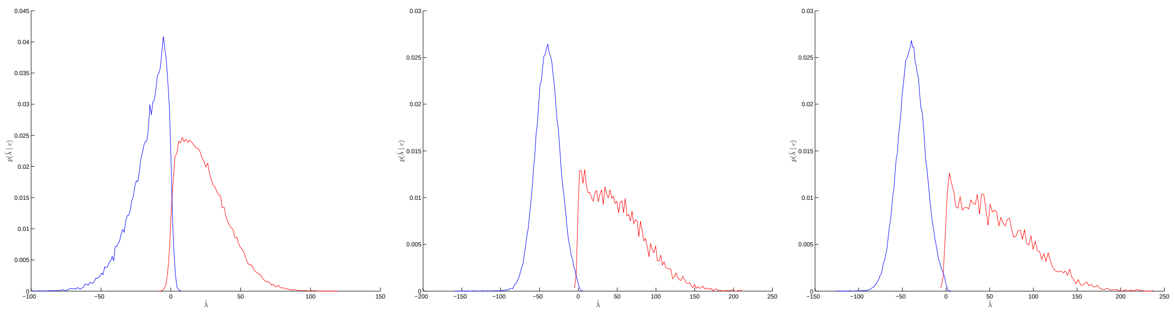


Figure 4.2: A few examples of the distribution of $p(\hat{\lambda} \mid c)$ for various choices of the prior on the true log-odds, $p(\lambda)$, when the classifi er's predictions are distributed normally around the true log-odds, $\hat{\lambda} \sim N(\lambda, 1)$. The prior used is a single Gaussian (*left*), a mixture of two Gaussians (*middle*), and a mixture of three Gaussians (*right*). 100K samples were drawn from each distribution. The asymmetry of the resulting distributions is very reminiscent of those seen in practice as shown in Section 3.2.

We do not directly work with such a graphical model, but instead use it to point out the recurring themes that we incorporate into our work. These properties can be formulated in terms of the probability estimates the classifiers emit or in terms of the log-odds. Readers interested in pursuing this model may also wish to consider implications discussed later in this chapter. For example, a simplified version of this model would assume the classifiers' log-odds estimates are independent given the true log-odds. Finally, rather than use such an independence assumption globally, we could posit that there exists a mixture of different regions where the parameters are specific to the region.

4.3 Dependence & Locality

Returning to our example (see Section 1.2.1) of a feature space where two mutually exclusive and exhaustive subsets are independent given the class, consider if we now have one classifier based on each subset and each outputs the log-odds using the true posterior based only on its subset (*i.e.*, $P(c \mid x_{i,1}, \ldots, x_{i,k})$ and $P(c \mid x_{i,k+1}, \ldots, x_{i,n})$). It can easily be shown by factorizing the joint that the optimal combination of these classifiers log-odds is

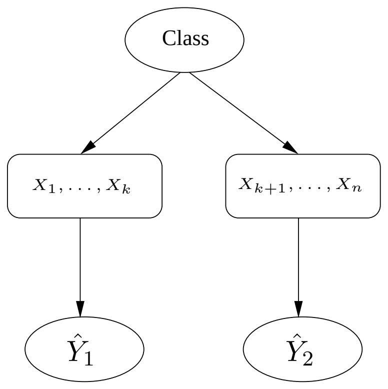


Figure 4.3: An influence diagram for two classifiers whose optimal combination is to allow the output of each ($\hat{Y}_1$ and $\hat{Y}_2$) to contribute independently to the final prediction. The input dimensions $X_1, \ldots, X_k$ are independent of dimensions $X_{k+1}, \ldots, X_n$ given the class variable; though, the interactions within the two feature sets may be arbitrarily complex (which is why they are depicted as within one box). One classifier's predictions ($\hat{Y}_1$) depend only on the values the first feature set takes ($X_1, \ldots, X_k$) while the other classifier's predictions ($\hat{Y}_2$) depend only on the values the second feature set takes ($X_{k+1}, \ldots, X_n$).

simply their sum with the extra prior removed.[3] This situation is depicted with an influence diagram in Figure 4.3. However, if the first classifier additionally modeled one of the attributes from the second subset, $P(c \mid x_{i,1}, \ldots, x_{i,k+1})$, this is no longer the optimal combination. The reason for this is the well-known fact that their common dependence must be factored out. In general then, we will have to consider the dependence of the outputs. This is not surprising given the theoretical proof of improvement reviewed in Section 3.1 required we estimate the joint distribution, and it is necessary to consider dependence when estimating the joint.

We can make this more explicit by considering a case when the features are partitioned into 3 subsets: $\mathbf{X}_1$, $\mathbf{X}_2$, and $\mathbf{S}$. Assume Classifier 1 uses $\mathbf{X}_1$ and $\mathbf{S}$ and outputs an uncalibrated estimate based on these variables of the form: $\hat{\lambda}_1 = a_1 \log \frac{P(+|\mathbf{x}_1,\mathbf{s})}{P(-|\mathbf{x}_1,\mathbf{s})} + b_1$. Likewise Classifier 2 outputs $\hat{\lambda}_2 = a_2 \log \frac{P(+|\mathbf{x}_2,\mathbf{s})}{P(-|\mathbf{x}_2,\mathbf{s})} + b_2$. Consider the optimal combination of these two classifiers. Given the prior log-ratio $\rho = \log \frac{P(+)}{P(-)}$ and the measure of the shared information $\lambda_S = \log \frac{P(+|\mathbf{s})}{P(-|\mathbf{s})}$, the optimal combination can be written as a linear combination with a non-constant bias term dependent only on the shared information:

[3]If given probability estimates, it is the normalized product.

$$\begin{aligned}
\lambda(\mathbf{x}) &= a_1^{-1}\left[\hat{\lambda}_1(\mathbf{x}) - b_1\right] + a_2^{-1}\left[\hat{\lambda}_2(\mathbf{x}) - b_2\right] - \lambda_S(\mathbf{x}) - \rho && (4.1)\\
&= w_1\hat{\lambda}_1 + w_2\hat{\lambda}_2 + b^*(\mathbf{x}) && (4.2)\\
&\quad \text{where } w_1 = a_1^{-1}, w_2 = a_2^{-1}, b^*(\mathbf{x}) = -\lambda_S(\mathbf{x}) - \rho - \frac{b_1}{a_1} - \frac{b_2}{a_2}. && (4.3)
\end{aligned}$$

Notice that the non-constant part of the bias term will correct the sign of the combined decision whenever the amount of double-counted information swamps that presented by the conditionally independent information each classifier contributes. This example illustrates that, rather than trying to directly estimate terms such as λ_S, we can simply combine the log-odds with a linear combination where all the weights except the bias are constant. Then the weights can be interpreted as implicitly representing these interactions. Thus by introducing non-constant weights we can capture a range of dependency interactions.

4.4 Variance, Sensitivity, & Locality

Now we turn to the issue of variance. As this section will demonstrate, the variance of several quantities will be of interest to us. These will be the sensitivity of the model, the covariance of the model's estimates with the true outputs, and the variance in error prediction. In brief, we will have to consider the sensitivity or variance in the model's output when considering how to normalize the current outputs, the covariance with the true outputs to determine how to rescale the normalized estimates, and the variance in error prediction both as a term to minimize and through its connection to the other two via the variance of the true outputs. We will elaborate on these more as we proceed to ease the task of the reader.

We start off by considering what would happen if our classifier had total information. In this case, all of the predictions would be based on the true posterior, and the classifier would suffer only the Bayes error. This notion of variance then is the generalization of refinement from the calibration literature. This is essentially a measure of the amount of information on average that was lacking to fully explain the deterministic portion of the output value. How might we extend this to be a reasonable definition of variance around a point in the reliability diagram?

From our discussion of the model incorporating the latent variable of the true log-odds in Figure 4.1, it is obvious. The prediction error variance is the spread around the average deviation from the true log-odds. For some simulated data, this term is computable. For real data, however, we must simply base our estimates on whether the example was labeled as belonging to the class or not. This has led some authors to claim that variance is not well-defined for classification, but it would be more appropriate to say it is not computable. If we work with a formal probabilistic model then clearly our assumptions about the variance in the model can have an impact via inference over the latent true log-odds. Additionally, we can attempt to model this factor by assuming that if the estimates of the classifiers are changing rapidly with small changes in the input data (*sensitivity*) then most likely the true log-odds are not changing rapidly in a like fashion, and therefore the variance in prediction error is high.

To clarify this further, consider the following setup. We are given a single base classifier's predictions, $\hat{\lambda}$, and we would like to find parameters a and b for a linear transformation such that $\hat{\lambda}^* = a\,\hat{\lambda} + b$ minimizes the expected squared error with the true log-odds. That is $\operatorname{argmin}_{a,b} = E\left[\left(a\,\hat{\lambda}(\mathbf{x}) + b - \lambda(\mathbf{x})\right)^2\right]$. Note that there is no assumption that the base classifier or the true log-odds are linear — simply that we want the best linear transformation. If the base classifier outputs non-linear predictions then after the transformation the transformed outputs will also be non-linear. We can solve this following the solution for the standard regression problem.

$$\begin{aligned}
E\left[\left(\hat{\lambda}^*(\mathbf{x}) - \lambda(\mathbf{x})\right)^2\right] &= \int p(\mathbf{x})\left[\hat{\lambda}^*(\mathbf{x}) - \lambda(\mathbf{x})\right]^2 d\mathbf{x} && (4.4)\\
&= \int p(\mathbf{x})\left[a\,\hat{\lambda}(\mathbf{x}) + b - \lambda(\mathbf{x})\right]^2 d\mathbf{x} && (4.5)\\
&= a^2\int p(\mathbf{x})\hat{\lambda}^2(\mathbf{x})\,d\mathbf{x} + b^2\int p(\mathbf{x})\,d\mathbf{x} + \int p(\mathbf{x})\lambda^2(\mathbf{x})\,d\mathbf{x}\\
&\quad +2ab\int p(\mathbf{x})\hat{\lambda}(\mathbf{x})\,d\mathbf{x} - 2a\int p(\mathbf{x})\hat{\lambda}(\mathbf{x})\lambda(\mathbf{x})\,d\mathbf{x}\\
&\quad -2b\int p(\mathbf{x})\lambda(\mathbf{x})\,d\mathbf{x} && (4.6)\\
\text{By replacement and since } &\int p(\mathbf{x})\,d\mathbf{x} = 1\\
&= a^2E\left[\hat{\lambda}^2(\mathbf{x})\right] + b^2 + E\left[\lambda^2(\mathbf{x})\right]\\
&\quad +2abE\left[\hat{\lambda}(\mathbf{x})\right] - 2aE\left[\hat{\lambda}(\mathbf{x})\lambda(\mathbf{x})\right] - 2bE[\lambda(\mathbf{x})] && (4.7)\\
\text{Which yields}\\
\frac{\partial E\left[\left(\hat{\lambda}^*(\mathbf{x}) - \lambda(\mathbf{x})\right)^2\right]}{\partial a} &= 2aE\left[\hat{\lambda}^2(\mathbf{x})\right] + 2bE\left[\hat{\lambda}(\mathbf{x})\right] - 2E\left[\hat{\lambda}(\mathbf{x})\lambda(\mathbf{x})\right] && (4.8)\\
\frac{\partial E\left[\left(\hat{\lambda}^*(\mathbf{x}) - \lambda(\mathbf{x})\right)^2\right]}{\partial b} &= 2b + 2aE\left[\hat{\lambda}(\mathbf{x})\right] - 2E[\lambda(\mathbf{x})] && (4.9)
\end{aligned}$$

Setting to zero and rearranging terms assuming $E\left[\hat{\lambda}^2(\mathbf{x})\right] \neq 0$ gives the system

$$a = \frac{E\left[\hat{\lambda}(\mathbf{x})\lambda(\mathbf{x})\right] - bE\left[\hat{\lambda}(\mathbf{x})\right]}{E\left[\hat{\lambda}^2(\mathbf{x})\right]} \tag{4.10}$$

$$b = E[\lambda(\mathbf{x})] - aE\left[\hat{\lambda}(\mathbf{x})\right] \tag{4.11}$$

Assuming $\text{VAR}\left[\hat{\lambda}(\mathbf{x})\right] \neq 0$ and solving gives

$$a = \frac{E\left[\hat{\lambda}(\mathbf{x})\lambda(\mathbf{x})\right] - E\left[\hat{\lambda}(\mathbf{x})\right]E[\lambda(\mathbf{x})]}{\text{VAR}\left[\hat{\lambda}(\mathbf{x})\right]} \tag{4.12}$$

$$= \frac{\text{COV}\left[\hat{\lambda}(\mathbf{x}), \lambda(\mathbf{x})\right]}{\text{VAR}\left[\hat{\lambda}(\mathbf{x})\right]} \tag{4.13}$$

$$b = E[\lambda(\mathbf{x})] - \frac{\text{COV}\left[\hat{\lambda}(\mathbf{x}), \lambda(\mathbf{x})\right]}{\text{VAR}\left[\hat{\lambda}(\mathbf{x})\right]}E\left[\hat{\lambda}(\mathbf{x})\right] \tag{4.14}$$

We can rewrite the final solution in a variety of forms to gain insight. For example, we can rewrite the solutions using the correlation coefficient between the true log-odds and estimated log-odds, $\rho_{\lambda,\hat{\lambda}}$, to write: $a = \frac{\sigma_\lambda}{\sigma_{\hat{\lambda}}}\rho_{\lambda,\hat{\lambda}}$, $b = E[\lambda(\mathbf{x})] - aE\left[\hat{\lambda}(\mathbf{x})\right]$. As can be seen from the example in Figure 4.4, the coefficient a corrects the predictions to be both correlated with and have a variance similar to the correct log-odds. Since the expected value of the corrected predictions will be $E\left[a\hat{\lambda} + b\right] = aE\left[\hat{\lambda}\right] + b = aE\left[\hat{\lambda}\right] + E[\lambda] - aE\left[\hat{\lambda}\right] = E[\lambda]$, the additive term b ensures the new predictions are weakly calibrated on average[4] — the average difference is zero but that does not imply $E\left[\hat{\lambda} - \lambda \mid \hat{\lambda}\right] = 0$ for all $\hat{\lambda}$. Also, note that if the predictions are independent of the true log-odds then $\rho_{\lambda,\hat{\lambda}} = 0$ and therefore $a = 0$, and the only correction that can be made is by predicting $b = E[\lambda]$ at all points.

Continuing in this vein, there are a variety of other facts that can be demonstrated. For example, the squared error of the corrected predictions is $E\left[\left(\hat{\lambda}^* - \lambda\right)^2\right] = \text{VAR}[\lambda]\,(1 - \rho^2_{\lambda,\hat{\lambda}})$ which is 0 iff $\rho_{\lambda,\hat{\lambda}} = \pm 1$, and therefore, the squared error of the corrected predictions goes to zero as the magnitude of the correlation between the original predictions and the true log-odds approaches 1.

However, rather than continuing along this line, we would like to examine the quantities used in deriving the solutions. As we have already mentioned b is related to a measure of calibration weighted by a. Now, we turn to $a = \frac{\text{COV}[\lambda,\hat{\lambda}]}{\text{VAR}[\hat{\lambda}]}$. First, we note that the

[4]By linearity of expectation, we have $E\left[\hat{\lambda}^* - \lambda\right] = E\left[a\hat{\lambda} + b\right] - E[\lambda] = E[\lambda] - E[\lambda] = 0$.

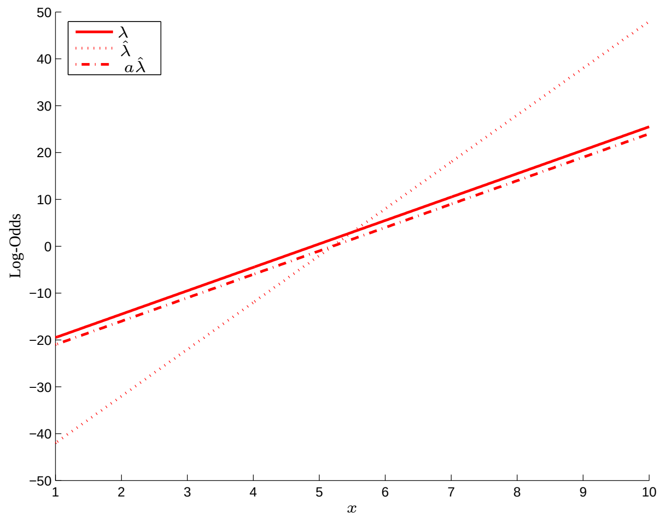


Figure 4.4: A simple example where the input space has a single dimension to illustrate the role of the ratio of standard deviations in $a = \frac{\sigma_\lambda}{\sigma_{\hat{\lambda}}}\rho_{\lambda,\hat{\lambda}}$. In the example, $p(x)$ is uniform over $[1, 10]$. In this example, the initial predictions are correct on average: $E[\lambda] = E\left[\hat{\lambda}\right]$. The predicted log-odds, $\hat{\lambda}$, are perfectly correlated with the true log-odds, λ. That is, $\rho_{\lambda,\hat{\lambda}} = 1$, but $a = 0.5$ and $b = 1.5$. As can be seen from the correction using just a, the coeffi cient forces the variation/slope of the predictions to behave on average like the true variation. The resulting correction by b must take into account the compression and rotation caused by a.

denominator, $\mathrm{VAR}\left[\hat{\lambda}\right]$, is a measure of the sensitivity of the model. That is, it captures how widely the estimates of the model vary regardless of the value of the true log-odds. Thus, given that model sensitivity is important for even this subcase of the combination problem, we expect it will play a role in the larger combination problem. Next, the covariance, $\mathrm{COV}\left[\lambda, \hat{\lambda}\right]$, captures whether the predictions vary in the same way as the true log-odds. The covariance is related to the spread around the average deviation by $\mathrm{VAR}\left[\hat{\lambda} - \lambda\right] = \mathrm{VAR}\left[\hat{\lambda}\right] + \mathrm{VAR}[\lambda] - 2\,\mathrm{COV}\left[\lambda, \hat{\lambda}\right]$. Thus, although the covariance is related to the variance of the error in prediction, it is unclear whether more can be gained in general in combination schemes from attempting to directly estimate the variance in error or the covariance.[5]

Next, as mentioned above the average difference is only a weak measure of calibration in that $E\left[\hat{\lambda} - \lambda\right] = 0$ does not imply $E\left[\hat{\lambda} - \lambda \mid \hat{\lambda}\right] = 0$, whereas the second condition is typically what is meant by well-calibrated as discussed earlier. However, we can consider

[5]As the reader is no doubt aware the expected squared error can be broken down into the variance in error and the square of the expected error as: $E\left[\left(\hat{\lambda} - \lambda\right)^2\right] = \mathrm{VAR}\left[\hat{\lambda} - \lambda\right] + E^2\left[\hat{\lambda} - \lambda\right]$. Therefore, even though the variance in error and average error determine the squared error, they are only indirectly related to the parameter values for a linear correction.

more local measures of calibration. In particular, we can consider using a linear correction when the parameters are determined locally. In order to do so, we need only define a distribution over the domain conditional on the current prediction point, $\mathbf{x_0}$. If we denote the locality or the neighborhood of the prediction point as $N(\mathbf{x_0})$, we can denote the local distribution as $p(\mathbf{x} \mid \mathbf{x} \in N(\mathbf{x_0}))$. Since this distribution integrates to unity, the derivation for the weights of the linear correction remain the same — the expectations, variance, and covariance are now computed using the local distribution. Likewise the parameters are now locally determined. It is up to the modeler how to represent locality. For example, if we define locality as a small window around the predicted value, $\hat{\lambda}$, then a locally linear correction will now be well-calibrated since each local correction will ensure $E\left[\hat{\lambda} - \lambda \mid \hat{\lambda}\right] = 0$.

Applying the linear correction method locally can be demonstrated concretely by generating some simple examples. We will consider an input space of one dimension where $p(x \mid c)$ is Gaussian and $P(c)$ is fixed. To draw points, we draw its class with probability $P(c)$ and then draw from the class-conditional distributions. 100 training points and 10000 hold-out points are drawn. We then fit a prediction model over the training data using either linear discriminant analysis (LDA) or quadratic discriminant analysis (QDA) and add-one-estimates for the priors [HTF01]. Locality is defined in these cases by choosing a fixed width window around each prediction point in the feature space. Finally, the hold-out data is used to estimate the correction factors globally or locally.

The first example generates the data with the same class-conditional variance and with $P(+) = 0.5$. The means were chosen randomly to obtain $p(x|-) \sim N(-0.4854, 1.6068)$ and $p(x|+) \sim N(0.3306, 1.6068)$. Using LDA the parameters are estimated to be $\hat{P}(+) = 0.4804$, $\hat{p}(x|-) \sim N(-0.2375, 1.7941)$, and $\hat{p}(x|+) \sim N(0.5184, 1.7941)$. We can visualize the true and estimated distributions as shown in Figure 4.5(a). The true and estimated posterior and log-odds are given in Figure 4.6.[6] Using a single global correction, we find that $a = 0.8295$ and $b = 0.1296$. In Figure 4.7, we see that local-estimation finds the same correction factors except at the edges where the holdout data is sparse. However, since the factors compensate for each other, even in this case the poor estimation does not hurt local correction (Figure 4.10a). Note that this also demonstrates how density in an area is inversely related to variance. Tresp & Taniguchi [TT95] exploit this fact when combining classifiers.

The second example generates the data with different class-conditional variances and with $P(+) = 0.5$. The means and variances were chosen randomly to obtain

[6]The reader can note from this example that by working with log-odds instead of in probability space, a global linear correction can capture far more of the typical behavior seen than in probability space where the functions are non-linear.

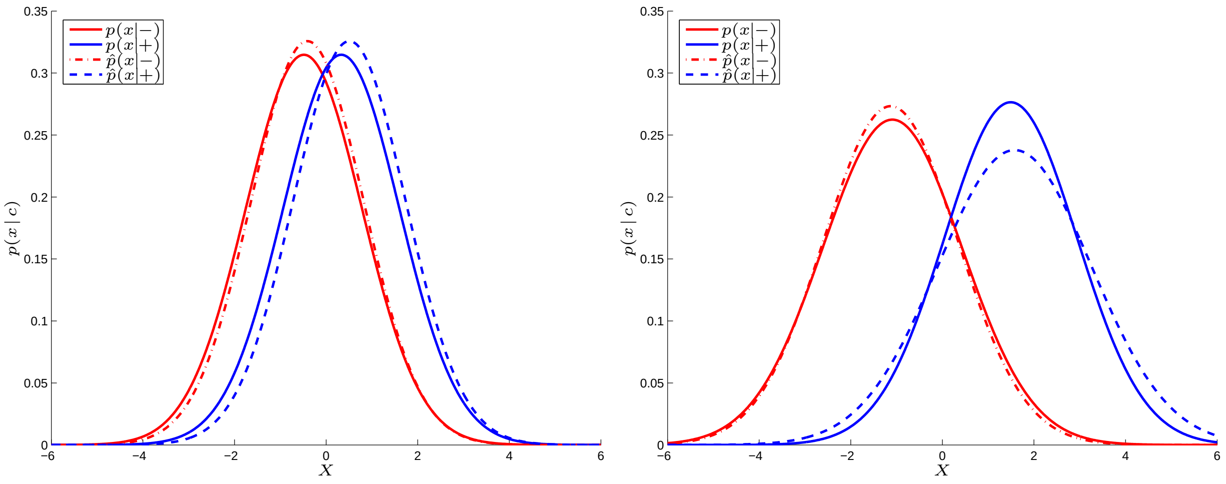


Figure 4.5: The class-conditional distribution of feature values for two synthetic examples and their estimated forms using 100 training examples. The first (*left*) example constrains the class-conditional variances to be equal and uses LDA to train the model. The second (*right*) example has class-specific variances and uses QDA to train the model.

$p(x|-) \sim N(-1.0912, 2.3114)$ and $p(x|+) \sim N(1.4935, 2.0829)$. Using QDA the parameters are estimated to be $\hat{P}(+) = 0.5000$, $\hat{p}(x|-) \sim N(-1.1280, 2.1314)$, and $\hat{p}(x|+) \sim N(1.5791, 2.8136)$ (see Figure 4.5(b)). The true and estimated posterior and log-odds are given in Figure 4.8. Using a single global correction, we find that $a = 1.0430$ and $b = -0.2412$. In Figure 4.10(b), we see that global correction is not in general sufficient when either the true or estimated model is non-linear.[7] In contrast, the locally linear model performs quite well.

In practical terms, what does this mean for us since we never have the true log-odds? Quite clearly, the direct generalization of this is logistic regression which calculates the weights directly. Therefore, it does not require we specify intermediate distributions to calculate expectations needed for the parameters. Similarly other linear combination models have similar interpretations of the weights even when they do not explicitly introduce distributional assumptions. Likewise locality can be introduced into these models by introducing approximations to the weight factors as part of the input to the combination model — depending on the form of the combination model these approximations could then either be used as non-constant bias correction factors, as coefficients, or implicitly by changing the combination function based on the values they take.

[7]The globally corrected model has corrected the high-density area of examples rather than the low-density edges.

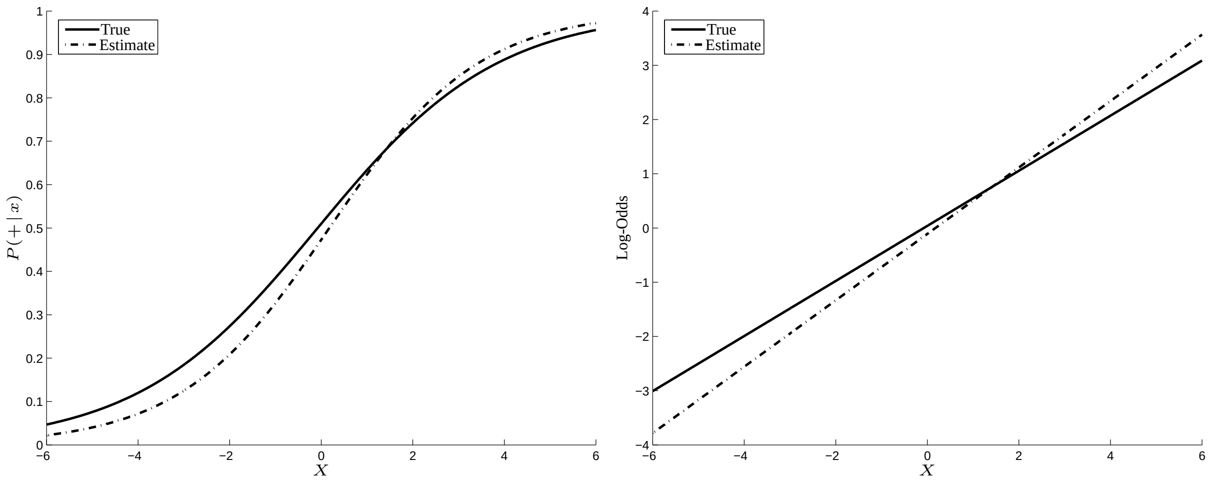


Figure 4.6: The posterior (*left*) and log-odds (*right*) for the example constrained to equal class-conditional variance.

In cases where we would like to work with generative models though, simplifications are sometimes possible. For example Kahn [Kah04] works with a generative model where $p(\hat{\boldsymbol{\lambda}} \mid c)$ is assumed to be Gaussian with equal class-conditional covariance. The resulting combination is a linear combination of the log-odds. The benefit of the model is a clean form of dealing with classifier interactions, but as we have pointed out in several places, the empirical behavior demonstrated by even a single base classifier is typically piecewise linear and not globally.

More importantly, these examples have helped highlight the important quantities and what their roles would be in a locally linear correction. The next section summarizes the properties discussed throughout the chapter and lays the remaining foundation for our approach which will implicitly capture behavior by creating combination rules that use approximations to these quantities.

4.5 Local Reliability, Variance, and Dependence

This section summarizes and collects the observations made in this chapter. Throughout the chapter, we offered arguments about the importance of various quantities and why they could also be characterized in terms of *locality*. *Locality* is roughly defined as nearness as a function of the example. One simple definition would be to characterize input nearness in terms of having a similar base classifier prediction value. Although it is clear from

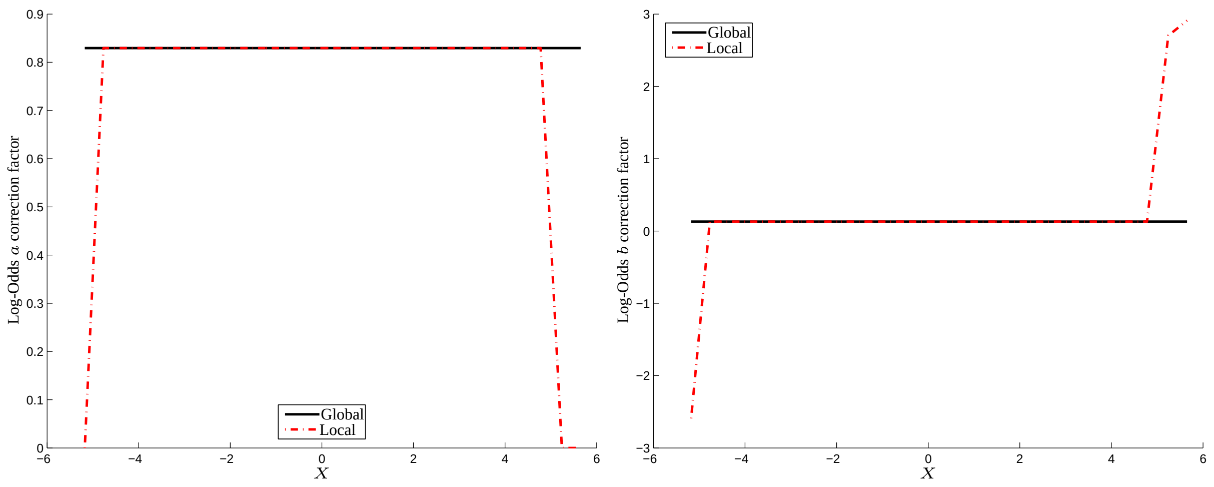


Figure 4.7: The coefficient a (*left*) and additive correction term b to perform linear correction estimated globally and locally using hold-out data for the example constrained to equal class-conditional variance. For this case where both the true and estimated log-odds are linear, a single value of a and b is sufficient to perform perfect correction. The local estimation deviates from this at the edges because of data sparsity.

the above that the prediction error variance around a log-odds prediction of -10 may be different than that around a prediction of 0, it is sometimes advantageous to consider other functions of the input features in addition to focusing on the classifier outputs.

A classifier may use data in different parts of the input space differently. Therefore, the reliability, variance, and dependence may vary locally depending on how well sampled the region of the input space is, the number of features that are locally relevant (*i.e.*, basically the complexity of the decision surface locally versus globally), and how the classifiers employ the data. Many combination schemes fail to account for locality and place only global weights on the classifier outputs. This reduces the ultimate expressive power of any combination method.

Any discussion of "local" implies that for each datapoint, $\mathbf{x}_0$, there is some neighborhood, $N(\mathbf{x}_0)$, that defines what is local to that point. In simple cases, such as defining locality only in terms of the classifier estimates, then the neighborhoods may be explicitly definable by binning around estimated log-odds values. In other cases, the neighborhoods are implicit and purely motivational. For the application of these concepts globally, it can just be assumed that the neighborhood covers the entire domain. Therefore, to make all of the above issues concrete, we need only give a mathematical definition for the local version assuming that we have some definition of neighborhood. Since the work in this disserta-

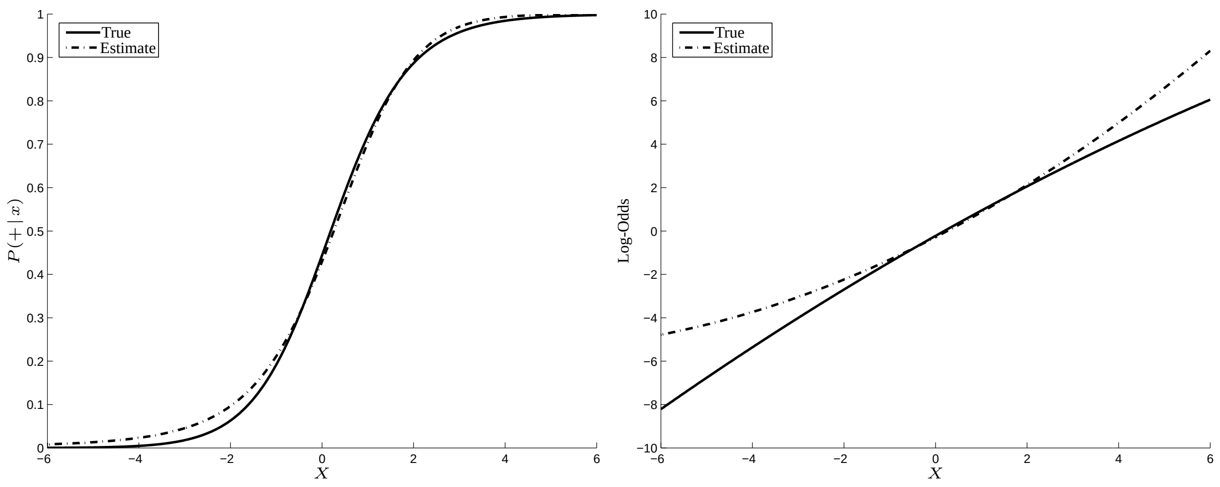


Figure 4.8: The posterior (*left*) and log-odds (*right*) for a 2-class example with class-specifi c variances.

tion uses a binary classification approach, we typically only need to estimate the quantities below for $y = +$ but we give a formulation for the multiclass problem. We now summarize the above issues as the following eight points and give mathematical definitions for those involving locality in terms of both log-odds and probability estimates:

1. Calibration or reliability of classifier outputs;[8]

2. Variance of estimate with available (or total) information — That is the variance of the error in prediction;

3. Dependence of classifier outputs;

4. Local reliability of classifier outputs;
   The average of predictions from a reliable classifier would equal the average actual value (in all neighborhoods):
   $\forall_y E_{p(\mathbf{x}|\mathbf{x}\in N(\mathbf{x}_0))}[P(c(\mathbf{x}) = y \mid \mathbf{x})] = E_{p(\mathbf{x}|\mathbf{x}\in N(\mathbf{x}_0))}[\hat{P}_{\mathcal{C}_i}(c(\mathbf{x}) = y \mid \mathbf{x})]$,
   or $\forall_y E_{p(\mathbf{x}|\mathbf{x}\in N(\mathbf{x}_0))}[\lambda(y)] = E_{p(\mathbf{x}|\mathbf{x}\in N(\mathbf{x}_0))}[\hat{\lambda}(y)]$.
   Therefore a reasonable measure of deviance from reliability within a region would be a measure of deviance from this, such as
   $\sum_y \left[E_{p(\mathbf{x}|\mathbf{x}\in N(\mathbf{x}_0))}[P(c(\mathbf{x}) = y \mid \mathbf{x})] - E_{p(\mathbf{x}|\mathbf{x}\in N(\mathbf{x}_0))}[\hat{P}_{\mathcal{C}_i}(c(\mathbf{x}) = y \mid \mathbf{x})]\right]^2$

[8]In the remainder of this work, we use "reliability" loosely to mean either the items explicitly discussed as reliability here or possibly touching on issues related to any of the types of variance discussed here. When it is necessary to make the distinction explicit, we do so.

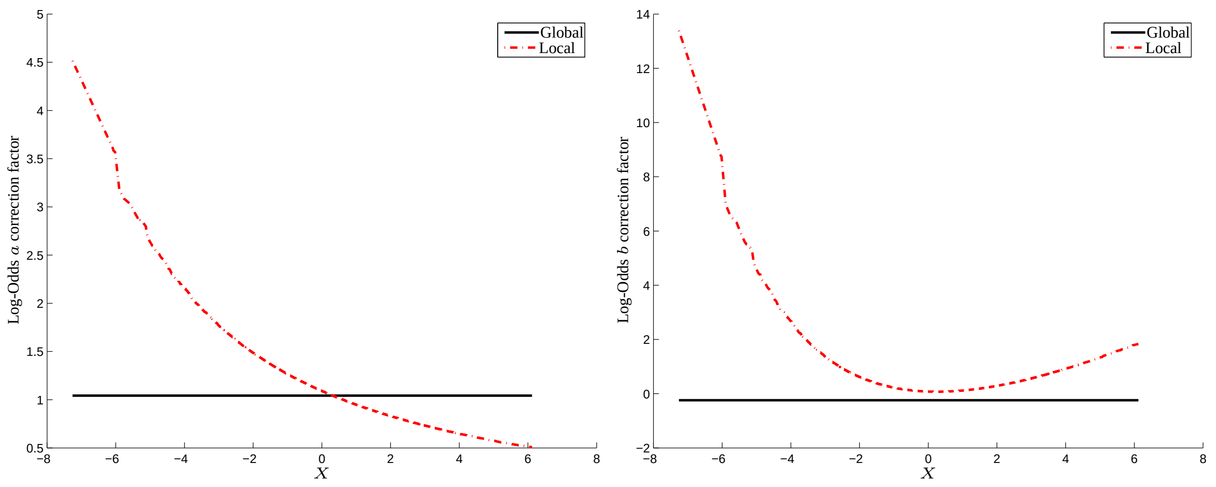


Figure 4.9: The coeffi cient a (*left*) and additive correction term b to perform linear correction estimated globally and locally using hold-out data for the example with class-specifi c variances. For this case, where both the true and estimated log-odds are non-linear, a global value of a and b is not adequate to perform perfect correction.

$= \sum_y \; E^2_{p(\mathbf{x}|\mathbf{x}\in N(\mathbf{x}_0))}[P(c(\mathbf{x}) = y \mid \mathbf{x}) - \hat{P}_{\mathcal{C}_i}(c(\mathbf{x}) = y \mid \mathbf{x})]$,

or $\sum_y \; \left[E_{p(\mathbf{x}|\mathbf{x}\in N(\mathbf{x}_0))}[\lambda(y)] - E_{p(\mathbf{x}|\mathbf{x}\in N(\mathbf{x}_0))}[\hat{\lambda}_i(c)]\right]^2$

$= \sum_y \; E^2_{p(\mathbf{x}|\mathbf{x}\in N(\mathbf{x}_0))}[\lambda(y) - \hat{\lambda}_i(y)]$.

5. Local variance of estimate with total information;

   A classifier that acted with total information would, of course, always predict the posterior. Within a region, the prediction error variance is then:
   $\sum_y \; \text{VAR}_{p(\mathbf{x}|\mathbf{x}\in N(\mathbf{x}_0))}\left[P(c(\mathbf{x}) = y \mid \mathbf{x}) - \hat{P}_{\mathcal{C}_i}(c(\mathbf{x}) = y \mid \mathbf{x})\right]$,
   or $\sum_y \; \text{VAR}_{p(\mathbf{x}|\mathbf{x}\in N(\mathbf{x}_0))}\left[\lambda(y) - \hat{\lambda}_i(y)\right]$.

6. Local dependence of classifier outputs;

   The ideal situation would be when the classifiers are independent given the class of the data they are predicting upon (as in Figure 4.3 above). If this were true, then the ideal would maintain that the joint distribution over the n classifier predictions (denoted as $\hat{y}_1, \ldots, \hat{y}_n$ here) could be factored into their separate predictions:
   $\forall_y P(\hat{y}_1, \ldots, \hat{y}_n \mid c(\mathbf{x}) = y, \mathbf{x} \in N(\mathbf{x_0})) = \prod_{\hat{y}_i} P(\hat{y}_i \mid c(\mathbf{x}) = y, \mathbf{x} \in N(\mathbf{x_0}))$.
   Therefore, a reasonable measure of deviance from independence would be the KL-divergence of the left distribution from the right distribution.

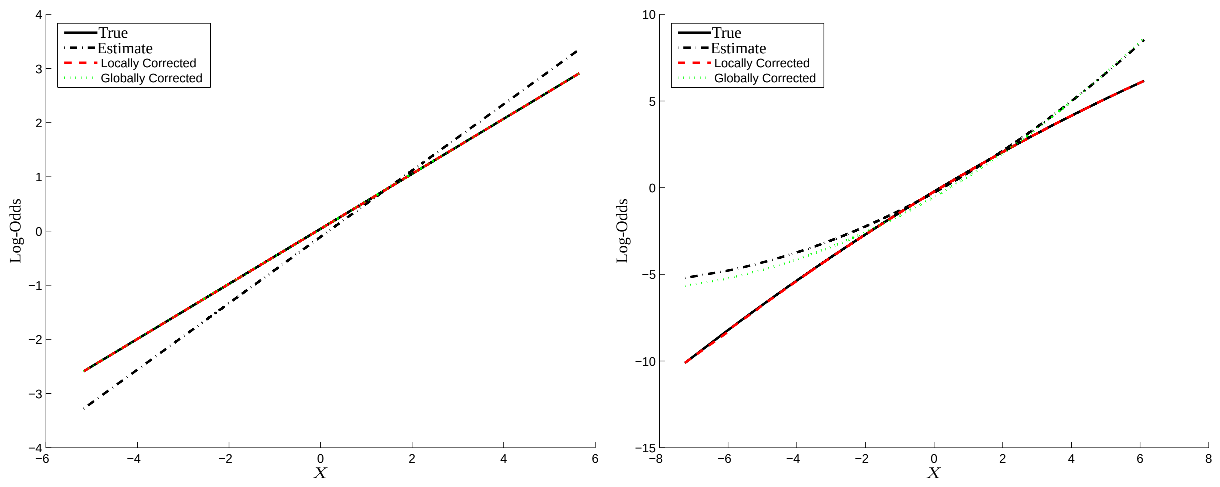


Figure 4.10: The locally linear and global corrections of the log-odds for the equal class-conditional variance example (*left*) and the non-equal example (*right*). As shown on the left, when both the true and estimated model are linear, global weights suffi ce to perform perfect correction. However, when either the true or estimated models are not linear, a locally linear model has the potential to perform far better correction, as shown on the right.

7. Noise sensitivity of a method;

   This is just a measure of variation with the region for a specified model (*i.e.*, deviance of the estimates for other datapoints belonging to the neighborhood from the mean estimate of the neighborhood). This term is:
   $\sum_y \ \mathrm{VAR}_{p(\mathbf{x}|\mathbf{x}\in N(\mathbf{x}_0))}\left[\hat{P}_{\mathcal{C}_i}(c(\mathbf{x}) = y \mid \mathbf{x})\right]$,
   or $\sum_y \ \mathrm{VAR}_{p(\mathbf{x}|\mathbf{x}\in N(\mathbf{x}_0))}\left[\hat{\lambda}_i(y|\mathbf{x})\right]$. It can be expedient though to consider a related term which is deviation from the query point, $\mathbf{x}_0$, instead of the mean prediction. This would be: $\sum_y \ \mathrm{VAR}_{p(\mathbf{x}|\mathbf{x}\in N(\mathbf{x}_0))}\left[\hat{P}_{\mathcal{C}_i}(c(\mathbf{x}_0) = y \mid \mathbf{x}_0) - \hat{P}_{\mathcal{C}_i}(c(\mathbf{x}) = y \mid \mathbf{x})\right]$,
   or $\sum_y \ \mathrm{VAR}_{p(\mathbf{x}|\mathbf{x}\in N(\mathbf{x}_0))}\left[\hat{\lambda}_i(y|\mathbf{x}_0) - \hat{\lambda}_i(y|\mathbf{x})\right]$.

8. Covariance of a method with complete information;

   Given the connection between error prediction variance, noise sensitivity, and variance via $\mathrm{VAR}[X - Y] = \mathrm{VAR}[X] + \mathrm{VAR}[Y] - 2\,\mathrm{COV}[X, Y]$, it is unclear if covariance need be separately estimated. However, we list it for completeness.
   $\sum_y \ \mathrm{COV}_{p(\mathbf{x}|\mathbf{x}\in N(\mathbf{x}_0))}\left[\hat{P}_{\mathcal{C}_i}(c(\mathbf{x}) = y \mid \mathbf{x}), P_{\mathcal{C}_i}(c(\mathbf{x}) = y \mid \mathbf{x})\right]$,
   or $\sum_y \ \mathrm{COV}_{p(\mathbf{x}|\mathbf{x}\in N(\mathbf{x}_0))}\left[\hat{\lambda}_i, \lambda_i\right]$.

Given these quantities of interest, Chapter 5 motivates and defines variables that are specific approximations to these quantities or variables intuitively tied to the neighborhood around a prediction point for the domain. These are then used in methods which can either directly use the approximated variables or use the neighborhood characterizations to define weights and combination functions dependent on the neighborhood.

Chapter 5

Reliability Indicators

This chapter describes the reliability indicators in detail. There are a number of reliability indicators that arise out of the internal workings of the models themselves. Since these variables play a more central role, we devote a significant portion of the chapter to their motivation and formulation. The remainder of the variables focus primarily on the difference between the original representation of the document and the represention after feature selection. These latter variables are presented with a brief motivation and description at the end of the chapter.

The reader should note that identifying and defining variables tied to the reliability of classifiers is both challenging and an open research problem. Even though we have made considerable progress in this arena, it remains an attractive area of future research.

5.1 Model-Specific Reliability Indicators

This section motivates a series of indicators based on the inner workings of each classification model. The variables all share a commonality in that they are related to the shift in the model's output relative to a slight shift in the input. Additionally, the computational complexity of producing each statistic relative to producing a single test prediction is analyzed.

5.1.1 Variables Based on the Unigram Classifier (Multinomial naïve Bayes)

For a two-class problem of discriminating class c from $\neg c$, the log-odds of the unigram classifier can be written as:

$$\log \frac{\hat{P}(c \mid d)}{\hat{P}(\neg c \mid d)} = \log \frac{\hat{P}(c)}{\hat{P}(\neg c)} + \sum_{w \in d} \left[\#(w,d) \log \frac{\hat{P}(w \mid c)}{\hat{P}(w \mid \neg c)} \right] \tag{5.1}$$

where $\#(w,d)$ denotes the number of times word w occurs in document d. Therefore, each occurrence of a word w contributes $\log \frac{\hat{P}(w|c)}{\hat{P}(w|\neg c)} = \log \hat{P}(w \mid c) - \log \hat{P}(w \mid \neg c)$ to the overall classification. Furthermore, if this quantity is positive, the word occurrence moves the decision toward the positive class c, and if this quantity is negative, the word occurrence moves the decision toward the negative class $\neg c$. In light of this, it seems natural to consider possible interpretations of functions of the log-likelihood ratio with respect to w, $\log \frac{\hat{P}(w|c)}{\hat{P}(w|\neg c)}$, and the log-likelihood with respect to w, $\log \hat{P}(w \mid C)$, as indicator variables.

First, consider the mean per-word log-likelihood ratio: $\frac{1}{|d|} \sum_{w \in d} \#(w,d) \log \frac{\hat{P}(w|c)}{\hat{P}(w|\neg c)}$. Why we may want to concern ourselves with this quantity can be motivated from several viewpoints. For example, consider how the output of the unigram classifier would change if we changed the document by (uniformly) randomly choosing a word in the document and eliminating it. More formally, let w_i refer to the ith unique vocabulary word that occurs in the bag-of-words representation of the document d and let d^{-i} denote the document obtained by removing a single occurrence of w_i from d. Let Δ denote a distribution over these altered documents such that the probability of generating d^{-i} is $\frac{\#(w,d)}{|d|}$ where $|d|$ denotes the total number of words in the document. Let $\hat{\lambda}_U(d)$ denote the log odds of the unigram model. Then we wish to know how much it will change our estimate if we remove a single word, or $E_\Delta[\hat{\lambda}_U(d) - \hat{\lambda}_U(d^{-i})]$ where $d^{-i} \sim \Delta$.

This expectation reduces to the average per-word log-likelihood ratio:

$$E_\Delta[\hat{\lambda}_U(d) - \hat{\lambda}_U(d^{-i})] = \tag{5.2}$$

$$= \sum_{d^{-i}} \left[\frac{\#(w_i,d)}{|d|} \left(\hat{\lambda}_U(d) - \hat{\lambda}_U(d^{-i}) \right) \right] \tag{5.3}$$

$$= \sum_{d^{-i}} \left[\frac{\#(w_i,d)}{|d|} \log \frac{\hat{P}(w_i \mid c)}{\hat{P}(w_i \mid \neg c)} \right] \tag{5.4}$$

$$= \frac{1}{|d|} \sum_{d^{-i}} \left[\#(w_i, d) \log \frac{\hat{P}(w_i \mid c)}{\hat{P}(w_i \mid \neg c)} \right] \tag{5.5}$$

$$= \frac{1}{|d|} \sum_{w \in d} \left[\#(w, d) \log \frac{\hat{P}(w \mid c)}{\hat{P}(w \mid \neg c)} \right]. \tag{5.6}$$

By inspection of the formula, it is also obvious that this quantity is tied to the prediction of the unigram classifier itself being equal to the prediction minus the log priors and divided by the document length. At the same time, it is a measure of the change around the prediction with a slight shift.

Continuing along the same vein, it is natural to consider the variance of the statistic: $\mathrm{VAR}_\Delta[\hat{\lambda}_U(d) - \hat{\lambda}_U(d^{-i})] = E_\Delta[\left(\hat{\lambda}_U(d) - \hat{\lambda}_U(d^{-i})\right)^2] - E_\Delta^2[\hat{\lambda}_U(d) - \hat{\lambda}_U(d^{-i})]$. Following a derivation similar to the above we can show that:

$$E_\Delta[\left(\hat{\lambda}_U(d) - \hat{\lambda}_U(d^{-i})\right)^2] = \frac{1}{|d|} \sum_{w \in d} \#(w, d) \left[\log \frac{\hat{P}(w \mid c)}{\hat{P}(w \mid \neg c)} \right]^2. \tag{5.7}$$

From these two terms, we can then compute the variance. Note that the variance will be near zero when all the words are pointing with approximately the same strength at the same class. As the variance increases, then the disagreement among the individual words is higher. Thus, regardless of the conditional independence assumption the model makes, we would expect that the predictions will be more reliable when the variance is low than when it is high.[1]

The previous two statistics were motivated in terms of the decision boundary of the unigram classifier, but it seems prudent to also consider statistics of the primary components of the classifier. That is we can also consider the mean per-word log-likelihood, *i.e.* $\frac{1}{|d|} \sum_{w \in d} \left[\#(w, d) \log \hat{P}(w|c) \right]$ and $\frac{1}{|d|} \sum_{w \in d} \left[\#(w, d) \log \hat{P}(w|\neg c) \right]$. Similarly, we can consider the variance of the per-word log-likelihood. Now, these statistics will measure the average strength for each class and the spread of how strongly each word is voting.[2]

Finally, note that computing each of these statistics requires the same run-time as making the original prediction, $O(|d|)$. Therefore, computing these statistics does not significantly impact our computational burden.

[1]To apply this to polychotomous learning problems, we believe it would be most beneficial to have one value per class and consider statistics based on either $\log \frac{\hat{P}(w|c)}{1 - \hat{P}(w|c)}$ or $\log \frac{\hat{P}(w|c)}{\max_{c' \neq c} \hat{P}(w|c')}$. Alternatively one could consider all class pairs.

[2]An interesting future direction is to consider similar statistics that do not weigh each word equally but instead use the model's estimate of the word's probability, $\hat{P}(w)$.

5.1.2 Variables Based on the naïve Bayes Classifi er (Multivariate Bernoulli naïve Bayes)

The variables motivated by the naïve Bayes Classifier are directly analogous to those discussed above for the unigram classifier — as would be expected since the models are highly related. However, the difference in the event space employed by each model dictates subtle changes that we need to address.

First, note the difference between these two models' event spaces. The unigram probability model can be thought of generatively as drawing a class according to a class prior, then drawing a document length, and finally drawing the words according to the class-conditional word distribution. In contrast, the naïve Bayes model draws a class according to a class prior and then for each word in the vocabulary draws whether or not the word occurs in the document according to a class-conditional distribution. Thus, the multivariate Bernoulli naïve Bayes classifier models a binary (Bernoulli) variable of whether or not a word occurs conditioned on the class for every word in the vocabulary. Let $V - d$ denote the set of features that do not take a value of "present" or 1 in d. Then, for a two-class problem of discriminating class c from $\neg c$, the log-odds of the naïve Bayes classifier can be written as:

$$\begin{aligned}
\log \frac{\hat{P}(c \mid d)}{\hat{P}(\neg c \mid d)} &= \log \frac{\hat{P}(c)}{\hat{P}(\neg c)} && (5.8)\\
&\quad + \sum_{w \in d} \left[\log \frac{\hat{P}(w=1 \mid c)}{\hat{P}(w=1 \mid \neg c)} \right] + \sum_{w \in V-d} \left[\log \frac{\hat{P}(w=0 \mid c)}{\hat{P}(w=0 \mid \neg c)} \right] \\
&= \log \frac{\hat{P}(c)}{\hat{P}(\neg c)} + \sum_{w \in V} \left[\log \frac{\hat{P}(w=0 \mid c)}{\hat{P}(w=0 \mid \neg c)} \right]. && (5.9)\\
&\quad - \sum_{w \in d} \left[\log \frac{\hat{P}(w=0 \mid c)}{\hat{P}(w=0 \mid \neg c)} \right] + \sum_{w \in d} \left[\log \frac{\hat{P}(w=1 \mid c)}{\hat{P}(w=1 \mid \neg c)} \right]
\end{aligned}$$

The log odds are often formulated as given in Line 5.9 for efficiency. The priors together with the first summation in the line is often termed the log odds of the null document since it is the log odds of a document that contains no word occurrences. This term is costly to compute since it includes every word in the (typically large) vocabulary. Let $|d|$ denote the number of "present" words in the document. Then, the costly summation must only be computed once during training, and at test time, each document's prediction can be made in $O(|d|)$ time instead of $O(|V|)$ by subtracting off the "absent" contributions of the words that are present and adding in their "present" contributions.

Now, instead of restricting our set of slight changes to the input to only the words present in the document, we will continue in the style of the model and consider how its output would change if we altered a single feature value for all possible values in the domain. Now, we will let d^{-i} denote the document identical to d except that feature i has been "flipped" and Δ is now the distribution over documents such that $P_\Delta(d^{-i}) = \frac{1}{|V|}$ for $i = 1 \ldots |V|$. Let $\hat{\lambda}_B(d)$ be the estimate of the multivariate Bernoulli naïve Bayes model for document d. We desire to determine the shift in the model output caused by changing a bit. $E\Delta[\hat{\lambda}_B(d) - \hat{\lambda}_B(d^{-i})]$ where $d^{-i} \sim \Delta$.

Let $w_i(d)$ denote the presence/absence value word i takes in document d, and let $w'_i(d)$ denote its complement. It is quite easy to see that this expectation reduces as follows:

$$\begin{aligned}
E_\Delta[\hat{\lambda}_B(d) - \hat{\lambda}_B(d^{-i})] &= && (5.10)\\
&= \frac{1}{|V|}\sum_{d^{-i}}\left[\left(\hat{\lambda}_B(d) - \hat{\lambda}_B(d^{-i})\right)\right] && (5.11)\\
&= \frac{1}{|V|}\sum_{d^{-i}}\log\left[\frac{\hat{P}(w_i(d)\mid c)}{\hat{P}(w_i(d)\mid\neg c)}\frac{\hat{P}(w'_i(d)\mid\neg c)}{\hat{P}(w'_i(d)\mid c)}\right] && (5.12)\\
&= \frac{1}{|V|}\sum_{w\in d}\log\left[\frac{\hat{P}(w=1\mid c)}{\hat{P}(w=1\mid\neg c)}\frac{\hat{P}(w=0\mid\neg c)}{\hat{P}(w=0\mid c)}\right] && (5.13)\\
&\quad+\frac{1}{|V|}\sum_{w\in V-d}\log\left[\frac{\hat{P}(w=0\mid c)}{\hat{P}(w=0\mid\neg c)}\frac{\hat{P}(w=1\mid\neg c)}{\hat{P}(w=1\mid c)}\right]\\
&= \frac{1}{|V|}\sum_{w\in V}\log\left[\frac{\hat{P}(w=0\mid c)}{\hat{P}(w=0\mid\neg c)}\frac{\hat{P}(w=1\mid\neg c)}{\hat{P}(w=1\mid c)}\right] && (5.14)\\
&\quad+\frac{1}{|V|}\sum_{w\in d}\log\left[\frac{\hat{P}(w=1\mid c)}{\hat{P}(w=1\mid\neg c)}\frac{\hat{P}(w=0\mid\neg c)}{\hat{P}(w=0\mid c)}\right]\\
&\quad-\frac{1}{|V|}\sum_{w\in d}\log\left[\frac{\hat{P}(w=0\mid c)}{\hat{P}(w=0\mid\neg c)}\frac{\hat{P}(w=1\mid\neg c)}{\hat{P}(w=1\mid c)}\right].\\
&= \frac{1}{|V|}\sum_{w\in V}\log\left[\frac{\hat{P}(w=0\mid c)}{\hat{P}(w=0\mid\neg c)}\frac{\hat{P}(w=1\mid\neg c)}{\hat{P}(w=1\mid c)}\right] && (5.15)\\
&\quad+\frac{2}{|V|}\sum_{w\in d}\log\left[\frac{\hat{P}(w=1\mid c)}{\hat{P}(w=1\mid\neg c)}\frac{\hat{P}(w=0\mid\neg c)}{\hat{P}(w=0\mid c)}\right].
\end{aligned}$$

Again, the formulation given in Line 5.14 allows this statistic to be efficiently computed in $O(|d|)$ time during prediction by computing the first summation at training time. In the unigram model, each word present in the document contributed according to the log-likelihood with respect to it and the corresponding statistic was the mean over those terms. Likewise, in the multivariate Bernoulli model, each word in the vocabulary contributes according to the log odds ratio of the likelihoods, and the resulting statistic is the mean over the per feature contributions.

The derivation for the variance, $\mathrm{VAR}_\Delta[\hat{\lambda}_B(d)-\hat{\lambda}_B(d^{-i})] = E_\Delta[\Big(\hat{\lambda}_B(d) - \hat{\lambda}_B(d^{-i})\Big)^2] - E^2_\Delta[\hat{\lambda}_B(d) - \hat{\lambda}_B(d^{-i})]$,
follows similar lines. To compute the first term, following the pattern above, we can easily derive:

$$\begin{aligned}
E_\Delta[\Big(\hat{\lambda}_B(d) - \hat{\lambda}_B(d^{-i})\Big)^2] &= \frac{1}{|V|}\sum_{w\in V}\left[\log\left(\frac{\hat{P}(w=0\mid c)}{\hat{P}(w=0\mid \neg c)}\frac{\hat{P}(w=1\mid \neg c)}{\hat{P}(w=1\mid c)}\right)\right]^2 && (5.16)\\
&\quad+\frac{1}{|V|}\sum_{w\in d}\left[\log\left(\frac{\hat{P}(w=1\mid c)}{\hat{P}(w=1\mid \neg c)}\frac{\hat{P}(w=0\mid \neg c)}{\hat{P}(w=0\mid c)}\right)\right]^2\\
&\quad-\frac{1}{|V|}\sum_{w\in d}\left[\log\left(\frac{\hat{P}(w=0\mid c)}{\hat{P}(w=0\mid \neg c)}\frac{\hat{P}(w=1\mid \neg c)}{\hat{P}(w=1\mid c)}\right)\right]^2.\\
&= \frac{1}{|V|}\sum_{w\in V}\left[\log\left(\frac{\hat{P}(w=0\mid c)}{\hat{P}(w=0\mid \neg c)}\frac{\hat{P}(w=1\mid \neg c)}{\hat{P}(w=1\mid c)}\right)\right]^2 && (5.17)\\
&\quad+\frac{1}{|V|}\sum_{w\in d}\left[-\log\left(\frac{\hat{P}(w=0\mid c)}{\hat{P}(w=0\mid \neg c)}\frac{\hat{P}(w=1\mid \neg c)}{\hat{P}(w=1\mid c)}\right)\right]^2\\
&\quad-\frac{1}{|V|}\sum_{w\in d}\left[\log\left(\frac{\hat{P}(w=0\mid c)}{\hat{P}(w=0\mid \neg c)}\frac{\hat{P}(w=1\mid \neg c)}{\hat{P}(w=1\mid c)}\right)\right]^2.\\
&= \frac{1}{|V|}\sum_{w\in V}\left[\log\left(\frac{\hat{P}(w=0\mid c)}{\hat{P}(w=0\mid \neg c)}\frac{\hat{P}(w=1\mid \neg c)}{\hat{P}(w=1\mid c)}\right)\right]^2 && (5.18)\\
&\quad+\frac{1}{|V|}\sum_{w\in d}\left[\log\left(\frac{\hat{P}(w=0\mid c)}{\hat{P}(w=0\mid \neg c)}\frac{\hat{P}(w=1\mid \neg c)}{\hat{P}(w=1\mid c)}\right)\right]^2\\
&\quad-\frac{1}{|V|}\sum_{w\in d}\left[\log\left(\frac{\hat{P}(w=0\mid c)}{\hat{P}(w=0\mid \neg c)}\frac{\hat{P}(w=1\mid \neg c)}{\hat{P}(w=1\mid c)}\right)\right]^2.\\
&= \frac{1}{|V|}\sum_{w\in V}\left[\log\left(\frac{\hat{P}(w=0\mid c)}{\hat{P}(w=0\mid \neg c)}\frac{\hat{P}(w=1\mid \neg c)}{\hat{P}(w=1\mid c)}\right)\right]^2 && (5.19)
\end{aligned}$$

This term need only be computed during training and is then used in combination with the previous statistic to compute the variance in $O(|d|)$ time. Finally, as for the unigram classifier, we also consider the mean and variance of the class-strength of each feature. That is the mean and variance of $\log\frac{\hat{P}(w(d)|c)}{\hat{P}(w'(d)|c)}$ and $\log\frac{\hat{P}(w(d)|\neg c)}{\hat{P}(w'(d)|\neg c)}$.

5.1.3 Variables Based on the kNN Classifier

The variables based on the kNN classifier stem from two basic motivations: (1) the long-standing intuition that the proximity of the neighbors has an impact on the classifier's reliability; (2) measuring sensitivity to a change in the input in a way that is connected to the internal mechanisms of the classifier.

The kNN classifier is one of the oldest classification algorithms and many variants have been employed [CH67, Yan99]. In its most basic version, the nearest neighbor classifier, a similarity measure is specified by the user and a test or query point is classified by finding the most similar training point and using the class label of that point to predict for the test point. The natural generalization is to find the k most similar or closest neighbors in the training set and take the majority vote of their class labels. A further generalization that has been shown to be quite competitive for text classification is to use some form of distance-weighted voting [Yan99] in order to allow closer neighbors to carry more weight.

While the following variables can be derived in any kNN framework, it is necessary to understand the role of the reliability indicators within our particular kNN classifier implementation. In our implementation, k is set to be $2(\lceil \log_2 N \rceil) + 1$ where N is the number of training instances. This rule for choosing k is theoretically motivated by results which show such a rule converges to the optimal classifier as the number of training points increases [DGL96]. In practice, we have also found it to be a computational convenience that frequently leads to comparable results with numerically optimizing k via a cross-validation procedure. As is quite common in text classification, we use the cosine similarity, $\cos(\vec{x}_1, \vec{x}_2)$, with higher values indicating greater similarity (closer neighbors). For those less familiar with the cosine of two vectors[3], it is equivalent to the inner product of two vectors if all of the vectors have been normalized to the unit N-sphere. The score used for a class y is:

$$s_{\vec{x}}(y) = \sum_{\vec{n} \in kNN(\vec{x}) | c(\vec{n}) = y} \cos(\vec{x}, \vec{n}) \quad - \sum_{\vec{n} \in kNN(\vec{x}) | c(\vec{n}) \neq y} \cos(\vec{x}, \vec{n}) \tag{5.20}$$

The class with the highest score is predicted as the class of the example. Since we apply the classifiers as binary classifiers, there are only two classes and $s_{\vec{x}}(-)$ is the additive inverse of $s_{\vec{x}}(+)$. Note that we can treat $s_{\vec{x}}(+)$ as an approximation for the log-odds of an example, $\hat{\lambda}_{kNN}(\vec{x})$; like log-odds, by default we predict the sign of the score function as the sign of the classifier and increasing magnitude indicates increased confidence. Finally, we

[3]We also use a *tfidf* weighting where the tf factor is the standard term frequency and the idf factor is $\log_2(N/df)$.

apply a threshold method referred to as s-cut in [Yan99] where instead of using the default threshold of 0, we learn a threshold by cross-validation over the training set.[4]

Some forms of distance-weighted voting include a decay factor on the distance between the query point and the neighbor such that for some values of the decay factor, the nearest neighbor will dominate the overall vote. While we could choose other forms of distance-weighted voting, they all attempt to balance issues like how much the absolute distance of the nearest neighbor influences the final prediction versus how much the relative difference in neighbors determines the prediction. Instead, we take the approach of choosing the common score function given in Eq. 5.20 and taking other factors into account through our definition and use of reliability indicators.

The first such issue is that not all areas of the input space are well-sampled. The convergence theorem cited above and other guarantees regarding kNN's performance [DHS01] essentially rely on the fact that, as long as the test distribution is the same as the training distribution, we are likely to have training examples near where testing examples are likely to occur. Therefore, if the decision function is smooth, we will get good estimates quickly. If it is not smooth, convergence still occurs but is slower. The simplest of the reliability indicators try to detect when an area is not well-sampled by introducing functions of the distances of neighbors. *NeighborhoodRadius* is simply the distance from the query point to the farthest neighbor included in the neighborhood. Thus, we would expect a small radius when the area is well-sampled and large when it is not. The next two reliability indicators, *MeanNeighborDistance* and *SigmaNeighborDistance*, take into account the distribution of neighbors by computing the mean and standard deviation of the distance from the query point to a neighbor. Thus, a low mean could indicate that the neighborhood is consistently well-sampled even though the farthest neighbor might be distant. Likewise, a high variance could indicate that the sampling quality is not as consistent since the neighbors are at very different distances.

Next, we consider how we might detect when the decision function is not smooth in a neighborhood. If the decision function is not smooth, then we would also expect that as we move out from the query point in different directions, the overall prevalence of a class in the training set would also change. Thus, we can also introduce a sensitivity based variable that computes how much the output would change as we change the query point slightly. Since we are concerned with how the decision function would change as we move toward our neighbor, then it is natural to consider a set of shifts defined in terms of the neighbors. In particular, let $\mathbf{d}_i$ denote the document that has been shifted by a factor α toward the ith

[4]Similar to the recalibration techniques discussed in Section 3.2, this essentially acts as a recalibration device where the estimated log-odds are shifted by the learned threshold.

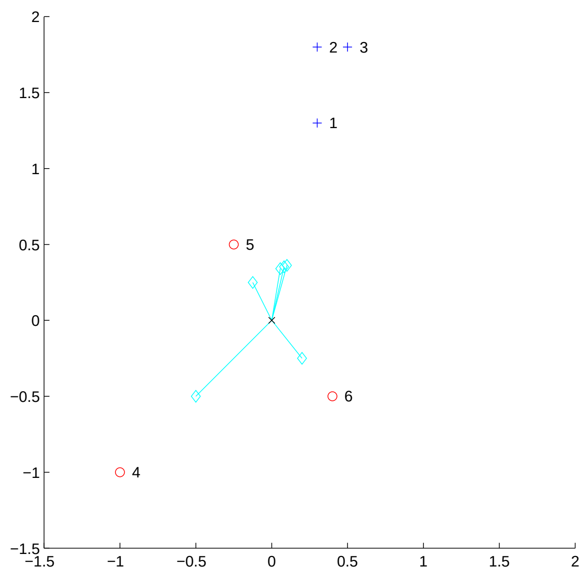


Figure 5.1: An example in Euclidean space of the kNN shifted instances produced for a query instance x using the other points shown as its neighborhood. The shifts are illustrated using cyan lines from the original instance. The nearness of neighbor 5 prevents the shifts toward neighbors $1-3$ from being larger. In contrast, the shift toward neighbor 4 is fully half the distance since it is away from the other neighbors. Since a shift toward each neighbor is weighted equally, the net effect is that a shift toward a dense area is more likely.

neighbor, *i.e.* $\mathbf{d}_i = \mathbf{d} + \alpha(\mathbf{n}_i - \mathbf{d})$. To determine α we choose the largest α such that the closest neighbor to the new point is the original document. Clearly, α will not exceed 0.5, and we can find it very efficiently using a simple bisection algorithm. Let Δ be the uniform distribution over the shifted points. That is, if $\mathbf{d}_i \sim \Delta$, then $P_\Delta(\mathbf{d}_i) = \frac{1}{k}$.[5] Likewise, Δ' will indicate the uniform distribution over all k shifted points and the original query. Figure 5.1 illustrates these shifts graphically for an example in Euclidean space and Table 5.1 gives the corresponding values of α for the shift toward each neighbor.

If the neighborhood is not smooth, then we would also expect to see that the class prediction is not constant across the shifts. To measure this, we compute the average prediction $E_{\Delta'}[s_{\hat{\mathbf{d}}}(+) \geq 0]$ where $\hat{\mathbf{d}} \sim \Delta'$ and denote it *kNNShiftMeanPred*. However, in addition to computing how rapidly the prediction is changing, we can also measure how much the confidence score differs from the prediction for the query. Therefore, we also compute $E_\Delta[s_{\mathbf{d}_i}(+) - s_{\mathbf{d}}(+)]$ and $\text{Var}_\Delta^{1/2}[s_{\mathbf{d}_i}(+) - s_{\mathbf{d}}(+)]$ and denote them as *kNNShiftMeanConfDiff* and *kNNShiftStdDevConfDiff*.

[5]Notice that Δ implicitly weights shifts toward more common changes in the document since more common change vectors will also have more neighbors on that side.

Shifted Point	α	$\mathbf{d}$	$\mathbf{n}_1$	$\mathbf{n}_2$	$\mathbf{n}_3$	$\mathbf{n}_4$	$\mathbf{n}_5$	$\mathbf{n}_6$
$\mathbf{d}_1$	0.27	0.36	0.97	1.46	1.51	1.73	0.36	0.91
$\mathbf{d}_2$	0.19	0.35	0.99	1.48	1.52	1.71	0.35	0.91
$\mathbf{d}_3$	0.20	0.38	0.96	1.45	1.49	1.75	0.38	0.91
$\mathbf{d}_4$	0.50	0.71	1.97	2.44	2.51	0.71	1.03	0.90
$\mathbf{d}_5$	0.50	0.28	1.13	1.61	1.67	1.53	0.28	0.92
$\mathbf{d}_6$	0.50	0.32	1.55	2.05	2.07	1.42	0.87	0.32

Table 5.1: Various quantities for the example in Euclidean space illustrated in Figure 5.1. α is the amount example $\mathbf{d}$ is shifted toward each neighbor to produce $\mathbf{d}_i$. Each row lists the Euclidean distances between the shifted point $\mathbf{d}_i$ and the original point $\mathbf{d}$ as well as each neighbor $\mathbf{n}_j$. The nearness of neighbor $\mathbf{n}_5$ prevents the shifted instances $\mathbf{d}_1$, $\mathbf{d}_2$, and $\mathbf{d}_3$ from shifting closer to neighbors $\mathbf{n}_1$, $\mathbf{n}_2$, and $\mathbf{n}_3$, respectively. Thus α for these shifted points is less than 0.5.

Analyzing the run-time of the kNN algorithm is rather tricky. The naïve implementation would scan all N training points every time a new query is seen. Let l denote the average number of non-zero features, then on average computing the cosine similarity between two points will be $O(l)$. Thus, if we are classifying M points total, the classification computation cost is $O(M\,[Nl + k\log k])$ for a naïve implementation of kNN.[6] Since we are applying the classifier to text documents, a typical speed-up exploits the fact that each document has very few non-zero features. We build an *inverted table* that indexes the training set by feature — for each feature, storing a list of the training documents that have a non-zero value. When we receive a new document, we sort the non-zero features in the document by their tfidf score and then proceed through each inverted list. After every feature's inverted list, we can bound the theoretically closest possible neighbor that we have not examined yet. When the bound gets tighter than the distance to the farthest point in the neighborhood, we can terminate early. Because we proceed through the features in tfidf order, we tend to find the closest neighbors right away and the majority of time is spent ensuring there are no closer neighbors.[7] Since we are interested in performing exact kNN, our performance gain is less than what it could be. Examining Table 5.2, we see for one small dataset that we have to examine less than half as many points on average with the sparse

[6]The $k\log k$ factors comes from tracking the top neighbors. Technically, this is $u\log k$ where u is the number of (update) times we find a neighbor closer than the worst, but u empirically tends to be polynomial in k.

[7]For the "Sparse *(K)*" entry in Table 5.2, we had to examine the inverted lists of 2.06 features on average to reach all the neighbors, but we had to examine 13.16 features on average to be sure there were no closer neighbors. Thus an approximation algorithm that halts after a preset number of features can be accurate and efficient. Additionally feature selection methods that favor rare features can introduce even more speed-ups.

Method	Avg # Dist Ops to Find Neighbors	Ratio to Baseline	Total Run Time (s)	Ratio to Baseline
Naïve *(k)*	9603	2.39	285.38	4.13
Sparse *(k)*	4016.40	1	69.1	1
Sparse *(2k)*	4698.15	1.17	80.07	1.16
Sparse w/RIVS *(2k)*	4698.15	1.17	196.37	2.84

Table 5.2: Effect on running time of computing the kNN reliability indicators for the Reuters 21578 corpus (9603 training examples, 3299 testing examples, 900 features used). The naï ve algorithm scans all training examples each time. The sparse algorithm uses speed-ups based on sparsity and just performs basic prediction; we show one version using the standard number of neighbors and one using twice that. The fi nal version also computes and writes the reliability indicators —using a neighborhood of $k = 29$ for prediction but $2k$ to compute the reliability indicators. For these comparisons, r-cut with $r = 1$ is used for prediction [Yan99].

algorithm and have a run time that is a quarter of the time used by the naïve approach. The exact speed-up depends on characteristics of the dataset, but achieving at least a factor of 4 seems to be common.

To compute the reliability indicators, it is clear that some of the variables can be computed with essentially no extra cost while performing the prediction including: *NeighborhoodRadius*, *MeanNeighborDistance*, *Mean*$\{$*Class*$\}$*NeighborDistance*, *SigmaNeighborDistance*, and *Sigma*$\{$*Class*$\}$*NeighborDistance*. These simply require updating a corresponding variable's state as each neighbor is found and voted toward the final classification. The primary difficulty comes in computing the variables that involve shifting the query point and classifying the shifted point. If we treated each shifted point as a query, the run-time becomes quite unreasonable. Instead, we take an approximation approach for these variables. Instead of finding the closest k neighbors we find the closest $2k$ neighbors. We still classify the query using its closest k neighbors and created a shifted query for each neighbor, but we find the k closest points to each shifted query among only the $2k$ points. Thus we avoid the cost of finding the actual neighborhood to the shifted point by using an approximation. For each test example, we have to perform an extra $O(lk^2 \log k)$ operations to evaluate the predictions for the shifted point. Table 5.2 shows that finding the larger neighborhood only penalizes us 16% in run time ("Sparse *(K)*" vs. "Sparse *(2K)*"), but that the extra overhead for computing the reliability indicators surrenders half of the speed-up we gained over the naïve algorithm ("Sparse w/RIVS *(2k)*").[8]

[8]Some of this is I/O overhead and could be optimized to be closer to the performance of "Sparse *(2K)* ".

5.1.4 Variables Based on the Decision Tree Classifi er

It is well-known that a standard decision tree corresponds to regions of hyper-rectangles in the $\mathbb{R}^D$ space and that two problems which occur frequently in practice are oversplitting and boundary sensitivity. The first results in using the estimate at a child node when a parent node would have been a better estimate. The second results in grossly misestimating the value to predict for examples that fall near the edges of the hyper-rectangles. Thus when considering how to capture the sensitivity of a decision tree model, we would like to favor shifts in the input to nearby leaves or to branches with similar values to an example.

Following the pattern set forth for the other classifiers, we again consider shifting the input toward other examples. However, in this case, the shift is somewhat more implicit. Assume we are given a document $\mathbf{d}$, and the path of nodes the document follows down the decision tree, $n_0, n_1, ..., n_l$ where n_0 is the root node, n_1 is the child of the root node, and n_l is the leaf node where document d ended up. Let $n_i^{(j)}$ denote the jth left branch in the ith node that was not taken. For a binary tree, there will be exactly one, but for a multi-way tree there can be many. Let $\mathcal{B}$ denote this set of B "local untaken" branches that lie along the path of the document through the decision tree. We will denote by $\mathbf{d}_i$ a document just like $\mathbf{d}$ except that it has been altered to go down branch i of $\mathcal{B}$. Note it would be possible to obtain such a document by changing only one feature. However, to reflect that we are more likely to end up in nearby leaves and examples close to the boundary are more likely to shift, we will use the distance (after normalizing to the unit sphere) between the document $\mathbf{d}$ and the centroid of documents at the node, $\bar{\mathbf{d}}(n_{b_i})$, where n_{b_i} denotes the node reached by taking the untaken branch b_i.

Since each step lower in the decision tree implies the examples have more features in common, then this distance will naturally be lower for nodes that are close together in the tree. Likewise, if an example is near the decision boundary then this will naturally account for it as well. Let ϵ be the minimum distance to an untaken centroid, $\epsilon = \min_{b_i \in \mathcal{B}} \|\bar{\mathbf{d}}(n_{b_i}) - \mathbf{d}\|$. Then we set the probability of drawing such a similar document $\mathbf{d}_i \sim \Delta'$ to be $P_{\Delta'}(\mathbf{d}_i) \propto \exp(-\|\bar{\mathbf{d}}(n_{b_i}) - \mathbf{d}\| + \epsilon)$ such that $\sum_{b_i \in B} P_{\Delta'}(\mathbf{d}_i) = 1$. Once we have taken a branch we will assume that the probabilities of ending up in some particular terminal leaf descendant of i, t_j^i, is determined by the relative frequencies in the training set, $\frac{c(t_j^i)}{c(n_{b_i})}$, where $c(\cdot)$ gives the training count at a node. Finally, every terminal leaf except the one that classified $\mathbf{d}$ in the tree is a descendant of some untaken branch. Let $\hat{\lambda}_D(t)$ denote the decision tree's estimate of the log-odds of the positive class at a leaf t. Let Δ denote the distribution such that the $P_\Delta(t_j^i) = \frac{c(t_j^i)}{c(n_{b_i})} P_{\Delta'}(\mathbf{d}_i)$. Then, we compute the mean change in output $E_\Delta[\hat{\lambda}_D(t_j^i) - \hat{\lambda}_D(n_l)]$ and the standard deviation $\mathrm{Var}_\Delta^{1/2}[\hat{\lambda}_D(t_j^i) - \hat{\lambda}_D(n_l)]$. We refer to these variables as *DTreeShiftMeanConfDiff* and *DTreeShiftStdDevConfDiff* respectively.

As far as the running time is concerned, we can precompute the centroids to store at all intermediate leaves of the tree during training. Likewise, since the mass below an untaken branch is proportioned according to the relative frequencies, we can store the intermediate sums and multiply them by the untaken branch probabilities determined at prediction time. Thus we only need to make three passes along the prediction path: (1) find the closest untaken branch centroid; (2) compute the probability normalization factor; (3) sum the intermediate sums stored at the highest untaken branch nodes. Therefore, computing these variables has an expected running time of $E[ml]$, where m is the average path length to the leaf of a tree and l is the average length of a document.

5.1.5 Variables Based on the SVM Classifi er

Similar to the other classifiers, the variables motivated by the SVM classifier try to capture the sensitivity of the model to changes in the input and intuitive notions of when an example falls into an area where the SVM solution is less reliable.

Overview of SVMs

There are two ideas key to the SVM classifier: the kernel and maximizing the margin. The easiest way to think of a kernel is as a special type of similarity function between two examples. The special stipulations a function must meet to be a kernel can be phrased in many ways, but the basic definition follows [CST00]. A function $K(\mathbf{x}, \mathbf{y}) : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is a kernel iff there exists a function ϕ which maps an example $\mathbf{x}$ from the input space $\mathcal{X}$ to a feature space $\mathcal{F}$ such that the for any two examples the kernel is the inner product of the transformed examples, $K(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}) \cdot \phi(\mathbf{y}) \rangle$. With respect to the kernel and the transformed feature space, the SVM classifier outputs a linear separator although it may be nonlinear with respect to the original input space. Kernel based methods have gained popularity for many reasons. Among these reasons are: (1) generalization results can be obtained without reference to the dimensionality of the feature space; (2) since only the kernel output value is needed, it is possible to work in intractably large feature spaces when the result of the inner product can still be efficiently computed directly; (3) it is not necessary to explicitly define the feature space since any symmetric positive semi-definite matrix M defines a valid kernel where $M_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$; (4) simple algorithms can easily be adapted to a new task by choosing an appropriate kernel; (5) it is easier to prove things about simple algorithms. Since we only apply a linear kernel to the text classification problem, we derive results in the standard Euclidean space. As mentioned above, if the examples have been normalized to the unit sphere (L2-norm) then the inner product between two examples is equivalent to the cosine similarity function. By defining what it means to project from one example to another through the kernel space, all of the following variables could be adapted to any kernel.

Given a set of training data $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ which have corresponding class labels $y_i \in \{-1, +1\}$, the score function $f(\mathbf{q})$ for an SVM can be written in one of two equivalent ways. In terms of the training examples or in terms of the feature space. The first is written as:

$$f(\mathbf{q}) = \sum_{i=1}^{N} \alpha_i y_i K(\mathbf{x}_i, \mathbf{q}) + b. \tag{5.21}$$

The second can be written as:

$$f(\mathbf{q}) = \langle \mathbf{w} \cdot \phi(q) \rangle + b \quad \text{where} \quad \mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \phi(\mathbf{x}_i). \tag{5.22}$$

The decision rule is then simply $\mathrm{sign}(f(q))$. Only the points that have non-zero α_i values change the scoring function and are therefore termed *support vectors*. When the feature space is tractable, such as in the linear case, the second formula gives a way of computing the normal to the separating hyperplane during training and quickly classifying examples at test time. When dealing with the feature space is not tractable (*e.g.*, high-order polynomial kernels), the first formula gives a way of classifying examples only in terms of the kernel values using a total of V kernel evaluations where V is the number of support vectors. Likewise the problem of training an SVM can be formulated as solving for the α's and b or as solving for a $\mathbf{w}$ and b.

In looking at how this solution is chosen, we come upon the second key concept for SVMs — choosing the maximum margin solution. When there is a perfect separator, this simply says choose the separator that has the largest minimum distance to any of the training points. When there is no perfect separator, the conditions must be generalized. Conceptually, we can think of the amount we must move each point to be on the correct side of the boundary. Additionally, we may want to penalize points that are on the correct side but too close to the boundary. Then a reasonable solution would be to choose the separator that minimizes the sum of these terms. The formulation implemented in SVM*light* that incorporates these ideas is the 1-norm soft margin SVM [Joa02, CST00]:

$$\text{minimize}_{\xi,\mathbf{w},b} \quad \frac{1}{2}\langle \mathbf{w} \cdot \mathbf{w} \rangle + C \sum_{i=1}^{N} \xi_i \tag{5.23}$$

$$\text{subject to} \quad y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \geq 1 - \xi_i, \quad i = 1, \ldots, N \tag{5.24}$$

$$\xi_i \geq 0 \quad i = 1, \ldots, N. \tag{5.25}$$

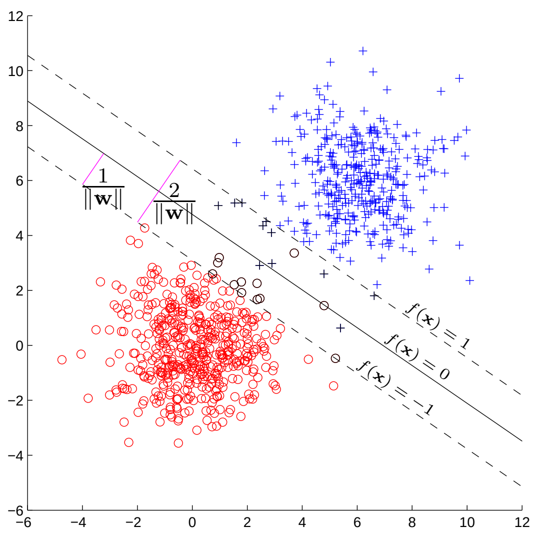


Figure 5.2: The SVM*light* solution with default C for an almost linearly separable problem. The decision boundary is shown with a solid line. The dashed lines show the limits of the margin. The support vectors are highlighted in black.

Figure 5.2 illustrates the solution for a nearly separable dataset. When the dataset has a perfect separator the only support vectors will lie at the boundaries of the margin on the $f(\mathbf{x}) = 1$ and $f(\mathbf{x}) = -1$ lines. In the general case, the support vectors will include all training points $\mathbf{x}_i$ such that $y_i f(\mathbf{x}_i) \leq 1$, *i.e.* all points within the margin and all of the points on the "wrong side" of the hyperplane. It can also be seen from one of the KKT conditions that the optimal solution must satisfy, $\forall_{i=1}^{N} \;\; \xi_i(\alpha_i - C) = 0$ [CST00] (p. 107). The ξ_i are referred to as slack variables since by the condition in 5.24 together with the minimization, they are the amount that a point falls on the wrong side of the margin. Thus the sum $\sum_{i=1}^{N} \xi_i$ is the total distance points must be moved in order to be left with a margin free of training points. As labeled in Figure 5.2, the margin has a width of $\frac{2}{\|\mathbf{w}\|}$; thus the optimization problem of Eq. 5.23 is a tradeoff between the margin size and the total slack controlled by the parameter C. Referring back to the KKT condition mentioned above and since $0 \leq \alpha_i \leq C$, one can easily see that any training point that has non-zero slack will have an alpha of C and for those training points that fall on the margin boundary and thus have a slack of zero, their respective α's may range subject to other conditions not discussed here. We have only touched on an understanding of SVMs necessary to motivate the following discussion. The interested reader should see [Joa02, CST00] for much more detailed information related to the theory and implementation of SVMs.

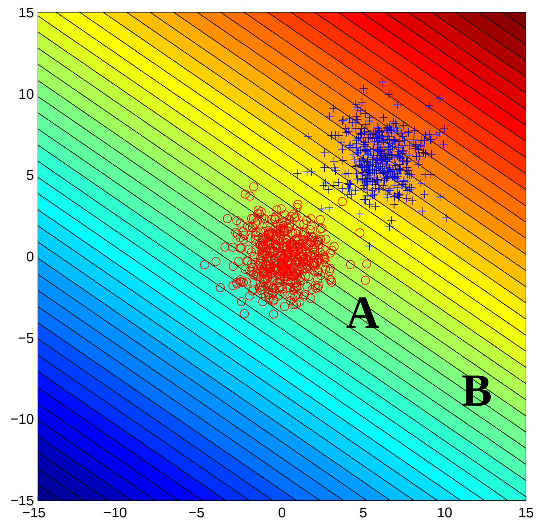


Figure 5.3: The contours for the score function, $f(\mathbf{x})$, of the SVM^{light} solution with default C. The labels “A” and “B” fall at the same distance to the separator but would we have equal confi dence at predicting “red circle” at both points?

Motivating SVM Failure Modes

As we mentioned in Section 3.2 the SVM’s score function $f(\mathbf{q})$ has empirically proven to behave like a linear transform of the log-odds of an example. Figure 5.3 shows that this function takes equal values parallel to the decision boundary. This seems generally acceptable for cases where there is good separation except that when comparing the points labeled “A” and “B”, most people find they are more uncertain about the label of “B” than “A”. How should this be quantified? Intuitively, we are not more certain that point “B” is a “blue cross”. Instead, we simply have less certainty about our probability, or in other words our estimate of variance is higher. Thus, we would like a reliability indicator that captures this intuitive observation mechanically.

We can easily construct an example that has the same decision boundary but is less good by increasing the nonseparability of the data. Figure 5.4 shows an extreme of one such nonseparable case. While the decision boundary is still about the best we can do for a linear separator, it seems clear that we actually want to decrease our estimate of relevance near the nonseparable mass; as we move further away along the isolines, it becomes increasingly unclear which class is best to predict. In practice, if the SVM solution has good generalization we will not find large clumps such as this, but we may find scattered points. We would like to define reliability indicators that will assist us in automatically detecting such behavior.

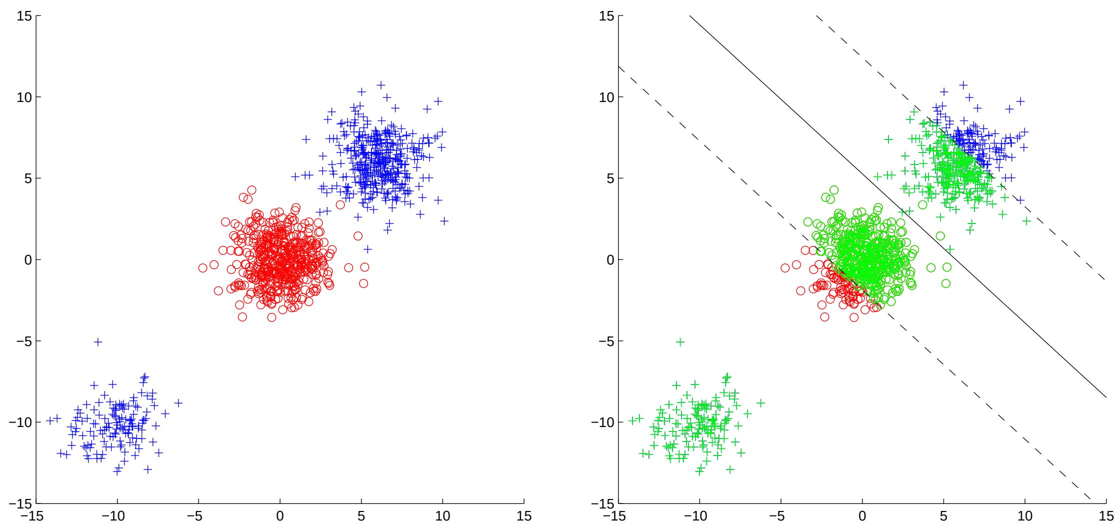


Figure 5.4: The same data as Figure 5.2 but with a large non-separable mass added. The set of support vectors (*in green*) has changed but the decision boundary is close to the same. Is it still reasonable to assume the true log-odds is a (piecewise) linear transform of $f(\mathbf{x})$?

SVM Variable Details

Now, similar to how we proceeded for the other classification algorithms, we will define a small set of shifts to the input example and measure statistics of the resulting change in the model's score function. In the derivation, we hope to both develop a reasonable estimate of the model's sensitivity and automatically capture the types of failure modes we discussed above.

Since we derive these for the linear kernel only, $\phi(\cdot)$ is the identity function and we drop it. Given a SVM model $f(\mathbf{x})$ and a set of data D, let $\mathcal{V} = \{\mathbf{x_i} \in D \mid y_i f(\mathbf{x}) \leq 1\}$. As discussed previously if D is the training data, then for the SVM optimization problem above, $\mathcal{V}$ is the set of support vectors. We may also choose to estimate these variables based on some other set of data D. Let $V = |\mathcal{V}|$. Now, we will consider shifting the document $\mathbf{d}$ toward each of the elements of $\mathcal{V}$.

In particular, let $\mathbf{d}_i$ denote the document that has been shifted by a factor β_i toward the ith element of $\mathcal{V}$, *i.e.* $\mathbf{d}_i = \mathbf{d} + \beta_i(\mathbf{v}_i - \mathbf{d})$. Similar to the derivation for the kNN classifier, to determine β_i we define it in terms of the closest point in $\mathcal{V}$ to d. Let ϵ be half the distance to the nearest point in $\mathcal{V}$, *i.e.* $\epsilon = \frac{1}{2}\min_{\mathbf{v}\in\mathcal{V}} \|\mathbf{v} - \mathbf{d}\|$. Then $\beta_i = \frac{\epsilon}{\|\mathbf{v}_i - \mathbf{d}\|}$.[9] Thus the shift vectors are all rescaled to have the same length. Now, we must define a probability for the shift. We use a simple exponential based on the relative distance from the document to the point and the closest point in $\mathcal{V}$. Let $\mathbf{d}_i \sim \Delta$ where $P_\Delta(\mathbf{d}_i) \propto \exp(-\|\mathbf{v}_i - \mathbf{d}\| + 2\epsilon)$ and

[9]We assume that the minimum distance is not zero. If it is zero, then we return zero for all of the variables.

$\sum_{i=1}^{V} P_\Delta(\mathbf{d}_i) = 1$.[10] Our first two variables are $E_\Delta[f(\mathbf{d}_i) - f(\mathbf{d})]$ and $\mathrm{Var}_\Delta^{1/2}[f(\mathbf{d}_i) - f(\mathbf{d})]$ and denote them as *SVMShiftMeanConfDiff* and *SVMShiftStdDevConfDiff*. Note that to compute the first of these, we can rewrite it as:

$$\sum_{i=1}^{V} P(\mathbf{d}_i)\,[f(\mathbf{d}_i) - f(\mathbf{d})] \tag{5.26}$$

$$= \sum_{i=1}^{V} P(\mathbf{d}_i)\,[\langle \mathbf{w} \cdot \mathbf{d}_i \rangle + b - f(\mathbf{d})] \tag{5.27}$$

$$= \sum_{i=1}^{V} P(\mathbf{d}_i)\,[\langle \mathbf{w} \cdot [\mathbf{d} + \beta_i(\mathbf{v}_i - \mathbf{d})] \rangle + b - f(\mathbf{d})] \tag{5.28}$$

$$= \sum_{i=1}^{V} P(\mathbf{d}_i)\,[\langle \mathbf{w} \cdot \mathbf{d} \rangle + \langle \mathbf{w} \cdot \beta_i(\mathbf{v}_i - \mathbf{d}) \rangle + b - f(\mathbf{d})] \tag{5.29}$$

$$= \sum_{i=1}^{V} P(\mathbf{d}_i)\,[\langle \mathbf{w} \cdot \beta_i(\mathbf{v}_i - \mathbf{d}) \rangle] \tag{5.30}$$

$$= \sum_{i=1}^{V} P(\mathbf{d}_i)\,[\beta_i \langle \mathbf{w} \cdot \mathbf{v}_i \rangle - \beta_i \langle \mathbf{w} \cdot \mathbf{d} \rangle] \tag{5.31}$$

$$= \sum_{i=1}^{V} P(\mathbf{d}_i)\beta_i\,[\langle \mathbf{w} \cdot \mathbf{v}_i \rangle - \langle \mathbf{w} \cdot \mathbf{d} \rangle] \tag{5.32}$$

$$= \sum_{i=1}^{V} P(\mathbf{d}_i)\beta_i\,[\langle \mathbf{w} \cdot \mathbf{v}_i \rangle + b - \langle \mathbf{w} \cdot \mathbf{d} \rangle - b] \tag{5.33}$$

$$= \sum_{i=1}^{V} P(\mathbf{d}_i)\beta_i\,[f(\mathbf{v}_i) - f(\mathbf{d})] \tag{5.34}$$

Similarly to compute *SVMShiftStdDevConfDiff*, we can use

$$\sqrt{\sum_{i=1}^{V} P(\mathbf{d}_i)\beta_i^2\,[f(\mathbf{v}_i) - f(\mathbf{d})]^2 - \left[\sum_{i=1}^{V} P(\mathbf{d}_i)\beta_i\,[f(\mathbf{v}_i) - f(\mathbf{d})]\right]^2} \tag{5.35}$$

Since the scores for each of the vectors $\mathbf{v}_i$ can be computed during training, we do not incur any additional time to compute those at test time. Instead, we have one pass over the V vectors to find the nearest neighbor, one pass to compute the probability function, and a final pass to compute the sum. Therefore, we can compute the variables in $3V$ or $O(V)$ time.

[10] As is standard to handle different document lengths, we take the distance between documents after they have been normalized to the unit sphere. This is not the case for the example figures with generated data we present here.

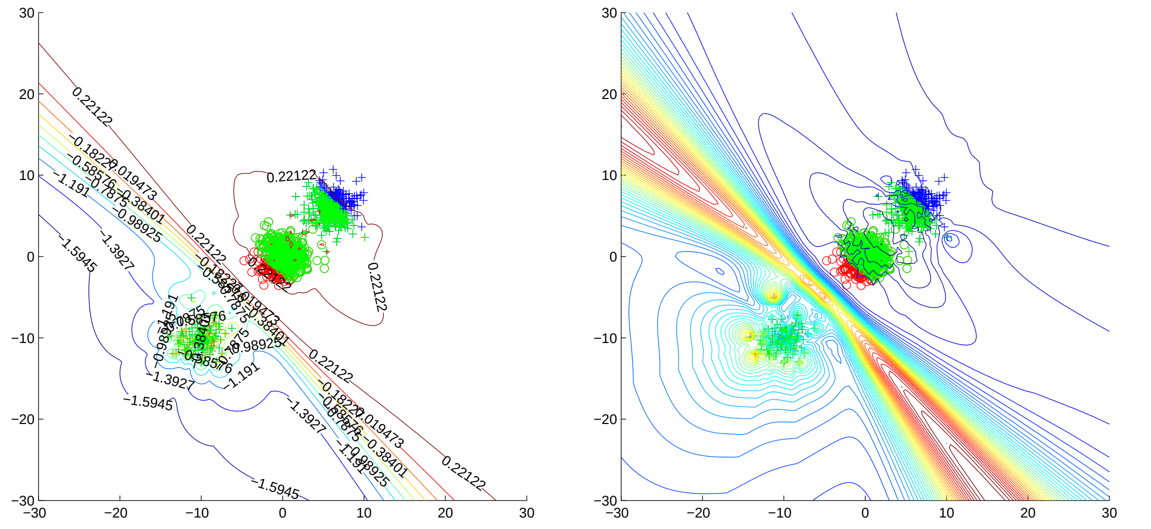


Figure 5.5: The contour plots of *meanGoodSVProximity (left)* and *stdDevGoodSVProximity (right)* appear to capture some of the motivating intuition. Note the negative values in the left plot near the nonseparable mass. In the right mass we see the goodness variance rises in the nonseparable mass as well as the regions to the side where its unclear which mass examples belong to. Meanwhile variance in the nicely separated region remains low and stable.

Inspired by the final form in Eq. 5.34 and by the idea that the total slack is the amount required to move all the points to the right side of the margin, we considered $E_{\Delta}[G(\mathbf{d}_i)]$ and $\mathrm{Var}_{\Delta}^{1/2}[G(\mathbf{d}_i)]$ where $G(\mathbf{d}_i) = y_i \beta_i f(\mathbf{v}_i)$. The idea is that $G(\cdot)$ is a “goodness” function. Note that $G(\mathbf{d}_i)$ will be positive when $\mathbf{v}_i$ is a support vector on the correct side of the hyperplane, and it will be negative when $\mathbf{v}_i$ is a support vector on the wrong side of the hyperplane. We receive a credit proportional to how far the support vector is on the correct side and the size of the shift we defined earlier. Thus if the expectation is positive, it means that, on average, the document is closer to “good” support vectors than to “bad” ones. Likewise, the variance captures the fact that there is high fluctuation among nearness to good and bad support vectors. We call these variables *meanGoodSVProximity* and *stdDevGoodSVProximity*. Figure 5.5 shows an example of the values they take on and demonstrates that they capture our intuition to a certain extent. Like the variables above they are also $O(V)$ to compute.

Finally, we argued that distance to known points is intuitively important as in Figure 5.3. Since we already find the nearest support vector to the document for the other variables, then we also output a signed distance-weighted function to the nearest support vector, specifically $y_n \exp(-\|\mathbf{v}_n - \mathbf{d}\|)$ where $\mathbf{v}_n = \min_{\mathbf{v} \in \mathcal{V}} \|\mathbf{v} - \mathbf{d}\|$. We call this variable *signed-NNSV*.

5.2 Inputs for STRIVE (Document Dependent)

The remainder of this chapter summarizes all of the inputs to the metaclassifiers — including the base classifier outputs and all the reliability-indicator variables. A brief motivation is provided for each reliability indicator that has not been previously discussed. As was the case with the earlier variables these variables do not exhaustively define the space of reliability indicators nor are they always crucial. Instead they simply attempt to mathematically capture the intuitions discussed through any or all of the following: approximations, convenient modeling choices, and algorithm-specific key quantities.

5.2.1 Outputs of Base Classifi ers

We considered the outputs of five base classifiers as inputs to *STRIVE*.

- *OutputOfDnet*
  This is the output of the decision tree built using the *Dnet* classifier (available as the WinMine toolkit [Mic01]. Its value is the smoothed log-odds of the estimated posterior probability at the leaf node of belonging to the class.

- *OutputOfSVMLight*
  This is the output of a Linear SVM model built using the *SVM*^{light} package [Joa99]. It is often termed the margin score or distance to the normal. If K is the kernel function, α_i is the non-negative weight that the SVM places on each training example (*i.e.*, support vectors have non-zero weight), and β is the bias or threshold, then this score is $\beta + \sum \alpha_i y_i K(\vec{x}_i, \vec{x})$.

- *OutputOfNaï veBayes*
  This is the output of the *naï ve Bayes* model built using a multivariate Bernoulli representation (*i.e.* only feature presence/absence in an example is modeled) [MN98]. It is the log-odds (or logistic) of the model's probability estimate of class membership, *i.e.* $\log \frac{P(c|d)}{1-P(c|d)}$. Because of machine floating-point precision issues, it is necessary that the implementation compute the log-odds directly.

- *OutputOfUnigram*
  This is the output of the *unigram* model (also referred to as a multinomial model [MN98]). It is the log-odds (or logistic) of the model's probability estimate of class membership, *i.e.* $\log \frac{P(c|d)}{1-P(c|d)}$. Because of machine floating-point precision issues, it is necessary that the implementation compute the log-odds directly.

- *OutputOfkNN*
  This is the output of the kNN classifier using distance-weighted voting [Yan99]. Unless otherwise mentioned, the value of k is set to be $2(\lceil \log_2 N \rceil) + 1$ where N is the number of training points. This rule for choosing k is theoretically motivated by results which show such a rule converges to the optimal classifier as the number of training points increases [DGL96]. In practice, we have also found it to be a computational convenience that frequently leads to comparable results with numerically optimizing k via a cross-validation procedure. As a distance-measure for neighbors, we use $\cos(\vec{x}_1, \vec{x}_2)$ with higher values indicating greater similarity (closer neighbors). The score used for a class y is:

$$s(y) = \sum_{\vec{n} \in kNN(\vec{x}) | c(\vec{n}) = y} \cos(\vec{x}, \vec{n}) \quad - \sum_{\vec{n} \in kNN(\vec{x}) | c(\vec{n}) \neq y} \cos(\vec{x}, \vec{n}) \tag{5.36}$$

5.2.2 Reliability Indicator Variables

Indicator variables are currently roughly broken into one of four types:

- Amount of information present in the original document;
- Information loss or mismatch between representations;
- Sensitivity of the decision to evidence shift;
- Basic voting statistics.

We group the reliability indicators into their primary type based on the main reasons why we would expect to see a link to classifier reliability. We note that this is only a soft clustering; some reliability indicators provide context information in more than one way.

Several of the variables listed below have an instantiation for each class in a learning problem. The variable counts we report tally each instance separately. For the variables below we list only one entry and use "{Class}" in the name of the variable to denote that this variable has one instantiation per class. Since our methodology built a binary classifier for each topic, then our experiments have a *Positive* and a *Negative* class version. In a two-class problem, the values of the two instantiations may be redundant. We have, however, retained each since in polychotomous (3 or more classes) discrimination they are more distinct.

After each bullet below, a number is given in parentheses, indicating the number of variables that this description includes. There are currently 70 document-dependent reliability indicators. After we discuss how to use these indicators, we conduct an analysis of the empirical impact of each indicator in Section 7.3.

Type 1: Amount of information present in the original document

- (1) *DocumentLength*
  The number of words in a document before feature selection. Presumably longer documents provide more information to base a decision upon. Therefore, longer documents will lead to more reliable decisions when *DocumentLength* is correctly modeled. Alternatively, models that do not correctly normalize for document length may be less reliable for extreme lengths (short or long) of documents.

- (1) *EffectiveDocumentLength*
  DocumentLength minus the number of out-of-vocabulary words in the document. Since a model cannot generalize strongly, other than by smoothing, for features that were not seen in the training set, this variable may be a better indicator of information present in the document than *DocumentLength*.

- (1) *NumUniqueWords*
  Number of distinct tokens in a document, *i.e.* $|\{w|w \in document\}|$, as opposed to length which counts repeats of a token in a document. The motivation is similar to *DocumentLength*, but here the variable is only counting each new word as an indicator of new information.

- (1) *EffectiveUniqueWords*
  NumUniqueWords minus the number of unique out-of-vocabulary words. This is the analogue of *EffectiveDocumentLength* and is included for similar reasons.

- (1) *PercentUnique*
  This is a measure of the variety in word choice in a document. It is equal to *NumUniqueWords* / *DocumentLength*. This can also be seen as 1 / average number of times a word is repeated in a document. Close to 1 means very few words (if any) are repeated in the document; close to 0 means the documents consists of very few unique words (possibly repeated many times). This is essentially a normalized version of *NumUniqueWords*; this variable will show high variance for short documents however. The intuition here is that more complex documents, while providing more information, also might be more difficult to classify since they may have many features, each carrying some small weight.

- (1) *PercentOOV*
  The percentage of the words in a document which weren't seen in the training set. It is equal to the number of out-of-vocabulary words divided by *DocumentLength*. Similar to *PercentUnique*, this variable can show high variance for short values. The intuition here is that the more novel words a document contains the more likely a classifier is to incorrectly classify the document into the a priori prevalent class. Typically unseen words slightly favor minority classes, since we have less samples from them. This is a variable that essentially allows a global smoothing model to be induced and its range is $[0, 1]$. Therefore, as it approaches 1, we would expect minority classes to be more likely than our base models might estimate.

- (1) *PercentUniqueOOV*
  The percentage of the words in a document, not counting duplicates, which weren't seen in the training set. This is the distinct token analogue for *PercentOOV*. Again, the motivations are similar to just using a different information model.

- (2) *PercentIn*$\{$*Class*$\}$*BeforeFS*
  Of all words occurring in the training set (*i.e.* out-of-vocabulary words are ignored), the percentage of words in a document that occurred at least once in examples belonging to the class. It is equal to the number of words that occurred in the class before feature selection divided by *EffectiveDocumentLength*. Similar to *PercentOOV*, this can be used to inductively learn smoothing behavior. The assumption is that if this variable is high, predictions that the example belongs to the class are more reliable. For the binary case with a negative class that effectively groups many classes together, this isn't quite expected with respect to *PercentInNegBeforeFS* (since predictions of "negative" would almost always expected to be more reliable under that assumption).

- (2) *UpercentIn*$\{$*Class*$\}$*BeforeFS*
  Of all words occurring in the training set (*i.e.* out-of-vocabulary words are ignored), the percentage of unique words in a document that occurred at least once in examples belonging to the class. This is the analogue to *PercentIn*$\{$*Class*$\}$*BeforeFS* using unique tokens as the basis for the information model.

- (2) *%Favoring*$\{$*Class*$\}$*BeforeFS*
  Of all words occurring in the training set, the percentage of words in a document that occurred more times in examples belonging to the class than in examples not belonging to the class. This is essentially a rough statistic for an *unnormalized* unigram

model (tied slightly into the smoothing related variables discussed above) that gives a very rough sense of the evidential weight of the original document.

- (2) *U%Favoring{Class}BeforeFS*
  Of all words occurring in the training set, the percentage of unique words in a document that occurred more times in examples belonging to the class than in examples not belonging to the class. This is the analogue to *%Favoring{Class}BeforeFS*.

Type 2: Information loss or mismatch between representations

While each of these variables are a measure of loss of information, they all generally have a paired variable of Type 1 that together give a more direct measure of information loss.

- (1) *DocumentLengthAfterFS*
  The number of words in a document after out-of-vocabulary words have been removed and feature selection was performed. Similar to *DocumentLength*, this is the measure of information that the classifier actually sees with respect to this document.

- (1) *UniqueAfterFS*
  The number of unique words remaining in a document after out-of-vocabulary words have been removed and feature selection was performed. This is the distinct token analogue of *DocumentLengthAfterFS* and is similarly expected to be used in conjunction with *NumUniqueWords* as a gauge of information loss.

- (1) *PercentRemoved*
  The percentage of a document that was discarded because it was out-of-vocabulary or removed by feature selection. It can have high variance for short documents. The intuition is that reliability of a classifier is higher for low values of *PercentRemoved*.

- (1) *UniquePercentRemoved*
  The percentage of unique words in a document that were discarded because they were out-of-vocabulary or removed by feature selection. The distinct token analogue of *PercentRemoved* where the information model is unique words.

- (2) *PercentIn{Class}AfterFS*
  Of all words occurring in the training set, the percentage of words remaining in a document after feature selection that occurred at least once in examples in the class. Together with *PercentIn{Class}BeforeFS*, this allows the model to represent shift in information content because of feature selection.

- (2) *UpercentIn{Class}AfterFS*
  Of all words occurring in the training set, the percentage of unique words remaining in a document (after feature selection) that occurred at least once in the class. Again, this is expected to be used in conjunction with *UpercentIn{Class}BeforeFS* to model information loss.

- (2) *%Favoring{Class}AfterFS*
  Of all words occurring in the training set, the percentage of words remaining in a document (after feature selection) that occurred more times in examples in the class than in examples not in the class. Like its BeforeFS counterpart, it is essentially like an unnormalized unigram model. We expect that it can be used in conjunction with *%Favoring{Class}BeforeFS* to measure how a feature selection method may have biased the information for a given document toward a particular class.

- (2) *U%Favoring{Class}AfterFS*
  Of all words occurring in the training set, the percentage of distinct words remaining in a document after feature selection that occurred more times in examples in the class than examples not in the class.

- (12) *FSInformationChange*
  The change in information according to some measure of information. There is one instantiation per measure of information and per measure of class. The difference produces a variable related to information loss due to feature selection. We would expect a large negative difference might give rise to false positives while a large positive difference might give rise to false negatives with respect to the class. The following are grouped under this heading:
  (1) *NumWordsDiscarded*
  = DocumentLength - DocumentLengthAfterFS
  (1) *NumTrainingWordsDiscarded*
  = EffectiveDocumentLength - DocumentLengthAfterFS
  (1) *NumFeaturesDiscarded*
  = NumUniqueWords - UniqueAfterFS
  (1) *NumTrainingFeaturesDiscarded*
  = EffectiveUniqueWords - UniqueAfterFS
  (2) *WordsSeenIn{Class}Delta*
  = PercentIn{Class}BeforeFS - PercentIn{Class}AfterFS
  (2) *FeaturesSeenIn{Class}Delta*
  = UpercentIn{Class}BeforeFS - UpercentIn{Class}AfterFS

(2) *PercentWordsPointingTo*{*Class*}*Delta*
= *%Favoring*{*Class*}*BeforeFS* - *%Favoring*{*Class*}*AfterFS*
(2) *UPercentWordsPointingTo*{*Class*}*Delta*
= *U%Favoring*{*Class*}*BeforeFS* - *U%Favoring*{*Class*}*AfterFS*

Type 3: Sensitivity of the decision to evidence shift

- (2) *UnigramStdDeviation*, *Naï veBayesStdDeviation*
In a binary class problem, the weight each word contributes to the unigram model's decision is $\log \frac{P(w|c)}{P(w|\neg c)}$. Similarly, each word's presence/absence contributes a weight of $\log \left[\frac{P(w(d)=\{\text{present,absent}\}|c)}{P(w(d)=\{\text{present,absent}\}|\neg c)} \frac{1-P(w(d)|\neg c)}{1-P(w(d)|c)}\right]$ to the naïve Bayes model. If this term is greater than 0, the word gives evidence to the positive class (c), and if it is less than zero, the word gives evidence to the negative class ($\neg c$). The reliability indicators are the standard deviation of these weights for the feature values (word occurrences or presence/absence) in a specific document. If the variance is close to zero, that means all of the words tended to point toward one class. As the variance increases, this means there was a large skew in the amount of evidence presented by the various words (possibly strong words pulling toward two classes). The intuition is that the reliability of naïve Bayes related classifiers will tend to decrease as this variable increases. The motivation behind these variables are described more fully in Sections 5.1.1 and 5.1.2.

- (4) *UnigramMeanLogOfStrengthGiven*{*Class*}, *Naï veBayesMeanLogOfStrengthGiven*{*Class*}
These variables represent the mean of the conditional contributions of a word given a class. For example *UnigramMeanLogOfStrengthGivenPositive* is the mean of $\log P(w|Class = Positive)$ over the words in the document and *Naï veBayesMean-LogOfStrengthGivenPositive* is the mean of $\log \frac{P(w(d)|Class=Positive)}{P(w'(d)|Class=Positive)}$ over the words in the vocabulary. The motivation behind these variables are described more fully in Sections 5.1.1 and 5.1.2.

- (4) *UnigramStdDeviationLogOfStrengthGiven*{*Class*}, *Naï veBayesStdDeviationLogOfStrengthGiven*{*Class*}
These variables represent the standard deviation of the conditional contributions of a word given a class. For example *UnigramStdDeviationLogOfStrengthGiven*{*Positive*} is the standard deviation of $\log P(w|Class = Positive)$ over the words in the docu-

ment and *Naï veBayesStdDeviationLogOfStrengthGiven*$\{$*Positive*$\}$ is the standard deviation of $\log \frac{P(w(d)|Class=Positive)}{P(w'(d)|Class=Positive)}$ over the words in the vocabulary. The motivation behind these variables are described more fully in Sections 5.1.1 and 5.1.2.

- (2) *UnigramMeanShift*, *Naï veBayesMeanShift*

  In a binary class problem, the weight each word contributes to the unigram model's decision is $\log \frac{P(w|c)}{P(w|\neg c)}$. Similarly, each word's presence/absence contributes a weight of $\log \left[\frac{P(w(d)=\{\text{present,absent}\}|c)}{P(w(d)=\{\text{present,absent}\}|\neg c)} \frac{1-P(w(d)|\neg c)}{1-P(w(d)|c)}\right]$ to the naïve Bayes model. If this term is greater than 0, the word gives evidence to the positive class (c), and if it is less than zero, the word gives evidence to the negative class ($\neg c$). The reliability indicators are the mean of these weights for the feature values (word occurrences or presence/absence) in a specific document. As discussed in Sections 5.1.1 and 5.1.2 this quantity is the mean change in the model's output if we uniformly randomly chose one word to delete (for the unigram classifier) or uniformly randomly chose one feature's value to change (for the naïve Bayes classifier).

- (1) *NeighborhoodRadius*

  The radius required to include all k points in the neighborhood for the kNN classifier. The radius is the Euclidean distance between the query point and its neighbors after the points are normalized to the unit sphere. If the radius around a point is comparatively small then the implication is that this portion of the space is extremely well sampled comparatively, and higher reliability is expected there.

- (3 = 1 [overall] + 2 [one per class]) *MeanNeighborDistance*
  Mean$\{$*Class*$\}$*NeighborDistance*

  The mean distance of the points in the neighborhood from the kNN classifier. The distance used is Euclidean distance between the query point and its neighbors after the points are normalized to the unit sphere. This has a similar motivation to *NeighborhoodRadius* — a smaller neighborhood implies better sampling in the area and higher reliability. Depending on the particular type of distance weighting used in the kNN classifier this variable may play a more or less important role. There is one instantiation that averages over all neighbors and then one per class that is the average of points belonging to that class.

- (3 = 1 [overall] + 2 [one per class]) *SigmaNeighborDistance*
  Sigma$\{$*Class*$\}$*NeighborDistance*

  The standard deviation of the distance of the points in the neighborhood from the kNN classifier. The distance used is Euclidean distance between the query point and its neighbors after the points are normalized to the unit sphere. The assumption is that query points that fall in neighborhoods with high variance in neighbor distance will be less reliable than those with a similar mean in distance but less variance. There is one instantiation that measures variance over all neighbors and then one per class that is based on points belonging to that class.

- (1) *kNNShiftMeanPred*
  Similar to variables related to other classifiers, this variable considers how the output of the kNN classifier changes with slight changes to the prediction point. Since the kNN classifier uses a neighborhood for prediction, we also use it to define the nature of the perturbations to the input. k new queries are created by shifting the document as far toward each neighbor as it can go while the original document remains its own closest neighbor. This variable is the average class prediction over the original document and the k new queries. If the neighborhood is smooth we expect to see values near 0 and 1. The motivation for this variable as well as the computational approximations used to compute it more efficiently are described more fully in Section 5.1.3.

- (1) *kNNShiftMeanConfDiff*
  Using the k new queries defined for *kNNShiftMeanPred*, this variable measures the average change in the kNN classifier's confidence score from the confidence assigned to the original prediction. A value near zero indicates that the query falls into a neighborhood that is, on average, smooth with respect to the confidence function. The motivation for this variable as well as the computational approximations used to compute it more efficiently are described more fully in Section 5.1.3.

- (1) *kNNShiftStdDevConfDiff*
  Using the k new queries defined for *kNNShiftMeanPred*, this variable measures the standard deviation of the difference between the kNN classifier's confidence score for the new query and the confidence assigned to the original prediction. A low variance indicates that the neighborhood is smooth with respect to the confidence function. The motivation for this variable as well as the computational approximations used to compute it more efficiently are described more fully in Section 5.1.3.

- (1) *SVMShiftMeanConfDiff* Similar to variables related to other classifiers, this variable considers how the output of the SVM classifier changes with slight changes to the prediction point. Since the SVM classifier can be seen as a function of the support vectors, we use the support vectors to define the nature of the perturbations to the input. Let v be the number of support vectors. Then, v new documents d' are created by shifting the document toward each support vector for a total distance equal to that of half the distance to the nearest support vector. A probability distribution Δ over shifts is then defined based on the distance to the support vector. Finally, this variable is the expected difference between the new output and the original output according to the probability function, $E_{\Delta}\left[f(d') - f(d)\right]$. The motivation for this variable and the mathematical details are described more fully in Section 5.1.5.

- (1) *SVMShiftStdDevConfDiff* This variable is just like *SVMShiftMeanConfDiff* except that it is the standard deviation of the statistic $\mathrm{Var}_{\Delta}^{1/2}\left[f(d') - f(d)\right]$. Again more details are in Section 5.1.5.

- (1) *meanGoodSVProximity* This variable uses the shifts defined for *SVMShiftMeanConfDiff*, but instead computes the expectation $E_{\Delta}\left[y_i \beta_i f(\mathbf{v}_i)\right]$ where $\mathbf{v}_i$ is the support vector that we are shifting toward. Thus, the variable will take an overall positive value if more “good” support vectors (on correct side of margin) are closer to the document than “bad” support vectors. See Section 5.1.5 for more details.

- (1) *stdDevGoodSVProximity* This variable is just like *meanGoodSVProximity* except that it is the standard deviation of the statistic $\mathrm{Var}_{\Delta}^{1/2}\left[y_i \beta_i f(\mathbf{v}_i)\right]$. See Section 5.1.5 for more details.

- (1) *signedNNSV* For documents that are farther from any data seen during training, we intuitively have less certainty about their class labels. This variable attempts to capture that by computing a simple signed function of the nearest support vector $y_n \exp(-\|\mathbf{v}_n - \mathbf{d}\|)$ where $\mathbf{v}_n = \min_{\mathbf{v} \in \mathcal{V}} \|\mathbf{v} - \mathbf{d}\|$.

- (1) *DTreeShiftMeanConfDiff*
  This is a sensitivity variable related to the decision tree model. Assume that there is a probability for drawing a document similar to the current one but different enough to branch down an alternate branch along the path. This variable defines a model for such a deviation and then captures the expected change in the model’s log-odds output. See Section 5.1.4 for more details.

- (1) *DTreeShiftStdDevConfDiff*
  This variable is identical to *DTreeShiftMeanConfDiff* except that it captures the variance in the model’s log-odds output according to the probability model over similar documents. Again see Section 5.1.4 for more details.

Type 4: Basic voting statistics

There are 2 reliability indicator variables whose primary type is considered to be this type. Both of these were introduced mainly to reduce the data required to learn m-of-n rules in the decision tree metaclassifier.

- (1) *PercentPredictingPositive*
  We refer to this in the main text as *NumVotingForClass*. This variable is the percentage of base classifiers (out of all base classifiers) that vote for membership in the class. In our experimental evaluation, we only used one instantiation of this variable. This was added to help the search space since learning this m-of-n type of feature can require significant data for a decision tree learning algorithm (unless it is specifically altered for this).

- (1) *PercentAgreeWBest*
  This variable is referred to as *PercentAgreement* in the main text. For polychotomous problems, *PercentAgreement* can be used to indicate among how many classes the classifiers fracture their votes. Since there are only two classes here, we altered it to indicate the percent agreement with the best base classifier (the classifier that performed best over the training data).

5.3 Task-Dependent Variables

Here are a few examples of variables which could be included in the models that are pooled across tasks. These are variables that do not vary from document to document within a collection or classification problem but vary across them.

After each bullet below, a number is given in parentheses, indicating the number of variables that this description includes.

- (1) *NumTrainingPoints*
  Different learning algorithms often have learning curves. Including this variable allows the metaclassifier to inductively learn about crossovers in the learning curves of the models.

- (1) *%TrainingPointsIn*$\{$*Positive*$\}$
  This has the same motivation as *NumTrainingPoints* except since the number of positives in text are rare, there is reason to suspect the learning curve might be more closely correlated with the number of positive training points.

- (1) *NumberOfSupportVectors*
  The generalization bounds for an SVM are tied to the number of support vectors in the solution. Therefore, this is an obvious candidate for predicting the usefulness of, at least, the SVM model.

Chapter 6

Background for Empirical Analysis

Most of the remaining chapters contain an accompanying empirical analysis for the approaches they discuss. This chapter collects many of the common key elements to these experiments and presents them together for the ease of the reader. We start with a description of the various performance measures used to evaluate the quality of a classification model. In addition, we present the motivation behind each measure, a description of why it is relevant to the task, and discuss the implications improvement according to a particular measure has beyond the obvious. Next, we give an overview of the primary text collections used throughout the dissertation and describe common processing steps that influence the learned models, *e.g.* feature selection. Finally, we present implementation details of the base classifiers and document particular parameter settings that influence the learned models.

6.1 Classifier Performance Measures

When selecting among classifier performance measures which have previously been used, we are faced with choosing from rank-based measures, probability loss functions, and functions such as accuracy that simply measure the membership predictions without regard to the classifier's confidence score. A variety of researchers have used rank-based [LC96] measures of performance because they were interested in interactive systems in which a rank list of codes for each document would be displayed to users. Many other applications such as automatic routing or tagging require that binary class membership decisions be made for each document as it is processed. When binary decisions must be made, the nature of the application often requires a different penalty for false positives than false negatives. In order to perform well for a particular cost function, one approach ranks the documents according to a classification confidence score and then optimizes a score threshold for the particular cost function. A classifier that effectively ranks examples can thus be used for a range of different cost functions. Additionally, producing a stable and accurate ranking is sufficient to show the existence of a monotone transformation of the classifier's scores to accurate probability estimates [LZ05]. However, producing good rankings does not ensure that the optimal threshold can be optimally selected during training nor does it guarantee we will choose the best transformation to probability spaces. For example, Hull *et al.* [HPS96] found that, although combination techniques were able to improve document ranking, they did considerably less well at estimating probabilities.

Therefore, at various points in this dissertation, we examine ranking, probability loss functions, and common classification measures in order to obtain a holistic picture of the performance trade-offs involved for a particular learning approach.

6.1.1 Classifi cation Measures

The F_β measure, and particularly F1 [vR79, YL99], is one of the more commonly used ways to assess text classifier performance. This measure attempts to balance a classifier's *precision* and *recall*. Precision measures the ability of a classifier to accurately find items that belong in a class:

$$Precision = \frac{Correct\ Positives}{Predicted\ Positives}. \tag{6.1}$$

Therefore a classifier that has low precision tends to produce false positives. Recall measures the ability of a classifier to accurately find *all* the items belonging in the class:

$$Recall = \frac{Correct\ Positives}{Actual\ Positives}. \tag{6.2}$$

A classifier with low recall tends to produce false negatives — not correctly finding all of the actual positives. Depending on the nature of the task, precision and recall can have varying roles. For example, if we are building classifiers that will automatically sort a user's e-mail into subject-oriented folders then errors in precision would route a message to the wrong folder whereas an error in recall would leave the message in the catch-all INBOX folder. Errors in recall may produce lower savings for a user, since now the user must sort the remaining items, but is unlikely to cause errors such as not responding to an important mail that has been misfiled. The F_β measure is a weighted harmonic mean of precision and recall.

$$F_\beta = \frac{(\beta + 1) * Precision * Recall}{\beta * Precision + Recall}. \tag{6.3}$$

The most common choice of β, 1, weights precision and recall equally and yields the F1:

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall}. \tag{6.4}$$

Generally, the optimal F1 is obtained where precision equals recall, but as in other threshold optimization, finding this trade-off point may be difficult in practice.

Occasionally, the user of a system can specify at training time an actual cost associated with a false positive, FP, and a false negative, FN. When this is possible, we can write the expected cost (or utility) of a classification algorithm as a linear utility function,

$$C(FP, FN) = FP * P(FalsePositive) + FN * P(FalseNegative) \tag{6.5}$$

where $P(FalsePositive)$ and $P(FalseNegative)$ is the probability under the underlying distribution[1] that the classifier emits a false positive/negative. The most commonly used function in the literature is the error rate which is $FP = FN = 1$. However, the importance of varying cost functions has been recognized by many researchers because applications rarely have equal costs for different types of errors [PF01]. The text classification community has been particularly aware of their importance and has included linear utility functions for some time in such evaluations as TREC (the **t**ext **re**trieval **c**onference) [HR99, RH00].

From a decision theory point of view [DHS01], it is only the ratio of costs that influences a decision and not the strict numbers. So, a decision-maker would make the same policy decisions for $C(10, 1)$ and $C(100, 10)$. More precisely, if we are given probability estimates $\hat{P}(+|x)$ and $\hat{P}(-|x)$, then the optimal decision [DHS01] is to predict positive whenever:

$$\frac{\hat{P}(+|x)}{\hat{P}(-|x)} \geq \frac{FP}{FN}. \tag{6.6}$$

For log-odds, the optimal decision is calculated by taking the log of both sides in whatever base desired. In order to assess how sensitive performance is to the utility measure in a typical range of cost ratios, we explicitly consider results for $C(10, 1)$, $C(1, 10)$, and the error function.

6.1.2 Probability Loss Functions

Often the costs of a false positive/negative are not known during training time or may be dynamic — changing according to a user-specified setting. In this case, if we assume all linear utility cost functions are equally likely during prediction, then an effective classifier is equivalent to accurately assessing the probability of each example. Given a probability

[1]Or in sample when estimating

estimate, it is straightforward to make a hard-classification based on dynamic costs by applying standard Bayesian decision theory:

$$\hat{Y}(X) = \begin{cases} + & \text{if } \hat{P}(+ \mid X) * FN \geq \hat{P}(- \mid X) * FP \\ - & \text{o.w.} \end{cases} \tag{6.7}$$

Therefore, the question becomes one of obtaining high-quality probability estimates from a classifier. We discuss how such estimates can be obtained in Chapter 3. Here we turn to the question of what loss functions are appropriate for judging the quality of probability estimates.

Two standard scoring functions for probabilities are log-loss and squared error. Both of these are *proper scoring rules* [DF83, DF86] in the sense that a classifier's view of its expected performance is maximized when the classifier actually issues a probability of $\hat{p}$ when it assesses the probability to be $\hat{p}$, *i.e.*, the classifier cannot expect to gain from "hedging its bets". Log-loss is defined as:

$$\text{Log-loss}(x, c(x)) = \log \hat{P}(c(x) \mid x). \tag{6.8}$$

Unless specified, we will report base e or the natural logarithm. Thus log-loss falls in $[-\infty, 0]$ and penalizes heavily when the classifier assigns low probability to the true class. When given in this form, the desire is to maximize the log-loss. When minimization is more convenient, one simply works with the negative of this quantity.

Squared error is defined as:

$$\text{Error}^2(x, c(x)) = [1 - \hat{P}(c(x) \mid x)]^2. \tag{6.9}$$

Unlike log-loss, squared error falls in a bounded interval, $[0, 1]$, and the aim is to minimize this quantity. Both scoring measures have their strengths and weaknesses. The large penalties that can be inflicted by log-loss sometimes leads to solutions that can overfit a single outlier simply to avoid an infinite penalty, whereas under decision theory seeing a cost-ratio that inflicts unbounded penalties is unlikely. On the other hand, since squared error is bounded by one, minimizing it can lead to making predictions that appear confident but in actuality are very wrong. Our use of these measures is primarily restricted to when we are explicitly concerned with the calibration of the systems in question.

6.1.3 Ranking Measures

In addition to the performance scoring measures discussed, we can also examine the benefits that a classifier has to offer over a range of cost functions by examining receiver-operating characteristic (ROC) curves. An ROC curve plots true positive rate versus false positive rate for a classifier. If a classifier generates a score that can be used to rank the test set, then an ROC curve is produced by setting a classification threshold between pairs of adjacent examples that have distinguishable scores and then plotting a point corresponding to the false positive and true positive rate that are obtained using that threshold. For a given classifier, C_i, the true positive rate and false positive rate are $P(C_i(x) = + \mid c(x) = +)$ and $P(C_i(x) = + \mid c(x) = -)$.

Figure 6.1a highlights several aspects of ROC curves using our earlier example of two class-conditionally independent classifiers where Classifier 1 was based on feature X_1 and Classifier 2 was based on feature X_2. Recall that if each classifier perfectly predicts the posterior conditioned only on its respective feature, then Classifier 1 does far better than Classifier 2. By choosing the standard threshold of 0.5, we obtain a single true positive and false positive rate for each classifier; these are shown by the single points in the graph. By varying the thresholds θ_1 and θ_2 that the posterior must exceed in order to predict positive, a range of different true positive and false positive rates are produced.

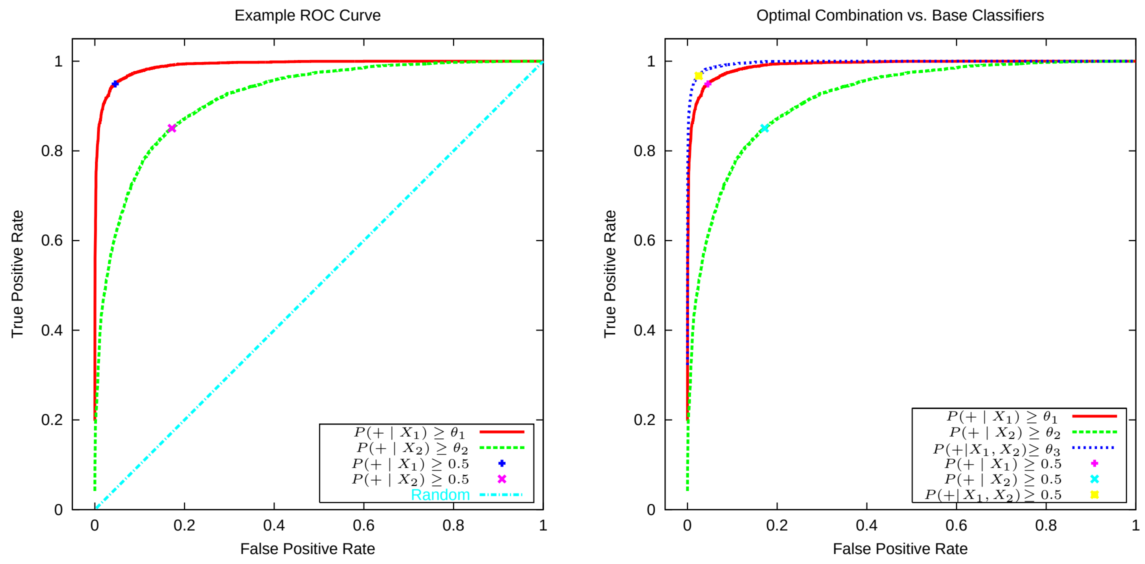


Figure 6.1: (a) At *left* an Example ROC Curve using the conditionally independent classifi er example of Section 1.2.1. (b) At *right*, the optimal combination of Classifi er 1 and 2 dominates both. The optimal combination has an error rate approximately half of Classifi er 1 and a sixth of Classifi er 2, but as the classifi ers get closer to perfect classifi cation, the graphical difference can appear deceptively small.

Perfect classification corresponds to a point in the top left (northwest) corner of the graph. Random performance falls on the $y = x$ line, and therefore anything above it performs better than the random baseline. As we know from the discussion above, the particular threshold that is optimal for a linear utility function depends on the relative weight of false positive to false negative. The ROC curve actually presents a summary of each classifier's performance under *any* linear utility function. For a given linear utility function, the error weights[2] define the slope of isolines that connect points in the graph with equal performance under that utility function (see Figures 6.4-6.6). Conceptually if a line with that slope is moved down in a parallel fashion from the northwest corner, then the first curve that touches a line is the optimal point for that linear utility function [PF01]. Therefore, for a set of classifiers, the optimal performance is defined by the convex hull of their curves. In cases like this example, where one curve is above all the others, that classifier is said to *dominate* the other classifiers, and if we are limited to selecting a single classifier to use for each linear utility function, the best choice is to always use the dominating classifier.

Of course, since we are interested in combining and not simply selecting, the optimal combination improves over both classifiers despite the fact that Classifier 1 dominates Classifier 2. This can be seen in Figure 6.1b. The optimal combination of Classifier 1 and 2 dominates both. The optimal combination has an error rate approximately half of Classifier 1 and a sixth of Classifier 2, but as the classifiers get closer to perfect classification, the graphical difference can appear deceptively small.

We can also visualize other metrics in ROC space [Fla03]. For example, Figure 6.2 contrasts the isolines of F1 in a precision-recall graph versus an ROC curve. In the precision-recall graph, the isolines do not change when the class prior is changed; however, as demonstrated by Figure 6.3, when positives become rare, F1 is extremely sensitive to any change in the upper left part of the ROC curve. In contrast, the isolines for error and linear utility functions are evenly spaced (Figures 6.4-6.6). These figures also demonstrate how varying the class prior is equivalent to varying the costs of a false positive/negative.

By examining the whole space of linear utility functions, we seek to understand how robust (in terms of sensitivity to the specific cost function) a combination method is. In order to attempt to summarize the the linear utility space of functions as a single scalar, we use area under the ROC curve.[3]

6.1.4 Summarizing Performance Scores

For each performance measure, we can either macro-average or micro-average. In macro-averaging, the score is computed separately for each class and then arithmetically averaged.

[2]Different class priors can actually be equivalently treated as a change in error weight ratio.

[3]Noting that the area under the curve is not a precise summary of the linear utility space.

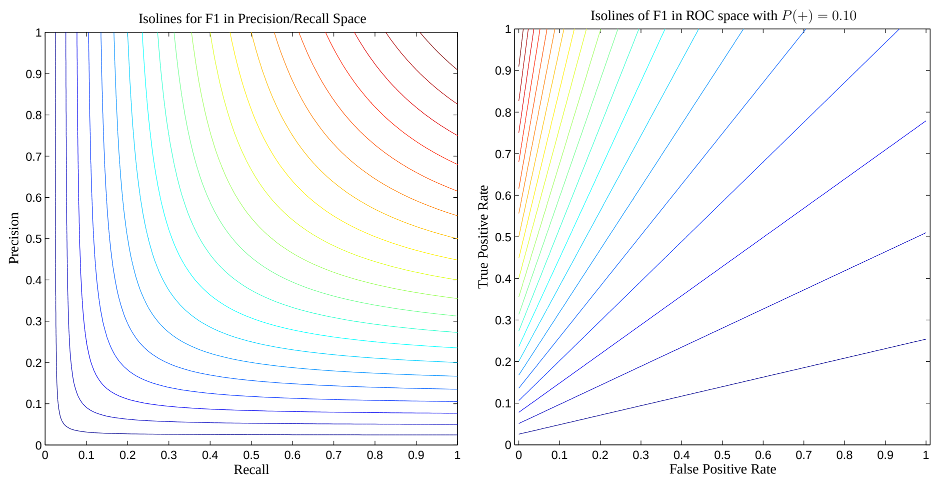


Figure 6.2: (a) At *left*, the isolines (or contours) connecting equal values of the F1 score in a Precision-Recall graph. The best performance is in the top right corner (red lines). (b) At *right*, the isolines connecting equal values of F1 in an ROC graph. The ROC graph has a free parameter of $P(+)$ that must be specifi ed to draw the contours. For this graph, $P(+) = 0.10$.

Although the classes are weighted equally in the macro-average, for some performance measures, in particular F1, predictions on instances of rare classes effectively carry more weight since the scores for rare classes are more sensitive to the changes of a few predictions over the positive instances; the net effect is that improvement in prediction over the rare classes tends to influence the macro-F1 score more than improving over the common classes. Micro-averaged values are computed directly from the binary decisions over all classes; since all instances are given equal weight, this places more weight on the common classes. For precision, recall, and F1 where micro-averaging is well-defined, we generally report both types of averages. For the other measures, there is no accepted definition of micro-averaging and some definitions (*e.g.*, linear utility functions) lead to identical macro and micro-averages. Therefore, for the other measures, we typically only report macro-averages.

6.2 Data

Here we briefly review the characteristics of the primary corpora we use for empirical evaluation throughout the dissertation. For other corpora and synthetic data only used in specific experiments, we describe them in the relevant section. Our primary empirical focus deals with several topic-classification corpora including: the *MSN Web Directory*;

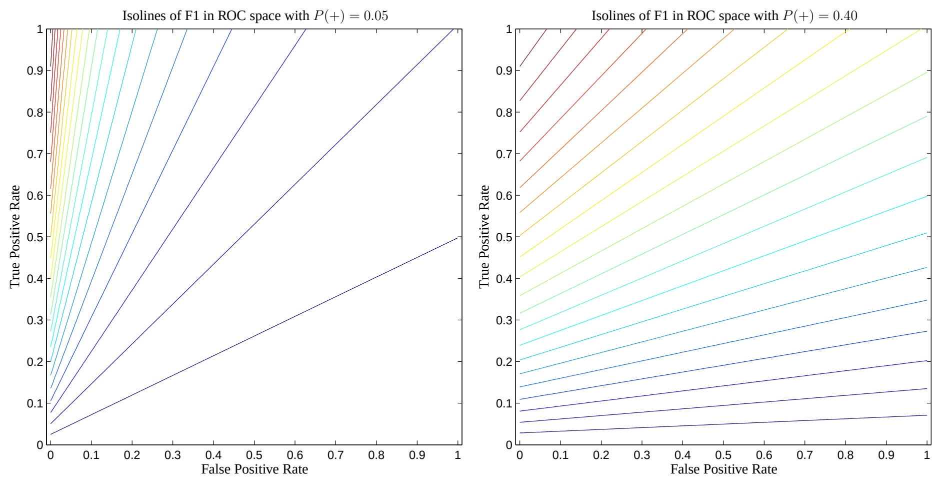


Figure 6.3: The effects of varying $P(+)$ from 0.05 to 0.40 on the isolines of F1 in ROC space.

two corpora drawn from the *Reuters* newswire; and the *TREC-AP* corpus. We have selected these corpora because they offer a wide array of topic classification problems from very broad topics over web pages to very narrow topics over financial news stories.

6.2.1 Chronological Split vs. Cross-Validation

Most text classification corpora are drawn from a source that has a natural time ordering. For example, every news story is published at a specific time (though we may only know the day), each e-mail is received at a given time, and a newsgroup posting happens at some specific instant. Because of this, text corpora can sometimes exhibit *topic drift* where the nature of the discussion or news events shift focus, and therefore the associated language distribution also shifts. As a result, many text classification researchers are proponents of splitting a corpus into a single training and test set chronologically [Lew92b]. This allows one to test a text classifier in a similar way to how it will be used. In many *applications* a chronological split — train on past, predict the future — is the only possible one even if there is a gradual shift in the underlying sampling distribution.

On the other hand, machine-learning researchers typically use cross-validation for evaluation because it allows one to average out the variance introduced by a single "unlucky/ lucky" split. In general, we follow the use of standard chronological splits in empirical evaluation in order to compare with other researchers for datasets where there is an established standard split and use appropriate statistical significance tests. For other datasets (*e.g.*, the

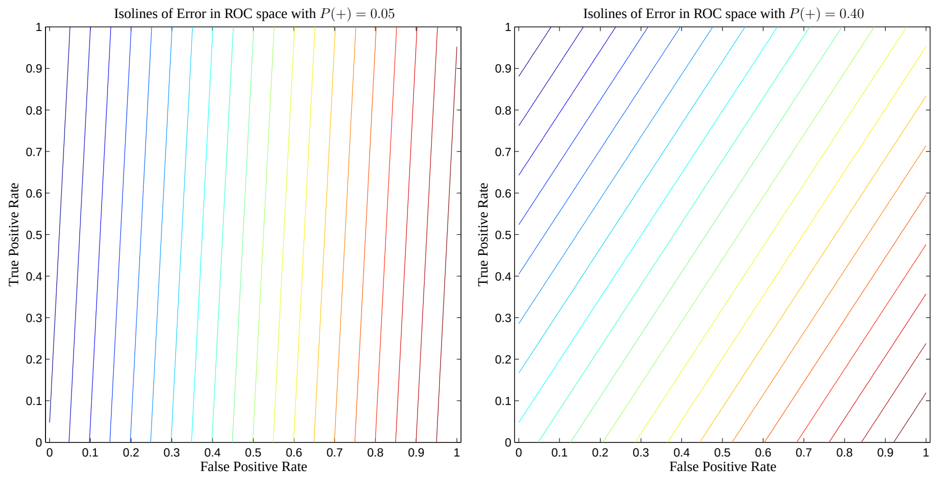


Figure 6.4: The effects of varying $P(+)$ from 0.05 to 0.40 on the isolines of Error in ROC space.

action-item corpus in Chapter 10), we use cross-validation techniques which allow us to make judgments of statistical significance with less training data.

6.2.2 MSN Web Directory

The MSN Web Directory is a large collection of heterogeneous web pages (from a May 1999 web snapshot) that have been hierarchically classified. We use the same chronological train/test split of 50078/10024 documents as that reported in [DC00].

The MSN Web hierarchy is a seven-level hierarchy; we use all 13 of the top-level categories. The class proportions in the training set vary from 1.15% to 22.29%. In the testing set, they range from 1.14% to 21.54%. The classes are general subject categories such as *Health & Fitness* and *Travel & Vacation*. Human indexers have assigned the documents to zero or more categories. Approximately 195K words appear in at least three training documents.

6.2.3 Reuters (21578)

The Reuters 21578 corpus [Lew97] contains Reuters news articles from 1987. For this data set, we use the ModApte standard chronological train/test split of 9603/3299 documents (8676 unused documents). The classes are economic subjects (*e.g.*, “acq” for acquisitions, “earn” for earnings, etc.) that human taggers applied to the document; a document may have multiple subjects. There are actually 135 classes in this domain (only 90 of which oc-

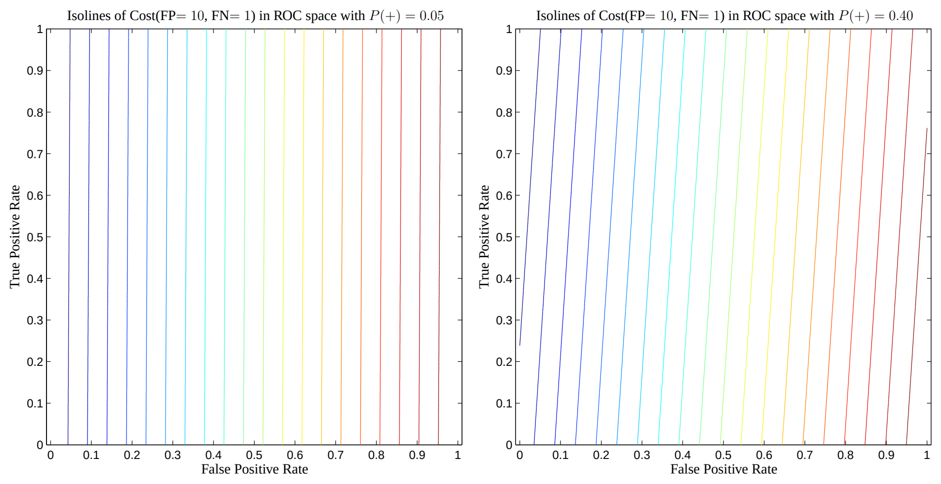


Figure 6.5: The effects of varying $P(+)$ from 0.05 to 0.40 on the isolines of Cost(FP= 10, FN= 1) in ROC space. Note that varying the costs of a linear utility function is exactly equivalent to varying the prior for the error scoring function.

cur in the training and testing set); however, we have only examined the ten most frequent classes since small numbers of *testing* examples makes estimating some performance measures unreliable due to high variance.[4] Limiting the topic set to the ten largest classes allows us to compare our results to previously published results [DPHS98, Joa98, MN98, Pla99].

The class proportions in the training set vary from 1.88% to 29.96%. In the testing set, they range from 1.7% to 32.95%. Approximately 15K words appear in at least three training documents.

6.2.4 TREC-AP

The TREC-AP corpus is a collection of AP news stories from 1988 to 1990. We use the same chronological train/test split of 142791/66992 documents that was used in [LSCP96]. As described in [LG94] (see also [Lew95]), the categories are defined by keywords in a keyword field. The title and body fields are used in the experiments below. There are twenty categories in total.

The frequencies of the twenty classes are the same as those reported in [LSCP96]. The class proportions in the training set vary from 0.06% to 2.03%. In the testing set, they range from 0.03% to 4.32%. Approximately 123K words appear in at least 3 training documents.

[4]Primarily area under the ROC curve.

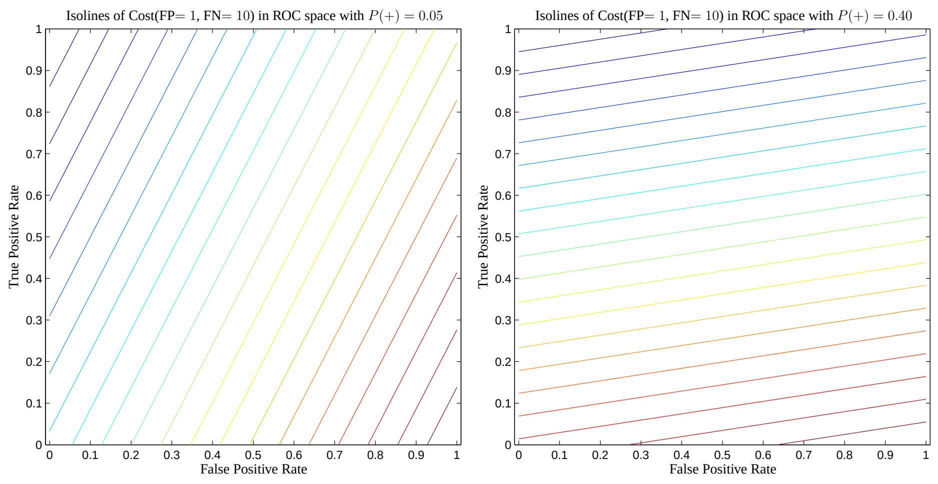


Figure 6.6: The effects of varying $P(+)$ from 0.05 to 0.40 on the isolines of Cost(FP= 1, FN= 10) in ROC space. Note that varying the costs of a linear utility function is exactly equivalent to varying the prior for the error scoring function.

6.2.5 RCV1-v2 (Reuters 2000)

To demonstrate we have not overfit our methods to the primary corpora used throughout the course of this research, in addition to the diversity of the granularity and type of documents among the corpora, we have included this corpus only after all of the combination methods and reliability indicators were fully developed. The *Reuters Corpus Volume 1* contains Reuters news articles from 1996-1997 which were released as a corpus by *Reuters, Ltd.* in 2000 for research purposes [RSW02]. RCV1-v2 is a modified version of the corpus which was extensively documented by [LYRL04] after they corrected various inconsistencies in the original corpus. We use the same chronological train/test split of 23149/781265 documents that was used in [LYRL04]. Unlike other large corpora, note that the training set is kept relatively small. There are several different types of codes that RCV1-v2 documents are labeled with (regions, industries, topics). We use only topics and restrict our attention to the 101 topics that have at least one labeled document in the training set. These topics are hierarchically organized and vary in their granularity. Topics dealing with financial aspects tend to be very fine grained while those such as politics are coarse grained. Further details of the semantics are available in [LYRL04].

To ensure comparability with other published results, we have started with the version of the documents provided in Appendix 12 of [LYRL04]. These documents have already been stemmed, stopworded, and tokenized. We treat these tokenized documents as if they were the original documents.

The frequencies of the 101 classes are the same as those reported in [LYRL04]. The class proportions in the training set vary from 2 documents (0.0086%) to 10786 documents (46.59%) with a median of 233 documents (1.007%). In the testing set, they range from 38 documents (0.0049%) to 370541 documents (47.43%) with a median of 8266 documents (1.058%). Approximately 21K words appear in at least 3 training documents.

6.3 Base Classifiers

We have selected five classifiers traditionally used for text classification for examination: decision trees, linear SVMs, naïve Bayes, a unigram classifier, and a kNN classifier. We have chosen these algorithms not only because they are known to perform well but also because they are different types of classifiers along several different dimensions. The SVM, unigram, and naïve Bayes algorithms we investigate produce a linear decision boundary where the kNN and Decision Tree classifiers are non-linear classifiers. In contrast, while the unigram and naïve Bayes algorithms we employ are generative classifiers and similar to each other, they differ from the SVM, kNN, and Decision Tree which are discriminative classifiers.

6.3.1 Decision Trees

For the decision-tree implementation, we have employed the WinMine[5] decision networks toolkit and refer to this as *Dnet* below [Mic01]. Dnet builds decision trees using a Bayesian machine learning algorithm [CHM97, HCM$^+$00]. While this toolkit is targeted primarily at building models to provide probability estimates, we found that Dnet models usually perform acceptably for the goal of minimizing error rate. However, we found that the performance of Dnet with regard to other measures is sometimes poor.

The probability that the model predicts is a Laplace correction of the empirical probability at a leaf node (see Eq. 6.10). As noted in [PD03], using a Laplace correction is often critical when a high quality probability-based ranking is desired. We have not observed any of the other problems noted in [PD03] with regard to obtaining probability estimates. This is likely because the Dnet algorithm attempts to optimize a likelihood based criterion instead of accuracy. As pointed out by Provost & Domingos [PD03], optimizing accuracy can lead to early stopping or overly aggressive pruning that harms probability estimates. The interested reader applying an alternate decision tree algorithm should consult [PD03] for more information.

[5]We thank Max Chickering and Robert Rounthwaite for their special support of the WinMine toolkit.

6.3.2 SVMs

For SVMs, we have used a linear kernel as implemented in the SVM^{light}package v6.01. Unless otherwise noted, we used a linear kernel and default settings for all other parameters. A more thorough discussion of SVMs is included in Section 5.1.5.

6.3.3 Naïve Bayes (multivariate Bernoulli)

The *naï ve Bayes* classifier has also been referred to as a multivariate Bernoulli model [MN98]. As has been noted elsewhere, the probability estimates used by the classifier must be smoothed to avoid zero probability estimates [Mit97].

The simplest smoothing method which has been called any of “add-one”, Laplace correction, or standard Laplace correction [Sim95, KBS97] is:

$$\hat{P}(C = c \mid z) = \frac{n_c + 1}{n + |C|}. \tag{6.10}$$

Here n is the number of observations for which condition z holds and n_c is the number of observations for which both z holds and variable C takes value c.

Kohavi, Becker, and Sommerfeld [KBS97] characterize as Laplace approaches any method which uses:

$$\hat{P}(C = c \mid z) = \frac{n_c + f}{n + |C|f} \tag{6.11}$$

where $f > 0$ is a parameter. They also introduce the Laplace m-estimate which sets $f = \frac{1}{N}$ where N is the total number of observations and not just those matching condition z.[6] Thus the effect of smoothing decreases as the total number of observations goes to infinity.

Finally, a more general method sometimes referred to as a Bayesian estimate or Bayesian m-estimate [Mit97] is:

$$\hat{P}(C = c \mid z) = \frac{n_c + mp_c}{n + m} \tag{6.12}$$

where p_c is a user-specified prior on class c (where $\sum_c p_c = 1$) and m is the “effective sample size” or the weight the prior will carry as measured in number of observations. The Laplace m-estimate is a special case of the Bayesian m-estimate where $p = \frac{1}{|C|}$ and $m = \frac{|C|}{N}$. In many applications, z is the null condition and $N = n$.

In using the multivariate Bernoulli classifier, the conditional word probabilities use a Bayesian estimate with an effective sample size of 1 and the empirical word frequency (*i.e.*

[6]They use m instead of N to refer to the total number of observations. We reserve m to refer to effective sample size as in the Bayesian estimate.

$p = \tilde{P}(w)$ and $m = 1$). The class estimates are smoothed using a Laplace m-estimate. Thus, we have:

$$\hat{P}(w = \{\text{present}, \text{absent}\} \mid c) = \frac{n_{c,w} + \frac{n_w}{N}}{n_c + 1} \tag{6.13}$$

$$\hat{P}(c) = \frac{n_c + \frac{1}{N}}{N + \frac{|C|}{N}}. \tag{6.14}$$

n_w is the number of documents with the word present / absent, $n_{c,w}$ is the number of documents in class c with the word present / absent, n_c is the number of documents in class c, and N is the total number of documents. Words that did not occur in the training set are ignored.

6.3.4 Unigram (multinomial naïve Bayes)

The *unigram* classifier uses probability estimates from a unigram language model [MN98]. This classifier has also been referred to as a multinomial naïve Bayes classifier. Probability estimates are smoothed in a similar fashion to smoothing in the naïve Bayes classifier. The resulting estimates are:

$$\hat{P}(w \mid c) = \frac{n_{c,w} + \frac{n_w}{N}}{n_c + 1} \tag{6.15}$$

$$\hat{P}(c) = \frac{n_c + \frac{1}{N}}{N + \frac{|C|}{N}}. \tag{6.16}$$

n_w is the number of times word w occurred, $n_{c,w}$ is the number of times word w occurred in documents in class c, n_c is the total number of word occurrences in class c, and N is the total number of word occurrences. As a reminder to the reader, the multinomial model counts repetitions of the same word in the document. Words that did not occur in the training set are ignored.

6.3.5 k-Nearest Neighbor

We employ a standard variant of the k-nearest neighbor algorithm used in text classification, kNN with s-cut score thresholding [Yan99]. We use a tfidf-weighting of the terms with a distance-weighted vote of the neighbors to compute the score before thresholding it. The score used for a class y for example $\mathbf{x}$ is:

$$s_{\mathbf{x}}(y) = \sum_{\mathbf{n} \in kNN(\mathbf{x}) | c(\mathbf{n}) = y} \cos(\mathbf{x}, \mathbf{n}) \quad - \sum_{\mathbf{n} \in kNN(\mathbf{x}) | c(\mathbf{n}) \neq y} \cos(\mathbf{x}, \mathbf{n}). \tag{6.17}$$

In order to choose the threshold value for s, we perform cross-validation over the training set. Unless noted elsewhere, the value of k is set to be $2(\lceil \log_2 N \rceil + 1)$ where N is the number of training points. This rule for choosing k is theoretically motivated by results which show such a rule converges to the optimal classifier as the number of training points increases [DGL96]. In practice, we have also found it to be a computational convenience that frequently leads to comparable results with numerically optimizing k via a cross-validation procedure.

6.3.6 Classifi er Outputs

Finally, we must address the question of what kind of outputs we expect from the classifiers. Again, recalling our example of class-conditionally independent classifiers from Section 1.2.1, if our combination procedure simply uses the class prediction of the base classifiers, then we often cannot improve or can only improve slightly even when the classifiers are based on distinct information sources. Additionally, experiments with stacking [Wol92, TW99] have shown using some kind of partially-ordered score leads to significantly better results. Therefore, we desire that the classifier outputs a confidence of some kind such that greater values of the score implies a greater confidence that the example belongs to the positive class.

However, working with arbitrary scores can be problematic since the inputs can be on vastly different scales. One alternative is to actually require probabilities. We can either produce probabilities from a score as described in Chapter 3 or use techniques as described in [LZ05] that produce probability estimates from models which only predict class membership.

Consider our idealized example again. If we are given completely correct posterior estimates by class-conditionally independent classifiers (Figure 4.3 and Section 1.2.1), the optimal combination is obtained by multiplying the probabilities and renormalizing. Since multiplicative models are sometimes more difficult to work with, we will prefer additive models and therefore work with the log-odds from the classifiers. In fact, because we will use a bias term in our additive models, we often can work with a score that's "similar to log-odds" but not scaled correctly — allowing our bias term to implicitly correct the misestimation of the base models.

6.4 Chapter Summary

This chapter presented an overview of the various performance measures used to compare the effectiveness of a combined model with that of the base classifiers or alternative models. In the remainder of the dissertation, we will focus on F1, linear utilitiy functions, and area under the ROC curve when evaluating the classification methods. In addition, this chapter presented the characteristics of the key datasets to be used in experimentation and the implementation details of the classifiers.

Chapter 7

Combining Classifiers using Reliability Indicators

From Chapter 3, we know that a classifier can improve a combination of classifiers as long as the class is not independent of the output of that classifier given the output of the other classifiers. This is the motivation behind stacking classifiers. However, we would like to account for more locality[1] than stacking allows — which can only use functions of the classifier outputs to set non-constant weights on the classifiers. Most other methods (*e.g.*, local cascade generalization which additionally uses all of the original features as well) are not suited for high-dimensional problems such as text. This part takes the insights gained from our work on calibration and introduces a new combination model for text classification that uses reliability indicator variables to create a low-dimensional abstraction of the properties likely to influence the local reliability, dependence, and variance of the classifier outputs. Since we then build a classifier using this representation, we are implicitly using the joint distribution over this space to combine the classifiers.

Our work is distinguished from earlier combination approaches for text classification by (1) the use of expressive probabilistic dependency models to combine lower-level classifiers, leveraging special signaling variables, referred to as reliability indicators, and (2) a focus on measures of classification performance rather than the more common consideration of ranking.

[1]See Chapter 4 for more on locality.

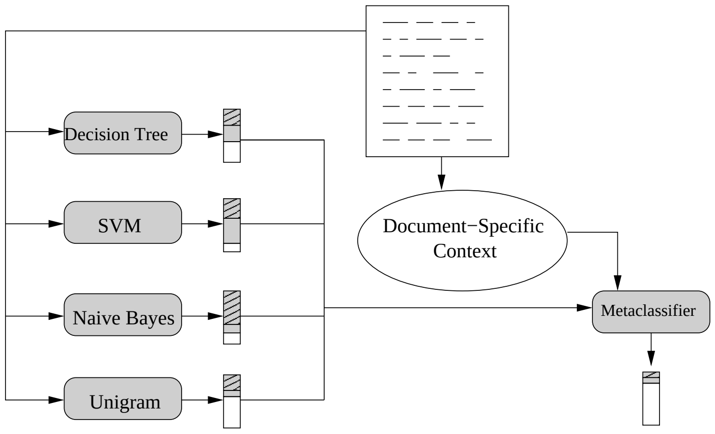


Figure 7.1: Schematic characterization of reliability-indicator methodology. The methodology formalizes the intuition shown here that document-specifi c context can be used to improve the performance of a set of base classifi ers. The output of the classifi ers is a graphical representation of a distribution over possible class labels.

7.1 Introduction

Previous approaches to classifier combination have typically limited the information considered at the metalevel to the output of the classifiers [TW99] and/or the original feature space [Gam98a]. Since a classifier rarely is the best choice across a whole domain, an intuitive alternative is to identify the document-specific context that differentiates between regions where a base classifier has higher or lower reliability.

Returning to the example from Chapter 1, Figure 7.1 shows an example using four base classifiers: decision tree, SVM, naïve Bayes, and unigram. When given a test document as input, each of the four base classifiers outputs a probability distribution over possible class labels (depicted graphically as a histogram in the figure). The metaclassifier uses this information along with document context (to be described in more detail) to produce a final classification of the document.

We address the challenge of learning about the reliability of different classifiers in different neighborhoods of the classification domain by introducing variables referred to as *reliability indicators* which represent the analytic "context" of a specific document. A reliability indicator is an evidential distinction with states that are linked probabilistically to regions of a classification problem where a classifier performs relatively strongly or poorly.

The reliability-indicator methodology was introduced by Toyama and Horvitz [TH00] and applied initially to the task of combining, in a probabilistically coherent manner, several distinct machine-vision analyses in a system for tracking the head and pose of computer users. The researchers found that different visual processing modalities had distinct context-sensitive reliabilities that depended on dynamically changing details of lighting, color, and the overall configuration of the visual scene. The authors introduced reliability indicators to capture properties of the vision analyses, and of the scenes being analyzed, that provided probabilistic indications of the reliability of the output of each of the modalities. To learn probabilistic models for combining the multiple modalities, data were collected about ground truth, the observed states of indicator variables, and the outputs from the concurrent vision analyses. The data was used to construct a Bayesian network model with the ability to appropriately integrate the outputs from each of the visual modalities in real time, providing an overall higher-accuracy composite visual analysis.

The value of the indicator-variable methodology in machine vision stimulated us to explore the approach for representing and learning about reliability-dependent contexts in text classification problems. For the task of combining classifiers, we formulate and include sets of variables that hold promise as being related to the performance of the underlying classifiers. We consider the states of reliability indicators and the scores of classifiers directly and, thus, bypass the need to make ad hoc modifications to the base classifiers. This allows the metaclassifier to harness the reliability variables if they contain useful discriminatory information and, if they do not, to fall back in a graceful manner to using the output of the base classifiers.

As an example, consider three types of documents where: (1) the words in the document are either uninformative or strongly associated with one class; (2) the words in the document are weakly associated with several disjoint classes; or (3) the words in the document are strongly associated with several disjoint classes. Classifiers (*e.g.*, a unigram model) will sometimes demonstrate different patterns of error on these different document types. If we can characterize a document as belonging to one of these model-specific failure types, then we can assign the appropriate weight to the classifier's output for this kind of document. We have pursued the formulation of reliability indicators that capture different association patterns among words in documents and the structure of classes under consideration. We seek indicator variables that would allow us to learn context-sensitive reliabilities of classifiers, conditioned on the observed states of the variable in different settings.

To highlight the approach with a concrete example, Figure 7.2 shows a portion of the type of combination function we can capture with the reliability-indicator methodology. The nodes on different branches of a decision tree include the values output by base classifiers, as well as the values of reliability indicators for the document being classified. The decision tree provides a probabilistic, context-sensitive combination rule indicated by the particular relevant branching of values of classifier scores and indicator variables. In this

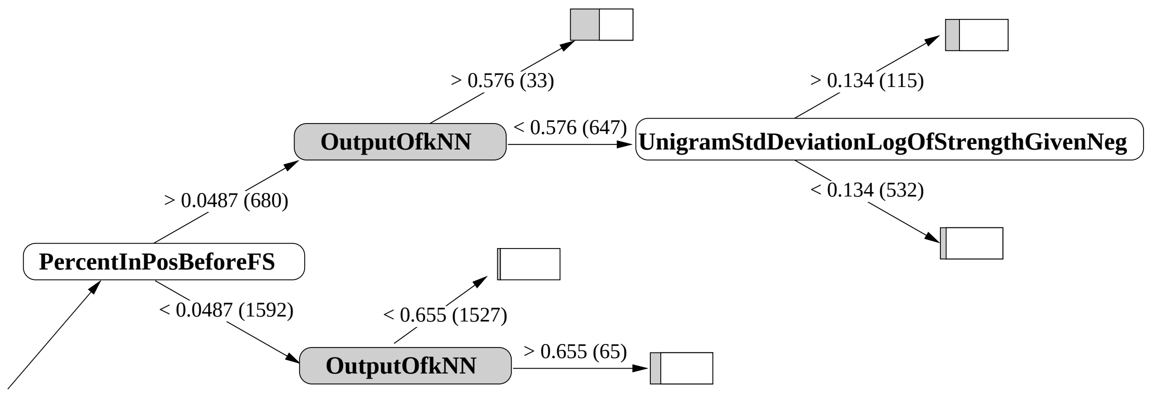


Figure 7.2: Portion of decision tree, learned by *STRIVE-D (norm)* for the *Business & Finance* class in the MSN Web Directory corpus, representing a combination policy at the metalevel that considers scores output by classifi ers (dark nodes) and values of indicator variables (lighter nodes). Higher in the same path, the decision tree also makes use of *OutputOfUnigram* and *OutputOfSVMLight*, as well as other indicator variables.

case, the portion of the tree displayed shows a classifier-combination function that considers thresholds on scores provided by a kNN classifier (*OutputOfkNN*) in conjunction with the context established by reliability-indicator variables (*PercentInPosBeforeFS* and *UnigramStdDeviationLogOfStrengthGivenNeg*) to make a final decision about a classification. Higher in the path to these nodes, the decision tree has also made use of the outputs of an SVM classifier and a unigram classifier as well as other indicator variables. The annotations in the figure show the threshold tests that are being performed, the number of examples in the training set that satisfy the test, and a graphical representation of the probability distribution at the leaves. The likelihood of class membership is indicated by the length of the bars at the leaves of the tree.

The variable *UnigramStdDeviationLogOfStrengthGivenNeg* represents the variance of unigram class-conditional weights for the negative class for words present in the current document. The intuition behind the formulation of this reliability-indicator variable is that examples are more likely to be negative when there is low variance in weights. The variable *PercentInPosBeforeFS* is the percentage of words in the document that occurred in a positive training example before feature selection. This can indicate an example is likely to be positive and can help mitigate overly aggressive feature selection by considering the state before feature selection. Notice that examples with the highest kNN scores (lowest leaf) are actually given a lower posterior probability of membership than those with lower kNN scores (highest leaf) because of the context established by the indicators and other classifier outputs. Chapter 5 gives further details about the motivation, derivation, and computation of reliability indicators used in these experiments.

The indicator variables used in our studies represent an attempt to formulate states that capture influential contexts. We constructed variables to represent a variety of contexts that held promise as being predictive of accuracy. These include such variables as the number of features present in a document before and after feature selection, the distribution of features across the positive vs. negative classes, and the mean and variance of classifier-specific weights.

We can broadly group reliability-indicator variables into one of four types, including variables that measure (1) the amount of information present in the original document, (2) the information loss or mismatch between the representation used by a classifier and the original document, (3) the sensitivity of the decision to evidence shift, and (4) some basic voting statistics.

DocumentLength is an example of a reliability-indicator variable of type 1. The performance of classifiers is sometimes correlated with document length, because longer documents give more information to use in making a classification. *DocumentLength* can also be informative because some classifiers will perform poorly over longer documents as they do not model the influence of document length on classification performance (*e.g.*, they double count evidence and longer documents are more likely to deviate from a correct determination).

PercentRemoved serves as an example of type 2. This variable represents the percent of features removed in the process of feature selection. If most of the document was not represented by the feature set employed by a classifier, then some classifiers may be unreliable. Other classifiers (*e.g.*, decision trees that model missing attributes) may continue to be reliable. When the base classifiers are allowed to use different representations, type 2 features can play an even more important role.

An example of type 3 is the *UnigramStdDeviation* variable. In a binary class problem, the weight each word contributes to the unigram model's decision is $\log \frac{P(w|c)}{P(w|\neg c)}$. This is the standard deviation of the weight each word contributes over the words in the document. Low variance means the decision of the classifier is unlikely to change with a small change in the document content; high variance increases the chances that the decision would change with only a small change in the document.

Finally, *NumVotingForClass* or *PercentAgreement* are examples of type 4 reliability indicators. These simple voting statistics improve the metaclassifier search space since the metaclassifier is given the base classifier decisions as input as well. For a two-class case the *PercentAgreement* variable may provide little extra information but for greater number of classes it can be used to determine if the base classifiers have fractured their votes among a small number of classes or across a wide array. At the end of this chapter, we discuss which reliability indicators were most useful in the final combination scheme.

Beyond the key difference in the semantics of their usage, reliability-indicator variables differ qualitatively from variables representing the output of classifiers in several ways. For one, we do not assume that the reliability indicators have some threshold point that classifies the examples better than random. We also do not assume that classification confidence shows monotonicity trends as in classifiers.

7.1.1 STRIVE: Metaclassifi er with Reliability Indicators

We refer to our classifier combination learning and inference framework as *STRIVE* for **St**acked **R**eliability **I**ndicator **V**ariable **E**nsemble. We select this name because the approach can be viewed as essentially extending the stacking framework by introducing reliability indicators at the metalevel. The *STRIVE* architecture is depicted graphically in Figure 7.4.

Our methodology maps the original classification task into a new learning problem. In the original learning problem (Figure 7.3), the base classifiers simply predict the class from a word-based representation of the document, or more generally, each base classifier outputs a distribution (possibly unnormalized) over class labels. *STRIVE* adds another layer of learning to the base problem. A set of reliability-indicator functions use the words in the document and the classifier outputs to generate the reliability indicator values, r_i, for a particular document. This process can be viewed as yielding a new representation of the document that consists of the values of the reliability indicators, as well as the outputs of the base classifiers. The metaclassifier uses this new representation for learning and classification. This enables the metaclassifier to employ a model that uses the output of the base classifiers as well as the context established by the reliability indicators to make a final classification.

We require the outputs of the base classifiers to train the metaclassifier. Thus, we perform cross-validation over the training data and use the resulting base classifier predictions, obtained when an example serves as a validation item, as training inputs for the metaclassifier. We note that, in the case where the set of reliability indicators are restricted to be the identity function over the original data, the resulting scheme can be viewed as a variant of cascade generalization [Gam98a].

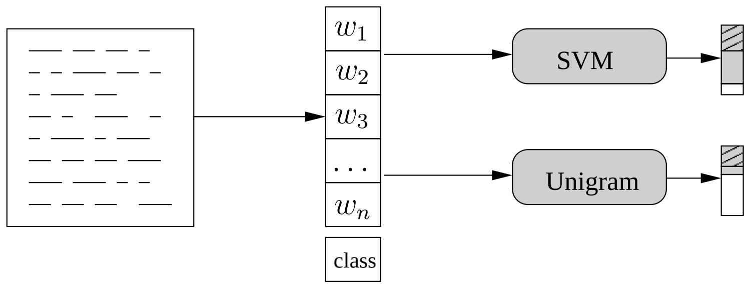


Figure 7.3: Typical application of a classifi er to a text problem. In traditional text classifi cation, a word-based representation of a document is extracted (along with the class label during the learning phase), and the classifi ers (here an SVM and Unigram classifi er) learn to output scores for the possible class labels. The shaded boxes represent a distribution over class labels.

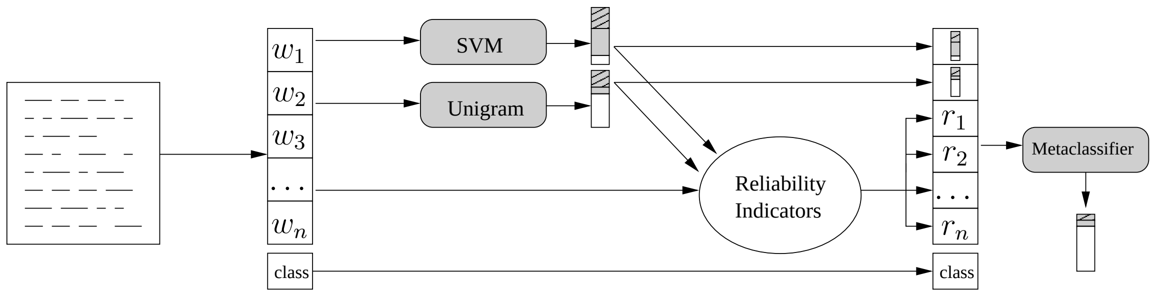


Figure 7.4: Architecture of *STRIVE*. In *STRIVE*, an additional layer of learning is added where the metaclassifi er can use the context established by the reliability indicators and the output of the base classifi ers to make an improved decision. The reliability indicators are functions of the document and/or the output of the base classifi ers.

7.2 Experimental Analysis

We performed a large number of experiments to test the value of probabilistic classifier combination with reliability-indicator variables. For the experiments below, we used only the top 1000 words with highest mutual information for the MSN Web Directory and TREC-AP corpus and top 300 words for Reuters for all base classifiers except the kNN classifier. Note that performance of the base classifiers using feature selection is generally as good if not better than performance using all of the features. Since the kNN classifier is computationally expensive, we desired to use the same feature representation across binary classification tasks within a corpus. Once neighbors are retrieved, the kNN classifier can make all class decisions quickly. As is commonly done (e.g. [LYRL04]), for each word we assigned a score of the max of the mutual information scores across binary tasks. The top features were then taken across these max scores. Since the same feature set was being used for all classes within a corpus, we used $3\times$ the number of features — 3000 words for MSN Web and TREC-AP and 900 for Reuters. For the reliability indicators that compare representations before and after feature selection, we only added instantiations for the non-kNN representation. The corpora are described in further detail in Section 6.2. We now describe the methodology and results.

Base Classifiers

In an attempt to isolate the benefits gained from the probabilistic combination of classifiers with reliability indicators, we worked to keep the representations for the base classifiers in our experiments nearly identical. We would expect that varying the representations (*i.e.*, using different feature-selection methods or document representations) would only improve the performance as this would likely decorrelate the performance of the base classifiers. One notable deviation is that a tfidf representation was used for the kNN classifier since that is standard in the text classification literature. As described in more detail in Section 6.3 we selected five classifiers as base classifiers: kNN, decision trees, linear SVMs, naïve Bayes, and a unigram classifier. We denote these below as *kNN*, *Dnet*, *SVM*, *naï ve Bayes*, and *Unigram*.

Basic Combination Methods

We perform experiments to explore a variety of classifier-combination methods and consider several different combination procedures. The first combination method is based on selecting one classifier for each binary class problem, based on the one that performed best for a validation set. We refer to this method as the *Best By Class* method.

Another combination method centers on taking a majority vote of the base classifiers. This approach is perhaps the most popular methodology used for the combination of text classifiers. Because we have five base classifiers, we do not have to address the issue of breaking ties.[2] We refer to this method as the *Majority* method.

Hierarchical Combination Methods

Stacking

Finally, we investigate several variants of the hierarchical models described earlier. As mentioned above, omitting the reliability-indicator variables transforms *STRIVE* to a stacking methodology [TW99, Wol92]. We refer to these classifiers below as *Stack-X* where *X* is replaced by the first letter of the classifier that is performing the metaclassification. Therefore, *Stack-D* uses a decision tree as the metaclassifier, and *Stack-S* uses a linear SVM as the metaclassifier. We note that *Stack-S* is also a weighted linear combination method since it is based on a linear SVM and uses only the classifier outputs. Therefore, demonstrating that *STRIVE* either outperforms *Stack-S* variants or is not statistically different is a key challenge of this dissertation.

We found it was difficult to learn the weights for an SVM when the inputs have vastly different scales. At times, it is not possible to identify good weights. To address the problem of handling inputs with greatly varying scales, we use an input normalization procedure: We normalize the inputs to the metaclassifiers to have zero mean and reduce the scale of the standard deviation.[3] In order to perform consistent comparisons, we perform the same

[2]When performing a majority vote, ties can be broken in a variety of ways (*e.g.*, breaking ties by always voting for *in class*). In earlier work [BDH02, BDH05], we experimented with several variants of these methods. The most successful broke ties by voting with the *Best By Class* classifier.

[3]Our original intention was to normalize the inputs to zero mean and unit standard deviation. However, we found slightly before press a line of code was commented out in our normalization code. The end result was that instead of using the normalization that produces zero mean and standard deviation, $f' = \frac{f-E[f]}{\sqrt{E[f^2]-E^2[f]}}$, we instead used $f' = \frac{f-E[f]}{\sqrt{E[f^2]}}$. The resulting f' has zero mean and a standard deviation of $\sigma_{f'} = \sqrt{1 - \frac{E^2[f]}{E[f^2]}}$. Since $E[f^2] \geq E^2[f]$ then $\sigma_{f'} \leq 1$, and $\sigma_{f'} = 1$ when $\frac{E^2[f]}{E[f^2]} = 0$. Thus, the scale of the feature variance has been reduced. Furthermore since the classifier outputs are similar to log-odds, then for stacking $E^2[f]$ and more importantly $\frac{E^2[f]}{E[f^2]}$ tend toward zero, and thus the resulting normaliztion differs little from standard normalization. We are primarily concerned with whether standard normalization would improve stacking further (if standard normalization hurts striving then we could use this normalization instead). Because the use of log-odds drives this normalization to behave similar to standard normalization for stacking and since we

alteration for the metaclassifiers using *Dnet*. Furthermore, as we will see in Chapter 8, this normalization is useful for additional reasons.

As might be expected the impact of normalization for decision-tree learners is relatively minimal and has both positive and negative influences. Because of this, we present only the version using normalized inputs in the main text, the non-normalized versions are available in a complete listing in Tables 7.11-7.13 at the end of the chapter. To denote the meta-classifiers whose inputs have been normalized in this manner, we append "*(norm)*" to their names.

STRIVE

Similar to the notation described above, we add a letter to *STRIVE* to denote the particular metaclassifier method being used. So, *STRIVE-D* is the *STRIVE* framework using *Dnet* as a metaclassifier. For comparison to the stacking methods, we evaluate *STRIVE-D* and *STRIVE-S*. Normalization, as above, is again noted by appending "*(norm)*" to the system names.

The experiments reported here use a total of 70 reliability indicators, including those specific examples given in Section 7.1. The full list of reliability indicators and the motivation for each is discussed in full in Chapter 5. These reliability indicators were formulated by hand as an attempt at representing potentially valuable contexts. Identifying new reliability indicators remains an open and challenging research problem both for this methodology and in general.

BestSelect Classifier

To study the effectiveness of the *STRIVE* methodology, we formulated a simple optimal combination approach as a point of reference. Such an upper bound can be useful as a benchmark in experiments with classifier combination procedures. This bound follows quite naturally when classifier combination is formulated as the process of *selecting* the best base classifier, on a per-example basis.

To classify a given document, if any of the classifiers correctly predict that document's class, the best combination would select any of the correct classifiers. Thus, such a classification combination errs only when all of the base classifiers are incorrect. We refer to

also compare to non-normalized metaclassifier inputs, the expected result is a negligible difference. Finally, in a small sample of experiments, this normalization in fact had negligible impact on the stacking methods compared to normalizing to unit standard deviation. Thus, while not the normalization we intended to use, this does not change the conclusions drawn from our experiments.

this classifier as the *BestSelect* classifier. If all of the base classifiers are better than random, the *BestSelect* is the theoretical upper-bound on performance when combining a set of classifiers in a *selection framework*.

We note that we are not using a pure selection approach, as our framework allows the possibility of choosing a class that none of the base classifiers predicted. In cases where the classifiers are not better than random (or are logically dependent), such an upper bound may be uninformatively loose. Even though we are not working in a pure *selection* framework, we found it is rarely the case the metaclassifier outputs a prediction which none of the base classifiers made. Therefore, we have employed this *BestSelect* bound to assist with understanding the performance of *STRIVE*.

7.2.1 Performance Measures

To compare the performance of the classification methods we look at a set of standard performance measures: the macro-averaged F1, micro-averaged F1, error, two linear utility functions — $C(10, 1)$ and $C(1, 10)$ — and area under the ROC curve. In addition, we computed and displayed a receiver-operating characteristic (ROC) curve, which represents the performance of a classifier under any linear utility function [PF01]. These measures are described in more detail in Section 6.1.

7.2.2 Experimental Methodology

As the categories under consideration in the experiments are not mutually exclusive, the classification was carried out by training n binary classifiers, where n is the number of classes. Decision thresholds for each classifier were set by optimizing them for each performance measure over the validation data. That is, a classifier could have different decision thresholds for each of the separate performance measures (and for each class). This ensures that the base classifiers are as competitive as possible across the various measures. For the micro performance measures, obtaining truly optimal performance requires optimizing all the thresholds in a corpus in conjunction; we have taken the more computationally efficient approach of using the macro-optimized thresholds (*i.e.*, the threshold for each class is set independently from the thresholds for the other classes).

To generate the data for training the metaclassifier, (*i.e.*, reliability indicators, classifier outputs, and class labels), we used five-fold cross-validation on the training data from each of the corpora. The data set obtained through this process was then used to train the metaclassifiers. Similar to the base classifiers, cross-validation over the meta-training set was used to set thresholds for each performance measure. Finally, the resulting metaclassifiers were applied to the separate testing data described above.

7.2.3 Results

	MacroF1	MicroF1	Error	C(1,10)	C(10,1)	ROC Area
Dnet	0.5477	0.5813	0.0584	0.3012	0.0772	0.8802
Unigram	0.5982	0.6116	0.0594	0.2589	0.0812	0.9003
naïve Bayes	0.5527	0.5619	0.0649	0.2853	0.0798	0.8915
SVM	$0.6727^{\boldsymbol{B}}$	$0.7016^{\mathbb{B}}$	0.0455	$0.2250^{\boldsymbol{B}}$	0.0794	0.9123
kNN	0.6480	0.6866	0.0464	0.2524	0.0733	0.8873
Best By Class	$0.6727^{\mathbb{D}}$	0.7016	$0.0452^{\boldsymbol{D}}$	0.2235	$0.0729^{\boldsymbol{D}}$	N/A
Majority	0.6643	0.6902	0.0479	0.2133^{BD}	0.0765	N/A
Stack-D (norm)	$0.6924^{\boldsymbol{BD}}$	$0.7233^{\mathbb{BD}}$	$0.0423^{\boldsymbol{BD}}$	$0.1950^{\boldsymbol{BD}}$	$0.0708^{\mathbb{D}}$	$0.9361^{\boldsymbol{B}\mathrm{S}}$
Stack-S (norm)	$0.6939^{\boldsymbol{BD}}$	$0.7250^{\mathbb{BD}}$	$0.0423^{\boldsymbol{BD}}$	$0.1971^{\boldsymbol{BD}}$	$0.0705^{\mathbb{D}}$	$0.9334^{\boldsymbol{B}}$
STRIVE-D (norm)	$0.6988^{\boldsymbol{BD}}$	$0.7327^{\mathbb{BD}}_{\mathbb{S}}$	$0.0413^{\boldsymbol{BD}}$	$0.1846^{\boldsymbol{BD}}_{\mathrm{S}}$	$0.0697^{\mathbb{D}}$	$\mathbf{0.9454}^{\boldsymbol{B}}_{\boldsymbol{SD}}$
STRIVE-S (norm)	$\mathbf{0.7173}^{\boldsymbol{BD}}_{\boldsymbol{S}\mathbb{R}}$	$\mathbf{0.7437}^{\mathbb{BD}}_{\mathbb{SR}}$	$\mathbf{0.0392}^{\boldsymbol{BD}}_{\boldsymbol{SR}}$	$\mathbf{0.1835}^{\boldsymbol{BD}}_{\mathrm{S}}$	**0.0682**	$0.9260^{\mathbb{B}}$
BestSelect	0.8719	0.8924	0.0223	0.0642	0.0565	N/A

Table 7.1: Performance on MSN Web Directory Corpus. The best performance (omitting the oracle *BestSelect*) in each column is given in bold. A notation of ‘B’, ‘D’, ‘S’, or ‘R’ indicates a method signifi cantly outperforms all (other) **B**ase classifi ers, **D**efault combiners, **S**tacking methods, or **R**eliability-indicator based Striving methods at the $p = 0.05$ level. A blackboard (hollow) font is used to indicate signifi cance for the macro-sign test and micro-sign test. A normal font indicates signifi cance for the macro t-test. For the macro-averages (*i.e.*, excluding micro F1) when both tests are signifi cant it is indicated with a bold, italicized font.

Tables 7.1, 7.2, and 7.3, present the main performance results over the three corpora. In terms of the various performance measures, better performance is indicated by larger F1 or ROC area values or by smaller $C(FP, FN)$ values. The best performance (ignoring *BestSelect*) in each column is given in bold.

To determine statistical significance for the macro-averaged measures, a one-sided macro sign test and two-sided macro *t*-test were performed [YL99]. For micro-F1, a two-sided micro sign test was performed [YL99]. Differences with a *p*-level above 0.05 were not considered statistically significant. The macro sign test uses the null hypothesis that the number of classes in which we improve versus the number in which we decrease is ran-

	MacroF1	MicroF1	Error	C(1,10)	C(10,1)	ROC Area
Dnet	0.7846	0.8541	0.0242	0.0799	0.0537	0.9804
Unigram	0.7645	0.8674	0.0234	0.0713	0.0476	0.9877
naïve Bayes	0.6574	0.7908	0.0320	0.1423	0.0527	0.9703
SVM	0.8545^{B}	$0.9122^{\mathbb{B}}$	0.0145^{B}	0.0499	0.0389	0.9893
kNN	0.8097	0.8963	0.0170	0.0737	0.0336	0.9803
Best By Class	0.8608^{D}	0.9149	0.0144	0.0496	0.0342	N/A
Majority	0.8498	0.9102	0.0155	0.0438	0.0437	N/A
Stack-D (norm)	0.8680	$0.9197^{\mathbb{B}}$	0.0136	0.0410	0.0366	0.9912
Stack-S (norm)	$\mathbf{0.8908}^{\mathrm{BD}}_{\mathrm{S}}$	$\mathbf{0.9307}^{\mathbb{BD}}_{\mathbb{S}}$	$0.0125^{\boldsymbol{BD}}$	$0.0372^{\boldsymbol{B}\mathbb{D}}$	$\mathbf{0.0331}_{\mathrm{S}}$	$\mathbf{0.9956}^{\boldsymbol{B}}_{\boldsymbol{S}}$
STRIVE-D (norm)	0.8555	0.9172	0.0144	0.0488	0.0364	0.9913
STRIVE-S (norm)	$0.8835^{\mathrm{BD}}_{\boldsymbol{R}}$	$0.9287^{\mathbb{BD}}_{\mathbb{R}}$	$\mathbf{0.0121}^{\mathrm{BD}}_{\boldsymbol{R}}$	$\mathbf{0.0352}^{\boldsymbol{B}\mathbb{D}}$	0.0343	$0.9948^{\boldsymbol{B}}_{\boldsymbol{R}}$
BestSelect	0.9611	0.9789	0.0036	0.0073	0.0173	N/A

Table 7.2: Performance on Reuters Corpus. The best performance (omitting the oracle *BestSelect*) in each column is given in bold. A notation of 'B', 'D', 'S', or 'R' indicates a method significantly outperforms all (other) **B**ase classifiers, **D**efault combiners, **S**tacking methods, or **R**eliability-indicator based Striving methods at the $p = 0.05$ level. A blackboard (hollow) font is used to indicate significance for the macro-sign test and micro-sign test. A normal font indicates significance for the macro t-test. For the macro-averages (*i.e.*, excluding micro F1) when both tests are significant it is indicated with a bold, italicized font.

domly distributed binomially with probability one half. The macro t-test compares whether the average difference across classes can be explained by the variance of drawing two samples from a t-distribution. Thus, the macro-sign test can detect when we are very likely to improve in an extremely high proportion of classes while the macro t-test detects when the amount of difference cannot be explained by random variation. Viewing the results of both tests generally indicate whether we can always expect to gain and how much.

The micro sign test is similar to the macro sign test but compares the decisions at an example-level instead of the performance at a class-level. The null hypothesis is that over the examples where two methods make different classification decisions the distribution of right/wrong can be explained by a binomial distribution with probability one half. Thus, the micro sign test indicates whether a method makes significantly different decisions than another method cumulatively across all classes.

A notation of 'B', 'D', 'S', or 'R' indicates a method significantly outperforms all (other) **B**ase classifiers, **D**efault combiners, **S**tacking methods, or **R**eliability-indicator based

	MacroF1	MicroF1	Error	C(1,10)	C(10,1)	ROC Area
Dnet	0.6007	0.5706	0.0064	0.0346	0.0081	0.9767
Unigram	0.6001	0.5695	0.0064	0.0347	0.0079	0.9819
naïve Bayes	0.5676	0.5349	0.0065	0.0455	0.0078	0.9755
SVM	$0.7361^{\boldsymbol{B}}$	$0.6926^{\mathbb{B}}$	$0.0049^{\boldsymbol{B}}$	$0.0282^{\mathbb{B}}$	0.0077	0.9715
kNN	0.6793	0.6533	0.0053	0.0371	0.0074	0.9238
Best By Class	$0.7356^{\boldsymbol{D}}$	$0.6925^{\mathbb{D}}$	$0.0049^{\boldsymbol{D}}$	0.0302	0.0073	N/A
Majority	0.7031	0.6534	0.0056	0.0307	0.0075	N/A
Stack-D (norm)	0.7331	$0.7007^{\mathbb{BD}}$	0.0050	**0.0251**	0.0073	**0.9886**
Stack-S (norm)	$0.7486^{\boldsymbol{BD}}$	$0.7011^{\mathbb{BD}}$	$0.0048^{\mathrm{B}\boldsymbol{D}}_{\boldsymbol{S}}$	$0.0263^{\mathbb{BD}}$	0.0072	$0.9834^{\mathbb{B}}$
STRIVE-D (norm)	0.7246	$0.6991^{\mathbb{BD}}$	0.0051	0.0268	0.0073	0.9870
STRIVE-S (norm)	$\mathbf{0.7532}^{\boldsymbol{BD}}_{\boldsymbol{R}}$	$\mathbf{0.7148}^{\mathbb{BD}}_{\mathbb{SR}}$	$\mathbf{0.0047}^{\boldsymbol{BD}}_{\mathrm{S}\boldsymbol{R}}$	$0.0277^{\mathbb{D}}$	**0.0071**	0.9771
BestSelect	0.8986	0.8356	0.0031	0.0133	0.0058	N/A

Table 7.3: Performance on TREC-AP Corpus. The best performance (omitting the oracle *BestSelect*) in each column is given in bold. A notation of 'B', 'D', 'S', or 'R' indicates a method significantly outperforms all (other) **B**ase classifiers, **D**efault combiners, **S**tacking methods, or **R**eliability-indicator based Striving methods at the $p = 0.05$ level. A blackboard (hollow) font is used to indicate significance for the macro-sign test and micro-sign test. A normal font indicates significance for the macro t-test. For the macro-averages (*i.e.*, excluding micro F1) when both tests are significant it is indicated with a bold, italicized font.

Striving methods at the $p = 0.05$ level. A blackboard (hollow) font is used to indicate significance for the macro-sign test and micro-sign test. A normal font indicates significance for the macro t-test. For the macro-averages (*i.e.*, excluding micro F1) when both tests are significant it is indicated with a bold, italicized font. When a method is part of a group, the letter of that group indicates the method beats all *other* methods in that group significant. For example, a $\mathbb{B}$ attached to the SVM classifier's macro F1 would indicate that the SVM classifier significantly outperforms on macro F1 all other base classifiers according to the macro sign test.

7.2.4 Discussion

First, we note that the base classifiers are competitive and consistent with the previously reported results over these corpora [ZO01, DC00, DPHS98, Joa98, Lew95, LG94, MN98].[4] Furthermore, the fact that the linear SVM tends to be the best base classifier is consistent with the literature [DPHS98, Joa98, YL99].

MSN Web Directory

Examining the main results for the MSN Web Directory corpus in Table 7.1 highlights several points. First, the basic combiners have only one significant win over the base classifiers, C(1,10) for the *Majority* vote approach. The results directly support the idea that the performance of a very good learner (*SVM*) tends to be diminished when combined via a majority vote scheme with weak learners; in addition, the win most likely results from the fact that the base learners (other than *SVM*) have a tendency to predict positively for a class. When false negatives are weighed more heavily, the shift toward predicting positive helps reduce the number of false negatives.

Both variants of Stacking and Striving often outperform the base classifiers, and with few exceptions, *Stack-S (norm)* and *STRIVE-S (norm)* outperform the base classifiers on nearly all the performance measures, often significantly. In fact this remains true in all of the corpora. The only consistent exception is the $C(10, 1)$ performance measure. This measure places a much higher penalty on false positives; therefore methods are pushed toward achieving correct negatives. Given the small number of positives in text classification corpora, achieving any gains is very challenging.

Both *STRIVE-D (norm)* and *STRIVE-S (norm)* show advantages that are robust across a variety of performance measures. Each shows a consistent improvement across a variety of performance measures over the state-of-the-art *SVM* classifier. For the thresholds optimized for error, *Stack-S (norm)* achieves a relative reduction in error over the *SVM* of 7% while *STRIVE-S (norm)* further improves to achieve a relative reduction in error of 14% over the *SVM — twice* the improvement that stacking yields (see Figure 7.7). When compared to the best theoretical performance that could be achieved by a per-example selection model using these base classifiers (as established by the *BestSelect* model), the error reduction

[4]While the results reported for Reuters are not directly comparable to those reported by Yang & Liu [YL99] as these investigators report results over all 90 classes and do not give a breakdown for the ten most frequent categories, others [ZO01, DPHS98, Joa98, MN98, Pla99] provide published baselines over the ten largest classes.

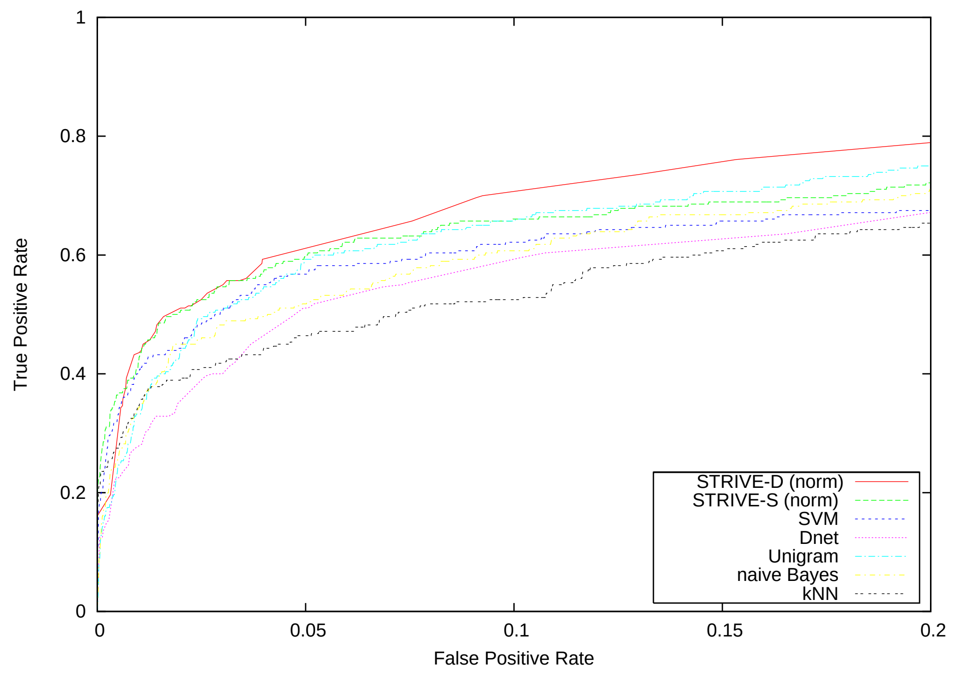


Figure 7.5: The ROC curve for the *Home & Family* class in the MSN Web Directory corpus from $[0, 0.2]$.

provided by the *STRIVE* combination methods is an even greater portion of the total possible reduction.

As can be inferred from the sign tests, these results are very consistent across classes. For example, by the ROC area measure of performance, *STRIVE-D (norm)* beats the base classifiers on 13/13 classes. The notable exception is the performance of *STRIVE-S (norm)* on ROC area; graphical inspection of the ROC curves suggests this result arises because the *STRIVE-S (norm)* places emphasis on the classifier that performs well in the early part of the curve.

Often, there is a crossover in the ROC curve between two of the base classifiers further out on the false-positives axis. Most utility measures in practice correspond to the early part of the curve (this depends on the particular features of the given curve). The *SVM* metaclassifier sometimes seems to lock onto the classifier that is strong in the early portion of the curve and loses out on the later part of the curve. Since the latter portion of the curve rarely matters, one could consider using an abbreviated version of curve area to assess systems. In tables 7.11-7.13, we present an additional measure of ROC area that only measures the area under the curve for the portion of the x-axis from $[0, 0.1]$. By this measure, it is possible to see that *STRIVE-S (norm)*'s performance is markedly better in the early part of the ROC curve.

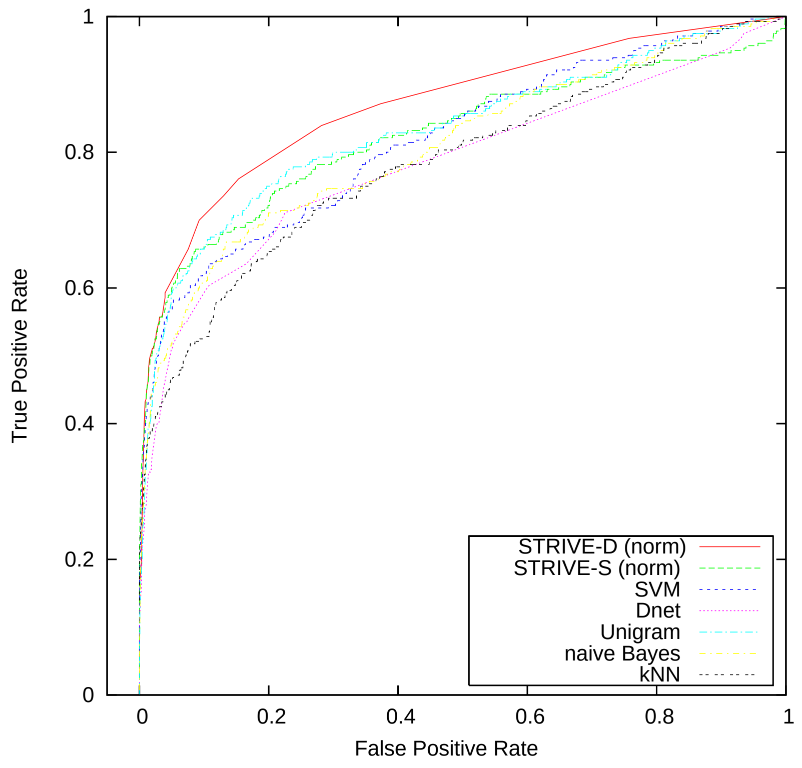


Figure 7.6: The full ROC curve for the *Home & Family* class in the MSN Web Directory corpus.

In Figures 7.6 and 7.5, we can see that the two *STRIVE* variants dominate the five base classifiers over much of the ROC space. In fact, *STRIVE-D* dominates (*i.e.*, its quality is greater than any other curve at every point) most of the MSN Web Directory corpus. We also can see (note the truncated scale) the base classifiers catching up with *STRIVE-S (norm)* on the right side of the curve. The base classifiers, in fact, do surpass *STRIVE-S (norm)* at points. As a result, *STRIVE-D* may be a more appropriate choice if the utility function penalizes false negatives significantly more heavily than false positives. However, as shown by its performance on the $C(1, 10)$ measure, *STRIVE-S (norm)* retains some robustness in this dimension due to its superior performance early in the curve.

In some cases, we can develop an understanding of why the decision tree is more appropriate for tracking crossovers. In the case portrayed in Figure 7.2, it appears that the tree establishes two separate score regions for *kNN* where the reliability indicators give further information about how to classify an example. Since a linear SVM is a weighted sum over the inputs, it cannot represent crossovers that are dependent on breaking a single variable into multiple regions (such as this one); it has to use the information present in other variables to try to distinguish these regions. Higher-order polynomial kernels are one way to allow an SVM to represent this type of information.

We attempted to do so, but the SVM had difficulty converging using a quadratic kernel. We then chose an alternate localized kernel. In Table 7.4, STRIVE-S Local (norm) uses a local product kernel of $K(\mathbf{x}_i, \mathbf{x}_j) = [\langle \rho(\mathbf{x}_i), \rho(\mathbf{x}_j)\rangle + 1] [\langle \Pi(\mathbf{x}_i), \Pi(\mathbf{x}_j)\rangle + 1]$ where $\Pi(\mathbf{x})$ is the projection into the subspace consisting of the base classifier outputs and ρ is the identity function. The resulting kernel has a subset of the terms in a quadratic kernel (which would use ρ in both cases). While the results are advantageous for the MSN Web corpus, it seems to lead to overfitting on the other two corpora. Nor does it achieve exactly the merger we desire by the ROC area performance measure. In order to prevent overfitting, we then attempted to merge the models by introducing *STRIVE-S Local (norm)*. This model restricts ρ to the subset of features included in the model learned by *STRIVE-D (norm)*. While this leads to substantially less overfitting while retaining some positive gains in MSN Web, its overall change does not seem to make it preferable to *STRIVE-S (norm)*.

MSN WEB						
	MacroF1	MicroF1	Error	C(1,10)	C(10,1)	ROC Area
Stack-S (norm)	0.6939	0.7250	0.0423	0.1971	0.0705	**0.9334**
STRIVE-S (norm)	0.7173	0.7437	0.0392	0.1835	0.0682	0.9260
STRIVE-S Local (norm)	**0.7251**	**0.7530**	**0.0386**	**0.1810**	**0.0656**	0.9150
STRIVE-S LSelect (norm)	0.7197	0.7496	0.0388	0.1860	0.0664	0.9097

Reuters						
	MacroF1	MicroF1	Error	C(1,10)	C(10,1)	ROC Area
Stack-S (norm)	**0.8908**	**0.9307**	0.0125	0.0372	**0.0331**	**0.9956**
STRIVE-S (norm)	0.8835	0.9287	**0.0121**	**0.0352**	0.0343	0.9948
STRIVE-S Local (norm)	0.8751	0.9261	0.0125	0.0382	0.0344	0.9890
STRIVE-S LSelect (norm)	0.8825	0.9280	0.0128	0.0389	0.0341	0.9921

TREC-AP						
	MacroF1	MicroF1	Error	C(1,10)	C(10,1)	ROC Area
Stack-S (norm)	0.7486	0.7011	0.0048	0.0263	0.0072	**0.9834**
STRIVE-S (norm)	**0.7532**	**0.7148**	**0.0047**	0.0277	**0.0071**	0.9771
STRIVE-S Local (norm)	0.7439	0.7084	0.0048	**0.0231**	**0.0071**	0.9725
STRIVE-S LSelect (norm)	0.7507	0.7104	**0.0047**	0.0273	**0.0071**	0.9731

Table 7.4: *STRIVE-S Local (norm)* uses a local product kernel of $K(\mathbf{x}_i, \mathbf{x}_j) = [\langle \rho(\mathbf{x}_i), \rho(\mathbf{x}_j)\rangle + 1] [\langle \Pi(\mathbf{x}_i), \Pi(\mathbf{x}_j)\rangle + 1]$ where $\Pi(\mathbf{x})$ is the projection into the subspace consisting of the base classifi er outputs and ρ is the identity function. The resulting kernel has a subset of the terms in a quadratic kernel. *STRIVE-S Local (norm)* restricts ρ to the subset of features included in *STRIVE-D (norm)* leads to substantially less overfi tting and positive gains in one corpus.

Note that in earlier work [BDH05], *Stack-S (norm)* did not consistently outperform the base classifiers. Two changes were key to achieving this change. First, we added another strong base classifier, *kNN*. Secondly, in the earlier work, the outputs of some classifiers were probabilities while others were log-odds. In the current form, all outputs are either log-odds or scores that demonstrate behavior similar to log-odds. As discussed in Chapter 4, a linear combination of log-odds can capture many desirable types of combination and recalibration interactions that are not attainable through a a linear combination of probabilities. However, even with the increase in *Stack-S (norm)*'s performance *STRIVE-S (norm)* improves enough beyond it to be statistically significant. In fact, across most measures *STRIVE-S (norm)* is now clearly the superior choice to *STRIVE-D (norm)*.

Reuters and TREC-AP

The results for Reuters and TREC-AP in Tables 7.2 and 7.3 are consistent with the above analysis. We note that the level of improvement over stacking (particularly in Reuters) is less pronounced for these corpora.

The decision tree meta-models show less consistency than the SVM meta-models. This is due in part to the nature of the models. While the SVM model cannot threshold regions of the space well, it can smoothly combine the various model outputs, whereas the decision tree meta-model fractures the data as it does so and cannot place weights on the outputs. This further emphasizes the need for a meta-model that provides the advantages of both models.

In both of these corpora *STRIVE-S (norm)* continues to outperform the base classifiers. In the TREC-AP corpus *STRIVE-S (norm)* also outperforms the stacking methods significantly on several measures. In the Reuters corpus, however, *STRIVE-S (norm)* is slightly outperformed by its counterpart *Stack-S (norm)* although not significantly. Returning to the central claim of this thesis that, since a linear combination is sometimes optimal, we aim to sometimes significantly outperform it and, in the remaining cases, achieve near the same performance. This empirical behavior upholds this central claim.

In Figure 7.7, we display the performance changes for *Stack-S (norm)* and *STRIVE-S (norm)* relative to the best base classifier — *SVM* classifier. Since F1 can also be computed as $\frac{2 * TruePos}{2 * TruePos + FalsePos + FalseNeg}$, we display the changes in the three components that determine F1: true positives, false positives, and false negatives. Not only does *STRIVE-S (norm)* achieve considerable reductions in error of 8-18% (using F1 optimized thresholds) and 5-16% (using error optimized thresholds), but in all but one case, it also increases by a fair margin the improvement attained by *Stack-S (norm)*. Furthermore, since *STRIVE-S*

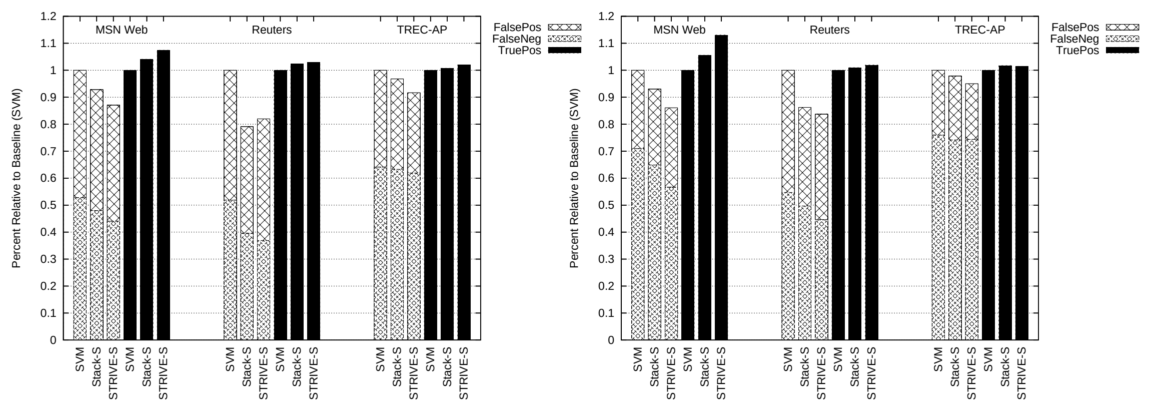


Figure 7.7: For *Stack-S (norm)* and *STRIVE-S (norm)* change relative to the best base classifi er — the *SVM* classifi er. On the *left*, we show the relative change using thresholds optimized for F1, and on the *right*, we show the relative change using thresholds optimized for error. In both fi gures, we display the changes in the three components that determine F1: true positives, false positives, and false negatives. Not only does *STRIVE-S (norm)* achieve considerable reductions in error of 8-18% (*left*) and 5-16% (*right*), but in all but one case, it also increases by a fair margin the improvement attained by *Stack-S (norm)*.

(norm) increases the number of true positives while decreasing both components of error, *STRIVE-S (norm)* improves both precision and recall.

Additional Experiments

In earlier work, we investigated whether the reliability indicators could be directly incorporated into the base classifiers. That is, we wanted to understand to what extent their information can be directly used to improve classification and to what extent it is conditional on the presence of classifier outputs. To examine these issues, we performed an experiment where we added all of the reliability indicators to the standard document representation and built a model using *Dnet*. Because the current work uses an expanded set of reliability-indicators, we do not directly include those tables here. However, the reader should note that, while including the reliability-indicators directly at the base level led to improvements over the base model, the STRIVE method was still superior.

7.3 An Analysis of Reliability Indicator Usefulness

In order to determine which variables show the most promise for future investigation, we would like to measure to what extent each reliability indicator contributes to the final model.

Since we do not want the analysis to be sensitive to linear relationships, we use the *STRIVE-D (norm)* model for analysis. We start with the final model and perform backward selection over the testing set by deleting the variable that results in the greatest increase in average logscore and retraining a new model. The average logscore for model M is:

$$LS(M) = N^{-1} \sum_{i=1}^{N} \log \hat{P}_M(c(x_i)|x_i). \tag{7.1}$$

The increase in average logscore for variable v is: $LS(M - v) - LS(M)$. Thus, if this quantity is negative, the variable must have participated in the model, and its deletion lead to a degradation in model quality. If the change is zero, then the variable either contributed no predictive power to the model or (more likely) did not participate in the model. This could either be because the variable was correlated with another variable present or was judged to have too little predictive power. In either case, it is not a good candidate for future study. Finally, if the change is positive, it means the variable must have participated in the model and deleting the variable resulted in an improvement in model quality.

Backward selection continued until each of the 70 reliability indicators were removed from the model. Thus the last variable deleted is the one that contributed the most to the model. The classifier outputs were always available to be included in the model. If multiple variables tied in a round to be deleted, then all tied variables were deleted. Each variable was assigned a rank from 1 to 70 according to the number of variables deleted previously plus one. Thus, if 1 variable was deleted in the first seven rounds and then 5 tied variables were deleted in round 8, the variable deleted in the next round would be assigned a 13. The final ranking roughly indicates the importance of the variables with 70 being best. Since a binary model is built for each class, this procedure was repeated for each binary classification model in a corpus.

Tables 7.5-7.6 present the *average* ranking of variables across classes in the MSN Web and Reuters corpus, respectively. Unfortunately, this procedure is too computationally intensive to perform for the TREC-AP corpus.[5] We also present the average across binary class models for all three corpora for the first round in Tables 7.7-7.9

We note that interpreting the average logscores in Tables 7.7-7.9 directly as importance can be misleading. For example, note that *SigmaPositiveNeighborDistance* is worst according to average change in logscore in Table 7.7 but best according to average ranking in Table 7.5. This variable often contributes to the models but by average logscore its effect is overall negative in the *first round* of backward selection. The reason for this is twofold. First, a negative score for deleting an important variable is often (artificially) near

[5]One round for 70 variables times 20 classes takes approximately four days.

zero initially, because many other variables provide some amount of redundant information. Secondly, the initial negative impact on the model can be a result of how the variable is used in conjunction with another variable. In fact, when another variable or two has been deleted, this variable's score often turns positive.

Thus, we believe the rankings presented 7.7-7.9 present a more accurate picture of variable importance. The logscores provide a valuable sign that addressing this behavior may improve how effectively the variables are used and bears further investigation. Additionally, when a variable's average logscore in the first round is zero, it is fairly safe to assume that the variable can be ignored because it is correlated with another variable or truly provides no information.

Interestingly the *Unigram*-based variables do not heavily influence the models according to either rankings or the logscores (where they often have zero). This was not the case in our earlier work [BDH02, BDH05], and it was in fact the success of these variables that prompted the creation of the other classifier-based variables. This suggests that either the inclusion of the other variables have made the *Unigram*-based variables obsolete, or more likely, the inclusion of the k*NN* classifier has decreased the importance of the *Unigram* classifier and the variables that are indicative of its performance.

Next we note that for the Reuters corpus, most of the variables in Table 7.6 have a ranking of at most 4. Given the small size of the corpus, this and the numerous zero scores in Table 7.7 indicate that there simply is not enough training data in the Reuters corpus to make effective use of many reliability indicators.

Comparing the remaining highly ranked variables in the Reuters corpus with the MSN Web ranking, we see many similarities. The classifier-vote based variable *PercentPredicting-Positive* is, unsurprisingly, very high in both. More interestingly, many of the k*NN* (*e.g. SigmaPositiveNeighborDistance*, *kNNShiftMeanPred*), decision tree (*e.g.*, *DtreeShiftStd-ConfDiff*), and *SVM* (*e.g. signedNNSV*, *SVMShiftMeanConfDiff*) based variables are ranked highly in both corpora. This verifies the importance of these variables and justifies continuing development of them. Given the strong ties of these variables to particular classification models, they are the best candidates for understanding how the information in these variables could be more directly integrated into the model during training.

Additionally, we see that many of the feature selection variables are also present. Though primarily it is the "before" variants or the "delta" variants that show up. Indicating that these variables are useful primarily as an indication of the information that has been lost. In addition to providing a starting point for future analysis, we feel the similarities across these corpora provide hope for models that can use the training data in conjunction to come up with models which effectively leverage information across the corpora.

SigmaPositiveNeighborDistance	57.31	*UPercentInPosBeforeFS*	18.77
MeanPositiveNeighborDistance	53.08	*NB_MeanLogOfStrengthGivenNeg*	18.54
PercentPredictingPositive	48.38	*FeaturesSeenInNegDelta*	18.31
U%FavoringNegBeforeFS	43.08	*NumFeaturesDiscarded*	18.15
%FavoringNegBeforeFS	41.38	*NB_StdDeviation*	18.00
WordsSeenInPosDelta	39.62	*MeanNeighborDistance*	17.92
U%FavoringPosBeforeFS	39.23	*NumUniqueWords*	17.77
%FavoringPosBeforeFS	38.15	*NB_StdDevLogOfStrengthGivenNeg*	17.69
SigmaNegativeNeighborDistance	37.31	*NumWordsDiscarded*	17.31
kNNShiftMeanConfDiff	36.54	*PercentOOV*	17.00
UpercentInNegativeBeforeFS	36.31	*NeighborhoodRadius*	17.00
signedNNSV	35.23	*PercentRemoved*	16.15
PercentInNegativeBeforeFS	33.92	*U%FavoringPosAfterFS*	15.62
kNNShiftMeanPred	33.15	*%FavoringNegAfterFS*	15.31
PercentInPosBeforeFS	31.85	*NB_MeanShift*	15.23
FeaturesSeenInPosDelta	31.31	*UniqueAfterFS*	15.00
PercentWordsPointingToPosDelta	30.31	*UPercentWordsPointingToPosDelta*	14.92
kNNShiftStdDevConfDiff	28.15	*DocumentLengthAfterFS*	14.62
UPercentWordsPointingToNegDelta	27.38	*EffectiveDocumentLength*	14.62
SVMShiftMeanConfDiff	26.31	*stdDevGoodSVProximity*	13.46
SVMShiftStdDevConfDiff	25.23	*PercentAgreeWBest*	13.15
PercentUnique	25.08	*U%FavoringNegAfterFS*	11.85
WordsSeenInNegDelta	24.46	*UnigramStdDeviation*	11.85
SigmaNeighborDistance	23.00	*PercentInNegAfterFS*	11.85
DtreeShiftStdDevConfDiff	22.46	*UPercentInPosAfterFS*	11.85
DtreeShiftMeanConfDiff	21.62	*UniStdDevLogOfStrengthGivenNeg*	11.85
NumTrainingWordsDiscarded	21.31	*UPercentInNegAfterFS*	11.85
meanGoodSVProximity	21.31	*UniMeanLogOfStrengthGivenNeg*	11.85
UniquePercentRemoved	21.15	*UnigramMeanShift*	11.85
PercentWordsPointingToNegDelta	21.15	*UniStdDevLogOfStrengthGivenPos*	11.85
EffectiveUniqueWords	20.77	*PercentInPosAfterFS*	11.85
PercentUniqueOOV	20.54	*UniMeanLogOfStrengthGivenPos*	11.85
NumTrainingFeaturesDiscarded	19.23	*%FavoringPosAfterFS*	11.77
NB_MeanLogOfStrengthGivenPos	19.00	*NB_StdDevLogOfStrengthGivenPos*	10.92
MeanNegativeNeighborDistance	18.85	*DocumentLength*	10.46

Table 7.5: In backward selection over the MSN Web *testing* set, deleting the variable that most improved the average logscore of the model allows us to rank the variables in rough order of impact by the average round a feature was deleted in. A higher average rank means a feature has greater impact on the model.

UPercentInPosBeforeFS	29.20	*stdDevGoodSVProximity*	4.00
PercentPredictingPositive	29.10	*PercentUniqueOOV*	4.00
SigmaPositiveNeighborDistance	23.00	*UniStdDevLogOfStrengthGivenNeg*	4.00
kNNShiftMeanPred	22.90	*PercentRemoved*	4.00
U%FavoringPosBeforeFS	22.30	*FeaturesSeenInNegDelta*	4.00
%FavoringPosBeforeFS	22.20	*NumFeaturesDiscarded*	4.00
PercentInPosBeforeFS	16.00	*UniqueAfterFS*	4.00
DtreeShiftStdDevConfDiff	15.70	*UPercentInNegAfterFS*	4.00
SVMShiftMeanConfDiff	15.50	*MeanNegativeNeighborDistance*	4.00
signedNNSV	10.80	*UniMeanLogOfStrengthGivenNeg*	4.00
NumTrainingFeaturesDiscarded	10.70	*NB_MeanLogOfStrengthGivenNeg*	4.00
PercentInNegAfterFS	10.60	*NeighborhoodRadius*	4.00
%FavoringPosAfterFS	10.50	*EffectiveUniqueWords*	4.00
PercentAgreeWBest	10.20	*NB_StdDevLogOfStrengthGivenPos*	4.00
PercentInNegativeBeforeFS	10.10	*UnigramMeanShift*	4.00
WordsSeenInPosDelta	9.90	*NumUniqueWords*	4.00
PercentUnique	9.80	*DtreeShiftMeanConfDiff*	4.00
MeanPositiveNeighborDistance	9.80	*UniStdDevLogOfStrengthGivenPos*	4.00
FeaturesSeenInPosDelta	9.70	*UPercentWordsPointingToPosDelta*	4.00
PercentOOV	9.60	*DocumentLengthAfterFS*	4.00
UPercentWordsPointingToNegDelta	9.60	*PercentInPosAfterFS*	4.00
%FavoringNegBeforeFS	9.30	*kNNShiftStdDevConfDiff*	4.00
SigmaNegativeNeighborDistance	8.90	*UniMeanLogOfStrengthGivenPos*	4.00
U%FavoringPosAfterFS	4.00	*EffectiveDocumentLength*	4.00
U%FavoringNegAfterFS	4.00	*NumWordsDiscarded*	3.90
UnigramStdDeviation	4.00	*NB_StdDeviation*	3.90
NB_MeanShift	4.00	*NB_MeanLogOfStrengthGivenPos*	3.90
DocumentLength	4.00	*SigmaNeighborDistance*	3.90
kNNShiftMeanConfDiff	4.00	*UpercentInNegativeBeforeFS*	3.90
NB_StdDevLogOfStrengthGivenNeg	4.00	*MeanNeighborDistance*	3.80
NumTrainingWordsDiscarded	4.00	*SVMShiftStdDevConfDiff*	3.70
UniquePercentRemoved	4.00	*WordsSeenInNegDelta*	3.60
meanGoodSVProximity	4.00	*PercentWordsPointingToPosDelta*	3.60
UPercentInPosAfterFS	4.00	*U%FavoringNegBeforeFS*	3.50
%FavoringNegAfterFS	4.00	*PercentWordsPointingToNegDelta*	3.50

Table 7.6: In backward selection over the Reuters *testing* set, deleting the variable that most improved the average logscore of the model allows us to rank the variables in rough order of impact by the average round a feature was deleted in. A higher average rank means a feature has greater impact on the model.

PercentPredictingPositive	-0.00062201	*UniStdDevLogOfStrengthGivenPos*	0
kNNShiftMeanPred	-0.00042239	*PercentInPosAfterFS*	0
MeanPositiveNeighborDistance	-0.00026129	*UniMeanLogOfStrengthGivenPos*	0
PercentInNegativeBeforeFS	-0.00018542	*SigmaNeighborDistance*	0.00000242
signedNNSV	-0.00015635	*DocumentLength*	0.00000354
NumTrainingFeaturesDiscarded	-0.00009238	*EffectiveDocumentLength*	0.00000425
NB_MeanLogOfStrengthGivenPos	-0.00007011	*NumFeaturesDiscarded*	0.00001003
UniquePercentRemoved	-0.00006485	*PercentUnique*	0.00001192
PercentWordsPointingToPosDelta	-0.00006022	*WordsSeenInPosDelta*	0.00001328
NB_MeanShift	-0.00005192	*UPercentWordsPointingToNegDelta*	0.00001446
FeaturesSeenInPosDelta	-0.00004302	*DocumentLengthAfterFS*	0.00001699
UniqueAfterFS	-0.00004212	*SigmaNegativeNeighborDistance*	0.00002219
NumUniqueWords	-0.00002368	*DtreeShiftMeanConfDiff*	0.00003575
MeanNeighborDistance	-0.00001981	*FeaturesSeenInNegDelta*	0.00003596
%FavoringPosAfterFS	-0.00001544	*PercentOOV*	0.00004415
WordsSeenInNegDelta	-0.00001418	*stdDevGoodSVProximity*	0.00005600
U%FavoringPosAfterFS	-0.00001300	*PercentWordsPointingToNegDelta*	0.00007022
MeanNegativeNeighborDistance	-0.00001187	*NB_StdDevLogOfStrengthGivenNeg*	0.00007023
EffectiveUniqueWords	-0.00001042	*UPercentWordsPointingToPosDelta*	0.00007758
SVMShiftStdDevConfDiff	-0.00000790	*kNNShiftStdDevConfDiff*	0.00008055
%FavoringNegAfterFS	-0.00000621	*U%FavoringPosBeforeFS*	0.00008772
NumTrainingWordsDiscarded	-0.00000546	*PercentInPosBeforeFS*	0.00008842
meanGoodSVProximity	-0.00000530	*kNNShiftMeanConfDiff*	0.00009565
PercentUniqueOOV	-0.00000423	*UpercentInNegativeBeforeFS*	0.00010505
NumWordsDiscarded	-0.00000116	*PercentRemoved*	0.00011062
U%FavoringNegAfterFS	0	*PercentAgreeWBest*	0.00011245
UnigramStdDeviation	0	*SVMShiftMeanConfDiff*	0.00011400
PercentInNegAfterFS	0	*DtreeShiftStdDevConfDiff*	0.00011990
UPercentInPosAfterFS	0	*U%FavoringNegBeforeFS*	0.00013617
NB_StdDeviation	0	*NeighborhoodRadius*	0.00014646
UniStdDevLogOfStrengthGivenNeg	0	*UPercentInPosBeforeFS*	0.00016584
UPercentInNegAfterFS	0	*%FavoringPosBeforeFS*	0.00019315
UniMeanLogOfStrengthGivenNeg	0	*NB_MeanLogOfStrengthGivenNeg*	0.00020146
NB_StdDevLogOfStrengthGivenPos	0	*%FavoringNegBeforeFS*	0.00020262
UnigramMeanShift	0	*SigmaPositiveNeighborDistance*	0.00040607

Table 7.7: This table shows the average reduction in logscore across classes caused by deleting each variable individually from the final models in the MSN Web *testing* set. A negative score indicates that the deleting the variable negatively impacts the models, since deleting it reduces the logscore. A score of zero indicates the variable has no impact on the models, while positive indicates the variable is included in the models but hurts them on average

PercentPredictingPositive	-0.00145601	*UniqueAfterFS*	0
SigmaPositiveNeighborDistance	-0.00052718	*NumFeaturesDiscarded*	0
PercentInPosBeforeFS	-0.00040701	*UPercentInNegAfterFS*	0
UPercentInPosBeforeFS	-0.00022327	*UniMeanLogOfStrengthGivenNeg*	0
U%FavoringPosBeforeFS	-0.00020924	*PercentInNegativeBeforeFS*	0
PercentUnique	-0.00016828	*NB_MeanLogOfStrengthGivenNeg*	0
DtreeShiftStdDevConfDiff	-0.00014711	*EffectiveUniqueWords*	0
%FavoringPosBeforeFS	-0.00014086	*NB_StdDevLogOfStrengthGivenPos*	0
U%FavoringNegBeforeFS	-0.00014017	*PercentAgreeWBest*	0
kNNShiftMeanPred	-0.00012356	*UnigramMeanShift*	0
SVMShiftStdDevConfDiff	-0.00009941	*NumUniqueWords*	0
WordsSeenInPosDelta	-0.00006886	*UniStdDevLogOfStrengthGivenPos*	0
%FavoringPosAfterFS	-0.00005240	*UPercentWordsPointingToPosDelta*	0
MeanNeighborDistance	-0.00004794	*PercentInPosAfterFS*	0
DocumentLengthAfterFS	-0.00002364	*kNNShiftStdDevConfDiff*	0
meanGoodSVProximity	-0.00001837	*EffectiveDocumentLength*	0
NumWordsDiscarded	-0.00001776	*UniMeanLogOfStrengthGivenPos*	0
U%FavoringPosAfterFS	0	*WordsSeenInNegDelta*	0.00001705
U%FavoringNegAfterFS	0	*NumTrainingWordsDiscarded*	0.00001747
PercentOOV	0	*NeighborhoodRadius*	0.00001917
UnigramStdDeviation	0	*NB_StdDeviation*	0.00002063
NB_MeanShift	0	*PercentWordsPointingToNegDelta*	0.00003801
DocumentLength	0	*%FavoringNegBeforeFS*	0.00004508
PercentInNegAfterFS	0	*SigmaNegativeNeighborDistance*	0.00005758
NB_StdDevLogOfStrengthGivenNeg	0	*FeaturesSeenInPosDelta*	0.00005871
UniquePercentRemoved	0	*PercentUniqueOOV*	0.00005899
UPercentInPosAfterFS	0	*MeanNegativeNeighborDistance*	0.00005920
%FavoringNegAfterFS	0	*UpercentInNegativeBeforeFS*	0.00006334
stdDevGoodSVProximity	0	*SVMShiftMeanConfDiff*	0.00007120
NumTrainingFeaturesDiscarded	0	*kNNShiftMeanConfDiff*	0.00007483
SigmaNeighborDistance	0	*MeanPositiveNeighborDistance*	0.00008386
UPercentWordsPointingToNegDelta	0	*PercentWordsPointingToPosDelta*	0.00009385
UniStdDevLogOfStrengthGivenNeg	0	*NB_MeanLogOfStrengthGivenPos*	0.00012559
PercentRemoved	0	*signedNNSV*	0.00018006
FeaturesSeenInNegDelta	0	*DtreeShiftMeanConfDiff*	0.00043114

Table 7.8: This table shows the average reduction in logscore across classes caused by deleting each variable individually from the final models in the Reuters *testing* set. A negative score indicates that the deleting the variable negatively impacts the models, since deleting it reduces the logscore. A score of zero indicates the variable has no impact on the models, while positive indicates the variable is included in the models but hurts them on average

%FavoringPosBeforeFS	-0.00059795	*PercentUniqueOOV*	0
SigmaPositiveNeighborDistance	-0.00047683	*UniStdDevLogOfStrengthGivenNeg*	0
kNNShiftStdDevConfDiff	-0.00025752	*FeaturesSeenInNegDelta*	0
SVMShiftMeanConfDiff	-0.00011884	*UPercentInNegAfterFS*	0
NB_MeanShift	-0.00004756	*UniMeanLogOfStrengthGivenNeg*	0
SigmaNegativeNeighborDistance	-0.00003388	*%FavoringPosAfterFS*	0
MeanNegativeNeighborDistance	-0.00001862	*NeighborhoodRadius*	0
SigmaNeighborDistance	-0.00001632	*PercentAgreeWBest*	0
%FavoringNegBeforeFS	-0.00001630	*UnigramMeanShift*	0
NumFeaturesDiscarded	-0.00001608	*UniStdDevLogOfStrengthGivenPos*	0
PercentUnique	-0.00001241	*PercentInPosAfterFS*	0
stdDevGoodSVProximity	-0.00000973	*EffectiveUniqueWords*	0.00000010
MeanPositiveNeighborDistance	-0.00000893	*NumTrainingFeaturesDiscarded*	0.00000020
PercentInNegativeBeforeFS	-0.00000813	*WordsSeenInNegDelta*	0.00000045
PercentWordsPointingToNegDelta	-0.00000577	*NumUniqueWords*	0.00000066
U%FavoringPosBeforeFS	-0.00000517	*PercentRemoved*	0.00000072
DtreeShiftStdDevConfDiff	-0.00000219	*PercentInNegAfterFS*	0.00000136
DocumentLengthAfterFS	-0.00000196	*PercentWordsPointingToPosDelta*	0.00000145
UniMeanLogOfStrengthGivenPos	-0.00000185	*EffectiveDocumentLength*	0.00000179
%FavoringNegAfterFS	-0.00000175	*NB_MeanLogOfStrengthGivenNeg*	0.00000198
NumWordsDiscarded	-0.00000155	*NB_StdDevLogOfStrengthGivenPos*	0.00000233
MeanNeighborDistance	-0.00000126	*kNNShiftMeanConfDiff*	0.00000421
FeaturesSeenInPosDelta	-0.00000119	*UPercentInPosBeforeFS*	0.00000447
PercentOOV	-0.00000095	*meanGoodSVProximity*	0.00000448
UPercentWordsPointingToPosDelta	-0.00000083	*NumTrainingWordsDiscarded*	0.00000449
DocumentLength	-0.00000076	*UpercentInNegativeBeforeFS*	0.00000641
UniqueAfterFS	-0.00000060	*DtreeShiftMeanConfDiff*	0.00000821
UPercentWordsPointingToNegDelta	-0.00000050	*kNNShiftMeanPred*	0.00001059
U%FavoringPosAfterFS	0	*SVMShiftStdDevConfDiff*	0.00001137
U%FavoringNegAfterFS	0	*U%FavoringNegBeforeFS*	0.00001328
UnigramStdDeviation	0	*WordsSeenInPosDelta*	0.00002368
NB_StdDevLogOfStrengthGivenNeg	0	*UniquePercentRemoved*	0.00002740
NB_StdDeviation	0	*PercentInPosBeforeFS*	0.00004043
UPercentInPosAfterFS	0	*PercentPredictingPositive*	0.00009150
NB_MeanLogOfStrengthGivenPos	0	*signedNNSV*	0.00026558

Table 7.9: This table shows the average reduction in logscore across classes caused by deleting each variable individually from the final models in the TREC-AP *testing* set. A negative score indicates that the deleting the variable negatively impacts the models, since deleting it reduces the logscore. A score of zero indicates the variable has no impact on the models, while positive indicates the variable is included in the models but hurts them on average

7.4 RCV1-v2

	MacroF1	MicroF1	Error	C(1,10)	C(10,1)	ROC Area
Dnet	*0.3999*	*0.6516*	*0.0185*	*0.0900*	*0.0282*	0.8983
Unigram	0.4651	0.7119	0.0170	0.0714	0.0267	0.9055
naïve Bayes	0.4255	0.6641	*0.0185*	0.0857	0.0250	0.9343
SVM	$\mathbf{0.6213}^{\boldsymbol{B}}_{\mathrm{S}}$	$0.8170^{\mathbb{B}}_{\mathbb{S}}$	$0.0109^{\boldsymbol{B}}$	$0.0496^{\boldsymbol{BD}}$	$0.0194^{\boldsymbol{BD}}$	$\mathbf{0.9703}^{\boldsymbol{B}\mathrm{D}}_{\mathrm{S}\boldsymbol{R}}$
kNN	0.5024	0.7496	0.0143	0.0817	0.0224	*0.8828*
Best By Class	$0.6187^{\boldsymbol{D}}_{\mathrm{S}}$	$0.8170_{\mathbb{S}}$	$0.0109^{\boldsymbol{D}}$	$0.0499^{\boldsymbol{D}}$	$0.0197^{\boldsymbol{D}}$	N/A
Majority	0.5429	0.7910	0.0131	0.0555	0.0219	N/A
Stack-D (norm)	0.6064	$0.8135_{\mathbb{S}}$	0.0109	0.0491	0.0195	0.9350
Stack-S (norm)	0.5952	0.7354	$\mathbf{0.0105}^{\boldsymbol{BD}}_{\boldsymbol{S}\mathbb{R}}$	0.0525	$0.0185^{\boldsymbol{BD}}_{\boldsymbol{S}}$	0.9229
STRIVE-D (norm)	0.5953	0.8128	0.0112	0.0501	0.0195	0.9332
STRIVE-S (norm)	$0.6111_{\boldsymbol{R}}$	$\mathbf{0.8235}^{\mathbb{BD}}_{\mathbb{SR}}$	$0.0106^{\mathrm{BD}}_{\boldsymbol{R}}$	$\mathbf{0.0485}^{\mathrm{D}}_{\mathrm{R}}$	$\mathbf{0.0184}^{\mathrm{B}\boldsymbol{D}}_{\boldsymbol{R}}$	$0.9473_{\mathbb{R}}$
BestSelect	0.8193	0.9389	0.0054	0.0138	0.0127	N/A

Table 7.10: Performance on RCV1-v2 Corpus. The best performance (omitting the oracle *BestSelect*) in each column is given in bold. A notation of 'B', 'D', 'S', or 'R' indicates a method significantly outperforms all (other) **B**ase classifiers, **D**efault combiners, **S**tacking methods, or **R**eliability-indicator based Striving methods at the $p = 0.05$ level. A blackboard (hollow) font is used to indicate significance for the macro-sign test and micro-sign test. A normal font indicates significance for the macro t-test. For the macro-averages (*i.e.*, excluding micro F1) when both tests are significant it is indicated with a bold, italicized font.

As discussed in Section 6.2.5, we performed experiments with the RCV1-v2 corpus after all methods were fully developed to demonstrate we had not overtuned these results on the earlier corpora. As discussed earlier, the standard chronological split used has 23149 training documents and 781265 testing documents with 101 topics that have at least one training document.

For all base classifiers except for the SVM, the top 1000 features by mutual information were used on a per class basis. All base classifiers except for the SVM used the same representation and settings as for the other experiments. In order to reproduce the settings given for the baseline in [LYRL04], the SVM used a normed tfidf representation with all features that occurred in at least 3 training documents.

Table 7.10 presents the summary for the primary methods discussed in this chapter. The results for *STRIVE-S (norm)* are generally consistent with the results presented earlier. The only notable exception is that neither *STRIVE-S (norm)* nor any other combination method is able to outperform the base SVM on MacroF1. However, according to every other performance measure except ROC area, *STRIVE-S (norm)* outperforms all the base classifiers

— usually with statistical significance. In fact, on those same measures *STRIVE-S (norm)* outperforms all other methods except for *Stack-S (norm)* on error. The mean difference in error is not significantly different from *Stack-S (norm)* as shown by the macro t-test, however, as demonstrated by the macro sign-test, it is outperformed on a statistically significant number of the classes.

Making a closer examination of the results for *Stack-S (norm)* shows that while it slightly outperforms *STRIVE-S (norm)* on error, *Stack-S (norm)* decreases performance on MacroF1 and greatly decreases performance on MicroF1. This appears to be in large part because of several classes where *Stack-S (norm)* greatly decreases F1 performance. Apparently, the linear combination of classifier outputs is much more sensitive to the particular threshold than the corresponding strive model. While there is a chance that this could be due to topic drift which occurs after the chronological split, Strive and the meta-classifiers using decision trees appear to be able to mitigate such effects.

Figure 7.8 presents a by-class analysis of the change in F1 and error for *STRIVE-S (norm)* and *Stack-S (norm)*. The lines shown are the least-squares fit where each topic receives the same weight. These figures clearly show how the stacking method severely hurts performance on several classes and drags down F1 performance while the striving method has little impact on macroF1 even on a per-class basis. In terms of error, we see that both methods perform worse over smaller topics but gain enough over larger topics to create a net positive effect. Striving seems to have less negative effects on a per-class basis but also requires slightly more positive examples to consistently boost baseline performance. Because of the smaller number of topics in the other corpora, it is difficult to tell whether these trends exist on a per-topic basis in those corpora as well.

To give a sense of the improvement when using the thresholded predictions for microF1, *STRIVE-S (norm)* commits 501400 false negative predictions versus the *SVM*'s 513511 false negatives and 369291 false positive predictions versus the *SVM*'s 391038. In total, *STRIVE-S (norm)* commits 12111 less false negatives, 21747 less false positives, and 33858 less errors in prediction than the *SVM*. To put this in context, Figure 7.9 again displays the changes in performance relative to the *SVM* classifier.

In summary, key lessons from the RCV1-v2 corpus are that while striving may not always increase F1 performance on a per-class basis, the net reduction in total number of errors causes both a significant increase in microF1 performance and overall error. Also, we note that in comparison to the other corpora, the SVM dominates the base classifiers more here. By optimizing k using cross-validation we expect the performance of kNN will increase to be closer to the SVM and thus increase the potential for classifier combination. Additionally, both striving and stacking seem to require between 64 and 256 (2^6 to 2^8)

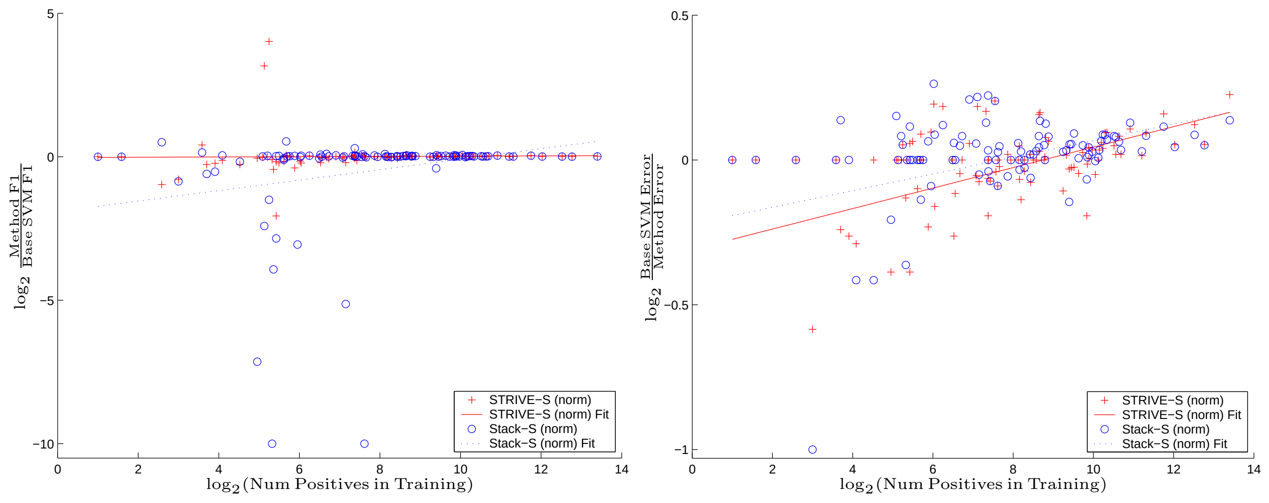


Figure 7.8: Each point presents the performance for a single class in the RCV1-v2 corpus. Improvement in F1 over the baseline SVM is shown on the *left* while improvement in error is shown on the *right*. As is typical, both axes are given in the log-domain. In case of a zero denominator or numerator, the log-ratio is defi ned as $10/-10$ respectively. On *left* we see that *Stack-S (norm)* severely decreases the F1 performance on several classes. On *right* we see that (when performance differs from the baseline) both methods show a larger increase in performance according to error over the baseline as the class becomes more prevalent. Striving appears to require slightly more positive examples than stacking which is expected given the higher dimensionality. The regression fi ts shown are fi t only to the classes where the metaclassifi er's performance differs from the baseline.

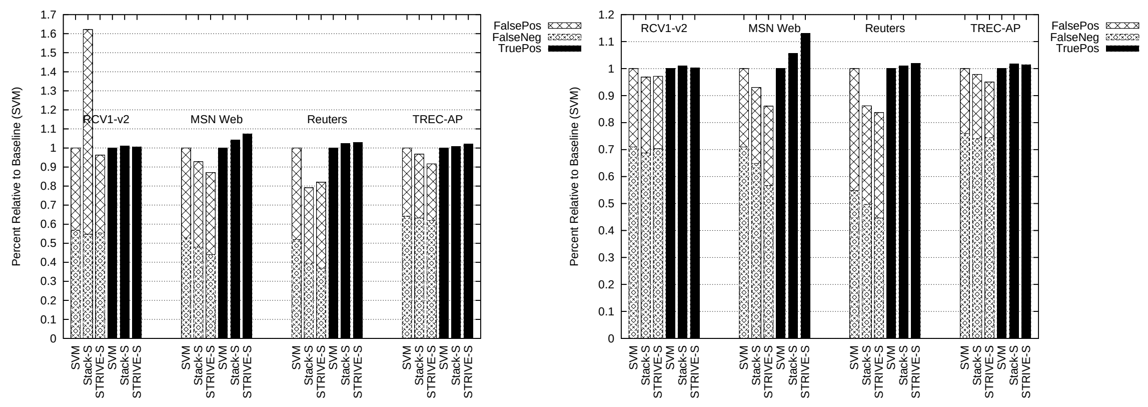


Figure 7.9: For *Stack-S (norm)* and *STRIVE-S (norm)* change relative to the best base classifi er —the *SVM* classifi er —over all the topic classifi cation corpora. On the *left*, we show the relative change using thresholds optimized for F1, and on the *right*, we show the relative change using thresholds optimized for error. In both fi gures, we display the changes in the three components that determine F1: true positives, false positives, and false negatives. Not only does *STRIVE-S (norm)* achieve considerable reductions in error of 4-18% (*left*) and 3-16% (*right*), but in all but one case, it also increases by a fair margin the improvement attained by *Stack-S (norm)*. Furthermore, *STRIVE-S (norm)* never hurts performance relative to the *SVM* on these performance measures as *Stack-S (norm)* does over RCV1-v2 on the far left.

positive examples to begin to have a net effect over baseline performance in error and are thus more effective for larger classes than smaller. In all, *STRIVE-S (norm)* continues to be the superior choice having less negative impact on macroF1 than stacking and increasing performance over the baseline according to microF1 and all of the linear utility performance measures.

7.5 Summary and Conclusions

In this chapter, we reviewed a methodology for building a metaclassifier for text documents that centers on combining multiple distinct classifiers with probabilistic learning and inference that leverages reliability-indicator variables. Reliability indicators provide information about the context-sensitive nature of classifier reliability, informing a metaclassifier about the best way to integrate the outputs from base-level classifiers. We introduced the *STRIVE* methodology that uses reliability indicators in a hierarchical combination model and reviewed comparative studies comparing *STRIVE* with other combination mechanisms.

We conducted experimental evaluations over four text-classification corpora (MSN Web, Reuters 21578, TREC-AP, and RCV1-v2) with a variety of performance measures. These measures were selected to determine the robustness of the classification procedures under different misclassification penalties. The empirical evaluations support the conclusion that a simple majority vote in situations where one of the classifiers performs strongly can weaken the best classifier's performance. In contrast, in all of these corpora across all measures, the *STRIVE* methodology was competitive. STRIVE using a SVM metaclassifier produced the top performer in nearly every category except ROC area and, outside of that, was never beaten statistically significantly. Furthermore, on a class-by-class basis, the *STRIVE* methodology using a meta-decision tree produced receiver-operating characteristic curves that dominated the other classifiers in nearly every class of the MSN Web corpus, which demonstrates that it provides the best choice for *any* possible linear utility function in this corpus.

In conclusion, the experiments show that stacking and *STRIVE* provide robust combination schemes across a variety of performance measures. In addition, the experiments show the central claim of this thesis that context-depended combination (*STRIVE-S (norm)*) procedures provide an effective way of combining classifiers that are generally superior to constant-weighted linear combinations of the classifier's estimates of the posterior or log-odds (*Stack-S (norm)*). In the remaining chapters, we turn to issues of how we can alleviate the need for training data and applying combination outside of topic classification.

	MacroF1	MicroF1	Error	C(1,10)	C(10,1)	ROC Area	ROC[0,0.1]
Dnet	*0.5477*	0.5813	0.0584	*0.3012*	0.0772	*0.8802*	0.5638
Unigram	0.5982	0.6116	0.0594	0.2589	*0.0812*	0.9003	0.6114
naï ve Bayes	0.5527	*0.5619*	*0.0649*	0.2853	0.0798	0.8915	*0.5516*
SVM	0.6727	0.7016	0.0455	0.2250	0.0794	0.9123	0.6960
kNN	0.6480	0.6866	0.0464	0.2524	0.0733	0.8873	0.6541
Best By Class	0.6727	0.7016	0.0452	0.2235	0.0729	N/A	N/A
Majority	0.6643	0.6902	0.0479	0.2133	0.0765	N/A	N/A
Stack-D	0.6924	0.7233	0.0423	0.1950	0.0708	0.9361	0.7356
Stack-S	0.6801	0.7118	0.0438	0.2076	0.0701	0.9286	0.7196
Stack-D (norm)	0.6924	0.7233	0.0423	0.1950	0.0708	0.9361	0.7356
Stack-S (norm)	0.6939	0.7250	0.0423	0.1971	0.0705	0.9334	0.7349
STRIVE-D	0.6975	0.7304	0.0413	0.1863	0.0703	**0.9459**	0.7460
STRIVE-S	0.6527	0.6767	0.0498	0.2251	0.0800	0.9159	0.6819
STRIVE-D (norm)	0.6988	0.7327	0.0413	0.1846	0.0697	0.9454	0.7457
STRIVE-S (norm)	0.7173	0.7437	0.0392	0.1835	0.0682	0.9260	0.7547
STRIVE-S Local (norm)	**0.7251**	**0.7530**	**0.0386**	**0.1810**	**0.0656**	0.9150	**0.7600**
STRIVE-S LSelect (norm)	0.7197	0.7496	0.0388	0.1860	0.0664	0.9097	0.7529
BestSelect	0.8719	0.8924	0.0223	0.0642	0.0565	N/A	N/A

Table 7.11: All results for the MSN Web Corpus discussed in this chapter. Ignoring *BestSelect*, the overall best in each column is shown in red bold and the overall worst is shown in blue italics.

	MacroF1	MicroF1	Error	C(1,10)	C(10,1)	ROC Area	ROC[0,0.1]
Dnet	0.7846	0.8541	0.0242	0.0799	*0.0537*	0.9804	0.8844
Unigram	0.7645	0.8674	0.0234	0.0713	0.0476	0.9877	0.9086
naï ve Bayes	*0.6574*	*0.7908*	*0.0320*	*0.1423*	0.0527	*0.9703*	*0.7841*
SVM	0.8545	0.9122	0.0145	0.0499	0.0389	0.9893	0.9429
kNN	0.8097	0.8963	0.0170	0.0737	0.0336	0.9803	0.9043
Best By Class	0.8608	0.9149	0.0144	0.0496	0.0342	N/A	N/A
Majority	0.8498	0.9102	0.0155	0.0438	0.0437	N/A	N/A
Stack-D	0.8680	0.9197	0.0136	0.0410	0.0366	0.9912	0.9453
Stack-S	0.8611	0.9174	0.0141	0.0398	0.0362	0.9952	0.9576
Stack-D (norm)	0.8680	0.9197	0.0136	0.0410	0.0366	0.9912	0.9453
Stack-S (norm)	**0.8908**	**0.9307**	0.0125	0.0372	**0.0331**	**0.9956**	**0.9628**
STRIVE-D	0.8551	0.9162	0.0141	0.0461	0.0364	0.9913	0.9509
STRIVE-S	0.8289	0.9061	0.0161	0.0515	0.0420	0.9908	0.9332
STRIVE-D (norm)	0.8555	0.9172	0.0144	0.0488	0.0364	0.9913	0.9507
STRIVE-S (norm)	0.8835	0.9287	**0.0121**	**0.0352**	0.0343	0.9948	0.9616
STRIVE-S Local (norm)	0.8751	0.9261	0.0125	0.0382	0.0344	0.9890	0.9534
STRIVE-S LSelect (norm)	0.8825	0.9280	0.0128	0.0389	0.0341	0.9921	0.9553
BestSelect	0.9611	0.9789	0.0036	0.0073	0.0173	N/A	N/A

Table 7.12: All results for the Reuters Corpus discussed in this chapter. Ignoring *BestSelect*, the overall best in each column is shown in red bold and the overall worst is shown in blue italics.

	MacroF1	MicroF1	Error	C(1,10)	C(10,1)	ROC Area	ROC[0,0.1]
Dnet	0.6007	0.5706	0.0064	0.0346	*0.0081*	0.9767	0.9086
Unigram	0.6001	0.5695	0.0064	0.0347	0.0079	0.9819	0.9034
naï ve Bayes	*0.5676*	*0.5349*	*0.0065*	*0.0455*	0.0078	0.9755	0.8454
SVM	0.7361	0.6926	0.0049	0.0282	0.0077	0.9715	0.9143
kNN	0.6793	0.6533	0.0053	0.0371	0.0074	*0.9238*	*0.8088*
Best By Class	0.7356	0.6925	0.0049	0.0302	0.0073	N/A	N/A
Majority	0.7031	0.6534	0.0056	0.0307	0.0075	N/A	N/A
Stack-D	0.7331	0.7007	0.0050	0.0251	0.0073	**0.9886**	**0.9565**
Stack-S	0.7213	0.6796	0.0051	0.0280	0.0073	0.9834	0.9304
Stack-D (norm)	0.7331	0.7007	0.0050	0.0251	0.0073	**0.9886**	**0.9565**
Stack-S (norm)	0.7486	0.7011	0.0048	0.0263	0.0072	0.9834	0.9411
STRIVE-D	0.7283	0.6958	0.0051	0.0267	0.0073	0.9870	0.9512
STRIVE-S	0.7075	0.6718	0.0052	0.0310	0.0073	0.9742	0.8959
STRIVE-D (norm)	0.7246	0.6991	0.0051	0.0268	0.0073	0.9870	0.9515
STRIVE-S (norm)	**0.7532**	**0.7148**	**0.0047**	0.0277	**0.0071**	0.9771	0.9239
STRIVE-S Local (norm)	0.7439	0.7084	0.0048	**0.0231**	**0.0071**	0.9725	0.9285
STRIVE-S LSelect (norm)	0.7507	0.7104	**0.0047**	0.0273	**0.0071**	0.9731	0.9150
BestSelect	0.8986	0.8356	0.0031	0.0133	0.0058	N/A	N/A

Table 7.13: All results for the TREC-AP Corpus discussed in this chapter. Ignoring *BestSelect*, the overall best in each column is shown in red bold and the overall worst is shown in blue italics.

	MacroF1	MicroF1	Error	C(1,10)	C(10,1)	ROC Area	ROC[0,0.1]
Dnet	*0.3999*	*0.6516*	*0.0185*	*0.0900*	*0.0282*	0.8983	0.6691
Unigram	0.4651	0.7119	0.0170	0.0714	0.0267	0.9055	0.6880
naï ve Bayes	0.4255	0.6641	*0.0185*	0.0857	0.0250	0.9343	0.7094
SVM	**0.6213**	0.8170	0.0109	0.0496	0.0194	**0.9703**	**0.8538**
kNN	0.5024	0.7496	0.0143	0.0817	0.0224	*0.8828*	*0.6154*
Best By Class	0.6187	0.8170	0.0109	0.0499	0.0197	N/A	N/A
Majority	0.5429	0.7910	0.0131	0.0555	0.0219	N/A	N/A
Stack-D (norm)	0.6064	0.8135	0.0109	0.0491	0.0195	0.9350	0.8078
Stack-S (norm)	0.5952	0.7354	**0.0105**	0.0525	0.0185	0.9229	0.7976
STRIVE-D (norm)	0.5953	0.8128	0.0112	0.0501	0.0195	0.9332	0.8033
STRIVE-S (norm)	0.6111	**0.8235**	0.0106	**0.0485**	**0.0184**	0.9473	0.8131
BestSelect	0.8193	0.9389	0.0054	0.0138	0.0127	N/A	N/A

Table 7.14: All results for the RCV1-v2 Corpus discussed in this chapter. Ignoring *BestSelect*, the overall best in each column is shown in red bold and the overall worst is shown in blue italics.

Chapter 8

Inductive Transfer for Classifi er Combination

From Chapter 7, we have seen that STRIVE can use a set of reliability indicators in conjunction with the outputs of various heterogeneous classifiers to produce a combination model that consistently outperforms a linear combination of classifier outputs. However, we also saw in Section 7.4 that, while the overall combination is successful, both STRIVE and stacking seem to be less successful and sometimes hurt performance versus the best base classifier on classes with few positive examples.

In this chapter, we turn our attention to the problem of scarce data. We introduce a generalization of STRIVE, called *LABEL* (Layered Abstraction-Based Ensemble Learning), that shows how data from one dataset can be used with that from other datasets to build an inductive model of classifier combination that transfers across all the datasets and improves performance in conjunction across the tasks. In addition to the general framework, we demonstrate empirically how this approach can be used in conjunction with a decision tree to increase the average performance for *STRIVE-D (norm)*. Finally, we summarize the interesting directions for future work in the same vein.

8.1 Introduction

Given the typical scarcity of labeled data for building predictive models, the Machine Learning community has pursued methods which make use of information sources beyond the labeled data associated with a pure supervised-learning framework. An example of research in this arena is multitask learning [Car97]. In multitask learning, additional information for building models comes in the form of labels for related functions which can be learned over the same input. Although such additional labels are typically unavailable at prediction time, results have demonstrated that generalization performance can be improved on the primary task by learning to predict the new variables in addition to the output variable of interest.

We are interested in improving the performance of predictive models for cases where we have inadequate amounts of labeled training data. In contrast to multitask learning, we seek to leverage labeled data from related problems over different examples to enhance the final model used in prediction. Problems related to this challenge have been termed *classifier re-use* [BG98] or *knowledge transfer* [CK97]. We introduce a new approach to the challenge that hinges on mapping the original feature space, targeted at predicting membership in a specific topic, to a new feature space aimed at modeling the reliability of an ensemble of text classifiers.

The approach, which we call **L**ayered **A**bstraction-**B**ased **E**nsemble **L**earning (*LABEL*), has two subcomponents. First, a set of classifiers is trained on each task according to the standard supervised learning framework; a problem or task consists of determining binary membership in a specific topic. Then, we build a context-sensitive ensemble model using these classifier outputs and a set of reliability indicators (see Chapter 7) that provide an abstraction of discriminatory context appropriate for modeling classifier reliability. We thus abstract away the problem of predicting specific class membership to that of predicting the reliability of a set of classifiers for a given class. As a result, both the input features and their relationship to the class variable are the same at the metalevel; this enables the simultaneous use of all the data as a model bias across the entire set of tasks.

For a review of related work, the reader should consult Section 2.2.1. First, we describe in detail how the *LABEL* methodology generalizes the *STRIVE* model by providing a means for using data across tasks. Then, we present an empirical analysis of this methodology applied to text classification and summarize the strengths and weaknesses of the approach. Finally, we discuss promising paths for future work.

8.2 Applying Inductive Transfer to Combination

In distinction to prior efforts, we introduce a representation that is semantically coherent across tasks. Such semantic coherence facilitates the use of standard methods of inductive transfer.

We note that the experiments in this chapter use 49 reliability indicators, a subset of the variables used in Chapter 7. See Bennett, Dumais, & Horvitz [BDH02, BDH05] for additional discussion of these reliability-indicators.

8.2.1 STRIVE

As discussed more extensively in Chapter 7, the *STRIVE* methodology transforms the original learning problem into a new learning problem. In the initial problem, the base classifiers simply predict the class from a word-based representation of the document. More generally, in the original problem, each base classifier outputs a distribution (possibly unnormalized) over class labels. *STRIVE* adds another layer of learning to the base problem. Reliability-indicator functions consider the words in the document and the classifier outputs to generate the reliability indicator values, r_i, for a particular document. This approach yields a new representation of the document that consists of the values of the reliability indicators, as well as the outputs of the base classifiers. The metaclassifier exploits this new representation for learning and classification. This enables the metaclassifier to employ a model that uses the output of the base classifiers as well as the context established by the reliability indicators to make a final classification.

8.2.2 LABEL: Layered Abstraction-Based Ensemble Learning

Intuitively, regardless of the particular topic or source (*e.g.*, news feed, web page, etc.), topic discrimination tasks share some common structure. For example, longer documents tend to provide more information for identifying topics. Furthermore, documents containing words strongly correlated with a single topic are more likely to belong to that topic than documents containing words strongly correlated with several topics. Additionally, these conditions may interact with each other based on their particular values. Researchers in the field may often make similar observations after studying multiple classification problems. We seek to design a system capable of both inducing such generalizations automatically and applying them to improve the predictive performance of models.

A training corpus in text classification consists of a set of example documents labeled with each of their proper topics from a prespecified corpus-specific topic list (a document may have more than one topic). When the same representation is used for each of the binary discrimination tasks in a corpus, standard multitask learning can be used to perform classification for all of the topics in the corpus' topic list. However, standard multitask learning cannot leverage information across corpora since it would typically require knowing whether a document belongs to each of the topics from all of the corpora (where we only have in-corpus information). Additionally, the basic feature space is quite different in documents from different corpora as particular language usage varies widely. Therefore, we desire a standard representation that has the same semantics across separate tasks from both the same and different corpora.

Although *STRIVE* uses data from each task separately to build a metaclassifier for that specific task, it is straightforward to extend it to make use of labeled data across tasks. The key point is that the reliability-indicators we chose carefully abstracted away from a document's task-specific statistical regularities of word usage while maintaining the discriminatory relationship of the document's context to the task. For example, documents that come from a general topic corpus where we are trying to distinguish *Health & Fitness* from *not Health & Fitness* tend to behave very differently at the word usage distribution level than documents from a narrow financial corpus where we are trying to distinguish *Corporate Acquisitions* from *not Corporate Acquisitions*. However, in terms of the abstraction that the reliability-indicator *UnigramStdDeviation* provides, we expect a unigram classifier to show poor reliability for a particular document from either task when *UnigramStdDeviation* is high.

With this approach, we treat the metaclassifier as an abstraction, moving the focus of the analysis from discriminating a specific topic (*e.g.*, *Corporate Acquisitions* vs. *not Corporate Acquisitions*) to the problem of discriminating topic membership (*i.e.*, *In-Topic* vs. *Out-of-Topic*). The base-level classifiers trained on a particular topic are used as the representation of topic-specific knowledge, while the metaclassifier provides information about how to leverage context across topic-classification in general.

Therefore, *LABEL*, like *STRIVE*, constructs models with the same type of combination rules as that shown in Figure 7.2. The differences from *STRIVE* are in the model construction procedure. After generating the metalevel data, the metafeatures are normalized to have zero mean and a reduced standard deviation scale within their particular task.[1] At this point *STRIVE* would use data from each task to separately build a metaclassifier for each task. *LABEL* departs from this by pooling all of the data together and building a single metaclassifier (with the class variable taking the value 1 if the document is *In-Topic* for the particular task and -1 otherwise).

We now give a more formal definition of the problem. For our purposes, a task is the approximation of a single binary function, $f_i(X_i) \in \{-1, 1\}$. The input domain of each of these tasks may differ; thus, X_i denotes an input example from the $i\text{th}$ task's domain. The labeled data for each task, L_i, consists of a set of tuples $\langle \vec{x}_{i,j}, f_i(\vec{x}_{i,j}) \rangle$ (where $j = 1, \ldots, |L_i|$). Given N tasks and a performance measure *perf*, we would like an inductive learning procedure, *Train*$(i, L_1, \ldots, L_N)$, that produces a model to generate predictions for the $i\text{th}$ task. Furthermore, we desire that our performance using all the data exceeds the performance using the data for each task separately: $\sum_{i=1}^{N} E_{P_i}[\textit{perf}(\textit{Train}(i, L_1, \ldots, L_N))] >$

[1]This is not necessary for *STRIVE-D*, but for *LABEL* this helps to deal with spurious statistical variance that arises from the tasks having different numbers of training examples. Also see the Footnote 3 on Page 123 regarding feature normalization.

$\sum_{i=1}^{N} E_{P_i}[\textit{perf}(\textit{Train}(i, L_i))]$ where P_i is the probability distribution on the $i\text{th}$ task. To be of practical use, the performance achieved using only labeled data from the task $E_{P_i}[\textit{perf}(\textit{Train}(i, L_i))]$ should be competitive with the best base classifiers for this task, otherwise the solution is trivial (simply ensure the models using only labeled data from the task perform as poorly as possible).

Before applying the resulting model for prediction, it is desirable to specialize this single metaclassifier for each task in two ways. First, each task may have different priors; so these priors should be taken into account at prediction time. This can be directly accomplished by obtaining probability predictions from the metaclassifier or simply setting a different threshold for each classification task. Secondly, tasks may diverge from the average case in different ways. Thus, we may want to only retain part of the general model. The best way to address this question depends, in part, upon the choice of classification algorithm for the metaclassifier. We discuss our particular choices in Section 8.3.2 below.

8.3 Experimental Analysis

We performed an empirical analysis over standard text classification corpora to explore the effectiveness of *LABEL*. We also performed ablation experiments to elucidate how *LABEL* achieves an improvement in generalization performance. Each of the classification models use a decision threshold specific to each task. The threshold for each model and task was empirically determined over the training data.

8.3.1 Base Classifi ers

We selected for our experiments four classifiers that have been used traditionally for text classification: decision trees, linear SVMs, naïve Bayes, and a unigram classifier. Note that this is a subset of the classifiers used in Chapter 7. In particular, we did not make use of the kNN classifier in these experiments.

For each base classifier, the settings and implementations are the same as discussed in Section 6.3. The exception is that for an implementation of linear SVMs, we used the *Smox* toolkit which is based on Platt's **S**equential **M**inimal **O**ptimization algorithm [Pla98]. Since *Smox* is the best base classifier in the experiments below, it is the only base classifier we report in summarizing our experimental results.

8.3.2 Metaclassifiers

As mentioned above, the inputs to the metaclassifiers are normalized to zero mean and a reduced standard deviation scale (as estimated during the training phase).[2] The experiments reported here use a total of 49 reliability indicators which were formulated by hand in an attempt to represent potentially valuable contexts (additional detail can be found in [BDH02, BDH05]).

For these experiments, we used only a decision-tree algorithm (using Dnet) as a metaclassifier. For this reason, we refer to the primary metaclassifier implementations below as *STRIVE-D* and *LABEL-D*. Since both systems use normalized inputs, we drop the "*(norm)*" at the end of the system names used in Chapter 7. We note that by comparing these two systems directly, we see the effects of *separately* building a metaclassifier per task versus building them in conjunction.

Here, we introduce one way to specialize the single metaclassifier model learned by *LABEL-D* for decision trees. Given a single metaclassifier decision tree model, instead of using the prediction at each leaf node as the aggregate distribution across tasks of *In-Topic* vs. *Out-of-Topic,* when predicting for task T, we use the estimate:

$$P(\textit{In-Topic}_i \,|\, \text{leaf} = l) = \frac{\textit{In-Topic}_{i,l} + mp_l}{m + \textit{In-Topic}_{i,l} + \textit{Out-of-Topic}_{i,l}}. \tag{8.1}$$

For the particular binary classification task i, $\textit{In-Topic}_{i,l}$ and $\textit{Out-of-Topic}_{i,l}$ are the number *in* and *out* of topic of those training examples that fall in the leaf node, respectively. p_l is the prior *at the leaf node* of *In-Topic* obtained from using all of the data across tasks. m is the effective sample size which determines how much evidential weight, measured in "number of observed datapoints", the prior carries.

We sampled the two extremes of this spectrum, $m = 0$ and $m = \infty$. By choosing $m = 0$, we specialize the *LABEL* model to a particular task by placing all of the weight on the task-specific data. This allows some leaves to effectively have no data in them; for those leaves, we use the overall prior of in-topic according to the task-specific data. We refer to this system as *LABEL-D (repop)* since this acts as if we completely repopulated the decision tree with task-specific data.

We also present the results obtained by making the prediction at a leaf node using all of the data across tasks equally (*i.e.,* $m = \infty$ and the right side of Equation 8.1 simply becomes p_l). We refer to this as *LABEL-D (general)* since the metaclassifier is not specialized for each task other than the decision threshold. Comparing these specific instantiations

[2]See the Footnote 3 on Page 123 regarding feature normalization.

allows us to determine if we are simply coincidentally finding a better tree structure using all the data or if the actual predictions based on all the data aids us as well.

8.3.3 Data

For the experiments, we again use the MSN Web Directory (13 classes), the Reuters 21578 corpus (10 classes), and the TREC-AP corpus (20 classes). For a detailed description of the corpora, see Section 6.2.

8.3.4 Performance Measures

To compare the performance of the classification methods we look at a set of standard accuracy measures. The F1 measure [vR79, YL99] is the harmonic mean of precision and recall where $Precision = \frac{Correct\ Positives}{Predicted\ Positives}$ and $Recall = \frac{Correct\ Positives}{Actual\ Positives}$. Additionally, we report error, emphasizing the *normalized error* score. Normalized error divides the error in each task by the error that would have been achieved by random guessing (the *a priori* prevalent class). A normalized error less than one indicates the method outperforms random guessing. The scores reported here are the arithmetic averages of the values across all tasks (for F1 this is termed macroF1 in the text classification literature).

8.4 Experimental Results

Table 8.1 summarizes the performance of the systems over all 43 classification tasks. Better performance is indicated by larger F1 and by smaller error or normalized error values. The best performance in each column is given bold. To determine statistical significance for the macro-averaged measures, a one-sided macro sign test was performed [YL99]. When comparing system A and system B, the null hypothesis is that system A performs better on approximately half the tasks for which they differ in performance. The results for *LABEL-D (general)* are significantly better than the other systems at the $p = 0.01$ level with the exception of the difference between the error metrics of *LABEL-D (general)* and *STRIVE-D* which are significant at the $p = 0.05$ level.

8.5 Summary of Basic LABEL Approach

First, we note that the base classifiers are competitive and more particularly the results for *Smox* are consistent with the best reported results over these corpora [DC00, DPHS98, Joa98]. Thus, we are challenged with an extremely competitive baseline.

Method	Macro F1	Error	norm Error
Smox	0.7411	0.0197	0.4789
STRIVE-D	0.7457	0.0191	0.4716
LABEL-D (repop)	0.7431	0.0188	0.4758
LABEL-D (general)	**0.7545**	**0.0181**	**0.4512**

Table 8.1: Inductive Transfer Performance Summary over all Tasks

In spite of this, *LABEL-D (general)* shows dominance for each of the performance measures. Additionally, comparing it directly to the most comparable version of *STRIVE-D* reported in Bennett, Dumais, & Horvitz [BDH02, BDH05], we see improvement over the same system that uses data from each task in isolation. Additionally, by comparing *LABEL-D (general)* to *LABEL-D (repop)*, we see that it is not simply the structure of the resulting decision trees, but that the predicted probabilities induced across the entire set of tasks are key to improving generalization performance. While the percentage improvement is small, we believe these results are very encouraging for the future use of inductive transfer to improve models of classifier reliability.

Similar results are observed for each corpus individually but are more pronounced in the Reuters corpus (which has the classes with the fewest number of positive examples) than the MSN Web or TREC-AP corpora.

8.6 Future Work

There are several interesting avenues for future work. First, one could conduct experiments which explore parametric variations of m to control how much weight task-specific data is given versus the weight given to data from across all tasks. Secondly, while we have only empirically investigated the decision-tree metaclassifier, one could pursue other classifiers as a metaclassifier. For example using an SVM as done for *STRIVE* would be of considerable interest. The analogy for m for an SVM would be the c parameter which controls the amount of regularization for a problem. In fact, we hypothesize that not only would c play an important role in specializing the metaclassifier for a task, but that setting c while building the general model will be crucial because of how c acts as a regularizer. For implementations like SVMLight that have methods for automatically setting c given a set of data, then a rule for setting c for the general model can be determined in terms of the c that would be used for each individual dataset (*e.g.*, the sum of the auto-determined c's for the individual problems).

While we have demonstrated here that data can be used in conjunction across all tasks to improve average performance, another interesting question is whether data can be used in isolation to improve performance on a separate class. That is, if we left one corpus out and trained a general model on the remaining corpora to test on the left-out corpus, how effectively would that model transfer compared to the one trained in conjunction?

Finally, a potentially promising area to pursue is the inclusion of task-dependent reliability-indicator variables while building the general model. We discuss a few examples in Section 5.3 including: *NumTrainingPoints*, *%TrainingPointsIn*$\{$*Positive*$\}$, and *NumberOfSupport-Vectors*. In addition, we can include task identifiers with each example. Together these variables can allow the general model to both model a task separately if it improves model fit as well as model behavior that varies across tasks — such as how likely a given base classifier is to perform well given a certain number of positive training examples.

8.7 Summary

In this chapter, we demonstrated how Layered Abstraction-Based Ensemble Learning can be used to reduce the demand for training data by using data in conjunction across tasks. Furthermore, we conducted a small empirical analysis to demonstrate the validity of the approach. In particular, we demonstrated how we could improve upon a *STRIVE* model using a decision-tree metaclassifier by using all the data across the tasks. Finally, this area contains many interesting directions for future work, and we provided the reader with a discussion of some of the more interesting avenues.

Chapter 9

Online Methods and Regret

Researchers have been able to prove a variety of performance guarantees for online prediction methods. While our primary concern is offline or batch prediction, a natural question is whether these online methods can also achieve good empirical performance on batch prediction with little modification.

This chapter presents a brief overview of key existing theoretical results for combining classifiers in an online framework while maintaining (*regret*) performance guarantees. In this chapter, we evaluate the empirical performance of several of these online algorithms in a batch setting and consider whether they are an attractive alternative to the metaclassifiers we have already discussed. Finally, in light of our empirical results, we also analyze what types of regret guarantees are most desirable to yield a combination that performs well in practice.

9.1 Online Learning

In the batch setting, there is a training phase over a set of (labeled) training examples and then a testing phase where the learner receives no further feedback about the label of the examples. In contrast, an online learner receives constant feedback after every example. This feedback can either provide *full-information* about the cost of all alternatives not actually explored or *partial-information* by specifying only the loss of the action the learner took.

For classification where a loss function is specified, giving the learner the true class label is an example of the full-information model. In a multiclass problem or for example-specific cost-sensitive learning, specifying the loss of the predicted class but not specifying what the loss would have been for predicting the other class labels is an example of the *partial-information* model. We will assume we are operating in the full-information model with a specified loss function and that our feedback comes as the true class label. Thus, at each timepoint, an example is presented to the learner for prediction, after the learner receives the true label of the example, it incurs some loss, and updates its model.

More formally for each timestep $t = 1, \ldots, T$, the learning algorithm, A receives an example $\mathbf{x}_t$. Then, the learner makes a prediction, $\hat{y}_{A,t}$. After predicting, the learner is notified of the correct class y_t and suffers loss $L(\hat{y}_{A,t}, y_t)$ for some specified loss function L. When the particular learning algorithm is clear in context we will use $\hat{y}_t$ instead of $\hat{y}_{A,t}$. The cumulative loss the algorithm suffers is $\sum_{t=1}^{T} L(\hat{y}_t, y_t)$, and we will be concerned with particular algorithms that can bound the cumulative loss relative to some other algorithm. The simplicity of the online learning framework has allowed results to be derived in simple cases where the sequence of examples is assumed to be *i.i.d.* from a fixed distribution as well as when they are chosen from a non-fixed distribution by an adversary. Given the strength of these results, it is tempting to see how well they fare in the context of our problem.

9.2 Regret and Combining Classifiers

In addition to the focus of empirically-based researchers, the problem of combining expert advice or predictions has long drawn the attention of the theoretical part of the machine learning community. Quite often, combination algorithms can be theoretically justified with loss bounds, and furthermore include in their design indicator variables which signal a particular expert is abstaining or lacking confidence in some way. However, with few exceptions [FMS04], they seldom address when a classifier should abstain in practice. A simple lack of confidence is not sufficient since that classifier may still provide more information than any other classifier. Instead, it is preferable for the metaclassifier or *decision maker* to determine on an example-by-example basis how strongly each base classifier will contribute to the final decision. In this view, the decision maker simply treats each indicator variable as a possibly noisy hint to use a different combination function than what may be best overall, but she is free to ignore the indicator variable if it appears to be of little value.

As a result, there is often a gap between methods that have theoretical guarantees and methods that perform the best in practice. In this section, we review some of the more relevant and common combination algorithms that have performance guarantees. We use the discussion to motivate important criteria to consider when employing a combination algorithm, and discuss why the presented algorithms often underperform traditional metaclassifier approaches such as stacking.

Two basic questions commonly drive the design of a metaclassification algorithm: (1) Can the combiner ever win big? (2) Can we prevent the combiner from hurting performance too badly on any single problem?

An affirmative answer to the first question ensures that some users of the algorithm will gain far more by using our algorithm instead of simply using a base classifier, while an affirmative answer for the second question ensures that those who do use our algorithm will not *regret* it too much. Bounds for regret are formalized mathematically relative to a specific class of alternative algorithms. *External* regret compares performance to a fixed policy that is not dependent on the choices the combiner makes while *internal* regret considers alternatives that may be slight modifications of the combiner's choices [BM05]. For example, typical external regret bounds limit the combiner's loss relative to the loss when using the single best expert to predict. A tight external regret bound guarantees the user of the algorithm that even though the best base classifier cannot always be determined reliably, the user cannot have a performance much worse than if she would have known the base classifier performance a priori. Internal regret bounds guarantee small loss relative to simple changes to the algorithm, for example: "When the distance to the SVM's normal was less than 1, I should have placed the weight I placed on the SVM on the Decision Tree classifier instead.".

Our desire to examine regret stems from two basic types of practical combination problems. In the first, as has been the case throughout this dissertation, a machine learning practitioner has models from different learning algorithms (*e.g.*, Decision Trees, SVMs, kNN), and while aware of their relative performance, he is unsure whether they offer distinct information that can be effectively combined by a metaclassifier into a model which outperforms the base components. Furthermore, the practitioner has a set of candidate indicator variables such as the size of the neighborhood around the prediction point for kNN or the variance in naïve Bayes confidence that may indicate when classifiers are more or less reliable. In this case, it is possible many of the indicators are not actually tied to the base classifier's performance and the metaclassifier should generalize appropriately.

In the second motivating application, the classifiers are trained over disjoint training sets[1] that cannot be shared because of proprietary or data-privacy concerns, but the predictions of all the classifiers can be obtained for a common validation and test set. The indicators in this case may capture properties such as the similarity score of the most similar in example in each training set. A good bound with respect to the weighted combination

[1]These training sets might be drawn from different distributions, but we assume the labeling is consistent. That is, $p(\mathbf{x}, y)$ may vary according to training set, but presumably $P(y \mid \mathbf{x})$ does not.

of these classifiers is equivalent to saying we are taking effective advantage of the information in the proprietary data while maintaining the privacy constraints.

In both cases, we would like to have some guarantee that the classifiers are being combined well and that the indicators are being effectively used. It is with these goals in mind that we turn to a discussion of a few key algorithms with performance guarantees.

9.3 Combination Algorithms with Regret Guarantees

The majority of algorithms with performance guarantees have been developed in an online setting but can also be applied in a batch setting. Although the guarantees may not directly apply in the batch setting they form a foundation for understanding the impact of the algorithms.

One of the oldest algorithms in this category is the Weighted Majority Algorithm (WMA). Littlestone and Warmuth [LW94] present several variants and theoretical results for them. WMA is related to the halving algorithm which at every point throws out half of the remaining consistent hypotheses based on which half errs on the current example, but instead of requiring an expert or hypothesis to be perfect, WMA maintains a weight on each expert that is modified based on its performance. In its most basic form, the weight on each expert is initialized to 1. To update the weights, whenever the WMA algorithm makes a mistake, the experts that were incorrect have their weights multiplied by β where β is a parameter such that $0 < \beta < 1$. Let ϵ_H be the number of errors committed by WMA so far, ϵ_i be the number of errors by expert i, and n the number of experts. Then for an online prediction setting, it can be shown [LW94]:

$$\epsilon_H \leq \frac{\log n + \epsilon_i \log \frac{1}{\beta}}{\log \frac{2}{1+\beta}} \tag{9.1}$$

For a value of $\beta = 0.5$, this yields $\epsilon_H \leq 2.41(\epsilon_i + \lg n)$. In other words, the combiner is within a small factor of the best expert and a logarithmic factor of the number of experts.

A very closely related algorithm to WMA is the Winnow algorithm. The Winnow algorithm [Lit88] is one of the most well-known algorithms for learning a threshold function of boolean inputs and mistake bounds have been derived for several variants of it. Blum [Blu97] introduced a variant of the algorithm, Winnow-Specialists, specifically designed for combining experts. Instead of requiring an expert to always make a prediction, experts are allowed to abstain or “sleep”. The algorithm predicts using a weighted combination of the experts that do not abstain. When updating the weights, only the weight of the experts that did not abstain are changed. When the combiner is incorrect[2], the classifier increases the weight on the experts voting correctly by $\frac{3}{2}$ and decreases the weight on the incorrect experts by $\frac{1}{2}$. If the n experts contain a subset of r infallible experts where at least one (possibly different) does not abstain on every example, then the online combiner's mistakes are limited as:

$$\epsilon_H \leq 2r \log_{\frac{3}{2}} 3n. \qquad (9.2)$$

The key difference of the sleeping experts formulation is that experts are not penalized if they know they are unsure and can abstain. Likewise, it makes working with an extremely large number of experts computationally efficient as long as only a small subset are awake for any particular example. Cohen & Singer [CS99] exploited this when they applied the algorithm as a base classifier to topic classification where each expert was a word or phrase; thus there were a large number of experts but only a small number (those present in the document) made predictions for any given document. Additionally, Blum performs an empirical evaluation using variants of WMA and Winnow-Specialist.

Freund *et al.* [FSSW97] introduce a generalization of the sleeping experts framework that demonstrates how to convert an expert combination algorithm which uses all awake experts to a sleeping expert combination that allows abstaining. The results they derive remove the restrictions that a subset of the n experts be infallible, and they derive a variety of theoretical results for various algorithms. We note this only to point out that the specialist algorithms are more robust than what might seem from the conditions on Blum's result.

Of course, returning to our motivation for this problem, our problem is that we don't know when we should put our experts to "sleep". We have predictions over all examples and we have indicators that we suspect might indicate a different weight on the experts is preferable. Thus, an algorithm that specifies how to use indicator variables would be preferable. Blum & Mansour [BM05] introduce such an algorithm for the case where the indicators are in $[0, 1]$ that generalizes the sleeping expert setting. The essential idea is that an indicator value of zero indicates abstaining and a value of one indicates fully voting. It is then up to the algorithm to learn which experts are best when a given indicator is on.

First, we present the basic framework. Let $\mathcal{I}$ be the set of indicators and $I \in \mathcal{I}$ be a particular indicator. Again, working in an online adversarial setting, $I(t)$ will denote the value that an indicator I takes at time t. Let l_e^t be the loss of expert e at time t where $l_e^t \in [0, 1]$. The combiner H will operate by specifying a probability distribution at time t, $\mathbf{p}^t$, over the experts, and the combiner suffers a loss equal to the expected loss: $l_H^t = \sum_{e=1}^{n} p_e^t l_e^t$.

[2]Blum points out it is possible to penalize the ones that are incorrect when the combiner is correct, but that for the proof to go through they can only be rewarded when the combiner is wrong.

Their algorithm is based on the idea that we keep a weight for each indicator/base classifier pair, and then compute the combiner's probability vector over experts by normalizing the linear combination of these weights and the indicator variables. First, each indicator-expert pair weight is initialized to one, $w_{e,I}^{t=1} = 1$. Then to compute the weight given to each expert, we compute the linear combination of the current weights according to how "on" each indicator is: $w_e^t = \sum_{I \in \mathcal{I}} I(t) w_{e,I}^t$. Next, we compute a normalizing term, $W^t = \sum_{e=1}^{n} w_e^t$, and produce a probability vector over the experts by normalizing the weight distribution, $p_e^t = \frac{w_e^t}{W^t}$. Prediction can either be performed by randomly choosing an expert according to the distribution p_e^t or by mixing the experts' predictions using this probability vector.[3] To update the weights we perform a multiplicative update based on how much better the expert was than the combiner and how "on" the particular indicator is $w_{e,I}^{t+1} = w_{e,I}^t \beta^{I(t)(l_e^t - \beta l_H^t)}$, where $\beta \in (0,1)$ is a parameter of the algorithm.

If we define the cumulative loss of an algorithm a under an indicator I as $\sum_{t=1}^{T} I(t) L_a$, then this algorithm achieves good external regret bounds with respect to all of the indicators. In particular, where m is the number of indicator variables :

$$L_{H,I} \leq \frac{L_{e,I} + (\log nm)/\log(1/\beta)}{\beta} \tag{9.3}$$

Note that by substituting in an "always on" indicator, typical regret bounds are achieved for the best expert problem.

Since this seems a promising comparison point, we examine this algorithm to determine its empirical performance and appropriateness for use throughout the rest of the dissertation. We refer to this algorithm as BMX since Blum & Mansour introduce it and prove external regret bounds using it. They also introduce a general method for converting external regret guarantees to internal regret guarantees and present a particular algorithm with internal regret guarantees [BM05]. We note this for the interested reader but do not present these as comparisons because we feel they still do not address the issues most pertinent to us.

For classification, the experts can give us estimates of the posterior probabilities or class predictions. We will use $\boldsymbol{\pi_{t,e}}$ to denote expert e's vector of posterior probability estimates over classes at time t. Likewise, we will use $\boldsymbol{\Pi_t}$ to denote the matrix composed of all the expert's probability vectors $\boldsymbol{\pi_{t,e}}$. Finally, $\hat{y}_{t,e}$ is the class prediction of expert e at time t. When a class prediction is taken from a posterior estimate, we assume $\hat{y}_{t,e} = \operatorname{argmax}_c \boldsymbol{\pi_{t,e}}[c]$.

[3]Blum & Mansour's analysis proceeds by randomly choosing an expert according to the vector since this allows for application to problems outside of classification. For our purposes, either can be done and a further discussion of this issue is below.

9.4 Empirical Analysis

As done throughout this dissertation and described in more detail in Sections 6.3 and 7.2 we selected five classifiers as base classifiers or input experts for the combination algorithms: kNN, decision trees, linear SVMs, naïve Bayes, and a unigram classifier. We denote these below as *kNN*, *Dnet*, *SVM*, *naï ve Bayes*, and *Unigram*.

We require the outputs of the base classifiers to train the metaclassifiers. Thus, we perform cross-validation over the training data and use the resulting base classifier predictions, obtained when an example serves as a validation item, as training inputs for the metaclassifier. As described in Section 6.3, each of the classifiers outputs a score that can be interpreted as an estimate of the log-odds for the example. We convert these scores to a probability by treating them as if they were the log-odds.

We present results here for the algorithms applied to two corpora, the Reuters corpus and the MSN Web Directory. As a reminder from Chapter 7, the Reuters corpus yielded less improvement for the combiners than was achieved in the MSN Web corpus. Thus, in judging whether these online algorithms are indeed attractive alternatives to using a linear SVM as a metaclassifier, these corpora form two useful sample points. We present only the results for *Stack-S (norm)* and *STRIVE-S (norm)* here for comparison since those were the most competitive.[4]

As done in the earlier experiments, for the experiments below, we used only the top 1000 words with highest mutual information for the MSN Web Directory and the top 300 words for Reuters for all base classifiers except the kNN classifier. Since the kNN classifier is computationally expensive, we desired to use the same feature representation across binary classification tasks within a corpus. Once neighbors are retrieved, the kNN classifier can make all class decisions quickly. As is commonly done (e.g. [LYRL04]), for each word we assigned a score of the maximum of the mutual information scores across binary tasks. The top features were then taken across these maximum scores. Since the same feature set was being used for all classes within a corpus, we used $3\times$ the number of features — 3000 words for MSN Web and 900 for Reuters. These settings are exactly as in our earlier experiments to perform a fair comparison. The corpora are described in further detail in Section 6.2.

To compare the performance of the classification methods we look at a set of standard performance measures: the macro-averaged F1, micro-averaged F1, error, two linear utility

[4]Although we note that for some of the ROC measures *STRIVE-D* variants were the most competitive. Since it is not pertinent to this discussion, we omit it here for brevity and refer the reader back to Chapter 7 if they have further interest.

functions — $C(10,1)$ and $C(1,10)$, the area under the ROC curve, as well as the abbreviated area under the ROC curve. These measures are described in more detail in Section 6.1 and in Chapter 7.

9.4.1 Combination Implementations

For most of the online algorithms, there are several common variants — both in the online learning case and when applied for batch learning. In addition, each algorithms has several parameters. In order to enable reproducibility, we give pseudocode for each algorithm and specify what parameter settings we use.

One common variant for converting an online algorithm to a batch setting is to make multiple passes through the training data and relax the award/penalty parameter β after each pass by driving the value of β toward 1. We perform this relaxation for all of the algorithms and present results for a single pass through the data and 10 passes through the data.

WMA

The implementation of WMA we use is essentially the WMG variant from Littlestone & Warmuth 1994 [LW94] but we use squared-difference in posterior probability for the weight updates instead of absolute difference. As done in [Blu97], we use $\beta = 0.5$. Blum notes that the algorithm showed little empirical sensitivity to the particular value of β. For those runs using more than one iteration over the training set, we use $\eta = 0.25$. It is common to include $|C| - 1$ experts that predict class c with probability 1 to allow the algorithm to automatically adjust a threshold. In addition to the base classifiers, we include these "constant experts" as well. Pseudocode for the prediction and training algorithms are given in Algorithms 9.4.1 and 9.4.2.

Algorithm 9.4.1: $\mathit{WMA\_predict}(n, \mathbf{w}, \mathbf{\Pi})$

```
// Preconditions: $\pi_e[c] \in [0,1]$. $1 = \sum_c \pi_e[c]$. For some $e, w_e \neq 0$.
$p_e = \frac{w_e}{\sum_{e=1}^{n} w_e}$  // weight combiner places on $e$
$\boldsymbol{\pi_H} = \sum_{e=1}^{n} p_e \boldsymbol{\pi_e}$
$\hat{y}_H = \mathrm{argmax}_c\, \pi_H[c]$
output $(\boldsymbol{\pi_H}, \hat{y}_H)$
```

Algorithm 9.4.2: $batch\_train\_WMA(R, n, \langle\langle \mathbf{\Pi}_1, y_1\rangle, \ldots, \langle \mathbf{\Pi}_T, y_T\rangle\rangle, \eta, \beta)$

```
// Preconditions: $\pi_{t,e}[c] \in [0,1]$. $1 = \sum_c \pi_{t,e}[c]$. $\beta, \eta \in (0,1)$.
w $\leftarrow \vec{1}^n$
for $r \leftarrow 1$ to $R$
  do
    for $t \leftarrow 1$ to $T$
      do
        $\boldsymbol{\pi}_{t,H}, \hat{y}_{t,H} \leftarrow WMA\_predict(n, \mathbf{w}, \mathbf{\Pi_t})$
        $P(c = y_t \mid x_t) \leftarrow 1$
        $P(c \neq y_t \mid x_t) \leftarrow 0$
        $w_e \leftarrow w_e \beta^{(\pi_e[y_t] - P(y_t|x_t))^2}$
    // Relax $\beta$ by driving it toward $1$.
    $\beta \leftarrow \beta + \eta(1 - \beta)$
output $(\mathbf{w})$
```

Winnow

The implementation of Winnow we use is essentially the same as that empirically explored by Blum [Blu97] in his variant of the original Winnow2 algorithm [LW94] for sleeping-experts. Since by default all of the experts are awake for our problem, the main differences from *WMA* are that *Winnow* only updates the weights on the experts when the combination algorithm makes a mistake and that the amount of weight change is based on 0/1 loss.

Two minor differences from Blum's implementation is that Blum promotes by $(1 + \beta)$ instead of β^{-1} for theoretical reasons but notes that empirical results show no significant impact from this change. Second, Blum's original formulation did not change the weights on the sleeping experts, but also did not ensure that the *probability* placed on those experts did not change as required in [FSSW97]. The formulation we give follows [FSSW97] in that the *sum* of the weights on the awake specialists is constant before and after an update. Since we only apply this when all experts are awake, it does not change our results. We simply note this for the reader interested in reapplying the algorithm.

Finally, as done in [Blu97], we use $\beta = 0.5$. Again, Blum notes that little empirical sensitivity was shown to the particular value of β. For those runs using more than one iteration over the training set, we use $\eta = 0.25$. We predict the most likely class instead of using a threshold parameter and again include the use of a set of $|C| - 1$ constant experts. Since we apply this for binary classification, this means we use a single constant always-awake expert that predicts class 1.

Pseudocode for the prediction and training algorithms for an implementation that performs n-way classification are given in Algorithms 9.4.3 and 9.4.4.

Algorithm 9.4.3: $\mathit{Winnow\_Specialists\_predict}(n, \mathbf{w}, \mathbf{\Pi}, \mathbf{A})$

```
// Preconditions: $A_e \in \{0,1\}$. For some $e, A_e \neq 0$ and $w_e \neq 0$. $\pi_e[c] \in [0,1]$. $1 = \sum_c \pi_e[c]$.
$p_e = \frac{A_e w_e}{\sum_{e=1}^{n} A_e w_e}$ // weight combiner places on $e$
$\boldsymbol{\pi_H} = \sum_{e=1}^{n} p_e \boldsymbol{\pi_e}$
$\hat{y}_H = \text{argmax}_c\, \pi_H[c]$
output $(\boldsymbol{\pi_H}, \hat{y}_H)$
```

Algorithm 9.4.4: $\mathit{batch\_train\_Winnow\_Specialists}(R, n, \langle\langle \mathbf{\Pi}_1, y_1\rangle, \ldots, \langle \mathbf{\Pi}_T, y_T\rangle\rangle, \eta, \beta)$

```
// Preconditions: $\pi_{t,e}[c] \in [0,1], 1 = \sum_c \pi_{t,e}[c]$. $\beta, \eta \in (0,1)$.
$\mathbf{w} \leftarrow \vec{1}^n$
for $r \leftarrow 1$ to $R$
  do
    for $t \leftarrow 1$ to $T$
      do
        $\mathbf{A} \leftarrow \vec{0}^n$
        if ($e$ is awake) then
          $A_e \leftarrow 1$
        $\boldsymbol{\pi}_{t,H}, \hat{y}_{t,H} \leftarrow \mathit{Winnow\_Specialists\_predict}(n, \mathbf{w}, \mathbf{\Pi_t}, \mathbf{A})$
        if $(y_t \neq \hat{y}_{t,H})$ then
          if ($e$ is awake) then
            $w_e \leftarrow w_e \beta^{2*\mathbb{1}(\hat{y}_{t,e} \neq y_t)-1} \frac{\sum_e A_e w_e}{\sum_e A_e w_e \beta^{2*\mathbb{1}(\hat{y}_{t,e} \neq y_t)-1}}$
    // Relax $\beta$ by driving it toward 1.
    $\beta \leftarrow \beta + \eta(1-\beta)$
output $(\mathbf{w})$
```

BMX

For the BMX algorithm, we need indicators in the $[0,1]$ interval to use. For this purpose, we use the set of reliability indicators described in Chapter 5. To map these indicators to the zero-one interval, we range-normalize each reliability indicator. That is, features that take a value less than the minimum ($\min_i$ for feature i) observed value in the training set are mapped to zero. Those greater than the maximum are mapped to one ($\max_i$ for feature i). For each remaining feature i a value v_i is mapped to $\frac{v_i - \min_i}{\max_i - \min_i}$. Additionally, we include the probabilities of "in-class" from the classifiers as indicators.

Our implementation of the BMX algorithm is exactly as that described in [BM05] except for one change in prediction. In the original, BMX uses p_e to randomly draw an expert and predicts with that expert's prediction. We use p_e to mix the posterior probabilities of the experts. This does not affect the training of the model. It only alters the predictions over the test set. In our experience, this always was a better alternative to randomly drawing an example.

For comparability to the other algorithms, we use $\beta = 0.5$. For those runs using more than one iteration over the training set, we use $\eta = 0.25$. As a loss function, we use squared error. We predict the most likely class instead of using a threshold parameter and again include the use of a set of $|C| - 1$ constant experts. Since we apply this for binary classification, this means we use a single constant expert that predicts class 1 with probability 1.

Since we are primarily interested in the suitability of the BMX algorithm as an alternative to other approaches to using reliability indicators, we provide greater detail in the pseudocode than for the other algorithms. Pseudocode for the prediction and training algorithms for an implementation that performs n-way classification are given in Algorithms 9.4.5-9.4.7.

We also present results for a modified version of the BMX algorithm which we refer to as *BMXmod* and differs in two points from BMX. We observed over holdout data that the indicators in BMX can occasionally act more like “importance functions” than “reliability indicators”. For example, when the sum across all of the indicators is high, that training example can cause a very large shift in the weights while another example that has a low sum causes very little change. To help prevent this, *BMXmod* (L1) normalizes the sum of the indicator variables to be 1. Second, the BMX algorithm computes the loss of the combination algorithm as the weighted combination (using p_e) of the loss of the experts. *BMXmod* directly computes the loss of the combiner by setting it to be the squared error of the combiner. Because of this change, using p_e to mix the expert’s estimates instead of randomly drawing an expert also affects the training of the model in *BMXmod* as well as the prediction.

Finally, the BMX algorithm updates the weights for nearly every training example. An alternative approach to consider is, like Winnow, to only update the weights if the combiner makes an error in prediction. To do so, we introduce a variant of *BMX* and *BMXmod* which we refer to as *WinBMX* and *WinBMXmod*, respectively. These algorithms only update the weights if the combiner’s log-odds of the correct class is less than one. Thus, this acts as a margin and for examples that are only slightly correct, we continue to update the weights.

Algorithm 9.4.5: $BMX\_predict(n, \mathbf{w}, \mathbf{I}, \mathbf{\Pi})$

```
// Preconditions: $I \in [0,1]$. $w_{e,I} \geq 0$. For some $(I,e), I \neq 0$ and $w_{e,I} \neq 0$.
// $\pi_e[c] \in [0,1]$. $1 = \sum_c \pi_e[c]$.
$\mathbf{w}' \leftarrow \vec{0}^n$
$W \leftarrow 0$
for $I \in \mathbf{I}$
  do for $e \leftarrow 1$ to $n$
  do $\begin{cases} w'_e \leftarrow I * w_{e,I} + w'_e \\ W \leftarrow W + w'_e \end{cases}$
for $e \leftarrow 1$ to $n$
  do $p_e = \frac{w'_e}{W}$ // weight combiner places on $e$
$\boldsymbol{\pi_H} = \sum_{e=1}^{n} p_e \boldsymbol{\pi_e}$
$\hat{y}_H = \mathrm{argmax}_c\, \pi_H[c]$
output $(\boldsymbol{\pi_H}, \hat{y}_H, \mathbf{p})$
```

Algorithm 9.4.6: $BMX\_update\_weights(n, \mathbf{w}, \mathbf{I}, \mathbf{L}, L_H, norm)$

```
// Preconditions: $I \in [0,1]$. $w_{e,I} \geq 0$. $L, L_H \in [0,1]$. $norm \in \{0,1\}$.
$c \leftarrow 1$
if $(norm)$ then
  {
  $old \leftarrow 0$
  $new \leftarrow 0$
  for $e \leftarrow 1$ to $n$
    do for $I \in \mathbf{I}$
    do if $(I \neq 0)$ then
    $\begin{cases} old \leftarrow old + w_{e,I} \\ new \leftarrow new + w_{e,I}\beta^{I(L_e - \beta L_H)} \end{cases}$
  if $(old \neq 0)$ then
   $c \leftarrow \frac{old}{new}$
  }
for $e \leftarrow 1$ to $n$
  do for $I \in \mathbf{I}$
  do if $(I \neq 0)$ then
 $w_{e,I} \leftarrow w_{e,I} c \beta^{I(L_e - \beta L_H)}$
output $(\mathbf{w}')$
```

Algorithm 9.4.7: $batch\_train\_BMX(R, n, \mathcal{L}, \langle\langle \mathbf{\Pi}_1, \mathbf{I}_1, y_1\rangle, \ldots, \langle \mathbf{\Pi}_T, \mathbf{I}_T, y_T\rangle\rangle, \eta, \beta)$

```
// Preconditions: $\pi_{t,e}[c], I_t \in [0,1], 1 = \sum_c \pi_{t,e}[c]$. $\beta, \eta \in (0,1)$.
//                $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^n \times \mathcal{Y} \times \mathcal{Y} \to [0,1]$.
$\mathbf{w} \leftarrow \vec{1}^n \times \vec{1}^{|\mathbf{I}|}$
for $r \leftarrow 1$ to $R$
  do
    for $t \leftarrow 1$ to $T$
      do
        $\boldsymbol{\pi}_{t,H}, \hat{y}_{t,H}, \mathbf{p} \leftarrow BMX\_predict(n, \mathbf{w}, \mathbf{\Pi}_{\boldsymbol{t}})$
        $P(c = y_t \mid x_t) \leftarrow 1$
        $P(c \neq y_t \mid x_t) \leftarrow 0$
        $L_t \leftarrow \vec{0}^n$
        $L_{t,H} \leftarrow 0$
        for $e \leftarrow 1$ to $n$
          do
            $L_{t,e} \leftarrow \mathcal{L}(\mathbf{P}(c \mid x_t), \mathbf{\Pi}_{\boldsymbol{t,e}}, y_t, \hat{y}_{t,e})$ // get loss of expert
            $L_{t,H} \leftarrow L_{t,H} + p_e L_{t,e}$ // Combiner's loss depends on $e$'s weight
    // Relax $\beta$ by driving it toward $1$.
    $\beta \leftarrow \beta + \eta(1 - \beta)$
output $(\mathbf{w})$
```

9.4.2 Results and Discussion

The results for the MSN Web corpus are presented in Table 9.1. The results for the Reuters corpus are presented in Table 9.2. We note that the Winnow and WMA variants use only the classifier outputs, and thus we are interested in them as an alternative metaclassifier in stacking to *Stack-S (norm)*. In contrast, the BMX variants use the indicators, and thus we are interested in them as an alternative metaclassifier in the striving framework to *STRIVE-S (norm)*.

First, we note that unlike *Stack-S (norm)*, WMA and Winnow often do not outperform even the base classifiers and rarely significantly. Of these options the clear best choice in both corpora is *Winnow* ($R = 10$) — Winnow using a relaxed β over multiple iterations. We also note that relaxation with multiple passes is clearly important to the implementation of Winnow here. There are two possible reasons why Winnow requires multiple passes. The first is since updates are only performed when the combiner makes a mistake, convergence to an optimal weight set is slow. The second possibility is that the convergence is slow as a result of the update for Winnow being based on 0/1 loss instead of squared loss. However, comparing *Winnow* ($R = 10$) to *Stack-S (norm)* over both corpora, even this variant is not a superior or even comparable alternative to *Stack-S (norm)*.

In contrast, the indicator-based online methods, in particular the WinBMX variants, often outperform the base classifiers. The WinBMX variants outperform the base classifiers relatively frequently in the Reuters corpus and very often in the MSN Web corpus. The relaxation with multiple iterations does not seem to be critical to any of the indicator-based online methods. Since WinBMX uses the same weight update (*i.e.*, squared-loss based), this makes it more likely that the slow convergence observed above for Winnow was a result of the update weight and not the number of updates.

Next, while each WinBMX variant does not always beat its BMX pairing, each does so consistently enough that WinBMX is clearly the superior choice to BMX. While the modification in the *mod* variants have some impact, this impact does not appear to be consistent across all performance measures, and thus these modifications are probably not desirable in all cases.

In comparing the indicator-based online alternatives to *STRIVE-S (norm)*, we see that *WinBMX* stays competitive with *STRIVE-S (norm)* in Reuters where striving does not yield as large of an improvement. However, in the MSN Web Corpus where striving obtains a much larger improvement, *WinBMX* and all of the indicator-based online methods are significantly beaten by *STRIVE-S (norm)* according to nearly every performance measure. Thus, it seems that indicator-based online methods are not able to utilize the indicators as effectively.

However, comparing the indicator-based online methods and in particular *WinBMX* to the basic methods, we see that the indicator-based methods do generally improve over the basic online methods. Thus, even among these methods that do not improve as much relative to the base classifiers as is desirable, the reliability indicators have an impact. Also, we note that interestingly *BMXMod* achieves the highest ROC area in the MSN Web corpus, although looking at the abbreviated area it does much worse. The fact that these methods are using the indicators, but not as well as desired, leaves some hope for future modifications. In the next section, we discuss those modifications we believe most likely to have an impact.

Finally, for the reader considering using the online methods, compared to the BMX variants, the WinBMX variants can be significantly more computationally efficient since the number of examples having weight updates is extremely small. Since every example for BMX has a prediction and update of roughly the same complexity, the Winnow variants experience at least a $2\times$ speed-up.

	MacroF1	MicroF1	Error	C(1,10)	C(10,1)	ROC Area	ROC [0,0.1]
Dnet	*0.5477*	0.5813	0.0584	0.3012	0.0772	*0.8802*	0.5638
Unigram	0.5982	0.6116	0.0594	0.2589	*0.0812*	0.9003	0.6114
naï ve Bayes	0.5527	*0.5619*	*0.0649*	0.2853	0.0798	0.8915	*0.5516*
SVM	$0.6727^{\boldsymbol{B}}$	$0.7016^{\mathbb{B}}$	0.0455	$0.2250^{\boldsymbol{B}}$	0.0794	0.9123	$0.6960^{\boldsymbol{B}}$
kNN	0.6480	0.6866	0.0464	0.2524	0.0733	0.8873	0.6541
Best By Class	$0.6727^{\mathbb{D}}$	0.7016	$0.0452^{\boldsymbol{D}}$	0.2235	$0.0729^{\boldsymbol{D}}$	N/A	N/A
Majority	0.6643	0.6902	0.0479	$0.2133^{\mathrm{BD}}_{\mathrm{O}}$	0.0765	N/A	N/A
WMA	0.6324	0.6756	0.0472	0.2560	0.0746	0.8884	0.6465
WMA ($R = 10$)	0.6317	0.6757	0.0472	0.2563	0.0746	0.8865	0.6461
Winnow	0.5918	0.6369	0.0479	*0.3109*	0.0724	$0.9303^{\boldsymbol{B}}_{\boldsymbol{O}}$	$0.7204^{\boldsymbol{B}}_{\boldsymbol{O}}$
Winnow ($R = 10$)	$0.6668_{\mathbb{O}}$	$0.7033^{\mathbb{BD}}_{\mathbb{O}}$	0.0453	$0.2439_{\mathbb{O}}$	0.0780	$0.9166^{\boldsymbol{B}}$	$0.7068^{\boldsymbol{B}}$
BMX	0.6657	$0.7069^{\mathbb{BD}}_{\mathbb{O}}$	$0.0440^{\mathrm{BD}}_{\mathrm{O}}$	0.2194_{O}	0.0731	$0.9287^{\boldsymbol{B}}$	0.7053
BMX ($R = 10$)	0.6603	$0.6993^{\mathbb{BD}}$	0.0453	0.2351	0.0743	$0.9227^{\boldsymbol{B}}$	0.7038
BMXmod	0.6727	0.7031	$0.0435^{\boldsymbol{BD}}_{\boldsymbol{O}}$	$0.2158^{\boldsymbol{B}}_{\boldsymbol{O}}$	0.0717	$\mathbf{0.9355}^{\boldsymbol{B}}_{\boldsymbol{O}}$	$0.7122^{\boldsymbol{B}}$
BMXmod ($R = 10$)	0.6710	$0.7049^{\mathbb{BD}}$	$0.0436^{\boldsymbol{BD}}_{\boldsymbol{O}}$	$0.2125^{\boldsymbol{B}}_{\boldsymbol{O}}$	0.0717	$0.9347^{\boldsymbol{B}}_{\mathbb{O}}$	$0.7115^{\boldsymbol{B}}$
WinBMX	$0.6811^{\boldsymbol{BD}}_{\boldsymbol{O}}$	$0.7115^{\mathbb{BD}}_{\mathbb{O}}$	$0.0428^{\boldsymbol{BD}}_{\boldsymbol{O}}$	$0.2084^{\boldsymbol{B}}_{\boldsymbol{O}}$	0.0729	$0.9304^{\boldsymbol{B}}$	$0.7195^{\boldsymbol{B}}$
WinBMX ($R = 10$)	$0.6841^{\mathrm{BD}}_{\boldsymbol{O}}$	$0.7114^{\mathbb{BD}}_{\mathbb{O}}$	$0.0434^{\boldsymbol{BD}}_{\boldsymbol{O}}$	$0.2088^{\boldsymbol{B}}_{\boldsymbol{O}}$	0.0754	$0.9267^{\boldsymbol{B}}$	$0.7223^{\boldsymbol{B}}$
WinBMXmod	$0.6810^{\boldsymbol{BD}}_{\boldsymbol{O}}$	0.7082	$0.0434^{\boldsymbol{BD}}_{\boldsymbol{O}}$	$0.2111^{\boldsymbol{B}}_{\boldsymbol{O}}$	0.0725	$0.9348^{\boldsymbol{B}}_{\boldsymbol{O}}$	$0.7190^{\boldsymbol{B}}$
WinBMXmod ($R = 10$)	$0.6827^{\boldsymbol{BD}}_{\boldsymbol{O}}$	$0.7103^{\mathbb{BD}}$	$0.0432^{\boldsymbol{BD}}_{\boldsymbol{O}}$	$0.2082^{\boldsymbol{B}}_{\boldsymbol{O}}$	0.0731	$0.9324^{\boldsymbol{B}}_{\mathbb{O}}$	$0.7201^{\boldsymbol{B}}$
Stack-S (norm)	$0.6939^{\boldsymbol{BD}}_{\boldsymbol{O}}$	$0.7250^{\mathbb{BD}}_{\mathbb{OI}}$	$0.0423^{\boldsymbol{BD}}_{\boldsymbol{O}}$	$0.1971^{\boldsymbol{BD}}_{\boldsymbol{O}\mathrm{I}}$	$0.0705^{\mathbb{D}}$	$0.9334^{\boldsymbol{B}}$	$0.7349^{\boldsymbol{B}}_{\boldsymbol{OI}}$
STRIVE-S (norm)	$\mathbf{0.7173}^{\boldsymbol{BDS}}_{\boldsymbol{OI}}$	$\mathbf{0.7437}^{\mathbb{BDS}}_{\mathbb{OI}}$	$\mathbf{0.0392}^{\boldsymbol{BDS}}_{\boldsymbol{OI}}$	$\mathbf{0.1835}^{\boldsymbol{BD}\mathrm{S}}_{\boldsymbol{OI}}$	**0.0682**	$0.9260^{\mathbb{B}}$	$\mathbf{0.7547}^{\boldsymbol{BS}}_{\boldsymbol{OI}}$
BestSelect	0.8719	0.8924	0.0223	0.0642	0.0565	N/A	N/A

Table 9.1: Comparison of the Online Combiners over the MSN Web Corpus. The best performance (omitting the oracle *BestSelect*) in each column is given in bold. A notation of 'B', 'D', 'S', 'R', 'O', or 'I' indicates a method signifi cantly outperforms all (other) **B**ase classifi ers, **D**efault combiners, **S**tacking methods, **R**eliability-indicator based Striving methods, **O**nline basic methods, or **I**ndicator-based online methods at the $p = 0.05$ level. A blackboard (hollow) font is used to indicate signifi cance for the macro-sign test and micro-sign test. A normal font indicates signifi cance for the macro t-test. For the macro-averages (*i.e.*, excluding micro F1) when both tests are signifi cant it is indicated with a bold, italicized font.

	MacroF1	MicroF1	Error	C(1,10)	C(10,1)	ROC Area	ROC [0,0.1]
Dnet	0.7846	0.8541	0.0242	0.0799	*0.0537*	0.9804	0.8844
Unigram	0.7645	0.8674	0.0234	0.0713	0.0476	0.9877	0.9086
naï ve Bayes	*0.6574*	*0.7908*	*0.0320*	*0.1423*	0.0527	*0.9703*	*0.7841*
SVM	0.8545^{B}	$0.9122^{\mathbb{B}}$	0.0145^{B}	0.0499	0.0389	0.9893	0.9429^{B}
kNN	0.8097	0.8963	0.0170	0.0737	0.0336	0.9803	0.9043
Best By Class	0.8608	0.9149	0.0144	0.0496	0.0342	N/A	N/A
Majority	0.8498	0.9102	0.0155	0.0438	0.0437	N/A	N/A
WMA	0.8432	0.9064	0.0155	0.0595	0.0376	0.9890	0.9370
WMA ($R = 10$)	0.8443	0.9056	0.0154	0.0618	0.0365	0.9846	0.9235
Winnow	0.8398	0.9074	0.0158	0.0698	0.0405	$0.9946^{\mathbb{B}}$	$0.9549^{\mathbb{B}}$
Winnow ($R = 10$)	0.8727^{B}	$0.9208^{\mathbb{B}\mathbb{D}}_{\mathbb{O}}$	0.0141	$0.0439^{\mathbb{B}}_{\mathrm{O}}$	0.0339	$0.9937^{\mathbb{B}}$	$0.9537^{\boldsymbol{B}}$
BMX	0.8640	0.9132	0.0144	0.0553	0.0365	0.9941	0.9487
BMX ($R = 10$)	0.8581	0.9109	0.0146	0.0576	0.0403	0.9928	0.9408
BMXmod	0.8616	0.9166	0.0141	$0.0402^{\boldsymbol{B}}$	0.0334	$0.9952^{\boldsymbol{B}}$	$0.9562^{\boldsymbol{B}}$
BMXmod ($R = 10$)	0.8643	0.9166	0.0140	$0.0403^{\boldsymbol{B}}$	0.0324	$0.9951^{\boldsymbol{B}}$	$0.9565^{\boldsymbol{B}}$
WinBMX	$0.8843^{\boldsymbol{B}\mathbb{D}}$	$0.9232^{\mathbb{B}\mathbb{D}}$	$0.0134^{\boldsymbol{B}\mathbb{D}}$	$0.0426^{\mathbb{B}}$	0.0363	$0.9933^{\boldsymbol{B}}$	$0.9545^{\boldsymbol{B}}$
WinBMX ($R = 10$)	0.8606	0.9122	0.0147	$0.0467^{\mathbb{B}}$	0.0379	0.9896	0.9453
WinBMXmod	0.8691	$0.9225^{\mathbb{B}\mathbb{D}}$	0.0132^{BD}	$0.0386^{\boldsymbol{B}\mathbb{D}}_{\mathrm{O}}$	**0.0318**	$0.9955^{\boldsymbol{B}}$	$0.9598^{\boldsymbol{B}}$
WinBMXmod ($R = 10$)	0.8747	$0.9232^{\mathbb{B}\mathbb{D}}$	0.0125^{BD}	$0.0388^{\boldsymbol{B}}$	0.0332	$0.9955^{\boldsymbol{B}}_{\mathbb{O}}$	$0.9612^{\boldsymbol{B}}_{\mathbb{O}}$
Stack-S (norm)	$\mathbf{0.8908}^{\mathrm{BD}}$	$\mathbf{0.9307}^{\mathbb{B}\mathbb{D}}_{\mathbb{O}\mathbb{I}}$	$0.0125^{\boldsymbol{BD}}$	$0.0372^{\boldsymbol{B}\mathbb{D}}_{\boldsymbol{O}}$	0.0331	$\mathbf{0.9956}^{\boldsymbol{B}}_{\mathbb{O}}$	$\mathbf{0.9628}^{\boldsymbol{B}}$
STRIVE-S (norm)	0.8835^{BD}	$0.9287^{\mathbb{B}\mathbb{D}}_{\mathbb{O}}$	$\mathbf{0.0121}^{\mathrm{BD}}$	$\mathbf{0.0352}^{\boldsymbol{B}\mathbb{D}}_{\boldsymbol{O}}$	0.0343	$0.9948^{\boldsymbol{B}}$	$0.9616^{\boldsymbol{B}}$
BestSelect	0.9611	0.9789	0.0036	0.0073	0.0173	N/A	N/A

Table 9.2: Comparison of the Online Combiners over Reuters. The best performance (omitting the oracle *BestSelect*) in each column is given in bold. A notation of 'B', 'D', 'S', 'R', 'O', or 'I' indicates a method signifi cantly outperforms all (other) **B**ase classifi ers, **D**efault combiners, **S**tacking methods, **R**eliability-indicator based Striving methods, **O**nline basic methods, or **I**ndicator-based online methods at the $p = 0.05$ level. A blackboard (hollow) font is used to indicate signifi cance for the macro-sign test and micro-sign test. A normal font indicates signifi cance for the macro t-test. For the macro-averages (*i.e.*, excluding micro F1) when both tests are signifi cant it is indicated with a bold, italicized font.

9.5 Reconciling Theory and Practice

While the preceding online algorithms are well-motivated and have attractive theoretical guarantees, in practice their performance was somewhat disappointing. From the practical standpoint, perhaps a guarantee relative to the best base classifier or the best base classifier weighted by indicator is too weak. One way of addressing this might be to broaden the class of regret alternatives to those that have internal regret guarantees.

While this is a step in the correct direction, we can gain insight from examining how metaclassification algorithms have actually gained in practice when applied to base classifiers — in contrast to combining human experts. The majority of schemes can be viewed as a weighted combination of classifier predictions or as selecting the best classifier in a context.

We will focus on the weighted combination scheme and then turn to selecting the best classifier. Continuing with online notation, consider when we have a two class problem where the class $y_t \in \{-1, 1\}$. Suppose base classifier e has a confidence score $\hat{\pi}_{t,e} \in [0, 1]$ and predicts the class $\hat{y}_{t,e} = \text{sign}(\hat{\pi}_{t,e} - 0.5)$. Then, as reviewed in Section 3.2, it is quite common in practice to have a machine learning classification model rank the examples well when sorted by $\hat{\pi}_{t,e}$ but whose classification decisions $\hat{y}_{t,e}$ are not optimal. Essentially, the model has not learned the correct threshold or bias term — instead of predicting $\text{sign}(\hat{\pi}_{t,e} - 0.5)$ it should output $\text{sign}(\hat{\pi}_{t,e} - b)$ where b is a constant in $[0, 1]$. A natural way to consider such a classifier in the combination framework is to combine it with a classifier that always outputs $\hat{\pi}_{t,e} = 1$ and learn a set of combination weights over the $\hat{\pi}_{t,e}$ (recasting the sign issue if necessary). Essentially, we want to learn a set of weights whose loss is close to the loss of the best *averaged prediction*. In contrast, most regret guarantees would give us bounds with respect to the *expected loss* of selecting an expert. In this case, the default classifier performs no better than the prior and the poor threshold of the best classifier is exactly what we would like to fix. Thus, performing as well as the best expected loss is not a great gain. While this is an exaggerated case, it is easy to construct real situations where these methods will not give enough weight to the default classifier and generalize worse than models without explicit guarantees.

If we now consider weighted averages of multiple classifiers, the optimal weight vector can be seen as trading off the accuracy of the classifiers, their co-dependencies, and a cumulative noise or bias term (when a constant prediction is included). By considering methods that apply well to this class, we may achieve looser or no explicit bounds but because the model has more flexibility we can achieve better generalization with sufficient training data.

The majority of bounds are with respect to expected loss; for example, see the definition of l_H^t in the BMX algorithm above. As pointed out in [FSSW97] achieving a bound with respect to averaged prediction is typically harder, and since most loss functions are convex, a tight bound on averaged prediction loss implies a tight bound on expected loss. Since seeing why we do not obtain a good bound in the opposite direction is a key point, it is worth going through in detail.

Let's deal with an online algorithm and restrict our attention to convex linear combinations of the n experts, *i.e.* $C = \{\mathbf{u} \mid \mathbf{u} \in \mathbb{R}^n, \sum u_i = 1, u_i \geq 0\}$. Let, $\hat{\mathbf{s}}_\mathbf{t} \in \mathbb{R}^n$ be the output of the experts, which may be log-odds, probabilities, or binary classes.

A good algorithm with respect to expected loss would output $\mathbf{w}_t$ such that

$$\sum_{t=1}^{T} \mathbf{w_t} \cdot L(\hat{\mathbf{s}}_\mathbf{t}, y_t) \leq \inf_{\mathbf{u}\in C} \sum_{t=1}^{T} \mathbf{u} \cdot L(\hat{\mathbf{s}}_\mathbf{t}, y_t) + Z_I \tag{9.4}$$

where Z_I is an appropriately defined "small" term (typically relative to T, number of experts, and the infimum).

A good algorithm with respect to the loss of an averaged prediction would output $\boldsymbol{\omega_t}$ such that:

$$\sum_{t=1}^{T} L(\boldsymbol{\omega_t} \cdot \hat{\mathbf{s}}_\mathbf{t}, y_t) \leq \inf_{\mathbf{u}\in C} \sum_{t=1}^{T} L(\mathbf{u} \cdot \hat{\mathbf{s}}_\mathbf{t}, y_t) + Z_{II}. \tag{9.5}$$

Assuming, we have a good algorithm with respect to the second of these, averaged prediction loss, we can easily derive a *good* bound for expected loss. In the following, let $\mathbf{w} = \operatorname{argmin}_{\mathbf{u}\in C} \sum_{t=1}^{T} \mathbf{u} \cdot L(\hat{\mathbf{s}}_\mathbf{t}, y_t)$.[5]

$$\begin{aligned}
\sum_{t=1}^{T} L(\boldsymbol{\omega_t} \cdot \hat{\mathbf{s}}_\mathbf{t}, y_t) \quad &\leq \quad \inf_{\mathbf{u}\in C} \sum_{t=1}^{T} L(\mathbf{u} \cdot \hat{\mathbf{s}}_\mathbf{t}, y_t) + Z_{II} && (9.6)\\
&\text{By definition of inf}\\
&\leq \quad \sum_{t=1}^{T} L(\mathbf{w} \cdot \hat{\mathbf{s}}_\mathbf{t}, y_t) + Z_{II} && (9.7)\\
&\text{By Jensen's inequality}\\
&\leq \quad \sum_{t=1}^{T} \mathbf{w} \cdot L(\hat{\mathbf{s}}_\mathbf{t}, y_t) + Z_{II} && (9.8)\\
&\text{By definition of } \mathbf{w}\\
&\leq \quad \inf_{\mathbf{u}\in C} \sum_{t=1}^{T} \mathbf{u} \cdot L(\hat{\mathbf{s}}_\mathbf{t}, y_t) + Z_{II} && (9.9)
\end{aligned}$$

[5]In the case where the minimum is not well-defined, the infimum will be defined and we are still guaranteed the result of the derivation since the inequality holds for all w. We omit the details of this.

Finally, because Z_{II} is guaranteed to be small relative to the smaller term of averaged prediction loss, it will also be sufficiently small for averaged loss.

Now, what about reversing the result? Let $\boldsymbol{\omega} = \mathrm{argmin}_{\mathbf{u}\in C} \sum_{t=1}^{T} L(\mathbf{u} \cdot \hat{\mathbf{s}}_{\mathbf{t}}, y_t)$

$$\begin{aligned}
\sum_{t=1}^{T} \mathbf{w}_{\mathbf{t}} \cdot L(\hat{\mathbf{s}}_{\mathbf{t}}, y_t) &\leq \inf_{\mathbf{u}\in C} \sum_{t=1}^{T} \mathbf{u} \cdot L(\hat{\mathbf{s}}_{\mathbf{t}}, y_t) + Z_I && (9.10)\\
&\leq \sum_{t=1}^{T} \boldsymbol{\omega} \cdot L(\hat{\mathbf{s}}_{\mathbf{t}}, y_t) + Z_I && (9.11)\\
&\text{Now we're stuck}
\end{aligned}$$

From Jensen's inequality, we obviously have $\sum_{t=1}^{T} L(\boldsymbol{\omega} \cdot \hat{\mathbf{s}}_{\mathbf{t}}, y_t) \leq \sum_{t=1}^{T} \boldsymbol{\omega} \cdot L(\hat{\mathbf{s}}_{\mathbf{t}}, y_t) + Z_I$, but what we want is to say that the combiner's loss is *close* to $\sum_{t=1}^{T} L(\boldsymbol{\omega} \cdot \hat{\mathbf{s}}_{\mathbf{t}}, y_t)$ which would require Jensen's inequality to be flipped to continue the derivation.

From a graphical point of view, choosing a solution with a loss close to the minimum of expected loss does not mean that using that same vector will be close to the minimum of the averaged prediction loss. Some vectors may have a large divergence in the two measures. We still have a loss bound on what our loss can be if we use the average loss solution, $\mathbf{w}_t$, to average the predictions but we are not guaranteed to be in the neighborhood of the minimum for average prediction loss. That is, the bound may be extremely loose with respect to the minimum.

Because of this and the closely related issues when combining automatically produced experts, metaclassifiers without explicit guarantees (*e.g.*, SVMs) may generalize better since their models allow for the expressivity to optimize for averaged prediction loss. While we do not pursue such a vein here, the interested reader may be able to construct algorithms with guarantees and that work well for combination by generalizing some of the basic combination results with averaged prediction guarantees (*e.g.*, exponentiated gradient methods [FSSW97]) along the indicator variable lines presented in [BM05].

9.6 Chapter Summary

In this chapter, we provided an overview of some key approaches to combining classifiers in an online classification framework and empirically analyzed their suitability for application to our batch prediction framework. While the methods did not suffer any large losses as guaranteed, our empirical analysis highlighted the lack of significant wins relative to using a linear SVM as a batch metaclassifier as explored earlier. In our analysis, we discussed how guarantees with respect to average loss of the experts is far weaker than the type of guarantee with respect to loss of averaged prediction that is needed. Finally, we suggested literature of interest and future directions that the interested reader can pursue to continue this line of research in a way that is likely to show higher empirical performance as well as theoretical guarantees.

Chapter 10

Action-Item Detection in E-mail

In this chapter, we demonstrate that classifier combination methods are applicable to text combination approaches outside of topic classification. In doing so, the aim is to demonstrate the flexibility and applicability of these methods to a range of problems. As such, we have chosen a problem, *action-item detection* in e-mail documents, that not only presents different challenges as a learning problem than those present in topic classification but is also focused on a different performance goal — improving the *ranking* of e-mails.

E-mail users have an increasingly difficult time managing their inboxes in the face of challenges that result from rising e-mail usage. This includes prioritizing e-mails over a range of sources from business partners to family members, filtering and reducing junk e-mail, and quickly managing requests that demand the receiver's attention or action. Automated *action-item detection* targets the third of these problems by attempting to detect which e-mails *require* an action or response with information, and within those e-mails, attempting to highlight the sentence or passage containing the action request.

Such a detection system can be used as one part of an e-mail agent which would assist a user in processing important e-mails more quickly than would have been possible without the agent. We view action-item detection as one necessary component of a successful e-mail agent which would perform spam detection, action-item detection, topic classification and priority ranking, among other functions. The utility of such a detector can manifest as a method of prioritizing e-mails according to task-oriented criteria other than the standard ones of topic and sender or as a means of ensuring that the email user hasn't dropped the proverbial ball by forgetting to address an action request.

In the context of this dissertation, action-item detection forms a very different type of text classification problem for empirical study than topic classification. In particular, while the cues indicating topic are spread throughout a document, the action-item(s) in an e-mail are typically localized, often in the context of a single sentence. Thus, the intuition is that the class of the document will interact differently with the document feature representation than in topic classification. Therefore, the natural question is whether this changes either the effectiveness of the base classifiers or the combination approaches.

Secondly, while it can be challenging to identify exactly which portions of a document make it about a specific topic during topic classification, identifying the sentences that determine the action-item status of an e-mail is relatively straightforward. Given such information, we would like to know whether it can be used effectively in the basic classification task, and if so, how we can incorporate this information into a reliability indicator based approach during classifier combination.

We first continue to layout and describe the basic problem of action-item detection. Then, we review related work for similar text classification problems such as e-mail priority ranking and speech act identification. Next we more formally define the action-item detection problem, discuss the aspects that distinguish it from more common problems like topic classification, and highlight the challenges in constructing systems that can perform well at the sentence and document level. From there, we move to a discussion of feature representation and selection techniques appropriate for this problem and how standard text classification approaches can be adapted to move smoothly from the sentence-level detection problem to the document-level classification problem. We then conduct an empirical analysis that helps us determine the effectiveness of our feature extraction procedures as well as establish baselines for a number of classification algorithms on this task. Next, we turn to the question of how we can combine a set of action-item detection systems and how both sentence-level and document-level classifiers can be combined. Finally, we summarize the implications for applying classifier combination techniques to other domains.

10.1 Why Action-Item Detection?

Action-item detection differs from standard text classification in two important ways. First, the user is interested both in detecting whether an email contains action items and in locating exactly where these action item requests are contained within the email body. In contrast, standard text categorization merely assigns a topic label to each text, whether that label corresponds to an e-mail folder or a controlled indexing vocabulary [Lar99, YZCJ02, LYJC04]. Second, action-item detection attempts to recover the email sender's intent — whether she means to elicit response or action on the part of the receiver; note that for this task, classifiers using only a bag-of-words representation do not perform optimally, as evidenced in our results below. Instead we find that we need more information-laden features such as higher-order n-grams. Text categorization by topic, on the other hand, works

From: Henry Hutchins <hhutchins@innovative.company.com>
To: Sara Smith; Joe Johnson; William Woolings
Subject: meeting with prospective customers
Sent: Fri 12/10/2005 8:08 AM

Hi All,
I'd like to remind all of you that the group from GRTY will be visiting us next Friday at 4:30 p.m. The current schedule looks like this:
+ 9:30 a.m. Informal Breakfast and Discussion in Cafeteria
+ 10:30 a.m. Company Overview
+ 11:00 a.m. Individual Meetings (Continue Over Lunch)
+ 2:00 p.m. Tour of Facilities
+ 3:00 p.m. Sales Pitch
In order to have this go off smoothly, I would like to practice the presentation well in advance. *As a result, I will need each of your parts by Wednesday.*
Keep up the good work!
–Henry

Figure 10.1: An E-mail with emphasized Action-Item, an *explicit* request that requires the recipient's attention or action.

very well using just individual words as features[Lew92a, ADW94a, DPHS98, Seb02]. In fact, genre-classification, which one would think may require more than a bag-of-words approach, also works quite well using just unigram features[LCJ03]. Topic detection and tracking (TDT), also works well with unigram feature sets [YCB$^+$99, ACD$^+$98]. We believe that action-item detection is one of the first clear instances of a text classification related task where we must move beyond bag-of-words to achieve high performance, albeit not too far, as bag-of-n-grams seem to suffice, given state-of-the-art classifiers.

10.2 Related Work

Several other researchers have considered very similar text classification tasks. Cohen *et al.* [CCM04] describe an ontology of "speech acts", such as "Propose a Meeting", and attempt to predict when an e-mail contains one of these speech acts. While their ontology mostly focused on types of speech acts that are specific kinds of action-item requests, we consider action-items in general to be an important specific type of speech act that falls within a much broader ontology of speech acts. Furthermore, while they provide results for several classification methods, their methods only make use of human judgments at the document-level. In contrast, we consider whether accuracy can be increased by using finer-grained human judgments that mark the specific sentences and phrases of interest.

Corston-Oliver *et al.* [CORGC04] consider detecting items in e-mail to “Put on a To-Do List”. This classification task is very similar to ours except they do not consider “simple factual questions” to belong to this category. We include questions, but note that not all questions are action-items — some are rhetorical or simply social convention, *e.g.*, “How are you?”. From a learning perspective, while they make use of judgments at the sentence-level, they do not explicitly compare what, if any, benefits finer-grained judgments offer. Additionally, they do not study alternative choices or approaches to the classification task. Instead, they simply apply a standard SVM at the sentence-level and focus primarily on a linguistic analysis of how the sentence can be logically reformulated before adding it to the task list. In this study, we examine several alternative classification methods, compare document-level and sentence-level approaches and analyze the machine learning issues implicit in these problems. We are also the first to examine in detail the gains from classifier combination in this problem.

For those that wish to purse further reading on this topic, a variety of learning tasks related to e-mail has been rapidly growing in the recent literature. For example, in a forum dedicated to e-mail learning tasks, Culotta *et al.* [CBM04] presented methods for learning social networks from e-mail. We do not use peer relationship information in building our classifiers; however, such methods could complement those here since peer relationships often influence word choice when requesting an action.

10.3 Problem Definition & Approach

In contrast to previous work, we explicitly focus on the benefits that finer-grained, more costly, sentence-level human judgments offer over coarse-grained document-level judgments. Additionally, we consider multiple standard text classification approaches and analyze both the quantitative and qualitative differences that arise from taking a document-level vs. a sentence-level approach to classification. We also focus on the representation necessary to achieve the most competitive performance. Finally, after demonstrating the difference in document-level and sentence-level approaches to the document classification task, we examine what can be gained from classifier combination in this problem.

10.3.1 Problem Defi nition

In order to provide the most benefit to the user, a system would not only detect the document, but it would also indicate the specific sentences in the e-mail which contain the action-items. Therefore, there are three basic action-item detection problems:

1. *Document detection*: Classify a document as to whether or not it contains an action-item.
2. *Document ranking*: Rank the documents such that all documents containing action-items occur as high as possible in the ranking.
3. *Sentence detection*: Classify each sentence in a document as to whether or not it is an action-item.

As in most Information Retrieval tasks, the weight the evaluation metric should give to precision and recall depends on the nature of the application. In situations where a user will eventually read all received messages, ranking (*e.g.*, via precision at recall of 1) may be most important since this will help encourage shorter delays in communications between users. In contrast, high-precision detection at low recall will be of increasing importance when the user is under severe time-pressure and therefore will likely not read all mail. This can be the case for crisis managers during disaster management. Finally, sentence detection plays a role in both time-pressure situations and simply to alleviate the user's required time to gist the message. In the first part of this chapter, we focus on standard performance measures such as F1 and accuracy. Then, as we return to classifier combination, we will discuss ranking measures such as ROC curves and area under the curve.

10.3.2 Approach

As mentioned above, the labeled data can come in one of two forms: a *document-labeling* provides a yes/no label for each document as to whether it contains an action-item; a *phrase-labeling* provides only a yes label for the specific items of interest. We term the human judgments a *phrase-labeling* since the user's view of the action-item may not correspond with actual sentence boundaries or predicted sentence boundaries. Obviously, it is straightforward to generate a document-labeling consistent with a phrase-labeling by labeling a document "yes" if and only if it contains at least one phrase labeled "yes".

To train classifiers for this task, we can take several viewpoints related to both the basic problems we have enumerated and the form of the labeled data. The *document-level* view treats each e-mail as a learning instance with an associated class-label. Then, the document can be converted to a feature-value vector and learning progresses as usual. Applying a document-level classifier to document detection and ranking is straightforward. In order to apply it to sentence detection, one must make additional steps. For example, if the classifier predicts a document contains an action-item, then areas of the document that contain a high concentration of words which the model weights heavily in favor of action-items can be indicated. The obvious benefit of the document-level approach is that training set collection costs are lower since the user only has to specify whether or not an e-mail contains an action-item and not the specific sentences.

In the *sentence-level* view, each e-mail is automatically segmented into sentences, and each sentence is treated as a learning instance with an associated class-label. Since the phrase-labeling provided by the user may not coincide with the automatic segmentation, we must determine what label to assign a partially overlapping sentence when converting it to a learning instance. Once trained, applying the resulting classifiers to sentence detection is now straightforward, but in order to apply the classifiers to document detection and document ranking, the individual predictions over each sentence must be aggregated in order to make a document-level prediction. This approach has the potential to benefit from more-specific labels that enable the learner to focus attention on the key sentences, instead of having to learn based on data for which the majority of the words in the e-mail provide no or little information about class membership.

Features

Consider some of the phrases that might constitute part of an action item: “would like to know”, “let me know”, “as soon as possible”, “have you”. Each of these phrases consists of common words that occur in many e-mails. However, when they occur as a phrase in the same sentence, they are far more indicative of an action-item. Additionally, order can be important: consider “have you” versus “you have”. Because of this, we posit that n-grams play a larger role in this problem than is typical of problems like topic classification. Therefore, we consider all n-grams up to size 4. Thus, we compare using a “bag of phrases and words” to simply using a “bag of words”.

When using n-grams, if we find an n-gram of size 4 in a segment of text, we can represent the text as just one occurrence of the n-gram or as one occurrence of the n-gram and an occurrence of each smaller n-gram contained by it. We choose the second of these alternatives since this will allow the algorithm itself to smoothly back-off in terms of recall. Methods such as naïve Bayes may be hurt by such a representation because of double-counting.

Since sentence-ending punctuation can provide information, we retain the terminating punctuation token when it is identifiable. Additionally, we add a *beginning-of-sentence* and *end-of-sentence* token in order to capture patterns that are often indicators at the beginning or end of a sentence. Assuming proper punctuation, these extra tokens are unnecessary, but often e-mail lacks proper punctuation. In addition, for the sentence-level classifiers that use n-grams, we additionally code for each sentence a binary encoding of the *position* of the sentence relative to the document. This encoding has eight associated features that represent which octile (the first eighth, second eighth, *etc.*) contains the sentence.

Implementation Details

In order to compare the document-level to the sentence-level approach, we compare predictions at the document-level. We do not address how to use a document-level classifier to make predictions at the sentence-level.

In order to automatically segment the text of the e-mail, we use the RASP statistical parser [Car02]. Since the automatically segmented sentences may not correspond directly with the phrase-level boundaries, we treat any sentence that contains at least 30% of a marked action-item segment as an action-item. When evaluating sentence-detection for the sentence-level system, we use these class labels as ground truth. Since we are not evaluating multiple segmentation approaches, this does not bias any of the methods. If multiple segmentation systems were under evaluation, one would need to use a metric that matched predicted positive sentences to phrases labeled positive. The metric would need to punish overly long true predictions as well as too-short predictions. Our criteria for converting to labeled instances implicitly includes both criteria. Since the segmentation is fixed, an overly long prediction would be predicting “yes” for many “no” instances since presumably the extra length corresponds to additional segmented sentences all of which do not contain 30% of an action-item. Likewise, a too short prediction must correspond to a small sentence included in the action-item but not constituting all of the action-item. Therefore, in order to consider the prediction to be too short, there will be an additional preceding/following sentence that is an action-item where we incorrectly predicted “no”.

Once a sentence-level classifier has made a prediction for each sentence, we must combine these predictions to make both a document-level prediction and a document-level score.[1] We use the simple policy of predicting positive when any of the sentences is predicted positive. In order to produce a document score for ranking, the confidence that the document contains an action-item is:

$$\psi(d) = \begin{cases} \frac{1}{n(d)} \sum_{s \in d | \pi(s)=1} \psi(s) & \text{if for any } s \in d, \pi(s) = 1 \\ \frac{1}{n(d)} \max_{s \in d} \psi(s) & \text{o.w.} \end{cases}$$

where s is a sentence in document d, π is the classifier’s 1/0 prediction, ψ is the score the classifier assigns as its confidence that $\pi(s) = 1$, and $n(d)$ is the greater of 1 and the number

[1]This combination problem differs in nature from those discovered previously in this dissertation. In this problem the set of experts (sentences) changes from document to document. We go further into detail on this point later in the chapter.

of (unigram) tokens in the document. In other words, when any sentence is predicted positive, the document score is the length normalized sum of the sentence scores above threshold. When no sentence is predicted positive, the document score is the maximum sentence score normalized by length. As in other text problems, we are more likely to emit false positives for documents with more words or sentences. Thus we include a length normalization factor.

10.4 Experimental Analysis for Action-Item Detection

10.4.1 The Data

Our corpus consists of e-mails obtained from volunteers at Carnegie Mellon University and cover subjects such as: organizing a research workshop, arranging for job-candidate interviews, publishing proceedings, and talk announcements. The messages were anonymized by replacing the names of each individual and institution with a pseudonym. After attempting to identify and eliminate duplicate e-mails, the corpus contains 744 e-mail messages.

After identity anonymization, the corpora has three basic versions. *Quoted material* refers to the text of a previous e-mail that an author often leaves in an e-mail message when responding to the e-mail. Quoted material can act as noise when learning since it may include action-items from previous messages that are no longer relevant. To isolate the effects of quoted material, we have three versions of the corpora. The *raw* form contains the basic messages. The *auto-stripped* version contains the messages after quoted material has been automatically removed. The *hand-stripped* version contains the messages after quoted material has been removed by a human. Additionally, the hand-stripped version has had any xml content and e-mail signatures removed — leaving only the essential content of the message. The studies reported here are performed with the hand-stripped version. This allows us to balance the cognitive load in terms of number of tokens that must be read in the user-studies we report — including quoted material would complicate the user studies since some users might skip the material while others read it. Additionally, ensuring all quoted material is removed prevents tainting the cross-validation since otherwise a test item could occur as quoted material in a training document.

Data Labeling

Two human annotators labeled each message as to whether or not it contained an action-item. In addition, they identified each segment of the e-mail which contained an action-item. A segment is a contiguous section of text selected by the human annotators and may span several sentences or a complete phrase contained in a sentence. They were instructed that an action item is “an explicit request for information that requires the recipient’s attention or a required action” and told to “highlight the phrases or sentences that make up the request”.

		Annotator 1	
		No	Yes
Annotator 2	No	391	26
	Yes	29	298

Table 10.1: Agreement of Human Annotators at Document Level

Annotator One labeled 324 messages as containing action items. Annotator Two labeled 327 messages as containing action items. The agreement of the human annotators is shown in Tables 10.1 and 10.2. The annotators are said to *agree at the document-level* when both marked the same document as containing no action-items or both marked at least one action-item regardless of whether the text segments were the same. At the document-level, the annotators agreed 93% of the time. The kappa statistic [Car96, CCM04] is often used to evaluate inter-annotator agreement:

$$\kappa = \frac{A - R}{1 - R}$$

A is the empirical estimate of the probability of **a**greement. R is the empirical estimate of the probability of **r**andom agreement given the empirical class priors. A value close to -1 implies the annotators agree far less often than would be expected randomly, while a value close to 1 means they agree more often than randomly expected.

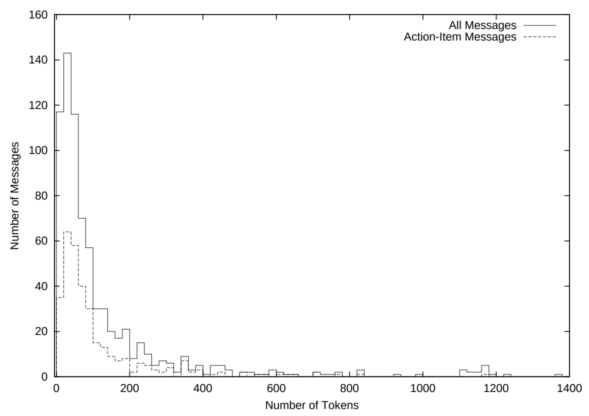
At the document-level, the kappa statistic for inter-annotator agreement is 0.85. This value is both strong enough to expect the problem to be learnable and is comparable with results for similar tasks [CCM04, CORGC04].

In order to determine the sentence-level agreement, we use each judgment to create a sentence-corpus with labels as described in Section 10.3.2, then consider the agreement over these sentences. This allows us to compare agreement over “no judgments”. We perform this comparison over the hand-stripped corpus since that eliminates spurious “no” judgments that would come from including quoted material, etc. Both annotators were free to label the subject as an action-item, but since neither did, we omit the subject line of the message as well. This only reduces the number of “no” agreements. This leaves 6301 automatically segmented sentences. At the sentence-level, the annotators agreed 98% of the time, and the kappa statistic for inter-annotator agreement is 0.82.

		Annotator 1	
		No	Yes
Annotator 2	No	5810	65
	Yes	74	352

Table 10.2: Agreement of Human Annotators at Sentence Level

In order to produce one single set of judgments, the human annotators went through each annotation where there was disagreement and came to a consensus opinion. The annotators did not collect statistics during this process but anecdotally reported that the majority of disagreements were either cases of clear annotator oversight or different interpretations of conditional statements. For example, *"If you would like to keep your job, come to tomorrow's meeting"* implies a required action where *"If you would like to join the football betting pool, come to tomorrow's meeting"* does not. The first would be an action-item in most contexts while the second would not. Of course, many conditional statements are not so clearly interpretable. After reconciling the judgments there are 416 e-mails with no action-items and 328 e-mails containing action-items. Of the 328 e-mails containing action-items, 259 messages have one action-item segment; 55 messages have two action-item segments; 11 messages have three action-item segments. Two messages have four action-item segments, and one message has six action-item segments. Computing the sentence-level agreement using the reconciled "gold standard" judgments with each of the annotators' individual judgments gives a kappa of 0.89 for Annotator One and a kappa of 0.92 for Annotator Two.



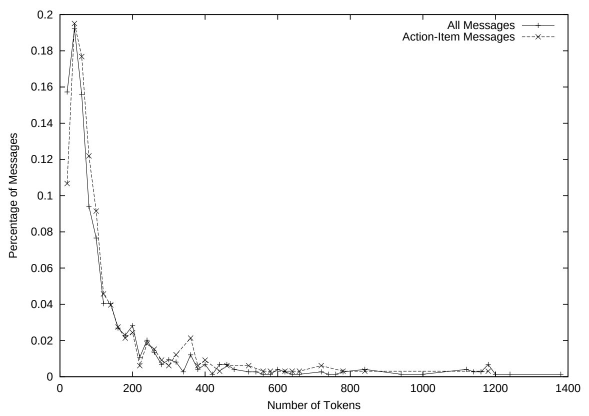


Figure 10.2: The Histogram (left) and Distribution (right) of Message Length. A bin size of 20 words was used. Only tokens in the body after hand-stripping were counted. After stripping, the majority of words left are usually actual message content.

In terms of message characteristics, there were on average 132 content tokens in the body after stripping. For action-item messages, there were 115. However, by examining Figure 10.2 we see the length distributions are nearly identical. As would be expected for e-mail, it is a long-tailed distribution with about half the messages having more than 60 tokens in the body (this paragraph has 65 tokens).

10.4.2 Classifi ers

In order to establish baselines, we have selected a variety of standard text classification algorithms. Later in this chapter, we return to exactly the same set of base classifiers used throughout the dissertation, but because action-item detection is relatively unstudied, in this section we choose a slightly different set of algorithms to study the features of action-item detection as a learning problem. In selecting algorithms, we have chosen algorithms that are not only known to work well but which differ along such lines as discriminative vs. generative and lazy vs. eager. We have done this in order to provide both a competitive and thorough sampling of learning methods for the task at hand. This is important since it is easy to improve a strawman classifier by introducing a new representation. By thoroughly sampling alternative classifier choices we demonstrate that representation improvements over bag-of-words are not due to using the information in the bag-of-words poorly.

kNN

As done throughout the dissertation, we employ a standard variant of the k-nearest neighbor algorithm used in text classification, kNN with s-cut score thresholding [Yan99]. We use a tfidf-weighting of the terms with a distance-weighted vote of the neighbors to compute the score before thresholding it. In order to choose the value of s for thresholding, we perform leave-one-out cross-validation over the training set. The value of k is set to be $2(\lceil \log_2 N \rceil + 1)$ where N is the number of training points. This rule for choosing k is theoretically motivated by results which show such a rule converges to the optimal classifier as the number of training points increases [DGL96]. In practice, we have also found it to be a computational convenience that frequently leads to comparable results with numerically optimizing k via a cross-validation procedure.

Unigram (multinomial Naïve Bayes)

We use a standard multinomial naïve Bayes classifier [MN98]. As done throughout the dissertation, in using this classifier, we smoothed word and class probabilities using a Bayesian estimate (with the word prior) and a Laplace m-estimate, respectively. To use terminology consistent with the rest of the dissertation, this classifier is referred to as a *Unigram* classifier. Note that this is distinct from a unigram or bag-of-words representation as also discussed here.

SVM

We have used a linear SVM with a tfidf feature representation and L2-norm as implemented in the SVM^{light} package v6.01 [Joa99]. All default settings were used.

Voted Perceptron

Like the SVM, the Voted Perceptron is a kernel-based learning method. We use the same feature representation and kernel as we have for the SVM, a linear kernel with tfidf-weighting and an L2-norm. The voted perceptron is an online-learning method that keeps a history of past perceptrons used, as well as a weight signifying how often that perceptron was correct. With each new training example, a correct classification increases the weight on the current perceptron and an incorrect classification updates the perceptron. The output of the classifier uses the weights on the perceptra to make a final “voted” classification. When used in an offline-manner, multiple passes can be made through the training data. Furthermore, it is well-known that the Voted Perceptron increases the margin of the solution after each pass through the training data [FS99]. Since Cohen *et al.* [CCM04] obtain worse results using an SVM than a Voted Perceptron with one training iteration, they conclude that the best solution for detecting speech acts may not lie in an area with a large margin. Because their tasks are highly similar to ours, we employ both classifiers to ensure we are not overlooking a competitive alternative classifier to the SVM for the basic bag-of-words representation.

10.4.3 Performance Measures

To compare the performance of the classification methods for this task, we look at two standard performance measures, F1 and accuracy.

10.4.4 Experimental Methodology

We perform standard 10-fold cross-validation on the set of documents. For the sentence-level approach, all sentences in a document are either entirely in the training set or entirely in the test set for each fold. For significance tests, we use a two-tailed t-test [YL99] to compare the values obtained during each cross-validation fold with a p-value of 0.05.

Feature selection was performed using the chi-squared statistic. Different levels of feature selection were considered for each classifier. Each of the following number of features was tried: $10, 25, 50, 100, 250, 750, 1000, 2000, 4000$. There are approximately 4700 unigram tokens without feature selection. In order to choose the number of features to use for each classifier, we perform nested cross-validation and choose the settings that yield the optimal document-level F1 for that classifier. For this study, only the body of each e-mail message was used. Feature selection is always applied to all candidate features. That is, for the n-gram representation, the n-grams and position features are also subject to removal by the feature selection method.

The document-level classifiers below that use a bag-of-words representation use all unigram tokens as the feature pool including sentence-ending markers and punctuation such as ".", "!", and "?". These classifiers are denoted as *Document BoW*. A second set of document-level classifiers that also include n-grams in the feature pool are denoted as *Document Ngram*. The sentence-level classifiers that use a bag-of-words representation also use all unigram tokens in the feature pool including the sentence-ending punctuation. These classifiers are denoted as *Sentence BoW*. Finally, the sentence-level classifiers denoted as *Sentence Ngram* additionally include n-grams and the encoding of the position of the sentence within the document in the feature pool.

10.4.5 Baseline Results for Action-Item Detection

The results for document-level classification are given in Table 10.3. The primary hypothesis we are concerned with is that n-grams are critical for this task; if this is true, we expect to see a significant gap in performance between the document-level classifiers that use n-grams (*Document Ngram*) and those using the bag-of-words representation (*Document BoW*). Examining Table 10.3, we observe that this is indeed the case for every classifier except the Unigram classifier. This difference in performance produced by the n-gram representation is statistically significant for each classifier except for the Unigram classifier and the accuracy metric for kNN (see Table 10.4). The Unigram classifier's poor performance with the n-gram representation is not surprising since the bag-of-n-grams causes excessive double-counting as mentioned in Section 10.3.2; however, the Unigram classifier is not hurt at the sentence-level because the sparse examples provide few chances for agglomerative effects of double counting. In either case, when a language-modeling approach is desired, modeling the n-grams directly may be preferable to using a multinomial naïve

Bayes model. More importantly for the n-gram hypothesis, the n-grams lead to the best document-level classifier performance as well.

As would be expected, the difference between the *sentence-level* n-gram representation and bag-of-words representation is small. This is because the window of text is so small that the bag-of-words representation, when done at the sentence-level, implicitly picks up on the power of the n-grams. Further improvement would signify that the order of the words matter even when only considering a small sentence-size window. Therefore, the finer-grained sentence-level judgments allows a bag-of-words representation to succeed but only when performed in a small window — behaving as an n-gram representation for all practical purposes.

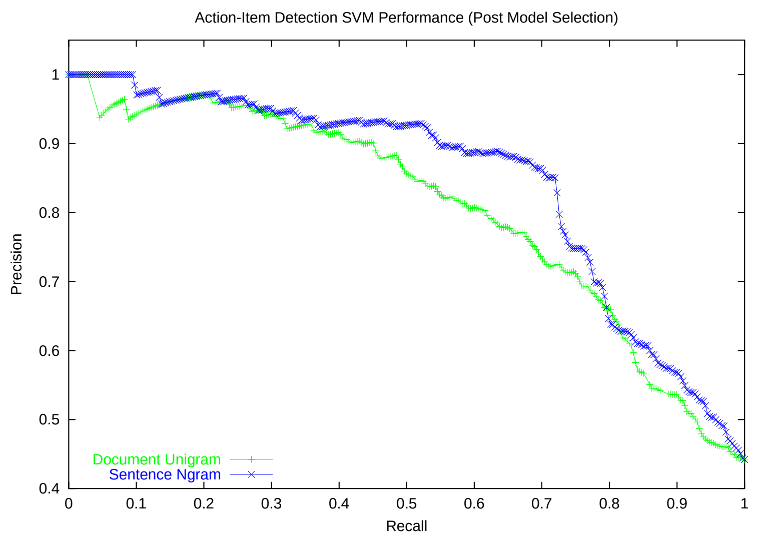


Figure 10.3: Both n-grams and a small prediction window lead to consistent improvements over the standard approach.

Further highlighting the improvement from finer-grained judgments and n-grams, Figure 10.3 graphically depicts the edge the SVM sentence-level classifier has over the standard bag-of-words approach with a precision-recall curve. In the high precision area of the graph, the consistent edge of the sentence-level classifier is rather impressive — continuing at precision 1 out to 0.1 recall. This would mean that a tenth of the user's action-items would be placed at the top of their action-item sorted inbox. Additionally, the large separation at the top right of the curves corresponds to the area where the optimal F1 occurs for each classifier, agreeing with the large improvement from 0.6904 to 0.7682 in F1 score. Considering the relatively unexplored nature of classification at the sentence-level, this gives great hope for further increases in performance.

Although Cohen *et al.* [CCM04] observed that the Voted Perceptron with a single training iteration outperformed SVM in a set of similar tasks, we see no such behavior here. This further strengthens the evidence that an alternate classifier with the bag-of-words representation could not reach the same level of performance. The Voted Perceptron classifier does improve when the number of training iterations are increased, but it is still lower than the SVM classifier.

Sentence detection results are presented in Table 10.6. With regard to the sentence detection problem, we note that the $F1$ measure gives a better feel for the remaining room for improvement in this difficult problem. That is, unlike document detection where action-item documents are fairly common, action-item sentences are very rare. Thus, as in other text problems, the accuracy numbers are deceptively high solely because of the default accuracy attainable by always predicting "no". Although, the results here are significantly above-random, it is unclear what level of performance is necessary for sentence detection to be useful in and of itself and not simply as a means to document ranking and classification.

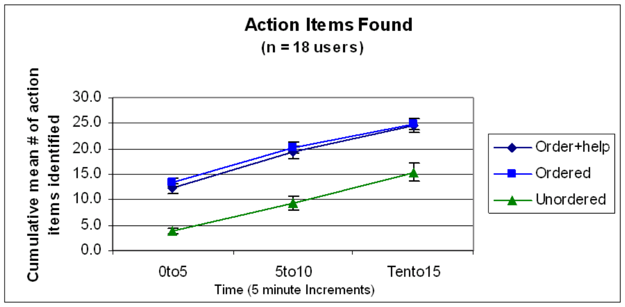


Figure 10.4: Users fi nd action-items more quickly when assisted by a classifi cation system.

Finally, when considering a new type of classification task, one of the most basic questions is whether an accurate classifier built for the task can have an impact on the end-user. In order to demonstrate the impact this task can have on e-mail users, we conducted a user study using an earlier less-accurate version of the sentence classifier — where instead of using just a single sentence, a three-sentence *windowed-approach* was used. There were three distinct sets of e-mail in which users had to find action-items. These sets were either presented in a random order (*Unordered*), ordered by the classifier (*Ordered*), or ordered by the classifier and with the center sentence in the highest confidence window highlighted (*Order+help*). In order to perform fair comparisons between conditions, the overall number of tokens in each message set should be approximately equal; that is, the cognitive reading load should be approximately the same before the classifier's reordering. Additionally, users typically show "practice effects" by improving at the overall task and thus performing better at later message sets. This is typically handled by varying the ordering of the sets across users so that the means are comparable. While omitting further detail, we note the sets were balanced for the total number of tokens and a latin square design was used to balance practice effects.

Figure 10.4 shows that at intervals of 5, 10, and 15 minutes, users consistently found significantly more action-items when assisted by the classifier, but were most critically aided in the first five minutes. Although, the classifier consistently aids the users, we did not gain an additional end-user impact by highlighting. As mentioned above, this might be a result of the large room for improvement that still exists for sentence detection, but anecdotal evidence suggests this might also be a result of how the information is presented to the user rather than the accuracy of sentence detection. For example, highlighting the wrong sentence near an actual action-item hurts the user's trust, but if a vague indicator (*e.g.*, an arrow) points to the approximate area the user is not aware of the near-miss. Since the user studies used a three sentence window, we believe this played a role as well as sentence detection accuracy.

10.4.6 Discussion

In contrast to problems where n-grams have yielded little difference, we believe their power here stems from the fact that many of the meaningful n-grams for action-items consist of common words, *e.g.*, "let me know". Therefore, the document-level bag-of-words representation cannot gain much leverage, even when modeling their joint probability correctly, since these words will often co-occur in the document but not necessarily in a phrase. Additionally, action-item detection is distinct from many text classification tasks in that a single sentence can change the class label of the document. As a result, good classifiers cannot rely on aggregating evidence from a large number of weak indicators across the entire document.

Even though we discarded the header information, examining the top-ranked features at the document-level reveals that many of the features are names or parts of e-mail addresses that occurred in the body and are highly associated with e-mails that tend to contain many or no action-items. A few examples are terms such as "org", "bob", and "gov". We note that these features will be sensitive to the particular distribution (senders/receivers) and thus the document-level approach may produce classifiers that transfer less readily to alternate contexts and users at different institutions. This points out that part of the problem of going beyond bag-of-words may be the methodology, and investigating such properties as learning curves and how well a model transfers may highlight differences in models which appear to have similar performance when tested on the distributions they were trained on.

Finally, the effectiveness of sentence-level detection argues that labeling at the sentence-level provides significant value. Further experiments would be required to see how this interacts with the amount of training data available. Sentence detection that is then agglomerated to document-level detection works surprisingly well given the low recall that would be expected with sentence-level items.

The baseline results have established how action-items can be effectively detected in e-mails. Our empirical analysis has demonstrated that in contrast to topic classification n-grams are of key importance to making the most of document-level judgments. When finer-grained judgments are available, then standard bag-of-words approaches using a small (sentence) window size can produce results almost as good as the n-gram based approaches.

10.5 Action-Item Detection vs. Topic Classification

Before we return to classifier combination, we highlight the differences that make action-item detection a different type of task than topic classification and thus suitable for demonstrating the applicability of our combination algorithms to other types of problems.

First, where topic classification has topical cues spread throughout the document, action-item detection often has a single sentence of interest and thus detecting that sentence is of critical importance. While we have seen that this can be exploited using sentence-level classifiers, it is of interest to see how these sentence-level classifiers can be integrated in a reliability-indicator framework.

From a machine learning perspective, the action-item corpus also has an entirely different balance of positives and negatives than typically seen for topic classification. In the corpus, action-item e-mails constitute 44% of the corpus. Thus, while striving may sometimes demand more positive training examples to work well (see Section 7.4), the more balanced nature of this problem may reduce the demand for training data.

Next, unlike topic-classification we also have classifiers built from different "views" — the document-level and the sentence-level. The next natural question is whether classifier combination can make use of the different information that the document-models and the sentence-models provide. We now turn to examine these issues of classifier combination.

10.6 Classifier Combination for Action-Item Detection

When considering classifier combination for action-item detection, there are several different challenges that present themselves. Key questions are how sentence-level classifier judgments can be combined into document-level judgments, how document-level and sentence-level models can be combined, whether gains can be achieved from combining different classification algorithms, and whether any changes to the reliability-indicators used for topic classification are necessary in the strive methodology.

Combining sentence-level judgments into document-level judgments provides a unique challenge distinct from the primary combination problem discussed in this dissertation. When we combine the models from different classification algorithms, we obtain a prediction from each model for every example we are considering. This is analogous to consulting an expert panel consisting of a fixed set of experts for each example. However, when combining the sentence predictions to make a document-level prediction, there is no consistency from document to document. The prediction on the first sentence in the first document is generally not related to the prediction on any sentence in the second document. The analogy here is that we consult a *different* set of experts for each example. This problem is beyond the scope of this dissertation, and we do not directly study the issue related to alternative methods for combining sentence predictions into a document prediction. Instead, we will continue to use the default method of combining sentence-level predictions to obtain a document-level prediction given in Equation 10.3.2. Our focus in this section will be solely on how we can combine the document predictions from the document-level and sentence-level models to yield a more reliable document ranking and what reliability-indicator changes are necessary for this task.

In particular, since a typical user will eventually process all received mail, we assume that producing a quality ranking will more directly measure the impact on the user than accuracy or F1. Therefore, in the remainder of the chapter our focus will be on ROC curves and area under the curve since both reflect the quality of the ranking produced. Additionally, while the previous section provided many different options in terms of representation and base classifiers, a typical question that comes up in classifier combination is how well a combination method deals with a large set of classifiers. Therefore, rather than pre-selecting among the best models for action-item detection, we will simply use all five base classifiers we used in Chapter 7 with both the bag-of-words and bag-of-n-grams representations at both the sentence-level and the document-level. This gives us a total of 20 different base classifiers. With only 744 e-mails in the action-item detection corpus, we are interested in seeing whether Strive can not only improve but also avoid harming performance because of correlated inputs. Furthermore, since it was *STRIVE-S (norm)* that was the primary competitor, we are primarily interested in investigating the behavior of this variant although we also provide results for the decision-tree metaclassifier for completeness. Finally, part of the value in any method is how much it must be tuned to the problem at hand. Therefore, our focus is not only on applying Strive but doing so in as much of an "out-of-the-box" manner as possible. To this end, we next address modifications needed for the reliability indicators before presenting and analyzing results for classifier combination for action-item detection.

10.7 Reliability Indicators for Action-Item Detection

First, for each of the document-level base classifiers, we can clearly still use the classifier-based reliability indicators introduced in Chapter 5. There were 6 unigram-model based variables, 6 naïve Bayes variables, 10 kNN variables, 5 SVM variables, and 2 decision-tree variables. Since we are constructing base classifiers for both the bag-of-words and bag-of-n-grams representations, this gives 58 reliability indicators ($58 = 2 \text{ representations} * [6 + 6 + 10 + 5 + 2]$).

Although the classifier-based reliability indicators are defined for each sentence prediction, in order to use them at the document-level we must somehow combine the reliability indicators over each sentence. The simplest method would be to take the average over each classifier-based indicator across the sentences in the document. We do so and thus obtain another 58 reliability indicators.

While combining the sentence-level predictions into a document-level prediction is outside the scope of our model, our model can benefit from some of the structure a sentence-level classifier offers when combining document predictions. Analogous to considering the variance of feature weights in the naïve Bayes model, we can consider such indicators as the mean and standard deviation of the classifier confidences over each sentence within the document. For each sentence-level base classifier, these then become two more indicators (mean and standard deviation) which we can benefit from when combining document predictions. Since we are using the same set of 5 base classifiers as elsewhere in the Strive experiments and we have base classifiers for the sentence-level bag-of-words and sentence-level bag-of-n-grams, this introduces 20 variables ($20 = 2 \text{ representations} * 2(\text{mean, stdev}) * 5 \text{ base classifiers}$).

Next, we could also extrapolate the feature-selection-based reliability indicators discussed in Chapter 5 to this problem. However, we do not for two reasons. First, while the feature selection variables are easily defined for this problem, they are more problematic to extrapolate to non-text classification problems since they generally rely on the sparse nature of text. Thus, it is of interest to see how well the combination methods will perform using only the classifier-based variables, which easily extrapolate to any classification problem. Secondly, the dimensionality of the meta-problem is already quite high and given the small amount of training data we have available, we seek to keep the dimensionality somewhat manageable at the meta-level.

Finally, we include the same 2 basic voting statistics reliability-indicators (*Percent-PredictingPositive* and *PercentAgreeWBest*) as discussed in Chapter 5. For the action-item problem, this yields a total of 138 reliability-indicators ($138 = 58 + 20 + 58 + 2$). With the 20 base classifier outputs, there are a total of 158 features for the Strive combiner to handle.

10.8 Experimental Analysis of Combining Action-Item Detectors

10.8.1 Classifiers

As mentioned above, the base classifiers use the same set of 5 classification algorithms used for the main Strive experiments in Chapter 7 and discussed in detail in Section 6.3. Namely, we use a decision-tree via a dependency network implementation, a unigram (multinomial naïve Bayes) classifier, a naïve Bayes (multivariate Bernoulli) classifier, a kNN classifier, and a linear SVM classifier. As done with the earlier action-item detection experiments, the kNN and SVM classifier use a normed tfidf representation. Since we apply the classifiers at both the sentence-level and document-level to a bag-of-words and bag-of-n-grams representation, we have a total of 20 base classifiers.

Also as done in Chapter 7, we try variants of both stacking and Striving using a linear SVM and a decision tree. We also present the results of the oracle *BestSelect* classifier.

10.8.2 Performance Measures

As mentioned above, the primary performance measure we are concerned with is area under the ROC curve as a measure of ranking performance. However, we present all of the performance measures used in Chapter 7 to give a complete picture.

10.8.3 Experimental Methodology

As done in Section 10.4, we perform standard 10-fold cross-validation on the set of documents. We note that the 10 folds are a new random draw and not identical to the experiments above. For the sentence-level approach, all sentences in a document are either entirely in the training set or entirely in the test set for each fold. For significance tests, we use a two-tailed t-test [YL99] to compare the values obtained during each cross-validation fold with a p-value of 0.05.

While the experiments in Section 10.4 performed nested cross-validation to automatically select the number of features used for each classification model, we instead simply choose to use the top 300 features by the chi-squared statistic at both the document-level and sentence-level for both the bag-of-words and bag-of-n-grams representation. Since there are approximately 4700 unigram tokens in total, this is roughly in-line with the level of feature reduction performed for the topic classification corpora in Chapter 7.

10.8.4 Results for Combining Action-Item Detectors

Table 10.7 presents the main summary of results. With regard to the earlier set of action-item detection experiments, several observations are worth noting. First, the *Unigram* classifier results are much higher for document-level n-gram than the earlier experiments. The reason is that the results for Section 10.7 used nested cross-validation to automatically select the number of features. In those experiments, the number of features selected for n-grams is much higher (average 1360 with some folds as low as 25 and some as high as 4000). In this situation, nested selection over the number of n-grams has much higher variance with regard to the *Unigram* model, and in earlier experiments, it often *incorrectly* overpicks the number of features. This allowed for more opportunities for double-counting evidence by the *Unigram* model (thus depressing performance).

In contrast, many of the results for *kNN* are much lower here than the earlier experiments. The reason again is in the feature selection. In the earlier experiments, the nested cross-validation correctly selected an appropriate number of features and helped *kNN* attain peak performance. The remaining models perform as would be expected from the earlier experiments. Altogether, the overall maximum performance of the base classifiers is still in line with the earlier experiments. Thus, we are still comparing to overall peak performance.

Now, we turn to the primary concern of whether Striving, and in particular, *STRIVE-S (norm)* can improve the ranking of the documents. Examining the results in Table 10.7, we see that *STRIVE-S (norm)* statistically significantly beats every other classifier according to ROC area. If we restrict our attention to just the early part of the curve, we see that *STRIVE-S (norm)* still wins but no long significantly over all the stacking models and base classifiers. This behavior in the early part of the curve is why *Stack-S (norm)* attains better performance than *Strive-S (norm)* over the linear utility functions although not significantly.

Furthermore, we can see that no method significantly beats the striving methods according to any measure. We can see this more clearly in Table 10.8 by restricting our attention to the most competitive base classifiers (the sentence-level n-gram), the default combiners, *Stack-S (norm)*, and *STRIVE-S (norm)*. While the success of the default combiners on error and F1 suggests we might be able to further exploit the potential of the underlying base models for combination, *STRIVE-S (norm)* still remains the clear best choice for ranking. While the different balance in positive/negatives appears to allow *STRIVE-S (norm)* to give more acceptable ROC performance than in topic classification, we see that the ROC area performance of *STRIVE-D (norm)* is relatively lower. This is primarily because the decision-tree method stops too early in building the decision tree because of the small number of training examples. As a result, the overall ranking has too coarse of a granularity and poor performance.

Finally, we graphically compare the ROC performance of *STRIVE-S (norm)*, *Stack-S (norm)*, and two of the most competitive base classifiers in Figure 10.5. We see that *STRIVE-S (norm)* loses by a very slight amount to *Stack-S (norm)* for the very early part of the curve but still beats the base classifiers. Later in the curve, it dominates all the classifiers. If we examine the curves using error bars (standard-deviation across cross-validation runs), we also see that the variance of *STRIVE-S (norm)* drops much faster than the other classifiers as we move to the right of the curve. Thus, across the runs *STRIVE-S (norm)* is achieving a much more consistent quality ranking than the other classifiers.

10.9 Summary

In this chapter, we first established the action-item detection problem as a text classification problem that is different from topic classification from both a semantic point of view and a machine learning perspective. We conducted experiments to demonstrate competitive baselines and to demonstrate how both n-grams and differing sentence-level and document-level views could be used to build more effective classifiers.

Next, we demonstrated that the Strive classifier combination approach is applicable to text classifier combination approaches outside of topic classification by combining the various base action-item detectors. We demonstrated Strive's flexibility and applicability to a range of problems by using it in a very "out-of-the-box" manner that required nearly no changes from early experiments. Furthermore, rather than pre-selecting the competitive base classifiers, we allowed the combination algorithm to automatically determine the weights. STRIVE-S (norm) generated document rankings with a higher ROC area and with less variation across folds than the other classifiers and combination methods.

Finally, since all of the reliability-indicators in this section are defined in terms of the base classification models, we have demonstrated that the Strive methodology is readily applicable to classification problems outside of text.

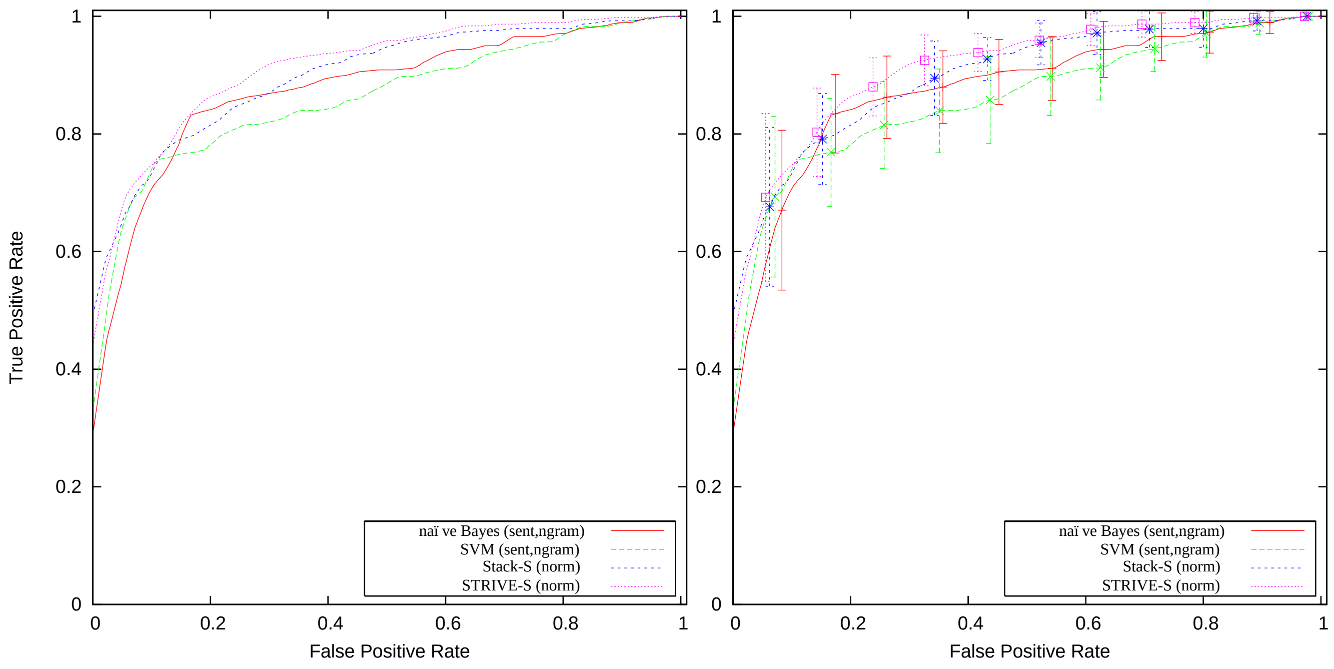


Figure 10.5: ROC curves without (*left*) and with (*right*) error bars for the action-item corpus of two of the most competitive base classifi ers versus Stacking and Striving. We see that Striving dominates the base classifi ers and only loses for a small portion of the curve to Stacking. As expected, the variance of all of the classifi ers drops as we move to the right. However, the variance for Striving drops far quicker than the others. Both argue that Striving presents the most robust ranking of the documents.

Acknowledgments

The material in this chapter is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. NBCHD030010. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency (DARPA), or the Department of Interior-National Business Center (DOI-NBC).

We would like to extend our sincerest thanks to Jill Lehman whose efforts in data collection were essential in constructing the corpus, and both Jill and Aaron Steinfeld for their direction of the HCI experiments. We would also like to thank Django Wexler for constructing and supporting the corpus labeling tools and Curtis Huttenhower's support of the text preprocessing package. Finally, we gratefully acknowledge Scott Fahlman for his encouragement and useful discussions on this topic.

	Classifiers	Document BoW	Document Ngram	Sentence BoW	Sentence Ngram
F1	kNN	0.6670 ± *0.0288*	0.7108 ± *0.0699*	0.7615 ± *0.0504*	**0.7790** ± *0.0460*
	Unigram	0.6572 ± *0.0749*	0.6484 ± *0.0513*	0.7715 ± *0.0597*	**0.7777** ± *0.0426*
	SVM	0.6904 ± *0.0347*	0.7428 ± *0.0422*	0.7282 ± *0.0698*	**0.7682** ± *0.0451*
	Voted Perceptron	0.6288 ± *0.0395*	0.6774 ± *0.0422*	0.6511 ± *0.0506*	**0.6798** ± *0.0913*
Accuracy	kNN	0.7029 ± *0.0659*	0.7486 ± *0.0505*	0.7972 ± *0.0435*	**0.8092** ± *0.0352*
	Unigram	0.6074 ± *0.0651*	0.5816 ± *0.1075*	0.7863 ± *0.0553*	**0.8145** ± *0.0268*
	SVM	0.7595 ± *0.0309*	0.7904 ± *0.0349*	0.7958 ± *0.0551*	**0.8173** ± *0.0258*
	Voted Perceptron	0.6531 ± *0.0390*	**0.7164** ± *0.0376*	0.6413 ± *0.0833*	0.7082 ± *0.1032*

Table 10.3: Average Document-Detection Performance during Cross-Validation for Each Method and the Sample Standard Deviation (S_{n-1}) in italics. The best performance for each classifi er is shown in bold.

	Document Winner	Sentence Winner
kNN	**Ngram**	Ngram
Unigram Classifier	BoW	Ngram
SVM	**Ngram**†	Ngram
Voted Perceptron	**Ngram**†	Ngram

Table 10.4: Significance results for n-grams versus a bag-of-words representation for document detection using document-level and sentence-level classifiers. When the F1 result is statistically significant, it is shown in bold. When the accuracy result is significant, it is shown with a†. This table emphasizes the hypothesis that n-grams or a "bag of words and phrases" outperforms a simple "bag of words" does, in fact, hold.

	F1 Winner	Accuracy Winner
kNN	**Sentence**	**Sentence**
Unigram Classifier	**Sentence**	**Sentence**
SVM	Sentence	**Sentence**
Voted Perceptron	Sentence	Document

Table 10.5: Significance results for sentence-level classifiers vs. document-level classifiers for the document detection problem. When the result is statistically significant, it is shown in bold. This table emphasizes the hypothesis that a sentence-level classifier outperforms a document-level classifier does, in fact, hold.

	Accuracy		F1	
	BoW	Ngram	BoW	Ngram
kNN	0.9519	0.9536	0.6540	0.6686
Unigram Classifier	0.9419	0.9550	0.6176	0.6676
SVM	0.9559	0.9579	0.6271	0.6672
Voted Perceptron	0.8895	0.9247	0.3744	0.5164

Table 10.6: Performance of the Sentence-Level Classifiers at Sentence Detection

	F1	Error	C(1,10)	C(10,1)	ROC Area	ROC Area [0,0.1]
Document-Level, Bag-of-Words Representation						
Dnet	0.7398	0.2244	0.5593	0.3856	0.8423	0.4974
Unigram	0.6905	0.3091	0.5513	0.4300	0.7537	*0.1729*
naïve Bayes	0.6729	0.2688	0.5432	0.4692	0.7745	0.2773
SVM	0.6918	0.2472	0.5392	0.3507	0.8367	0.4959
kNN	0.6695	0.3467	0.5660	0.4166	0.7669	0.2822
Document-Level, Ngram Representation						
Dnet	0.7412	0.2110	0.6228	0.4005	0.8473	0.5458
Unigram	0.7361	0.2729	0.5581	0.4719	0.8114	0.2787
naïve Bayes	0.7534	0.1896	0.5405	0.4424	0.8537	0.5069
SVM	0.7392	0.2124	0.5780	0.3361	0.8640	0.5503
kNN	0.7021	0.2539	0.5251	0.4452	0.8244	0.4607
Sentence-Level, Bag-of-Words Representation						
Dnet	0.7793	0.1894	0.5227	0.3308	0.8885	0.6164
Unigram	0.7731	0.2136	0.5815	*0.4746*	0.8645	0.4637
naïve Bayes	0.7888	0.1893	0.5653	0.4353	0.8699	0.4869
SVM	0.6985	0.1988	0.5496	0.4031	0.8548	0.5822
kNN	*0.6328*	*0.3803*	0.5629	0.4098	*0.6823*	0.2065
Sentence-Level, Ngram Representation						
Dnet	0.7521	0.1841	0.5335	0.3577	0.8723	0.5974
Unigram	0.8012	0.1868	0.6126	0.4503	0.8723	0.5362
naïve Bayes	0.8010	0.1747	0.5857	0.4018	0.8777	0.5413
SVM	0.7842	**0.1693**	0.5768	0.4073	0.8620	0.5963
kNN	0.6811	0.2647	0.5615	0.3872	0.8078	0.4424
Default Combiners						
Majority BBC	**0.8038**	0.1761	0.5511	0.3844	N/A	N/A
Best By Class	0.8006	0.1734	*0.6235*	0.3535	N/A	N/A
Stacking						
Stack-D (norm)	0.7885	0.1828	0.6046	0.3644	0.8752	0.5444
Stack-S (norm)	0.7765	0.1814	**0.4797**$_{\text{S}}$	**0.2970**	0.8996$_{\text{S}}$	0.6400$_{\text{S}}$
Striving						
STRIVE-D (norm)	0.7718	0.1949	0.5512	0.3724	0.8724	0.5333
STRIVE-S (norm)	0.7813	0.1868	0.5056	0.3134	**0.9145**$^{\text{B}}_{\text{SR}}$	**0.6436** $_{\text{R}}$
Oracle						
BestSelect	0.9894	0.0134	0.2486	0.1411	N/A	N/A

Table 10.7: Average base classifi er and classifi er combination performance during cross-validation over the Action-Item Detection Corpus. The best performance (omitting the oracle *BestSelect*) in each column is given in bold. The worst performance is given in italics. A notation of 'B', 'D', 'S', or 'R' indicates a method signifi cantly outperforms all (other) **B**ase classifi ers, **D**efault combiners, **S**tacking methods, or **R**eliability-indicator based Striving methods at the $p = 0.05$ level using a two-tailed t-test.

	F1	Error	C(1,10)	C(10,1)	ROC Area	ROC Area [0,0.1]
B: Dnet (sent,ngram)		R	B,D	B		B
B: Unigram (sent,ngram)	B,R,S					
B: naïve Bayes (sent,ngram)	R,***S***	R,S			B	
B: SVM (sent,ngram)	R,S	B,D,R,S				
B: kNN (sent,ngram)						
D: Majority	B,D,R,***S***	R,S	D		N/A	N/A
D: Best By Class	R,S	D,R,S		B,D	N/A	N/A
S: Stack-S (norm)		R	B,D,R	B,D,R	***B***	B
R: STRIVE-S (norm)	S		B,D	B,D	***B,R,S***	B,S

Table 10.8: Summary of performance on the action-item detection task. The columns show the group names for which the row method is better (restricted to just those shown here). ”Better” here means has a better average across cross-validation runs. When statistically signifi cantly better (by 2-sided t-test $p = 0.05$), results are printed in a red bold italic font.

Chapter 11

Summary and Future Work

A classification algorithm uses datapoints that have been labeled with classes by an authority or human to learn a model that, with high accuracy, can automatically predict the class the authority would have assigned to future instances. For example, the datapoint might be a particular stock and its class either “up” or “down” depending on how its value will change over the next 24 hours, or the datapoint might be a particular e-mail whose class could be the folder in which the receiver will place that e-mail. In the first case, we would like to predict which way a stock will move before it occurs, while in the second our goal is to automatically sort emails to save the user time. In both cases, these challenges can be approached by using machine learning algorithms to build a statistical model which will predict the class of an example based on past observations.

Decision trees, kNN, SVMs, language models, and naïve Bayes are a few of the classification algorithms that have been developed over the years. Generally, each of these models are designed using a different set of assumptions regarding the data. For example, they can be dichotomized as to whether they are linear vs. non-linear models, generative vs. discriminative, or high-bias vs. low-bias. While some classification algorithms often work well, none of these algorithms dominate all classification problems. Furthermore, even when one classification algorithm significantly outperforms another for a given classification problem, it is rarely the case that the worse classifier’s errors are a superset of the better. This fact has long motivated the desire to combine models in order to obtain better, or more robust, overall performance. Schemes to do this have varied widely from simple voting to metaclassifiers that model how the base classifiers interact. However, we are also faced with the result that obtaining an optimal combination or meta-algorithm over all problems is not possible [Wol95]. As a result, the key is to construct a combination algorithm that performs well with respect to many of the commonly observed behaviors within a domain.

In this dissertation, we focused on text classification, which plays a key role in a variety of applications. Furthermore, with the surge in digital text media, text classification has become increasingly important. Text classification techniques can assist in junk e-mail detection [SDHH98], allow medical doctors to more rapidly find relevant research [HBLH94], aid in patent searches [Lar99], improve web searches [CD00], and serve as a backend in a multitude of other applications. In order to effectively combine predictions within this domain, it is necessary to first understand the typical behavior of classification methods within the domain.

Therefore, in Chapter 3 we demonstrated that when recalibrating the probabilities or log-odds estimates of classification systems, an asymmetric or piecewise linear model is preferable to Gaussian or linear systems. Furthermore, we showed why recalibration is necessary for specific classifiers as well as why classifiers can be expected to behave asymmetrically in general. Finally, we demonstrated how an appropriate statistical family of asymmetric distributions could be efficiently fit to the data to yield improved probability estimates. Recalibration is an important subcase of combination, because, in addition to helping understand the nature of the classifier outputs, a metaclassifier applied to a single input *base* classifier is equivalent in many senses to recalibrating the base classifier.

In Chapter 4 we expanded upon the view of a metaclassifier applied to a single base classifier as recalibration. Extending this analysis to a series of canonical examples, we showed how calibration, dependence, and variance play a role in classifier combination. We formalized these concepts and extended them to both global and local settings. We further emphasized through synthesized data how these quantities can be precisely computed and how to make use of them in classifier combination.

Because these values cannot be computed without the true-class and posterior information, in Chapter 5 we motivated and developed a series of reliability-indicators whose goal is to capture quantities related to the local reliability, dependence, and variance of the classifiers. For example *kNNShiftStdDevConfDiff* captures how much the output of the kNN score function varies as the example being classified moves toward each of its neighbors. We go on to define 70 such variables tied to feature selection and various aspects of classification models, such as when decision trees and linear SVMs are unlikely to work well for a particular example.

We then delved into the key contribution of this dissertation in Chapter 7 where we demonstrated a metaclassifier approach, *STRIVE*, which builds a **st**acked **r**eliability-**i**ndicator **v**ariable based **e**nsemble using the classifier outputs and reliability indicators. The resulting model performs combination based on the characteristics of the particular example, and because of the reliability indicators, the model can take into account the local behavior of the base classifiers when weighting their combination. STRIVE both extends the known ceiling of performance by 3-18% across various performance measures in a variety of text classification corpora and also outperforms standard approaches such as a constant-weighted linear combination of the classifier outputs. In addition to improving prediction performance, this work also points the way for how the base classifiers can be changed to directly account for these instabilities. Furthermore, since the majority of these variables are defined in terms of model properties instead of domain properties, the approach is reusable outside of text classification.

Since many of the reliability-indicators interact with the classifier outputs in similar ways across different text classification problems, a natural question is whether the STRIVE combination model from one dataset can be transferred to another. Since labeled training data is at a premium and even more crucial for meta-models, such work can alleviate the need for training data. Using *LABEL* (Layered Abstraction-Based Ensemble Learning) in Chapter 8, we show how to do just that, and the resulting model further improves performance.

In Chapter 9, we considered adapting online methods to the problem of classifier combination. While the methods did not suffer any large losses as guaranteed, our empirical analysis highlighted the lack of competitiveness compared to the use of standard batch classification algorithms as metaclassifiers. In our analysis, we also discussed how regret guarantees with respect to average loss of the experts is far weaker than the type of guarantee with respect to loss of averaged prediction that is needed.

Since the majority of the corpora used for experimentation have been topic classification datasets, in Chapter 10 we turned to less standard text classification problems such as finding e-mails containing "action-items" and identifying the particular sentences of interest within them. We demonstrated how labeled data at the sentence-level can be used to create sentence-level classifiers that are combined into more effective document-level classifiers, how feature representation trade-offs differ in this task from topic classification, and how users aided by an action-item detection system find action-items more quickly. We then applied STRIVE to this problem in an out-of-the-box manner and also achieved performance gains for this task. The resulting combination consistently led to improved rankings with less performance variance over the training splits than the alternative measures; this provided evidence that the STRIVE system is a widely applicable approach.

Consider our thesis statement:

> Context-dependent combination procedures provide an effective way of combining classifiers that are generally superior to constant-weighted linear combinations of the classifiers' estimates of the posterior or log-odds. Furthermore,

> context can be leveraged in text classifier combination via an abstraction of the local reliability, dependence, and variance of the base classifier outputs. Finally, these abstractions help identify opportunities for data re-use that can be employed to significantly improve classification performance.

and our primary criteria for evaluation:

> As a demonstration of the suitability of these methods for text classification though, we set the goal of statistically significantly outperforming the current state-of-the-art base classification methods over several standard text classification corpora. Furthermore, since we argue that our representation of context is key, we will empirically demonstrate that these methods outperform simple constant-weighted combinations of the classifier outputs in some corpora and, in the remaining ones, achieve a statistically negligible difference.

We have clearly established that STRIVE, a context-dependent combination approach, provides an effective way of combining classifiers that is generally superior to constant-weighted linear combinations of the classifier's estimates. We also showed that it significantly outperforms the base classifiers in a variety of corpora and usually significantly outperforms linear combinations of the classifier outputs. Besides presenting arguments of the importance of the local reliability, dependence, and variance of the base classifier outputs, we also introduced reliability-indicators that capture aspects of these quantities. With our introduction of the *LABEL* extension of STRIVE to an inductive transfer framework, we showed how these abstractions help identify opportunities for data re-use and further improve performance. Thus, we have achieved all aspects of our key claims.

11.1 Key Contributions

Other combination approaches have used some form of non-constant weights on the classifiers — such as local cascade generalization [Gam98a, Gam98b], hierarchical mixture of experts [JJNH91, JJ94], and stacking using an appropriate non-linear metaclassifier [Wol92]. In contrast to these methods, our work makes two key contributions. First, we demonstrate improvement relative to state-of-the-art base classifiers that have been tuned to be as competitive as possible and not simple strawmen. Second, our approach focused on richer definitions of locality. This can be crucial since using all of the base features within the metaclassifier is often not feasible for text problems because of the high dimensionality of text. Furthermore, the low-dimensional representation of locality enables us to begin to understand why the classifiers fail in addition to benefitting in performance from detecting such failures. Additionally, in contrast to using the base features at the meta-level, the reliability indicators provide a representation which enable us to extend the methods of inductive transfer to classifier combination.

Many previous combination approaches (*e.g.* metaselection [LL01]) define features related to classifier performance, but then use those features only to select a single classifier to apply to the problem. In contrast, our approach gives a more general way to blend classifiers based on the specific documents. For features that may be relevant to choosing a classifier for a problem but static across all documents within that problem (*e.g.*, number of training examples), our work on Inductive Transfer demonstrates how to extend our combination framework to obtain the benefits of both metaselection and combination.

Typical multitask learning and inductive transfer approaches rely on the input space being the same [TO96, CK97, Car97]. We have identified how classifier combination can be transformed to be within this framework and demonstrated that it leads to performance gains when using data in conjunction across corpora. This helps alleviate the need for training data when training metaclassification models. More finely tuned inductive transfer methods may be able to offer even more improvement.

In addition to these key contributions there are a variety of other contributions throughout the dissertation. Of those, some of the more important ones follow. To our knowledge, we were the first to show the empirical asymmetry of classifier probability estimates, explain why these tend to occur, and develop parametric recalibration methods that can exploit this fact. Also, in establishing the differences between action-item detection and topic classification, we were the first to analyze in detail the trade-off between sentence-level and document-level judgments as well as analyze the types of models that result. Finally, integrating sentence-level and document-level judgments in the STRIVE framework is also one of the first examples of a combination of sentence-level and document-level models.

11.2 Directions for Future Work

There is the potential for many immediate directions for future work that are discussed throughout the dissertation. In this section, we highlight the more promising and larger scope problems.

An interesting problem is to generalize the recalibration framework discussed in Chapter 3 to include abstaining. That is, the goal would be to recalibrate a classifier's probability estimates so that the post-processing either issues an improved estimate or "abstains". Such a system would be of use in a variety of contexts — including being more readily applicable to online frameworks like sleeping experts and BMX discussed in Chapter 9. One obvious path to pursue would be to use a parametric model where points can be omitted during the training phrase according to a penalty function that increases according to the number of points omitted. Thus, learning the model would be a trade-off in the fit of the model and the omission penalty. Additionally, it could include the fit of the model that predicts when to omit if the rule for omission is data-driven.

A second promising area related to recalibration is deriving a graphical model similar to the simple model given in Figure 4.1 that treats the true log-odds as a latent variable. Because even straightforward parameterizations for this simple framework lead to the type of asymmetry seen in practice (Figure 4.2), we are hopeful that a similar graphical model will also better capture the recalibration process. After finding a formulation that is successful for recalibration, it could be extended to combination by replicating the model.

A challenging but potentially very rewarding problem is to study how the base classifier algorithms can be directly modified to capture the information some of the more useful reliability indicators are currently capturing. In particular, in Tables 7.5 and 7.6, we see that the kNN based variables and, to a lesser extent, the SVM variables seem to play an important role. If the classifiers can be directly modified, it may directly provide for more reliable predictions. Depending on what modifications are necessary, there is also a potential for a reduction in the computational cost of estimating these and training the metaclassification models.

While we have demonstrated that inductive transfer can be successfully applied to classifier combination, we have merely scratched the surface of how much further improvement can be gained via inductive transfer. Investigating whether new inductive transfer methods ([Zha05]) are applicable and can lead to further improvements in this area is a problem with large potential.

Finally, continuing to identify and define reliability-indicator variables tied to the reliability of classifiers is both challenging and an open research problem. While much of our attention has been focused on the development of these variables, it remains an attractive area of future research.

11.3 Summary

This dissertation focused on developing a metaclassification scheme, STRIVE, which used the outputs of classifiers (decision trees, kNN, linear SVMs, and two variants of naïve

Bayes) in conjunction with a set of reliability-indicators we defined. The resulting model used the training data to automatically detect regions of poor classifier reliability and generate a more reliable combined prediction. STRIVE extended the known ceiling of performance over state-of-the-art classifiers by 3-18% across various performance measures in a variety of text classification corpora. More importantly, we achieved our key goal of empirically validating the thesis that locality-based metaclassifiers generally outperform constant-weighted linear combinations of classifier outputs. Furthermore, we improved over alternative approaches such as stacking using decision trees. In addition, since labeled training data is at a premium, we demonstrated how labeled data from one problem can be inductively transferred to improve the combination model used for another problem. Finally, after illustrating how text classification tasks such as finding “action-items” in e-mail differ from more common text classification tasks like topic classification, we developed methods to effectively classify such e-mails. Using these methods as base classifiers, we then applied STRIVE in an out-of-the-box fashion to improve overall performance on this task as well. In conclusion, the STRIVE system is a widely applicable approach that has already yielded the best known performance in a number of problems and holds promise for others.

Bibliography

[Abr63] Norman Abramson. *Information Theory and Coding*. McGraw-Hill, New York, 1963.

[ACD$^+$98] James Allan, Jaime Carbonell, George Doddington, Jonathan Yamron, and Yiming Yang. Topic detection and tracking pilot study: Final report. In *Proceedings of the DARPA Broadcast News Transcription and Understanding Workshop*, Washington, D.C., 1998.

[ADW94a] Chidanand Apte, Fred Damerau, and Sholom M. Weiss. Automated learning of decision rules for text categorization. *ACM Transactions on Information Systems*, 12(3):233–251, July 1994.

[ADW94b] Chidanand Apte, Fred Damerau, and Sholom M. Weiss. Towards language independent automated learning of text categorization models. In *SIGIR '94*, pages 23–30, 1994.

[AKTV$^+$01] Khalid Al-Kofahi, Alex Tyrrell, Arun Vacher, Tim Travers, and Peter Jackson. Combining multiple classifiers for text categorization. In *CIKM '01, Proceedings of the 10th ACM Conference on Information and Knowledge Management*, pages 97–104, November 2001.

[BCB94] Brian T. Bartell, Garrison W. Cottrell, and Richard K. Belew. Automatic combination of multiple ranked retrieval systems. In *SIGIR '94, Proceedings of the 17th Annual International ACM Conference on Research and Development in Information Retrieval*, pages 173–181, 1994.

[BCCC93] N. Belkin, C. Cool, W.B. Croft, and J.P. Callan. The effect of multiple query representations on information retrieval system performance. In *SIGIR '93, Proceedings of the 16th Annual International ACM Conference on Research and Development in Information Retrieval*, pages 339–346, 1993.

[BDH02] Paul N. Bennett, Susan T. Dumais, and Eric Horvitz. Probabilistic combination of text classifiers using reliability indicators: Models and results. In *SIGIR '02, Proceedings of the 25th Annual International ACM Conference on Research and Development in Information Retrieval*, pages 207–214, August 2002.

[BDH05] Paul N. Bennett, Susan T. Dumais, and Eric Horvitz. The combination of text classifiers using reliability indicators. *Information Retrieval*, 8:67–100, 2005.

[Ben00] Paul N. Bennett. Assessing the calibration of naive bayes' posterior estimates. Technical Report CMU-CS-00-155, Carnegie Mellon, School of Computer Science, 2000.

[Ben02] Paul N. Bennett. Using asymmetric distributions to improve classifier probabilities: A comparison of new and standard parametric methods. Technical Report CMU-CS-02-126, Carnegie Mellon, School of Computer Science, 2002.

[BF95] Leo Breiman and Jerome H. Friedman. Predicting multivariate responses in multiple linear regression. Technical report, November 1995. ftp://-ftp.stat.berkeley.edu/pub/users/breiman/curds-whey-all.ps.Z.

[BFOS84] L. Breiman, J.H. Friedman, R.A. Olshen, and P.J. Stone. *Classification and Regression Trees*. Wadsworth International Group, Belmont, CA:, 1984.

[BG98] Kurt D. Bollacker and Joydeep Ghosh. A supra-classifier architecture for scalable knowledge reuse. In *ICML '98*, pages 64–72, 1998.

[BK99] Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1-2):105–139, July 1999.

[Blu97] Avrim Blum. Empirical support for winnow and weighted-majority algorithms: Results on a calendar scheduling domain. *Machine Learning*, 26(1):5–23, 1997.

[BM05] Avrim Blum and Yishay Mansour. From external to internal regret. In *Conference on Computational Learning Theory*, 2005.

[Bre96] Leo Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

[Bri50] G.W. Brier. Verification of forecasts expressed in terms of probability. *Monthly Weather Review*, 78:1–3, 1950.

[Car96] Jean Carletta. Assessing agreement on classification tasks: The kappa statistic. *Computational Linguistics*, 22(2):249–254, 1996.

[Car97] Rich Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, July 1997.

[Car02] John Carroll. High precision extraction of grammatical relations. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING)*, pages 134–140, 2002.

[CBM04] Aron Culotta, Ron Bekkerman, and Andrew McCallum. Extracting social networks and contact information from email and the web. In *CEAS-2004 (Conference on Email and Anti-Spam)*, Mountain View, CA, July 2004.

[CCM04] William W. Cohen, Vitor R. Carvalho, and Tom M. Mitchell. Learning to classify email into "speech acts". In *EMNLP-2004 (Conference on Empirical Methods in Natural Language Processing)*, pages 309–316, 2004.

[CD00] Hao Chen and Susan T. Dumais. Bringing order to the web: Automatically categorizing search results. In *CHI '00*, pages 145–152, 2000.

[CH67] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13:21–27, 1967.

[CHM97] D.M. Chickering, D. Heckerman, and C. Meek. A Bayesian approach to learning Bayesian networks with local structure. In *UAI '97, Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence*, pages 80–89, 1997.

[CK97] William W. Cohen and Daniel Kudenko. Transferring and retraining learned information filters. In *AAAI '97*, pages 583–590, 1997.

[CNM04] Rich Caruana and Alexandru Niculsecu-Mizil. Ensemble selection from libraries of models. In *International Conference on Machine Learning (ICML 2004)*, 2004.

[CORGC04] Simon Corston-Oliver, Eric Ringger, Michael Gamon, and Richard Campbell. Task-focused summarization of email. In *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*, pages 43–50, 2004.

[CS99] William W. Cohen and Yoram Singer. Context-sensitive learning methods for text categorization. *ACM Transactions on Information Systems*, 17(2):141–173, 1999.

[CST00] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, Cambridge, UK, 2000.

[CV95] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, November 1995.

[DC00] Susan T. Dumais and Hao Chen. Hierarchical classification of web content. In *SIGIR ’00, Proceedings of the 23rd Annual International ACM Conference on Research and Development in Information Retrieval*, pages 256–263, 2000.

[DF83] Morris H. DeGroot and Stephen E. Fienberg. The comparison and evaluation of forecasters. *Statistician*, 32:12–22, 1983.

[DF86] Morris H. DeGroot and Stephen E. Fienberg. Comparing probability forecasters: Basic binary concepts and multivariate extensions. In P. Goel and A. Zellner, editors, *Bayesian Inference and Decision Techniques*. Elsevier Science Publishers B.V., 1986.

[DGL96] Luc Devroye, László Györfi, and Gábor Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer-Verlag, New York, NY, 1996.

[DHS01] Richard Duda, Peter Hart, and David Stork. *Pattern Classification*. John Wiley & Sons, Inc., New York, NY, 2001.

[Die00] Thomas Dietterich. Ensemble methods. In Josef Kittler and Fabio Roli, editors, *MCS ’00, Proceedings of the 1st International Workshop on Multiple Classifier Systems*, number 1857 in Lecture Notes in Computer Science, pages 1–15. Springer, 2000.

[Dom94] Pedro Domingos. The RISE system: Conquering without separating. In *Proceedings of the Sixth IEEE International Conference on Tools with Artificial Intelligence*, pages 704–707. IEEE Computer Society Press, 1994.

[DP96] Pedros Domingos and Michael Pazzani. Beyond independence: Conditions for the optimality of the simple bayesian classifier. In *ICML ’96*, 1996.

[DPHS98] S. T. Dumais, J. Platt, D. Heckerman, and M. Sahami. Inductive learning algorithms and representations for text categorization. In *CIKM '98, Proceedings of the 7th ACM Conference on Information and Knowledge Management*, pages 148–155, 1998.

[Fla03] Peter Flach. The geometry of ROC space: Understanding machine learning metrics through ROC isometrics. In *ICML '03*, pages 194–2001, 2003.

[FMS04] Yoav Freund, Yishay Mansour, and Robert E. Schapire. Generalization bounds for averaged classifiers (how to be a Bayesian without believing). *The Annals of Statistics*, 32(4):1698–1722, August 2004.

[Fre95] Yoav Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–285, 1995.

[Fre98] Dayne Freitag. Multistrategy learning for information extraction. In *ICML '98*, 1998.

[Fri77] J.H. Friedman. A recursive partitioning decision rule for non-parametric classification. *IEEE Transactions on Computers*, pages 404–408, 1977.

[FS97] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, August 1997.

[FS99] Yoav Freund and Robert Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296, 1999.

[FSSW97] Yoav Freund, Robert E. Schapire, Yoram Singer, and Manfred K. Warmuth. Using and combining predictors that specialize. In *STOC '97, Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing*, pages 334–343, 1997.

[Gam98a] João Gama. Combining classifiers by constructive induction. In *ECML '98, Proceedings of the 10th European Conference on Machine Learning*, pages 178–189, 1998.

[Gam98b] João Gama. Local cascade generalization. In *ICML '98, Proceedings of the 15th International Conference on Machine Learning*, pages 206–214, 1998.

[GCSR95] Andrew B. Gelman, John S. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis*. Chapman & Hall/CRC, 1995.

[Goo52] I.J. Good. Rational decisions. *Journal of the Royal Statistical Society, Series B*, 1952.

[GZ86] Prem K. Goel and Arnold Zellner, editors. *Bayesian Inference and Decision Techniques: Essays in Honor Of Bruno De Finetti*. Elsevier, 1986.

[HBH88] E.J. Horvitz, J.S. Breese, and M. Henrion. Decision theory in expert systems and artificial intelligence. *International Journal of Approximate Reasoning*, Special Issue on Uncertain Reasoning, 2:247–302, 1988.

[HBLH94] W. Hersh, C. Buckley, T. Leone, and D. Hickam. OHSUMED: An interactive retrieval evaluation and new large test collection for research. In *SIGIR '94, Proceedings of the 17th Annual International ACM Conference on Research and Development in Information Retrieval*, pages 192–201, 1994.

[HCM$^+$00] D. Heckerman, D.M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie. Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research*, 1:49–75, 2000.

[HMRV98] Jennifer A. Hoeting, David Madigan, Adrian E. Raftery, and Chris T. Volinsky. Bayesian model averaging. Technical Report 9814, Department of Statistics, Colorado State University, May 1998.

[HMS66] E.B. Hunt, J. Marin, and P.J. Stone. *Experiments in Induction*. Academic Press, New York, 1966.

[HPS96] David Hull, Jan Pedersen, and Hinrich Schuetze. Method combination for document filtering. In *SIGIR '96, Proceedings of the 19th Annual International ACM Conference on Research and Development in Information Retrieval*, pages 279–287, 1996.

[HR99] David A. Hull and Stephen Robertson. The trec-8 filtering track final report. In E. M. Voorhees and D. K. Harman, editors, *NIST Special Publication 500-246: The Ninth Text REtrieval Conference (TREC-8)*, pages 35–56. Department of Commerce, National Institute of Standards and Technology, 1999.

[HS90] Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, October 1990.

[HTF01] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag, 2001.

[JJ94] Michael I. Jordan and Robert A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6:181–214, 1994.

[JJNH91] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3:79–87, 1991.

[Joa98] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *ECML '98, Proceedings of the 10th European Conference on Machine Learning*, pages 137–142, 1998.

[Joa99] Thorsten Joachims. Making large-scale svm learning practical. In Bernhard Schölkopf, Christopher J. Burges, and Alexander J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, pages 41–56. MIT Press, 1999.

[Joa02] Thorsten Joachims. *Learning to Classify Text Using Support Vector Machines: Methods, Theory, and Algorithms*. Kluwer, 2002.

[Kah04] Joseph M. Kahn. *Bayesian Aggregation of Probability Forecasts on Categorical Events*. PhD thesis, Stanford University, June 2004. Department of Engineering-Economic Systems.

[KBS97] Ron Kohavi, Barry Becker, and Dan Sommerfield. Improving simple bayes. In *ECML '97 (poster), Proceedings of the 10th European Conference on Machine Learning*, pages 78–87, 1997.

[KC00] Hillol Kargupta and Philip Chan, editors. *Advances in Distributed and Parallel Knowledge Discovery*. AAAI Press/MIT Press, Cambridge, Massachusetts, 2000.

[KKP01] Samuel Kotz, Tomasz J. Kozubowski, and Krzysztof Podgorski. *The Laplace Distribution and Generalizations: A Revisit with Applications to Communications, Economics, Engineering, and Finance*. Birkhäuser, 2001.

[Kle99] Lawrence A. Klein. *Sensor and Data Fusion Concepts and Applications*. Society of Photo-optical Instrumentation Engineers, 2nd edition, 1999.

[KMT$^+$82] J. Katzer, M. McGill, J. Tessier, W. Frakes, and P. DasGupta. A study of the overlap among document representations. *Information Technology: Research and Development*, 1:261–274, 1982.

[Lar99] Leah S. Larkey. A patent search and classification system. In *Proceedings of the Fourth ACM Conference on Digital Libraries*, pages 179 – 187, 1999.

[LC96] Leah S. Larkey and W. Bruce Croft. Combining classifiers in text categorization. In *SIGIR '96, Proceedings of the 19th Annual International ACM Conference on Research and Development in Information Retrieval*, pages 289–297, 1996.

[LCJ03] Yan Liu, Jaime Carbonell, and Rong Jin. A pairwise ensemble approach for accurate genre classification. In *Proceedings of the European Conference on Machine Learning (ICML)*, 2003.

[Lea78] Edward E. Leamer. *Specification Searches: Ad Hoc Inference with Nonexperimental Data*. John Wiley & Sons, USA, 1978.

[Lew92a] David D. Lewis. An evaluation of phrasal and clustered representations on a text categorization task. In *SIGIR '92, Proceedings of the 15th Annual International ACM Conference on Research and Development in Information Retrieval*, pages 37–50, 1992.

[Lew92b] David D. Lewis. *Representation and Learning in Information Retrieval*. PhD thesis, University of Massachusetts, February 1992. COINS TR 91-93.

[Lew95] David D. Lewis. A sequential algorithm for training text classifiers: Corrigendum and additional data. *ACM SIGIR Forum*, 29(2):13–19, Fall 1995.

[Lew97] David D. Lewis. Reuters-21578, distribution 1.0. http://www.daviddlewis.com/resources/testcollections/reuters21578, January 1997.

[LG94] David D. Lewis and William A. Gale. A sequential algorithm for training text classifiers. In *SIGIR '94, Proceedings of the 17th Annual International ACM Conference on Research and Development in Information Retrieval*, pages 3–12, 1994.

[Lit88] Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, 1988.

[LJ98] Y.H. Li and A.K. Jain. Classification of text documents. *The Computer Journal*, 41(8):537–546, 1998.

[LL01] Wai Lam and Kwok-Yin Lai. A meta-learning approach for text categorization. In *SIGIR '01, Proceedings of the 24th Annual International ACM Conference on Research and Development in Information Retrieval*, pages 303–309, 2001.

[LR94] David D. Lewis and M. Ringuette. Comparison of two learning algorithms for text categorization. In *Proceedings of the Third Annual Symposium on Document Analysis and Information Retrieval*, 1994.

[LSCP96] David D. Lewis, Robert E. Schapire, James P. Callan, and Ron Papka. Training algorithms for linear text classifiers. In *SIGIR '96, Proceedings of the 19th Annual International ACM Conference on Research and Development in Information Retrieval*, pages 298–306, 1996.

[LTB79] D.V. Lindley, A. Tversky, and R.V. Brown. On the reconciliation of probability assessments. *Journal of the Royal Statistical Society*, 142(2):146–180, 1979.

[LW94] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.

[LYJC04] Y. Liu, R. Yan, R. Jin, and J. Carbonell. A comparison study of kernels for multi-label text classification using category association. In *The Twenty-first International Conference on Machine Learning (ICML)*, 2004.

[LYRL04] David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004. http://www.jmlr.org/papers/volume5/lewis04a/lewis04a.pdf.

[LZ05] John Langford and Bianca Zadrozny. Estimating class membership probabilities using classifier learners. In *AI & Statistics*, 2005.

[Mer95] Christopher J. Merz. Dynamical selection of learning algorithms. In D. Fisher and H. Lenz, editors, *Learning from Data: Artificial Intelligence and Statistics, 5*. Springer-Verlag, 1995.

[Mer98] Christoper J. Merz. *Classification and Regression by Combining Models*. PhD thesis, University California Irvine, Information and Computer Science, 1998. http://www.ics.uci.edu/˜pazzani/merz.ps.

[Mer99] Christopher J. Merz. Using correspondence analysis to combine classifiers. *Machine Learning*, 36(1-2):33–58, July 1999.

[Mic01] Microsoft Corporation. WinMine Toolkit v1.0. http://research.microsoft.com/˜dmax/WinMine/ContactInfo.html, 2001.

[Mit97] Tom M. Mitchell. *Machine Learning*. McGraw-Hill Companies, Inc., 1997.

[MK60] M.E. Maron and J.L. Kuhns. On relevance, probabilistic indexing, and information retrieval. *Journal of the ACM*, 7(3):216–244, July 1960.

[MLW92] Brij Masand, Gordon Linoff, and David Waltz. Classifying news stories using memory based reasoning. In *SIGIR '92*, pages 59–65, 1992.

[MN98] Andrew McCallum and Kamal Nigam. A comparison of event models for naive bayes text classification. In *Working Notes of AAAI '98 (The 15th National Conference on Artificial Intelligence), Workshop on Learning for Text Categorization*, pages 41–48, 1998. TR WS-98-05.

[MP99] Christopher J. Merz and Michael J. Pazzani. A principal components approach to combining regression estimates. *Machine Learning*, 36(1-2):9–32, July 1999.

[MRF01] R. Manmatha, T. Rath, and F. Feng. Modeling score distributions for combining the outputs of search engines. In *Sigir '01*, 2001.

[MT94] Ryszard Michalski and Gheorghe Tecuci, editors. *Machine Learning: A Multistrategy Approach*, volume IV. Morgan Kaufmann Publishers, Inc., 1994.

[PD03] Foster Provost and Pedros Domingos. Tree induction for probability-based rankings. *Machine Learning*, 52(3):199 – 215, September 2003.

[PF01] Foster Provost and T. Fawcett. Robust classification for imprecise environments. *Machine Learning*, 42:203–231, 2001.

[Pla98] John C. Platt. Fast training of support vector machines using sequential minimal optimization. In Bernhard Schölkopf, Christopher J.C. Burges, and Alexander J. Smola, editors, *Advances in Kernel Methods – Support Vector Learning*. MIT Press, 1998.

[Pla99] John C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In Alexander J. Smola, Peter

Bartlett, Bernhard Scholkopf, and Dale Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 61–74. MIT Press, 1999.

[RC95] T.B. Rajashekar and W.B. Croft. Combining automatic and manual index representations in probabilistic retrieval. *Journal of the American Society for Information Science*, 6(4):272–283, 1995.

[RH00] Stephen Robertson and David A. Hull. The trec-9 filtering track final report. In E. M. Voorhees and D. K. Harman, editors, *NIST Special Publication 500-249: The Ninth Text REtrieval Conference (TREC-9)*, pages 25–40. Department of Commerce, National Institute of Standards and Technology, 2000.

[RS02] Miguel E. Ruiz and Padmini Srinivasan. Hierarchical text categorization using neural networks. *Information Retrieval*, 5(1):87–118, 2002.

[RSW02] T. Rose, M. Stevenson, and M. Whitehead. The Reuters Corpus Volume 1 – from Yesterday's News to Tomorrow's Language Resources. In *Proceedings of the Third International Conference on Language Resources and Evaluation*, 2002. http://about.reuters.com/researchandstandards/corpus/-LREC_camera_ready.pdf.

[Sch90] Robert Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[SDHH98] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A bayesian approach to filtering junk e-mail. In *Working Notes of AAAI '98 (The 15th National Conference on Artificial Intelligence), Workshop on Learning for Text Categorization*, pages 55–62, 1998. TR WS-98-05.

[Seb02] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, March 2002.

[SF95] J.A. Shaw and E.A. Fox. Combination of multiple searches. In D. K. Harman, editor, *TREC-3, Proceedings of the 3rd Text REtrieval Conference*, number 500-225 in NIST Special Publication, pages 105–108, 1995.

[SFBL98] Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686, 1998.

[Sim95] Jeffrey S. Simonoff. Smoothing categorical data. *Journal of Statistical Planning and Inference*, 47(1-2):41–69, 1995.

[SS00] Robert E. Schapire and Yoram Singer. BoosTexter: A boosting-based system for text categorization. *Machine Learning*, 39:135–168, 2000.

[STP01] Maytal Saar-Tsechansky and Foster Provost. Active learning for class probability estimation and ranking. In *IJCAI '01*, 2001.

[TH00] Kentaro Toyama and Eric Horvitz. Bayesian modality fusion: Probabilistic integration of multiple vision algorithms for head tracking. In *ACCV 2000, Proceedings of the 4th Asian Conference on Computer Vision*, 2000.

[TO96] Sebastian Thrun and Joseph O'Sullivan. Discovering structure in multiple learning tasks: The tc algorithm. In *ICML '96*, pages 489–497, 1996.

[TT95] Volker Tresp and Michiaki Taniguchi. Combining estimators using non-constant weighting functions. In *NIPS '94*, 1995.

[TW99] K.M. Ting and I.H. Witten. Issues in stacked generalization. *Journal of Artificial Intelligence Research*, 10:271–289, 1999.

[Vap00] Vladimir Vapnik. *The Nature of Statistical Learning*. Springer, New York, 2nd edition, 2000.

[vR79] C. J. van Rijsbergen. *Information Retrieval*. Butterworths, London, 1979.

[WAD$^+$99] Sholom Weiss, Chidanand Apte, Fred Damerau, David Johnson, Frank Oles, Thilo Goetz, and Thomas Hampp. Maximizing text-mining performance. *IEEE Intelligent Systems*, 14(4):63–69, 1999.

[Win69] Robert L. Winkler. Scoring rules and the evaluation of probability assessors. *Journal of the American Statistical Association*, 1969.

[WJB97] Kevin Woods, W. Philip Kegelmeyer Jr., and Keven Bowyer. Combination of multiple classifiers using local accuracy estimates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):405–410, 1997.

[Wol92] David H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

[Wol95] David H. Wolpert. The relationship between PAC, the statistical physics framework, the bayesian framework, and the VC framework. In David H.

Wolpert, editor, *The Mathematics of Generalization*, pages 117–214. Addison-Wesley, Reading, MA, 1995.

[Yan99] Yiming Yang. An evaluation of statistical approaches to text categorization. *Information Retrieval*, 1(1/2):67–88, 1999.

[YAP00] Yiming Yang, Thomas Ault, and Thomas Pierce. Combining multiple learning strategies for effective cross validation. In *ICML '00, Proceedings of the 17th International Conference on Machine Learning*, pages 1167–1182, 2000.

[YCB$^+$99] Y. Yang, J.G. Carbonell, R. Brown, Thomas Pierce, Brian T. Archibald, and Xin Liu. Learning approaches to topic detection and tracking. *IEEE EXPERT, Special Issue on Applications of Intelligent Information Retrieval*, 1999.

[YL99] Yiming Yang and Xin Liu. A re-examination of text categorization methods. In *SIGIR '99, Proceedings of the 22nd Annual International ACM Conference on Research and Development in Information Retrieval*, pages 42–49, 1999.

[YZCJ02] Y. Yang, J. Zhang, J. Carbonell, and C. Jin. Topic-conditioned novelty detection. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, July 2002.

[ZE01] Bianca Zadrozny and Charles Elkan. Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. In *ICML '01*, 2001.

[ZE02] Bianca Zadrozny and Charles Elkan. Reducing multiclass to binary by coupling probability estimates. In *KDD '02*, 2002.

[Zha05] Jian Zhang. Sparsity models for multi-task learning. In *NIPS '05*, 2005.

[ZO01] Tong Zhang and Frank J. Oles. Text categorization based on regularized linear classification methods. *Information Retrieval*, 4:5–31, 2001.

[ZY04] Jian Zhang and Yiming Yang. Probabilistic score estimation with piecewise logistic regression. In *International Conference on Machine Learning (ICML 2004)*, 2004.