Interactive Machine Learning in Diamond

Shiva Kaul

CMU-CS-10-120
May 2010

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Mahadev Satyanarayanan, Chair
Rahul Sukthankar

Submitted in partial fulfillment of the requirements for the degree of Master of Science.

This research was supported by the Braam Family Fund and by the National Science Foundation (NSF) under grant number CNS-0614679. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily represent the views of the NSF or Carnegie Mellon University.

Keywords: Information retrieval, Active learning, Human-computer interaction

Contents

List of Figures

Abstract

Unindexed search systems, such as Diamond, are more useful than indexed search systems precisely when the cost of indexing cannot be amortized and classifiers are inexpensive to create. This thesis establishes the latter condition for many image classification tasks. To accommodate a wide variety of visual phenomena, a flexible, learned image representation, Semantic Texton Forests, is adapted for use in Diamond. To reduce the amount of interaction required to produce a high-quality classifier, a novel active learning algorithm, Active Learning by Measure Approximation, is theoretically developed. To consolidate all components of the system, a usable interface, Algum, is implemented. The result is an effective workflow realizing the Diamond vision of iterated, interactive hypothesis exploration.

Acknowledgements

First, I would like to thank Satya, who was a great advisor and mentor. Due to his imagination and ability to effortlessly context switch among different topics, I was able to explore machine learning as part of a systems group. He embodies a great deal of engineering wisdom which positively impacted my thesis.

Rahul Sukthankar was an amazingly helpful guide to many topics in computer vision, machine learning, and information retrieval. No matter how hectic his travel schedule was, he was always available for discussion. Without him, we wouldn't have access to the vast amount of computing power at Intel Research.

Adam Goode has been my page fault handler ever since I became a graduate student. Adam, Jan Harkes, and Benjamin Gilbert are all very knowledgeable, talented, fun to converse with, and tolerant of my absurd questions. Wolfgang Richter, my good colleague and officemate, helped conduct many experiments in this thesis.

I am very grateful to Mark Stehlik, who enabled and supported my academic development. Pranjal Awasthi offered a lot of good feedback and discussion on my research direction. I am inspired by the work of my Carnegie Mellon colleagues Andreas Krause, Steve Hanneke, Nina Balcan, and Carlos Guestrin.

Our real-world applications were facilitated by the Department of Dermatology at UPMC. Drazen Jukic, Jonhan Ho, Laura Drogowski were all very patient and fun to work with.

Finally, I'd like to dedicate this thesis to my family: Rakesh, my father; Sushma, my mother; Dhruva, my brother; Ashok, my uncle; and Veena, my aunt.

Shiva Kaul
Pittsburgh, PA
May 2010

Chapter 1

Introduction

This primary goal of this thesis is to enable untrained users to easily produce high-quality image classifiers. This goal emerges from a comparison of Diamond [31], an unindexed search system, with traditional indexed search systems.

1.1 The Diamond System for Unindexed Search

Standard search systems are indexed; they maintain auxiliary metadata structures to enable efficient operations upon the stored data. These systems are typically designed for relatively stable datasets and rigid search tasks for which the index can be optimized. Given an index, a search task can be formalized as a retrieval or ranking problem. The answers to these problems are respectively evaluated by metrics such as precision/recall and discounted cumulative gain.

Indexed systems are not appropriate for many natural, modern search tasks. Real world data is often volatile; for example, consider searching streaming data or monitoring network events. Users may not start with a well-defined search task, in which case search behavior is better described as an iterated process of discovery and refinement. In many scenarios, computational resources are cheap relative to human attention. Finally, and most importantly, users often wish to perform very flexible queries for which establishing an index is difficult. Some of these difficulties are practical; for example, it is not always straightforward to establish indices for multimedia search [21]. Further difficulties are theoretical. The bias-variance (here, quality-flexibility) tradeoff is exacerbated for indices which are established before queries are answered. For high-dimensional data, concentration of measure - a statistical phenomenon that keeps random elements close to their expectations with high probability - prevents pivot-based indexing schemes from outperforming brute force [63].

The Diamond system addresses search tasks in this second regime. A Diamond search usually starts where an indexed search has ended. For example, if the user wants to find pictures of shiny metal objects, a web API call to Flickr can return a set of images tagged with 'metal'. This set is called the scope of the Diamond search. The user produces some *filters*: classifiers parametrized by thresholds. In our example, the filters could be an edge detector and an RGB histogram, where the threshold units are 'number of edges' and Euclidean distance. The classifiers are assembled in series to form a conjunction, or *searchlet*, which is shipped to servers and evaluated upon the scoped data. Results are streamed to the user and cached. Upon seeing the results, the user may refine his searchlet and iterate the search process. Since Diamond searches are unbounded, they cannot be formulated as retrieval or ranking problems. Nevertheless, given a searchlet, a single Diamond search can be evaluated in terms of precision/recall (for example) and the rate at which results appear.

As mentioned previously, indexed and unindexed search systems are complementary. There is, however, a threshold at which unindexed search becomes more economical than indexed search, and vice versa. This threshold is determined by the cost of indexing relative to the cost of producing and executing an effective searchlet. Here, cost is a holistic measurement not only of computational expenditure but also organizational and mental effort. Diamond is useful precisely when the costs of indexing cannot be amortized and good searchlets can be inexpensively created. Establishing the latter condition is the focus of this thesis.

1.2 Producing Image Searchlets in OpenDiamond

The OpenDiamond platform is a concrete implementation of the Diamond architecture [52]. It consists of core Diamond infrastructure as well as client applications optimized for specific search tasks. This thesis focuses on image search, though some general principles herein described are also applicable to different search tasks. I will henceforth restrict discussion to cheaply producing good image searchlets.

The problem of translating domain knowledge into a searchlet decomposes into two subproblems: (1) choosing a suitably flexible searchlet space, and (2) finding an appropriate searchlet within that space. Previous approaches addressed only the first problem. One approach involves a toolbox of computer vision algorithms included with OpenDiamond. These algorithms include RGB histogram, Gabor texture, and face recognition. These basic, well-known algorithms functioned as primitive filters from which more powerful searchlets could be assembled in a 'mix-and-match' fashion. Another approach involves high-level imaging languages such as ImageJ and MATLAB. The user is free to upload handcrafted code in these languages.

Since the second problem is left open, neither of these solutions is suitable for direct use by untrained users. Currently, the true 'user interface' to Diamond is a computer vision expert to whom a classification task is explained. This expert subsequently designs a method customized for the search task. While he was at it, he may as well have constructed an indexing scheme.

Summoning a computer vision expert to conduct a search is impractical and anathema to the principles of Diamond. In order to be useful, Diamond must help the user select a searchlet, assuming the user is an expert in his domain and nothing else. Such automated

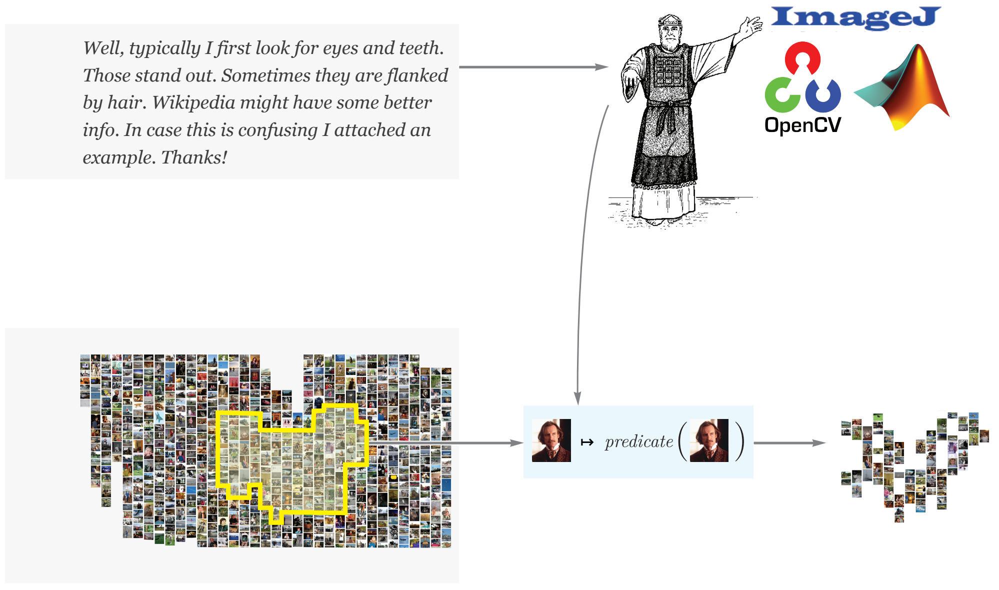

Well, typically I first look for eyes and teeth. Those stand out. Sometimes they are flanked by hair. Wikipedia might have some better info. In case this is confusing I attached an example. Thanks!
ImageJ
OpenCV
↦ predicate

Figure 1.1: The Diamond workflow before this thesis.

assistance based on formally-provided domain knowledge is the province of machine learning. Though learning methods are popular in computer vision and information retrieval, a few serious issues hinder their application to Diamond.

- Usability: Diamond is a complete system, not an algorithm. Any use of machine learning must be accompanied by sensible user interfaces.
- Quality: though there are a plethora of learning-based vision algorithms for classification, one must be chosen which meets Diamond's requirements for flexibility and quality. Remaining practical issues must be addressed.
- Sample complexity: creating a good searchlet may still be too time-consuming if a large number of labels are required.

1.3 Main Contributions and Organization of Thesis

This thesis addresses each of the enumerated issues and implements Algum[1], a compelling software system beside Diamond. The result is an improved workflow for conducting image search in Diamond.

In chapter 2, I examine the deficiencies of existing work through a test deployment. In chapters 3 and 5, I describe two modern machine learning algorithms for use in this setting. The first is Semantic Texton Forests, an existing computer vision algorithm which is

[1]Strictly, Algum refers to the web application. Loosely, it refers to the whole training system.

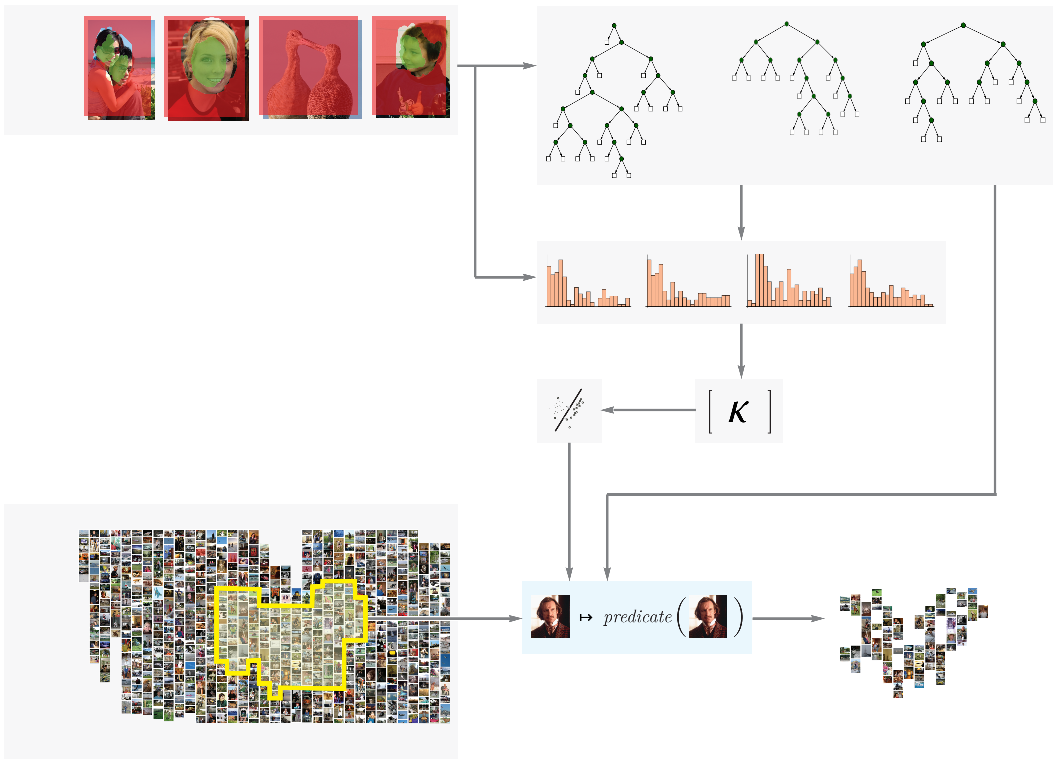


Figure 1.2: The Diamond workflow after this thesis.

adapted for use in Diamond. The second is Active Learning by Measure Approximation, a novel learning algorithm which reduces the number of labels required to produce a good searchlet. In chapter 4, I describe Algum, a web application that realizes the Diamond vision of interactive hypothesis exploration.

Chapter 2

Lessons from LANL's ISIS

Plan to throw one away. You will anyhow.
– Fred Brooks, *The Mythical Man Month*

ISIS (Intelligent Searching of Images and Signals) is a research project hosted at Los Alamos National Laboratory. It was most active between 1999 and 2004. It produced a complete, open-source software suite including (1) Genie: a learning algorithm based upon genetic programming (2) Aladdin, a user interface for producing input for Genie [50]. I investigated them as an existing system whose appeals and flaws could be observed as an informal guide for my own research. My findings, simple though instructive, are summarized in this brief chapter.

2.1 Image representation and classification with Genie

Image representation refers to a mathematical description of the contents of an image. One simple example is the bag of colors, which, for a set of colors, counts how many pixels have that color. More advanced representations include SIFT and SURF [6]. It is not unusual to compose simple representations to form more complicated ones. These representations are often handcrafted by experts to isolate a specific visual phenomenon.

When one algorithm must simultaneously handle a wide variety of visual phenomena, as in the Diamond setting, the image representation is often learned at runtime rather than fixed during development. Genie is one such learning method. Genie asserts that the essence of an image can be extracted by a circuit (a directed acyclic graph from inputs to output) composed from primitive image transformations. These primitive transformations come from basic arithmetic, morphology (e.g. erosion, dilation), signal processing (e.g. low-band filters), and related fields. The quality of the representation is judged by the performance of a resultant classifier on the training data, which take the form of figure 2.2.

Now that a space of hypotheses (the circuits) and an objective (empirical classifier performance) have been defined, learning an image representation has been formulated as an

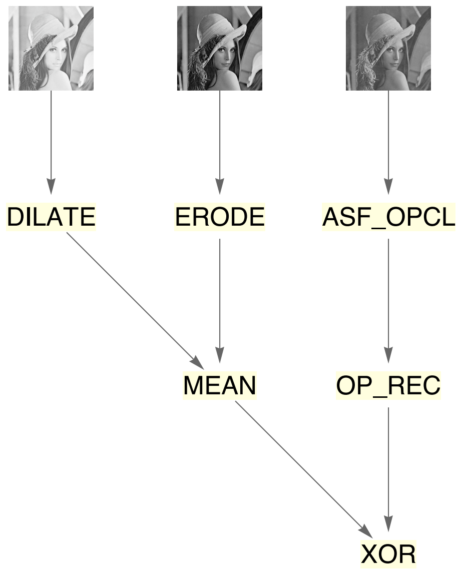
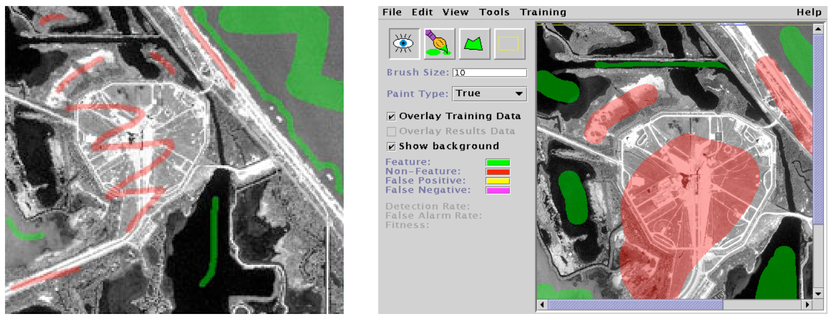


Figure 2.1: A simple circuit composed from image transformations.



Brush Size: 10
Paint Type: True
☑ Overlay Training Data
☐ Overlay Results Data
☑ Show background

Figure 2.2: Partially labeled image used as training data, and the Aladdin interface for producing them [50].

optimization problem. Genie solves this problem using evolutionary techniques, which take random hypotheses and iteratively refines them through a process of pruning and mutation. More specifically, Genie is a kind of genetic programming. It is ‘genetic’ in the sense that the mutations involve pairs of hypotheses and are analogous to biological recombination. It is ‘programming’ in the sense that a circuit is executable. Recent results in computational learning theory suggest that Genie can learn the correct circuit ‘efficiently’, at least if the mutations are merely random [61]. Genie is implemented in Perl, C and IDL, a scientific language similar to MATLAB.

2.2 Labeling with Aladdin

Genie labels are represented with FITS, a file format for dense matrices. FITS labels are produced and manipulated by Aladdin, a user interface written in Java. Aladdin features a paint-style interface for producing labels. By contrast, other interfaces, such as LabelMe [51], utilize vector shapes for labeling contiguous regions.

2.3 Use of ISIS software

One compelling application of Diamond is interactive, search-assisted diagnosis (ISAD) in clinical pathology. Pathologists perform a classification task upon slides of skin tissue; they must determine if the samples are indicative of a malady such as cancer. Occasionally, the classification is based on a common visual phenomenon. However, most clinical time and effort is spent on corner cases. In these situations, pathologists currently conduct manual, time-consuming searches of medical case history for strange visual phenomena.

We applied the ISIS software to this real-world scenario in order to establish priorities for a new system. Pathologists at the University of Pittsburgh Medical Center used Aladdin to label scanned slides, then uploaded the labels to a web server. They found the labeling interface intuitive, but felt the uploading procedure was cumbersome; to label a single image, they had to find the image file in Aladdin, label the image, save the label to a file, find the image file in a web browser, find the label file in a web browser, hit upload, and wait.

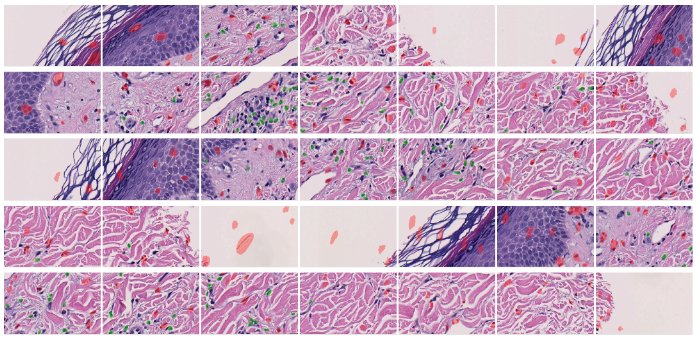


Figure 2.3: Labels isolating Eosinophils provided by a clinical pathologist.

In order to use Genie from within Diamond, IDL code was replaced with MATLAB code, and the FITS format was replaced with the MAT format. Even so, Genie was generally unwelcome on Diamond servers because the portions written in C require deprecated libraries. Though individual instances of Genie are single-threaded, multiple processes can be forked, both locally and remotely, in order to exploit parallelism.

2.4 Summary of Findings

From our use of Genie and Aladdin, three criteria emerged for a new system.

- Scalability: Genie is embarassingly parallel. We found this feature very compelling to the Diamond setting, where computational resources are plentiful relative to user attention.
- Paint-style interface: users found this modality of input preferable for binary labels.
- Integrated label/train interface: saving a label and providing it to Genie take almost as much time as labeling itself. This transaction cost should be eliminated.
- Modern implementation: deprecated libraries, proprietary programming languages, and obscure file formats should be replaced with open, modern alternatives.

Some of the work in this thesis may be viewed as improvements upon the ISIS software. The image representation algorithm described in chapter 3 supplants Genie. The user interface described in chapter 5 does the same to Aladdin.

Chapter 3

Image Representation by Semantic Texton Forests

As described in Chapter 2, image representations are mathematical descriptions of the contents of an image. Such representations are used to learn a classifier. In order to achieve the flexibility mandated by Diamond, image representations are themselves learned at runtime rather than fixed for a specific application.

Semantic Texton Forests are a learned image representation developed by Jamie Shotton and Matthew Johnson [56]. (I shall use STF to refer both to the image representation and the method of learning the representation.) As its name suggests, STF represents images using trees that are grown from training data. It offers the following advantages over methods such as Genie:

- Quality: STF is considered to be competitive by computer vision practicioners [58].
- Functionality: in addition to image representation, STF prescribes a method for classification.
- Implementation: A reference implementation of all of the above is freely available under the GPL license.
- Simplicity: STF is a simple machine learning algorithm. It does not rely on complicated procedures with origins in morphology, signal processing, or even continuous mathematics.

This chapter gives an overview of how STF is used to represent and classify images. It describes improvements we made both to the algorithm and the reference implementation, as well as how the algorithm is integrated into Diamond.

Parameter	Description	Default value
d	box size	15
D	max tree depth	10
T	number of trees	5
p	data per tree	0.25
s	sampling step	8
m	feature count	400

Figure 3.1: Input parameters to STF training.

3.1 Training algorithm description

As input, the STF training procedure takes partially labeled images such as the one in figure 2.2. The procedure has three phases. In the first phase, a forest is learned from the training data. This forest is used to compile images into histograms. In the second phase, the histograms are processed into a Mercer kernel, which basically serves as a similarity measure among the images. In the third phase, the kernel is used to learn a binary classifier. Both the forest and the classifier are returned as output.

The training procedure uses the parameters in figure 3.1, which are described later in more detail.

3.1.1 Learning the forest for compiling images into histograms

The forest consists of T decision trees whose decisions are based on simple arithmetic operations such as addition and subtraction. After the training sample is preprocessed, each tree is learned from a subsample. The resulting tree, when applied to an image, returns a histogram expressing some spatial, hierarchical structure.

Preprocessing

First, the images are converted from RGB to LAB. The latter is perceptually uniform, so Euclidean distance on the color coordinates corresponds to visually perceived distance [45]. Next, every sth labeled pixel is extracted from each image, except for those in the $d/2$-pixel border because the algorithm will consider pixel neighborhoods of size d. An imbalance in the training data – e.g. 5000 pixels of one class and 500 of another – can be mitigated by assigning higher weight to the pixels in the underrepresented class. Finally, geometric and photometric transformations of the training data are calculated and themselves added to the training set. This offers crude but adequate protection from perturbances and inconsistencies related to scale (i.e. zoom) and exposure (i.e. lighting).

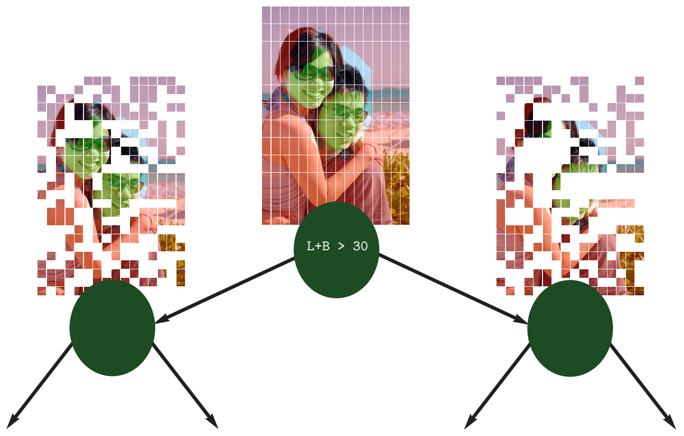


Figure 3.2: Information gain is used to choose decisions.

Training

In order to produce T trees, T subsamples are drawn from the training data in a process called binomial resampling: for each subsample, every pixel is sampled independently with probability p. The subsamples may therefore overlap. Each subsample is independently used to learn a binary decision tree. Each decision node is a thresholded arithmetic operation upon the LAB values of a pixel and possibly those of a neighbor within a square of side length d. For example, a pixel could be sent left if 30 is more than the sum of its L component and its neighbor's B component, and sent right otherwise. Sum, difference, absolute difference, and unary selection are the primitive operations. One of these is chosen for each node in a greedy fashion. That is, without considering future nodes, the algorithm chooses the decision that 'best separates' pixels of different classes. Here, 'best separates' means maximal empirical information gain. Suppose a node is given 100 pixels of one class and 100 pixels of another. This sample has high uncertainty or, precisely, entropy. A node with high information gain would separate the sample into two nodes with distributions having lower entropy, ideally sending all pixels of one class to the left, and the other class to the right.

Choosing the node with maximal information gain is an optimization problem. For the sake of simplicity and computational efficiency, this optimization is crudely approximated by a sampling procedure: randomly sample a 'fair amount' (m) of decisions and choose the one with the highest information gain. In typical decision-tree-growing fashion, the pixels that fall below the threshold are used to train the left subtree, and the rest are used for the right subtree. This continues until one of the following conditions is met: (a) a maximum depth D is reached, (b) all the pixels have the same label, or (c) discriminative decisions aren't generated.

Every node can be assigned a position identifier that is global to all binary trees. For example, the node found by traveling left twice and right once from the root can be assigned

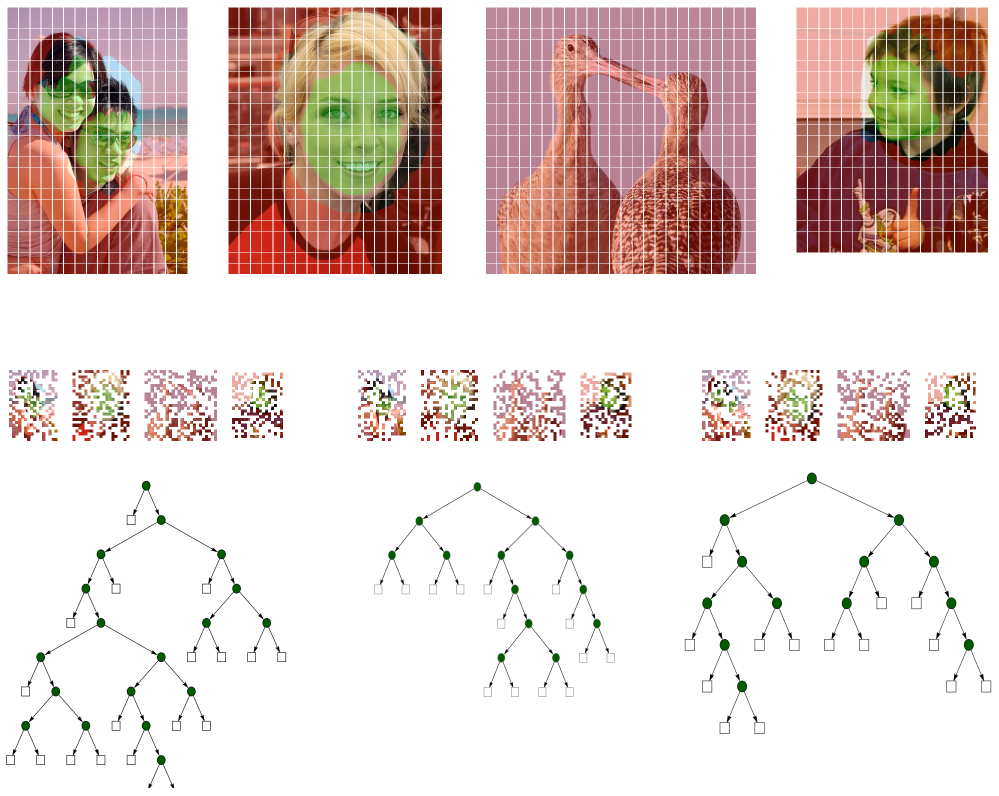


Figure 3.3: Binomial resampling and internal randomization are the basis of the ensemble method.

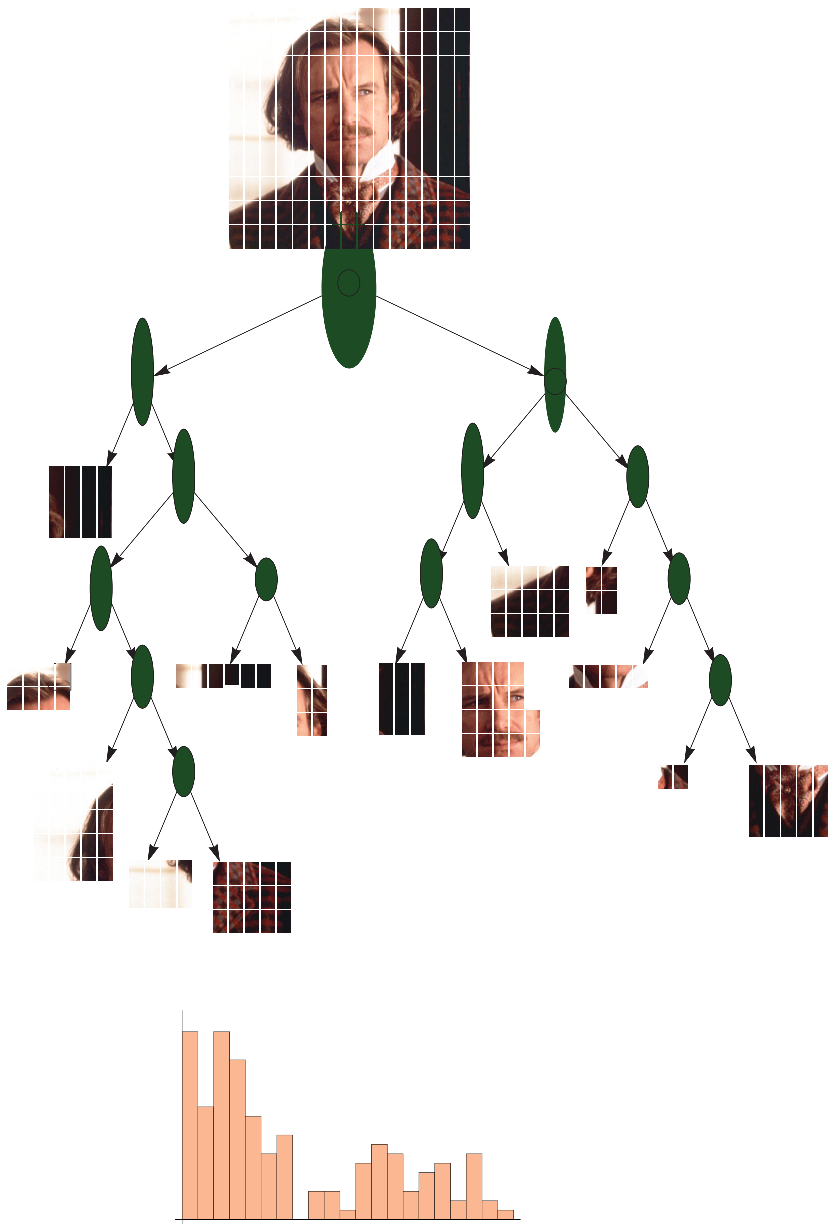


Figure 3.4: In the histograms, the x-axis is the position identifier of each node, and the y-axis is a count of visits of pixels to the node.

001. Given an image, consider the number of pixels that travel to each node. The map from node positions to counts is a histogram (or "bag of semantic textons") which captures some kind of hierarchical or structural information about the image. Since D is the maximum depth, histograms are values in $\mathbb{Z}_{+0}^{2^{D+1}-1}$. The sum of all the histograms is henceforth used as a surrogate for the image.

3.1.2 Measuring Histogram Similarity with the Pyramid Match Kernel

Now that histograms are the primary object of interest, a notion of similarity between histograms will be introduced in order to learn a classifier. This similarity measure will be defined so that flexible, nonlinear classifiers can be learned while retaining the computational efficiency of linear methods. Kernel methods are the most well-known and well-studied formalism for achieving this [55]. The main intuition behind kernel methods is that nonlinear classifiers upon the data (i.e. histograms) are dual to linear classifiers in a high-dimensional (perhaps infinite-dimensional) space. Let $\phi(h)$ be a map from histograms into this high-dimensional 'feature' space. The inner product in the feature space, $\langle \phi(h_1), \phi(h_2) \rangle$, is a natural similarity measure; it is highest when its arguments are equal, and zero when its arguments are orthogonal. Because the feature space is high-dimensional, it may be computationally intractable to map the data into the feature space with ϕ and then directly compute the inner product. Attention is restricted to feature spaces whose inner product can be efficiently computed from the data without the feature map. These are "reproducing kernel Hilbert spaces" defined by an efficiently computable 'kernel' function $\kappa(h_1, h_2) = \langle \phi(h_1), \phi(h_2) \rangle$.

Since the histograms can be high-dimensional, the usual ℓ_p-induced inner product may not be appropriate as a kernel. The fundamental problem is that the histograms are really bags, not vectors, and defining similarity for bags is not straightforward. The pyramid match kernel is commonly used in this scenario. I omit details on this construction, referring readers to the original paper by Grauman and Darrell [26]. It is essentially a multilevel match between two 'pyramids' or progressively scaled histograms.

Given κ, the kernel matrix $K = \{\kappa(h_i, h_j)\}$ is computed. This distills all the feature information into a matrix of real numbers. It is the object upon which a classifier will be learned.

3.1.3 Classification with a support vector machine

Now that our data points are points in some reproducing kernel Hilbert space, a hyperplane (aka halfspace, linear separator) represents a classifier. 'Support vector machine' is a fancy name for a simple hyperplane with some remarkable properties [62]. Let the margin between a data point and the hyperplane be how far (according to an inner product) the point is from being on the wrong side of the hyperplane. The SVM maximizes this margin

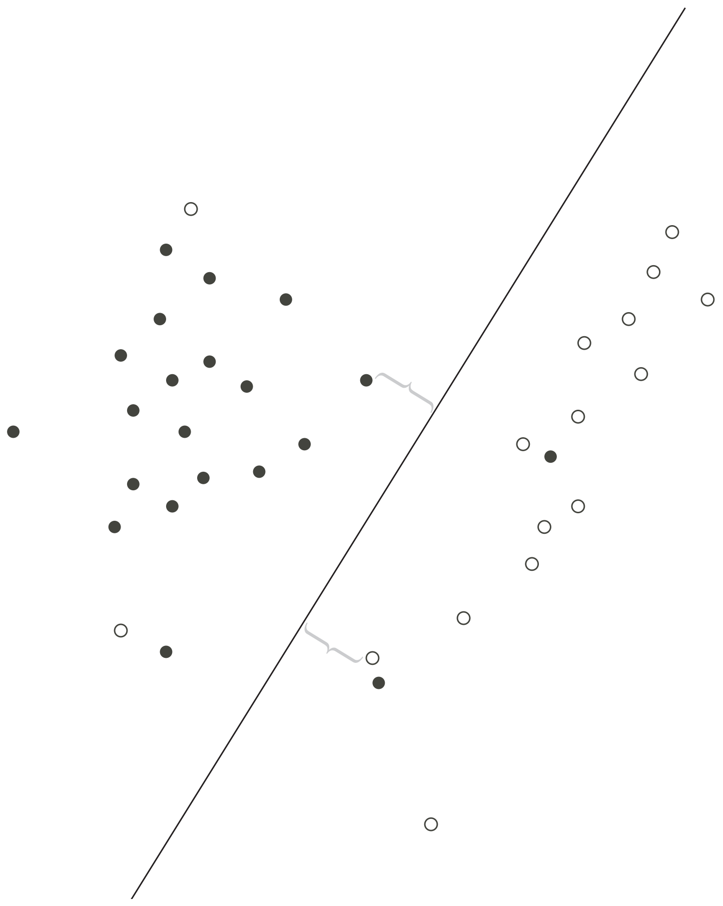


Figure 3.5: The support vector machine.

over all the data points, and thereby earns the nickname ‘maximum margin hyperplane’. SVMs are no longer considered state-of-the-art for most specialized classification tasks. However, they are still considered an excellent general purpose technology, are widely understood, and have fast, stable implementations. The SVM enjoys a vast swath of practical and theoretical justification:

- Insofar as they represent classifiers, hyperplanes are rather uncomplicated objects. This intuitive notion is formalized by concepts such as VC dimension [62] and Rademacher complexity [55]. Analysis of such properties shows that SVMs have low capacity for overfitting.
- In some sense, the size of the margin represents how well the SVM was able to separate the training data. A high margin is evidence of benign learning conditions, which can form the basis of statistical quality guarantees [5].
- A hyperplane can be defined using inner products such as the kernel function defined in the previous section. This makes it possible to learn an SVM in a high-dimensional space.
- Finding the SVM is a convex optimization problem, so it can be solved quickly.
- Finding the SVM is also a regularization problem which leads to sparse solutions. (“Support vector” refers to the handful of data points which actually end up defining the hyperplane.) This sparsity yields even more performance guarantees, both statistical and computational [55].

Given the kernel matrix, solving the convex optimization problem for SVM results in a classifier. This classifier and the decision tree forest are returned as output from the entire training procedure. In order to classify a new image, the forest compiles the image into a histogram, and the classifier is applied to the histogram via the pyramid match kernel.

3.2 Parameter Improvements

Potential algorithmic improvements to STF are discussed in chapter 6. But even without changing the algorithm, statistical improvements can be realized by adjusting the parameters. These should reflect the scarcity of data and the relative abundance of computational resources in the Diamond setting. For example, the default sampling step $s > 1$ hastens training by throwing away training data; this should be amended by setting $s = 1$.

One parameter which merits experimental investigation is T, the number of trees. The justification for learning an entire forest, rather than just a single tree, comes from the theory of ensemble methods, where the output of many individual hypotheses is aggregated. (You may have heard of ‘bagging’ [9], which STF technically is not; see chapter 6 for more details.) Such methods are generally believed to earn lower variance by their greater

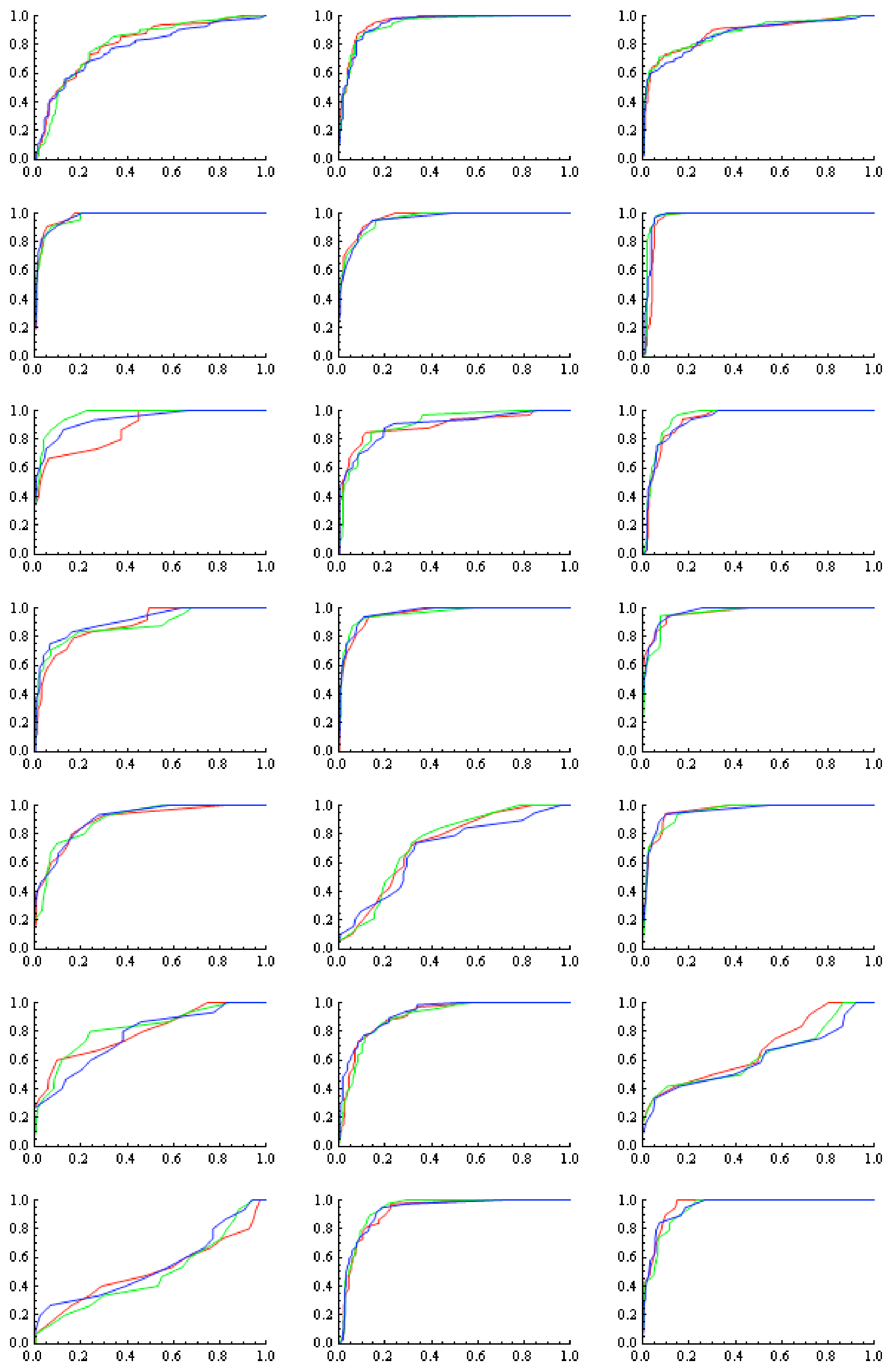


Figure 3.6: STF's performance upon the Microsoft Research Cambridge Object Recognition Image Database, Version 2. Each plot is a collection of empirical ROC curves for a single label class (e.g. cows, flowers). Within each figure, curves with the same color were produced using the same run, i.e. the same `DecisionForest` and `Categorizer`. The runs are different solely due to internal randomization.

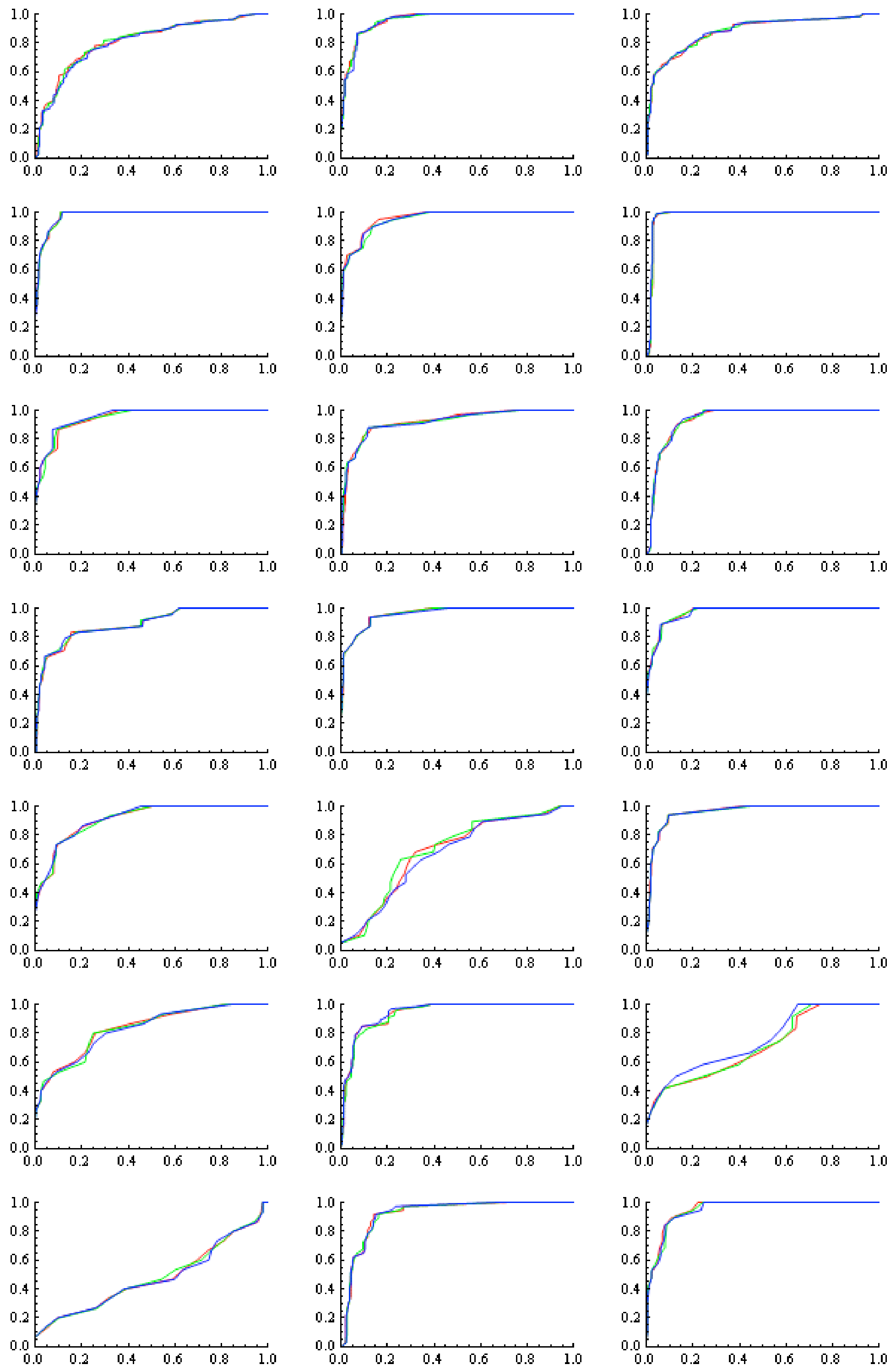


Figure 3.7: The experiment in figure 3.6 is repeated with $T = 50$. Notice how the curves are closer together.

computational effort. Such beliefs are founded, one way or another, on the law of large numbers (LLN) with respect to T.

The default $T = 5$ seems far too low to invoke the LLN. A value of $T = 50$ may be a better choice for Diamond. This hypothesis is initially investigated on a basic experiment upon a dataset from Microsoft Research. In figures 3.6 and 3.7, the setting of T does not appear to influence the average position of the ROC curves, which is excellent in either case. However, with $T = 50$, there is less variance among the runs.

The initial investigation motivates a larger experiment on a harder dataset. In figures 3.8, 3.9, and 3.10, all settings of T lead to acceptable average positions of the ROC curves. However, all settings also produce poor results on at least one run. This suggests a deficiency in the subsampling method; see chapter 6 for discussion. Nevertheless, $T = 50$ again achieves lower variance than the other settings.

These experiments suggest that increasing T results in better statistical performance at the cost of worse computational performance. The latter is addressed in the implementation improvements of the next section.

3.3 Implementation Description

Shortly after the publication of Semantic Texton Forests, the authors released an open-source reference implementation in C#. This implementation consists of three libraries, each licensed under the GPL.

`SVM.dll`: a port of the Java LIBSVM library produced using an automated source code translator.

`VisionNET.dll`: a standard library of vision-related classes which are typically included in specialized languages such as MATLAB but not in general-purpose languages such as C# or Java. These classes include includes classes for labeled image datasets, kernel methods, decision trees, and performance evaluation. It also includes implementations of core image processing algorithms such as color conversion and convolutions.

`SemanticTextonForests.exe`: an implementation of the Semantic Texton Forests organized as follows. Unlabeled images are loaded as `Bitmap`s. The labels, being PNGs, are also loaded as `Bitmap`s and subsequently wrapped as `LabelImage`s. The unlabeled images and labels are paired together as `LabeledImage`s, which are all assembled into a `LabeledDataSet`.

Each pixel comparison operation is generated by `BinaryImageFeatureFactory`, and each pixel inspection operation is generated by a `UnaryImageFeatureFactory`. These are all assembled into a `CombinationFeatureFactory`. T subsamples of `ImageDataPoint`s are produced from the `LabeledDataSet`. The `CombinationFeatureFactory` and each list of `ImageDataPoint`s is used to learn a `DecisionTree`. At each stage of the learning process, `IFeature`s are drawn from the `*FeatureFactory`'s and the best one used to form a `Decision` wrapped in a `DecisionTreeNode`. The `DecisionTrees`

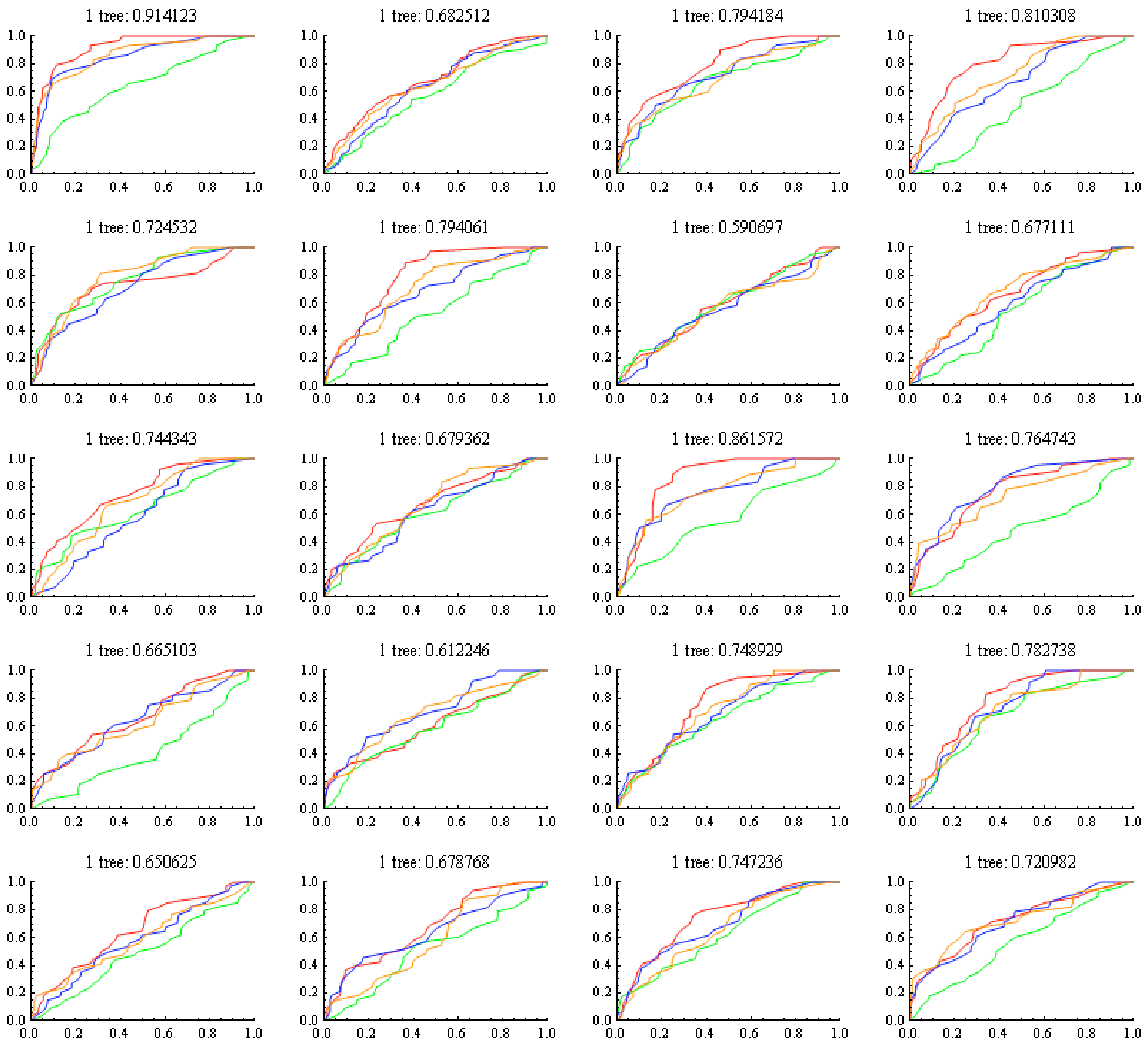


Figure 3.8: The same experiment as in 3.6 is repeated upon the VOC 2009 dataset with $T = 1$. This dataset is harder and STF's performance is worse.

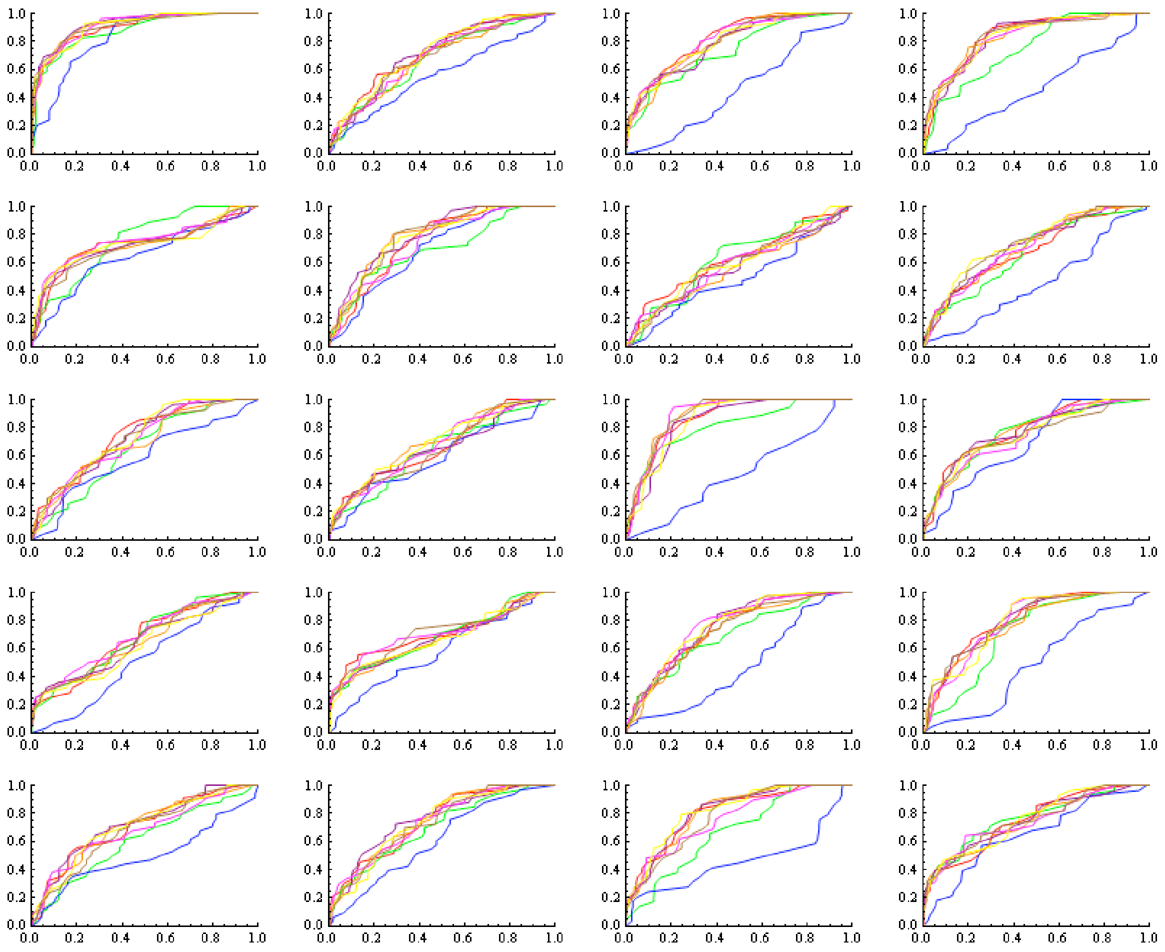


Figure 3.9: The same experiment as in 3.8 is repeated with $T = 5$.

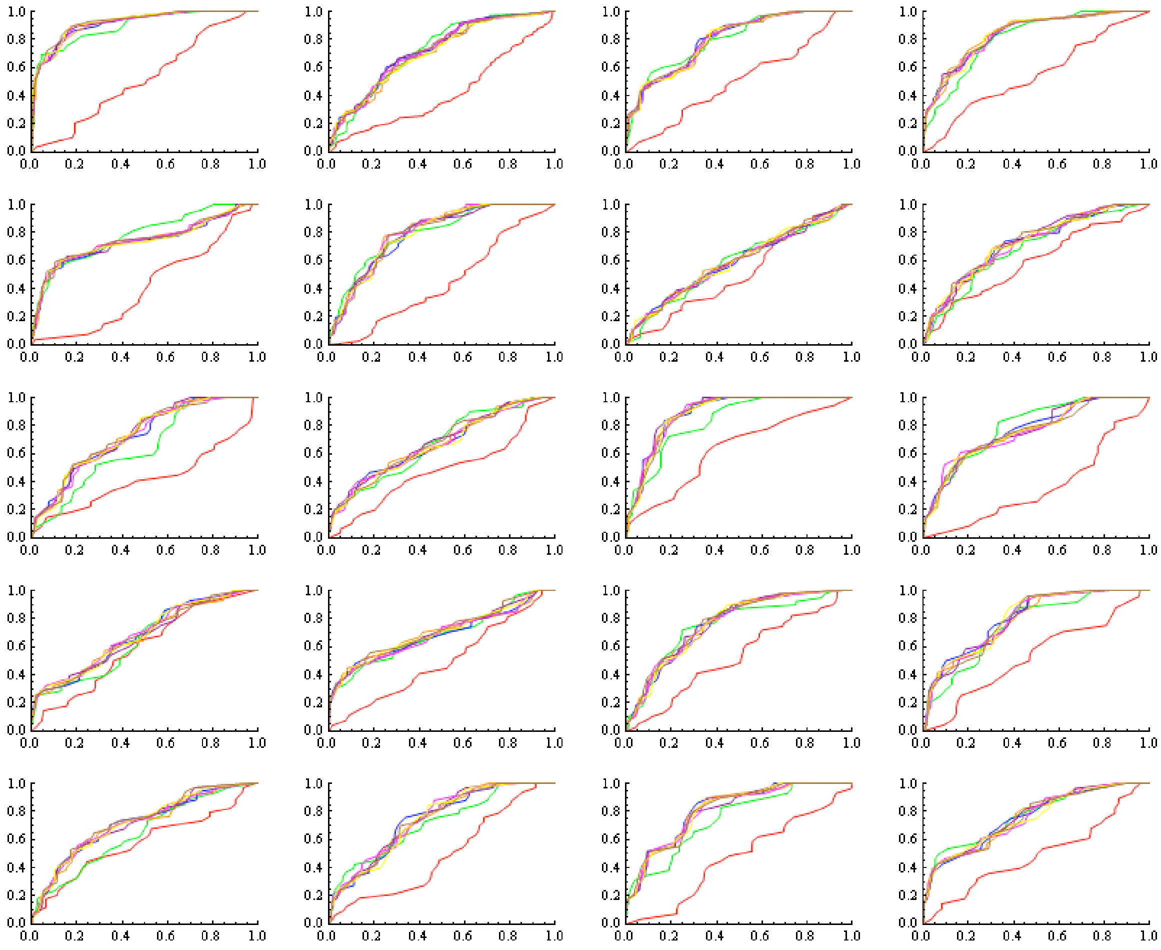


Figure 3.10: The same experiment as in 3.8 is repeated with $T = 50$.

are assembled into a `DecisionForest`.

The `Categorizer` is learned next. Each training `Bitmap` is converted to a `TreeHistogram` using the previously learned `DecisionForest`. Using `PyramidMatch.Similarity` as a `SimilarityMetric` among the `TreeHistograms`, a `SimilarityMatrix` (i.e. kernel matrix) is computed. Finally, a `CategoryModel` (i.e. classifier) is trained for each label class.

3.4 Implementation Improvements

The reference implementation is a stable, easily extensible, and designed for running research experiments. As is, however, it is not suitable for use within Diamond. I improved the reference implementation's performance and cross-platform compatibility. Like the original software, these improvements are freely available under the GPL.

3.4.1 Parallel/distributed computation

The reference implementation lacks some basic optimizations [1]. Except for basic numerical operations, the code doesn't exploit parallelism because it is single-threaded. This is especially unfortunate because many portions of the code are embarassingly parallel. For example:

- Extractions from the training data (e.g. subsampling) can be done in parallel
- The forest can be learned in parallel
- Recursively learning each decision trees is exponentially parallel - each subtree can be grown independently
- Each of the SVMs can be trained in parallel

Each of these opportunities can be consummated with simple fork-join parallelism. Though C# did not provide high-level facilities for doing so when the reference implementation was written, it now provides a healthy suite of concurrent data structures and task-oriented mechanisms for concurrency and parallelism.

Parallelizing the reference implementation wasn't entirely straightforward, however. The code harbored a fair amount of mutable shared state which is accessed frequently while learning decision trees. This state was systematically teased out by keeping the mutable state local to each tree and passing it to the methods that mutate it.

[1]For their award-winning demo at CVPR, the authors commissioned Microsoft Research to develop a high-performance, real-time implementation of parts of STF; it has not been released by Microsoft Research.

3.4.2 Port to Mono/Linux

C# and the virtual machine upon which it runs, the CLI, have two popular implementations. One is the official .NET implementation for Microsoft Windows. The other is Mono, which is open-source and cross-platform. Since OpenDiamond is implemented only for Linux, STF must run on Mono. The reference implementation, however, depends upon Windows-specific functionality. This dependency, upon investigation, is superficial; all the Windows-specific functionality can be replaced by cross-platform analogues. By replacing this code, STF correctly builds with and runs on Mono 2.6, the stable version at the time of writing.

3.4.3 Integration within Diamond

The reference implementation is set up only to perform experiments. In Diamond, STF is invoked in two different contexts. During training, STF is provided a description of the learning problem and returns serializations of the forest and classifier. During search, as part of the Diamond `adiskd` server, STF unserializes these data structures and then classifies a stream of images.

This functionality is implemented in the simplest way possible. The web application invokes a program which reads the learning problem description over standard input and writes the data structures to standard output. `adiskd` invokes a program which reads the data structures over standard input, then subsequently reads PPM-formatted images from standard input and writes their classifications to standard output.

Chapter 4

Algum: the user interface

Algum is a web application that produces STF classifiers for use in OpenDiamond. Its primary goal is to provide a seamless and pleasing training experience which realizes the Diamond vision of iterated, interactive hypothesis exploration. The user experience is enhanced by a variety of advanced new web technologies. The interactive learning process is supported by mechanisms for improving classifier quality. Algum is an acronym for Active Learning Graphical User interface for Markup. The term also refers to a type of wood used to construct holy objects in the Bible. Our hopes for Algum's output are more modest, but nonetheless high.

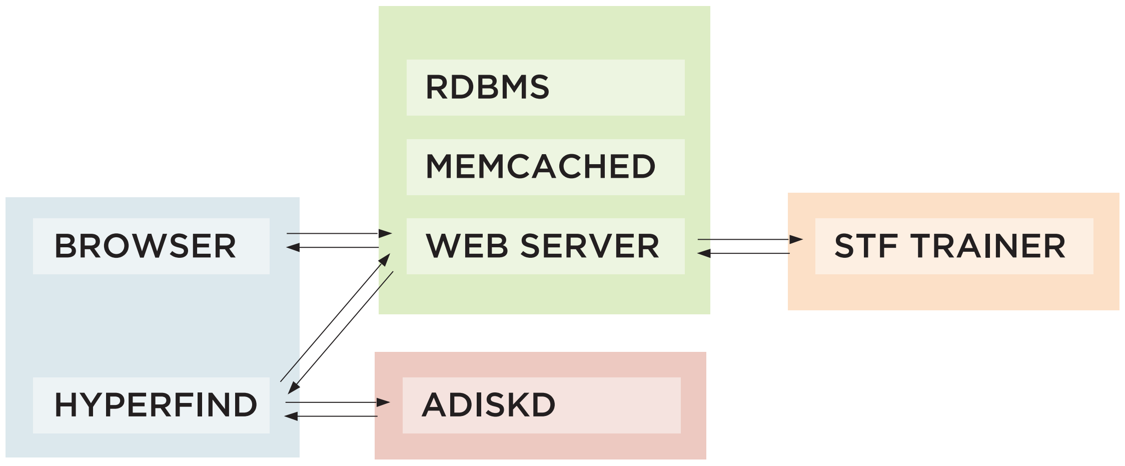
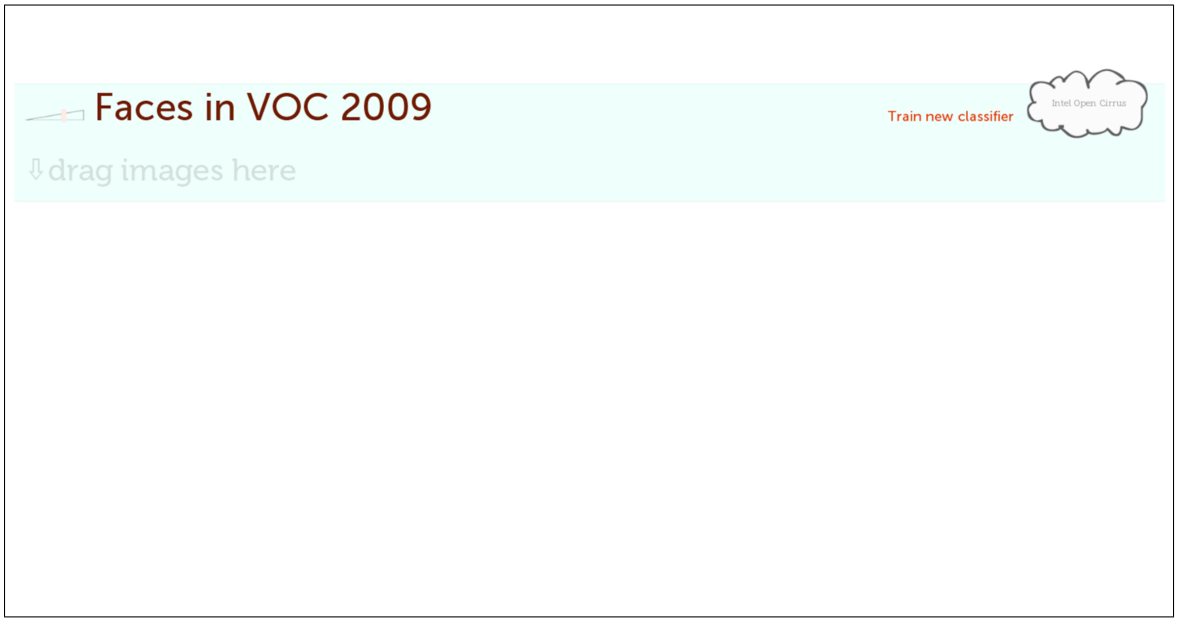
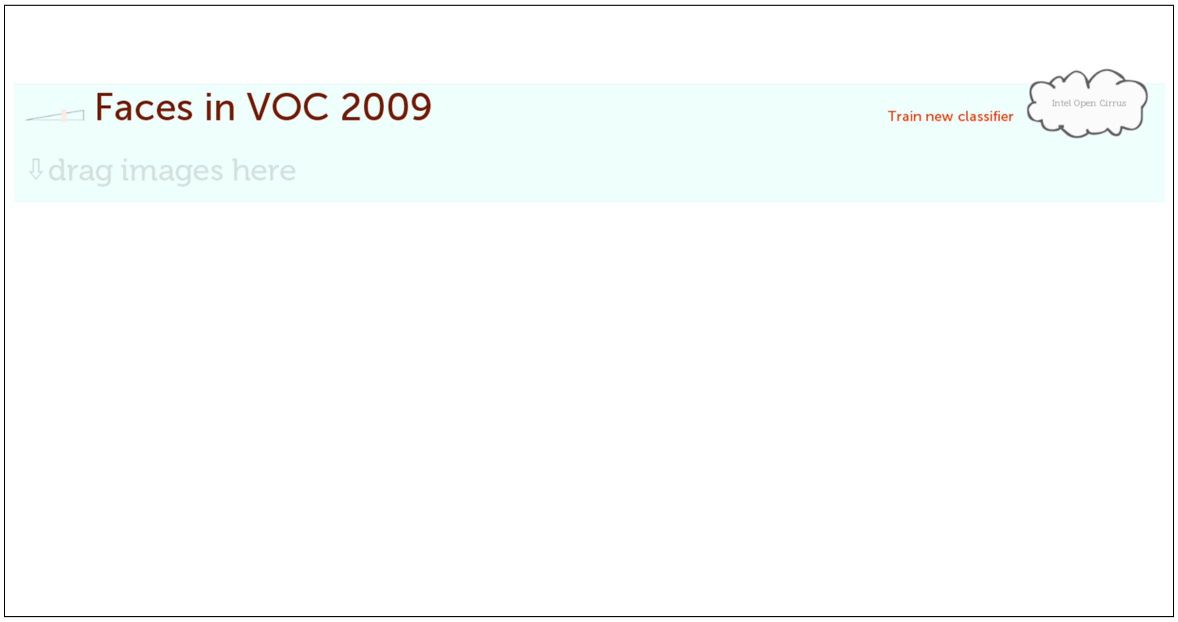


Figure 4.1: High-level software architecture of Algum elaborated by this chapter.

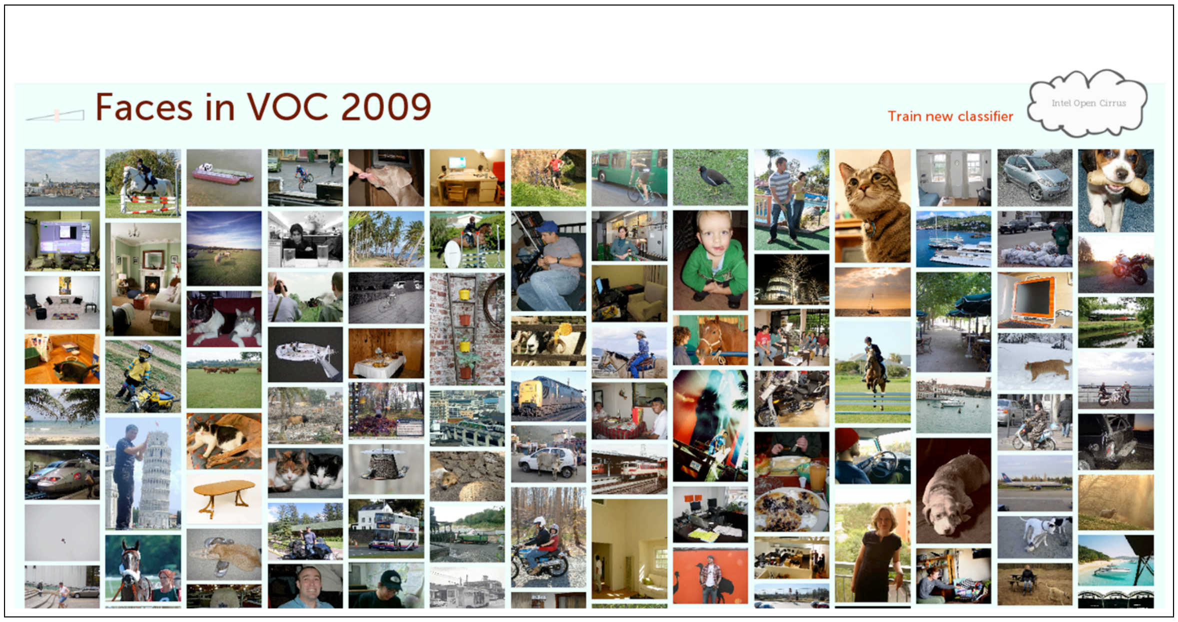
This chapter gives a visual walkthrough of Algum's main features. It describes the application's data model and key architectural properties. It gives an overview of the implementation using JRuby, Ruby on Rails, MySQL, and memcached.



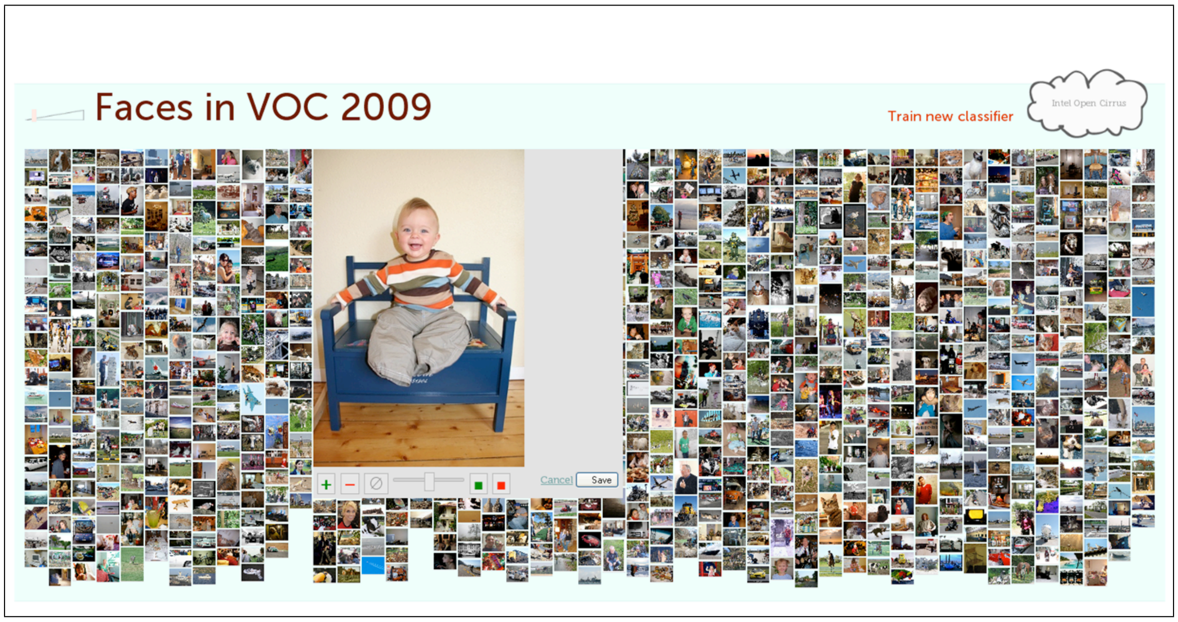
Here Algum will be used to produce a searchlet for human faces in a popular computer vision dataset. This is called the classification task. The user will populate the task with unlabeled images, label a few of those images, and begin training. The entire session is authenticated and encrypted.



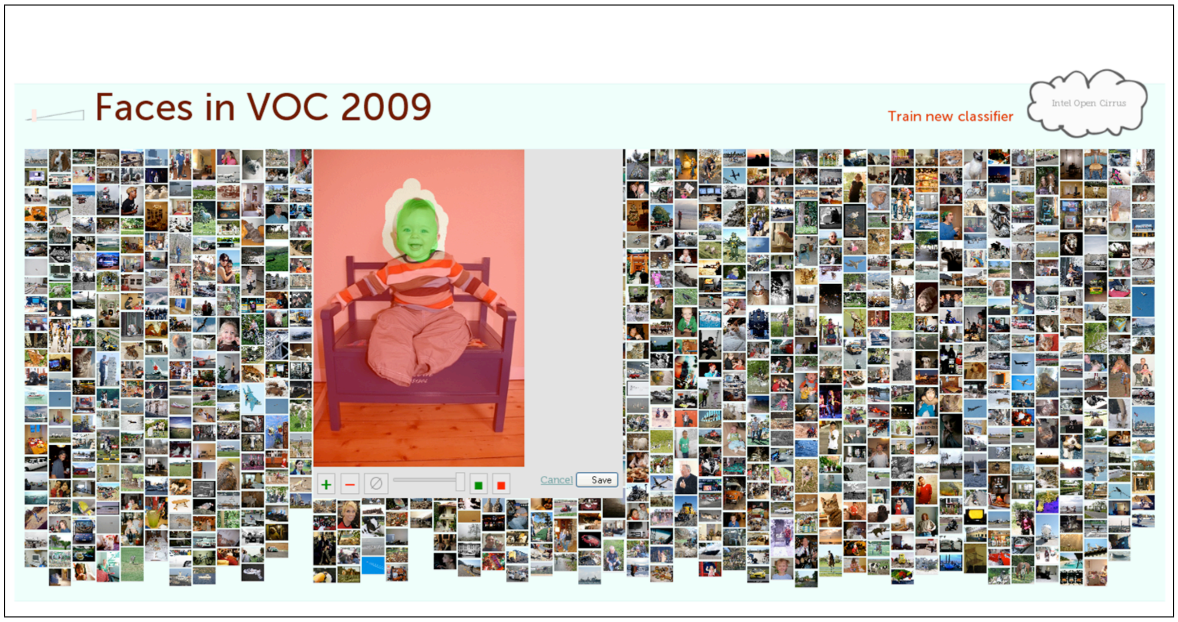
To add unlabeled images to the task, the user can simply drag and drop images from another application. For example, images in Diamond application (e.g. Hyperfind) or the filesystem. Progress bars are displayed while the images upload through asynchronous `XmlHttpRequests` ('AJAX') requests. Once complete, the images render in place without a page refresh.



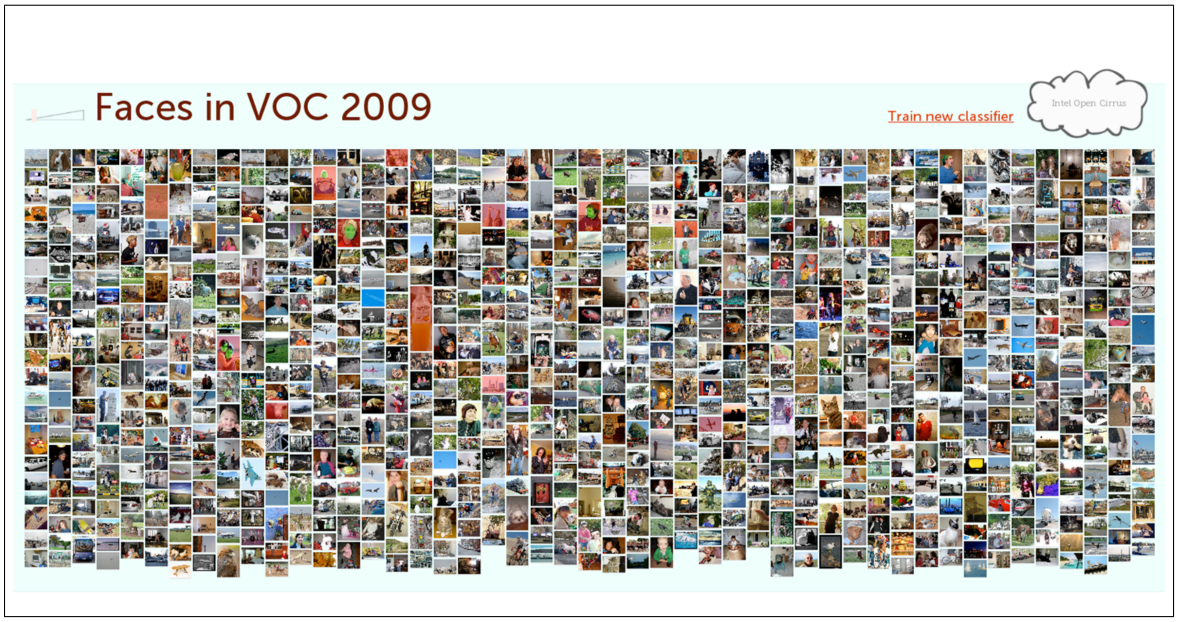
Algum is designed to handle many unlabeled images, as may be necessary for active learning. Viewing and managing lots of images is facilitated by multicolumn layout. Using the slider in the top left, the thumbnails are magnified while the tight layout is preserved.



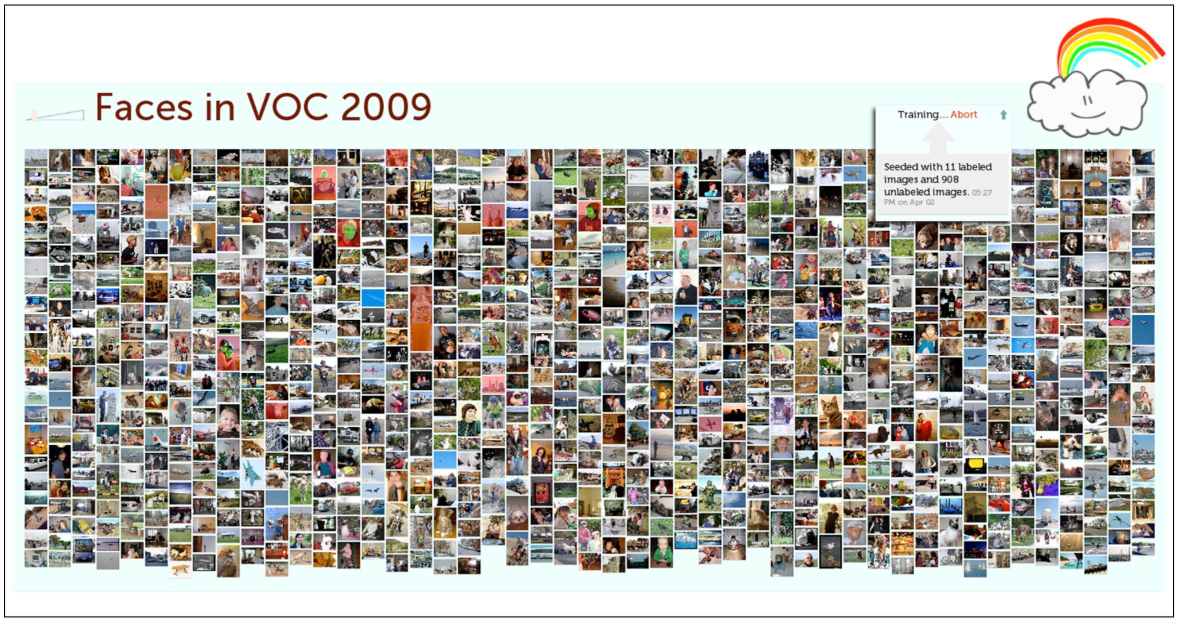
The user clicks on an image to display a label editor, which implements the same basic functionality as LANL's Aladdin. Of course, unlike Aladdin, its 'paint' interface is implemented entirely in Javascript, which eliminates fumbling with temporary files. The Algum labeling tools are also more complete.



Since Algum trains binary searchlets, only positive and negative input is required. Not all pixels need to be labeled. The label editor produces small PNGs which are fast to upload.

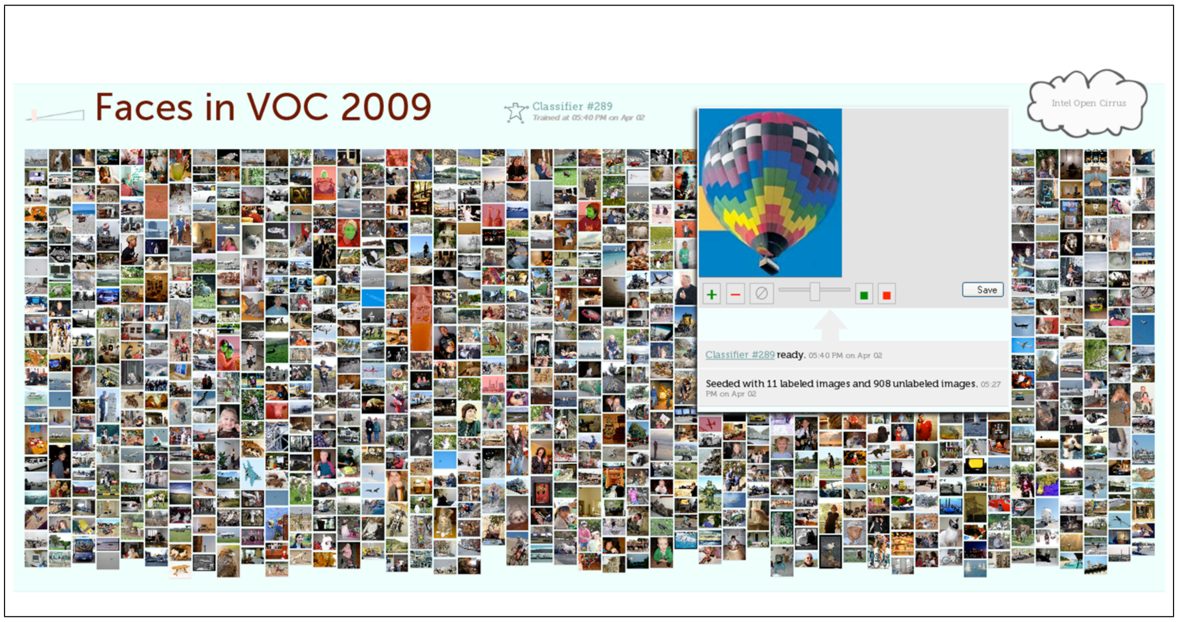


Though not strictly necessary, the user is typically expected to label a few examples as a 'seed' to training. Once this initial labeling is complete, the training process can begin. For obvious performance reasons, training occurs on a different machine than the webserver. The backend compute cloud is visualized in Algum's user interface. Before training starts, different clouds can be selected. Upon pressing 'train new classifier', the webserver forks a thread for communicating with the compute cloud. Expiring, self-certifying URLs (see) for the images and labels are sent to the compute cloud, which subsequently starts training.



Seeded with 11 labeled images and 908 unlabeled images. 05:27 PM on Apr 02

As the learning algorithm happily crunches away, the user's browser intermittently polls the webserver for a complete classifier. An arbitrary number of tasks can train simultaneously. The training processes are persisted even if the user's session is unexpectedly terminated. The user can unilaterally abort the training processes.

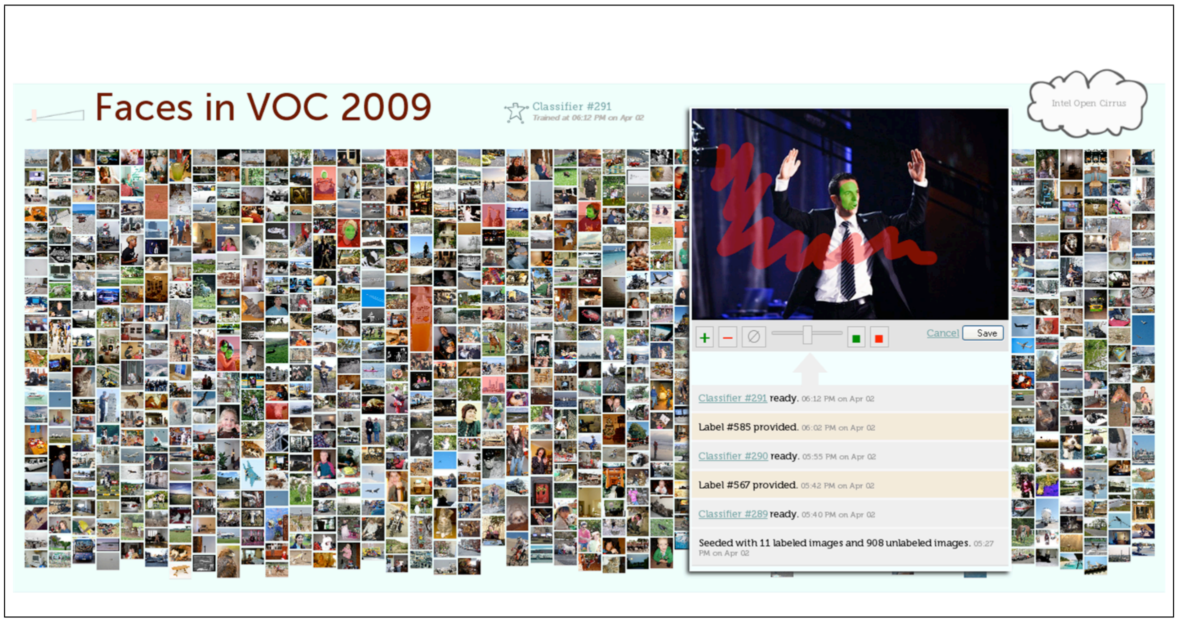


Classifier #289 ready. 05:40 PM on Apr 02

Seeded with 11 labeled images and 908 unlabeled images. 05:27 PM on Apr 02

Eventually, the compute cloud returns serialized classifier data structures which are assembled by the webapp into a ZIP file. Through a drag-and-drop interface, an expiring, self-certifying URL for the ZIP file can be provided to Hyperfind, a Java user interface for performing Diamond searches. When a search is begun, Hyperfind downloads the ZIP file and ships it to `adiskd` in the usual fashion.

The initial searchlet produced by the seed labels is likely to be inadequate. As mentioned previously, Algum supports two ways of improving the searchlet when unsatisfactory results are received. The first way is to provide discretionary relevance feedback: the user can drag egregious false positives from Hyperfind into Algum, label them as negative, and generate another classifier. The second way is to engage in algorithmically-driven active learning, which is described in more detail in chapter 5. Here the user is prompted to label an image of the algorithm's choosing. Though this choice is currently random, an active learning algorithm could be easily connected to the existing infrastructure.



Classifier #291 ready. 06:12 PM on Apr 02

Label #585 provided. 06:02 PM on Apr 02

Classifier #290 ready. 05:55 PM on Apr 02

Label #567 provided. 05:42 PM on Apr 02

Classifier #289 ready. 05:40 PM on Apr 02

Seeded with 11 labeled images and 908 unlabeled images. 05:27 PM on Apr 02

All classifier history is kept in the database for subsequent analysis.

4.1 Architecture

Algum utilizes a fairly typical model-view-controller architecture.

4.1.1 Data Model

Algum employs a simple data model which captures all the salient interactions described in the walkthrough.

A `User` has many classification `Tasks` (e.g. faces, shiny metal, cows). A `Task` consists of `Images`, which may have `Labels` (i.e. PNG overlays). `Images` and `Labels` are immutable, but `Labels` can be refined through versioning. Each training session is captured in a `ClassifierSequence`, which is seeded with labeled and unlabeled `Images`. A `Classifier` is followed by a `LabelRequest`, which is either a `LabelCreationRequest` associated with an unlabeled `Image`, or a `LabelElaborationRequest` associated with a (presumably uninformative) `Label`. The `LabelRequest` is fulfilled by a `Label`, which is subsequently used to generate a new `Classifier`.

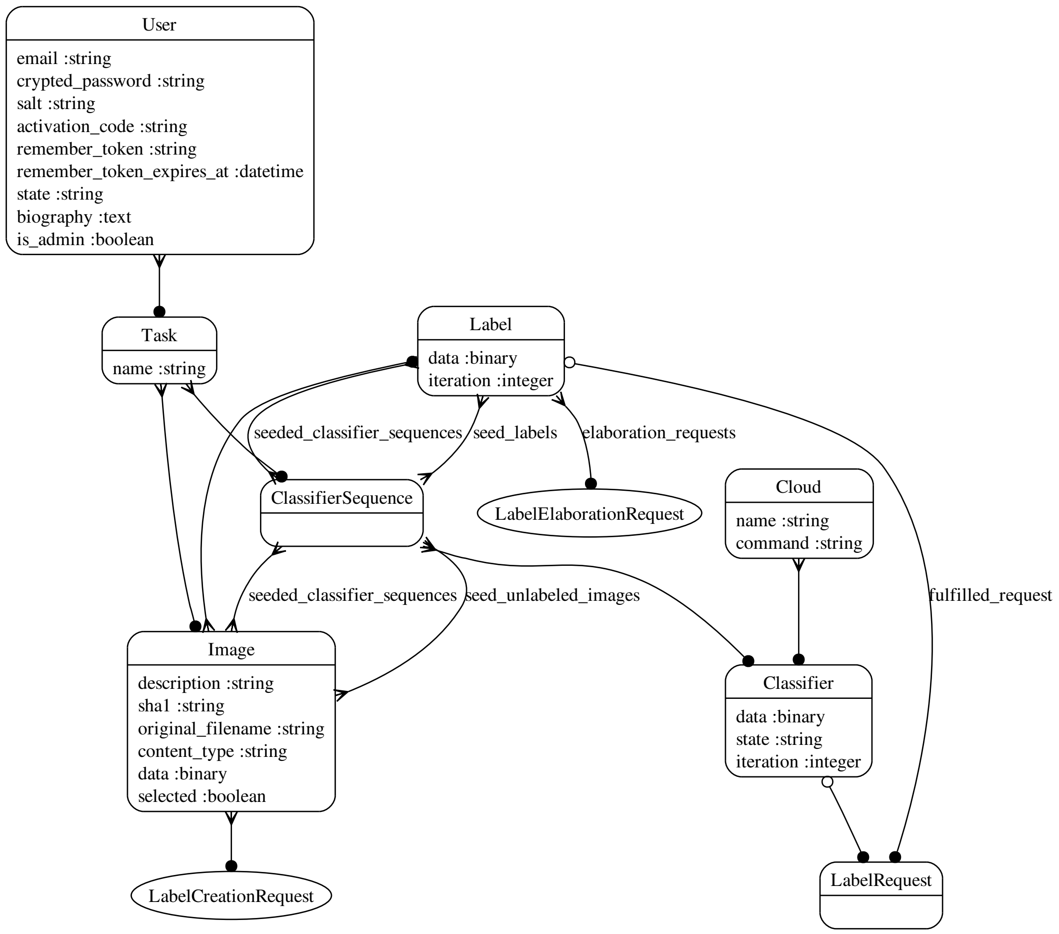


Figure 4.2: Models and their relationships.

4.1.2 REST

HTTP underlies the most successful distributed hypermedia system: the World Wide Web. The aspects of HTTP that made it so successful are isolated by REST (REpresentation State Transfer), a client-server software architecture developed in parallel with HTTP [23]. Though REST prescribes a number of constraints upon client-server interaction, its practical implications for web application development can be boiled down to a few maxims. HTTP should be treated as an application protocol, not an RPC layer. For example, URLs should have meaning to users, and inbuilt HTTP mechanisms for authentication, content negotation, and caching should be preferred over external schemes. The application should provide representations (e.g. different serializations) of resources (e.g. Tasks, Images, etc.). Upon these resources, the semantics of HTTP methods (e.g. GET, POST, PUT, and DELETE) should be respected. Finally, client-server interaction should be state-free whenever possible.

These principles guide the architecture of Algum. The URI design, summarized in figure 4.3 is especially RESTful. The URIs are all uniform and function differently based on the invoking HTTP method. Most of the URIs map the HTTP verbs to create, read, update, and delete operations upon the models described in section FIXME. Some of the URIs, such as those pertaining to the user session, do not correspond to models, but nonetheless deal with resources.

Algum uses basic HTTP authentication and relies on encryption to provide security.

4.1.3 HMAC-SHA1 for URI Capabilities

Data served by Algum is read by the user's browser, the compute cloud, and Hyperfind. Specifically, images and labels are read by both the browser and cloud, and classifiers are read by both the browser and Hyperfind. This poses a usability or security hazard since all data on Algum is protected by authentication. It would be very inconvenient for the user to authenticate multiple times. The URIs given to Hyperfind and the compute cloud should be self-certifying: the clients should not have to provide additional authentication information. However, the URIs should not grant permanent access to the resources. Finally, additional state within Algum should be avoided.

This problem is solved elegantly by endowing URLs with capabilities using cryptographic techniques. HMAC (Hash-based Message Authentication Codes) is a commonly used technology for message authentication [39]. To the best of my knowledge, Amazon's Simple Storage Service first demonstrated the use of HMAC for URI capabilities. Suppose we would like to provide metered access to some resource:

```
http://algum/images/3.png?size=large
```

First, endow the URI with capabilities, such as an expiration date:

URI name	HTTP method	URI pattern	URI handler
labels	GET	/labels(.:format)	controller: labels, action: index
	POST	/labels(.:format)	controller: labels, action: create
new_label	GET	/labels/new(.:format)	controller: labels, action: new
edit_label	GET	/labels/:id/edit(.:format)	controller: labels, action: edit
label	GET	/labels/:id(.:format)	controller: labels, action: show
	PUT	/labels/:id(.:format)	controller: labels, action: update
	DELETE	/labels/:id(.:format)	controller: labels, action: destroy
images	GET	/images(.:format)	controller: images, action: index
	POST	/images(.:format)	controller: images, action: create
new_image	GET	/images/new(.:format)	controller: images, action: new
edit_image	GET	/images/:id/edit(.:format)	controller: images, action: edit
image	GET	/images/:id(.:format)	controller: images, action: show
	PUT	/images/:id(.:format)	controller: images, action: update
	DELETE	/images/:id(.:format)	controller: images, action: destroy
task_images	GET	/tasks/:task_id/images(.:format)	controller: images, action: index
	POST	/tasks/:task_id/images(.:format)	controller: images, action: create
new_task_image	GET	/tasks/:task_id/images/new(.:format)	controller: images, action: new
edit_task_image	GET	/tasks/:task_id/images/:id/edit(.:format)	controller: images, action: edit
task_image	GET	/tasks/:task_id/images/:id(.:format)	controller: images, action: show
	PUT	/tasks/:task_id/images/:id(.:format)	controller: images, action: update
	DELETE	/tasks/:task_id/images/:id(.:format)	controller: images, action: destroy
tasks	GET	/tasks(.:format)	controller: tasks, action: index
	POST	/tasks(.:format)	controller: tasks, action: create
new_task	GET	/tasks/new(.:format)	controller: tasks, action: new
edit_task	GET	/tasks/:id/edit(.:format)	controller: tasks, action: edit
task	GET	/tasks/:id(.:format)	controller: tasks, action: show
	PUT	/tasks/:id(.:format)	controller: tasks, action: update
	DELETE	/tasks/:id(.:format)	controller: tasks, action: destroy
classifiers	GET	/classifiers(.:format)	controller: classifiers, action: index
	POST	/classifiers(.:format)	controller: classifiers, action: create
new_classifier	GET	/classifiers/new(.:format)	controller: classifiers, action: new
edit_classifier	GET	/classifiers/:id/edit(.:format)	controller: classifiers, action: edit
classifier	GET	/classifiers/:id(.:format)	controller: classifiers, action: show
	PUT	/classifiers/:id(.:format)	controller: classifiers, action: update
	DELETE	/classifiers/:id(.:format)	controller: classifiers, action: destroy
label_requests	GET	/label_requests(.:format)	controller: label_requests, action: index
	POST	/label_requests(.:format)	controller: label_requests, action: create
new_label_request	GET	/label_requests/new(.:format)	controller: label_requests, action: new
edit_label_request	GET	/label_requests/:id/edit(.:format)	controller: label_requests, action: edit
label_request	GET	/label_requests/:id(.:format)	controller: label_requests, action: show
	PUT	/label_requests/:id(.:format)	controller: label_requests, action: update
	DELETE	/label_requests/:id(.:format)	controller: label_requests, action: destroy
passwords	GET	/passwords(.:format)	controller: passwords, action: index
	POST	/passwords(.:format)	controller: passwords, action: create
new_password	GET	/passwords/new(.:format)	controller: passwords, action: new
edit_password	GET	/passwords/:id/edit(.:format)	controller: passwords, action: edit
password	GET	/passwords/:id(.:format)	controller: passwords, action: show
	PUT	/passwords/:id(.:format)	controller: passwords, action: update
	DELETE	/passwords/:id(.:format)	controller: passwords, action: destroy
new_user_password	GET	/users/:user_id/password/new(.:format)	controller: passwords, action: new
edit_user_password	GET	/users/:user_id/password/edit(.:format)	controller: passwords, action: edit
user_password	GET	/users/:user_id/password(.:format)	controller: passwords, action: show
	PUT	/users/:user_id/password(.:format)	controller: passwords, action: update
	DELETE	/users/:user_id/password(.:format)	controller: passwords, action: destroy
	POST	/users/:user_id/password(.:format)	controller: passwords, action: create
users	GET	/users(.:format)	controller: users, action: index
	POST	/users(.:format)	controller: users, action: create
new_user	GET	/users/new(.:format)	controller: users, action: new
edit_user	GET	/users/:id/edit(.:format)	controller: users, action: edit
user	GET	/users/:id(.:format)	controller: users, action: show
	PUT	/users/:id(.:format)	controller: users, action: update
	DELETE	/users/:id(.:format)	controller: users, action: destroy
	*	/activate/:activation_code	controller: users, action: activate
sessions	GET	/sessions(.:format)	controller: sessions, action: index
	POST	/sessions(.:format)	controller: sessions, action: create
new_session	GET	/sessions/new(.:format)	controller: sessions, action: new
edit_session	GET	/sessions/:id/edit(.:format)	controller: sessions, action: edit
session	GET	/sessions/:id(.:format)	controller: sessions, action: show
	PUT	/sessions/:id(.:format)	controller: sessions, action: update
	DELETE	/sessions/:id(.:format)	controller: sessions, action: destroy
login	*	/login	controller: sessions, action: new
logout	*	/logout	controller: sessions, action: destroy
root	*	/	controller: sessions, action: new
	*	/javascripts/label-editor.js	controller: labels, action: label_editor, format: js

Figure 4.3: URL design of Algum.

```
   http://algum/images/3.png?size=large&expires=2010-01-01
```

Then, canonicalize the URI in some fashion. For example, impose an alphabetic ordering on the query parameters.

```
        algum/images/3.png?expires=2010-01-01&size=large
```

Compute HMAC for the canonicalized URI and append it to the capability-endowed URI to produce the URI sent to the client:

```
http://algum/images/3.png?size=large&expires=2010-01-01&mac=2b3J9aZ01jIF5
```

When a client requests a URI, strip off the code, canonicalize, compute HMAC, and compare the result to the provided code. Grant access iff they match.

This technique possesses all of the required properties:

- self-certifying: the client is given a single string that doesn't need to be accompanied by login info
- self-describing: capabilities are stored as near-plaintext in the URI.
- efficient: authentication takes constant time wrt to the size of the resource. Fast HMAC implementations are widely available.
- nearly stateless: aside from the HMAC secret key, the server doesn't have to create or maintain any state to perform authentication.
- easy to implement: transparent on clients, trivial on servers.

A disadvantage of this technique should be noted as well: in HTTP, appending the capabilities and mac to the URI prevent the client from caching the underlying resource

```
                http://algum/images/3.png?size=large
```

between different expiration-dated URIs

```
http://algum/images/3.png?size=large&expires=2009-12-31&mac=2b3J9a
http://algum/images/3.png?size=large&expires=2010-01-01&mac=f3Ba8z
```

This can be fixed by having the client strip the capabilities and signature from the URI and provide them separately as headers (X-Capabilities and X-Signature.)

Name	Lines	LOC	Classes	Methods
Controllers	545	391	11	48
Helpers	137	71	0	10
Models	518	410	16	39
Libraries	210	102	0	22
HTML Templates	525	474		
Javascript	416	364		47
Total	2351	1812	27	166

Figure 4.4: The Algum codebase is relatively small.

4.2 Implementation

Architecturally, Algum is a fairly typical web application. Since there aren't any special constraints upon implementation, the main engineering priorities are correctness, conciseness, and performance.

4.2.1 JRuby on Rails

Ruby on Rails web application framework geared towards brevity. Considering the relatively small size of the Algum codebase, summarized in figure 4.4, Rails seems to deliver on its original motto of "less code than most frameworks spend doing XML situps."

JRuby is an implementation of the Ruby scripting language for the Java virtual machine. This implementation is used for three primary reasons. Many components of OpenDiamond are already written in Java. JRuby integrates Ruby with Java libraries which tend to be superior to their Ruby counterparts. Portions of Algum, particularly those related to image processing, directly utilize Java libraries. Finally, JRuby threads are scheduled by the operating system, whereas threads in the standard Ruby implementation are scheduled in userspace. In particular, when one JRuby thread performs I/O, the others do not block. This behavior is critical for long-running training threads in the web application server.

4.2.2 MySQL and memcached

MySQL is a standard open-source SQL RDBMS. Algum stores all of its data, including images and their labels, in MySQL. This approach simplifies transaction processing and database administration (e.g. backups). Another approach is to keep images in the filesystem and store references in the database. Though the latter method is higher performance – the operating system could optimize I/O via `splice()` or `sendfile()` – it is more failure prone and cumbersome to manage.

To maintain adequate performance, Algum employs caching on the server. Images and labels are easily cached since they are immutable. Algum also caches fragments of HTML which are expensive to generate. All of these objects are cached using a straightforward mechanism based on `memcached`, a popular key-value cache daemon which runs on a separate process or machine. Images and labels are never expired. HTML fragments are immediately expired when their constituent models are updated.

4.2.3 Client-side and interface

Algum also employs client-side HTTP caching. When immutable resources such as images and labels are served, they are accompanied with uniquely identifying `ETag`s and long-dated expiration times. The client caches these resources and revalidates them with 'conditional GET' requests.

In order to provide for a seamless and pleasing user experience, Algum utilizes a variety of new browser features, most of which are part of the forthcoming HTML 5 specification.

- Multicolumn image display: HTML tables and CSS 2 waste space when the images have different dimensions. The CSS 3 Multicolumn module enables uniform, space-saving layout.
- File API uploading: traditional HTML file uploads support only a single file at a time. The HTML 5 File API enables drag-and-drop uploading of multiple images, which was previously possible only with special browser plugins.
- Canvas labels: binary formats like MAT are small but cannot be manipulated by web browsers. Text-based formats like JSON are ineffiicent. The HTML 5 Canvas produces PNGs, which are small and consumed directly by STF.
- Web fonts: Algum prefers clean code to complicated layouts. To achieve a distinctive visual appearance, Algum utilizes boutique typefaces.

Chapter 5

Active learning and ALMA

STF, described in the chapter 3, is an algorithm for learning image representations and image classifiers. As input, STF takes a set of weakly labeled images, and outputs a decision forest and classifier. Since it is merely fed data, STF is called a passive learning algorithm. Such algorithms are designed for learning upon a fixed dataset; as mentioned in the introduction of the thesis, Diamond is designed for a different scenario. In Diamond, the user is capable of interacting with a learning algorithm, but his time is precious.

Active learning is a model that captures many essential properties of learning within Diamond. Unlike passive learning algorithms, which initially receive labeled datasets, active learning algorithms initially receive *unlabeled* datasets. The purpose of the algorithm is to choose the next unlabeled example for the user to label, with the goal of minimizing the number of labels (i.e. total user effort) required to produce a classifier (or searchlet) of high quality. In this sense, active learning algorithms are superior to passive learning algorithms in a variety of settings [37, 7]. Consider the canonical task of learning a threshold on the real line.

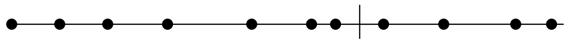


Figure 5.1: Examples left of the threshold are negative; the rest are positive.

To achieve an error rate of ϵ, a passive learner requires $\Omega(1/\epsilon)$ labels [1]. An active learner using binary search requires only $O(\log 1/\epsilon)$ labels. This task is an example of *pool-based* active learning, where algorithms sample labels for a finite set of independently and identically distributed (iid) unlabeled data. In *stream-based* active learning, algorithms selectively sample [12], potentially in a strictly online fashion [47], labels from an infinite stream of iid unlabeled data. In this chapter, we focus on pool-based active learning because it may not be appropriate or desirable to model the unlabeled data as an infinite stream. Furthermore, in the Diamond setting, it is worth computing over the entire pool if it saves user attention.

To obtain a general-purpose algorithm suitable for use in computer vision, we augment a pool-based active learning procedure with a theoretical bound on label complexity. Kernels, misspecified models, and multicategory classification are supported through convex, classification-calibration surrogate loss that maintains asymptotic consistency. The computationally-intractable querying criterion is approximated by a Markov Chain Monte Carlo procedure with strong bounds on statistical error and computational complexity. In relation to existing algorithms, our algorithm, Active Learning by Measure Approximation, is effective upon standard computer vision image classification benchmarks.

5.1 Algorithm setting and requirements

This chapter considers a slightly broader set of priorities.

- Computational tractability: this is essential to any empirical, applied science, especially one involving large datasets. Computer vision algorithms are often expected to operate interactively or in real-time.
- Multicategory classification: most classification tasks, datasets, and competitions in computer vision involve multiple categories. Large numbers of categories, as in the Caltech 256 dataset, are not unusual.
- Learning from kernels and hypothesis priors: learning algorithms for standard tasks should operate upon standard inputs. Conceptually, requiring extra domain knowledge offloads a portion of the learning problem. Practically, unusual inputs are burdensome for users to provide to algorithms. Methodologically, the modularity afforded by a standard interface sustains fast-paced, independent research in both classification and image understanding. In image classification, this standard interface consists of kernels on data and (Bayesian) priors on hypotheses.
- Pool-based operation: though computer vision datasets can be large, unlabeled data are not free. They may be provided by the same agent whose labeling effort is economized. In fact, unlabeled data are often scarce relative to the the computational resources required to operate upon them. It is not practical to run stream algorithms on pools that are quickly exhausted.

The last of these requirements has been the hardest to fulfill in conjunction with strong quality guarantees. Most theoretical results apply only to the stream model, where the infinite stream admits direct application of iid learning theory and calculation of otherwise unobtainable statistics — perhaps most famously, arbitrarily precise confidence bounds in the A^2 algorithm [2]. In this setting a wide array of results have been proved: robustness against misspecified models (the 'noisy' setting) [16], adaptation to benign noise conditions [28] minimax bounds [11], and strict improvements in sample complexity over passive learning [4]. These results give rise to practical and theoretically supported algorithms such as importance weighted active learning [7], perceptron active learning [17], and query by committee [24].

It is relatively difficult to work with the pool model. Each label request introduces a dependence among the remaining elements in the pool and thereby prevents the the application of usual iid learning theory. Furthermore, high lower bounds exist even in benign conditions. Assuming a correctly specified model (the ‘separable’ or ‘realizable’ setting), active learning cannot achieve an improvement over passive learning in worst case label complexity for learning non-homogenous linear separators in two dimensions [13]. As a result, relatively few theoretical results apply to the pool model. Analysis in terms of the splitting index [14] or neighborly condition [49] assumes a correctly specified model. Some nominally pool-based results require infinite pools; these include margin-based active learning [3], Dasgupta’s splitting algorithm [14], and the sufficient conditions for agnostic learnability [64].

Pool algorithms are common in computer vision [34, 36] and machine learning [30] literature. Many of these algorithms achieve the principal objectives listed above while excelling in at least one. For example, some active learning algorithms are very computationally efficient with respect to the amount of data [36] and categories [34]. Though these algorithms usually achieve experimental improvements, the vast majority do not have strong quality guarantees.

The only two pool algorithms with quality guarantees require domain knowledge beyond the training data. Hierarchical sampling for active learning [15] performs a hierarchical clustering of the data and queries nodes that shrink confidence sets associated with each cluster. This method has a label complexity guarantee relative to the quality of the clustering. However, clustering quality is domain-dependent and it is unrealistic to ask for an appropriate clustering algorithm along with the training data.

The greedy algorithm of [13] instead requires a prior, and has a label complexity guarantee relative to the optimal querying strategy. However, the algorithm assumes a correctly specified model, is computationally intractable, only performs binary classification, and does not support training data provided as a kernel matrix.

In this chapter, we address each of the shortcomings of [13] through approximations that retain the original quality guarantees. Misspecified models, multicategory classification, and kernels are supported through a convex ‘surrogate’ relaxation of the loss function which carefully preserves consistency with multiple categories. An intractable integral is approximated to arbitrary precision by an efficient sampling algorithm with strong, well-developed bounds on the number of required samples. The proposed algorithm, Active Learning by Measure Approximation (ALMA) is the first to satisfy all of the aforementioned properties while being supported by a quality guarantee.

5.2 Related approaches

A myriad of greedy querying criteria have been proposed for pool-based active learning. These include uncertainty sampling, expected model change, variance reduction, estimated error reduction, and density-weighted methods [54]. To the best of our knowledge, our criterion is the only one with a theoretical guarantee on label complexity relative to the optimal non-greedy criterion. Similarly, other active learning algorithms utilize convex surrogate loss for multicategory classification. However, none of these methods preserve the consistency of the degenerate binary case.

The Kernel Query By Commitee algorithm [24] also involves kernels and samples efficiently from a high-dimensional function space. That algorithm uses kernels only for sampling, whereas we use kernels to define the function space. Though their construction works for any function space, it is limited to application in the Query By Committee algorithm. Our method works for any algorithm which chooses among functions in a reproducing kernel Hilbert space defined by a kernel matrix.

Like the algorithm upon which it is based, ALMA is one of the many active learning algorithms designed to shrink a function space. This general approach is not new, having been employed by the 1992 paper [12] that coined the term 'active learning'. This approach has a natural geometric interpretation which has been explored in significant detail [60].

5.3 Problem formalization

The data-generating mechanism is a distribution $\mathbb{P}$ on the product space $\mathcal{X} \times \mathcal{Y}$, where the pattern space $\mathcal{X}$ is arbitrary and the label space $\mathcal{Y} = \{1, \ldots, |\mathcal{Y}|\}$. Drawn independently from $\mathbb{P}$ are the data $\{(X_1, Y_1), \ldots, (X_n, Y_n)\} \subseteq \mathcal{X} \times \mathcal{Y}$. The X_i are all accessible at once (this is the 'pool'), but the Y_i must be sequentially queried in the order specified by our sampling strategy. Let $S_T = \{(X_{\pi_1}, Y_{\pi_1}), \ldots, (X_{\pi_T}, Y_{\pi_T})\}$ be the labeled sample at time $0 \leq T \leq n$. Using this sample, our objective is to make a hypothesis $\hat{h}$ with minimal risk $R(\hat{h}) = \mathbb{P}(\hat{h}(X) \neq Y)$. Internal randomization is accommodated by a confidence parameter: with probability $1 - \delta$, $\hat{h}$ should satisfy $R(\hat{h}) < R^* + \epsilon$ where $R^* = \inf_h R(h)$ is the noise rate over a hypothesis space.

The hypothesis space is defined by a kernel function $k(x_1, x_2) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ which induces a reproducing kernel Hilbert space

$$f = \left\{ \sum_{i=1}^{n} \alpha_i k(X_i, \cdot) : \alpha_i \in \mathbb{R} \right\}$$

as an inner product $k(x_1, x_2) = \langle \phi(x_1), \phi(x_2) \rangle_k$ where $\phi : \mathcal{X} \to f$ is a potentially intractable feature mapping. (See chapter 3 for a motivation.) The induced norm is $||f||_k = \sqrt{\langle f, f \rangle_k}$. The kernel is provided to the algorithm as a finite, positive semi-definite Gram matrix $\mathbf{k} = \{k_{i,j} = k(x_i, x_j) : 1 \leq i, j \leq n\}$. Multiple categories are accommodated by the $|\mathcal{Y}|$-power product Hilbert space $\mathcal{F} = f^{|\mathcal{Y}|} =$

$$\left\{ \sum_{i=1}^{n} \{\boldsymbol{\alpha}_{i,1} k(x_i, \cdot), \ldots, \boldsymbol{\alpha}_{i,|\mathcal{Y}|} k(x_i, \cdot)\} : \boldsymbol{\alpha}_{i,\cdot} \in \mathbb{R}^{|\mathcal{Y}|} \right\}.$$

Let $\kappa : \mathbb{R}^{n \times |\mathcal{Y}|} \to \mathcal{F}$ denote this mapping from coefficients $\boldsymbol{\alpha}$ to functions $f \in \mathcal{F}$, and let $f_1, \ldots, f_{|\mathcal{Y}|}$ denote the components of f.

Further domain knowledge is provided by a probability measure V on the Borel measurable space $(\mathcal{F}, \mathcal{B}(\mathcal{F}))$. V is typically the product measure formed from a probability measure on $(f, \mathcal{B}(f))$. V can be interpreted as a Bayesian prior, but we do not perform Bayesian inference; it is a device to facilitate query selection through the average-case analysis of [13]. V can be chosen to be uniform or non-informative. Unlike [13], V does not necessarily assign positive measure to a set containing a consistent function. Reasonable technical conditions on V are discussed in §5.4.2.

5.4 Active learning by measure approximation

Both ALMA and the original algorithm [13] iteratively shrink a set of functions specified by $\mathcal{C} : 2^{\mathcal{X}\times\mathcal{Y}} \to 2^{\mathcal{F}}$ which, given a sample, returns the set of remaining functions. In the original algorithm, which assumes a correctly specified model, $\mathcal{C}$ is the version space: the set of functions consistent with the labeled sample. The original algorithm requests

$$\mathrm{argmax}_x \mathrm{min}_{y\in\mathcal{Y}} V(\mathcal{C}(S_T + \{x, y\})) \tag{5.1}$$

which, for binary classification ($|\mathcal{Y}| = 2$), is equivalent to

$$\mathrm{argmin}_x \mathrm{max}_{y\in\mathcal{Y}} V(\mathcal{C}(S_T + \{x, y\})). \tag{5.2}$$

In this section we augment the original algorithm to address the aforementioned issues. The critical properties we exploit for computational tractability are duality, present due to the structure of the function space, and convexity, introduced to handle misspecified models. To efficiently construct $\mathcal{C}$ and choose a function from it, we derive convex dual constraints and optimize within them. To efficiently choose a label, we establish a dual probability measure and efficiently sample from it by virtue of its logconcave density. Both of these approximations have strong quality guarantees.

5.4.1 Misspecified models, multicategory classification, and kernels

To accommodate misspecified models, we generalize the version space to a candidate space of all functions with empirical risk below some threshold. That is,

$$\mathcal{C}(S_T) = \left\{ f : \hat{R}_\Psi(f) \le \hat{R}_\Psi(f^*_T) + \Delta(S_T) \right\},$$

where the empirical Ψ-risk is

$$\hat{R}_\Psi(f) = \frac{1}{T} \sum_{t=0}^{T} \Psi_{Y_{\pi_t}}(f(X_{\pi_t}))$$

and f^*_T is the (regularized) empirical Ψ-risk minimizer over $\mathcal{C}(S_T)$. This generalization, previously adopted by active learning algorithms with strong label complexity improvement guarantees [16, 7], leaves the important choices of a slack function Δ and a loss function Ψ.

Slack function

The slack function should bound the discrepancy between the empirical risk and the true risk. Basic slack functions are coarsely parametrized by T and constants such as δ. Tighter slack functions depend on the data and capacity of the hypothesis space [16]. Judicious choice of a slack function can significantly improve label complexity; for example, [28] exploits benign noise conditions by replacing the slack function of [16] for one involving local Rademacher complexity. Here, we choose a conservative, data-independent slack function $\Delta \in \Omega(n^{-1/2})$, where $O(n^{-1/2})$ is the standard worst-case rate of passive learning [11].

Multiclass loss

For computational tractability, Ψ is usually taken to be a convex surrogate of the 0-1 loss. Ψ is called *Bayes consistent* or *classification calibrated* if asymptotic minimization of Ψ-risk implies asymptotic minimization of risk:

$$\hat{R}_\psi \to R^*_\psi \implies \hat{R} \to R^* \text{ as } T \to \infty$$

When $|\mathcal{Y}| = 2$, typical choices of Ψ — hinge loss, squared hinge loss, logistic loss, exponential loss — are all Bayes consistent [5]. For multicategory problems ($|\mathcal{Y}| \geq 2$), it is possible to retain binary loss and perform classification through an aggregation such as one-versus-rest, but such approaches are suboptimal [43, 34]. However, generalizing binary loss is not obvious. Bayes consistency is subtle in the multicategory setting and excludes seemingly straightforward generalizations; Tewari and Bartlett provide a generalization that works [59]. If a binary, convex, margin-based loss function $\psi(f_i(x)y_i) = \psi(m)$ is differentiable on $(-\infty, 0]$ and $\psi'(0) < 0$, then the multicategory loss function

$$\Psi_{Y_{\pi_t}}(f(X_{\pi_t})) = \sum_{y \neq Y_{\pi_t}} \psi(-f_y(X_{\pi_t}))$$

is Bayes consistent under the sum-to-zero constraint

$$\left\{ f : \sum_{y \in \mathcal{Y}} f_y(x) = 0 \right\} = \mathcal{Z}.$$

As in [43], we use the hinge loss $\psi(m) = \max(0, 1 - m)$. The empirical Ψ-risk term is:

$$\hat{R}_\Psi(f) = \frac{1}{T} \sum_{t=1}^{T} \sum_{y \neq Y_{\pi_t}} \max(0, f_y(X_{\pi_t}) - \mathbf{Y}_{t,y})$$

where $\mathbf{Y}_{\cdot,y}$ is a $T \times |\mathcal{Y}|$ matrix with row t equal to 1 in the Y_{π_t}th column and $-1/(|\mathcal{Y}| - 1)$ elsewhere. We introduce norm regularization and thereby gain the benefits of the maximum margin hyperplane [62]. The non-operational optimization problem for f_T^* is:

$$\begin{aligned} \underset{f}{\text{minimize}} \quad & \hat{R}_\Psi(f) + \frac{1}{2}\rho \sum_{y \in \mathcal{Y}} ||f_y||_k^2 \\ \text{subject to} \quad & \mathcal{Z}, \end{aligned}$$

where ρ is a regularization constant. The non-operational constraints specifying $\mathcal{C}$ are

$$\mathcal{C}(S_T) = \Big\{ f : \hat{R}_\Psi(f) \leq \hat{R}_\Psi(f_T^*) + \Delta(S_T) \Big\} \cap \mathcal{Z}.$$

These are operationalized by formulating the Wolfe dual, which replaces direct inner product computation with kernel invocation. We adapt the derivation of [43] and skip the intermediate primal Lagrangian for brevity. In the dual problem, we optimize $\mathbf{A} \in \mathbb{R}^{T \times |\mathcal{Y}|}$ and compute a normalized coefficient

$$\boldsymbol{\alpha}_{\cdot,y} = \eta(\mathbf{A})_{\cdot,y} = -(\mathbf{A}_{\cdot,y} - \overline{\mathbf{A}})/(T\rho),$$

where $\overline{\mathbf{A}} = \left(\sum_{y \in \mathcal{Y}} \mathbf{A}_{\cdot,y} \right) / |\mathcal{Y}|$. The coefficient $\boldsymbol{\alpha}$ defines an $f \in \mathcal{F}$ in the usual manner. Let $\hat{R}_\Psi(\mathbf{A})$ denote the empirical Ψ-risk of the function so defined:

$$\hat{R}_\Psi(\mathbf{A}) = \hat{R}_\Psi \left(\sum_{t=1}^{T} \{ \eta(\mathbf{A})_{t,y} k(X_{\pi_t}, \cdot) : y \in \mathcal{Y} \} \right).$$

Theorem 1 of [43] shows that, for the purpose of regularized risk minimization, imposing the sum-to-zero constraint upon only the training data is equivalent to imposing the sum-to-zero constraint upon all possible X. The dual form of the sum-to-zero constraint is:

$$\big\{ \mathbf{A} : (\mathbf{A}_{\cdot,y} - \overline{\mathbf{A}})\mathbf{1} = 0 \ \forall\, y \in \mathcal{Y} \big\} = \underline{\mathcal{Z}}.$$

The dual optimization problem for f_T^* defined by $\mathbf{A}_T^*$ is:

$$\begin{aligned} \underset{\mathbf{A}}{\text{maximize}} \quad & \frac{1}{2} \sum_{y \in \mathcal{Y}} (\mathbf{A}_{\cdot,y} - \overline{\mathbf{A}}) \mathbf{k}[\pi] (\mathbf{A}_{\cdot,y} - \overline{\mathbf{A}}) + \\ & T\rho \sum_{y \in \mathcal{Y}} \mathbf{A}_{\cdot,y} \mathbf{Y}_{\cdot,y} \\ \underset{\forall y \in \mathcal{Y},\ 1 \leq t \leq T}{\text{subject to}} \quad & 0 \leq \mathbf{A}_{t,y} \leq \mathbb{I}(Y_{\pi_t} \neq y) \\ & \underline{\mathcal{Z}}, \end{aligned}$$

where $\mathbf{k}[\pi]_{i,j} = \mathbf{k}_{\pi_i,\pi_j}$ is a permuted restriction of $\mathbf{k}$ to the data $X_{\pi_1}, \ldots, X_{\pi_T}$, and $\mathbb{I}(\cdot)$ is the indicator function. The hypothesis formed from f_T^* is $\hat{h}(X) = \text{argmax}_y f_{T,y}^*(X)$. Finally, the dual constraints corresponding to $\mathcal{C}$ are:

$$\underline{\mathcal{C}}(S_T) = \Big\{ \mathbf{A} : \hat{R}_\Psi(\mathbf{A}) \leq \hat{R}_\Psi(\mathbf{A}_T^*) + \Delta(S_T) \Big\} \cap \underline{\mathcal{Z}}$$

5.4.2 Querying via measure approximation

Though the maximin (equation (5.1)) and minimax (equation (5.2)) sampling criteria are both equivalent for binary classification ($|\mathcal{Y}| = 2$), they are not when $|\mathcal{Y}| > 2$. The criterion should ensure that queries eliminate a large proportion of the candidates. Maximin sampling offers no such guarantee; it would choose a point yielding candidate spaces with measures $(0.7, 0.2, 0.1)$ over a point yielding candidate spaces with measures $(0.5, 0.3, 0.2)$, though the latter is clearly superior against an adversarial oracle. Here we show how the minimax criterion can be practically computed by formulating, and subsequently sampling from, a dual measure on Euclidean space.

Measure representation and conditions

Any measure V on $(\mathcal{F}, \mathcal{B}(\mathcal{F}))$ has a dual measure

$$\underline{V_\alpha}(\underline{B}) = V(\{f : \kappa^{-1}(f) \in \underline{B}\})$$

on $(\mathbb{R}^{n\times|\mathcal{Y}|}, \mathcal{B}(\mathbb{R}^{n\times|\mathcal{Y}|}))$ due to κ which, recall, is the map from coefficients to functions. We call the measure dual because any $\underline{V_\alpha}$ has

$$V(B) = \underline{V_\alpha}(\{\boldsymbol{\alpha} : \kappa(f) \in B\}).$$

This holds because κ is an isomorphism between $\mathcal{F}$ and $\mathbb{R}^{n\times|\mathcal{Y}|}$. An $\boldsymbol{\alpha}$ cannot define more than one f because κ is a function. Each f is uniquely determined by coefficients multiplied with rows of $\mathbf{k}$, or a suitably transformed version of it. If the determinant of $\mathbf{k}$ is non-zero then the spanning functions $\{k(x_i, \cdot) : 1 \le i \le n\}$ are linearly independent and therefore form a basis. If the determinant is zero, the spanning functions are linearly dependent and therefore unique coordinates for f are not guaranteed. To produce an orthonormal basis, first note that

$$\begin{aligned} k(x_i, x_j) &= \sum_{a=1}^{n}\sum_{b=1}^{n} \alpha_a \beta_b k(x_i, x_j) \\ &\text{with: } \alpha_a = 1 \text{ if } a = i, 0 \text{ otherwise} \\ &\text{and: } \beta_b = 1 \text{ if } b = j, 0 \text{ otherwise} \\ &= \langle \sum_{a=1}^{n} \alpha_a k(x_i, \cdot), \sum_{b=1}^{n} \beta_b k(x_j, \cdot) \rangle_k \\ &= \langle k(x_i, \cdot), k(x_j, \cdot) \rangle_k, \end{aligned}$$

where the first equality follows from simple algebra and the rest follow from the definition of the inner product k. By lemma 1 of [24], given the eigenvectors $\gamma_1, \dots, \gamma_b$ and corresponding eigenvalues $\lambda_1, \dots, \lambda_b$ of $\mathbf{k}$, an orthonormal basis for $\text{span}\{k(x_i, \cdot) : 1 \le i \le n\} = f$ is

$$\left\{ \sum_{j=0}^{n} \frac{\gamma_i(j)}{\sqrt{\lambda_i}} k(X_j, \cdot) : 1 \le i \le b \right\}.$$

This transformation of $\mathbf{k}$ is efficiently precomputed and yields a simple way to uniformly sample from f and, by extension, $\mathcal{F}$: just uniformly sample the coefficients to multiply with the basis. For notational convenience, let $\mathbf{k}$ and related quantities such as n refer to the transformed kernel matrix if such a transformation is necessary.

Since κ and κ^{-1} are continuous, κ is bijective and bimeasurable and therefore an isomorphism. Since the normalization function η is continuous, $\underline{V_\alpha}$ is the measure induced by the random matrix η which maps from the probability space $(\mathbb{R}^{n\times|\mathcal{Y}|}, \mathcal{B}(\mathbb{R}^{n\times|\mathcal{Y}|}), \underline{V})$. $\underline{V}(\underline{\mathcal{C}})$ is the dual equivalent of $V(\mathcal{C})$, and it is $\underline{V}$ which we will approximate. Practically, it may be easier to just specify $\underline{V}$ directly.

In any case, $\underline{V}$ must satisfy some technical conditions in order for efficient approximation algorithms to be applied. $\underline{V}$ must have a logconcave density with respect to Lebesgue measure $\mathcal{L}$; that is, there exists a nonnegative integrable function p with convex support such that $\log p(\mathbf{A})$ is concave and $\underline{V}(\underline{B}) = \int_{\underline{B}} p(\mathbf{A}) d\mathcal{L}(\mathbf{A})$ for every $\underline{B} \in \mathcal{B}(\mathbb{R}^{n\times|\mathcal{Y}|})$. We also require that $\underline{V}$ isn't too diluted or concentrated in the following sense:

- The variance of $\underline{V}$ is bounded by S.
- If the $\underline{V}$-measure of a level set $\{\mathbf{A} : p(\mathbf{A}) \geq c\}$ is greater than $1/8$, then the level set contains a ball of radius $s > 0$.

We require that the support of p be bounded so its volume can be meaningfully approximated. Finally, p must be 'rounded' so that $\sqrt{S}/s \in O(\sqrt{n})$. This can be accomplished by placing p into isotropic position via an affine transformation. Using the technique described in Section 6 of [44], such a transformation can be determined and applied in $O^*(n^4)$ time, where $O^*(\cdot)$ ignores logarithmic terms and error parameters. This technique requires a point which maximizes p. A maximal point can be found via optimization, but providing one reduces the computational complexity of the algorithm.

Approximate integration

The constraints specifying $\underline{\mathcal{C}}$ are linear, so $\underline{\mathcal{C}}$ is the intersection of a finite number of halfspaces i.e. a convex polyhedron. This intersection is bounded due to the sum-to-zero constraint and the bounded support of p, so $\underline{\mathcal{C}}$ is a convex polytope. $\underline{V}(\underline{\mathcal{C}}) = \int_{\underline{\mathcal{C}}} p(\mathbf{A})\mathcal{L}(d\mathbf{A})$ is a high-dimensional integral which is generally difficult to exactly compute. Even when $\underline{V}$ is uniform, making $\underline{V}(\underline{\mathcal{C}})$ a volume, exact computation is **#P**-hard [19] since we are provided only a 'halfspace representation' of the polytope. Fortunately, effective and practical approximation algorithms are available due to the existence of a logconcave density p. The state-of-the-art algorithm by Lovasz and Vempala [44] takes the same 'multi-phase Monte Carlo' approach as previous work in volume computation and can be interpreted as a form of simulated annealing. Here we outline the algorithm while verifying the conditions for its application; refer to [44] for full implementation details.

1. From a sequence of 'temperatures' $\{\tau_m\}_{m=1}^{M}$ decreasing to 1, construct a corresponding sequence of functions $\{q_m\}_{m=1}^{M}$ of the form $p(\mathbf{A})^{1/\tau_m}$. The parameters M and

$\{\tau_m\}$ are chosen to scale with n. The functions are nearly uniform over $\underline{\mathcal{C}}$ at the beginning of the sequence and nearly equal to p at the end. The sequence connects a function that is easy to integrate to a function that is hard to integrate with pairs that are 'close'.

2. Approximate the ratio of the integrals of each pair (q_m, q_{m+1}). The first function q_0 is uniform on $\underline{\mathcal{C}}$, so take the volume as the integral, and uniformly sample a set of points from $\underline{\mathcal{C}}$. The number of points scales with n. Since q_m and q_{m+1} are close, randomly sampled points from q_m are 'warm starts' for randomly sampling from q_{m+1}. The approximation is basically an average of q_{m+1} evaluated at these points.

3. Return the product of the integral ratios as an approximation of the integral $\underline{V}(\underline{\mathcal{C}})$.

The key subprocedure invoked by the algorithm is for efficiently sampling from a density restricted to $\underline{\mathcal{C}}$ given some starting point $\mathbf{B} \in \underline{\mathcal{C}}$. This is typically implemented by a Markovian geometric walk starting from a point $\mathbf{B} \in \underline{\mathcal{C}}$. Among the various geometric walks, we choose the hit-and-run walk because it mixes rapidly for logconcave densities [44]. The hit-and-run walk iterates this procedure:

1. Pick a uniform random line through $\mathbf{B}$. As mentioned in §5.4.2, it suffices to uniformly sample each element of $\mathbf{M}$ randomly and use $\mathbf{A} \mapsto \mathbf{MA} + \mathbf{B}$.

2. Walk along the line to a point chosen from the restriction of p to the line segment within $\underline{\mathcal{C}}$.

Unlike some other sampling algorithms, this one has strong guarantees on the worst-case number of calls to p required to achieve a certain quality of approximation. If a maximal point is provided, the integration is $O^*(n^4)$, otherwise it is $O^*(n^{4.5})$ due to the maximization step [44]. These bounds are considered loose; far fewer calls are required in practice.

Now it is possible to compute (5.2). This computation is trivially parallelizable. Approximation of all $n \times |\mathcal{Y}|$ measures may be amortized because, intuitively, the candidate spaces should shrink as we see more labels. This monotonicity can be exploited to avoid unnecessary measure approximations. Monotonicity is enforced by instead measuring

$$\underline{\mathcal{C}}'(S_T) = \bigcap_{t=1}^{T} \underline{\mathcal{C}}(S_t).$$

Now $\underline{V}(\underline{\mathcal{C}}'(S_T)) \geq \underline{V}(\underline{\mathcal{C}}'(S_T + \{x, y\}))$. For the first query, approximate and save the $n * |\mathcal{Y}|$ measures. When considering an x for a subsequent query, approximate the $\underline{V}(\underline{\mathcal{C}}'(S_T + \{x, y\}))$ in decreasing order of $\underline{V}(\underline{\mathcal{C}}'(S_T))$. If $\underline{V}(\underline{\mathcal{C}}'(S_T + \{x, y\}))$ is larger than the remaining 'stale' approximations, then it will remain the maximum; break and return it. Though this heuristic is not guaranteed to reduce the worst-case amount of computation, it maintains correctness and allows us to treat measure approximation as a black box.

5.4.3 Remarks

The preceding sampling techniques may also be used to solve the convex optimization problem in section 5.4.1. Thus, ALMA lends itself to a self-contained software implementation.

In the original setting, ALMA asymptotically retains the original quality guarantee with arbitrarily small error. Under a correctly specified model, the slack function can be set to zero and the SVM may use a hard margin. For binary classification, the $|\mathcal{Y}|$-power Hilbert space and multicategory loss function degenerate to usual hinge loss [43]. This loss is Bayes consistent, so the surrogate risk minimization implies risk minimization. Also, the minimax criterion becomes equivalent to the maximin criterion. Measure approximation is arbitrarily precise.

5.5 Experimental results

We evaluate our proposed method, ALMA, against both passive learning and state-of-the-art pool-based active learning methods on standard image classification datasets. Our experimental methodology, described below, is motivated by Monteleoni and Kääriäinen [47].

We present results on experiments using two publicly-available datasets, Caltech-256 [27] and Microsoft Research Cambridge Object Identification (MSRCORID) [46]. We used images from four distinct categories from each dataset, extracted features, generated kernel matrices, and used these to train and test three algorithms including ALMA. SVM, specifically *SVM*^{multiclass} provided by [35], serves as the strong passive learning baseline algorithm in our experiments, while pKNN+AL, described in [34] and provided by [33], represents the state-of-the-art in pool-based active learning. We selected pKNN+AL because it consistently outperforms both [38] and [60] in experiments.

We used SURF [6] to extract low-level features on a densely-sampled grid, from which pyramid match kernels [25] were computed using the LIBPMK [42] implementation.

From Caltech-256 [27], we chose the following four classes: ladders, bathtubs, binoculars, and gorillas. We took 210 images from each class, used a fixed test set size of 70 images per class, and used a training set size of 20 giving 7 unique training sets. We ran each algorithm 7 times — once for each training set —and averaged the results from these runs to produce the error rates shown in figures 5.2 and 5.3.

From MSRCORID [46], we also chose four classes: windows, cars, clouds, and animals. We took 360 images from each class, used a fixed test set size of 120 images per class, and used a training set size of 20 giving 12 unique training sets. We ran each algorithm 12 times and averaged the results from these runs to produce the error rates shown in figure 5.3.

Figure 5.2 shows that both active learning algorithms consistently outperform *SVM*^{multiclass} [35] with fewer examples, and that ALMA performs comparably with other contemporary active learning algorithms with the same number of requested labels. Figure 5.3 reinforces both

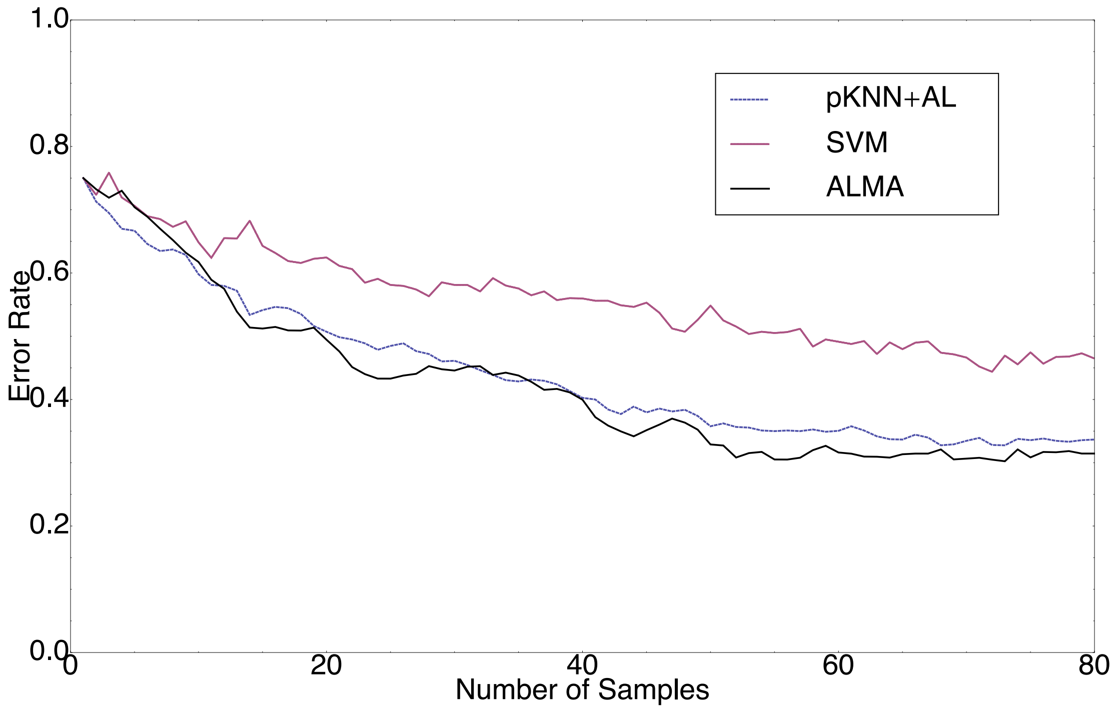


Figure 5.2: Error rates on Caltech-256 [27]. Both active learning methods outperform SVM, and ALMA slightly outperforms pKNN+AL.

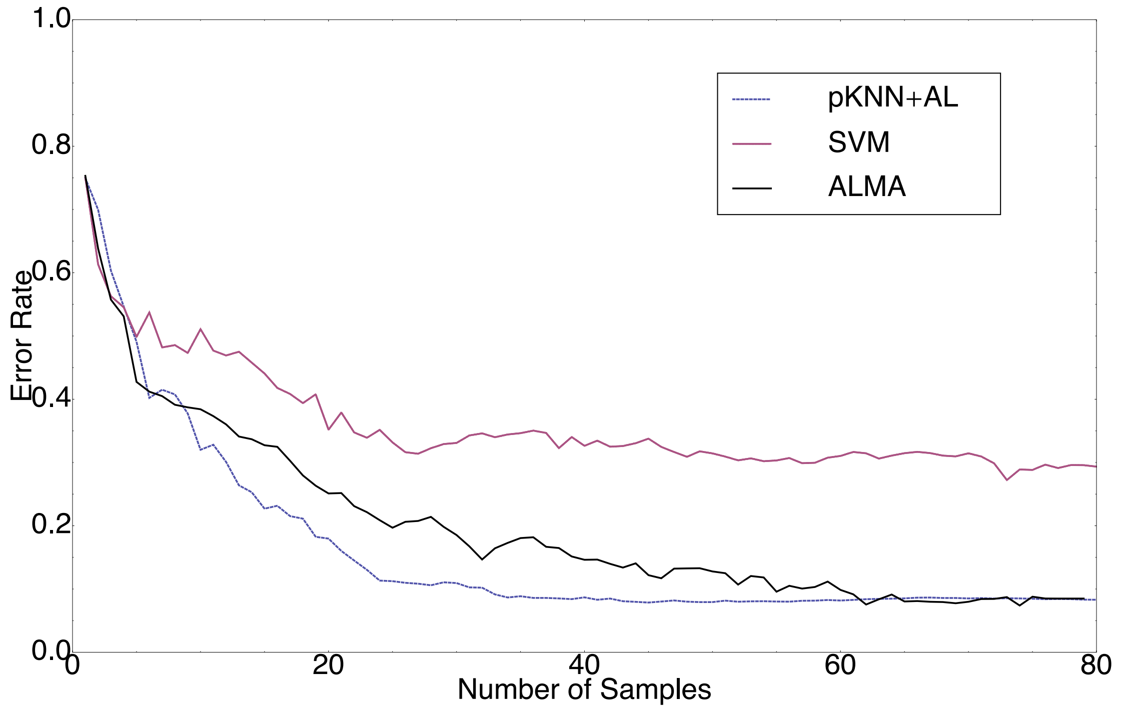


Figure 5.3: Error rates on the Microsoft Research Cambridge Object Identification (MSR-CORID) dataset [46]. Both active learning methods outperform SVM. ALMA converges more slowly than pKNN+AL.

the performance of active learning algorithms over passive learners, and the comparable performance of ALMA against pKNN+AL. Though not always outperforming pKNN+AL, ALMA yields comparable error rates. pKNN+AL outperforms ALMA with fewer requested labels, but pKNN+AL's performance plateaus after 40 labeled examples while ALMA continues improving, and exceeding pKNN+AL after 60 labeled examples.

5.6 Conclusion

By augmenting an impractical algorithm with a relatively rare quality guarantee, we obtained a general-purpose algorithm that, with respect to the aforementioned priorities in computer vision, is the first of its kind. Each generalization included a quality guarantee. Duality allowed us to reformulate the optimization and measure approximation problems, and convexity allowed us to efficiently solve these problems in their new setting. In our experiments, which spanned different levels of difficulty, ALMA was superior to passive learning and competitive with state-of-the-art active learning.

To efficiently sample high-dimensional functions, we utilized a geometric walk within the dual polytope. This technique may be applied to a wide class of algorithms which utilize hypotheses defined by a reproducing kernel Hilbert space.

As described in the next chapter, integrating ALMA and Algum is future work.

Chapter 6

Related and Future Work

This thesis introduces many new concepts into Diamond. Many future research opportunities were discovered due to the exploratory nature of this work.

6.1 Diamond

This thesis builds upon, but does not modify, any existing concepts in Diamond or code in OpenDiamond. For a thorough listing of related work to Diamond, see the original Diamond paper [31].

Filter scheduling via shared conjunctive queries: though heuristic algorithms have been proposed for filter reordering, more principled approaches are possible. For single-user Diamond search, filter reordering is equivalent to min-sum set cover. This problem is NP hard and, unless P = NP, cannot be approximated by a factor better than 4. A hardness result and a greedy 4-approximation are given by [22]. Fortunately, the number of filters is usually small in practice. For multi-user Diamond search, (dynamic) filter reordering is equivalent to evaluating ”shared conjunctive queries” in which runtime adaptivity is exploited [48].

6.2 Semantic Texton Forests

Since STF is meant to be used quasi-interactively, performance improvements are very welcome.

Distributed computation with Dryad: our STF utilizes concurrency and parallelism facilities in C# 4.0 originally released as the Parallel Computing Toolkit. These facilities have been augmented by Dryad [32], which is essentially a drop-in replacement enabling distributed parallel computation. DryadLINQ is available at no cost for non-commercial use.

Virtual machine improvements: the Mono compiler will soon support Low-Level Virtual

Machine [41] as a backend. This software boasts massively improved performance on 'scientific computation' tasks such as STF. As compilers mature, further optimizations could be realized by treating the decision forests and classifiers as constants during compilation rather than objects deserialized at runtime. (Also see the `al/mkbundle` functionality for reducing assembly size, as well as full Ahead-Of-Time compilation.)

New machine learning techniques may be applied to improve the classification quality of STF. STF is an interesting method because it uses the structure of a good classifier to represent the structure of images. That is, the path a training datum takes down the tree is used as a feature rather than the value assigned to a leaf node. It is plausible, though not necessarily true, that improving the output of the classifier would improve the representation of the images. This is related to work on generative/discriminative and interpretable [57] models in machine learning.

Subsampling: the binomial sampling method used in STF belongs to a broader class of 'resampling' methods [40]. The binomial resampling method used in STF is somewhat odd. In particular, it gives some trees more training pixels than other trees. More typical and well-studied approaches would allocate exactly the same amount of training pixels to each tree. One such approach is 'bagging' (bootstrap aggregation), wherein pixels are sampled with replacement from the training dataset. A mildly different and perhaps more modern approach is 'moonbagging' (m/n bootstrap aggregation) [10], wherein the same number of pixels are sampled *without* replacement from the training dataset. Though these techniques sound similar, they depend on very different theoretical justifications. Bagging is based on the assumption of analogy: that the relationship between the population and the training sample is similar to the relationship between the training sample and the subsamples. In moonbagging, subsamples are individually bona-fide iid samples directly from the population, though collectively they are dependent.

Consistent learning of decision forests: a classifier is called consistent when its excess risk converges in probability to zero. Biau, Devroye, and Lugosi show that a classic bagged decision forest is not consistent, and fix the method by utilizing more complicated decisions at each node [8]. Though this result does not directly apply to STF decision trees, two factors suggest there isn't much hope. Binomial sampling is not as trustworthy as the bootstrap. Also, in order to select each node, STF performs approximate, not exact, optimization.

6.3 (Interactive) Learning Algorithms

Better learning could occur through better modeling of the training labels and thresholds.

Imperfect labelers: human users do not provide perfect image labels. Any classification algorithm used within Diamond should accommodate label noise, as ALMA does. However, if Algum is extended to have social features (see below), this would not be sufficient. Each user may make distinctive errors; for example, one may not have a penchant for color. Since Algum records the source of each label, such systematic errors could be accommodated [18].

Support and Interpretation of Searchlet Threshold: thresholds are an important feature for Diamond search. Sometimes, the 'interactive' component of a Diamond search consists solely of threshold adjustment. Diamond does not interpret the thresholds; the only property they must satisfy is that a result returned at a high threshold must also be returned at a low threshold. SVMs, as used in STF or ALMA, do not support thresholds. In order to add support, the threshold must be interpreted probabilistically. These are some suggested interpretations, each leading to different classification techniques:

- the threshold represents 1 minus the probability of misclassification, whether false positive or false negative: this leads to the Bayesian inference procedure described below. The advantage of this interpretation is that it accommodates both kinds of error. The disadvantage is that it requires a Bayesian prior.
- the threshold represents the cost of different kinds of misclassification: this leads to the cost sensitive classification paradigm. Of course, this approach presumes that each kind of misclassification actually has a well-defined cost [20].
- the threshold represents 1 minus the false positive probability: this leads to Neyman-Pearson approach to statistical learning inspired by the Neyman-Pearson approach to hypothesis testing [53]. The threshold fixes the false positive probability; under this constraint, the classifier achieving the lowest false negative probability is sought. One advantage of this approach is that it naturally adjusts to positive/negative imbalances in the distribution.

Improving the efficiency of ALMA:

ALMA is not an optimal algorithm for active learning, and active learning is not an optimal model of interaction within Diamond. These are some opportunities for reducing training time and the number of required labels.

Improvements to ALMA: A data-dependent slack function could reduce the number of required labels. Sampling could be amortized to a greater degree by not treating measure approximation as a black box. Integration could be performed on $\mathbb{R}^{T\times|\mathcal{Y}|}$ instead of $\mathbb{R}^{n\times|\mathcal{Y}|}$ by imposing further conditions on the density p. If the measure V is to be interpreted as a Bayesian prior, then it would be preferable to perform Bayesian inference. The particular multicategory SVM we utilize can form the basis of fully Bayesian inference [65].

Active learning and kernel learning: this thesis has not addressed how to optimally combine STF and active learning. ALMA is kernelized, but it assumes the kernel matrix is constant. By contrast, every time an additional labeled image is provided, STF learns a different kernel. ALMA and STF could be easily integrated by using the first few training data to learn a fixed kernel, and using ALMA to request further examples. Another approach is to learn the kernel and perform active learning simultaneously. However, it is not known if active learning is better than passive learning in this setting.

More flexible queries: in active learning, only one kind of query can be issued: "what is the label of this datum?". This restriction is both burdensome, leading to unusually high lower bounds, as well as unrealistic. Other models of interactive learning are available. For example, Hanneke proposes a cost-based model wherein the algorithm may issue a variety of queries for various costs [29]. Though such models are immature, they have promise for Diamond.

6.4 Algum

Training runtime estimation: 'normal' STF training sessions take over 10 minutes, so Algum is typically used asynchronously; that is, hit 'train' and come back later. Users would appreciate some kind of estimate of when to return. Generally stated, for some fixed algorithm, we have a dataset $\{Z_i, X_i, N_i, T_i\}_{i=1}^n$ where Z_i is the input to the algorithm, X_i is a p-dimensional vector of features extracted from Z_i (e.g. number of entries), N_i is the number of core operations performed by the algorithm (e.g. number of comparisons), and T_i is the recorded runtime of the algorithm. The X_i come from a fixed but unknown distribution. T_i is assumed to be a function of N_i, T_i, an independent, uniformly distributed U_i for internal randomization, and an error term ϵ_i due to perturbances in the computing environment. We will assume the features are fairly well chosen (i.e. not much sparsity there.) Most existing work on 'runtime estimation' deals with the opposite scenario of job scheduling, where the machine is fixed and the algorithms vary. Some form of nonparametric regression could work as a first-order approximation.

Many negative examples: in most image searches, only a small fraction of the images are returned as hits. To reflect this imbalance, image classifiers are typically trained with many negative examples. Algum lacks a facility for quickly labeling many negative examples. Some computer vision datasets include 'clutter' examples which are all negative. In a similar spirit, Algum could include a default clutter set.

Social features: Algum is similar to LabelMe, a web application developed at MIT designed, not surprisingly, for labeling images [51]. Unlike Algum, all images and labels in LabelMe are public. Algum could benefit from some form of image sharing among tasks and users.

Bibliography

[1] András Antos and Gábor Lugosi. Strong minimax lower bounds for learning. *Mach. Learn.*, 30(1):31–56, 1998.

[2] Maria-Florina Balcan, Alina Beygelzimer, and John Langford. Agnostic active learning. In *ICML*, pages 65–72, 2006.

[3] Maria-Florina Balcan, Andrei Broder, and Tong Zhang. Margin based active learning. In *COLT*, 2007.

[4] Maria-Florina Balcan, Steve Hanneke, and Jennifer Wortman. The true sample complexity of active learning. In *COLT*, pages 45–56, 2008.

[5] Peter L. Bartlett, Michael I. Jordan, and Jon D. McAuliffe. Large margin classifiers: Convex loss, low noise, and convergence rates. In *NIPS*, 2003.

[6] Herbert Bay, Tinne Tuytelaars, Van Gool, and L. Surf: Speeded up robust features. In *9th European Conference on Computer Vision*, Graz Austria, May 2006.

[7] Alina Beygelzimer, Sanjoy Dasgupta, and John Langford. Importance weighted active learning. *CoRR*, abs/0812.4952, 2008.

[8] Gérard Biau, Luc Devroye, and Gábor Lugosi. Consistency of random forests and other averaging classifiers. *J. Mach. Learn. Res.*, 9:2015–2033, 2008.

[9] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[10] Buhlmann and Bin Yu. Analyzing bagging. *Annals of Statistics*, 30:927–961, 2002.

[11] Rui Castro and Robert D. Nowak. Minimax bounds for active learning. In *COLT*, pages 5–19, 2007.

[12] David Cohn, Les Atlas, and Richard Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1992.

[13] Sanjoy Dasgupta. Analysis of a greedy active learning strategy. In *NIPS*, 2004.

[14] Sanjoy Dasgupta. Coarse sample complexity bounds for active learning. In *NIPS*, pages 235–242, 2005.

[15] Sanjoy Dasgupta and Daniel Hsu. Hierarchical sampling for active learning. In *ICML*, pages 208–215, 2008.

[16] Sanjoy Dasgupta, Daniel Hsu, and Claire Monteleoni. A general agnostic active learning algorithm. In *NIPS*, 2007.

[17] Sanjoy Dasgupta, Adam Tauman Kalai, and Claire Monteleoni. Analysis of perceptron-based active learning. *JMLR*, 10:281–299, 2009.

[18] Pinar Donmez and Jaime G. Carbonell. Proactive learning: cost-sensitive active learning with multiple imperfect oracles. In *CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge management*, pages 619–628, New York, NY, USA, 2008. ACM.

[19] M. E. Dyer and A. M. Frieze. On the complexity of computing the volume of a polyhedron. *SIAM J. Comput.*, 17(5):967–974, 1988.

[20] Charles Elkan. The foundations of cost-sensitive learning. In *IJCAI*, pages 973–978, 2001.

[21] Christos Faloutsos. *Searching Multimedia Databases by Content*. Kluwer Academic Publishers, Norwell, MA, USA, 1996.

[22] Uriel Feige and Prasad Tetali. Approximating min sum set cover. *Algorithmica*, 40(4):219–234, 2004.

[23] Roy Thomas Fielding. *REST: Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.

[24] Ran Gilad-Bachrach, Amir Navot, and Naftali Tishby. Query by committee made real. In *NIPS*, 2005.

[25] K. Grauman and T. Darrell. The pyramid match kernel: Efficient learning with sets of features. *JMLR*, 8, 2007.

[26] Kristen Grauman and Trevor Darrell. The pyramid match kernel: Efficient learning with sets of features. *J. Mach. Learn. Res.*, 8:725–760, 2007.

[27] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical Report 7694, California Institute of Technology, 2007.

[28] Steve Hanneke. Adaptive rates of convergence in active learning. In *COLT*, 2009.

[29] Steve Hanneke. Theoretical foundations of active learning. Technical report, Carnegie Mellon, 2009.

[30] Steven C. H. Hoi, Rong Jin, Jianke Zhu, and Michael R. Lyu. Batch mode active learning and its application to medical image classification. In *ICML*, pages 417–424, 2006.

[31] Larry Huston, Rahul Sukthankar, Rajiv Wickremesinghe, M. Satyanarayanan, Gregory R. Ganger, Erik Riedel, and Anastassia Ailamaki. Diamond: A storage architecture for early discard in interactive search. In *FAST '04: Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, pages 73–86, Berkeley, CA, USA, 2004. USENIX Association.

[32] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *EuroSys '07: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, pages 59–72, New York, NY, USA, 2007. ACM.

[33] Prateek Jain and Ashish Kapoor. *Probabilistic Nearest Neighbor Classifier with Active Learning*. Microsoft Research, Redmond, `http://www.cs.utexas.edu/users/pjain/pknn/`.

[34] Prateek Jain and Ashish Kapoor. Active learning for large multi-class problems. In *CVPR*, 2009.

[35] Thorsten Joachims. Making large-scale support vector machine learning practical. In *Advances in kernel methods: support vector learning*, pages 169–184. MIT Press, 1999.

[36] Ajay Joshi, Fatih Porikli, and Nikolaos Papanikolopoulos. Multi-class active learning for image classification. In *CVPR*, 2009.

[37] Matti Kääriäinen. Active learning in the non-realizable case. In *ALT*, pages 63–77, 2006.

[38] Ashish Kapoor, Kristen Grauman, Raquel Urtasun, and Trevor Darrell. Gaussian processes for object categorization. *IJCV*, 2009.

[39] H. Krawczyk, M. Bellare, and R. Canetti. Hmac: Keyed-hashing for message authentication, 1997.

[40] S.N. Lahiri. *Resampling Methods for Dependent Data*, volume 100. June 2005.

[41] Chris Lattner and Vikram Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04)*, Palo Alto, California, Mar 2004.

[42] John J. Lee. LIBPMK: A pyramid match toolkit. Technical Report MIT-CSAIL-TR-2008-17, MIT Computer Science and Artificial Intelligence Laboratory, April 2008.

[43] Yoonkyung Lee, Yi Lin, and Grace Wahba. Multicategory support vector machines: Theory and application to the classification of microarray data and satellite radiance data. *Journal of the American Statistical Association*, 99:67–81, 2004.

[44] L. Lovász and S. Vempala. Fast algorithms for logconcave functions: Sampling, rounding, integration and optimization. In *Proc. Foundations of Computer Science*, 2006.

[45] Costas Petrou Maria Petrou. *Image Processing: The Fundamentals*. 2010.

[46] Microsoft Research Cambridge. Microsoft Research Cambridge Object Recognition Image Database. `http://research.microsoft.com/en-us/projects/objectclassrecognition/`.

[47] Claire Monteleoni and Matti Kääriäinen. Practical online active learning for classification. In *CVPR*, 2007.

[48] Kamesh Munagala, Utkarsh Srivastava, and Jennifer Widom. Optimization of continuous queries with shared expensive filters. Technical Report 2005-36, Stanford InfoLab, 2005.

[49] Robert Nowak. Generalized binary search. In *46th Annual Allerton Conference on Communication, Control, and Computing*, pages 568–574, 2008.

[50] Simon Perkins, James Theiler, Steven P. Brumby, Neal R. Harvey, Reid Porter, John J. Szymanski Jeffrey J. Bloch, John J. Szymanski, and Jerey J. Bloch. Genie: A hybrid genetic algorithm for feature classification in multi-spectral images. In *Proc. SPIE*, pages 52–62, 2000.

[51] Bryan C. Russell, Antonio Torralba, Kevin P. Murphy, and William T. Freeman. Labelme: A database and web-based tool for image annotation. *Int. J. Comput. Vision*, 77(1-3):157–173, 2008.

[52] Sukthankar R. Goode A. Huston L. Mummert L. Wolbach A. Harkes J. Gass R. Schlosser S Satyanarayanan, M. The opendiamond platform for discard-based search. Technical Report CMU-CS-08-132, Carnegie Mellon, 2008.

[53] C. Scott and R. Nowak. A neyman-pearson approach to statistical learning. *Information Theory, IEEE Transactions on*, 51(11):3806–3819, 2005.

[54] Burr Settles. Active learning literature survey. Technical Report CS-TR-1648, University of Wisconsin–Madison, 2009.

[55] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA, 2004.

[56] Jamie Shotton, Matthew Johnson, and Roberto Cipolla. Semantic texton forests for image categorization and segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

[57] Erik Strumbelj and Igor Kononenko. An efficient explanation of individual classifications using game theory. *Journal of Machine Learning Research*, 11:1–18, 2010.

[58] Rahul Sukthankar. personal communication, 2009.

[59] Ambuj Tewari and Peter L. Bartlett. On the Consistency of Multiclass Classification Methods. *JMLR*, 8:1007–1025, 2007.

[60] Simon Tong and Edward Chang. Support vector machine active learning for image retrieval. In *ACM Multimedia*, pages 107–118, 2001.

[61] Leslie G. Valiant. Evolvability. *J. ACM*, 56(1):1–21, 2009.

[62] V. Vapnik. *Statistical Learning Theory*. Wiley, 1998.

[63] I. Volnyansky and V. Pestov. Curse of Dimensionality in Pivot-based Indexes. *ArXiv e-prints*, June 2009.

[64] Liwei Wang. Sufficient conditions for agnostic active learnable. In *NIPS*, 2009.

[65] Zhihua Zhang and Michael I. Jordan. Bayesian multicategory support vector machines. In *UAI*, 2006.