Conditional Random Fields for Activity Recognition

Douglas L. Vail

CMU-CS-08-119

April, 2008

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

Thesis Committee:
Manuela Veloso, Co-Chair
John Lafferty, Co-Chair
Carlos Guestrin
Dieter Fox, University of Washington

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

This research was sponsored by a Graduate Fellowship from the National Physical Science Consortium; Directorate of Contracting under grant number DABT63-99-1-0013; National Science Foundation under grant number IIS-9900298; Air Force Research Laboratory under cooperative agreement number F30602-00-2-0549; L3 Communication Integrated Systems, L.D. under grant number 4500244745; SRI International under grant number 03-000211; Department of Justice under cooperative agreement number 2005IJCXK057; DARPA under grant number NBCH1040007. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: Conditional Random Fields, Activity Recognition, Robotics, Undirected Graphical Models, Feature Selection, M-estimators

For my father, who started it all

Abstract

To act intelligently in the presence of others, robots must use information from their sensors to recognize the behaviors and activities of the other agents in their environment. Robots must map from low-level, difficult to interpret data, such as position information extracted from video, to abstract states, in particular, the activities of the other agents. In this thesis, we explore how to bridge the gap from noisy, continuous observations about the world to high-level, discrete activity labels for robots in the environment.

We contribute the use of conditional random fields (CRFs) for activity recognition in multi-robot domains. We explore the appropriateness of CRFs with an empirical comparison to hidden Markov models. We elucidate the properties of CRFs that make them well suited to the activity recognition, namely discriminative training, the ability to robustly incorporate rich features of the observations, and their nature as conditional models, with a variety of synthetic and real robot data.

Accurate activity recognition requires complex and rich features of the observations. We choose the most informative features from a large set of candidates using feature selection. We adapt two feature selection algorithms, grafting and ℓ_1 regularization, to conditional random fields. We also investigate a third feature selection algorithm, which was originally proposed for CRFs in a natural language processing domain, in an activity recognition context. In particular, we focus on scaling feature selection to very large sets of candidate features that we define succinctly using a rich relational feature specification language.

The reduced feature sets that we discover via feature selection enable efficient, real-time inference. However, feature selection and training for conditional random fields is computationally expensive. We adapt an M-estimator, introduced by Jeon and Lin for log-density estimation in ANOVA models, for fast, approximate parameter estimation in CRFs. We provided an in depth, empirical evaluation of the properties of the M-estimator and then we introduce a new, efficient feature selection algorithm for CRFs based around M-estimation to identify the most important features.

Acknowledgments

I gratefully acknowledge the support of my advisors, Manuela Veloso and John Lafferty. It has been a privilege to work with two such extraordinary people and to grow under their guidance. These past seven years have been good for me, both as a person and a researcher, and I owe both Manuela and John a deep debt of thanks for their role as my mentors. I am grateful for the support and insight of my committee members, Carlos Guestrin and Dieter Fox. I thank them for their generosity with their time and suggestions for improving my thesis. I thank the members of the CMDragons07 robot soccer team, James Bruce, Michael Licitra, Stefan Zickler, and Manuela Veloso for sharing their log data from the RoboCup 2007 robot soccer championship. In particular, James Bruce and Stefan Zickler helped me work with their simulator and the data. I am grateful for the support of my friends in the robot soccer lab, James Bruce, Sonia Chernova, Scott Lenser, Colin McMillen, Maayan Roth, and the other members of the Coral research group.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Robots use their sensors to discover information about their own state and the state of their environment, including the actions and behaviors of other agents, either robot or human. Robot sensor data is typically continuous and noisy. Robots often combine sensor readings across time in order to extract useful information.

In contrast, robots often make decisions based on discrete or abstract state information that is far removed from noisy, low level sensor readings. For example, a robot vacuum cleaner might choose to pause cleaning if it detects that its owner is speaking on the telephone. The focus of this thesis is on methods to bridge the gap between the low level information provided by sensors and high level, abstract labels for the roles, activities, or behaviors of other robots or humans in the robot's environment.

In the remainder of the introduction, we discuss the properties of robot sensor data that make activity recognition from robot sensors a challenging problem. We describe the approach that we take in this thesis before concluding with a discussion of the contributions of the thesis and a chapter by chapter guide to the thesis.

1.1 Characteristics of Sensor Data

Recognizing activities based on robot sensor data is challenging due to a number of properties of the sensor data. Specifically, sensor data is:

Multi-dimensional: Robots have many sensors such as accelerometers, joint encoders, motor current readings, thermometers, and cameras. Integrating information from many (potentially non-independent) sources of data adds to the challenge of activity recognition.

Temporal: Sensor data arrives in a sequence over time. Richer information about the state of the world can be extracted by considering multiple samples from the stream rather than treating each sensor frame in isolation. In order to effectively classify sensor data, we require temporal models, which increases the complexity and computational requirements that we face.

Continuous and High Bandwidth: Sensor readings are often real valued and sampled at high frequencies. It is difficult to map from rapidly evolving streams of noisy, continuous values and long-lived, discrete state labels such as activity names.

Difficult to label: It is often difficult to obtain large amounts of fully labeled data when working with robots. Running experiments with physical hardware requires a substantial time investment and it is not always clear how labels should be assigned to the resulting data once it is collected. We require algorithms that are data efficient in the sense that they learn accurate models from limited amounts of training data.

1.2 Approach

We use graphical models, specifically, conditional random fields, as our approach for predicting activity labels from robot sensor data. Conditional random fields are well suited to the activity recognition problem because:

- Conditional random fields are models for structured prediction, i.e., they represent relationships between labels. We model activity recognition as a temporal classification problem using CRFs to model the sequence of activity labels rather than predicting the labels independently.
- Conditional random fields can robustly incorporate complex, arbitrary features of their observations in order to fuse information from a diverse set of sensors.
- Conditional random fields can draw on information from the entire observation sequence when predicting single labels, which means that they can consider aggregates and averages of high bandwidth sensor data without violating model assumptions.

- Conditional random fields are discriminatively trained and well suited to regularization. We can adjust the degree of smoothing in the model to minimize inaccuracies due to over-fitting to a limited training set.

Conditional random fields are defined in terms of feature functions. A key consideration when designing a model is which features should be included. In this thesis, we apply feature selection to conditional random fields for activity recognition. We specify a small collection of rules or feature prototypes using a relational feature specification language. The prototypes are automatically expanded into a large vector of candidate features and feature selection chooses a small subset of candidate features for the final model.

1.3 Contributions

The contributions of this thesis are as follows:

- We provide a tutorial introduction to conditional random fields and a discussion of CRFs from an activity recognition perspective. In particular, we focus on feature functions as a means of incorporating domain knowledge into the model.
- We present an empirical comparison between conditional random fields and hidden Markov models for activity recognition using a robot tag domain as well as synthetic data sets to highlight the properties of conditional random fields that make them well suited to activity recognition.
- We empirically compare three feature selection algorithms, grafting, ℓ_1 regularization, and a heuristic gain metric, for feature selection in conditional random fields. We examine computational efficiency and the sparsity of the models produced by the algorithms.
- We explore feature selection in a multi-robot activity recognition domain. Specifically, we introduce a relational language for succinctly specifying feature templates or prototypes. We analyze the scalability of feature selection algorithms for CRFs and discuss how to scale to extremely large sets of candidate features.
- We investigate an M-estimator, originally proposed by Jeon and Lin [42] for certain classes of ANOVA log-density estimation, to address the computational challenge of training conditional random fields on large data sets. We explore the suitability of the M-estimator for parameter estimation in conditional random fields and explore the properties of the M-estimator using a variety of synthetic and real data.

- We adapt the M-estimator for feature selection in conditional random fields. We present an empirical evaluation of the M-estimator for feature selection for a multi-robot activity recognition task.

1.4 Guide to the Thesis

The thesis is organized as follows:

Chapter 2 We present an introduction to conditional random fields from an activity recognition perspective. In particular, we emphasize the properties of conditional random fields, such as their ability to incorporate complex features of sensor data, that make them well suited for activity recognition based on robot sensor data. We provide an empirical comparison between CRFs and HMMs to highlight the properties properties of CRFs that make them well suited to robot data.

Chapter 3 We provide an empirical comparison of three feature selection algorithms for conditional random fields. We use a variety of synthetic and robot data to illustrate the performance of each algorithm according to a variety of metrics such as the sparsity of the resulting models, data efficiency, and model selection speed.

Chapter 4 We explore how well the feature selection algorithms from the previous chapter scale to large sets of candidate features and multi-agent domains. In particular, we define a relational feature specification language for multi-robot domains. We automatically construct large candidate sets of features using our relational feature specification language. We empirically explore issues such as computational efficiency and how the feature selection algorithms can be modified to allow for tractable feature selection in large models.

Chapter 5 We tackle the computational challenge of training and feature selection for conditional random fields. We investigate the use of an asymptotically consistent M-estimator, originally proposed by Jeon and Lin [42] for log-density ANOVA estimation, for fast and approximate training of conditional random fields. In particular, we compare the accuracy of the approximate model to a standard CRF and explore the trade off between training speed and model accuracy. Returning to feature selection, we explore use the M-estimator for feature selection.

Chapter 6 We give a brief overview of related work.

Chapter 7 We summarize our major findings and present future work.

Chapter 2

Conditional Random Fields for Activity Recognition

Conditional random fields are discriminatively trained models for structured classification. Activity recognition, where the observations and labels arrive in a sequence over time, is a structured domain and conditional random fields are well suited for activity recognition. We present an introduction to conditional random fields from an activity recognition perspective and compare their performance with hidden Markov models, which are also popular models for sequential classification.

2.1 Introduction

Conditional random fields (CRFs) are models for structured classification [56]. In particular, CRFs are undirected graphical models, where the structure of the graph encodes independence relationships between labels. The term *conditional* appears in the name of the model because CRFs are probabilistic models that *condition* on the observations. In other words, the structure of the graph encodes independence relationships between labels and *not* the observations. The model assumes no independence relationships between observations and, as a consequence, inference in CRFs remains tractable even when they incorporate complex and arbitrary features of the observations.

In this chapter, we describe conditional random fields from an activity recognition point of view. We begin by motivating the activity recognition problem and by highlighting the properties of conditional random fields that make them well suited to the activity recognition task. Our focus in this thesis is on sequential classification. We present a formal definition of the sequential supervised learning problem and then a concrete example in the form of a robot tag activity recognition task.

2.1.1 Activity Recognition

In order to act effectively in the presence of others, robots require information about the other actors in their environment. Often, the other agents are either neutral, e.g. pedestrians near a vacuum cleaning robot, or hostile, e.g. an opposing team in robot soccer [49], to the robot. In both situations, the other agents in the environment will not directly communicate their behaviors or actions to a robot. Neutral agents cannot be bothered to explain themselves to a robot and hostile agents deliberately withhold information. In domains where the activities or behaviors others are pertinent to making decisions and where that information is not explicitly communicated, robots must rely on activity recognition to extract the required information from their sensory data.

Mapping from noisy, continuous sensor readings to discrete activity labels is a difficult task. In particular, models that incorporate the sensor readings directly, without additional processing, tend to perform poorly. Individual sensor readings, such as an image from a video camera, are difficult to interpret in their raw form. The relevant data are present, but they are hard to leverage for classification. Instead of using the raw sensor data, we typically transform the data via feature functions. Feature functions map the sensor data into a form that is more useful for activity recognition.

Conditional random fields, which are defined in terms of feature functions, are well suited to activity recognition from robot sensor data. The relevant properties of CRFs that make them well suited for activity recognition are:

- Conditional random fields can robustly incorporate complex, non-independent features of the observations. Feature functions allow human model designers to inject their domain knowledge into the model. Because they allow for non-independent features, CRFs give model designers additional flexibility in leveraging their domain knowledge when choosing features for the model.

- Conditional random fields are discriminatively trained to model the labels alone. With robot data in particular, the observation data is complex and difficult to model. Conditional random fields do not waste modeling effort on the difficult to predict sensor data, but instead focus their modeling effort on the labels. Discriminative models tend to have lower asymptotic error rates than generative models with similar structures [78].

- Conditional random fields, because they allow for arbitrary features of the observations and because they do not require additional dependencies to model the observations, often allow us to use model topologies that allow for exact inference over the labels. In this thesis, we focus exclusively on conditional random fields with a linear chain structure (first-order Markov assumption) over the labels because it allows for fast inference and yields acceptably high classification accuracies.

Next, we formally define the sequential supervised learning problem and present an example domain that centers around the game of robot tag.

2.1.2 The Sequential Supervised Learning Problem

In the sequential supervised learning problem [23], our task is to construct a hypothesis (or classifier) h from a training set that contains a set of labeled *sequences*. Every position in a sequence has an observation vector x_t and a discrete label y_t, where t is the index into the sequence. For time series data, t represents a discrete time step in the sequence. The observation vector x_t contains one or more variables that may either be discrete or real valued. The labels y_t are drawn from a finite set of discrete values and, in our notation, we use $|y_t|$ to represent the size of the label set. We refer to all observations in a sequence as $X = \{x_1, x_2, ..., x_T\}$, where T is the length of the sequence. Similarly, we refer to the labels for a single sequence as $Y = \{y_1, y_2, ..., y_T\}$. A labeled sequence is the pair (X, Y) and we define the training set of n labeled sequences as $D = \{(X, Y)^{(i)}\}_{i=1}^{n}$. While we use T to denote the length of arbitrary sequences, it does not necessarily hold that the length of $(X, Y)^{(i)} \in D$ is equal to the length of $(X, Y)^{(j)} \in D$ when $i \neq j$. Furthermore, while h is trained on labeled sequences, the label sequence is unavailable at test time and the trained classifier h maps from X to Y. That is, given a sequence of observations X, h predicts a corresponding label sequence $\widehat{Y}$.

We make different assumptions about the distribution of the data in the sequential problem versus the classical supervised learning problem. In the classical version, we want to predict a single label y from a single observation vector x and we assume that individual observation-label pairs (x, y) in the data are independent and identically distributed (iid). In the sequential classification problem, we assume that the data are iid at the level of entire sequences. That is, all of the variables in $(X, Y)^{(i)}$ are independent of the variables in $(X, Y)^{j}$ for $i \neq j$, but we assume nothing about the relationship between the variables within a single sequence x_t and y_t. It is important to note that while we make no independence assumptions within a sequence when defining the learning problem, the models that we use to solve that learning problem will depend on such assumptions for a tractable solution.

In the next section, we present a robot tag activity recognition domain as a concrete example of the sequential supervised learning problem.

2.1.3 The Robot Tag Domain

We use a domain inspired by the children's game of Tag as a benchmark problem for activity recognition. We implemented our tag domain within the CMDragons robot soccer system [14, 16], a champion robot soccer team from the Small Size League of the RoboCup robot soccer world championship [49]. In the tag domain, three holonomic (omni-directional) robots move on a playing field. Two of the robots navigate between a series of random locations on the field. Each time they reach a target location, they randomly select a new point and navigate to that target. Rather than selecting a new target after reaching a goal location, the active robot, which we call the *seeker*, constantly updates its navigation target. Instead of choosing its target randomly, the seeker uses the current position of its closest teammate as its target. I.e. the seeker chases the closest other robot, which is why we refer to the domain as "Tag" even though the pursued robot does not actively flee from the seeker. In our domain, the seeker *tags* another robot by approaching within a threshold distance of the target (4 cm). Once a robot has been tagged, it swaps roles with the seeker and pauses in place briefly to allow the previous seeker to escape.

The two non-seeker robots are passive in the sense that once they choose a target, they navigate directly to that target without regard to the position of the seeker. The passive robots choose new points according to two schemes. In the first scheme, which we call the *hourglass* scheme, they choose successive corners of the field and move in an hourglass pattern. If the seeker lies in the quadrant that are scheduled to move into, they move to the corner of the quadrant furthest from the seeker. In the second scheme, which we call *unconstrained*, the non-seeker robots sample new targets uniformly, but with rejection, from within the playing area. They reject candidate target points that lie within a 1 meter radius of the seeker to provide a slight bias away from the seeker. However, once they select a target, they will move directly toward it, even if that entails moving closer to the seeker.

The playing area is 3.5 by 4 meters in size, the robots are 20 cm in diameter, and the seeker must approach within 4 cm of another robot in order to tag it. When a robot is first tagged, it pauses in place for 5 seconds in order to avoid immediately re-tagging the previous seeker. The robots receive sensor readings and make decisions at a frame rate of 60 hz. At

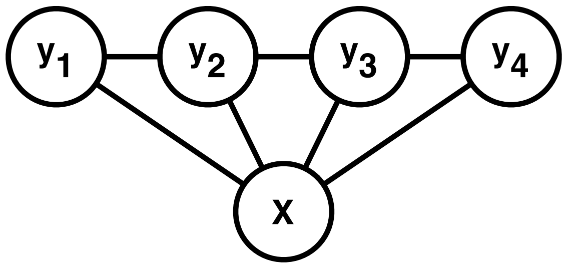


Figure 2.1: The graphical structure of a linear-chain conditional random field. The individual labels y_t form the backbone of the chain. A linear chain structure corresponds to making a first-order Markov assumption over the labels. The entire observation sequence forms a single node X, which connects to all labels, indicating that the model makes no independence assumptions between the observations from individual time steps x_t and that the feature functions we use in the model are functions of the entire observation sequence (as well as pairs of adjacent labels).

the beginning of each frame, two overhead cameras capture images of the field. A color vision system [15] uses colored markers on the tops of the robots to extract their position and orientation from the camera images. The system tracks the robots over time with an extended Kalman filter and uses the positions for real time path planning. The behaviors, which control the individual robots, are defined in terms of hierarchical state machines and specify robot actions in terms of navigation targets.

In the formal notation of the sequential supervised learning problem, the label sequence $Y = \{y_1, y_2, ..., y_T\}$ specifies which of the three robots is the seeker at each time step. An individual label y_t takes on a value from the set $\{\text{robot}_1, \text{robot}_2, \text{robot}_3\}$ to indicate that the corresponding robot is the seeker at time t. The observation sequence $X = \{x_1, x_2, ..., x_T\}$ contains the positions of the six robots at each time step. A single observation vector x_t is a 6-tuple that contains a two-dimensional Cartesian field position for each of the three robots. The field positions are measured in millimeters and the center point of the field is the origin of the plane.

2.2 Representation

In sequential classification, we map from an observation sequence X to a label sequence $\widehat{Y}$, where $\widehat{Y}$ is an approximation of the true label sequence Y. Conditional random fields model the conditional likelihood $P(Y|X)$. We use CRFs to predict a sequence of labels by computing

$$\operatorname*{argmax}_{\widehat{Y}} p(\widehat{Y}|X) \tag{2.1}$$

if we want to maximize the likelihood of the entire label sequence as a whole or

$$\operatorname*{argmax}_{\widehat{Y}} \prod_{t} p(y_t|X) \tag{2.2}$$

if we want to maximize the marginal probabilities of individual labels. In the second case, we maximize the probability that any individual prediction $\widehat{y}_t$ is correct without regard to the probability of the entire sequence $p(\widehat{Y}|X)$, which generally suffers under this second approach because $\widehat{Y}$ can include unlikely or impossible transitions between y_{t-1} and y_t.

There are two main challenges in using a probabilistic approach to sequential classification:

- We require an efficient representation for $p(Y|X)$. The number of possible label sequences $|Y|$ grows exponentially with the length of the sequences. A naive representation of the conditional probability will quickly become intractable due to the space required to store the model parameters.
- We require efficient algorithms for prediction label sequences, e.g. by solving (2.1) or (2.2). More generally, we require efficient inference algorithms to answer queries using the model.

We address the first issue, that of representation in the remainder of this section. We defer our discussion of inference until section 2.3.

2.2.1 Conditional Independence Assumptions

Undirected graphical models, such as conditional random fields, provide compact representations for complex probability distributions by leveraging conditional independence assumptions between the variables of the model [84]. I.e. we make assumptions about the structure of the data in order to create tractable representations. In the specific case of conditional random fields, we make independence assumptions over only the labels. In this thesis, we assume that the labels form a linear chain. In general, other model structures are possible, although the structure of the model has a dramatic effect on the efficiency of inference; in particular, exact inference tends to be intractable when the graph structure of the labels contains loops. The model structure also implicitly defines conditional independence relationships between the model variables, as we describe here.

Figure 2.1 shows the structure of a linear-chain CRF. The graphical structure of the model implicitly defines the conditional independence assumptions that it makes between the variables (nodes). Conditional independence obeys a simple rule in undirected graphical models. Two variables y_i and y_j are conditionally independent iff all paths in the graph between the two nodes are blocked by an observed variable [57]. In figure 2.1, the observation sequence X is always observed so all paths between labels that pass through X are blocked. It then follows that all past labels y_i s.t. $i < t$ are conditionally independent of all future labels y_j s.t. $j > t$ given the value of y_t (and X). In other words, the CRF shown in figure 2.1 makes a first-order Markov assumption over the label sequence.

The graphical structure of the model also provides a functional form for the distribution. I.e., if we write down a function for $p(Y|X)$, then that function must factor into the product of several different terms, where each term depends on a subset of the variables in the full model. The graphical structure of the model specifies how variables are grouped together to form these terms, which are called *clique potentials*.

More formally, let G be an undirected graph with edge set E and vertex set V. Then let $p(V)$ be a probability density that obeys the conditional independence assumptions encoded by G according to the rules for conditional independence in undirected graphs. The Hammersly-Clifford theorem states that $p(V)$ factors into the product of separate functions and that the functions, called clique potentials, are computed on the cliques of the graph [8, 37].

$$p(V) \propto \prod_{c \in cliques(G)} \psi(c) \tag{2.3}$$

where ψ is an arbitrary non-negative function of the variables in clique c. Concretely, for the graph structure shown in figure 2.1, the cliques of the graph contain the adjacent pairs and the entire observation sequence $c_t = (y_{t-1}, y_t, X)$ and the joint probability of all variables in the model, which we can represent, but not efficiently reason about with this particular structure, must factor as

$$p(X, Y) \propto \prod_t \psi_t(y_{t-1}, y_t, X) \tag{2.4}$$

In general, we compute a normalization constant

$$Z = \sum_V \prod_{c \in cliques(G)} \psi(c) \tag{2.5}$$

which is also referred to as the partition function, in order to obtain a proper distribution that sums to one. Computing the partition function requires that we sum over all possible assignments to the variables in V. This normalization computation is why inference in a *joint* model $p(X, Y)$ is intractable for the model structure shown in figure 2.1. To allow tractable inference, we compute the conditional probability $p(Y|X)$ rather than the joint. We can efficiently compute the normalization constant for this conditional probability with dynamic programming, as we will describe in later sections.

The Hammersley-Clifford theorem specifies a factorization for the joint probability of all variables in the model. The conditional probability $p(Y|X)$ must factor according to the clique potentials of $p(X,Y)$. This is trivially apparent from the fact that we can marginalize to compute $p(Y|X)$ from $p(X,Y)$ by substituting the known values of X and renormalizing to obtain a proper (conditional) distribution. The conditional probability of the labels given the observations in a CRF factors according to

$$p(Y|X) = \frac{1}{Z_X} \prod_t \psi_t(y_{t-1}, y_t, X) \tag{2.6}$$

The normalization constant is computed by summing over all possible label sequences Y', which is tractable for certain structures, such as linear label chains, via dynamic programming, as:

$$Z_X = \sum_{Y'} \prod_t \psi_t(y_{t-1}, y_t, X) \tag{2.7}$$

The normalization constant depends on a specific sequence of observations X because CRFs condition on observations rather than modeling them.

2.2.2 Clique Potentials

Conditional random fields represent $p(Y|X)$ as a normalized product of clique potentials. In general, for undirected graphical models, clique potentials are arbitrary non-negative functions. Conditional random fields, which are log-linear models [77], use a particular functional form for their clique functions:

$$\psi_t(y_{t-1}, y_t, X) = \exp(w^T f_t(y_{t-1}, y_t, X)) \tag{2.8}$$

where w is a real valued weight vector and f is a vector of feature functions. The weights w are the model parameters, which we estimate during training. The feature functions f_i in the feature vector f are designed to capture important domain properties, which we discuss in the next section. The form of the clique potentials, e raised to a linear function, guarantees that the clique potential is non-negative.

As defined in (2.8), clique potentials can vary across time steps. In other words, the model can use entirely different functions ψ_t for different values of t. In practice, we generally define time-invariant or *stationary* clique potentials that are the same function (that takes on different values) across time

$$\psi(t, y_{t-1}, y_t, X) = \exp(w^T f(t, y_{t-1}, y_t, X) \tag{2.9}$$

We add the index t as an argument to indicate which clique the function is being evaluated over. In our robot soccer domain, the arguments might be:

$$(t = 10, y_{t-1} = \text{robot}_1, y_t = \text{robot}_2, X) \tag{2.10}$$

and the feature functions might use t to compute values in terms of the current observation for $t = 10$, x_{10}.

2.2.3 Features

Feature functions $f_i(t, y_{t-1}, y_t, X)$ capture local properties and interactions between the variables in each clique. For example, one feature in the tag domain might indicate that robot$_1$ is within 4 cm of robot$_2$. A second feature might indicate that robot$_2$ is moving away from robot$_3$. Feature functions are arbitrary, real valued functions of their arguments that allow us to inject domain knowledge, such as the seeker in tag chases the closest robot, into the model. Concretely, using t (the current time) and X (the positions of all six robots over the entire sequence) we can write a function that evaluates to 1 if the distance between robot$_2$ and the closest other robot at time t is greater than that same distance at time $t-1$ and the current label y_t indicates that robot$_2$ is the seeker. Non-zero values of this feature suggest that robot 2 is not the seeker, because it is moving away from the closest other robot, and the corresponding weight in the model will take on a negative value during training to capture this information. To illustrate how features encode information, we describe how to create a CRF analog to a hidden Markov model [91].

A hidden Markov model is a tuple (π, T, O). The multinomial distribution $\pi = p(y_1)$ is the prior probability of the initial label. The transition matrix T encodes the transition dynamics $p(y_t|y_{t-1})$ between time steps. O is the observation model and represents the likelihood of seeing a particular observation given the current state $p(x_t|y_t)$. HMMs model the joint probability

$$p(X, Y) = p(y_1)p(x_1|y_1) \prod_{t=2}^{T} P(y_t|y_{t-1})P(x_t|y_t) \tag{2.11}$$

Joint probability in an HMM factors over the same clique potentials as the feature functions of a CRF. Neglecting the prior, we can define a CRF with the clique potentials

$$\psi(t, y_{t-1}, y_t, X) = p(y_t|y_{t-1})p(x_t|y_t) \tag{2.12}$$

where we use t as an index into X to retrieve the current observation vector x_t.

Equation (2.12) shows that we can convert an HMM into an undirected graphical model, but we have not yet shown how to define the equivalent CRF in terms of features. Beginning with the transition matrix, we add features

$$f_k = (y_{t-1} \overset{?}{=} i)(y_t \overset{?}{=} j) \tag{2.13}$$

for all possible pairs of labels i, j. We use the notation $a \overset{?}{=} b$ to indicate a binary valued test of whether a is equal to b. The expression evaluates to

$$a \overset{?}{=} b \equiv \begin{cases} 1 & \text{if } a = b \\ 0 & \text{if } a \neq b \end{cases} \tag{2.14}$$

Only a single transition feature takes on a non-zero value in any given clique potential. The single active feature can be thought of as selecting an entry out of the CPT for $p(y_t|y_{t-1})$ and the corresponding weight as encoding $\log(p(y_t|y_{t-1}))$. We say that the weight is equivalent to $\log(p)$ rather than p because, in log-linear models, we exponentiate the sum of the features when evaluating the clique potentials.

In the case of the HMM's observation model, we create features

$$f_k = (y_t \overset{?}{=} i)(x_t \overset{?}{=} j) \tag{2.15}$$

for all states i and all discrete observations j. The corresponding weights w_k encode information from the observation model and $w_k \propto \log(p(x_t|y_t))$.

Feature functions are not limited to discrete observations. For example, we encode uni-variate Gaussian distributions over continuous observations by including the sufficient statistics of a Gaussian as features:

$$\begin{aligned} \exp(w^T f) &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{(x_t - \mu)^2}{2\sigma^2}) \\ \exp(w^T f) &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp(\frac{-x_t^2}{2\sigma^2}) \exp(\frac{x_t \mu}{\sigma^2}) \exp(\frac{-\mu^2}{2\sigma^2}) \\ w^T f &= \frac{-1}{2\sigma^2} x_t^2 + \frac{\mu}{\sigma^2} x_t + \left(\frac{-\mu^2}{2\sigma^2} - log(\sqrt{2\pi\sigma^2}) \right) \end{aligned}$$

To model an observation x_t as a Gaussian, we add the following features to the model:

$$\begin{aligned} f_k &= (y_t \overset{?}{=} j) \\ f_{k+1} &= (y_t \overset{?}{=} j) x_t \\ f_{k+2} &= (y_t \overset{?}{=} j) x_t^2 \end{aligned}$$

Note that separate features $f_k = (y_t \overset{?}{=} j)$ are not necessary for different continuous variables. Including a single feature of this form in the model is sufficient and discriminative training will assign it the appropriate weight. Because f_{k+1} and f_{k+2} depend on the value of the continuous observation, the model includes copies of those features for each continuous observation.

The final piece of the HMM is the prior over initial states. We can add a fixed label START-LABEL for y_{-1} and add transition features

$$f_k(t, y_{t-1}, y_t, X) = (t \overset{?}{=} 1)(y_t \overset{?}{=} i) \tag{2.16}$$

for all states i. We can define time varying features by using t as an explicit term. However, we rarely do so because creating features that depend on specific values of t leads to larger feature vectors and harms generalization to test data.

HMMs are very simple models and, through more complex feature functions, CRFs can capture concepts that cannot be represented in an HMM. For example, CRFs can link transitions to observations. An HMM cannot capture the concept that a transition always follows a given observation. In the tag domain, if we have a function $g(t, X)$ that evaluates to 1 if robot-1 was within 4 cm of robot 2 at time $t-1$, then the feature

$$f_k(t, y_{t-1}, y_t, X) = (y_{t-1} \overset{?}{=} \text{robot}_1)(y_t \overset{?}{=} \text{robot}_2) g(t, X) \tag{2.17}$$

captures the property that if robot 1 is the seeker and approaches to within 4 cm of robot 2 then the seeker role will be transferred to robot 2. CRFs make fewer assumptions about the observations than HMMs and therefore can capture concepts that cannot be represented in an HMM. This additional flexibility is not unique to CRFs. For example, dynamic Bayesian networks [76] can create links between observations and state transitions by modeling transitions as $p(y_t|y_{t-1}, x_{t-1})$, that is, by making transitions depend on the previous observation as well as the previous state.

2.3 Inference

The second challenge of using probabilistic models for sequential classification is creating algorithms to efficiently answer queries, such as: what is the most likely sequence of labels for these observations? In this section, we describe inference in CRFs, which is the process that we use to answer such queries.

2.3.1 Computing the Normalization Constant

The first task that we consider is computing the normalization constant Z_X. The normalization constant is required for most of the inference tasks that we discuss later in the chapter. Computing Z_X is challenging because it requires a summation over all possible label sequences and the number of possible sequences grows exponentially with sequence length. In general, computing Z_X is intractable for graphs with arbitrary structures. For tree-structured graphs, including the linear chains that we consider, there is an efficient dynamic programming algorithm for computing Z_X, which we describe here for the linear chain case.

To compute the partition function, we sum the product of the clique potentials for each possible assignment to the label sequence Y. For convenience, we assume that the first entry in the sequence at $t = -1$ has a fixed label $y_{-1} = \textsc{Start-Label}$. We return to the notation of clique potentials as $\psi(t, y_{t-1}, y_t, X)$ rather than expanding them to the $\exp(w^T f)$ form for the sake of brevity. But the clique potentials retain their log-linear form.

To compute the partition function, we must compute:

$$Z_X = \sum_Y \prod_{t=1}^{T} \psi_t(y_{t-1}, y_t, X) \tag{2.18}$$

As a concrete example, let the length of the sequence be $T = 5$. In this example then:

$$Z_X = \sum_{y_1}\sum_{y_2}\sum_{y_3}\sum_{y_4}\sum_{y_5} \psi_1(y_0, y_1, X)\psi_2(y_1, y_2, X)\psi_3(y_2, y_3, X)\psi_4(y_3, y_4, X)\psi_5(y_4, y_5, X) \tag{2.19}$$

If inference is going to scale, we need an algorithm to evaluate equation (2.19) with lower complexity than $O(|y_t|^T)$, which is the complexity of the naive calculation.

Each term in the expanded equation contains two labels that are local to its portion of the graph. However, equation (2.19) recomputes every ψ term when any y_t changes, even though many ψ terms are unaffected by that local change. For example, ψ_5 is recomputed each time y_1 takes on a different value even though ψ_5 does not depend on y_1. In the remainder of this section, we describe how to restructure the computation to eliminate this inefficiency. In the general case, the procedure that we describe is known as variable elimination [1], bucket elimination [20], or the sum-product algorithm [53]. Our particular derivation for linear-chain CRFs is equivalent to the backward pass of the forward-backward algorithm for HMMs [91] and similar to the discussion in [101].

The general rule that we follow is to "push sums into products" to eliminate redundant computation. For example, pushing σ_{y_5} into the products of (2.19) yields

$$Z_X = \sum_{y_1} \sum_{y_2} \sum_{y_3} \sum_{y_4} \psi_1(y_0, y_1, X)\psi_2(y_1, y_2, X)\psi_3(y_2, y_3, X)\psi_4(y_3, y_4, X) \sum_{y_5} \psi_5(y_4, y_5, X) \tag{2.20}$$

We no longer evaluate ψ_1 through ψ_4 for different assignments to y_5.

We pushed σ_{y_5} past the products over ψ_1 to ψ_4 because they do not depend on y_5. Furthermore, because ψ_5 does not depend on y_1 to y_3, we cache the summation over ψ_5 for reuse. The potential ψ_5 depends on y_4, so we cached $|y_t|$ different summations of ψ_5, with y_4 as the index into the cache.

$$\beta_4(y_4) = \sum_{y_5} \psi_5(y_4, y_5, X) \tag{2.21}$$

We use β in our notation for the vector of cached values after [91]. Substituting β_4 into the original expression *eliminates* variable y_5, which is why one name for this algorithm is *variable elimination*.

$$Z_X = \sum_{y_1} \sum_{y_2} \sum_{y_3} \sum_{y_4} \psi_1(y_0, y_1, X)\psi_2(y_1, y_2, X)\psi_3(y_2, y_3, X)\psi_4(y_3, y_4, X)\beta_4(y_4) \tag{2.22}$$

Caching β_4 reduces the amount of required computation to $|y_t|^{T-1} + |y_t|^2$ operations. The $|y_t|^2$ term is from storing $|y_t|$ values in β_4, one for each possible assignment of y_4. The cached values are computed by summing over y_5.

We eliminate y_4 in the same way by pushing σ_{y_4} past products that do not depend on y_4:

$$Z_X = \sum_{y_1} \sum_{y_2} \sum_{y_3} \psi_1(y_0, y_1, X)\psi_2(y_1, y_2, X)\psi_3(y_2, y_3, X) \sum_{y_4} \psi_4(y_3, y_4, X)\beta_4(y_4) \tag{2.23}$$

We construct a vector of cached values indexed by the non-eliminated variables from the mixed term $\psi_4(y_3, y_4, X)$, i.e. y_3 is the index for β_3:

$$\beta_3(y_3) = \sum_{y_4} \psi_4(y_3, y_4, X)\beta_4(y_4) \tag{2.24}$$

Note that we use cached values of β_4 rather than computing

$$\beta_3(y_3) = \sum_{y_4} \psi_4(y_3, y_4, X) \sum_{y_5} \psi_5(y_4, y_5, X) \tag{2.25}$$

Reuse of cached values is the source of our increased efficiency. Reusing previous values of β yields an efficient dynamic programming algorithm. Eliminating y_4 while computing Z_X

yields a computational cost of $|y_t|^{T-2}+2|y_t|^2$ operations. Each successive variable elimination reduces the exponent in the first term by one at the cost of $|y_t|^2$ operations.

We repeat the procedure for the remaining variables:

$$Z_X = \sum_{y_1} \psi_1(y_0, y_1, X) \sum_{y_2} \psi_2(y_1, y_2, X) \sum_{y_3} \psi_3(y_2, y_3, X)\beta_3(y_3) \tag{2.26}$$

$$\beta_2(y_2) = \sum_{y_3} \psi_3(y_2, y_3, X)\beta_3(y_3) \tag{2.27}$$

$$Z_X = \sum_{y_1} \psi_1(y_0, y_1, X) \sum_{y_2} \psi_2(y_1, y_2, X)\beta_2(y_2) \tag{2.28}$$

$$\beta_1(y_1) = \sum_{y_2} \psi_2(y_1, y_2, X)\beta_2(y_2) \tag{2.29}$$

$$Z_X = \sum_{y_1} \psi_1(y_0, y_1, X)\beta_1(y_1) \tag{2.30}$$

We have improved the time complexity for computing Z_X from $O(|y_t|^T)$ operations to $O(T|y_t|^2)$, which is linear in the length of the sequence.

We can write a succinct definition of β using induction:

$$\beta_T(y_T) = 1 \tag{2.31}$$

$$\beta_t(y_t) = \sum_{y_{t+1}} \psi_{t+1}(y_t, y_{t+1}, X)\beta_{t+1}(y_{t+1}) \tag{2.32}$$

where we interpret the meaning of entries as sums over label suffixes:

$$\beta_t(y_t) = \sum_{Y_{[t+1..T]}} \psi_{t+1}(y_t, y_{t+1}, X) \prod_{t'=t+2}^{T} \psi_{t'}(y_{t'-1}, y_{t'}, X) \tag{2.33}$$

Note that the summation in (2.33) includes y_{t+1} as well; we break out the ψ_{t+1} term because there is no sum over y_t.

2.3.2 Marginal Probabilities of Labels

When we predict labels from a sequence of observations, we are usually interested in either the most likely assignment to individual labels y_t, i.e. $\text{argmax}_{y_t}\, p(y_t|X)$, or the most likely assignment of labels for the entire sequence Y, i.e. $\text{argmax}\, Y p(Y|X)$. In this section, we describe how to compute $p(y_t|X)$ using the the backward style of variable elimination described in the previous section as well as a forward style of variable elimination that we describe below. The algorithms described here can be used to compute $\text{argmax}_Y\, p(Y|X)$ as well, using the Viterbi or max-product algorithm, which Rabiner describes in detail in [91].

To compute the marginal probability of a single label, we compute a sum over many possible label sequences:

$$P(y_t = \text{label}|X) = \sum_{\tilde{Y}} P(\tilde{Y}|X) \tag{2.34}$$

where $\tilde{Y}$ is the label sequence that contains all labels y_1 to y_T except for y_t, which is fixed at label. We consider a concrete case for a sequence of length $T = 5$ and where we compute the marginal probability $p(y_3 = \text{label}|X)$

$$\begin{aligned} P(y_3 = \text{label}|X) = \frac{1}{Z_X} \sum_{y_1} \psi_1(y_0, y_1, X) \sum_{y_2} \psi_2(y_1, y_2, X) \\ \psi_3(y_2, \text{label}, X) \sum_{y_4} \psi_4(\text{label}, y_4, X) \sum_{y_5} \psi_5(y_4, y_5, X) \end{aligned} \tag{2.35}$$

Noting that the right portion of this expression is identical to β, we substitute β to simplify the expression.

$$P(y_3 = \text{label}|X) = \frac{1}{Z_X} \sum_{y_1} \psi_1(y_0, y_1, X) \sum_{y_2} \psi_2(y_1, y_2, X) \psi_3(y_2, label, X) \beta_3(label) \tag{2.36}$$

Our cached values of β represent a sum over label suffixes and allow us to eliminate all of the terms that fall after y_3 in the sequence. We can use variable elimination, starting from the front of the sequence rather than the end, to create an analogous term that represents a sum over all possible label prefixes.

Forward elimination is virtually identical to the backward case. The only difference is due to the inclusion of the start state y_0. Taking this difference into account, the induction for the forward elimination order is:

$$\alpha_1(y_1) = \psi_1(y_0, y_1, X) \tag{2.37}$$

$$\alpha_t(y_t) = \sum_{y_{t-1}} \alpha_{t-1}(y_{t-1}) \psi_t(y_{t-1}, y_t, X) \tag{2.38}$$

The cached values of α_t represent a sum over label prefixes:

$$\alpha_t(y_t) = \sum_{Y_{[1..t-1]}} \left(\prod_{t'=1}^{t-1} \psi_{t'}(y_{t'-1}, y_{t'}, X) \right) \psi_t(y_{t-1}, y_t, X) \tag{2.39}$$

When computing α_t, we sum over all labels that fall before t. Substituting α into equation (2.36) yields a simple expression for the marginal probability of a single label

$$P(y_3 = \text{label}|X) = \frac{\alpha_3(\text{label})\beta_3(\text{label})}{Z_X} \tag{2.40}$$

in the specific case of our example and in general

$$P(y_t|X) = \frac{\alpha_t(y_t)\beta_t(y_t)}{Z_X} \tag{2.41}$$

Once we cache values of α and β, which requires $O(T|y_t|^2)$ time, computing individual marginal probabilities only requires a $O(1)$ look up and division.

2.4 Parameter Estimation

To date, we have assumed that the values w_i in the weight vector w are known. Our ability to evaluate

$$\psi(t, y_{t-1}, y_t, X) = \exp(w^T f(t, y_{t-1}, y_t, X)) \tag{2.42}$$

depends on the availability of w. In this section, we describe how to train the model by maximizing the conditional likelihood $p(Y|X)$ of the training data, i.e., we describe how to compute the maximum likelihood estimate (MLE) of the model parameters given a labeled training set.

2.4.1 Computing the log likelihood and its gradient

To maximize $p(Y|X)$, we must first be able to compute $p(Y|X)$. For convenience, we will work with the log likelihood $\ell(Y|X)$ rather than the likelihood itself for numerical reasons. Defining our optimization in terms of the likelihood, we have

$$\operatorname*{argmax}_w p(Y|X) = \operatorname*{argmax}_w \frac{1}{Z_X} \exp(w^T F(X,Y)) \tag{2.43}$$

$$Z_X = \sum_{Y'} \exp(w^T F(X, Y')) \tag{2.44}$$

$$F(X,Y) = \sum_{t=1}^{T} f(t, y_{t-1}, y_t, X) \tag{2.45}$$

We define the new term $F(X, Y)$ to represent the sum of the feature vectors over time. The identity $\exp(a)exp(b) = \exp(a + b)$ allows us to make this change. We take the logarithm of (2.43) to get the log-likelihood

$$\ell(Y|X) = w^T F(X, Y) - \log(Z_X) \tag{2.46}$$

Both of the objective functions in (2.43) and (2.46) are convex functions. In general, convex optimization [11] is a tractable and well studied problem. In particular, given a convex objective function and its gradient, there are efficient algorithms, such as conjugate gradient [87] and quasi-Newton methods such as L-BFGS [70], that provide numerical methods for optimization. These algorithms work well for parameter estimation in conditional random fields. In particular, Wallach describes the empirical performance of conjugate gradient [115] and Sha and Pereira demonstrate that L-BFGS, a quasi-Newton method, offers even better performance [97]. More recent work on stochastic gradient descent methods has offered even higher performance, e.g. [114].

Computing the first term of the objective function in (2.46) is trivial. In the previous section, we provided an efficient algorithm for computing Z_X. The procedure for computing $\log(Z_X)$ is virtually identical, with only minor modifications to account for the transition to log-domain arithmetic; we must perform the entire summation for Z_X in the log-domain to avoid numerical overflow rather than taking the logarithm after the fact. With an objective function in hand, we require the gradient for that objective function:

$$\begin{aligned}
\frac{\partial \ell}{\partial w_i} w^T F(X, Y) - \log(Z_X) &= F_i(X, Y) - \frac{1}{Z_X} \frac{\partial \ell}{\partial w_i} Z_X && (2.47)\\
&= F_i(X, Y) - \frac{1}{Z_X} \frac{\partial \ell}{\partial w_i} \sum_{Y'} \exp(w^T F(X, Y')) && (2.48)\\
&= F_i(X, Y) - \frac{1}{Z_X} \sum_{Y'} \exp(w^T F(X, Y')) F_i(X, Y') && (2.49)\\
&= F_i(X, Y) - \sum_{Y'} p(Y'|X) F_i(X, Y') && (2.50)
\end{aligned}$$

The first term in the gradient is the empirical sum of the features over the training set. The second term is the expected value of each feature under the model. These two terms are equal for w^*, the weight vector that maximizes $\ell(Y|X)$. Computing the first term is a trivial. Computing the second term requires the same dynamic programming techniques we use to compute Z_X, which we describe below.

Computing the gradient requires evaluating the feature expectations under the model

$$\sum_{Y'} p(Y'|X) F_i(X, Y') \tag{2.51}$$

Expanding $p(Y'|X)$ and F_i yields:

$$\sum_{Y'} \left(\frac{1}{Z_X} \prod_{t=1}^{T} \exp(w^T f(t, Y'[t-1], Y'[t], X)) \right) \left(\sum_{t'=1}^{T} f(t', Y'[t'-1], Y'[t'], X) \right) \tag{2.52}$$

We rearrange this expression to get a form similar to our inference problems in section 2.3, where we pushed sums into products:

$$\frac{1}{Z_X} \sum_{t=1}^{T} \sum_{Y'} f(t, y't-1, y'_t, X) \prod_{t'=1}^{T} \exp(w^T f(t', y'_{t'-1}, y'_{t'}, X)) \tag{2.53}$$

The arguments of the feature functions factor over the cliques of the graph, meaning that we can exploit the structure of the problem to efficiently compute the summation. Breaking apart the sum over Y' to isolate the edge (y'_{t-1}, y'_t) yields:

$$\frac{1}{Z_X} \sum_{t=1}^{T} \sum_{y'_{t-1}} \sum_{y'_t} f(t, y'_{t-1}, y'_t, X) \exp(w^T f(t, y'_{t-1}, y'_t, X))$$
$$\sum_{Y'_{[1..t-2]}} \prod_{t'=1}^{t-1} \exp(w^T f(t', y'_{t'-1}, y'_{t'}, X)) \sum_{Y'_{[t+1..T]}} \prod_{t''=t+1}^{T} \exp(w^T f(t'', y'_{t''-1}, y'_{t''}, X) \tag{2.54}$$

We can substitute cached values from α and β into this expression to eliminate the summation over the bulk of the labels:

$$\frac{1}{Z_X} \sum_{t=1}^{T} \sum_{y'_{t-1}} \sum_{y'_t} f(t, y'_{t-1}, y'_t, X) \exp(w^T f(t, y'_{t-1}, y'_t, X)) \alpha_{t-1}(y'_{t-1}) \beta_t(y'_t) \tag{2.55}$$

The resulting computation allows us to efficiently compute the gradient of the conditional log likelihood.

2.4.2 Training with multiple sequences

Up until now, we have assumed a single sequence when talking about computing $p(Y|X)$ and $\ell(Y|X)$. In practice, the training set often contains many sequences. Because we assume that the sequences are iid, this does not complicate our algorithms. The combined conditional log likelihood of a set of sequences is:

$$\sum_{i=1}^{n} \ell(Y^{(i)}|X^{(i)}) \tag{2.56}$$

Similarly, the over all gradient can be computed by summing the gradient contributions of the individual sequences.

2.4.3 Regularization

Training conditional random fields by maximizing $\ell(Y|X)$ tends to over-fit the model parameters to the training data. Often, classification accuracy is improved by adding a regularization or smoothing term to the objective function. The regularization term is a penalty function computed over the model parameters. Large parameter values result in larger penalties. In this section, we consider two penalty functions the ℓ_1 norm and the ℓ_2 norm of the weight vector. Penalizing by the ℓ_2 norm does not materially affect the optimization. Applying an ℓ_1 penalty creates a non-differentiable objective function and complicates training, although the problem remains convex and the solution is tractable. We begin with our discussion of the ℓ_2 penalty, as it is the easier method to use for smoothing.

ℓ_2 Regularization

When training under an ℓ_2 penalty, we optimize

$$w_\lambda^* = \underset{w}{\operatorname{argmax}}\ \ell(Y|X;w) - \frac{1}{2}\lambda w^T w \tag{2.57}$$

We explicitly include w as a parameter of $\ell(Y|X)$ to indicate that the likelihood of the data is computed according the the parameter values of w. The constant λ controls the degree of smoothing due to the penalty. Higher values of λ result in more smoothing and $\lambda = 0$ corresponds to the original objective function. Both the penalty term and $\ell(Y|X;w)$ are convex functions of w and therefore their sum is convex as well. We can compute the gradient of the penalty term ($\frac{\partial \ell}{\partial w_i} = -\lambda w_i$) and we estimate w^* by numerical optimization exactly as before. We choose a value for λ by using a holdout set, cross-validation, or similar technique.

In equation 2.57, we used $w^T w$ as shorthand for $\sum_i w_i^2$. In practice, we do not always sum over all of the weights. We often will omit the weights corresponding to features of the form ($y_t \overset{?}{=}$label) from the penalty. The rational for omitting these weights is that generalized linear models represent conditional probabilities as $P(y|x) \propto \exp(ax + b)$, where $ax + b$ is the canonical equation for a line. The weights that correspond to these features are analogous to the intercept term b. We penalize the parameters in a that are combined with the observed evidence, but we usually do not penalize the intercept terms.

ℓ_1 Regularization

When training under an ℓ_1 penalty, we optimize

$$w_\lambda^* = \underset{w}{\operatorname{argmax}}\ \ell(Y|X;w) - \lambda \sum_i |w_i|, \tag{2.58}$$

As in the ℓ_2 case, λ is a positive scalar value that controls the degree of smoothing and the penalized objective function is convex. However, the penalty function is not differentiable at zero.

While more difficult than training with an ℓ_2 penalty, using an ℓ_1 penalty produces sparse models where many of the parameters exactly equal to zero [39]. For intuition, consider the weight $w_i = 0$. Under an ℓ_2 penalty, the partial derivative of the penalty is λw_i. To a first order approximation, the change in the penalty is zero for moving w_i away from zero. With an ℓ_1 penalty, the relevant partial is $\pm\lambda$, so the change in the penalty is proportional to λ. Under an ℓ_2 penalty, a small non-zero derivative in the *unpenalized* objective function will move w_i away from zero. In the ℓ_1 case, λ serves as a threshold and prevents w_i from becoming non-zero to buy small improvements in the unpenalized objective function.

Training under an ℓ_1 penalty for feature selection is a well studied technique for linear regression and it is called the LASSO [106]. In the linear regression case, there is an efficient algorithm to compute w_λ^* in the model across all settings of λ – the regularization path of the model – ranging from λ sufficiently large to eliminate all features to $\lambda = 0$ where the weights are not penalized [26]. In the generalized linear model setting, which includes CRFs, there is no such efficient algorithm for testing all relevant settings of λ. In fact, it is challenging to train the model with a single, fixed λ.

Despite the difficulty, there has been significant recent work on training generalized linear models under an ℓ_1 penalty. [82] use a predictor-corrector method to recover the full regularization path, although potentially at a high computational cost. [58] consider the specialized case of logistic regression. They exploit the existence of efficient algorithms for performing ℓ_1 regularized linear regression for efficiently training in the logistic regression case. [50] also consider logistic regression and they use an interior point method and warm start technique to sample many different settings of λ along the regularization path. Their warm start technique starts with a large value of λ and performs a series of relaxations where the model is initialized with the parameters from the previous iteration. Others divide the search space into orthants, regions over which the weights do not change their sign and apply conventional optimization techniques within the constrained subspace [3, 46]. Notably, [3] present a modification of L-BFGS that performs as well or better than L-BFGS on the ℓ_2 penalized optimization problem.

We describe a reparameterization that transforms the unconstrained optimization in (2.58) into a constrained problem with a well defined first derivative as done by, for example, [45]. Combined with the warm start technique used by [50], this reparameterization allows us to train CRFs with an ℓ_1 penalty [109]. In later experiments, we use the orthant-wise optimization technique of Andrew and Gao [3] because it offers superior performance. We present a less efficient method here to avoid discussing the internals of the L-BFGS algorithm, which are modified with Andrew and Gao's technique.

We define a new objective function by reparameterizing the original function in terms of two vectors, w^+ and w^-, that are related to w from equation (2.58) by $w = w^+ - w^-$. We constrain the entries of w^+ and w^- to be non-negative, yielding the new optimization problem:

$$\underset{w^+,w^-}{\text{argmax}}\ \ell(Y|X) - \lambda \sum_i w_i^+ - \lambda \sum_i w_i^-, \tag{2.59}$$

$$s.t.\ w_i^+ \in w^+ \geq 0, w_i^- \in w^- \geq 0 \tag{2.60}$$

We solve this constrained optimization problem using projected conjugate gradient. Briefly, algorithms such as conjugate gradient optimize through a series of minimizations along a line. Each minimization begins with an initial point x_0 and a search direction d. The objective function is evaluated and candidate points x, where $x = x_0 + \alpha d$, $\alpha \geq 0$, and a line search algorithm is used to choose the optimal value of α. The key idea of projected conjugate gradient is that candidate points x are projected back into the feasible region before being evaluated. In our case, that amounts to computing x_p, such that $x_p = max(0, x)$. This type of projected optimization algorithm has long been used to solve constrained optimization problems with simple inequality constraints [7] and is an efficient method for training CRFs with an ℓ_1 penalty.

2.5 Implementation

We have omitted a number of practical details that must be considered to build a working conditional random field implementation. In this section, we list a few important considerations regarding log-domain arithmetic, efficiency, and we provide pseudo-code for the most important algorithms.

2.5.1 Log-domain addition

Computing the product of clique potentials over long sequences results in values that cannot be represented with available floating point data types. To sidestep this issue, we store and operate on the logarithms of the values of interest. Probabilities, which are never negative, fit well with this representation.

Multiplication between log-domain values is trivial due to the identity $\log(ab) = \log(a) + \log(b)$. Adding log-domain values with minimum loss of precision requires more care. The naive method of computing $\log(a+b)$ starting from $\bar{a} = \log(a)$ and $\bar{b} = \log(b)$ is by computing $\log(\exp(\bar{a}) + \exp(\bar{b}))$. The naive method is unacceptable because computing one or both of a and b might cause overflow or underflow. Instead, we use an alternate method that avoids explicitly computing a or b. We rearrange the naive computation as follows:

$$\begin{aligned}
\log(a+b) &= \log(\exp(\bar{a}) + \exp(\bar{b})) && (2.61)\\
&= \log\left(\exp(\bar{a})\left(1 + \frac{\exp(\bar{b})}{\exp(\bar{a})}\right)\right) && (2.62)\\
&= \log(\exp(\bar{a})(1 + \exp(\bar{b} - \bar{a}))) && (2.63)\\
&= \bar{a} + \log(1 + \exp(\bar{b} - \bar{a})) && (2.64)
\end{aligned}$$

Rearrangement allows us to compute $\exp(\bar{b} - \bar{a})$ in place of $\exp(\bar{a})$ and $\exp(\bar{b})$. In the case where $\bar{a} > \bar{b}$, this minimizes the loss of numeric precision because $\exp(\bar{b} - \bar{a})$ is small. In the case where $\bar{a} < \bar{b}$, we swap the values before performing the addition as in the pseudo-code implementation in algorithm Log-Add.

```
Log-Add($\bar{a}, \bar{b}$)
1  if $\bar{a} = -\infty \wedge \bar{b} = -\infty$
2      then return $-\infty$
3  if $\bar{a} > \bar{b}$
4      then return $\bar{a} + \log(1 + \exp(\bar{b} - \bar{a}))$
5      else  return $\bar{b} + \log(1 + \exp(\bar{a} - \bar{b}))$
```

Log-Add takes two log-domain values as its arguments and returns their sum, also as a log-domain value, with minimum loss of precision. The algorithm tests to verify that either a or b is non-zero on line 1 to prevent a subtraction between two infinite values, which may result in an undefined return value depending on details of the hardware and floating point settings. The remainder of the algorithm is a straight forward implementation of equation (2.64) that minimizes the loss of precisions by subtracting off the larger of $\bar{a}$ and $\bar{b}$ as appropriate.

2.5.2 The Forward-Backward Algorithm

We provided an inductive definition of α and β in section 2.3. Here we present pseudo-code versions of the log-domain computation. Together, these two methods are known as the forward-backward algorithm [91].

```
FORWARD-PASS($X, w$)
 1  for $y_t \leftarrow 1$ to NUM-LABELS
 2      do $t \leftarrow 1$
 3         $\bar{\alpha}[y_t][t] \leftarrow w^T f(t, \text{START-LABEL}, y_t, X)$
 4  for $t \leftarrow 2$ to length($X$)
 5      do for $y_t \leftarrow 1$ to NUM-LABELS
 6             do accumulator $\leftarrow -\infty$
 7                for $y_{t-1} \leftarrow 1$ to NUM-LABELS
 8                    do edge-contrib $\leftarrow w^T f(t, y_{t-1}, y_t, X) + \bar{\alpha}[y_{t-1}][t-1]$
 9                       accumulator $\leftarrow$ LOG-ADD(accumulator, edge-contrib)
10                $\bar{\alpha}[y_t][t] \leftarrow$ accumulator
11  return $\bar{\alpha}$
```

FORWARD-PASS takes a sequence of observation vectors and a vector of weights and produces the matrix $\bar{\alpha}$, where $\bar{\alpha}[y_t][t] = \log(\alpha_t(y_t))$. FORWARD-PASS fills in the values in $\bar{\alpha}$ using the induction given in equation (2.38). Lines 1–3 initialize the induction and handle the special case of the first clique potential, which includes a special start state. On line 6, we clear our accumulator using $infty$ rather than 0 because we are working with log-domain values. The pseudo-code for the backward pass is a close analog to the code for the forward pass.

```
BACKWARD-PASS($X, w$)
 1  for $y_t \leftarrow 1$ to NUM-LABELS
 2      do $\bar{\beta}[y_t][\text{length}(X)] = 0$
 3  for $t \leftarrow \text{length}(X) - 1$ to 1
 4      do for $y_t \leftarrow 1$ to NUM-LABELS
 5             do accumulator $\leftarrow -\infty$
 6                for $y_{t+1} \leftarrow 1$ to NUM-LABELS
 7                    do edge-contrib $\leftarrow w^T f(t+1, y_t, y_{t+1}, X) + \bar{\beta}[y_{t+1}][t+1]$
 8                       accumulator $\leftarrow$ LOG-ADD(accumulator, edge-contrib)
 9                $\bar{\beta}[y_t][t] \leftarrow$ accumulator
10  return $\bar{\beta}$
```

BACKWARD-PASS takes a sequence of observation vectors and a vector of weights as arguments and produces a matrix $\bar{\beta}$, where $\bar{\beta}[y_t][t] = \log(\beta_t(y_t))$. Lines 1–2 initialize the induction. We use $\log(1) = 0$ in the initialization rather than 1 because we are using log-domain values.

2.5.3 Computing $\ell(Y|X)$

The partition function Z_X is required to compute the log-likelihood. Computing $\log(Z_X)$ from $\bar{\alpha}_t$ or $\bar{\beta}_t$ is trivial, as shown below.

```
CALC-LOG-Z($\bar{\alpha}$)
1  $\bar{z} \leftarrow -\infty$
2  for $y_T \leftarrow 1$ to NUM-LABELS
3      do $\bar{z} \leftarrow$ LOG-ADD($\bar{z}, \bar{\alpha}[y_T][T]$)
4  return $\bar{z}$
```

CALC-LOG-Z computes the log-domain version of the normalization constant Z_X from the vector $\bar{\alpha}_T$ with a sum over log-domain values. With the normalization constant in hand, it is straightforward to compute the log-likelihood of a sequence.

```
CALC-LOG-LIKE($X, Y, w$)
1  feature-sums[1..NUM-FEATURES] $\leftarrow 0$
2  for $t \leftarrow 1$ to $T$
3      do feature-sums $\leftarrow$ feature-sums $+ f(t, Y[t-1], Y[t], X)$
4  $\bar{z} \leftarrow$ CALC-LOG-Z(FORWARD-PASS($X, w$))
5  return $w^T$ feature-sums $- \bar{z}$
```

CALC-LOG-LIKE computes the conditional log likelihood for a single sequence (X, Y). Lines 1–3 compute the empirical sums of the features on the sequence, line 4 computes the normalization constant, and line 5 combines these two pieces to produce the actual log likelihood. In terms of computational cost, the call to FORWARD-PASS is the most expensive portion of the code. The feature sums, which do not depend on w, can be precomputed and cached. And the cost of taking an inner product and subtracting a scalar value in line 5 is minor when compared to the $O(T|y_t|^2)$ cost of FORWARD-PASS.

```
Calc-Gradient($X, Y, w$)
 1  $gradient[1..\text{Num-Features}] \leftarrow 0$
 2  for $t \leftarrow 1$ to $T$
 3      do $gradient \leftarrow gradient + f(t, Y[t-1], Y[t], X)$
 4  $\bar{\alpha} \leftarrow$ Forward-Pass($X, w$)
 5  $\bar{\beta} \leftarrow$ Backward-Pass($X, w$)
 6  $\bar{z} \leftarrow$ Calc-Log-Z($\bar{\alpha}$)
 7  $t \leftarrow 1$
 8  for $y_t \leftarrow 1$ to Num-Labels
 9      do $log\text{-}p \leftarrow w^T f(t, \text{Start-Label}, y_t, X) + \bar{\beta}[y_t][t] - \bar{z}$
10         $p \leftarrow \exp(log\text{-}p)$
11         $gradient \leftarrow gradient - p \cdot f(t, \text{Start-Label}, y_t, X)$
12  for $t \leftarrow 2$ to Length($X$)
13      do for $y_t \leftarrow 1$ to Num-Labels
14             do for $y_{t-1} \leftarrow$ to Num-Labels
15                    do $log\text{-}p \leftarrow \bar{\alpha}[y_{t-1}][t-1] + w^T f(t, y_{t-1}, y_t, X) + \bar{\beta}[y_t][t] - \bar{z}$
16                       $p \leftarrow \exp(log\text{-}p)$
17                       $gradient \leftarrow gradient - p \cdot f(t, y_{t-1}, y_t, X)$
18  return $gradient$
```

Calc-Gradient begins by computing the empirical sum of the features over the training set in lines 1–3. As in Calc-Log-Like, the feature sums can be cached between iterations to avoid wasted computation. Lines 4–5, which compute $\bar{\alpha}$ and $\bar{\beta}$ are computationally expensive. This cost can be reduced by sharing cached values of $\bar{\alpha}$ between Calc-Log-Like and Calc-Gradient. Lines 7–11 handle the special case of the gradient calculation for $t = 1$. Special handling is needed to handle $y_0 =$ Start-Label and because $\bar{\alpha}$ is not used when computing *log-p* for the first time step. Lines 12–17 repeat the same basic pattern as the special case for $t = 1$, except that $\bar{\alpha}$ is used to compute *log-p* and the loop in line 14 sums over all possible labels for the previous time step.

2.5.4 Optimization

There are a number of ways to optimize a CRF implementation. The central ideas that we use are that feature vectors are often sparse and that computations can be cached for reuse. Features generally use an indicator function to select a particular combination of labels and, potentially, observations, which results in a feature vector that contains many values that are exactly equal to zero. The sparsity of the feature vector allows for fast operations on the features.

We perform two key operations with the feature vector. We take the inner product between f and another vector, typically w. We also add a scaled version of f into an accumulator vector, for example, during the gradient calculation. The sparsity of the feature vector improves the efficiency of these operations by allowing us to eliminate multiplications by zero when computing the inner product and by skipping the zero terms when performing the scaled accumulate.

The sparsity of the feature vector also facilitates caching feature evaluations. In Forward-Pass and Backward-Pass, we evaluate $f(t, y_{t-1}, y_t, X)$ for all possible combinations of y_{t-1} and y_t. However, these values do not depend on w and consequently can be computed once at the start of training and cached. In the worst case, this cache requires $O(T|y_t|^2|f|)$ storage. A sparse feature vector f allows us to cache only non-zero features.

When there is not sufficient memory to cache full feature vectors, caching the inner product between the features and the weight vector can improve performance. The inner product is required in Forward-Pass, Backward-Pass, as well as in Calc-Gradient. However, because the inner product depends on w, this cache must be recomputed during each iteration of the optimization. But, since only a scalar value needs to be cached, the required memory is $O(T|y_t|^2)$, which makes this optimization applicable when there is insufficient memory to store entire feature vectors.

The final optimization that we mention is that the feature sums over time F also do not depend on w and therefore can be computed once at the start of training and reused. In practice, it is worth noting that the contributions due to $F^{(i)}$ from different sequences $i \in \{1...n\}$ are additive. When computing $\ell(Y^{(i)}|X^{(i)}$, we add in a term for $w^T F^{(i)}$ for each sequence. And in the gradient computation, we add in a term of $F^{(i)}$ directly. Rather than caching many individual feature sums, the sums themselves $F^{(i)}$ can be added together to form a single vector. This can result in substantial memory savings in situations where the feature vector is large and there are many sequences.

2.6 Properties of Conditional Random Fields

In this section, we use synthetic data to illustrate the properties of conditional random fields that make them well suited for the activity recognition problem. In particular, we compare the accuracy of generative versus discriminative models in experiments with a CRF, which contains features that make it equivalent to a discriminatively trained HMM, and an HMM with MLE parameters. We also show that conditional random fields are robust to non-independent observations and illustrate how to create features that link state transitions to particular observations.

2.6.1 Generative versus Discriminative Models

Ng and Jordan [78] compared the classification accuracy of a naive Bayes classifier to logistic regression on data sets from the UCI Machine Learning Repository [4] as the size of the training set was varied. They found that naive Bayes outperformed logistic regression for very small training sets, but logistic regression had a lower asymptotic error rate as the size of the training set increased. We performed the same comparison for the sequential classification case using HMMs and CRFs. We created an HMM that makes a naive Bayes style independence assumption over entries in the observation vector x_t. We also created a CRF with a set of features that gave it the same representational power as the HMM. The CRF was effectively a discriminatively trained HMM.

We created synthetic data using the Pima Indians Diabetes data set from the UCI repository. The data set contained 768 instances with binary labels and 8 continuous attributes. Each instance contains data from a single person, so there is no notion of sequence in the data. We created sequential data using a first-order Markov transition function as follows. We broke the full data set into training and test sets, each containing half of the instances. We subdivided the training and test sets according to the labels of each instances to produce four groups of instances: positive training examples, negative training examples, positive test examples, and negative test examples. We generated a chain of labels from the first-order Markov chain. We filled in observations vectors for each label in the sequence by randomly choosing an example from the appropriate sample pool. When drawing observations for a training sequence with a positive label, we copied observations from a data point in the positive test pool. We drew uniformly (with replacement) from each pool.

Figure 2.2 shows average error rates and 95% confidence intervals for CRFs and HMMs on synthetic sequences. We generated 1,000 training and test sequences for each training sequence length that we considered between 10 and 500 time steps. The test sequences had a fixed length of 500; only the length of the training sequences was varied. The results show the same trend that [78] described where the generative model (the HMM) outperforms the discriminative model for small training sets (short sequences) and the discriminative model (the CRF) outperforms the generative model as the length of the amount of training data grows sufficiently large. The CRF comes to dominate the HMM for very short sequence lengths, which suggests that there is no problem using CRFs with robot data even though gathering robot data is time consuming and difficult. Our empirical results for the CRF versus HMM case support the idea that a discriminatively trained CRF will have a lower

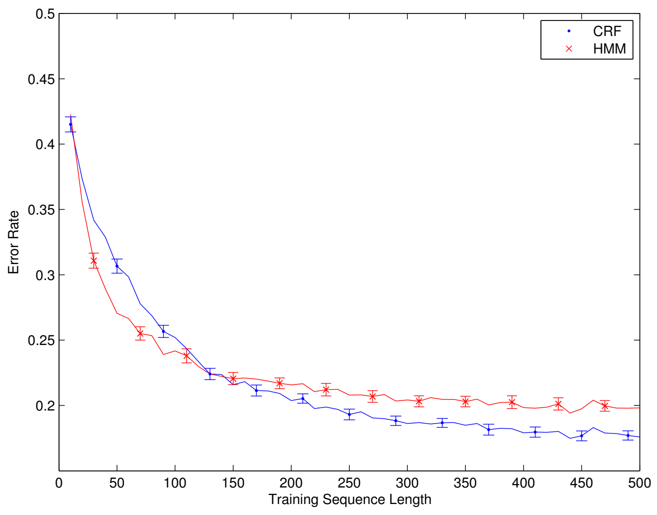


Figure 2.2: A comparison between the error rates on test data of a conditional random field and a hidden Markov model trained on synthetic sequences as the length of the training sequence is varied. We chose the features of the conditional random field to make it equivalent to a discriminatively trained hidden Markov model. For short training sequences, the hidden Markov model has a lower error rate. As the amount of training data is increased, the conditional random field comes to dominate the HMM. The error bars indicate 95% confidence intervals. The pattern where the generative model does better with little training data and the discriminative model dominates in the asymptotic case is exactly what we would expect from [78], which showed the same trend between logistic regression versus a naive Bayes classifier.

error rate than an equivalent generative model.

2.6.2 Non-independent Observations

Conditional random fields can incorporate non-independent features without suffering due to over-counting the evidence. Discriminative training in CRFs gives us the freedom to consider a rich class of features without suffering a penalty if the various features are not independent. In these experiments, we empirically show that generative models, such as HMMs are harmed by redundant features because they over-count the evidence presented by those features.

We constructed a synthetic data set with a mix of independent and non-independent observations. We generated a sequence of binary labels according to a first-order Markov chain with a 75% chance of self-transitions. We created observation vectors x_t that contained a mix of independent and non-independent features. We included an equal number of each type – independent or non-independent – in x_t, but we varied the length of x_t across trials. So, if x_t had a total size of $2n$ observations, n of the observations were independent and n of them were correlated with one another. We constructed the correlated observations by taking a single copy of the label y_t and adding 20% noise. We then duplicated this single noisy value n times to create n identical observations. To create the independent observations, we again copied the label variable y_t. However, we added 25% noise to each of n independent copies of y_t rather than adding noise a single time and duplicating one value, we made n copies of y_t and independently chose whether or not to corrupt a given copy with noise.

Figure 2.3 shows the error rates for an HMM and a CRF with HMM-equivalent features as the number of each type of observation n is varied. The conditional random field is able to use the increasing number of independent observations to steadily decrease its error rate as n increases. The error rate of the HMM increases with n. Despite the increasing amount of information present in the observation vector x_t, the performance of the HMM degrades. The decreasing accuracy is due to the fact that the duplicate observations are less noisy than the independent observations and the HMM makes a naive Bayes type of assumption in its observation model. Because the HMM assumes the n non-independent observations are independent, it over counts the information that they provide about the label. Because the non-independent observations are less noisy than the independent ones, the independent observations cannot correct for the case where the duplicate observations are incorrect. The HMM could avoid this problem of non-independent observations by modeling the joint probability of the observation vector. However, modeling the joint would require a number of parameters that is exponential in n, which is what motivated the naive Bayes assumption in the first place. As a discriminative model, the CRF avoids over counting evidence and does not pay the resulting penalty in classification accuracy that follows from making a naive Bayes type independence assumptions between non-independent observations.

2.6.3 Transitions that Depend on Observations

Conditional random fields can include complex features of adjacent label pairs and the entire observation sequence. In particular, CRFs can use information from the observations to influence transition probabilities. For example, in an assisted living or nursing context, a robot charged with aiding the elderly needs to detect emergencies, such as an unexpected fall. Observing the patient on the floor may indicate that a fall took place or maybe the patient is stretched out to watch television. On the other hand, observing the patient falling almost

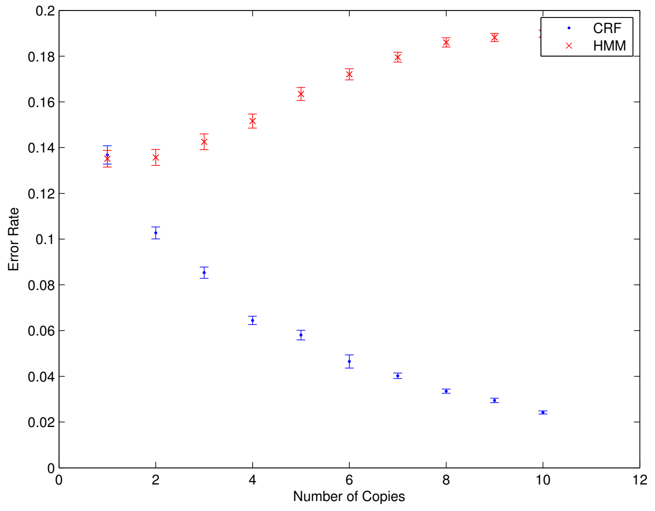


Figure 2.3: Error rates for a conditional random field and a hidden Markov model with a mix of independent and non-independent observations. The observation vector x_t contained a total of $2n$ values (n is the horizontal axis). The first n values were identical. We copied the label y_t with 20% noise once and repeated that single value n times. The second n values were independent. We created n independent copies of the label y_t with 25% noise. The error rate of the conditional random field fell as n increased because the CRF can exploit the increasing amount of information in the second n values of x_t. The error rate of the HMM increased because of the increasing amount of redundant information in the first n values of x_t.

certainly indicates that a transition from the robot's nominal state to an emergency state is in order. The short lived observation of the fall indicates that a transition into a long-lasting state is in order. The long lived observation of the patient on the floor is evidence that an emergency may be underway, but it is not as conclusive as the fall, which is the reason the situation became an emergency.

HMMs, which model transitions as $p(y_t|y_{t-1})$ cannot include observed information to influence transition probabilities. More complex directed models, such as dynamic Bayesian networks [76] do have this ability, but they are limited in the amount of past information that they can incorporate; the number of parameters in the model becomes intractable in DBNs if information from too many previous time steps is used. Conditional random fields, because they condition on the observations, can include arbitrary features of the entire observation sequence. For example, when estimating the state at time t, a CRF can incorporate a binary feature that indicates whether or not the observations show a patient falling at any point in the k minutes preceding t. A DBN, because it is a joint model of both labels and observations, would need to include links from the current state to all observations in the k minute window. To detect falls at arbitrary time steps, all labels would need to link to large swathes of the observation sequence, meaning that each observation has many parents. The size of the CPTs that represent the probability of observations given their parent labels would become intractably large because the number of parameters in the CPTs is exponential in the number of their parents. CRFs, which do not model the probability of the observations, do not include such parameters and do not suffer from this explosion in the number of model parameters when features are computed over many past observations.

We generated synthetic data where the state transitions depend on the previous observation to highlight the effect of linking transitions to observations. The CRF that we used in these experiments was *not* equivalent to an HMM, because it included features that cannot be represented in an HMM. Each observation vector x_t contained two values. The first value was a copy of the correct label y_t with 25% noise. The second value was the boolean result of a fair coin flip that contained no information about the label. We generated the label sequences using first-order Markov dynamics with one important difference. Rather than generating a single conditional probability table to store $p(y_t|y_{t-1})$, we created two tables. We used the coin flip from x_{t-1} to select which transition probabilities to use when sampling y_t. I.e. the coin flip is a mixture parameter for our transition model. We generated our two transition matrices by randomly sampling a value from $[0, 1]$, which we used as the self-transition probability in the first transition model. In the second transition model, we used $1 - p$ as the probability of a self-transition.

Figure 2.4 shows the average error rates of a CRF and an HMM as the transition probability was varied. The HMM, which expresses $p(X, Y)$ as the product of terms $p(y_t|y_{t-1})p(x_t|y_t)$, was not able to use the observed outcome of the coin flip from the previous time step and had a higher error rate than the CRF when the transition dynamics of the two different transition matrices were the most different at either side of the plot. The CRF, which included features of the form $f = (y_{t-1} \stackrel{?}{=} i)(y_t \stackrel{?}{=} j)(x_{t-1} \stackrel{?}{=} k)$, was able to model the two different transition functions and gained an advantage from doing so at the edges of the plot where the functions were the most different. In our synthetic example, a DBN which linked the current state to the previous observation would also perform well. However, because DBNs are joint models that include observation probabilities, there is a limit to the number of parents that any given observation can have before the model grows intractably large.

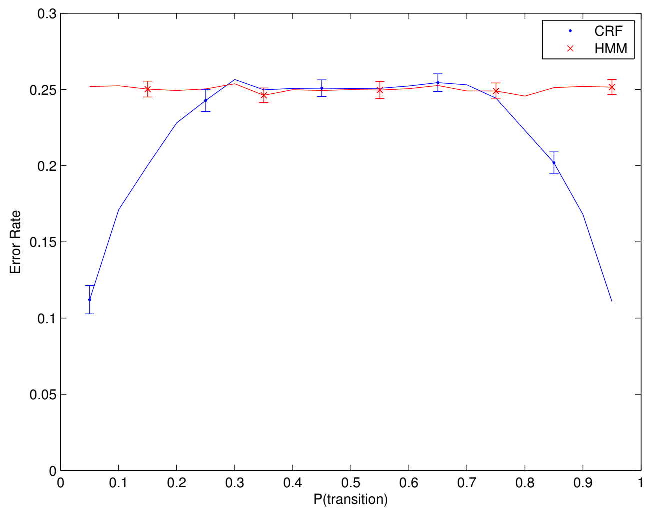


Figure 2.4: Error rates for a conditional random field and a hidden Markov model when label transitions depend on the observation. The observation vector x_t contains two values. The first value is a copy of the label y_t with 25% noise. The second value is the binary result of a fair coin flip and carries no information about the label. The label transition probability depends on the result of the coin flip. In the coin toss is heads, the label remains unchanged with probability p. If the coin comes up tails, the label changes with probability p. Because the CRF can include features that link transitions to the previous observation, the error rate for the CRF falls as p moves away from .5.

2.7 Tag Activity Recognition Experiments

We present activity recognition results using the robot tag example domain that we first introduced in [112]. In the next section, we review the formal definition of the classification problem, i.e. we specify what the labels and observations are in this domain, and introduce the notation that we use to specify feature functions. We compare a CRF to an HMM using the same set of features. In both cases, we compute a function $g(t, X)$ using information from the observations. In the HMM, we model $g(t, X)$ as a uni-modal Gaussian. In the CRF, we add equivalent features that take the form:

$$f(t, y_{t-1}, y_t, X) = (y_t \overset{?}{=} \text{label}) g(t, X) \tag{2.65}$$
$$f(t, y_{t-1}, y_t, X) = (y_t \overset{?}{=} \text{label}) g(t, X)^2 \tag{2.66}$$

In some cases, which we explicitly note, we include features that the HMM cannot represent. These features take the form

$$f(t, y_{t-1}, y_t, X) = (y_{t-1} \overset{?}{=} \text{label})(y_t \overset{?}{=} \text{label})g(t, X) \tag{2.67}$$

$$f(t, y_{t-1}, y_t, X) = (y_{t-1} \overset{?}{=} \text{label})(y_t \overset{?}{=} \text{label})g(t, X)^2 \tag{2.68}$$

We have no way of linking these features to transitions in the HMM, but we do include $g(t, X)$ as a uni-modal Gaussian in our model of $p(x_t|y_t)$.

2.7.1 Notation

In the robot tag domain, we identify which robot is the seeker at each time step. We refer to the ID of the seeker at time t as the label y_t and define the set of labels for the entire sequence $Y = \{y_1, y_2, ..., y_T\}$. Similarly, we define the input observations $X = \{x_1, x_2, ..., x_T\}$ where x_t is a vector of the observations from time step t. In the tag domain, the observation vector contains the two dimensional position of each robot in the environment; i.e., x_t contains three pairs of Cartesian coordinates.

We define functions $g(t, X)$ that we use as building blocks for features in the CRF and HMM. We use $\text{pos}_{r,t}$ for the position of robot r at time t. Position values are available directly from the observation x_t at each time step. We also define also use $\text{vel}_{r,t} = \text{pos}_{r,t} - \text{pos}_{r,t-1})$ to estimate the velocity of robot r at time t.

2.7.2 Experiments

We generated experimental data from both the hourglass and the unconstrained variants of the tag domain. These data were generated from a physics-based simulator for holonomic robots that included realistic acceleration and velocity constraints on the robots. We generated training and test sets that were each approximately 20 minutes in length. Since the simulator operates at a rate of 60 hz, this means that each sequence contained more than 70,000 observation vectors, each labeled with the identity of the seeker at that time step. The observation vectors themselves consisted of three two-dimensional positions, one for each robot. These positions form the basis of the features passed to the models for classification.

In all experiments, the CRF included intercept features ($f_i = (y_t \overset{?}{=} j)$) as well as transition features ($f_i = (y_{t-1} \overset{?}{=} i)(y_t \overset{?}{=} j)$) as a base model. We added more specialized features on top of these. We did not augment the HMM with these types of features because it already has a separate transition model and prior probability over start states.

2.7.3 Feature Combinations and Results

We experimented with a wide variety of features in both models. Below, we discuss and interpret the performance of the CRF and the HMM with various combinations of features using the results shown in table 2.1.

Raw Positions Only

In the initial set of experiments, we trained the HMM and CRF on the raw positions of each robot; in other words, the models were given x_t as input at each time step. The goal of these experiments was to verify that some sort of computed features are indeed required for reasonable classification accuracy.

To complement the HMM's Gaussian observation model, the CRF included features for $x_t[k]$ and $(x_t[k])^2$ for each of the k values in x_t. Concretely, we added features built around

$$g(t, X) = x_t[k] \tag{2.69}$$

to the model, where k is the index to one coordinate of a robot's location in the raw observation vector.

Table 2.1 show two things. First, the raw observations by themselves do not provide enough information for either model to perform well. In the case of the hourglass domain, the problem is simple enough that the CRF, which correctly labels 53.6% of the test set, performs far better than the baseline performance achievable by random guessing (approximately 33%). For this simple domain, at least, there is an advantage to training the model discriminatively as the CRF does better than the HMM, although this advantage seems to disappear with the more complex, unconstrained data set where both models perform poorly.

Velocities

We used our domain knowledge that the velocities of the robots are important in the tag domain to replace the raw observations. We build our new set of features around robot velocities:

$$g(t, X) = \mathrm{vel}_{r,t} \tag{2.70}$$

Table 2.1 confirms that using velocity as a feature in the classifier yields dramatically improved classification accuracy.

Velocity Thresholds

Incorporating velocity information into the models greatly improved accuracy. The seeker's five second pause is an obvious explanation for the improvement. Rather than feeding raw velocities into the models, we constructed features that test if the velocity is below a certain threshold for a single time step or for a series of consecutive time steps. Such features more clearly capture the notion of a stopped or stopping robot. As a natural complement to testing if a robot's velocity is below a threshold, it makes intuitive sense to test if the robot's velocity is above a threshold as well. This second form of feature will be correlated with non-seeker robots as well as the seeker when it has finished pausing after first being tagged. We constructed features from the functions:

$$g(t, X) = \prod_{i=t-w+1}^{t} \|\text{vel}_{r,i}\|_2 \overset{?}{\leq} k$$

$$g(t, X) = \prod_{i=t-w+1}^{t} \|\text{vel}_{r,i}\|_2 \overset{?}{\geq} k$$

In our experiments, we chose the velocity threshold k to be 20% of the robots' maximum velocity.

In addition to testing the effectiveness of incorporating specific domain knowledge into the models, these velocity thresholds allow us to perform another test as well; hidden Markov models assume that observations are independent given their labels. Creating features from overlapping portions of the observation sequence X, violates the assumptions of the model. An HMM with such overlapping features is no longer a proper generative model and the likelihood function for the HMM is no longer correct due to over counting of the evidence. By testing the HMM and CRF with differently sized windows, we tested how much, if at all, violating the independence assumptions of the HMM harmed classification accuracy.

The results for the CRF in table 2.1 are unambiguous. The conditional random field becomes more accurate as the size of the window is increased. Incorporating information from many different time steps into feature functions does not violate any of the independence assumptions made by the CRF.

The HMM results are less clear cut. With the hourglass data set, the accuracy of the HMM decreases as the window size is increased, perhaps due to the increasingly severe violation of the independence assumptions of that model. In contrast, HMM performance increases as the window size is increased with the more complex, unconstrained data set. It is not clear why performance suffers with larger window sizes on the simple data set but then improves on the more complex data.

Chasing Features

The velocity threshold features showed that using domain knowledge to create features can improve model accuracy. Aside from the seeker pausing in place after being tagged, the other defining characteristic of the seeker role is that the seeker chases other robots. To capture this property of the tag domain, we created a "chasing" function to capture whether each robot was moving towards one of the other robots using the dot product of robot velocity with the vector pointing from one robot to the other.

$$g(t, X) = \frac{(\text{pos}_{r_2,t} - \text{pos}_{r_1,t})}{\|\text{pos}_{r_2,t} - \text{pos}_{r_1,t}\|_2} \frac{\text{vel}_{r_1,t}}{\|\text{vel}_{r_1,t}\|_2} \tag{2.71}$$

As with the velocity threshold functions, incorporating domain knowledge in the form of feature functions improves accuracy. Table 2.1 shows that classification accuracy in the hourglass domain breaks 95% for the first time while accuracy in the unconstrained domain breaks 80% for the first time.

Distance Thresholds

Activity recognition in the game of tag is a matter of detecting transitions. In one time step, robot 1 is the seeker. During the next time step, robot 1 approaches close enough to tag a different robot and that new robot becomes the seeker. We are interested in detecting the points where the seeker role is passed between players.

By looking at the distances between pairs of robots over a series of adjacent time steps, we can create features to detect when a hand off of the seeker role may have occurred. At first, this simply appears to be a matter of tracking whenever two robots are in close proximity and having a feature act as a flag at these points, with one flag for each pair of robots that could be involved in the transaction.

However, there are many time steps when robots will be close to each other and no hand off will take place; when a robot is first tagged, it halts in place. It takes the former seeker time to deaccelerate and move away from the newly created seeker. During this time, both robots will be within the threshold distance, although no hand off is possible. In other words, it is only possible for a hand off to occur during the first frame that two robots are within the threshold distance.

A second complication arises because the execution order of the robot behaviors is not fixed in the simulator. In some situations, the non-seeker will close the gap between itself and the seeker and in others the seeker will close the gap with its action. Depending on which order these events take place, the hand off of the seeker role can occur in either the first time step when the robots are below the threshold distance or during the second time step. Accounting for both possible times when a hand off could have occurred, we define the function

$$g(t,X) = ((dist_t(r_1,r_2)\overset{?}{\leq}k) + (dist_{t-1}(r_1,r_2)\overset{?}{\leq}k))(dist_{t-2}(r_1,r_2)\overset{?}{>}k)\overset{?}{>}0 \tag{2.72}$$

We use $g(t,X)$ to create transition linked features that detect when a hand off was possible as well as the negation of those features to capture situations when it was impossible for the seeker role to be transferred between two robots. Note that the HMM cannot link transitions to observations, so it is at a disadvantage in these experiments. We used the following features in the CRF and incorporated $g(t,X)$ as an observation in the HMM.

$$f_k(t,y_{t-1},y_t,X) = (y_{t-1}\overset{?}{=}\text{label})(y_{t-1}\overset{?}{=}\text{label})g(t,X) \tag{2.73}$$

$$f_k(t,y_{t-1},y_t,X) = (y_{t-1}\overset{?}{=}\text{label})(y_{t-1}\overset{?}{=}\text{label})(1-g(t,X)) \tag{2.74}$$

In the negation case, the model learned negative weights to make label sequences Y in which such transitions occur less likely than sequences that lack forbidden state transitions.

The results in table 2.1 are surprising. These distance threshold based state transitions exactly capture the underlying dynamics of the process that generated the data, yet using only these features results in lackluster performance. The first set of results, labeled "Distance (U)", show what happens when the feature values are not normalized to have mean zero and a variance of one. After all, such normalization is rarely necessary with binary features.

However, consider how often two robots will first approach close enough together for a tag event to take place. These are the only situations where the distance threshold features will take on a non-zero value. This situation happens approximately 200 times in the twenty minute training set. Now consider how often the features corresponding to self-transitions take on a non-zero value. One of these features fires for virtually each of the roughly $70,000$ time steps in the training set. The expected value of the regular transition features is much larger than the expected value of the distance threshold features. Any difference in the empirical value of the regular transitions and their expected value under the model will dominate the function gradient during optimization.

Rather than eliminating the regular transition features, which would leave the optimizer stranded in a plateau and unable to make any progress, we normalize the distance thresholded transitions. Normalizing the distance threshold features produces no appreciable change in the accuracy in the hourglass domain and the accuracy in the unconstrained domain improves slightly. However, there is a major difference in the log likelihood of the test set for both domains, which indicates that the model achieves a better fit on the test set even if this improvement is not enough to greatly change the accuracy of the models.

	Hourglass			Unconstrained		
Features	HMM	CRF	$\ell(Y\|X)$	HMM	CRF	$\ell(Y\|X)$
Positions	33.1	53.6	-959.7	37.1	37.8	-1354.5
Velocities	68.4	89.4	-717.1	55.7	70.4	-1206.5
Velocity Thresholds						
W = $\frac{1}{60}$th sec.	62.5	71.2	-818.0	46.8	58.6	-1148.6
W = 0.1 sec.	63.0	73.9	-784.3	46.0	62.4	-1099.2
W = 0.5 sec.	63.6	80.6	-708.8	68.9	71.9	-983.1
W = 1.0 sec.	60.2	83.1	-721.8	67.8	75.3	-980.9
W = 1.5 sec.	56.9	85.5	-731.7	68.8	77.8	-1004.7
W = 2.0 sec.	53.7	87.1	-751.1	72.1	77.3	-1036.3
Chasing	75.8	95.4	-622.3	65.5	80.4	-1058.3
Distance (U)	46.6	49.5	-779.7	43.5	42.3	-604.4
Distance (N)	46.6	49.9	-200.6	43.5	58.0	-223.4
Distance & Chasing (U)	75.6	99.3	-90.6	65.8	93.9	-181.8
Distance & Chasing (N)	75.6	99.3	-115.3	65.8	97.6	-112.2
Many Features	72.4	98.1	-172.2	63.4	98.5	-178.9
Redundant Features	72.4	95.7	-509.3	52.7	93.8	-6432.3

Table 2.1: CRF and HMM accuracy for identifying the seeker in the simplified hourglass and unconstrained tag domains. (U) and (N) indicate that the binary distance threshold features were either unnormalized or normalized in the corresponding trials.

Distance Thresholds with Chasing Features

A natural question that arises is whether or not the distance threshold features are capable of increasing the accuracy of the CRF. In theory, they perfectly capture the characteristics of the domain, but in practice training the model with only these features is difficult. As a test, we combined the distance threshold features with the chasing features to see if the combination of the features together would out perform either set of features on its own.

As the results in table 2.1 illustrate, there is indeed a synergistic effect between the two feature sets. One set of features, the chasing features, captures the behavior of the seeker when it is moving. The other set, the distance threshold features, captures when the seeker role transitions from one robot to another. The resulting models exhibit higher log likelihoods on the test set as well as, when the distance thresholds are normalized, better accuracies than any other features tried thus far.

Many Features

In the final set of experiments, we examined how the models perform when all of the previously discussed features were used at the same time. In the case of the windowed velocity thresholds, we used a a window size of 1 to avoid deliberately breaking the independence assumptions of the HMM.

As a second experiment, we included all of these “helpful” features, as well as redundant features to test how vulnerable the models are to over fitting. The extra features we added were the raw position observations, the change in distance between each robot and its closest neighbor, each robot’s x and y velocity components, and a series of dot products along each robot’s trajectory to measure how linearly the robot was moving as non-seeker robots tend to travel in straight lines. While all of these additional features potentially include information about the domain, this information is either provided by the existing features or of somewhat dubious utility.

The results in table 2.1 show an initial increase in accuracy for the CRF when the first collection of features are added to the model and then a decrease with the addition of redundant features on the unconstrained data set. On the hourglass data set, even the addition of the first set of features lowers the final accuracy of the model, probably because of over fitting. Over fitting reduces accuracy immediately with the simple data set but only with the addition of redundant features for the more complex data. The degree of over fitting is especially evident from the large decrease in the log likelihood of the test set for the unconstrained data set with redundant features.

2.7.4 Discriminative Training and Smoothing in the HMM

The results in 2.1 show that a CRF, with HMM-equivalent features, consistently outperforms an HMM when we use MLE to learn model parameters in the HMM. In this section, we explore ways to close the accuracy gap between the two models. We investigate smoothing the observation model of the HMM to see if additional smoothing can increase its accuracy. We also consider discriminative training for the HMM to see if the accuracy gap is due to the different training regimes.

To model the observation probability $p(x_t|y_t)$, we make a naive Bayes style independence assumption over the components of the observation vector in the HMM. Breaking the prob-

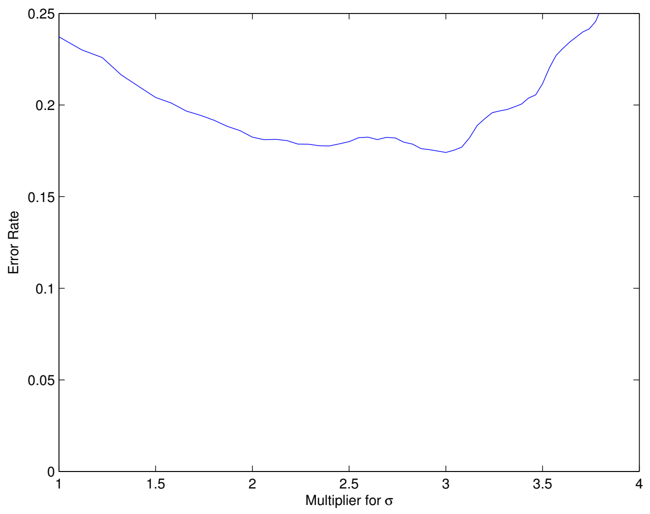


Figure 2.5: Smoothing the HMM Observation Model: We tested the effect of smoothing the observation model of the HMM on the error rate in the tag domain. To smooth the Gaussian observation model, we multiplied the original standard deviations of the elements in the observation vector σ_i by a scalar value.

ability down into the individual elements of the observation vector yields

$$p(x_t|y_t) = \prod_i p(x_t[i]|y_t) \tag{2.75}$$

where $x_t[i]$ is the i-th element of the observation vector at time t. We model individual elements as Gaussians and use (μ_i, σ_i) the refer to the parameters of the distribution that we use to model the i-th element.

To smooth the HMM, we begin with the MLE parameters and then we increased the variance of each normal distribution. If $p(x_t[i]|y_t) \sim N(\mu_i, \sigma_i^2)$ in the MLE parameters, we multiply σ by a scalar λ to yield a smoothed distribution $N(\mu_i, (\lambda\sigma_i)^2)$. Figure 2.5 shows the error rate as λ is increased and the HMM contains velocity and "chasing" features. We chose those two particular features because they do not violate the independence assumptions of the HMM and there is a large gap between the HMM and CRF accuracies when they are combined. Up to a certain point, smoothing the model does lower the error rate of the HMM on test data. However, smoothing alone is insufficient to close the accuracy gap between the HMM and the CRF. The error rates for an unsmoothed HMM, a CRF, and the most accurate smoothed HMM are shown in table 2.2.

CRF	93.5%
HMM with MLE parameters	76.3%
Smoothed HMM	82.6%
Discriminatively trained HMM	94.2%

Table 2.2: Accuracies for the CRF, a traditionally trained HMM, a smoothed HMM, and a discriminatively trained HMM with velocity and chasing features.

Smoothing alone does not close the gap between an HMM with MLE parameters and a discriminatively trained CRF, even when the CRF only contains features that make it equivalent to a discriminatively trained HMM. To test whether or not the gap between the two models was due to discriminative training, we implemented a simple coordinate-wise discriminative training method for HMMs. Our algorithm maximizes the conditional likelihood of the training set by adjusting the model parameters of the HMM one at a time in a series of discrete steps. The step size that we used each iteration was proportional to $1/\sqrt{j}$, where j is the current iteration of training and we normalized multi-nomial distributions, e.g. the transition matrix, after each step to maintain proper distributions. Similarly, we also bounded the variances of Gaussians to non-negative values. In practice, there are faster alternatives for discriminatively training HMMs, e.g. [19], and there is no reason to use the HMM parameterization in place of a CRF with HMM-equivalent features. The purpose of our training algorithm is to test whether or not a discriminatively trained HMM is as accurate as a CRF with HMM-equivalent features and to directly compare the discriminatively trained model parameters to the MLE HMM parameters.

Table 2.2 shows the accuracies of a CRF with HMM-equivalent features and a discriminatively trained HMM. Both the CRF and the discriminatively trained HMM yield high accuracies on test data. The discriminatively trained HMM has slightly higher performance. One possible explanation is that the CRF over-fits more severely. Training the CRF took approximately 1 minute of CPU time. We stopped HMM training after 48 hours; our coordinate-wise algorithm is effective, but not efficient. Early stopping of HMM training may have resulted in less over-fitting.

Both version of the HMM had approximately the same transition matrices and prior probabilities for the initial state. The two models differed only in their observation models. Figure 2.6 shows the normal distributions over each of the 9 elements of the observation vector for the two models. The MLE parameters reflect the true distribution of the data in the training set. The parameters in the discriminatively trained HMM do not accurately reflect the distribution of the training data. Instead, the discriminatively trained version of the observation model contains parameters that maximize the conditional probability of

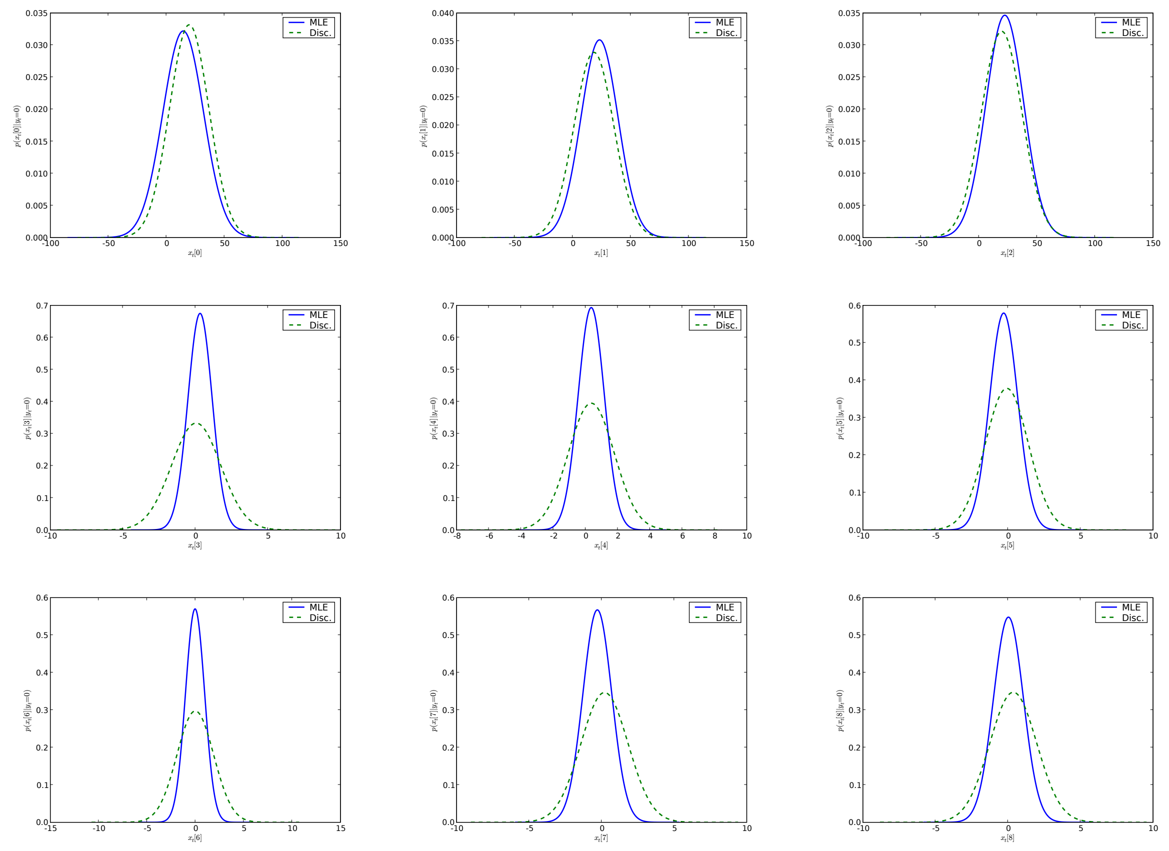


Figure 2.6: We compare the observation models from a discriminatively trained HMM with the MLE parameters. The top row represents the velocity of each robot. The bottom two rows represent the "chasing" feature, which detects if one robot is moving towards another.

the labels $p(Y|X)$ given the training data. In the case of robot velocities, the resulting distributions are similar to the MLE parameters, with slightly shifted means. In the case of the chasing features, the means moved slightly and the variance increased significantly. Increasing the variance by the same factor for all distributions, as in our previous experiments, did not reduce the error rate as well as using discriminative training to choose means and variances on an individual basis rather than applying the same amount of smoothing across all observations.

The accuracies in table 2.2 show that discriminative training closes the gap between the HMM and a CRF. The plots in figure 2.6 show that discriminative training produces distributions over the observations that do not reflect the empirical distribution of the observations in the training set. In our results, the best models of $p(Y|X)$, which produce higher classification accuracies than a joint model, do not model the true distribution of the observations in X.

2.8 Chapter Summary

In this chapter:

- We presented a comprehensive introduction to conditional random fields that focused on why CRFs are well suited to activity recognition from robot sensor data.
- We described how conditional random fields represent conditional likelihood and how to use inference in the models to answer queries about label sequences.
- We provided a detailed description of how to implement a conditional random field and gave pseudo-code implementations for key functions. We also discussed how to optimize a CRF implementation.
- We used synthetic data to highlight important properties of conditional random fields for activity recognition from robot sensor data.
- We provided a detailed example of designing features for activity recognition using a robot tag domain.
- We compared between a CRF and a discriminatively trained HMM and demonstrated that discriminative training has the potential to produce more accurate models.

Chapter 3

Feature Selection

In this chapter, we present an introduction to feature selection in conditional random fields. We introduce three feature selection algorithms and explore the properties of the algorithms using a variety of real and synthetic data.

3.1 Introduction

In the previous chapter, we used a robot tag domain to explore activity recognition in conditional random fields. We found that low level information on its own, i.e. the positions of the robots, was not sufficient for accurate activity recognition. We found that complex, domain specific features were needed to produce low error rates in the final model. However, we also found that including too many features led to over-fitting, which reduced our classification accuracy on unseen test data. These observations provide our motivation for feature selection. We require complex features to achieve high classification accuracy, but we must be selective about which features we include to avoid over-fitting.

We have a second motivation for limiting the number of features in our model. The time required to perform inference on unseen test data increases linearly with the number of features. We desire models that are useful on real robots. Real robots must rapidly respond to their changing environment, so fast inference is critical. Furthermore, real robots face severe computational limitations. Computationally intensive tasks, such as vision and localization, compete with other models for processing time. Feature selection is desirable because it produces light-weight models that require fewer resources to implement in a real robotic system.

We begin our discussion of feature selection with a formal categorization for different types of features and with a general overview of activity recognition. We review the main classes of feature selection algorithms and justify our choice of embedded methods for feature selection in conditional random fields.

3.1.1 Types of Features

Previous work on feature selection distinguishes between various features based on their *relevance* to the classification task [10]. Intuitively, a feature is relevant if it provides information about the labels. John et al. formally define two classes of relevant features: *strongly relevant* and *weakly relevant* [43]. We restate their definition, updated for the sequential classification setting, below:

Strongly Relevant A feature is strongly relevant if it is possible to find two training examples that differ only in their label and the value of that particular feature. I.e., the classifier should assign different labels to the examples and the sole distinguishing characteristic is that one feature. Using the formal definition of John et al.: feature f_i is strongly relevant iff there exist two labeled sequences of equal length (X, Y) and (X', Y') such that $P(X, Y) > 0$; $P(X', Y') > 0$; $Y \neq Y'$; and $\forall t \forall j \neq i : f_j(t, y_{t-1}, y_t, X) = f_j(t, y'_{t-1}, y'_t, X')$ where y_{t-1}, y_t are adjacent labels from Y and y'_{t-1}, y'_t are adjacent labels from Y'.

Weakly Relevant A feature is weakly relevant if it is not strongly relevant given the current set of features in the model, but it can be made strongly relevant by removing some or all of other features.

Irrelevant A feature is irrelevant if it is not strongly relevant when it is the sole feature in the model.

The relevance of a feature depends on the other features in the model. A strongly relevant feature might become weakly relevant as new features are added to the model. The notion that relevance leads into the final definition of this section:

Incremental Usefulness Given a learner and its current set of active features A, a feature f_i is incrementally useful if the error rate of the learner trained with feature set $A \cup f_i$ is lower than the error rate of the learner trained with feature set A [18].

3.1.2 Types of Feature Selection Algorithms

Feature selection algorithms fall into three broad categories: *Filters*, *Wrappers*, and *Embedded Methods* [10]. We review these three categories of feature selection algorithms and discuss their suitability for feature selection in conditional random fields.

Filters

Filters use a fast heuristic to estimate the relevance of all candidate features and then train the model using the highest ranked set of features [10]. The relevance of each feature is computed in a batch process and all top candidates are added as a single group; filters do not reevaluate candidate features in light of an existing active set. As a consequence, feature selection with filters is computationally efficient because each feature is evaluated a single time.

A wide range of heuristics are used with filter methods, e.g. correlation, mutual information, and information gain [10, 36]. For example, the RELIEF algorithm [47, 48] computes a heuristic estimate of feature relevance on an instance by instance basis. For a single labeled training example $e = (x, y)_i$, RELIEF locates two other training examples e_y and $e_{\bar{y}}$ such that e_y is the example most similar to e that has the same label as e and $e_{\bar{y}}$ is the example most similar to e with a different label. RELIEF is restricted to binary classification tasks, although others have introduced extended versions for multi-class problems [52]. RELIEF increases the weight of features that help distinguish between the neighbors e_y and $e_{\bar{y}}$. The intuition is that features that help distinguish between closely related examples are relevant.

Filter methods suffer in the presence of many *weakly relevant* features [10], which makes them poorly suited for activity recognition domains. In the Tag domain from the previous chapter, features that link state transitions to the observation that one robot is within a threshold distance of another perfectly capture the transition dynamics of the domain, if the correct value for the distance threshold is used. However, usually the correct distance is unknown and we must include many candidate distances. Filter methods will select all such features even though only the distance threshold closest to the true value is strongly relevant, because filters do not reevaluate their scoring metrics in light of previously selected features.

Wrappers

Wrappers evaluate candidate sets of features by fully training a model that includes only the candidate features and evaluating it using cross-validation or held-out data [51]. I.e. wrappers use the classification algorithm as a subroutine in a search over the space of possible feature sets. A given feature can either be active or inactive, so there are 2^n possible feature sets, where n is the number of features. Due to the size of the search space, exhaustive search is intractable.

Wrappers potentially avoid the problem of weakly relevant features because they evaluate candidate sets of features using the actual classification algorithm. By definition, weakly relevant features provide strictly less information than strongly relevant features, so the search space will always contain a smaller model that performs at least as well as any model that contains weakly relevant features, neglecting noise due to cross-validation and assuming ties are broken in favor of smaller models. Similarly, wrappers can select features that are only useful in the presence of other features. For example, wrappers can potentially find the correct set of features when the correct label depends on the parity of those features.

Wrappers have the ability to discover the optimal feature subset. However, feature selection using wrappers requires a large amount of computation. Using a wrapper strategy with forward search (no backtracking) to choose 100 features from a candidate set of 1,000 features requires training the model more than 95,000 times. Training a CRF after making a small change in its feature set is faster than training a CRF from scratch. However, CRFs often include thousands or millions of features, e.g. [97], and simple forward selection with wrappers trains the model $O(n^2)$ times. Wrappers do not scale to large feature sets. The largest number of features considered in [51] was 180 features. The median number of features in that work was 13. Computational considerations rule out wrappers for large scale feature selection.

Embedded Methods

Embedded feature selection algorithms are the least clearly defined of the three categories. Embedded methods combine the process of feature selection with the process of training the final model [10]. Combining feature selection with training allows embedded methods to strike a balance between filters and wrappers. Filters select the candidate features before training the model and therefore cannot fully capture interactions between groups of features. Wrappers train the model for every candidate subset of the features, which comes with a high computational cost. Embedded methods are less myopic than filters; they take previously selected features into account. They are also more computationally efficient than wrappers, e.g. they can use cheaper (albeit less accurate) scoring metrics to search through the space of feature sets.

Embedded feature selection algorithms have a long history. For example, inductive decision tree learners (e.g. [12, 89, 90]), can be viewed as embedded feature selection algorithms. Choosing a feature or attribute to create a split in a decision tree can be viewed as selecting a feature. Learning an entire decision tree can be viewed as embedded feature selection because previously chosen features affect which features are chosen for splits further down in the tree; during learning, the training set is divided and examples are passed down a particular path from the root to the leaves of the tree as if the examples were being classified. Each leaf is then split by choosing a feature based only on the examples present at that leaf. In this way, the previously selected features, which select the examples present in each leaf, affect the choice of future features.

We focus on embedded methods for feature selection in conditional random fields. When ruling out filters, we presented the example of choosing a distance threshold in the Tag activity recognition task. Because embedded features take previously selected features into account, they handle weakly relevant features more robustly than filters. In addition, embedded methods are more tractable from a computational standpoint than wrappers. We will discuss three different embedded methods: grafting [85], ℓ_1 regularization (e.g., [39]), and a CRF-specific feature induction algorithm due to McCallum [73]. We discuss these three algorithms after a brief discussion of the evaluation metrics that we will use to compare them.

3.1.3 Evaluating Feature Selection Methods

We will evaluate feature selection algorithms across several different dimensions.

Trade-off between Accuracy and Sparsity Our primary concern is to create models that can accurately label unseen sequences of observations. An obvious metric for comparing different feature selection algorithms is to compare how accurately the resulting models label test data. A second metric to consider is the sparsity of the resulting models. It is not enough for a model to have a low error rate; for the sake of parsimony and to reduce the computational burden of applying the model, we require accurate models that incorporate the minimum number of features. There is, of course, a trade-off between these two dimensions and it is interesting to compare how different feature selection algorithms manage the trade off between classification accuracy and the number of features in the model and to identify what factors, such as the size of the training set, affect that trade-off.

Robustness to Irrelevant Features The purpose of feature selection is to select a small subset of *strongly relevant* features from a potentially large set of either *weakly relevant* or *irrelevant* features. To compare different feature selection algorithms, we can examine what happens to the classification accuracy of the resulting models as the training set is augmented with an increasing number of irrelevant and weakly relevant features. We can also track the number of such undesirable features that are included in the final models as the amount of noise in the training set increases.

Data Efficiency Training data is a scarce and precious resource. Good feature selection algorithms should be able to identify strongly relevant features from modest amounts of training data. We can compare how well the different algorithms fare as the size of the training set is varied.

Time Efficiency We will consider this in more detail in the next chapter, but it is important to consider the computational burden of feature selection. Even if the resulting sparse model allows for cheap inference at test time, we require algorithms that efficiently discover that sparse set of features from a potentially large candidate set.

Generalization Closely related to data efficiency, how well does the resulting model generalize across related data sets? For example, it is much easier in practice to gather data in simulation than using real robots. It would be beneficial if we could use abundant simulator data to train a model that also performs well when tested on data from real robots. In other words, how well do the resulting models generalize between related sets of data? A trade-off that we make with feature selection algorithms is that less sparse algorithms may generalize better than a more efficient counter part. Classification accuracy and generalization to new domains is interesting to test to see if one algorithm produces models that are less brittle than the others.

Effectiveness for Parameter Selection We can consider a variety of parameters in our model. For example, in the last chapter, we presented a tag domain where one robot needed to approach within a fixed distance of another in order to tag its target. As designers of models, we do not always know the correct value of such thresholds. Can feature selection choose the correct parameter given features that incorporate many options? This is exactly the same as identifying the most strongly relevant feature in the presence of other weakly relevant features. A common issue is whether or not a given feature should be state-linked or transition-linked. Can feature selection choose between the two? An additional type of parameter discovery that we can do is to correct degree of polynomials when we model our data with an additive model. Including x^2 features gives us a Gaussian. Are higher order terms helpful or do they merely over-fit?

3.2 Feature Selection in Conditional Random Fields

In this section, we talk about feature selection in the specific context of conditional random fields. We specify what we mean by *feature* in the case of CRFs. We then describe three feature selection algorithms, two of which are general algorithms that can be applied to many types of classifier, and a third, CRF-specific algorithm.

3.2.1 Defining "Feature"

We define conditional random fields as a vector of weights and a vector feature function $f(t, y_{t-1}, y_t, X)$. We will often write a single prototype, such as:

$$f_i(t, y_{t-1}, y_t, X) = (y_{t-1} \overset{?}{=} \text{label})(y_t \overset{?}{=} \text{label}) \tag{3.1}$$

as a shorthand for several individual features in the vector function f. Expanding out this shorthand for the binary label case would yield:

$$f_i(t, y_{t-1}, y_t, X) = (y_{t-1} \overset{?}{=} \text{True})(y_t \overset{?}{=} \text{True}) \tag{3.2}$$

$$f_j(t, y_{t-1}, y_t, X) = (y_{t-1} \overset{?}{=} \text{False})(y_t \overset{?}{=} \text{True}) \tag{3.3}$$

$$f_k(t, y_{t-1}, y_t, X) = (y_{t-1} \overset{?}{=} \text{True})(y_t \overset{?}{=} \text{False}) \tag{3.4}$$

$$f_l(t, y_{t-1}, y_t, X) = (y_{t-1} \overset{?}{=} \text{False})(y_t \overset{?}{=} \text{False}) \tag{3.5}$$

In summary, a single *prototype* is responsible for generating several *features*. When we talk about feature selection, we are talking about choosing among the features rather than among the prototypes. As an example of why we do this, consider the Tag domain from the previous chapter. A useful prototype in Tag is:

$$f_k = (y_{t-1} \overset{?}{=} i)(y_t \overset{?}{=} j)(\text{dist}(i, j) \overset{?}{\leq} v) \tag{3.6}$$

The three features generated from the above prototype that capture self-transitions, i.e. situations where the "It" role is not transferred, are irrelevant – of course robot i is within v millimeters of itself! In more complex situations, e.g. if we tested the distance between robots not involved in the role transfer, additional features are both irrelevant and increase over-fitting. Selecting on a feature-wise rather than prototype-wise basis allows the feature selection process to potentially reduce over-fitting at the cost of an increased number of candidates (by a factor of $|y_t|^2$ in the above example).

In general, we use two types of features:

State-Linked Features are features with prototypes of the form:

$$f_i = (y_t \overset{?}{=} \text{label}) g(t, X), \tag{3.7}$$

where g is an arbitrary function that does not depend on y_{t-1} or y_t. E.g., we capture information about continuous variables with $g(t, X) = x_t$ and $g(t, X) = x_t^2$.

Transition-Linked Features are features that with prototypes of the form:

$$f_i = (y_{t-1} \overset{?}{=} \text{label})(y_t \overset{?}{=} \text{label}) g(t, X), \tag{3.8}$$

where g is an arbitrary function that does not depend on y_{t-1} or y_t. We encode first-order Markov transitions by setting $g(t, X) = 1$.

3.2.2 Grafting

Grafting is an embedded feature selection method that combines the task of training the model with choosing which features to include in it [85]. Grafting uses a greedy forward selection strategy to choose features. It begins training with an empty model that includes no features. During each each iteration, grafting adds a single feature to the set of active features and retrains the model using the expanded set. Training the model with the new feature is significantly faster than training the model from scratch because the weights from the previous iteration provide a starting point close to the new optimum; training must discover a weight for the new feature and adjust the weights of the original features to take into account the added feature, but in general the new optimum will lie close to the previous weight vector.

Grafting uses a heuristic to choose which feature to add during each iteration. In theory, we would like to maximize the *incremental usefulness* (section 3.1.1) of the added feature. Computing the true gain in accuracy for each candidate feature would, of course, yield a wrapper based approach and we have already rejected wrapper approaches because they are too slow. Rather than training the model with each candidate feature, grafting uses the gradient of the objective function as a heuristic to choose among candidate features and selects f_i according to:

$$\underset{i}{\operatorname{argmax}} \left| \frac{\partial}{\partial w_i} \ell(Y|X; w) \right| \tag{3.9}$$

Less formally, grafting computes the gradient of the objective function $\ell(Y|X)$ with respect to the weights and chooses the feature corresponding to the weight with the largest absolute partial derivative[1].

Grafting can incorporate regularization in order to smooth the model during training and feature selection. Perkins et al. discuss how to incorporate an ℓ_0 penalty, which charges a fixed penalty for each non-zero weight in the model; an ℓ_1 penalty, which charges a penalty proportional to the absolute magnitude of the weights; and an ℓ_2 penalty, which charges a penalty proportional to the sum of the squares of the weights. Using any of these regularization methods requires choosing scalar parameters to control the degree of smoothing. To avoid choosing between one and three extra parameters, we used no regularization when training with grafting. Instead, after using grafting to choose a subset of the features, we use ℓ_2 regularization to smooth a model that includes only the chosen subset of the features. Retraining with regularization after the fact provides the benefit of smoothing the final model parameters, although with the cost of retraining and potentially changing which features are included because no smoothing is done during feature selection.

3.2.3 ℓ_1 Regularization

Training a model with an ℓ_1 penalty produces a *sparse* model where many of the parameters are exactly equal to zero [39]. In conditional random fields, we remove features with zero weight from the model because they do not affect classification. We use ℓ_1 regularization for feature selection by varying the magnitude of the ℓ_1 penalty to create candidate models with different numbers of active features. We then use cross-validation or a holdout set to choose among the resulting candidates.

Instead of maximizing the log-likelihood $\ell(Y \mid X)$, we maximize the penalized log-likelihood to learn the parameter vector w^*:

$$w^*(\lambda) = \operatorname*{argmax}_w \; \ell(Y \mid X; w) - \lambda \sum_i |w_i|, \tag{3.10}$$

where λ is a scalar parameter that controls the degree of smoothing. We vary the magnitude of λ to create different candidate models. High values of λ correspond to large penalties for

[1]Grafting uses the gradient of the log-likelihood of the training set rather than the gradient of the error rate for the same reason that we train our models by maximizing log-likelihood rather than accuracy – we cannot exactly compute the gradient of the error rate. However, Gross el al. [35] describe a training method for conditional random fields that minimizes an approximation of the error rate. Their approximation is differentiable and, in principle, the gradient of the approximate loss might provide a good heuristic for grafting.

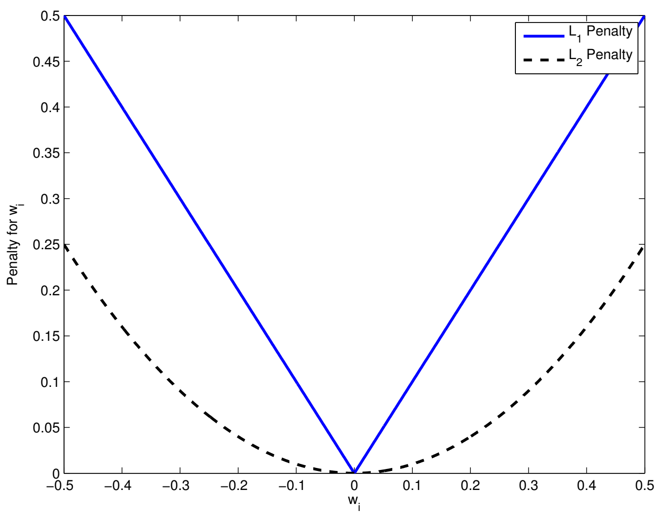


Figure 3.1: The penalty associated with non-zero values of a single weight w_i under an ℓ_1 penalty and an ℓ_2 penalty. The derivative of the ℓ_2 penalty is zero or close to zero in the neighborhood $w_i = 0$. Moving w_i slightly away from zero to improve the objective function $\ell(Y|X)$ incurs almost no penalty under ℓ_2 regularization. Moving w_i away from zero under ℓ_1 regularization immediately incurs a penalty proportional to δw_i. Therefore, w_i will remain zero unless the gradient of the objective function $\partial\ell(Y|X)/\partial w_i$ is large enough to offset the penalty.

non-zero weights and hence result in models with fewer non-zero parameters.

In addition to producing sparse models, the penalized objective function (3.10) also smooths the model, which tends to reduce over-fitting to the training set. That is to say, because the penalty is proportional to absolute magnitude of the weights, smaller weights are favored over larger weights, which results in a more parsimonious model. The penalty term is convex, and therefore the penalized objective function remains convex, although, adding the penalty term means that the objective function is no longer differentiable, which complicates training, as we described in chapter 2.

Figure 3.1 provides intuition as to why training with an ℓ_1 penalty results in models with some parameters exactly equal to zero, but training with other common penalty functions, such as an ℓ_2 penalty, which penalizes by the square of the weights, does not. Consider a single weight in the neighborhood around $w_i = 0$. Under an ℓ_2 penalty, the partial derivative of the penalty is proportional to λw_i. To a first order approximation, the change in the penalty is zero for moving w_i away from zero. With an ℓ_1 penalty, the relevant partial

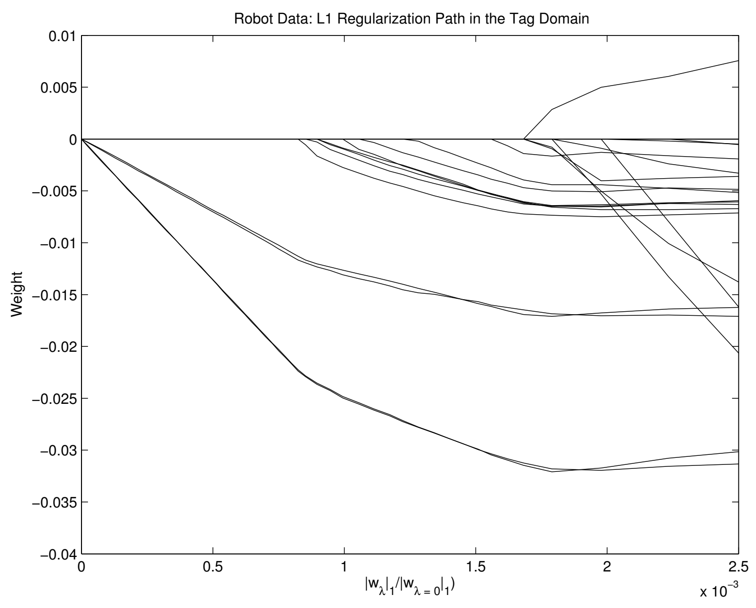


Figure 3.2: A portion of the ℓ_1 regularization path for a CRF trained for activity recognition in the robot tag domain. Each line in the plot represents the weight associated with a feature in the CRF. For high values of the regularization parameter λ (left side), all features have weights of zero and the model contains no features. As λ is decreased, features enter the model by taking on non-zero weights. The horizontal axis corresponds to $|w_\lambda|_1/|w_{\lambda=0}|_1$, that is, the ℓ_1 norm of the weights for each value of λ normalized by the ℓ_1 norm of the weights that result when the model is trained without regularization.

is $\pm\lambda$, so the change in the penalty is proportional to λ. Under an ℓ_2 penalty, a small non-zero derivative in the *unpenalized* objective function will move w_i away from zero. In the ℓ_1 case, λ serves as a threshold and prevents w_i from becoming non-zero to buy small improvements in the unpenalized objective function. We could potentially charge a larger penalty for moving weights away from zero, e.g. an ℓ_0 penalty, which charges a fixed amount per non-zero weight, but that would create a non-convex objective function and significantly increase the difficulty of parameter estimation.

To perform feature selection using ℓ_1 regularization, we train the model using many different values of the smoothing parameter λ and choose among the resulting set of weights using cross-validation or classification accuracy on a holdout set. We then select the features with non-zero weights. We begin with λ_0 large enough that all weights in the model are equal to zero by computing $\lambda_0 = \max|\partial\ell(Y|X)/\partial w_i|$ for $w = 0$. In other words, we compute the gradient at $w = 0$ and take the absolute value of the partial with the largest magnitude for λ_0. We then train the model many times with successively smaller values of λ. We decay λ between iterations according to $\lambda_{i+1} = \alpha\lambda_i$, where α is a positive decay parameter with a value less than 1 such as .9. We initialize training for each new value of λ with the trained weights from the previous value of λ. As Koh et al. noted, this warm start technique dramatically reduces the required training time. Using this warm start technique, we have tested 500 descending values of λ in the time it would take to train the model from scratch five times; reusing previous results makes training orders of magnitude faster. Figure 3.2 shows an example of the first few models that result from this process.

3.2.4 Greedy Feature Induction

In [73], McCallum introduced a CRF specific feature induction algorithm that computes the approximate gain [in the log-likelihood of the training set according to the model] that would result from adding and re-training with each candidate feature. In [67], Liao et al. introduce a modified version of boosting [28], called *virtual evidence boosting*, that uses a smoother gain approximation, which we discuss in chapter 6. In this section, we present the derivation for McCallum's approximate gain computation.

Exactly computing the incremental usefulness of individual features requires a prohibitive amount of computation. However, by making a mean field assumption that removes temporal links from the model either by substituting in true labels for adjacent time steps (i.e. pseudo-likelihood) or by using the most likely labels for neighboring time steps (i.e. a mean field assumption), we can efficiently approximate the incremental usefulness of single features.

We begin with an exact expression for the conditional likelihood of the training set and make a series of approximations to derive the heuristic gain estimate of [73]:

$$p_0(Y|X;w) = \frac{\exp(w^T F(X,Y))}{\sum_{Y'} \exp(w^T F(X,Y'))} \tag{3.11}$$

$$F(X,Y) = \sum_t f(t, y_{t-1}, y_t, X) \tag{3.12}$$

Augmenting the model with an additional feature g that has weight μ yields:

$$p_g(Y|X;w,\mu) = \frac{\exp(w^T F(X,Y) + \mu G(X,Y))}{\sum_{Y'} \exp(w^T F(X,Y') + \mu G(X,Y'))} \tag{3.13}$$

$$= \frac{\exp(w^T F(X,Y)) \exp(\mu G(X,Y))}{\sum_{Y'} \exp(w^T F(X,Y')) \exp(\mu G(X,Y'))} \tag{3.14}$$

$$G(X,Y) = \sum_t g(t, y_{t-1}, y_t, X) \tag{3.15}$$

Optimizing this expression, over both μ and w, is equivalent to regular maximum likelihood training. To speed the gain computation, we will hold w fixed and consider an optimization over μ alone. This is an approximation and will yield a different result than optimizing over both w and μ because we prevent w from changing in response to the inclusion of g. To isolate μ, we multiple the numerator and denominator by the reciprocal of the original normalization constant:

$$Z_0 = \sum_{Y'} \exp(w^T F(X, Y')) \tag{3.16}$$

Multiplying the numerator and denominator yields:

$$p_g(Y|X; w, \mu) = \frac{\frac{1}{Z_0} \exp(w^T F(X, Y)) \exp(\mu G(X, Y))}{\sum_{Y'} \frac{1}{Z_0} \exp(w^T F(X, Y')) \exp(\mu G(X, Y'))} \tag{3.17}$$

$$= \frac{p_0(Y|X) \exp(\mu G(X, Y))}{\sum_{Y'} p_0(Y'|X) \exp(\mu G(X, Y'))} \tag{3.18}$$

The advantage of this form is that it allows us to compute p_0 once and reuse that same value during the gain computations for many candidate features g. Discovering the best μ for each candidate feature is then a one dimensional optimization problem that can be solved using Newton's method or a simple line search rather than a more expensive algorithm such as conjugate gradient, as is required for the multi-dimensional case when w is also updated in response to candidate features.

To reduce the expense of computing the normalization term, McCallum uses a mean field approximation, similar to pseudo-likelihood, that treats the individual time steps in the training sequence as independent. In other words, the approximation removes the edges between time steps in the graph structure of the model and yields the following expression for the approximate likelihood of the training set:

$$\tilde{p}_0(Y|X; w) = \prod_t \frac{\exp(w^T [f(t, \dot{y}_{t-1}, y_t, X) + f(t+1, y_t, \dot{y}_{t+1}, X)])}{\sum_{y'_t} \exp(w^T [f(t, \dot{y}_{t-1}, y'_t, X) + f(t+1, y'_t, \dot{y}_{t+1}, X)])} \tag{3.19}$$

We write $\tilde{p}_0$ instead of p_0 to indicate that we are computing an approximation similar to pseudo-likelihood. As before, we use y'_t to indicate summing over potential labels. In the approximate case, y'_t represents a single label rather than an entire label sequence. Finally, we use $\dot{y}_{t-1}$ and $\dot{y}_{t+1}$ to indicate the values used in place of adjacent labels when computing the feature vectors that involve y_t. These values are held fixed at either the true labels from the training set, which is exactly pseudo-likelihood, or assigned as the most likely labels according to the label-wise marginal probabilities computed by the current model, which amounts to making a mean field approximation. McCallum discusses both variations [73].

As before, we compute the gain in terms of adding a single new feature g with weight μ:

$$\tilde{p}_g(Y|X;w,\mu) = \prod_t \frac{\exp(w^T\dot{F}(X,y_t) + \mu\dot{G}(X,y_t))}{\sum_{y'_t}\exp(w^T\dot{F}(X,y'_t) + \mu\dot{G}(X,y'_t))} \tag{3.20}$$

$$= \prod_t \frac{\exp(w^T\dot{F}(X,y_t))\exp(\mu\dot{G}(X,y_t))}{\sum_{y'_t}\exp(w^T\dot{F}(X,y'_t))\exp(\mu\dot{G}(X,y'_t))} \tag{3.21}$$

$$\dot{F}(X,y_t) = f(t,\dot{y}_{t-1},y_t,X) + f(t+1,y_t,\dot{y}_{t+1},X) \tag{3.22}$$

$$\dot{G}(X,y_t) = g(t,\dot{y}_{t-1},y_t,X) + g(t+1,y_t,\dot{y}_{t+1},X) \tag{3.23}$$

$$\tag{3.24}$$

To isolate μ, we will multiply both the numerator and denominator by the reciprocal of the normalization constant from a single time step in the model for $\tilde{p}_0$:

$$z_0 = \sum_{y'_t}\exp(w^T\dot{F}(X,y'_t)) \tag{3.25}$$

which yields:

$$\tilde{p}_g(Y|X;w,\mu) = \prod_t \frac{\frac{1}{z_0}\exp(w^T\dot{F}(X,y_t))\exp(\mu\dot{G}(X,y_t))}{\sum_{y'_t}\frac{1}{z_0}\exp(w^T\dot{F}(X,y'_t))\exp(\mu\dot{G}(X,y'_t))} \tag{3.26}$$

$$= \prod_t \frac{\tilde{p}_0(y_t|X;w)\exp(\mu\dot{G}(X,y_t))}{\sum_{y'_t}\tilde{p}_0(y_t|X;w)\exp(\mu\dot{G}(X,y'_t))} \tag{3.27}$$

where $\tilde{p}_0(y_t|X;w)$ is an approximate version of the marginal $p_0(y_t|X;w)$. Of course, since the marginals do not depend on μ, there is no reason to avoid the true marginal probability from the current model. We can use the true marginals to improve the quality of the approximation.

In practice, we work with the log-domain version of the above, which, when we substitute the true marginal probabilities (or log probabilities) is:

$$\tilde{\ell}_g(Y|X;w,\mu) = \sum_t \ell(y_t|X;w) + \mu\dot{G}(X,y_t) - \log\left(\sum_{y'_t} p(y_t|X;w)\exp(\mu\dot{G}(X,y'_t)\right) \tag{3.28}$$

The normalization term remains awkward due to the need to take the logarithm of a sum. However, we can use our log-domain addition operator $\bigoplus$ that we defined in chapter 2 to rewrite the expression as:

$$\tilde{\ell}_g(Y|X;w,\mu) = \sum_t \ell(y_t|X) + \mu\dot{G}(X,y_t) - \bigoplus_{y'_t}\Big(\ell(y_t|X) + \mu\dot{G}(X,y'_t)\Big) \tag{3.29}$$

We can further simplify by noting that the first term does not depend on μ or g and omit it to yield an expression for the approximate gain for incorporating g into the model with weight μ:

$$\text{gain}(g, \mu) = \ell(X, Y; w, \mu) - \ell(X, Y; w) \tag{3.30}$$

$$\text{gain}(g, \mu) \approx \sum_t \mu \dot{G}(X, y_t) - \bigoplus_{y'_t} \Big(\ell(y_t|X) + \mu \dot{G}(X, y'_t) \Big) \tag{3.31}$$

McCallum also considered an additional method for speeding training. Because we assume that the individual time steps are independent, that means it is easy to compute the approximate gain using only a subset of the training data. Specifically, McCallum considered the subset of examples currently misclassified by the algorithm. As a result, the gain computation grows faster as the model improves and the number of training points used in the gain computation shrinks as the error rate drops.

3.3 Experiments with Synthetic Data

We used experiments with synthetic data to illustrate the differences between the three feature selection methods.

3.3.1 Synthetic Data with Weakly Relevant Features

We created synthetic data that contained a mix of strongly relevant and weakly relevant features. We generated label sequences using a first-order Markov transition matrix with a 75% chance of self-transitions between binary labels. We generated an observation sequence where each individual observation vector x_t contained k groups of n correlated continuous values. We sampled the n values in each group so that one value is strongly relevant and the remaining $n-1$ values are weakly relevant. Each observation vector x_t contained k strongly relevant values and $(n-1)k$ weakly relevant values.

We sampled the individual values in the observation vector from the three normal distributions shown in figure 3.3. We drew the values so that they were either informative, i.e. they provided information about the true label y_t, or uninformative. Informative observations were drawn from $N(\mu_T, \sigma^2)$ or $N(\mu_F, \sigma^2)$, depending on the correct value (true or false) of the label y_t. To provide an ambiguous observation, we drew from the average of the two informative distributions $N(\frac{\mu_T+\mu_F}{2}, 2\sigma^2)$.

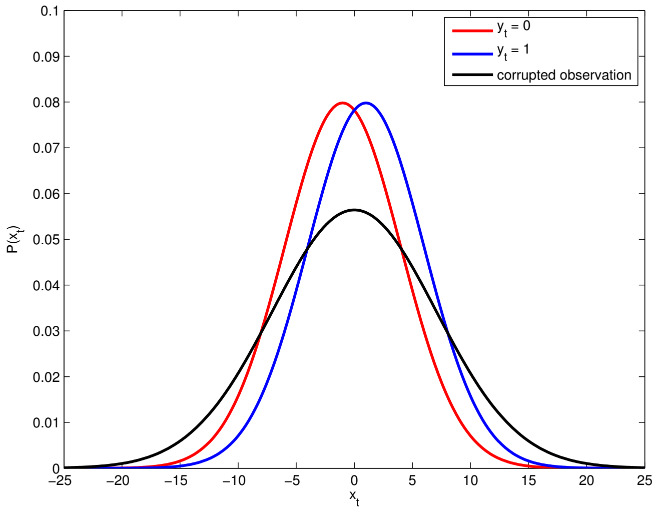


Figure 3.3: We used three Gaussian distributions as the emission model to generate observations in the synthetic data set. In the synthetic data, the observation at a given time step t is either informative or uninformative. For simplicity, although x_t is a vector, we use x_t here as if it were a scalar value. In the case of informative observations, the observation x_t is sampled from either $P(x_t|y_t = \text{false}) = N(\mu_F, \sigma^2)$ or $P(x_t|y_t = \text{true})N(\mu_T, \sigma^2)$ depending on the value of the binary label y_t. In the case where the observation is corrupted by noise and carries no information about the label, the observation x_t is sampled from $P(x_t) = N(\mu_C, 2\sigma^2)$. In our experiments $\mu_F = -1, \mu_T = 1, \mu_C = 0, \sigma^2 = 25$.

We created a group of n correlated observations by drawing a single value v_0 from the informative distribution corresponding to the true label y_t. We copied v_0 into the remaining values $v_1 ... v_{n-1}$. The n observations were *not* independent samples from the informative distribution, they were copies of a single sample. We then drew a single uniform random value $c \sim U(0, 1)$. We used this single random value to determine which of the $n - 1$ weakly relevant observations to replace with an uninformative observation. For each of the $n - 1$ values $v_i, i \in [1, n - 1]$, we replaced v_i with a sample from the non-informative distribution if $.05i < c$. Because we use the same random sample c to test all $n - 1$ correlated values and all n values were originally copies of the same sample from the informative distribution, v_i is at least as informative as v_j for $i < j$. Since v_0 is never replaced with an uninformative value, it strictly dominates the other copies and renders them weakly relevant.

3.3.2 Weakly Relevant Features with Grafting

Grafting uses the gradient of the log-likelihood to select which features to add at each iteration. One of the parameters that we control is the number of features that we add before retraining the model and recomputing the gradient. If we add only a single new feature before retraining the model and recomputing the gradient, we will always take into account the effect of previously selected features when computing scores for candidate features. If we add m features at a time, the scores for the m features are not independent. In domains with many weakly relevant features, selecting features via grafting with high values of m creates larger models.

We use synthetic data with many weakly relevant features to illustrate this point. Using the scheme outlined in the previous section, we generated sequences that contained $k = 10$ strongly relevant features and $(n-1)k = 40$ weakly relevant features, i.e. $n = 5$. We used grafting to train CRFs that modeled the observations as independent, uni-variate normal distributions and varied the number of features that we added at each time step.

Figure 3.4 shows the error rate on test data versus the number of features in the models for grafting when we add 1, 8, or 32 features before retraining and recomputing the gradient. As we expected, there is a clear trend towards larger models when we add more features per iteration. Adding more than a single feature at a time reduces the total amount of computation required for feature selection, because the computational cost for grafting mostly comes from retraining after adding new features, but the price for reduced computation is less sparse models and slightly increased error rates due to over-fitting.

3.3.3 Weakly Relevant Features with the Mean Field Heuristic

When we use the mean field heuristic to score candidate features, we must choose between computing the score on the full training set or only the mislabeled portion of the data. We also must choose whether or not to use the true labels from the training set (pseudo-likelihood) or the most likely labels according to the model. The first decision, whether or not to use all of the training set, affects how much computation is required to score candidate features. The second decision, how to select label values for adjacent time steps, does not appreciably change the computational requirements of scoring candidates, but it can potentially change the quality of the scoring metric.

Figure 3.5 shows that there is a large difference between the variants that use the entire training set and the variants that compute the approximate gain using only mislabeled

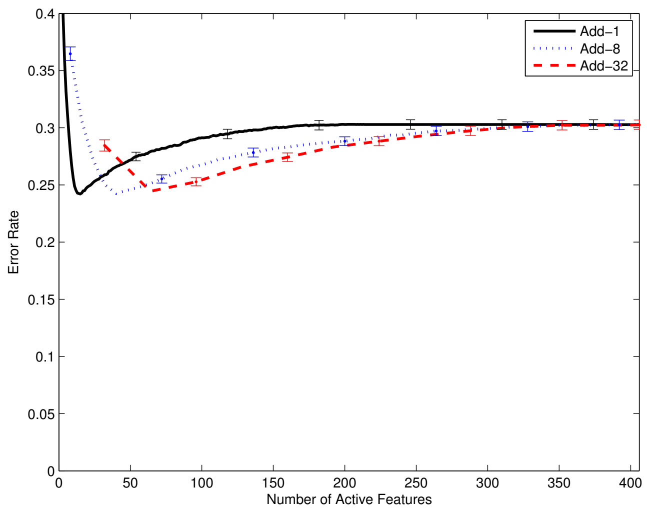


Figure 3.4: Error rate versus model size as we vary the number of features that we add before retraining and recomputing the gradient in grafting.

examples. Using the entire training set reduces the number of features in the model where the test loss is minimized and slightly lowers the best error rate for the model. There is no significant different for pseudo-likelihood versus mean field when the entire training set is used, but for the mistakes only case, the mean field algorithm appears to select slightly more features than pseudo-likelihood, although the two variants very similar performance.

Varying the Amount of Training Data

We use our synthetic data with weakly relevant features to compare the performance of the three feature selection algorithms as the size of the training set is varied. In many domains, it is difficult or costly to acquire large amounts of training data. We desire feature selection algorithms that perform well when little data is available. We compare how well the feature selection methods perform in terms of model sparsity, error rate on test data, and the number of weak features that they include in the model. As before, we use the same synthetic data with $k = 10$ strongly relevant features and 40 weakly relevant features. Based on the results of our previous experiments, we add one feature per iteration to grafting. For the mean field heuristic, we use the entire training set when scoring features and we use the true labels from the training set for adjacent time steps rather than the most likely labels according to the model.

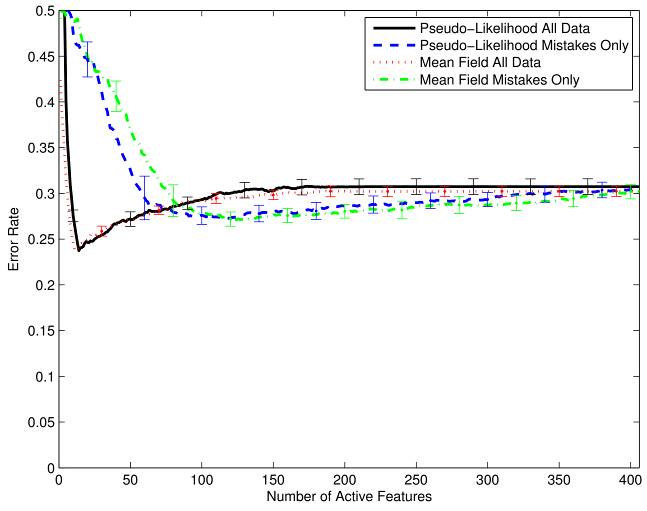
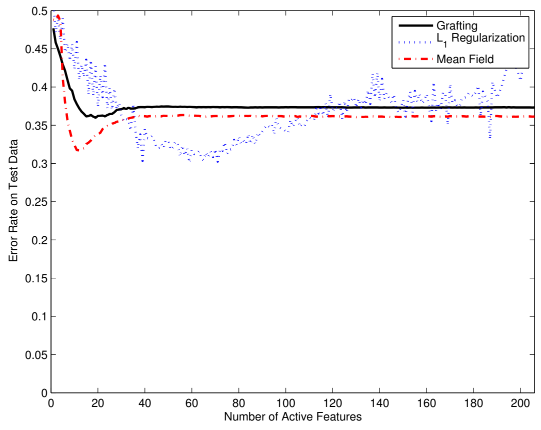


Figure 3.5: Error rate versus the number of features when we perform feature selection using the mean field heuristic to score features. We varied whether the true labels or most likely labels (according to the model) were used for the pseudo-likelihood style approximation. We also varied whether or not only mislabeled training examples were used to compute the approximate gain scores for the features.

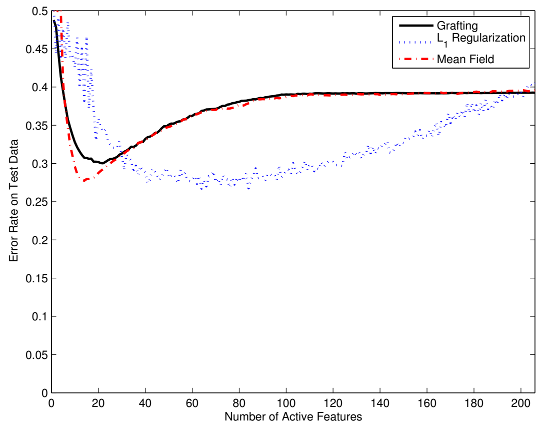
Figure 3.6 shows the average accuracy of the models produced by the feature selection algorithms versus the size of those models for training sequence lengths of 128, 256, 512, and 1024 time steps. When $T = 128$, there is a noticeable gap between the minimum error rate of grafting and the other two algorithms. Greedy algorithms tend to over-fit when too little training data is available and grafting is the most greedy of the three algorithms. Forward selection using the mean field heuristic to score candidate features is also greedy, but the scoring metric is less myopic than the gradient of the log-likelihood. ℓ_1 regularization is the least greedy, because it allows weights to freely enter and exit from the model during training. Furthermore, ℓ_1 regularization smooths the final model, which tends to reduce over-fitting. As the length of the training sequences increases, the gap in the minimum error rate between the algorithms vanishes.

A second noticeable trend in figure 3.6 is that ℓ_1 regularization consistently produces larger models than the other two algorithms. The most accurate models for the other two algorithms consistently contain fewer features than the best models produced by ℓ_1 regularization. ℓ_1 regularization has a larger "sweet spot" than the other two algorithms, which produce accurate models in only a narrow range of sizes. Because ℓ_1 regularization produces smoothed models, it discovers accurate sets of parameters across a wide range of model sizes.

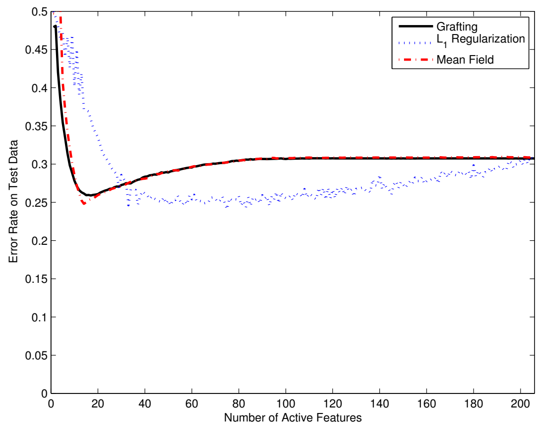
Synthetic Data: Accuracy versus Model Size



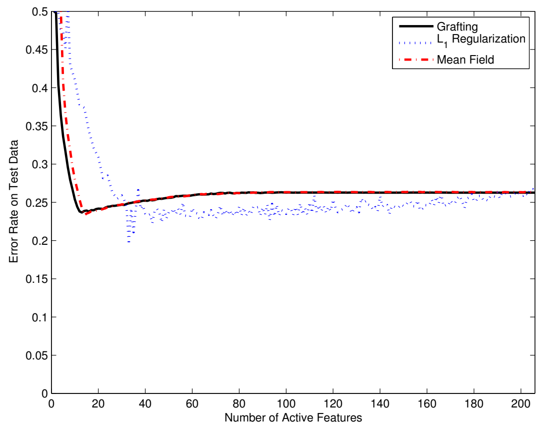
a) Training Sequence Length = 128



b) Training Sequence Length = 256



c) Training Sequence Length = 512



d) Training Sequence Length = 1024

Figure 3.6: Starting from a model with no features, the three feature selection algorithms augment a CRF with additional features during training. In general, the error rate on test data decreases as relevant features are added and then increases as additional features lead to over fitting. The four plots show this behavior as the length of the training sequence varies from 128 (a) to 1024 (d) time steps. In all cases, the length of the test sequence was fixed at 1000 time steps and the curves are averages from slightly more than fifty independent trials where new synthetic sequences were created for each trial.

In practice, we use cross-validation or a hold-out set to choose one of the candidate models produced by feature selection. Figure 3.7 displays the error rates and other properties of the candidate models that we selected rather than of all candidate models. When we look at the error rate versus the length of the training sequence, we see that all of the models begin at approximately 50% error, which corresponds to random guessing to choose a label. The error rates drop as the size of the training set increases, although the error rate for grafting decreases more slowly than for the other two algorithms. The CRFs include features of the form:

$$f = (y_t \overset{?}{=} \text{label}) v_i \tag{3.32}$$

$$f = (y_t \overset{?}{=} \text{label}) v_i^2 \tag{3.33}$$

for each value of the fifty observations $v_i \in x_t$ and each of the two possible label assignments. Looking at the total number of active features in the final models, it is clear that ℓ_1 regularization is less selective than the other two algorithms. As more training data becomes available, it moves towards including all 20 strongly relevant features. The number of weakly relevant features included by ℓ_1 regularization does seem to level off eventually, however. The variance in the size of the models chosen by grafting and using the mean field heuristic increases dramatically when the error rate starts to drop. The two greedy approaches seem to have higher variance when little training data is available.

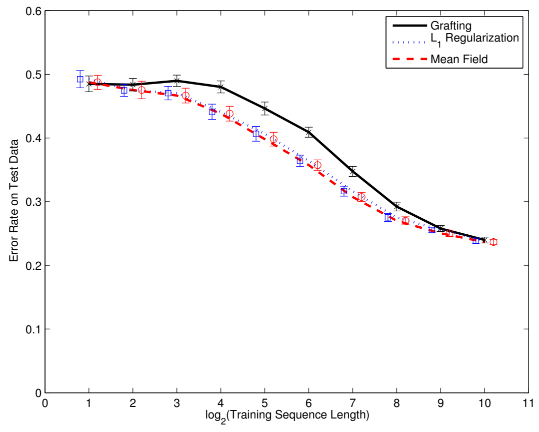
Varying the Number of Weakly Relevant Variables

We tested the performance of the three feature selection algorithms as we increased the number of weakly relevant features in the model. We constructed synthetic data using the same rules as before with k groups of n observations, where $n - 1$ of the observations are weakly relevant. We again used $k = 10$ groups and we varied n, the number of observations in each group.

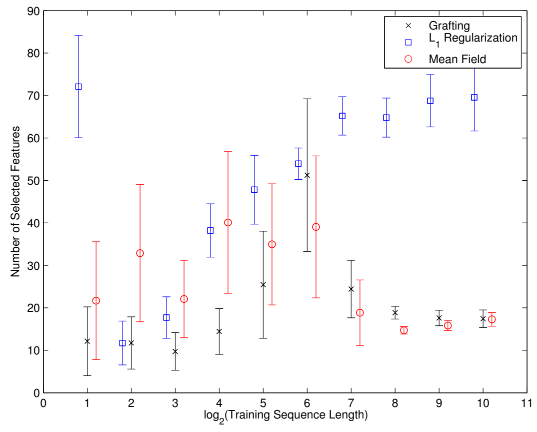
The results in figure 3.8 exhibit the same pattern that we saw as the amount of training data was varied. The two greedy algorithms select fewer features than ℓ_1 regularization and have a more pronounced dip around the best model versus the broad basin of good models discovered by ℓ_1 regularization. The actual value of the minimum loss on test data does not vary strongly with the number of weakly relevant variables. However, the amount of over fitting does increase with the number of weakly relevant variables.

Figure 3.9 summarizes the properties of the models that were selected by minimizing the loss on the holdout data. The error rate does not change significantly with the number of weakly relevant features. There is not a visible trend in the total number of features selected with the two greedy algorithms, but ℓ_1 regularization yields an increasing number of features as the number of weakly relevant variables increases. Breaking down the additional

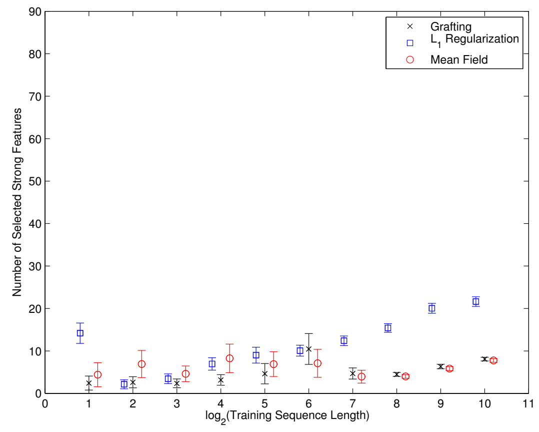
Synthetic Data: Performance of the Selected Model



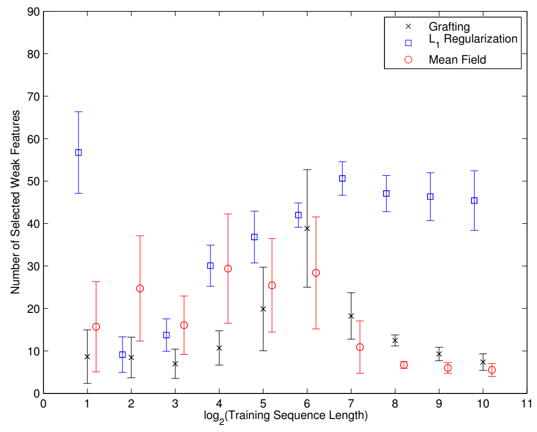
a) Error Rate



b) Total Number of Selected Features



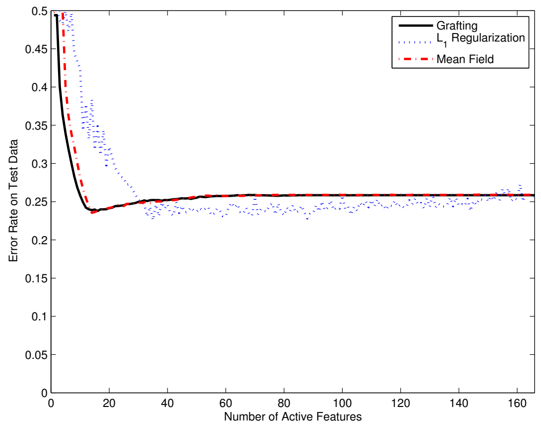
c) Number of Selected Strong Features



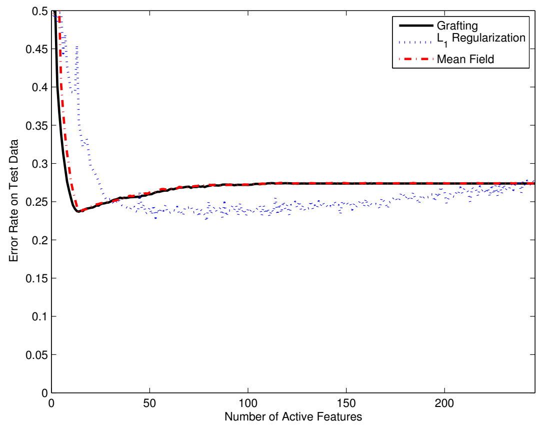
d) Number of Selected Weak Features

Figure 3.7: We used a holdout set to select model parameters for ten different training sequence lengths. We used each algorithm to generate candidate feature sets for each length and chose the models with the lowest error rates on the held-out data. Plot (a) shows the resulting error rates on unseen test data for each training sequence length. The x-axis is $\log_2(sequencelength)$. The remaining three plots characterize the resulting models for each sequence length by displaying the total number of features included in the final model (b), the number of strong features included in each model (c), and the number of weak features included in each model (d). All results are averages over slightly more than 50 independent trials and .95 confidence intervals are shown.

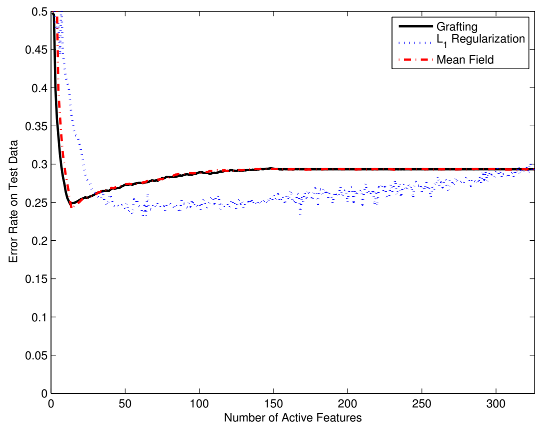
Synthetic Data: Accuracy versus Model Size



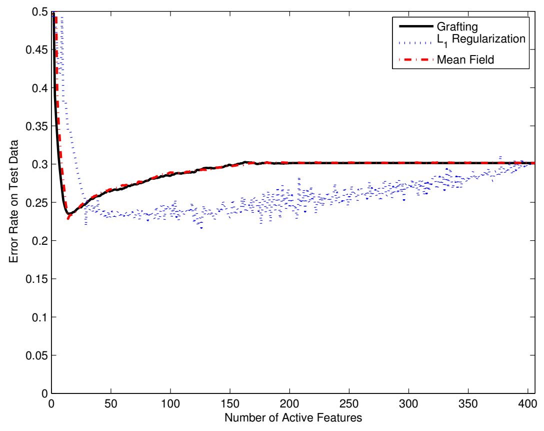
a) 3 Weakly Relevant Variables



b) 5 Weakly Relevant Variables



c) 7 Weakly Relevant Variables

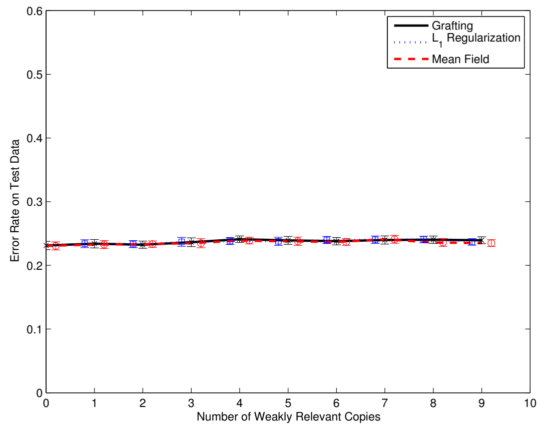


d) 9 Weakly Relevant Variables

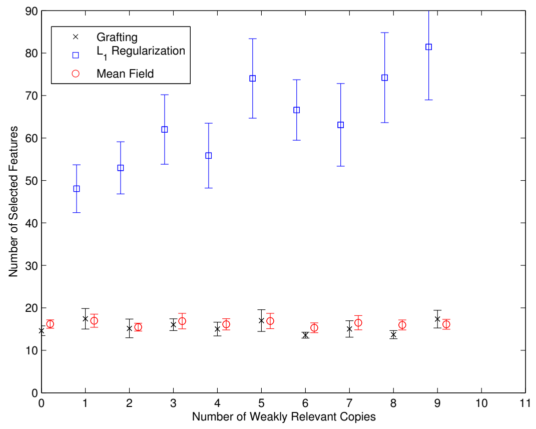
Figure 3.8: Error rate versus the number of weakly relevant observation copies per strongly relevant observation.

features into strongly relevant and weakly relevant, the additional features selected under an ℓ_1 penalty are weakly relevant.

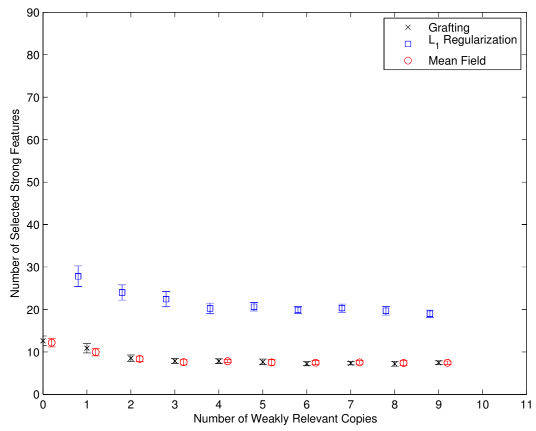
Synthetic Data: Performance of the Selected Model



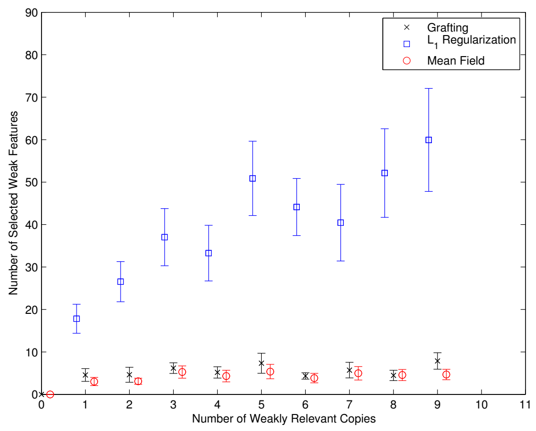
a) Error Rate



b) Total Number of Selected Features



c) Number of Selected Strong Features



d) Number of Selected Weak Features

Figure 3.9: We varied the number of weakly relevant copies that we included for each strongly relevant observation and report error rates as well as model composition as the number of copies is increased from 1 (2 observations per group, 20 total) to 9 (10 observations per group, 100 total).

3.3.4 Model Selection

For any given function of the observations $g(t, X)$, we can create two types of features: state-linked or transition linked.

$$f_i = (y_t \overset{?}{=} \text{label})\, g(t, X) \qquad \text{State-linked} \qquad (3.34)$$

$$f_j = (y_{t-1} \overset{?}{=} \text{label})(y_t \overset{?}{=} \text{label})\, g(t, X) \qquad \text{Transition-linked} \qquad (3.35)$$

We tested the three feature selection algorithms with respect to their ability to detect whether or not a given function $g(t, X)$ should be used in either a state-linked or transition-linked context.

We created synthetic data with binary labels and continuous observations. We drew our label sequences from a Markov chain with a 75% chance of self-transitions. We sampled pairs of observations:

$$x_t[i] \sim \begin{cases} N(\mu_T, \sigma^2) & \text{if } y_t = \text{true} \\ N(\mu_F, \sigma^2) & \text{if } y_t = \text{false} \end{cases} \tag{3.36}$$

$$x_t[i+1] \sim \begin{cases} N(\mu_T, \sigma^2) & \text{if } y_{t-1} \text{ xor } y_t = \text{true} \\ N(\mu_F, \sigma^2) & \text{if } y_{t-1} \text{ xor } y_t = \text{false} \end{cases} \tag{3.37}$$

$$\tag{3.38}$$

where $\mu_F = -1$, $\mu_T = 1$, $\sigma = 5$. I.e., the observation vector contains pairs of co-variates that carry information about the current label and the parity of the previous and current labels respectively. We always added observations in pairs, one state-linked observation paired with one transition-linked observation, and varied the number of pairs from between 1 and 10.

The results in figure 3.10 show the error rates of the final models as well as what types of features were incorporated into the final models by the three feature selection algorithms. The error rate decreased as more state-linked and transition-linked pairs of variables were added to the observation vector. Each pair contained two independent and informative observations, so this is what we would expect; adding independent observations improves accuracy. It is not clear why the error rate leveled off rather than continuing to decrease as more features were added.

Figure 3.10 shows the compositions of the models. Because the number of features changed with the number of variables in the data sets, we plot information about the number of correctly parametrized features, i.e., features that use state-linked features for state-linked variables and transition-linked features for transition-linked variables; the number of informative but overparameterized features, i.e., transition-linked features used with state-linked variables; and the number of irrelevant features, i.e., state-linked features coupled with transition-linked variables, as the percentage of each type of feature included in the final model out of the total number of such features available. For example, if there is one state-linked variable and one transition linked variable in the data, there are then 12 correctly parametrized features. Four of the twelve correctly parametrized features are from the prototypes $f = (y_t \overset{?}{=} \text{label}) x_{state}$ and $f = (y_t \overset{?}{=} \text{label}) x_{state}^2$ and the remaining eight are from the prototypes $(y_{t-1} \overset{?}{=} \text{label})(y_t \overset{?}{=} \text{label}) x_{trans}$ and $(y_{t-1} \overset{?}{=} \text{label})(y_t \overset{?}{=} \text{label}) x_{trans}^2$. The model contains eight features that are informative but over-parametrized due to the state-linked variable being used in features that capture state transitions as follows: $(y_{t-1}\overset{?}{=}\text{label})(y_t\overset{?}{=}\text{label})x_{state}$ and $(y_{t-1}\overset{?}{=}\text{label})(y_t\overset{?}{=}\text{label})x^2_{state}$. Finally, the model contains four irrelevant features from the prototypes $f = (y_t\overset{?}{=}\text{label})x_{trans}$ and $f = (y_t\overset{?}{=}\text{label})x^2_{trans}$, which contain no information because it depends on the parity of y_{t-1} and y_t, x_{trans} needs to be considered *both* of those label variables in order to provide useful information. In the plot, if the composition of the data set results in twelve relevant features and three of these are included in the model, the resulting point would be plotted as containing 25%. When the data set contains two state-linked and two transition-linked variables, then there are a total of 24 relevant features and 6 of them would have to be present for the final model to score 25%.

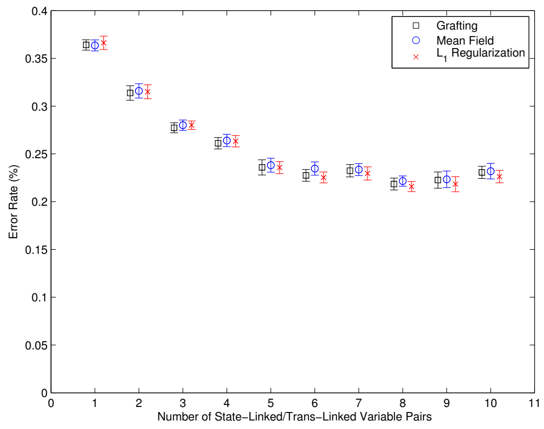
Looking at the trend of correctly parametrized features, grafting selects significantly fewer features than the other two algorithms. The number of included features, as a percentage of available features, remains relatively flat as the number of variables in the observation vector increases. It is possible for models to perform well without including all of the correctly parametrized features because the model is over-parametrized; we include both features for y_t = true and y_t = false, which is not necessary in the binary case. Just like representing the probability of a coin flip as $p_{heads} = .6$ and $p_{tails} = .4$, we can eliminate half of the values. So it is not concerning that not all of the parameters are chosen, although it is of note that grafting produces the same accuracy with fewer features.

If we consider informative, but over-specialized features, i.e., transition-linked features for state-linked observations, grafting produces the smallest models. The mean field heuristic selects the most over-specialized features and ℓ_1 regularization falls between the other two. Turning to the irrelevant feature case, the mean field heuristic and ℓ_1 regularization switch places with the later incorporating significantly more irrelevant features than the other two algorithms. The mean field heuristic's tendency to select transition-linked features for state-linked observations is explained by the fact that we use a pseudo-likelihood style approximation to break temporal links in the mean-field heuristic. Because we use the true values for adjacent labels from the training set, transition-linked features receive artificially high scores, even when coupled with state-linked observations.

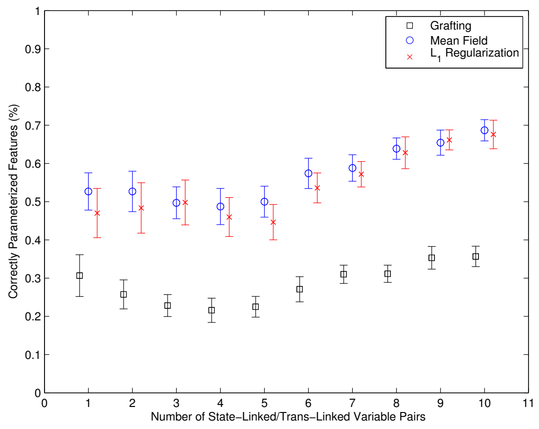
Figure 3.11 shows the order in which the three feature selection algorithms add the different types of features for observations that contain ten feature pairs. We include both variants of the mean field heuristic – the variant that uses the true labels for adjacent time steps and the variant that uses the most likely labels for adjacent time steps. We also show error rate as the gray dotted line as a percentage (out of 100 rather than 1), although it is not included in the key and the label on the y-axis specifies the number of features; error rate is the only one of the trends that decreases.

Grafting (Figure 3.11 (a)) adds a mix of correctly parametrized features (transition-linked

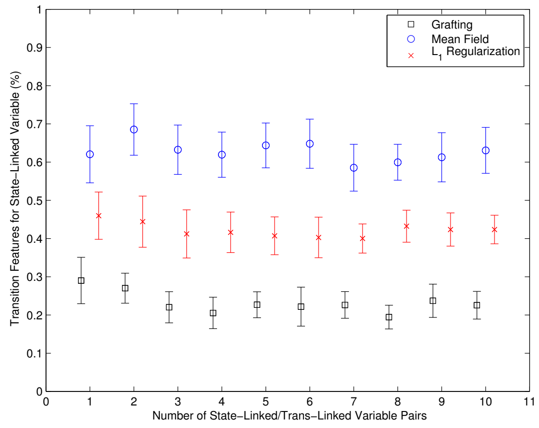
Synthetic Data: Model Selection



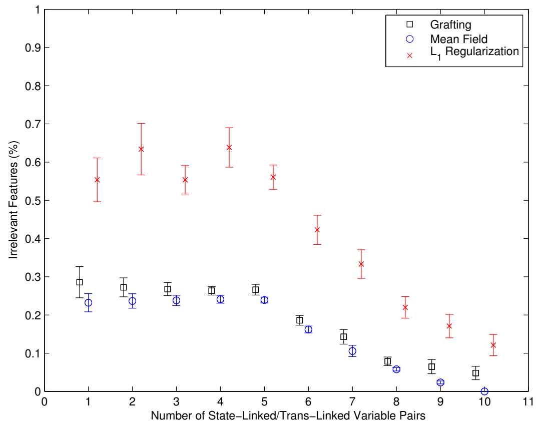
a) Error Rate



b) Correctly Parametrized Features



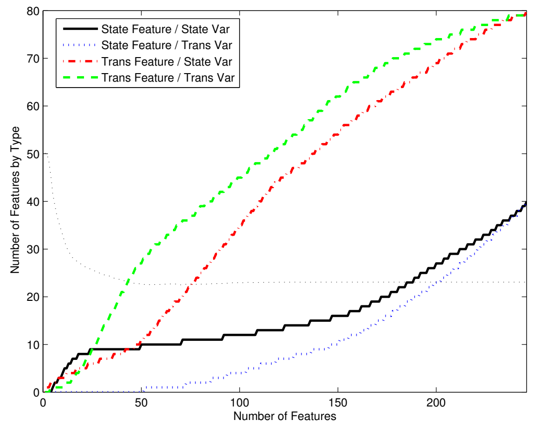
c) Informative-Over-Specialized Features



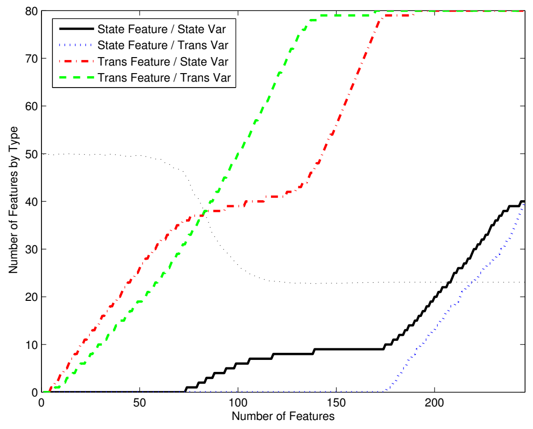
d) Irrelevant Features

Figure 3.10: Selecting State-Linked or Transition-Linked features

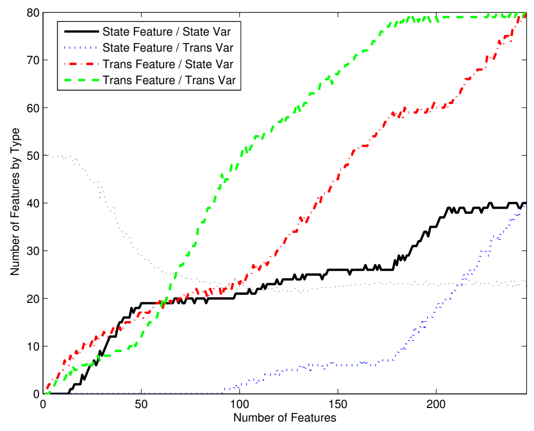
features with transition-linked observations and state-linked features with state-linked observations) as well as useful-but-over-parametrized features (transition-linked features with state-linked observations) until the error rate becomes flat at about 50 features. Then irrelevant features are mixed in with the others to over-fit to noise in the training data. None of the four classes of features are exhausted until the very end. There are 80 of each type of transition-linked feature and 40 of each type of state-linked feature and no feature class reaches the limit until the model approaches its maximum number of features.



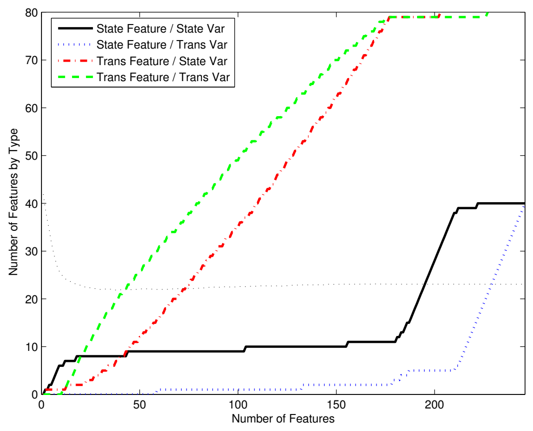
a) Grafting



b) Mean Field: True Label (PL)



c) ℓ_1 Regularization



d) Mean Field: Most Likely Label

Figure 3.11: Selecting State-Linked or Transition-Linked features

ℓ_1 regularization (Figure 3.11 (c)) also adds a mix of correctly parametrized features until the error rate plateaus at around 80 or 90 features, which falls after grafting bottoms out. ℓ_1 regularization is less greedy than grafting and does max out its counts for the correctly parametrized features before the model reaches its maximum size; transition-linked features for transition-linked observations level off before the maximum size, as do state-linked features for state-linked observations.

The mean field heuristic (Figure 3.11 (b) and (d)) changes its selection order depending on whether or not we use the true labels from the training set or the most likely labels according to the current model for adjacent time steps when we break the temporal links in the model. In the pseudo-likelihood case, where we use the true labels (b), the mean field heuristic chooses transition-linked features to the exclusion of state-linked features and the error rate has a long plateau before it drops as state-linked features are added to the model. Pseudo-likelihood over emphasizes transition linked features. When the most likely adjacent labels are used, the error rate begins to drop earlier during the training process and state-linked features are included from the start.

3.4 Tag Experiments

We used our robot tag domain from the previous chapter as a benchmark problem to compare the three feature selection algorithms. We generated data sets using both actual robots and a robot simulator to test how well models trained on simulated data generalized to data from real robots. In both cases, the training, hold out, and test sets each contained ten minutes of data, or since the system operates at 60 hz, 36,000 time steps.

The experiments with regularization required testing many different values of the regularization parameter λ. In the case of training with an ℓ_1 penalty, there is a clear notion of a regularization path starting with a λ set high enough that all features have zero weights and extending through lower values of λ until most or all weights in the model have non-zero weights. To generate such a path, we chose an initial value λ_0 large enough that no features were present in the model and then set $\lambda_{k+1} = .9\lambda_k$ for each succeeding trial. In the case of ℓ_2 regularization, there is not such a clear notion of a regularization path, so we reused the same λ values that were used in the ℓ_1 trials.

3.4.1 Features in the CRF

The CRF included the same features in all of experiments. There were a total of 1200 features in the model, all of which were designed to be relevant to the classification task. We use pos(i) to denote the position vector for robot i and vel(i) to denote the velocity of robot i, as estimated from the difference of two adjacent position observations.

Intercept and Transition Features

$$f = (y_t \overset{?}{=} i) \tag{3.39}$$

$$f = (y_{t-1} \overset{?}{=} i)(y_t \overset{?}{=} j) \tag{3.40}$$

These features allow the model to encode prior probabilities $p(y_t)$ and transition probabilities $p(y_t \,|\, y_{t-1})$.

Raw Observations

$$f = (y_t \overset{?}{=} i)x_t[k] \tag{3.41}$$

$$f = (y_t \overset{?}{=} i)x_t[k]^2 \tag{3.42}$$

These features allow the model to estimate sufficient statistics corresponding to modeling each component of the observation x_t as a one dimensional normal distribution.

Robot Velocities

$$f = (y_t \overset{?}{=} i)\text{vel}(j) \tag{3.43}$$

$$f = (y_t \overset{?}{=} i)(\text{vel}(j))^2 \tag{3.44}$$

Velocities are of interest because the seeker robot pauses in place after being tagged.

Chasing Features

$$f = (y_t \overset{?}{=} i)(\text{pos}(j) - \text{pos}(k))^T vel(k) \tag{3.45}$$

$$f = (y_t \overset{?}{=} i)((\text{pos}(j) - \text{pos}(k))^T vel(k))^2 \tag{3.46}$$

As an example of a more complex feature, the chasing features attempt to capture whether or not robot k appears to be chasing the closest other robot j. It computes the direction from robot k to robot j and then projects the velocity of robot k onto this direction to determine if robot k is moving towards robot j.

Velocity Correlations

$$f = (y_t \overset{?}{=} i)\text{vel}(j)^T\text{vel}(k) \tag{3.47}$$

$$f = (y_t \overset{?}{=} i)(\text{vel}(j)^T\text{vel}(k))^2 \tag{3.48}$$

We included features to capture the correlation between the velocities of robot j and robot k by considering the inner product of their velocities.

Velocity Thresholds

$$f = (y_t \overset{?}{=} i)(\text{vel}(j) \overset{?}{\leq} k) \tag{3.49}$$

These features test whether the velocity of robot j falls below a threshold k. The thresholds k were chosen to range from 5 to 100 mm/sec in steps of 5 mm/sec.

Distance Thresholds

$$f = (y_{t-1} \overset{?}{=} i)(y_t \overset{?}{=} j)((\text{pos}(i) - \text{pos}(j))^T(\text{pos}(i) - \text{pos}(j)) \overset{?}{\leq} k^2) \tag{3.50}$$

$$f = (y_{t-1} \overset{?}{=} i)(y_t \overset{?}{=} j)((\text{pos}(i) - \text{pos}(j))^T(\text{pos}(i) - \text{pos}(j)) \overset{?}{\geq} k^2) \tag{3.51}$$

Distance thresholds k where chosen from 200 mm to 1000 mm in steps of 50 mm. Recall that the robots have radii of 90 mm; distance thresholds ranged from robots touching to almost a meter apart. We also include equivalent features to indicate when the distance was greater than the same thresholds.

3.4.2 Tag Simulation Results

Table 3.1 shows the results of feature selection using data from a robot simulator. The first column lists the feature selection algorithm that we used. We tested both variants of the mean field heuristic. PL indicates the pseudo-likelihood style scoring and ML indicates that we used the maximum likelihood predictions from the model. The second column lists the number of features with non-zero weights in the final model. When we train on simulated data and perform no feature selection, 34 of the 1200 weights remain at zero because those features are unsupported in the noise free training data.

We are interested in how well models trained on data from a simulator generalize to data from real robots. The third column shows the mean error rate for the models when evaluated against test data from the simulator. The fourth column provides the error rate when the test data comes from real robots. There is little variation in error rate when we test against simulated data. No feature selection performs the worst, due to over-fitting. The most greedy approach, grafting, has the second highest error rate. There is a large difference when we compare how well the models generalize to real data. In every case, the error rate increases. But for the ML variant of the mean field heuristic, it increases quite sharply from 3.8% to 56.8%. The weights chosen by the mean field heuristic when trained on simulated data are specific to simulated data. This high level of over-fitting may be because feature selection depends on which labels are the most likely according to the model and feature selection further reinforces biases in the model.

Table 3.1: Tag Simulation Results

Selection Algorithm	Number of Features	Unsmoothed Simulation	Unsmoothed Robots	Retrained with ℓ_2 Smoothing Simulation	Retrained with ℓ_2 Smoothing Robots
None	1164	8.8	26.7	3.4	N/A
Grafting	37	6.2	12.5	2.2	7.5
Mean Field (PL)	165	1.6	13.5	1.4	5.9
Mean Field (ML)	132	3.8	56.8	2.6	4.6
ℓ_1 Reg.	161	1.8	8.2	3.2	7.4

The error rates when we test on simulation data suggest that the models are over-fitting. We retrained the models using only the features chosen by feature selection and an ℓ_2 penalty for smoothing. The results in the 5th column of table 3.1 show that a smoothed model, containing only the selected features, outperforms the original model for all of the algorithms except for ℓ_1 regularization. ℓ_1 regularization does not require additional smoothing to reduce over-fitting.

We also retrained the models, using only the selected features and with ℓ_2 smoothing on real robot data. When we retrain on robot data, the high error rate from the mean field heuristic vanishes. The large error from before was due to the values of the weights, not the selected set of features. Using the features and weights generated by training on simulated data produces a very high error rate when we test on real robot data. The error rates for the other models also decrease slightly from when we used the original weight parameters with the real robot test set. Even ℓ_1 regularization sees a benefit from retraining on the real data (and choosing a model with a holdout set that is also composed of real robot data).

3.4.3 Tag Real Data Results

We performed the same set of experiments as above using real robot data for training and feature selection and the results are shown in table 3.2. With real robot data, no feature selection assigned non-zero weight to all 1200 candidate features; there were no unsupported features with noisy, real data. Grafting chose very few features (11) for activity recognition from the real data. The additional noise inherent in real data potentially results in more over-fitting and may explain why grafting chose so few features; past 11 features, grafting may have chosen features that matched the training data closely, but did not generalize well to the holdout or test sets.

When we used the chosen models to predict labels for simulated data, except for the no

Table 3.2: Tag Robot Results

Selection Algorithm	Number of Features	Unsmoothed Simulation	Unsmoothed Robots	Retrained with ℓ_2 Smoothing Simulation	Retrained with ℓ_2 Smoothing Robots
None	1200	23.5	16.1	N/A	9.7
Grafting	11	3.3	7.0	2.8	7.0
Mean Field (PL)	190	3.0	7.6	0.8	7.7
Mean Field (ML)	190	13.8	25.6	1.5	7.2
ℓ_1 Reg.	227	7.1	5.9	2.8	7.3

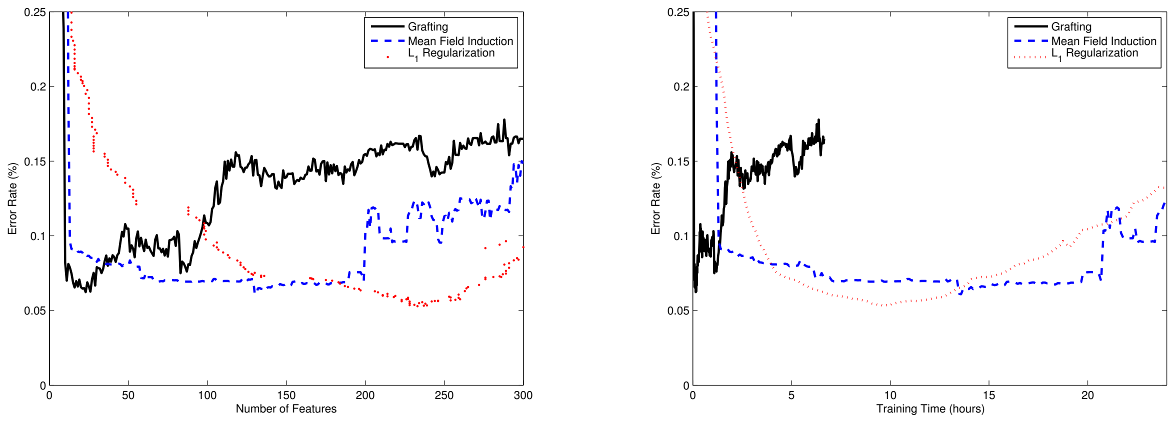
feature selection case, the error rate decreased. The simulated labels are easier to predict, so generalization works well moving from a hard problem to an easier problem. Retraining the models with only the selected features and an ℓ_2 penalty for smoothing decreased the error rate for no feature selection and the ML variant of the mean field heuristic, the two algorithms that over-fit the worse. Grafting chose so few features that smoothing did not affect the error rate and ℓ_1 regularization, as before, was harmed by retraining with an ℓ_2 penalty.

Figure 3.12 (a) shows two different views of feature selection using the three algorithms with the real robot data set. The first plot shows the error rate versus the size of the models. The error rate for grafting is not smooth and contains many local minima, which suggests that grafting is not adding features in the optimal order. On the real data, grafting chose very few features, possibly because the gradient is a poor heuristic for choosing which feature to add next and grafting quickly begins to over-fit. The error rate for ℓ_1 regularization follows a much smoother path. The mean field heuristic (for the pseudo-likelihood case) selects an intermediate number of features.

Figure 3.12 (b) shows the error rates of the candidate models versus the cumulative training time to generate each candidate model. Grafting, with its simple, greedy heuristic requires far less time than the other two algorithms. ℓ_2 regularization takes longer than grafting, but produces lower error rates, albeit from less sparse models. The mean field heuristic takes the longest of the three methods to reach its minimum error rate.

3.5 Chapter Summary

We have provided an empirical comparison of three feature selection algorithms for conditional random fields. Comparing the algorithms along the metrics that we outlined earlier,



a) Error Rates vs. Size b) Error Rate vs. Training Time

Figure 3.12: Error rate versus the number of features in the model and error rate versus training time for the three feature selection algorithms.

we found:

Accuracy and Sparsity Grafting, the greediest of the algorithms selects the fewest features. To reduce over-fitting, models produced by grafting should be retrained with an ℓ_2 penalty. ℓ_1 regularization produced significantly larger models than grafting, but those models tended to have slightly lower error rates on test data than smoothed models generated by running grafting and then retraining with an ℓ_2 penalty.

Data Efficiency We noticed a correlation between more greedy algorithms and higher error rates for limited training data.

Time Efficiency Grafting, with its simple heuristic, is the fastest of the methods. The mean field heuristic, which also computes scores for individual candidate features, is the slowest algorithm because its scoring metric is many times slower than the gradient computation that grafting uses to score the features.

Generalization ℓ_1 regularization showed the best performance in generalizing from a model trained on simulator data to classifying real robot data. It is unclear which variant of the mean field heuristic we should use. The pseudo-likelihood variant tended to over emphasize transition-linked features with synthetic data. And the most likely neighbor variant performed poorly on the tag data sets.

Model Selection All of the feature selection methods worked, to some extent, for model selection. The pseudo-likelihood variant of the mean field heuristic performed the worst because it over emphasizes transition features due to the assumption that it uses to break temporal links for its gain computation.

Chapter 4

Multiple Robots and Many Features

In this chapter, we turn to the problem of feature selection in multi-robot domains with many features. We use data recorded by the CMDragons robot soccer team during the games of the RoboCup 2007 world championship to compare the effectiveness of grafting, ℓ_1 regularization, and the mean field heuristic for feature selection. To cope with the complexity of activity recognition in a multi-robot domain, we motivate and then introduce relational spatial features. In addition to using these relational spatial features to address the challenges of activity recognition in multi-robot domains, we also consider issues of scalability as we select features from large candidate sets.

4.1 The RoboCup Small Size League

In the Small Size League, two teams, with five robots per team, compete in a twenty minute soccer match. Each team is fully autonomous. Off-board computers make all decisions for each team and control the robots wirelessly. Video data from cameras mounted above the field and short messages from the referee, e.g. announcing penalties or kickoffs, are the only inputs to the system; the human team members cannot touch the robots or off-board computers except during timeouts or halftime. The Small Size League is an example of *centralized control*, by each team's off-board computers, using *global information*, from the overhead cameras.

The playing field is approximately 5 m by 3 m in size, the robots are 18 cm in diameter, and a golf ball is used as a soccer ball. A pattern of colored dots is attached to the top of

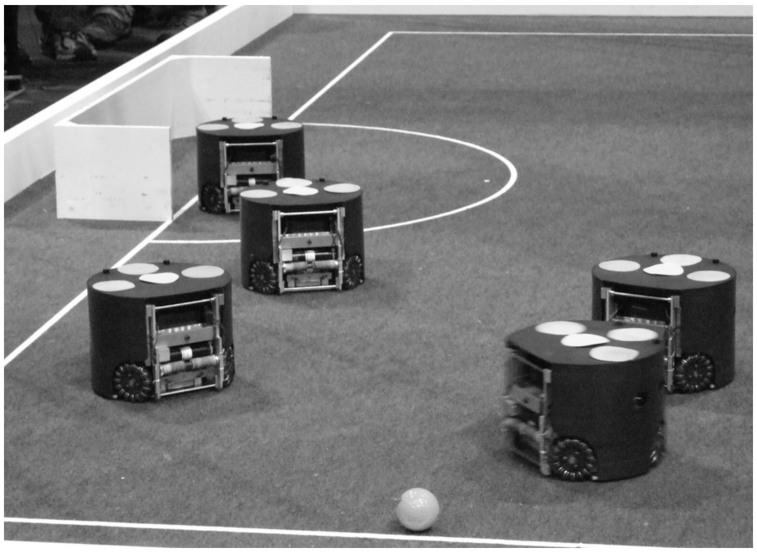


Figure 4.1: *RoboCup Small Size Robots:* In the Small Size League, two teams with five robots per team compete in a soccer match. The robots use rotating rubber bars to grip a golf ball for "dribbling" and can kick the ball at velocities of up to 10 m/s to pass or shoot on goal. The robots are controlled wirelessly by an off-board computer that tracks all ten robots and the ball using overhead cameras mounted above the field. The off-board computer uses the colored circles on top of each robot to identify it and track its position and orientation.

each robot to allow the vision systems to accurately track their positions and orientations, as shown in figure 4.1. In terms of capabilities, the robots are capable of omni-directional movement at velocities as high as 2.5 m/s (5.6 mph). Robots can kick the ball, using a solenoid, driven by capacitors, as fast as 10 m/s (22.4 mph). To manipulate the ball, the robots use rubber coated bars, called dribblers, to rapidly spin the ball and make it adhere to the front of the robot. Using a dribbler, a robot can drive backwards or rotate while maintaining possession of the ball. However, the rules limit the size and configuration of the dribblers so that the ball will come loose if the robot moves too quickly while grasping the ball. Finally, because the body of the robot can occlude the ball from the overhead cameras, the robots have an infra-red beam sensor to detect when the ball is present on the dribbler.

The state of the game evolves rapidly. The robots are able to quickly move the ball from one side of the field to the other and passing between teammates is common. During play, the roles of the robots change quickly as the ball is passed from teammate to teammate. These roles, which we describe below, are the subject of our classification.

Role Name	Role Description
Kickoff	The active role before play begins and when play is stopped, e.g. due to a goal, the robot positions around the ball at a fixed offset.
Mark Opponent	The robot plays man-to-man defense against one particular opponent.
Position for Pass	The robot moves to positions that give its teammate, which has the ball, the opportunity to pass to it.
Receive Chip	The robot moves to where it estimates that a chip pass will land and then matches its velocity to that of the incoming ball to cushion the initial impact.
Position	The robot positions on the field when not executing specialized behaviors to receive passes.
Penalty Kick	The robot takes a penalty kick.
Wall	The robot joins with its teammates to form a wall structure in order to block and opponent kick.
Set-Play-Kick	A tightly coupled play where one robot kicks the ball to a second robot, which deflects the ball into the goal.
Attacker	The primary offensive role, usually reserved for the robot closest to the ball
Defend Circle	Defend the goal at a fixed radius

Table 4.1: Roles of the CMDragons'07 robot soccer team.

4.1.1 Roles

In general, teams in the Small Size League define the behaviors of their robots in terms of a finite set of roles, where the role of a robot identifies its short term purpose or goal. As a specific example, the CMDragons robot soccer team [16] defines robot behaviors hierarchically using the Skills, Tactics, and Plays (STP) architecture [13]. At the base of the hierarchy, skills are the lowest level behaviors of the robot, such as dribbling the ball or moving to a specific field location. At the next highest level of the hierarchy, tactics control the behavior of a specific robot. For example, a defender tactic might combine several skills to maintain a position between the ball and the goal, intercept shots on goal, and kick the ball out of the defense zone when the ball is intercepted. Finally, at the highest level, plays control the behavior of multiple robots simultaneously and allow for coordinated actions, such as passing.

In our discussion, we use the term role to describe robot behaviors at approximately the same level of abstraction as tactics in the CMDragons nomenclature. We use the term role rather than tactic because we are interested in classifying roles across different teams rather than being tied to the specific behavior architecture used by a single team. We classify the roles of the robots rather than recognize individual skills, such as dribbling, because knowledge of opponent roles provides more useful information for making strategic decisions and because skills tend to vary between teams, which makes generalization more difficult. Similarly, we do not classify opponent plays because plays will vary widely between teams and we are interested in responding to individual robots. For example, we would like to distinguish between a robot that is defending its goal and one that is waiting to receive a pass. We need to block passes to the later, but not to the former. In summary, we chose to classify roles because that provides a high enough level of abstraction to provide strategic information about single opponent robots and roles will tend to generalize better than plays.

In principle, each team defines its own unique set of roles. In practice, many roles are shared across teams. We would like to apply a classifier to new opponents to identify roles from this common set. However, in the current work, our training data comes from a single team and we are limited to predicting the roles of the robots from that team. To test generalization with our limited data, we use two games as a training set, a single game as a hold out set to choose model parameters, and test exclusively on data from the final match. The final is never used as a training or hold out set. Table 4.1 lists the roles of the CMDragons'07 robot soccer team. The role of *Goalie* is not included because the goalie never changes roles.

The classification task, which we formalize below, is to recognize robot roles from the available sensor data, which consists of the positions of the robots and the ball. The goal is to create a classifier that can provide useful information to robots that are playing against the team whose roles are being classified.

4.1.2 The Classification Task

The classification task is to map from a sequence of *observations* about the world $X = \{x_1, x_2, ..., x_T\}$ to a sequence of *labels* $Y = \{y_1, y_2, ..., y_T\}$. We use x_t and y_t to refer to observations or labels from a single time step t and T as the length of the sequence. Individual observations x_t are vectors of real values that contain the observed information from a single time step. In our domain, x_t contains 32 real valued elements that specify the positions and orientations of the ten robots as (x, y, θ) and the position of the ball (x, y). The labels y_t are drawn from the set of roles listed in table 4.1 and correspond to the role of a single robot t. Notably, the label y_t does not contain the joint role for all robots, which we discuss in a later section.

4.2 Activity Recognition with Simple Features

In chapter 2, we presented an activity recognition domain based on the children's game of tag. We showed that CRFs can accurately identify which robot is "it" during a tag match using features such as the velocity of the robots and the distances between them. Robot soccer is a significantly more complex domain than tag. The size of the label set increases from three labels, which identify the robot that is "it" in tag, to ten labels, which specify the role of a single robot. The number of robots increases from three, in tag, to ten, in robot soccer. In soccer, the robots interact with objects in their environment, namely the ball and goals. The robots execute a much richer set of behaviors than in tag, one that includes cooperative actions such as passing, and they respond to inputs, which are not available to the classifier, from the referee. Despite the increased complexity of the new domain, we tested simple spatial features such as distances and velocities for role recognition in RoboCup soccer.

We defined the features in terms the distances between objects in the soccer domain. The distances between the objects capture important properties of the domain, such as which robot is the closest to the ball and which robot is closest to its own goal. We identified fourteen important objects in the domain and used the pairwise distances between them as features in the model. The objects that we used were the ten robots, the two goals, the ball, and the center of the field. Therefore, we considered $\binom{14}{2} = 91$ different distances. We also defined features using the velocities of objects, as well as the usual intercept terms and features that encode first-order Markov transition dynamics.

In these experiments, we set aside the issue of feature selection and focus on the classification accuracy of a conditional random field with simple spatial features in order to verify that such spatial features are sufficient for high classification accuracy. We tested our model using data recorded by the CMDragons'07 small size team during RoboCup 2007. We used data from the final match as a test set and data from a semi-final, quarter-final, and one round robin game as the training and hold out sets. We used the hold out set to choose λ, the smoothing parameter, for ℓ_2 regularization. While the CMDragons'07 system operates at a frame rate of 60 hz, we sub-sampled from the full data sets at a rate of 2 hz to reduce the amount of data and speed the training process.

Table 4.2 shows the error rate on test data when we use pair-wise distances and object velocities as features in a CRF. Simple spatial features, despite the fact that they capture many relevant properties of the RoboCup domain, do not allow a CRF to achieve a low classification error rate. In the tag domain, distance and velocity features fully captured the relevant properties of the domain and made role information accessible to the classifier. In

Feature Selection	Smoothing	Error Rate
None	No smoothing	29.1%
None	ℓ_2 smoothing	25.4%

Table 4.2: CRF error rates when predicting the roles of the CMDragons'07 team using pair-wise distance and velocity features.

robot soccer, while the distances and velocities do carry information about the role of the robots, more complex features, which incorporate additional domain knowledge, are required for accurate classification.

4.3 Scaling to Many Features

The approach that we take in this chapter is to generate a large set of candidate features from a much smaller collection of manually specified templates, which we describe in later sections. We rely on feature selection to efficiently select the most relevant subset of features from the large pool of candidates. In this section, we consider the issues involved in scaling feature selection to large numbers of candidate features.

4.3.1 Lack of Sparsity in Features in Activity Recognition

In the Natural Language Processing (NLP) community, conditional random fields with hundreds of thousands or millions of features are common, e.g. [97]. The NLP community is able to use such large models because their features are extremely sparse. For example, if we treat a text document as a sequence, the words in the document are the discrete observations and the features in the CRF take a form similar to either of

$$f_i(t, y_{t-1}, y_t, X) = (y_t \overset{?}{=} \text{label})(x_t \overset{?}{=} \text{word}) \tag{4.1}$$

$$f_i(t, y_{t-1}, y_t, X) = (y_{t-1} \overset{?}{=} \text{label})(y_t \overset{?}{=} \text{label})(x_t \overset{?}{=} \text{word}) \tag{4.2}$$

These features are extremely sparse. The cardinality of the label set depends on the application. For example, the label set in the CoNLL-2000 shared task, which required segmenting text documents into phrases, the label set cardinality was approximately 20 classes [107]. The cardinality of x_t depends on the size of the dictionary, which might include ten thousand or more common words. With a label set cardinality of 20 and a 10,000 word dictionary, the feature prototype in (4.1) generates 200,000 weights in the model. Equation (4.1), again using 20 labels and a 10,000 word dictionary, creates 4,000,000 weights in the model. When evaluating the feature vector, only one feature out the 200,000 features for (4.1) and only one feature out of the 4,000,000 for (4.2) have a non-zero value, which means that we can very efficiently compute the dot product of the feature vector and the weight vector, because most of the values in the feature vector are zero.

In activity recognition, we do not have discrete variables with enormous cardinalities. Instead, we have features like

$$f_i(t, y_{t-1}, y_t, X) = (y_t \overset{?}{=} \text{label})g(t, X) \tag{4.3}$$

$$f_i(t, y_{t-1}, y_t, X) = (y_{t-1} \overset{?}{=} \text{label})(y_t \overset{?}{=} \text{label})g(t, X) \tag{4.4}$$

where $g(t, X)$ is either a real valued function or, in some cases, a binary feature or discrete variable with a cardinality much more modest than 10,000 dictionary words. There is a limited degree of sparsity due to the labels – using our label set size of 20, this accounts for a 20 or 400 fold reduction in the work required to compute a dot product, but it falls far short of the 200,000 or 4,000,000 fold reductions in computational expense seen with our text classification example.

4.3.2 Conditional Random Fields: The Price of Training

We train conditional random fields to maximize the conditional log-likelihood of the training set using iterative optimization algorithms, such as conjugate gradient [87] or L-BFGS [70]. The algorithms begin each iteration by evaluating the conditional likelihood and its gradient. They use the gradient to compute a search direction, usually by combining the gradient with previously stored information, and perform a line search to find the optimal step size in the search direction. The line search evaluates the function at a series of points but does not compute the gradient at those points. The computational cost for each iteration of training is the cost of one gradient computation, to compute the search direction, plus the cost of potentially many function evaluations (in practice, on the order of 10 to 50 evaluations with the implementation of L-BFGS that we use), for the line search. The overhead of combining the gradient with previously stored information to form the search direction is negligible; virtually all of the computational work goes into computing the value of the function or the gradient rather than working with those quantities inside of the optimization algorithm.

We use belief propagation to compute the conditional log-likelihood. Because the labels form a linear chain, belief propagation is equivalent to the forward-backward algorithm for hidden Markov models [91] and requires

$$O(\text{num-labels}^2 \cdot \text{sequence-length}) \tag{4.5}$$

operations. However, unlike the HMM case, where we assume evaluating $P(x_t|y_t)$ and $P(y_t|y_{t-1})$ is $O(1)$, our operations are dot products between the weight vector and the feature vector, so a more accurate description of the computational complexity includes a term for the number of weights:

$$O(\text{num-labels}^2 \cdot \text{sequence-length} \cdot \text{num-weights}) \tag{4.6}$$

Equation (4.6) also provides a bound on the computational cost of computing the gradient. The cost for computing the gradient comes from computing the same series of dot products that are required to evaluate the function and from accumulating on the order of $O(\text{num-labels}^2 \cdot \text{sequence-length})$ scaled feature vectors to compute the expected values of the features under the model.

So far, we have ignored sparsity in our discussion of computational cost. There are two sources of sparsity that we must consider. First, the feature vector, while not as sparse as in natural language processing domains, does possess a degree of sparsity. Because virtually any feature imaginable includes a factor of $(y_t \overset{?}{=} \text{role})$, at most num-weights/num-labels features have non-zero values in any given feature vector. The second source of sparsity is from the weight vector during feature selection. In the early stages of feature selection, when many features are inactive, the vast majority of the weights are exactly equal to zero. We can exploit the sparsity in w to dramatically speed the computation of the dot products. A tighter bound on the complexity of computing the conditional log-likelihood of the training set is

$$O(\text{num-labels} \cdot \text{sequence-length} \cdot \text{num-non-zero-weights}) \tag{4.7}$$

When computing feature expectations during the gradient computation, we can exploit the sparsity of the feature vector, but not the weight vector, during the scaled accumulate step; we must use the full number of features in the bound rather than the number of non-zero weights.

$$O(\text{num-labels} \cdot \text{sequence-length} \cdot \text{num-weights}) \tag{4.8}$$

Evaluating the conditional likelihood is extremely efficient for sparse w, but sparse weights do not speed the accumulate step of the gradient computation. In the following sections, we consider the implications of the fact that function evaluations scale to high dimensions with sparse weights but gradient evaluations do not.

4.3.3 Scaling Grafting

The models that we train during the process of grafting are sparse. Each iteration of grafting begins with a gradient evaluation that includes all features in the model, but once we use the full gradient to choose a new feature, the model that we train is sparse. In the first iteration of grafting, the model contains one feature. In the second iteration, it contains two features, and so forth. As we described in section 4.3.2, we can use the sparsity of the weight vector to speed evaluations of the conditional log likelihood. In addition, since the majority of the features are inactive, and therefore their weights are constrained to remain zero, we can also avoid computing the gradient of the conditional likelihood with respect to the inactive weights because we do not need to include them in the search direction. Ignoring inactive weights while retraining the CRF between iterations of grafting makes grafting tractable for large candidate sets of features.

In our experiments for RoboCup domain, described later in this chapter, the average time to compute the full gradient for 92,310 features is 4.6 seconds. At the start of each iteration, grafting computes the full gradient in order to choose which feature to add to the model. Once the new feature is added, grafting retrains the model for 25 iterations of L-BFGS, which means that grafting computes the gradient 25 times per iteration. If we generate candidate models with between 1 and 500 features by adding a single feature each time, we need to perform 12,500 iterations of L-BFGS. If we computed the full gradient during the 25 iterations of L-BFGS training, we would need approximately 16 hours of computer time. We are neglecting the cost of the function evaluations, so the 16 hour figure is a severe underestimate. If we used a candidate set of one million features, we would need approximately 7 days of computer time; the cost of computing the full gradient grows linearly with the number of features.

If we avoid computing the gradient with respect to the inactive features, each iterations of L-BFGS takes 1.2 seconds and 500 iterations of grafting takes 4.2 hours. Each iteration of grafting includes 25 iterations of L-BFGS. We performed 12,500 gradient computations and between 10 and 50 times that many function evaluations in 4.2 hours (measured time) versus 16+ hours (conservative estimate) for the gradient computations alone. We reduced the time for a full iteration of L-BFGS, including a gradient computation and 10 to 50 function evaluations, to less than the time for a single gradient computation over the full feature vector. To train models with up to 500 features under grafting, we evaluate the full gradient 500 times rather than 12,500 times, which means that grafting scales very well if we limit the size of our final model to a modest number of features.

4.3.4 Scaling the Mean Field Heuristic

The computational requirements for the mean field heuristic follow the same pattern as the requirements for grafting. As with grafting, we can divide the computational cost between two phases of the algorithm, namely selecting which feature to add to the model and retraining the model with the new feature included. Because the mean field heuristic is also a greedy forward selection algorithm, retraining the model has low computational requirements if we exploit the sparsity of the candidate models and limit the size of the final model. Unlike grafting, which chooses new features via a single gradient evaluation, the cost of selecting new features using the mean field heuristic is prohibitive.

Section 3.2.4 describes how the mean field heuristic computes scores for individual features in detail. Briefly, the mean field heuristic computes the approximate improvement in the conditional log-likelihood of the training set that would result from adding a single additional feature to the existing set of features. We compute the approximate gain for each inactive feature and add the feature with the largest gain to the model. The gains are approximate because we assume that the labels at each time step are independent and we hold the weights of the existing features fixed and only adjust the weight of the candidate feature whose gain is being estimated.

To compute the approximate gain for a single candidate feature, we accumulate log-likelihood estimates across independent time steps. To compute the approximate gain for a single time step, we need to compute a normalization constant by summing over all possible labelings. The computational complexity of computing the approximate gain for a single feature is then

$$O(\text{num-labels} \cdot \text{sequence-length}) \tag{4.9}$$

and computing the approximate gain for all candidate features is

$$O(\text{num-labels} \cdot \text{sequence-length} \cdot \text{num-weights}) \tag{4.10}$$

The time complexity for one round of the mean field heuristic (4.10) is the same as for the full gradient computation when we exploit sparsity in the feature vector (4.8). However, in the case of the mean field heuristic, we actually compute the gain for a single feature several times. We use line search to adjust the weight of the candidate feature as part of the gain computation. Furthermore, the most often repeated operation when computing the gradient is addition. When computing the approximate gains with the mean field heuristic, we need to use log-domain arithmetic, which makes heavy use of transcendental functions, i.e. log and exp. These functions are significantly more expensive to evaluate than addition. Using the mean field heuristic with large numbers of features is tractable in an asymptotic complexity sense, but the constant factor in front of the asymptotic complexity results in a prohibitive amount of computation.

Evaluating the approximate gain for the $92,310$ candidate features that we use with our RoboCup data set requires approximately two hours. Our implementation of the mean field heuristic was carefully optimized using a profiler and uses single-instruction-multiple-data (SIMD) vector instructions from either the AltiVec [22] or SSE [93] CPU extensions to compute the approximate gains. If we limit the maximum model size to 500 features and use this highly tuned implementation to score candidate features, we would spend approximately 40 days of CPU time evaluating features. The candidate evaluations are trivial to parallelize, and our implementation does effectively achieve a linear speedup by using additional processors, but without access to a large cluster of machines, the scalability of the mean field heuristic is limited by the need for independent line searches for each candidate feature at each time step and by the need for log-domain arithmetic.

4.3.5 Scaling ℓ_1 Regularization

Training a conditional random field under an ℓ_1 penalty is tractable for high dimensional feature vectors. Both the objective function $\ell(Y|X)$ and the penalty term $\sum_i |w_i|$ are convex. However, as with the mean field heuristic, training the model, while not intractable in an asymptotic sense, requires significant amounts of computation. In practice, we would like to leverage the sparsity of the candidate models to speed the training process.

In section 3.2.3, we described how to use ℓ_1 regularization for feature selection. Briefly, we choose an initial value for the scalar λ, which controls the degree of smoothing and sparsity in the model, such that $w^* = 0$ is the optimal weight vector to maximize $\ell(Y|X) - \sum_i |w_i|$. Each iteration, we decay λ by multiplying it by some rate constant less than 1, such as .95, and retrain the model to maximize the penalized objective function using the updated value of λ in the penalty term. The result of this training process is a series of candidate models with a generally increasing number of features as λ decreases.

When we train a model under an ℓ_2 penalty, we run a small number, typically 25, iterations of L-BFGS to retrain the model between different values for λ. When we use orthant-wise L-BFGS [3], which handles the non-differentiable ℓ_1 penalty function, we use a larger number of iterations, e.g. 250. We allow extra iterations in the ℓ_1 case because orthant-wise L-BFGS prevents weights from crossing zero in an iteration. If a weight is positive at the start of an iteration, it will not be below zero after the line search. Weights can only change their sign after taking on a value of zero in the preceding iteration.

On average, it takes 4.6 seconds of CPU time to compute the gradient of the conditional log-likelihood for the 92,310 features we use in our RoboCup domain. If we test 100 different values of λ and allow 250 iterations of orthant-wise L-BFGS between changes to λ, approximately 32 hours of CPU time are spent on the gradient computation alone. If we scale up to 1,000,000 features, approximately 320 hours of CPU time are needed. The cost of function evaluations during the line searches is omitted from these figures. We can reduce the number of gradient evaluations by performing several iterations of coordinate-wise training using only a limited set of weights between changes to λ.

During coordinate-wise training, we keep a large portion of the weights in the model fixed at their current values. We then use orthant-wise L-BFGS to maximize the conditional log-likelihood by modifying a smaller number of unfixed weights. The advantage of this approach is that we only need to compute the gradient with respect to the active weights during L-BFGS. So, if we fix 90% of the weights in the model at their current values and vary only 10% of them, then an iteration of L-BFGS requires approximately 10% of the CPU time that it would otherwise need. After a fixed number of iterations or when training with the limited set of features converges, we select a new set of active features and repeat the process.

We choose which features to activate by computing the gradient of the *unpenalized* objective function. We activate an individual feature and vary its weight during training iff

$$w_i \neq 0 \ \vee \ \frac{\partial \ell(Y|X;w)}{\partial w_i} > \lambda \tag{4.11}$$

We do not modify weights that are currently zero if the benefit of moving w_i away from zero, as measured by the derivative of the *unpenalized* conditional log-likelihood, is outweighed by the penalty incurred for moving w_i away from zero, i.e. $\lambda|\delta w_i|$. Repeated iterations of coordinate-wise training will reach the global optimum – weights are always activated when changing them allows for an improvement in the penalized objective function – but with many fewer gradient evaluations for weights that do not improve the model in the local neighborhood of the current weight vector.

4.4 Relational Spatial Features

In order to achieve high classification accuracy, we need to create rich features of the observations. In this section, we introduction relational spatial features, which provide a compact and sparse notation for specifying a set of candidate features. In multi-robot domains in particular, it is useful to design features around the relationships between the objects and robots that are present in the environment. For example, we often want to detect situations where one object is physically close to another. In this section, we motivate and describe a relational language that allows us to specify a small set of feature templates, which we then automatically expand into a large set of candidate features. We rely on feature selection to choose the most relevant features from the large set of candidates.

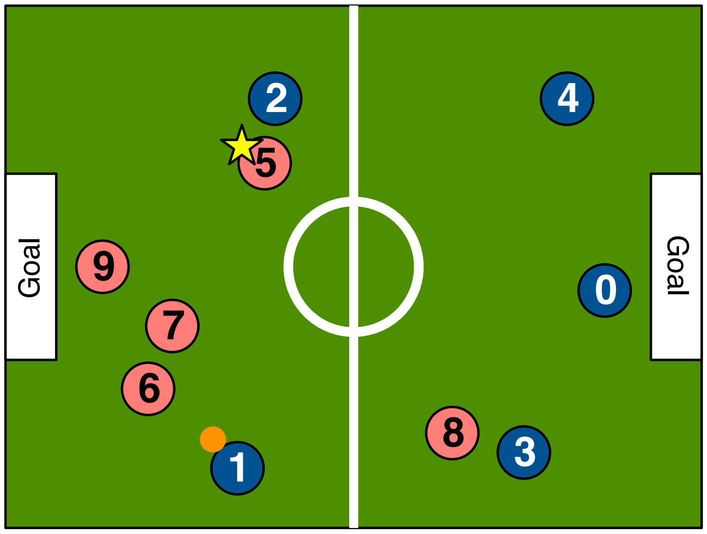


Figure 4.2: *Relational Feature Motivation* An example scenario for classification. Robot 5 is the subject of the classification, as indicated by the star. The ground truth that we are trying to recognize is that robot 5 is executing the *Marking* role. A useful feature for recognizing that robot 5's role is *Marking* is the distance between robot 5 and robot 2; robots closely guard the opponent robot that they are marking.

4.4.1 A Motivating Example

Relational features provide a compact and sparse notation for specifying features in multi-robot domains. The configuration shown in figure 4.2 motivates the use of relational features. In the example, we are predicting the role of robot 5, indicated by the star in the figure. The ground truth that we are trying to recognize is that robot 5 is executing the *Marking* role. Specifically, robot 5 is marking robot 2 to prevent robot 2 from receiving a pass from robot 1. Robots in the *Marking* role are typically close to one of their opponents and that chosen opponent is typically not the closest on its team to the ball. Therefore, the distance between the subject of the classification (robot 5) and the opponent that is both closest to the subject and not the closest opponent to the ball is a relevant feature. We can use a functional notation to succinctly express this distance relationally:

$$\begin{aligned} g_m(t, X) = \text{distance}(&\mathbf{subject}, \\ &\text{closest}(\mathbf{subject}, \\ &\quad \text{xor}(\mathbf{opponents}, \\ &\quad\quad \text{closest}(\mathbf{ball}, \mathbf{opponents})) \end{aligned} \tag{4.12}$$

In our notation, we use functions such as *distance*, *closest*, and *xor* to operate on sets of objects in the environment. We use the set **subject**, which contains one element, to represent the robot whose role is being classified; the set **ball**, also containing one element, that represents the ball; and the set **opponents**, which contains five elements, that represent the robots that are the subject's opponents. The *closest* function computes the distance between its first argument, which is a set and must have a cardinality of one, and each object in its second argument, which is also a set, but one that is not limited to one element. *closest* returns the object from the second set that is closest to the object in the first set. Similarly, the function *distance* requires two sets that each contain one element and it computes the Euclidean distance between them. Equation (4.12) succinctly captures the distance between the subject of the classification and the opponent closest to the subject that is not also the closest opponent to the ball. We can create features from equation (4.12) by testing for particular labels:

$$f_i(t, y_{t-1}, y_t, X) = (y_t \overset{?}{=} \text{label}) g_m(t, X). \tag{4.13}$$

For notational convenience, we omit t and X as arguments to *distance* and the other functions, but the current time t and the observation sequence X provide the information underlying the entire feature evaluation.

Capturing the same distance without using a relational notation is a more difficult matter.

$$\begin{aligned} g_2(t,X) = & \text{distance}(r_5, r_2) \\ & \left.\begin{array}{l} ((\text{distance}(r_0, \text{ball}) \overset{?}{<} \text{distance}(r_2, \text{ball})) \vee \\ (\text{distance}(r_1, \text{ball}) \overset{?}{<} \text{distance}(r_2, \text{ball})) \vee \\ (\text{distance}(r_3, \text{ball}) \overset{?}{<} \text{distance}(r_2, \text{ball})) \vee \\ (\text{distance}(r_4, \text{ball}) \overset{?}{<} \text{distance}(r_2, \text{ball}))) \end{array}\right\} \text{r_2 is not the closest opponent to the ball} \\ & \left.\begin{array}{l} ((\text{distance}(r_2, r_5) \overset{?}{<} \text{distance}(r_0, r_5)) \wedge \\ (\text{distance}(r_2, r_5) \overset{?}{<} \text{distance}(r_1, r_5)) \wedge \\ (\text{distance}(r_2, r_5) \overset{?}{<} \text{distance}(r_3, r_5)) \wedge \\ (\text{distance}(r_2, r_5) \overset{?}{<} \text{distance}(r_4, r_5))) \end{array}\right\} \text{r_2 is the closest opponent to r_5} \end{aligned} \tag{4.14}$$

Equation (4.14) partially captures the relationship described in (4.12) with respect to robot 2, but many separate cases must be considered in order to fully represent the same relationship over all robots. A notation that allows us to manipulate sets of objects using logical and relational operators, e.g. by choosing the robot that is closest to another, provides a compact means for specifying features.

4.4.2 Components of Relational Spatial Features

In this section, we introduce the components of our relational feature specification language. We defer a discussion of the more domain-specific aspects to section 4.5. We built our relational feature specification language around a collection of three primitives:

Object Sets identify the various objects in the environment. At the lowest level are one element sets that contain each object in the world, e.g. the robots, goals, and ball. We build more complex sets from unions of these atomic sets, e.g. we define the set of teammates, which does not include the subject

$$\textbf{teammates} = \bigcup_{i=2}^{5} (\textbf{teammate-i})\,, \tag{4.15}$$

where the subject is **teammate-1** and is therefore omitted.

Selection Operators map from object sets to object sets using information from the observation sequence. In the previous example, we used the selection operator closest. Evaluating

$$\text{closest}(\textbf{ball}, \textbf{teammates}) \tag{4.16}$$

maps from **teammates** to the single teammate that is closest to the ball.

Evaluation Operators map from candidate sets to real values. For example,

$$\text{distance}(\textbf{subject}, \textbf{ball}) \tag{4.17}$$

computes the distance between the robot whose role is being classified and the ball. Evaluation operators may require that some or all of the object sets passed as arguments contain only a single object.

Object sets, selection operators, and evaluation operators allow us to create relational features that evaluate to real valued numbers. These values can be incorporated into more traditional CRF features, for example by squaring them, computing the change in their value over time, or testing whether or not their value exceeds a threshold.

4.4.3 Relational Markov Networks

In this chapter, we consider relational spatial features for activity recognition. There is an extensive literature that discusses relational learning and relational models, e.g., relational

Markov networks [105]. As we discuss in chapter 6, relational Markov networks have been applied to activity recognition [64]. These types of models use the relationships between objects or entities in the domain to define the conditional independence assumptions of the model. These models use relational rules to implicitly define their graphical structure, which they instantiate based on observed data.

In our setting, we use relational rules to specify functions of the observations based on relationships between the objects and entities that we observed. Because we work with conditional random fields, which do not assume any conditional independencies between the observations, our relational feature functions do not change the structure of the graphical model and do not affect the complexity of the inference problem. We use a relational notation to express rich feature functions that capture domain knowledge rather than to build up a rich model structure.

4.5 Relational Features for RoboCup

Previously, we motivated the use of relational features for multi-robot activity recognition. In this section, we describe the specific object sets, selection operators, and evaluation operators that we used to build features for the Small Size RoboCup domain. We then provide a detailed description of the specific features that we used for activity recognition.

4.5.1 Object Sets, Selection Operators, and Evaluation Operators

We built relational features out of three components: object sets, selection operators, and evaluation operators. The end goal was to create real valued functions $g(t, X)$ that make useful information in the observation sequence available to the classifier. In particular, we wanted to capture information about the relationships between the robots and objects in the environment.

Object sets are the basic building block of our language. We created an object set with one element for each robot, the ball, the goals, and the center of the field. Then we defined additional object sets as the union of the sets containing single objects. These unions represent important groupings of the robots, e.g. the subject's teammates, the subject's opponents, the two goals, etc. Table 4.3 lists the complete set of objects that we used when defining our features. While we define objects for individual teammates and opponents, we typically did

Set Name	Set Description
subject	The robot whose role is being classified.
teammate-i	Indicates an individual robot on the subject's team. We use **teammate-1** as a synonym for **subject** and use **teammate-2** through **teammate-5** to refer to the subject's teammates.
teammates	All of the robots on the subject's team with the exception of the subject itself: $\bigcup_{i=2}^{5}$ **teammate-i**
own-team	All of the robots on the subject's team, including the subject itself: **subject** $\cup$ **teammates**
opponent-i	Individual robots on the opposing team. The index i runs from 1 to 5.
opp-team	All robots on the opponent team: $\bigcup_{i=1}^{5}$ **opponent-i**
robots	All robots: **own-team** $\cup$ **opp-team**
other-robots	All robots excluding the subject: **teammates** $\cup$ **opp-team**
own-goal	The goal which the subject's team defends.
opp-goal	The goal which the opponent team defends.
goals	Both goals: **own-goal** $\cup$ **opp-goal**
center	The center of the field, where the ball is placed for kick-offs.
ball	The ball.

Table 4.3: Object sets for defining relational features

not use them. It was more useful to talk of entire teams or the subject's teammates than to refer to individual robots (other than the subject) as teammate-i or something similar.

Objects sets do not depend on information from t or X; object sets are static. Selection operators, on the other hand, make use of information from t and X to operate on the object sets. For example, we often want to pick the element from an object set that is the closest to another object. In this case, the selection operator would use t as an index into X to retrieve the positions for all of the objects in the set and then compute distances using those positions in order to choose which element to return. Table 4.4 lists the selection operators that we used to define our features. For notational convenience, we do not list t and X as arguments to selection operators, but they are nevertheless parameters of each and every selection operator.

Object sets are groups of important objects in the environment and selection operators pick and chose among the elements of those sets. Evaluation operators map from object sets,

Operator	Description
And(a, b)	Returns an object set containing the elements that are present in both a and b: $a \cap b$.
Or(a, b)	Returns an object set containing the elements that are present in either of a or b: $a \cup b$.
XOR(a, b)	Returns an object set that represents the element-wise exclusive-or of a and b.
closest(a, b)	The object set a must contain only a single element. closest computes the distance from each element in object set b to the single element in a and returns the element from b that minimizes this distance: $\text{argmin}_{b_i \in b} \text{distance}(a, b_i)$.
furthest(a, b)	Is analogous to closest, only it selects the element from b that is furthest from the single element in a.
map-pov$_i(a)$	Described in section 4.6.2, map-pov remaps the elements of set a to as if robot i were the subject of the classification.

Table 4.4: Selection operators for defining relational features

often containing only a single object, to real values that we can incorporate into features for our classifier. Like selection operators, these evaluation operators also depend on t and X. Again, we omit those parameters for the sake of concise notation. A second group of evaluation operators, which we call compound (as opposed to simple), operate on nested evaluation operators. For example, we might want to compute a flag that indicates whether or not a distance is less than some threshold k. The operator for this test would nest a distance evaluation operator and evaluate that nested operator in the process of evaluating the threshold test. Table 4.5 lists the evaluation operators that we use when defining features. Simple evaluation operators provide a way of accessing information about any particular object, such as its position, heading, or velocity. Compound evaluation operators operate on either simple or compound operators to compute thresholds, changes over time, and so forth.

4.5.2 Features of the Small Size Domain

We defined approximately 30 feature prototypes using our relational feature specification language. In this section, we give detailed definitions of our feature prototypes in terms of

Evaluation Operator	Description
x-position(i)	Defined when candidate set i contains a single object. It computes the x position of the object on the field. The x-axis runs between the two goals and passes through the center of the field.
y-position(i)	Analogous to x-position except it computes the y position. The y-axis is the mid-field line.
heading(i)	Defined when candidate set i contains a single object and that object has a well defined orientation. In practice, this operator is only defined when the object in i is a robot as neither the ball nor goals have a heading.
velocity(i)	Computes the velocity of the single object in candidate set i by subtracting the position of the object at time $t-1$ from the position of the object at time t.
distance(i, j)	Computes the distance between the single object in candidate set i and the single object in candidate set j.
closer(i, j, x)	computes a binary flag that indicates if distance(i, k) $<$ distance(j, k).
further(i, j, k)	computes a binary flag that indicates if distance(i, k) $>$ distance(j, k).
delta(x)	Computes the change in the nested evaluation operator x between time t and time $t-1$. If $t-1$ falls outside of the sequence, this operator evaluates to zero.
less-than(x, k)	Computes a binary flag that indicates whether the nested evaluation operator x is less than a constant k.
greater-than(x, k)	See less-than.

Table 4.5: Evaluation operators for defining relational features. We distinguish between simple evaluation operators, which operate on object sets or the output of selection operators versus compound evaluation operators which operate on nested evaluation operators.

the function $g(t, X)$. We used g to build state and transition linked features of the form:

$$f_i = (y_t \overset{?}{=} \text{label}) g(t, X) \tag{4.18}$$

$$f_j = (y_{t-1} \overset{?}{=} \text{label})(y_t \overset{?}{=} \text{label}) g(t, X) \tag{4.19}$$

All told, we created 26,430 unique feature/weight pairs from our approximately 30 relational feature prototypes.

Testing A closer than B to C

Our first set of prototypes tests whether one object is closer than another to a reference point. E.g., "Is the subject of the classification closer to the ball than any of the four other robots on its team?" The ball is the reference point in this example. We provide function definitions for all of the closer prototypes that we created. We used an identical set of prototypes based around the further evaluation operator, which we omit

To identify the role of the subject, we care if it is closer to the ball than any of its opponents or teammates. Below, we test if the subject is closer to the ball than its teammates, opponents, or any other robot:

$$g(t, X) = \begin{array}{l}\mathrm{closer}(\mathbf{subject}, \\ \quad \mathrm{closest}(\mathbf{ball}, \mathbf{teammates}), \\ \quad \mathbf{ball})\end{array} \tag{4.20}$$

$$g(t, X) = \begin{array}{l}\mathrm{closer}(\mathbf{subject}, \\ \quad \mathrm{closest}(\mathbf{ball}, \mathbf{opponents}), \\ \quad \mathbf{ball})\end{array} \tag{4.21}$$

$$g(t, X) = \begin{array}{l}\mathrm{closer}(\mathbf{subject}, \\ \quad \mathrm{closest}(\mathbf{ball}, \mathbf{other\text{-}robots}), \\ \quad \mathbf{ball})\end{array} \tag{4.22}$$

To identify whether the subject is in an offensive or defensive role, we tested whether the subject was closer to its own goal than any of its opponents. Robots playing defense often position themselves between opponents and the goal. Similarly, we tested if the subject was closer to its opponents' goal than any of its teammates. The robot closest to the opponent goal is generally not playing defense. Equation (4.25) gives the formal encoding for these features:

$$g(t, X) = \begin{array}{l}\mathrm{closer}(\mathbf{subject}, \\ \quad \mathrm{closest}(\mathbf{own\text{-}goal}, \mathbf{opponents}), \\ \quad \mathbf{own\text{-}goal})\end{array} \tag{4.23}$$

$$g(t, X) = \begin{array}{l}\mathrm{closer}(\mathbf{subject}, \\ \quad \mathrm{closest}(\mathbf{opp\text{-}goal}, \mathbf{teammates}), \\ \quad \mathbf{opp\text{-}goal})\end{array} \tag{4.24}$$

$$\tag{4.25}$$

At kickoff, one robot from the team playing offense enters the center circle to kick the ball. In general, the knowledge that the subject is located in the middle of the field carries information about its role. We created the below prototypes to test when the subject was the closest robot on its team to the center, when the subject was closer than its opponents, and when the subject was closet of all robots:

$$g(t, X) = \text{closer}(\textbf{subject}, \text{closest}(\textbf{center-field}, \textbf{teammates}), \textbf{center-field}) \tag{4.26}$$

$$g(t, X) = \text{closer}(\textbf{subject}, \text{closest}(\textbf{center-field}, \textbf{opponents}), \textbf{center-field}) \tag{4.27}$$

$$g(t, X) = \text{closer}(\textbf{subject}, \text{closest}(\textbf{center-field}, \textbf{robots}), \textbf{center-field}) \tag{4.28}$$

The pattern of asking whether A or B is closer to C can be used to capture information about the state of the game. For example, despite the fact that the goals do not move, we can use the pattern "Is the subject's goal closer to the ball than the opponent's goal" to test whether the subject's team is on offense or defense. Similarly, we can use the distances between the goals and the subject to test which side of the field the subject is located on. Finally, the robot on each team that is closest to the ball is a special case since it is likely to interact with the ball.

$$g(t, X) = \text{closer}(\textbf{own-goal}, \textbf{opp-goal}, \textbf{ball}) \tag{4.29}$$

$$g(t, X) = \text{closer}(\textbf{own-goal}, \textbf{opp-goal}, \textbf{subject}) \tag{4.30}$$

$$g(t, X) = \text{closer}(\textbf{own-goal}, \textbf{opp-goal}, \text{closest}(\textbf{ball}, \textbf{teammates})) \tag{4.31}$$

$$g(t, X) = \text{closer}(\textbf{own-goal}, \textbf{opp-goal}, \text{closest}(\textbf{ball}, \textbf{opponents})) \tag{4.32}$$

Some roles, such as forming a wall to block opponent kicks require grouping together with teammates. Other roles, such as marking an opponent require moving close to an opponent robot. Below, we ask the question: "Is the subject closer to a teammate or an opponent?".

We also consider the question of which team is closer to the ball, since this generally indicates which team has possession at any given time.

$$g(t,X) = \begin{array}{r}\text{closer}(\text{closest}(\mathbf{subject}, \mathbf{teammates}),\\ \text{closest}(\mathbf{subject}, \mathbf{opponents}),\\ \mathbf{subject})\end{array} \tag{4.33}$$

$$g(t,X) = \begin{array}{r}\text{closer}(\text{closest}(\mathbf{ball}, \mathbf{own\text{-}team}),\\ \text{closest}(\mathbf{ball}, \mathbf{opp\text{-}team}),\\ \mathbf{ball})\end{array} \tag{4.34}$$

Distances Between Objects

Evaluating how one distance compares to another provides useful information about the subject's role. The actual distances themselves also carry information. We defined a collection of features to compute various important distances. For each of the distances below, we consider the distance itself, the change in the distance since the last time step, and whether that distance exceeded various thresholds. For the distances given by $g(t, X)$, we added features:

$$f_i(t, y_{t-1}, y_t, X) = (y_t \overset{?}{=} \text{label}) g(t, X) \tag{4.35}$$

$$f_i(t, y_{t-1}, y_t, X) = (y_t \overset{?}{=} \text{label}) \text{delta}(g(t, X)) \tag{4.36}$$

$$f_i(t, y_{t-1}, y_t, X) = (y_t \overset{?}{=} \text{label})(g(t, X) \overset{?}{\leq} k) \tag{4.37}$$

$$f_i(t, y_{t-1}, y_t, X) = (y_{t-1} \overset{?}{=} \text{label})(y_t \overset{?}{=} \text{label})(g(t, X) \overset{?}{\leq} k) \tag{4.38}$$

$$\begin{aligned} k \in \{&100, 125, 150, 175, 200, 250, 300,\\ &350, 400, 450, 500, 600, 700, 800\\ &900, 1000 \text{ millimeters}\}\end{aligned} \tag{4.39}$$

One of the most obvious distances to use as $g(t, X)$ is the distance between the subject and the ball:

$$g(t, X) = \text{distance}(\mathbf{ball}, \mathbf{subject}) \tag{4.40}$$

Similarly, the distance between the robot closest to the ball and the ball carries information about the game. Here we compute that distance for each of the two teams:

$$g(t, X) = \begin{array}{c}\text{distance}(\text{closest}(\mathbf{ball}, \mathbf{own\text{-}team}), \\ \mathbf{ball})\end{array} \tag{4.41}$$

$$g(t, X) = \begin{array}{c}\text{distance}(\text{closest}(\mathbf{ball}, \mathbf{opp\text{-}team}), \\ \mathbf{ball})\end{array} \tag{4.42}$$

$$\tag{4.43}$$

Above, we computed a binary flag to indicate the side of the field that contained the ball. Here we compute the distance between the ball and either of the goals, as well as between the ball and center field. The last of these distances helps to identify kickoff situations.

$$g(t, X) = \text{distance}(\mathbf{own\text{-}goal}, \mathbf{ball}) \tag{4.44}$$

$$g(t, X) = \text{distance}(\mathbf{opp\text{-}goal}, \mathbf{ball}) \tag{4.45}$$

$$g(t, X) = \text{distance}(\mathbf{center\text{-}field}, \mathbf{ball}) \tag{4.46}$$

The distance between the subject and its closest teammate and opponent can help classify whether the subject is marking another robot or forming a wall with its teammates.

$$g(t, X) = \begin{array}{c}\text{distance}(\text{closest}(\mathbf{subject}, \mathbf{teammates}), \\ \mathbf{subject})\end{array} \tag{4.47}$$

$$g(t, X) = \begin{array}{c}\text{distance}(\text{closest}(\mathbf{subject}, \mathbf{opponents}), \\ \mathbf{subject})\end{array} \tag{4.48}$$

The distance between the subject and the robot on its own team that is closest to the ball can help identify if the subject is at an appropriate distance to receive a pass. Or, if the distance is zero, that the subject is the closest on its team to the ball. The same distance computed with respect to the opponents can tell us if the subject is close to an attacking opponent and trying to capture the ball or if the subject is further away, perhaps defending its own goal.

$$g(t, X) = \begin{array}{c}\text{distance}(\text{closest}(\mathbf{ball}, \mathbf{own\text{-}team}, \\ \mathbf{subject})\end{array} \tag{4.49}$$

$$g(t, X) = \begin{array}{c}\text{distance}(\text{closest}(\mathbf{ball}, \mathbf{opp\text{-}team}), \\ \mathbf{subject})\end{array} \tag{4.50}$$

The distance between the subject and either goal carries information about whether the subject is playing offense or defense.

$$g(t, X) = \text{distance}(\textbf{own-goal}, \textbf{subject}) \tag{4.51}$$
$$g(t, X) = \text{distance}(\textbf{opp-goal}, \textbf{subject}) \tag{4.52}$$

Finally, we created a more complex feature that measures the distance between the subject and the closest opponent to the subject that is not the closest opponent to the ball. In other words, we measure the distance between the subject and an opponent that is likely to receive a pass to determine if the subject is marking that opponent.

$$\begin{aligned} g(t, X) = \text{distance}(&\textbf{subject}, \\ &\text{closest}(\textbf{subject}, \\ &\quad \text{xor}(\textbf{opp-team}, \\ &\qquad \text{closest}(\textbf{ball} \\ &\qquad\quad \textbf{opp-team})))) \end{aligned} \tag{4.53}$$

Headings and Velocities

The headings of robots and the velocities of the robots and the ball also capture important information about the state of the game. For example, we can use the ball velocity to detect passing and dribbling. We created state-linked features for headings and velocities. In the case of headings, we also used the *delta* evaluation operator to include the change in an object's heading as a state-linked feature.

We defined features in terms of the ball's velocity, the velocity of the subject, as well as the velocities of the robots on either team that were closest to the ball:

$$g(t, X) = \text{velocity}(\textbf{ball}) \tag{4.54}$$
$$g(t, X) = \text{velocity}(\textbf{subject}) \tag{4.55}$$
$$g(t, X) = \text{velocity}(\text{closest}(\textbf{ball}, \textbf{own-team})) \tag{4.56}$$
$$g(t, X) = \text{velocity}(\text{closest}(\textbf{ball}, \textbf{opp-team})) \tag{4.57}$$

We defined features based on the headings of the subject and the robots on either team that were closest to the ball:

$$g(t, X) = \text{heading}(\textbf{subject}) \tag{4.58}$$

$$g(t, X) = \text{heading}(\text{closest}(\textbf{ball}, \textbf{own-team})) \tag{4.59}$$

$$g(t, X) = \text{heading}(\text{closest}(\textbf{ball}, \textbf{opp-team})) \tag{4.60}$$

4.5.3 Experiments

We repeated our experiments with the same RoboCup data as in our initial experiments with relational features instead of simple spatial features. Due to the large size of the feature set, we also tested feature selection to test if a small subset of the 26430 relational features generated from our templates can offer high classification accuracy. We compared traditional maximum likelihood parameter estimation, grafting, and ℓ_1 regularization for estimating the model parameters. We did not test the mean field heuristic due to the computational demands of that algorithm.

Table 4.6 summarizes the results of our experiments. Overall, relational features produced lower error rates than our earlier experiments with simple spatial features. Relational features provide a succinct means for specifying domain knowledge, which simplifies the process of creating informative features for the model. As in previous chapters, we observe a trend where grafting produces very sparse, but slightly less accurate models than the other approaches. ℓ_1 regularization offers an intermediate between the small models produced by grafting and the large, but accurate models, produced by training the model with all of the candidate features and ℓ_2 smoothing. As before, there appears to be little or no benefit to retraining the models with ℓ_2 smoothing after feature selection; ℓ_2 smoothing provides the most benefit in the full model.

Relational features provide a concise and flexible method for defining features that capture domain knowledge and allow for high classification accuracies. In the next section, we explore how relational features can be used to capture information about the roles of other robots in multi-robot settings. Soccer is, of course, a multi-robot domain and the roles of the robots on a single team are not independent. In the next section, we investigate whether classification accuracy can be further improved by automatically expanding the set of relational features to provide information about the roles of a robot's teammates.

Training	Error Rate	Model Size
ML	14.2%	26430
ML / ℓ_2	11.0%	26430
ℓ_1	10.9%	817
ℓ_1 / ℓ_2	11.2%	817
Grafting	12.2%	260
Grafting / ℓ_2	12.1%	260

Table 4.6: While training on data from three earlier games and testing with data from the final, we compared maximum likelihood training with no smoothing; maximum likelihood training with ℓ_2 smoothing; ℓ_1 regularization for feature selection / smoothing; ℓ_1 regularization for feature selection followed by ℓ_2 smoothing; grafting for feature selection; and grafting for feature selection followed by ℓ_2 smoothing. We show average error rates on test data for the chosen subset of features and the size of the final model.

4.6 Multi-Robot Activity Recognition

Robot soccer is a multi-robot activity recognition task. When predicting the role of one robot, knowledge about the roles of the other robots provides useful information. We must consider how to include information about the roles of other robots in a tractable and computationally efficient manner. We begin with a discussion of multi-robot activity recognition in general and then describe our approach, where we use relational features to automatically expand the feature set to include information about the roles of other robots.

4.6.1 Reasoning about the Roles of Other Agents

Information about the roles of a robot's teammates and opponents provides information about that robot's own role. As we discuss below, there are several ways of incorporating role information from other robots into the classification. In general, we desire a model that can be used for real time activity recognition in a real robot system. In the case of the RoboCup Small Size League, this means that the classifier must operate at 60 hz and use only a modest portion of the CPU cycles available on a single workstation; the remaining CPU cycles are needed for path planning, vision, and all of the other computationally expensive tasks required to control a five robot team in real time. With computational limitations in mind, we prefer models with structures that permit exact inference over more complex models that require approximate inference, such as loopy belief propagation [75] or Markov chain Monte Carlo [31], because inference in more complex models tends to require more computational effort.

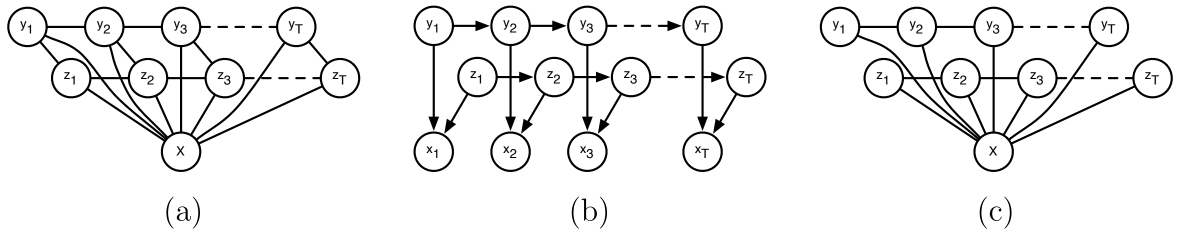


Figure 4.3: *Factored Model Structures* Two label chains Y and Z are shown along with an observation sequence X. The labels for individual time steps y_t and z_t represent the roles of robots y and z respectively at time t. The observation vector at each time step x_t contains the positions of the robots and ball. (a) Dynamic conditional random fields model dependencies between robots with links that directly connect the label chains. (b) Factorial hidden Markov models model dependencies between robots indirectly by including directed links from the label chains to the observations, which, because values for the observations are known, correlates the label chains. (c) A conditional random field with the same structure as a factorial HMM is equivalent to independent CRFs because known values of the observation sequence make the separate label chains *independent*.

Modeling the Joint Role

Modeling the joint role of an entire team, that is, using a single random variable that takes on k^n values, where k is the number of single-agent roles and n is the number of agents, is intractable. The storage space required for the model parameters and the time required to sum over states grows exponentially with the number of agents.

Factoring Roles

Factoring the joint role of the team into separate chains of random variables, where each chain represents the role of one robot over time, avoids a combinatorial explosion in the size of the state space. The storage requirements for a factored model increase only linearly with the number of robots. To model dependencies between roles, we can add links between the label chains. Conditional random fields with factored state and connections between the label chains are called dynamic conditional random fields [100][1]. However, because the graph structure contains loops, the time required to perform inference grows exponentially with

[1] The term "Factorial CRF" is sometimes used to describe dynamic CRFs that contain several label chains that link to both the observations and the other label chains. We avoid this term to avoid confusion with factorial HMMs, which do not contain links between the label chains.

the tree-width of the graph. The tree-width of the graph grows linearly with the number of robots and the length of the sequences, so approximate inference methods are necessary.

Linking Roles via Observations

Factorial hidden Markov models are directed graphical models with factored state that use edges between label chains and observation vectors to indirectly link the label chains, as shown in figure 4.3. A factorial HMM models the observation vectors x_t more accurately than a traditional HMM, because the factorial model accounts for the fact that the observations depend on the roles of all the robots. Both factorial HMMs and HMMs are joint models of the label chains and observations. Because factorial HMMs better explain the observations, a single factorial HMM with n label chains tends to be more accurate than n independent HMMs, with a separate HMM for each label chain [30]. However, exact inference is intractable in factorial HMMs. Known values for the observations correlate the label chains and create loops in the graph, according to the rules for conditional independence (*d-separation*) in directed graphs [84].

In an undirected model, we cannot indirectly link the label chains in the same fashion as factorial HMMs. A CRF, with n label chains and *no links between the label chains*, shown for $n = 2$ in figure 4.3, is exactly equivalent to n independent CRFs. Known values for the observations make the separate label chains independent according to the rules for conditional independence in undirected graphs [57]. Rather than adding links between label chains, we use n independent CRFs and augment them with features that carry information about the roles of the other robots. Introducing features that represent the roles of other robots is an *approximation* to the dynamic CRF case, which directly models interactions between label chains at the cost of NP-hard exact inference. We adjust the fidelity of the approximation by controlling the number and configuration of the stand-in features that we include in the model.

We can create a feature f_{r_1} to act as a stand-in for the role of robot 1 when predicting the role of another robot, which we refer to as robot 2. We build f_{r_1} around

$$g_{r_1} : (t, X) \rightarrow \mathit{role}, \tag{4.61}$$

where g_{r_1} is an arbitrary classifier that predicts the role of robot 1 at time t. We then create a feature to use g_{r_1}'s prediction to classify robot 2's role:

$$f_{r_1}(t, y_{t-1}, y_t, X) = (y_t \overset{?}{=} r_2\text{'s role})(g_{r_1}(t, X) \overset{?}{=} r_1\text{'s role}). \tag{4.62}$$

Because conditional random fields are discriminatively trained and robustly incorporate non-independent features, we can include an ensemble of such classifiers and rely on the training algorithm to select the appropriate weights for each classifier in the ensemble. We can also create features that incorporate classifiers that predict roles for several robots, although this quickly leads to intractably large feature sets. We can control the size of the feature set by adjusting the number of different classifiers that we use to predict the roles of the other robots and by limiting the number of predictions for different robots in each feature, e.g. to pairwise interactions.

We can include information about the roles of the other robots in other ways. For example, we can define

$$g_{r_1} : (t, X) \to \Re \tag{4.63}$$

$$f_{r_1}(t, y_{t-1}, y_t, X) = (y_t \overset{?}{=} r_2\text{'s role}) g_{r_1}(t, X), \tag{4.64}$$

where g_{r_1} is an arbitrary function that carries information about robot 1's role. This formulation takes the same form as a conventional feature in a CRF. The only difference is that g_{r_1} is chosen to provide information about robot 1's role rather than robot 2's role, which is purely a semantic distinction that is invisible when training the model. The advantage of this more indirect approach, where we include features that carry information about the roles of others, but that are not direct prediction of the role, is that we can automate the process of feature creation. We design a candidate set of features to predict the role of one robot. And then we automatically remap those features to reflect the point of view of other robots and to therefore help predict the roles of the other robots.

4.6.2 Automatically Constructing Features

We use relational features to our advantage in multi-robot domains. In section 4.6.1, we discussed computationally efficient ways of incorporating role information into the classification of the subject's role. One method was to embed classifiers as features. A second method, which we discuss here, is to remap the elements in the object sets. Viewing the activity recognition problem from a single agent perspective, we create many features designed to categorize the role of the subject. We can augment the set of features with remapped versions of the original features that categorize the roles of other robots as if the other robots were the subject.

As a concrete example, the distance between the subject and its closest opponent can help identify if the subject is executing the *Marking* role:

$$\begin{aligned}&\text{distance}(\textbf{subject},\\&\qquad\text{closest}(\textbf{subject},\\&\qquad\qquad\textbf{opponents})).\end{aligned} \tag{4.65}$$

To remap the point of view so that the feature uses a different robot as the subject of the classification, we wrap all of the object-sets in the feature with the function map-pov$_i$, where i identifies the robot that will serve as the new subject. Modifying (4.65) yields:

$$\begin{aligned}&\text{distance}(\text{map-pov}_i(\textbf{subject}),\\ &\qquad \text{closest}(\text{map-pov}_i(\textbf{subject}),\\ &\qquad\qquad \text{map-pov}_i(\textbf{opponents}))).\end{aligned} \tag{4.66}$$

The function map-pov$_i$ is a selection operator that maps an object set to a new object set, depending on which robot is the new subject, as specified by i. There is a one to one correspondence between elements in the original object set and the elements in the new object set:

$$\text{map-pov}_i(s) = \{\, \forall e \in s : \text{remap}_i(e)\,\} \tag{4.67}$$

To create a feature that retains the current subject rather than changing it, we would define remap$_i$ as:

$$\text{remap}_{\text{subj}}(s) = s \tag{4.68}$$

A more useful case is to change the point of view of the feature so that one of the subject's teammates becomes the new subject of the feature. For example, if the new subject is teammate 2, we would define remap$_i$ as:

$$\text{remap}_{\text{tm}_1}(s) = \begin{cases} \textbf{subject} & \text{if } s = \textbf{tm-2} \\ \textbf{tm-2} & \text{if } s = \textbf{subject} \\ s & \text{otherwise} \end{cases} \tag{4.69}$$

The majority of the elements in the original object set are simply copied by the third clause of the case statement above. The ball is the same, no matter which robot is the subject. Similarly, the entries for the goals and the majority of the teammates remain unchanged. The only difference between the new and the old object sets is that the original subject is replaced by its teammate and vice versa.

Remapping the feature point of view so that an opponent becomes the subject is more complicated. For example, to select opponent 1 as the subject, we would define remap_i as:

$$\text{remap}_{opp_1}(s) = \begin{cases} \textbf{opp-1} & \text{if } s = \textbf{subject} \\ \textbf{opp-2} & \text{if } s = \textbf{tm-2} \\ \dots & \dots \\ \textbf{opp-5} & \text{if } s = \textbf{tm-5} \\ \textbf{subject} & \text{if } s = \textbf{opp-1} \\ \dots & \dots \\ \textbf{tm-5} & \text{if } s = \textbf{opp-5} \\ \textbf{own-goal} & \text{if } s = \textbf{opp-goal} \\ \textbf{opp-goal} & \text{if } s = \textbf{own-goal} \\ s & \text{otherwise} \end{cases} \tag{4.70}$$

We need to account for the fact that teammates must be swapped with opponents and that the two goals must be swapped.

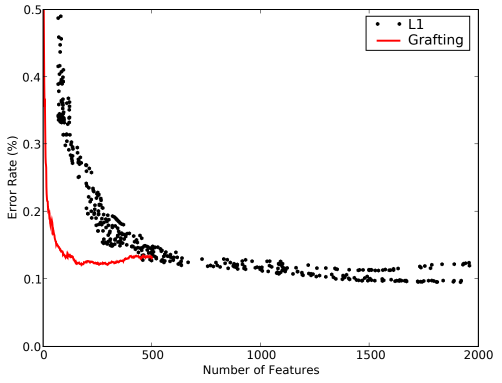
In general, we can start from a set of features that are useful for recognizing the role of the subject; create additional copies of that base set of features, where the copies are remapped so that other robots are the subject; and, because the roles of other robots carry information about the role of the original subject, we can use the remapped features to help classify the role of the original subject. In the next section, we explore this approach with the RoboCup data set.

4.6.3 Multi-Robot Relational Feature Experiments

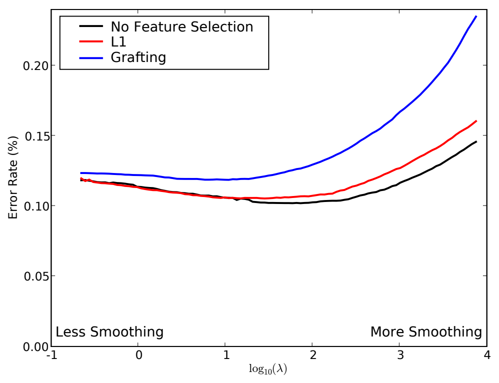
We return to the data recorded by the CMDragons'07 small size team to compare feature selection algorithms with relational features where we automatically remap the subject of the classification in the features to capture information about the roles of the other robots. We have previously published these results in [111].

Accuracy and Sparsity

In feature selection, there is a trade-off between the size of the model and the accuracy of the model. Up to a point, adding features will decrease the error rate of the model. Past a certain point, additional features increase the error rate of the model due to over-fitting. Figure 4.4 (a) shows error rate versus model size for grafting and ℓ_1 regularization. Grafting,



a) Error Rates versus Model Size



b) ℓ_2 Regularization Paths

Figure 4.4: a) Error rate on test data versus model size is shown for the successive models proposed by grafting and ℓ_1 regularization. Grafting quickly converges to a minimum and then shows evidence of over-fitting. ℓ_1 regularization produces larger models, but achieves a lower error rate overall. b) Error rate on test data versus the smoothing parameter λ is shown for retraining models that contain only the features selected by grafting and ℓ_1 regularization using ℓ_2 regularization for smoothing. On average, grafting selected 220 features and ℓ_1 regularization selected 1823. For comparison, we also show the ℓ_2 regularization path for the no feature selection case that included all 92,310 features. Grafting, which uses the fewest features shows the highest error rate. ℓ_1 regularization, with an intermediate number of features performs almost identically to the CRF with all 92,310 features.

the purely greedy algorithm, shows a more rapid decrease in the error rate as features are added. ℓ_1 regularization shows a less steep decline in the error rate, but achieves, on average, a lower error rate than grafting because it is less greedy and able to remove or swap out one feature for another during the course of training. The results in figure 4.4 (a) are from three trials where a round-robin, quarter-final, and semi-final game were each used in turn as the hold out set and the other two were used for training. The final was always used as the test set. The results for grafting, which predictably adds a single feature per iteration are shown as an average over the three trials. The results for ℓ_1 regularization are plotted as independent points for each trial because the model size changed unpredictably between iterations.

Grafting does not explicitly smooth the final model. We explored using ℓ_2 regularization to smooth models containing only the selected features to see if it would lower the error rate further. As a comparison, we also applied ℓ_2 regularization to smooth the full model that contained 92,310 features. Figure 4.4 (b) shows the average error rate on test data as the smoothing parameter λ varies during ℓ_2 penalized training. With smoothing, grafting

Training	Error Rate	Model Size	Time (hours)
ML	15.7%	92310	3.1
ML / ℓ_2	10.5%	92310	11.2
ℓ_1	10.3%	1823	6.9
ℓ_1 / ℓ_2	10.7%	1823	1.7 (8.6)
Grafting	12.3%	220	1.9
Grafting / ℓ_2	12.0%	220	0.6 (2.5)

Table 4.7: While training on data from three earlier games and testing with data from the final, we compared maximum likelihood training with no smoothing; maximum likelihood training with ℓ_2 smoothing; ℓ_1 regularization for feature selection / smoothing; ℓ_1 regularization for feature selection followed by ℓ_2 smoothing; grafting for feature selection; and grafting for feature selection followed by ℓ_2 smoothing. We show average error rates on test data for the chosen subset of features, the size of the final model, and the training time required to discover the final model. When feature selection was followed by smoothing, we show the time required for smoothing followed by the cumulative time for feature selection and smoothing in parentheses.

shows a higher error rate than either ℓ_1 regularization or the full model (both also with smoothing), possibly because grafting omits relevant features by being too greedy during the feature selection process. The features selected by ℓ_1 regularization achieve an error rate that is virtually identical to the full model even though the former is a small subset of the features present in the full model.

Table 4.7 gives the error rates on test data for the models selected using the hold out set. Both the regular and ℓ_2 smoothed models produced via ℓ_1 regularization perform almost identically to the full model with ℓ_2 smoothing. Some sort of smoothing, either by selecting a small subset of the available features or by applying an ℓ_2 penalty is necessary to achieve low error rates as the full model without ℓ_2 smoothing has the highest error rate. Grafting falls in the middle between the most accurate training methods and the unsmoothed full model. However, if we use the size of the final model or the training time required to discover the final model as a metric, then grafting comes out ahead. On average, grafting discovers smaller models more quickly than ℓ_1 regularization. And both feature selection algorithms discover the final model in less time than it takes to train the full model with ℓ_2 smoothing, which provides strong motivation for using feature selection.

Comparing the results in table 4.7, which automatically remaps object sets as if the other robots were the subject of the classification, with the results in table 4.6, which use relational features but do not remap the point of view, shows similar error rates across the experiments. In our domain, additional features designed to capture information about the roles of the

Actual Role	Position for Ready	Mark	Position for Pass	Receive Chip	Position	Penalty Kick	Wall	Set Play Kick	Attacker	Defend Circle	Label Frequency
Pos. for Ready	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2
Mark	0.0	3.6	0.6	0.0	0.0	0.0	0.7	0.0	0.1	0.9	6.0
Pos. for Pass	0.0	0.2	8.8	0.0	0.1	0.0	0.3	0.0	0.4	0.4	10.2
Receive Chip	0.0	0.0	0.3	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.4
Position	0.0	0.1	0.3	0.1	0.3	0.0	0.1	0.0	0.0	0.3	1.2
Penalty Kick	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Wall	0.0	0.1	0.1	0.0	0.0	0.0	38.1	0.0	0.2	1.1	39.6
Set Play Kick	0.0	0.0	0.1	0.0	0.0	0.0	0.1	1.6	0.1	0.0	1.9
Attacker	0.0	0.1	0.4	0.0	0.0	0.0	0.4	0.1	7.1	0.6	8.6
Defend Circle	0.0	0.2	0.3	0.0	0.0	0.0	0.9	0.0	0.4	30.0	31.9

Table 4.8: The confusion matrix for the robot soccer role recognition task. The value in each cell specifies a percentage of the time steps in the test sequences broken down according to the true label in the test set and the predicted label according to a CRF trained with ℓ_1 regularization. The elements in each row all share the same true label in the test set. The columns indicate the classifier's prediction for each label. The rightmost column contains the sum of each row and specifies the percentage of the test set that took on each label.

other robots do not seem to improve classification accuracy. However, as we show later in this chapter, feature selection chooses the remapped features a significant fraction of the time; the remapped features are useful for the classification, but they do not significantly improve the classification accuracy.

Table 4.8 shows the performance of the CRF produced by ℓ_1 regularization (without post-feature selection ℓ_2 smoothing) in detail. While the majority of the labels in the test set belong to the *wall* and *defend circle* roles, which account for 39.6% and 31.9% of the labels respectively, the robots spend significant portions of their time in strategically interesting roles. Specifically, they spend significant time in the *position for pass*, *set-play-kick*, *mark*, and *attacker* roles. The first two roles are of particular interest to an opposing team. The CRF correctly identifies 86% of the time steps where a robot is attempting to receive a pass and 84% of the time steps where a robot is positioning itself to deflect the ball directly into the goal. Activity recognition can provide potentially useful information about opponent

y_{t-1}	y_t	Siblings	$g(t, X)$
Wall	Wall	1	1 (First-order Markov transition)
*	Defend Circle	1	distance(closest(**subject**, **opp-team**), **subject**)
*	Wall	2	distance(map-pov$_{tm_2}$(**subject**), map-pov$_{tm_2}$(**own-goal**))
*	Wall	2	distance(map-pov$_{tm_2}$(**subject**), map-pov$_{tm_2}$(**opp-goal**)
Defend Circle	Defend Circle	3	further(**subject**, closest(**ball**, **other-robots**), **ball**)
*	Position for Pass	2	distance(**subject**, closest(**subject**, **teammates**))
*	Attacker	2	distance(**subject**, closest(**subject**, **teammates**))
Mark	Mark	3	further(**subject**, closest(**ball**, **other-robots**), **ball**)
*	Attacker	2	distance(**subject**, **ball**)
Wall	Wall	1	distance(**subject**, **ball**) $\overset{?}{<}$ 800 mm

Table 4.9: A summary of the first ten features chosen by grafting. The first two columns specify the label values associated with the selected feature. We denote the case of state-linked features, where the previous label is not tested, with a * in place of a label for y_{t-1}. The siblings column specifies the number of related features, with the same value for $g(t, X)$, but different label tests, that also have non-zero weight. State-linked features form groups of 10 siblings and transition-linked features for groups of 100 siblings.

robots.

Characterizing the Selected Features

Table 4.9 shows the first ten features that grafting chose for the robot soccer role recognition task. The first ten features contain a mix of real valued distances, binary tests of whether one object is further from a reference point than a second object, and single feature that encodes a first-order Markov self-transition for the Wall role. The first ten features had few siblings, that is, few other features, with the same $g(t, X)$ and a different label permutation, received non-zero weight. The list contained two features that were automatically remapped to reflect the role of a different robot; the additional features that we generated to exploit the correlations between role labels were chosen by feature selection. Examining the first 100 features chosen by grafting, we find that:

Training	Error Rate	Model Size
ML	18.6%	92310
ℓ_1	8.0%	1449
Grafting	10.9%	137
Grafting / ℓ_2	10.6%	137

Table 4.10: We tested generalization across different portions of the same game (the final) by training on date from the first half of the final and testing on data from the second half.

- 86 of the first 100 features are state linked; the other 14 are transition linked. Only 1 of the transition features is a bare first-order Markov transition, where $g(t, X) = 1$. The others all use $g(t, X)$ to incorporate information from the observation sequence. Feature selection exploits features that link transitions to observations as well as state-linked features.

- The median number of siblings for the first 100 features was 2. The mean was 1.8. For a given function $g(t, X)$ in a state-linked feature, 8/10ths of the label permutations associated with that feature commonly had zero weight. For a given function $g(t, X)$ in a transition-linked feature, 98/100ths of the label permutations associated with that feature commonly had zero weight. These results validate our approach by selecting features on a weight-by-weight basis rather than activating all of the weights associated with a particular value of g.

- 44 of the first 100 features chosen by grafting made use of the *remap-pov* operator that we created for multi-robot activity recognition. The features that we automatically generated are identified by feature selection as among the most relevant.

Generalization Across Halves

Table 4.7 shows that we can accurately predict the roles of a single team against different opponents. Table 4.10 provides similar results for the case where we train the model using data from the first half of a game (the final) and test against data from the second half of the same game. The motivation for this experiment is to show that we can, in principle, use only data gathered during the first half of a game to perform accurate activity recognition during the second half. In reality, there is not enough time between halves of a robot soccer game to label training data and train a model. However, these results show that in general, conditional random fields trained by grafting or ℓ_1 regularization can achieve high accuracies for activity recognition from a limited amount of training data.

4.7 Chapter Summary

In this chapter, we examined feature selection for activity recognition in multi-robot domains where we need to handle very large numbers of features. More specifically,

- We described the RoboCup Small Size League and introduced a benchmark activity recognition problem using logged data from the RoboCup'07 robot soccer world championship.
- We discussed multi-robot aspects of activity recognition. Specifically, we discussed the trade off between accuracy and computational complexity when incorporating information about the roles of other robots. We discussed how conditional random fields, which condition on observations and can incorporate arbitrary features of the observations, are good candidate models in multi-robot domains.
- We analyzed the scalability of grafting, the mean field heuristic, and ℓ_1 regularization to domains with large numbers of features. Grafting scales without modification, the mean field heuristic does not scale well, and we presented a coordinate-wise training method to reduce the training time of ℓ_1 regularization.
- We provided experimental evidence:
  - Grafting is the most computationally efficient method of feature selection of the algorithms that we considered. It produces the sparsest final models at the expense of reduced accuracy due to over-fitting. The over-fitting takes place during the feature selection process; retraining the model under an ℓ_2 penalty to provide after the fact smoothing does not improve performance.
  - Using ℓ_1 regularization results in larger models than grafting, but the final models are more accurate on test data due to less over-fitting. Retraining with an ℓ_2 penalty does not improve classification accuracy. The smoothing from the ℓ_1 penalty sufficiently reduces over-fitting.
  - The full model without smoothing under-performs the smaller models produced by feature selection, even without additional smoothing. Smoothing the full model produces the same accuracy as ℓ_1 regularization at the price of a high computational cost for training and a high cost for using the final model for classification.

Chapter 5

An M-estimator for Fast Training and Feature Selection

In this chapter, we present an M-estimator that allows for fast, approximate parameter estimation in conditional random fields. We introduce the M-estimator, provide a proof that it is asymptotically consistent, and present an empirical validation that demonstrates that the M-estimator performs well in practice. In closing, we apply the M-estimator to the problem of feature selection in conditional random fields for activity recognition.

5.1 Introduction

Parameter estimation in conditional random fields is computationally intensive. Maximizing the conditional likelihood of the training set requires repeated inference in the model. Each round of inference requires belief propagation across the full model in order to compute the normalization constant. Belief propagation in linear chain CRFs is tractable, i.e. we use an efficient, polynomial time algorithm for it, but it requires significant time in practice. For example, the training time for some of the models that we used in the previous chapter was on the order of hours or days.

Training other sequence models, such as hidden Markov models and dynamic Bayesian networks [76], does not require computing an expensive normalization constant. But these alternate models lack many of the desirable properties of CRFs. They are not conditional models and they cannot easily incorporate complex, non-independent features of the observations.

In this chapter, we introduce an M-estimator for fast, approximate training in CRFs. The key property of the M-estimator is that it does not require that we compute a normalization constant during training. We avoid the need for normalization by changing the objective of the learning process. Instead of estimating parameters that represent $\ell(Y|X)$ from the training data, we seek to discover parameters that capture the difference between $\ell(Y|X)$ and a base model q_0. Using a cheaply trained model, such as an HMM, as the base model allows us to significantly reduce the computational effort of parameter estimation in the CRF.

5.1.1 Approximate Parameter Estimation

Conditional random fields represent the conditional log-likelihood $p(Y|X)$ of data using features of the form $f_k(t, y_{t-1}, y_t, X)$. In particular, we treat the observation sequence X as a single random variable. Features of this form are attractive because they allow us to capture arbitrarily complex relationships between observations through rich feature functions. Despite the presence of arbitrary functions of the observations, inference in CRFs is tractable because we condition on X. On the other hand, Markov random fields represent the joint probability $p(X, Y)$. Exact inference in an MRF with the same topology as a CRF and with the same rich features of the observations is intractable because it would require summing over all possible observations to compute the normalization constant.

The proposed M-estimator offers an approximate method for estimating the parameters of an MRF with the same graphical structure and features as a CRF. The M-estimator begins with a base model $q_0(X, Y)$ and learns a series of corrections to minimize the difference between $q_0(X, Y)$ and the true joint density $p(X, Y)$. If we represent the correction factors in the same form as the feature functions of a CRF, we can estimate approximate parameter values for those rich feature functions in a joint model $p(X, Y)$.

Estimating the model parameters is only the first half of the problem. The normalization constant is also required in order to use the MRF for inference. The M-estimator allows us to discover the model parameters, but we cannot efficiently make use of them in the joint model. However, the parameters allow us to efficiently compute the conditional probability $p(Y|X)$, which only requires summing over label sequences. The structure of the model allows for efficient summation over the labels, but not the observations. M-estimation discovers parameters for joint model that cannot be evaluated efficiently. But the resulting joint model is useful because we can efficiently evaluate queries over a subset of its variables, namely the label sequence.

In chapter 2, we described how to convert an HMM, which is a joint model of $p(X, Y)$, into a CRF. The parameters in the resulting CRF are not the same as the MLE parameters for $p(Y|X)$ that traditional, discriminative CRF training produces. Similarly, the parameters that result from M-estimation, although we use them to compute a conditional probability, also differ from the MLE parameters. Initializing a CRF using an HMM yields weights for simple Markov transitions and observation probabilities for the raw variables. It does not provide approximate weights for more complex features, such as whether or not the distance between two robots exceeds a threshold. On the other hand, the M-estimator, which represents $p(X, Y)$ in terms of rich features, such as thresholds on distance between robots, does provide weight values for those rich features. M-estimation allows us to perform approximate parameter initialization for arbitrary features of the observations and label pairs.

5.1.2 M-estimators

Our approach is based on a novel M-estimator that was recently proposed by Jeon and Lin [42] in the context of certain non-parametric ANOVA models using regression splines. M-estimation is the traditional statistical term for an estimator of the form

$$\widehat{\theta}_n = \operatorname*{argmax}_{\theta} M_n(\theta) = \operatorname*{argmax}_{\theta} \frac{1}{n} \sum_{i=1}^{n} m_\theta(X_i, Y_i). \tag{5.1}$$

The canonical example and source of the name M-estimator is the maximum likelihood estimator, where $m_\theta(X_i, Y_i) = \log p(X_i, Y_i; \theta)$ [113]. Statisticians have traditionally investigated certain M-estimators as robust alternatives to maximum likelihood. In particular, they use M-estimators to learn models that are robust to the presence of outliers. For example, the mean of a sample is sensitive to the presence of outliers because it minimizes the squared error over the sample points. An alternate measure of central tendency, due to Hubert [40], uses squared distance to compute error terms for points close to the center and replaces squared distance with linear distance for points further from the center. As a result, distant points do not influence the location of the center nearly as much as under the squared error penalty.

Statisticians use M-estimators to learn robust models that share maximum likelihood estimation's desirable properties of consistency [59] and asymptotic normality [113]. Although we provide a proof of asymptotic consistency for Jeon's estimator, we are motivated by *computational* efficiency rather than robustness or statistical efficiency. For us, the key property of the M-estimator that we consider is that the M-estimator circumvents the calculation of the normalizing constant in the setting of discrete, undirected graphical models.

5.1.3 CRF Training as M-Estimation

Maximum likelihood estimation, which is how we have trained our CRFs to date, is a form of M-estimation. When we use an M-estimator for parameter estimation, we maximize the objective function (5.1) over a training set of n examples $\{(X_i, Y_i)\}_{i=1}^n$. We vary the model parameters θ to maximize the objective function over the training set and hope that the resulting vector $\widehat{\theta}_n$ lies close to the unknown and truly optimal parameter vector θ^*.

With conditional random fields, X_i and Y_i represent *sequences* of observations and labels. To create an objective function with the same form as (5.1), we substitute θ in place of our usual weight vector w and define $F(X, Y)$ as the sum of the feature vectors over a sequence:

$$F(X, Y) = \sum_{t=1}^{T} f(t, y_{t-1}, y_t, X). \tag{5.2}$$

Using this notation, the conditional probability of a label sequence $Y = (y_1, \dots, y_T)$ is then

$$p(Y \mid X) = \frac{1}{Z_\theta(X)} \exp\left(\theta^T F(X, Y)\right) \tag{5.3}$$

where the normalization constant $Z_\theta(X)$ is given by

$$Z_\theta(X) = \sum_{Y'} \exp\left(\theta^T F(X, Y')\right). \tag{5.4}$$

During training, we operate on the log-likelihood of the label sequence given the observations. Along with its gradient, the conditional log-likelihood is:

$$\ell(Y|X;\theta) = \frac{1}{n} \sum_{i=1}^{N} \left[\theta^T F(X_i, Y_i) - \log Z_\theta(X_i)\right] \tag{5.5}$$

$$\nabla \ell(\theta) = \frac{1}{n} \sum_{i=1}^{n} \left[F(X_i, Y_i) - \mathbb{E}_\theta[F \mid X_i]\right] \tag{5.6}$$

where $\mathbb{E}_\theta$ represents the expectation under the model with the current parameters θ. Note that the expression in (5.5) is equivalent to (5.1). In other words, when using MLE for parameter estimation in CRFs

$$m_\theta(X_i, Y_i) = \theta^T F(X_i, Y_i) - \log Z_\theta(X_i) \tag{5.7}$$

In the following section, we discuss parameter estimation for conditional random fields using a different function for m_θ. The key property of this new m-function is that it does not require a normalization constant $Z_\theta(X)$ and therefore requires much less computation to evaluate than (5.5).

5.2 A Computationally Efficient M-Estimator

To avoid the expensive computation of $\log Z_\theta(X)$ and its gradient, we propose an alternate objective function that sidesteps the need for this normalization constant. As noted, this estimator was first proposed for log-density ANOVA estimation by [42]. The key property of the new M-estimator is that it starts from a *base model* q_0 and attempts to improve upon q_0 by estimating $\widehat{r}(x,y) = \log p(x,y) - \log q_0(x,y)$, where $p(x,y)$ is the true density of the observation/label pairs (X,Y). For example, q_0 may be a hidden Markov model or defined in terms of a logistic regression model. We compute the estimated density as

$$\widehat{p}(x,y) \propto q_0(x,y)\exp(\widehat{r}(x,y)). \tag{5.8}$$

We assume that $\widehat{r}(x,y) = \widehat{\theta}^T f(x,y)$ is a linear function of feature functions $f(x,y)$ and the M-estimator takes the form

$$\widehat{\theta}_n = \operatorname*{argmin}_\theta \frac{1}{n}\sum_{i=1}^{n} \exp\left(-\theta^T F(X_i,Y_i)\right), +\theta^T \mathbb{E}_{q_0}\left[F\right] \tag{5.9}$$

and the estimated model is then

$$p_{\widehat{\theta}_n}(x,y) \propto q_0(x,y)\exp\left(\widehat{\theta}_n^T f(x,y)\right). \tag{5.10}$$

In the case of sequence models and when q_0 factors according to the edges of a CRF, the conditional probability of the labels is easily computed as:

$$p_{\widehat{\theta}_n}(y \,|\, x) = \frac{q_0(x,y)\exp\left(\widehat{\theta}_n^T f(x,y)\right)}{\sum_y q_0(x,y)\exp\left(\widehat{\theta}_n^T f(x,y)\right)}. \tag{5.11}$$

However, the estimation of the parameter vector in (5.9) does not require the potentially expensive computation of a normalization constant—either for the joint model $p(x,y)$ or the conditional $p(y \,|\, x)$. The computation of the feature expectations under the base model, $\mathbb{E}_{q_0}[f]$, can be computed once and reused during each iteration of the minimization. Moreover, for many classes of base models and feature functions, in particular many hidden Markov models, these expectations can be computed in closed form.

The estimator is based on the M-function

$$m_\theta(X_i,Y_i) = \exp\left(-\theta^T F(X_i,Y_i)\right) + \theta^T \mathbb{E}_{q_0}[F]. \tag{5.12}$$

Therefore, the estimate is $\widehat{\theta}_n = \operatorname{argmin}_\theta M_n(\theta)$, where

$$\begin{aligned} M_n(\theta) =& \frac{1}{n}\sum_{i=1}^{n} m_\theta(X_i, Y_i) \\ =& \frac{1}{n}\sum_{i=1}^{n} \exp\left(-\theta^T F(X_i, Y_i)\right) + \theta^T \mathbb{E}_{q_0}\left[F\right]. \end{aligned} \tag{5.13}$$

In this estimation scheme, the estimate of θ is used to *correct* the density estimate of the base model q_0 rather than to compute the density directly. On an intuitive level, the second term, the expected values of the features under q_0 plays the same role as the normalization constant in traditional CRF training. Increasing the weight w_k for positive F_k in the first term decreases the objective function. Increasing w_k for a positive term $\mathbb{E}_{q_0}\left[F_k\right]$ increases the objective function and prevents w_k from increasing without bound. More formally, suppose that the true density is $p(X, Y)$. Then by the law of large numbers,

$$M_n(\theta) \xrightarrow{P} M_\infty(\theta) = \mathbb{E}_p[e^{-\theta^T F}] + \theta^T \mathbb{E}_{q_0}[F] \tag{5.14}$$

for each θ. Now, suppose that $f^*(X, Y) = \theta^T F(X, Y)$ is an *arbitrary* (measurable) function of X and Y, obtainable by allowing any feature functions. Then, as we show below, the population minimizer of $M_\infty(f^*) = \mathbb{E}_p[e^{-f^*}] + \mathbb{E}_{q_0}[f^*]$ is:

$$f^*(x, y) = \log p(X, Y) - \log q_0(X, Y) \tag{5.15}$$

and the density is thus given by $p(X, Y) \propto q_0(X, Y)e^{f^*(X,Y)}$. Therefore, to estimate $p(X, Y)$, we use the estimator:

$$\widehat{p}(X, Y) \propto q_0(X, Y)e^{\widehat{\theta}_n^T F(X,Y)}. \tag{5.16}$$

Expanding out the derivation of (5.16) in more detail, we estimate the parameter vector $\widehat{\theta}^n$ by minimizing an M-function M_θ. During the minimization, we optimize over θ to minimize equation (5.13). The first step is to expand out the expectation under q_0 on the right side of the equation.

$$\frac{1}{n}\sum_{i=1}^{n} \exp\left(-\theta^T F(X_i, Y_i)\right) + \theta^T \int_{X,Y} F(X, Y) q_0(X, Y)\, dX dY \tag{5.17}$$

The gradient at $\widehat{\theta}^n$, which minimizes the expanded equation above, is equal to zero. If we take $F_k(X, Y)$ as the sum over the sequence for a particular feature f_k then

$$\frac{\partial m_\theta}{\partial \theta_k} = -\frac{1}{n}\sum_{i=1}^{n} \frac{F_k(X_i, Y_i)}{\exp\left(\theta^T F(X_i, Y_i)\right)} + \int_{X,Y} F_k(X, Y) q_0(X, Y)\, dX dY = 0 \tag{5.18}$$

for all of the features in the model f_k. As the number of training sequences n approaches infinity, the first term of the gradient approaches its expectation under the true distribution $p(X,Y)$. In the limit, we can expand the first term of the expectation just like we expanded the expectation under q_0.

$$-\frac{1}{n}\sum_{i=1}^{n}\frac{F_k(X_i,Y_i)}{\exp\left(\theta^T F(X_i,Y_i)\right)} \longrightarrow -\int_{X,Y}\frac{F_k(X,Y)}{\exp\left(\theta^T F(X,Y)\right)}p(X,Y)\,dXdY \tag{5.19}$$

We substitute the expectation under $p(X,Y)$ as $n \to \infty$ into (5.18) to yield an expression that contains both $p(X,Y)$ and $q_o(X,Y)$. The motivation for this step is to create an expression with both those terms that we manipulate to define $p(X,Y)$ in terms of the base model q_0.

$$\int_{X,Y}\frac{F_k(X,Y)}{\exp\left(\theta^T F(X,Y)\right)}p(X,Y)\,dXdY = \int_{X,Y} F_k(X,Y)q_0(X,Y)\,dXdY \tag{5.20}$$

at $\widehat{\theta}^n$, which minimizes m_θ. We see that

$$\frac{p(X,Y)}{\exp\left(\theta^T F(X,Y)\right)} = F_k(X,Y)q_0(X,Y) \tag{5.21}$$

Rearrangement yields (5.16), which defines $p(x,y)$ in terms of the base model q_0 and a correction factor, defined in terms of our features, which encodes the difference between $\log p(X,Y)$ and $\log q_0(X,Y)$. We provide a more rigorous proof of the M-estimator's asymptotic consistency in section 5.2.2.

5.2.1 The Base Model

We experimented with a variety of different base models. In general, our choice of q_0 is guided by three factors.

1. The computational cost of training the base model is an important consideration. Since our goal is to reduce the computational requirements of training a CRF, it makes little sense to choose a q_0 that takes significant resources to train.

2. Efficient inference in the combined model is important. In general, we can use any q_0 that supports the true distribution p. However, if exact inference in q_0 is intractable, then exact inference in the final model will be intractable as well, because we need to evaluate q_0 in order to evaluate $p(X,Y)$ using the final model. Therefore, we prefer base models that factor in such a way as to allow for efficient exact inference.

3. We must be able to efficiently compute feature expectations under the base model. We require $\mathbb{E}_{q_0}[F]$ during M-estimation of the final model parameters $\widehat{\theta}^n$.

In the remainder of this section, we analyze the models that we chose as q_0 with regard to the above points.

Hidden Markov Models

In the case of sequence models, HMMs are a natural choice for q_0. Training a hidden Markov model requires little computation. The MLE or MAP parameter estimates simply require tabulating sufficient statistics. There is no need for iterative numerical optimization as with CRFs. Furthermore, hidden Markov models factor along the same cliques as a CRF. As we described in chapter 2, we can represent any hidden Markov model as a conditional random field. Because HMMs factor in the same way, we can treat q_0 as an ordinary feature in the conditional random field with a fixed weight of 1. Ignoring the prior over initial states, the resulting feature is:

$$f_k(t, y_{t-1}, y_t, X) = p(y_t|y_{t-1})p(x_t|y_t) \tag{5.22}$$

In cases like this where q_0 factors in the same way as a CRF, we can treat the resulting model as a regular conditional random field.

Finally, we need to compute $\mathbb{E}_{q_0}[F]$. Here again, an HMM is an appropriate choice. Computing exact values of the feature expectations is tractable for many classes of features via dynamic programming and knowledge of the stationary distribution of the Markov chain. For the remaining classes of features, it is trivial to sample new sequences from hidden Markov models and the feature expectations are readily approximated by numerical integration. Due to its general applicability and ease of implementation, we use the numerical integration approach in our experiments.

Logistic Regression

The base model q_0 is a joint model over sequences. On the other hand, logistic regression is a conditional model that treats individual labels independently. We get around the conditional nature of logistic regression by using $\tilde{p}(X)$, i.e. the empirical distribution over observation during training, as we explain below in our discussion of our three criteria for evaluating base models.

While parameter estimation in logistic regression is computationally intensive, it is less computationally intensive than in CRFs. We pay a price in moving to a discriminative model for the individual labels, but inference is a factor of $|y_t|$ cheaper than in CRFs. In our experience, training a logistic regression model requires fewer iterations to converge compared to CRFs, which results in additional computational savings. Logistic regression factors according to the same cliques as a CRF, so inference in the final model is efficient.

Our final requirement is for an efficient means of computing $\mathbb{E}_{q_0}[F]$. One method is to sample from the model using the empirical distribution of $\tilde{p}(X)$ in the training set. I.e., we represent the joint $p(X, Y) = p(X)p(Y|X)$ using logistic regression for $p(Y|X) = \prod_t p(y_t|x_t)$ and the training set as an empirical estimate of $p(X)$. When sampling, we copy the true values for the observations into the sampled sequences and draw labels from $p(y_t|x_t)$ in order to build up the label sequence on top of the copied observation sequence. Alternately, we can skip sampling all together. We can compute exact probabilities for state transitions over each edge $p(y_{t-1}, y_t|X)$ from the model. We can accumulate $\mathbb{E}[f(t, y_{t-1}, y_t, X)]$ by scaling by the marginal probabilities of each transition.

Conditional Random Fields

It is possible to use conditional random fields with different or fewer features as the base model q_0. Training a CRF with a limited compliment of features takes less time than training the full model. In particular, when iteratively adding new features during greedy forward feature selection algorithms, the number of active features is much lower than the total number of candidate features. Inference is the final model, which requires evaluating $q_0(X, Y)$ is efficient because the overall model is in fact a CRF. Rather than treating q_0 as a new feature in the final model, we can add the features of q_0 and their weights directly into the final model to form a regular CRF. There is no need to treat the features of q_0 separately from the features of the final model.

As in our discussion of logistic regression as a candidate for q_0, CRFs are also conditional models. We can compute exact feature expectations for CRFs given the empirical distribution of the observations $\tilde{p}(X)$. After all, computing feature expectations is an integral part of the gradient computation for MLE training of CRFs.

5.2.2 Asymptotic Consistency

We present a proof, due to Lafferty, that the estimator (5.13) is asymptotically consistent, such that

$$\widehat{\theta}_n \xrightarrow{P} \theta_* = \inf_\theta M_\infty(\theta) = \inf_\theta \mathbb{E}_p[m_\theta(X,Y)]. \tag{5.23}$$

In particular, when $p(x,y) = q_0(x,y)\exp(\theta_*^T f(x,y))$, then $\widehat{\theta}_n \xrightarrow{P} \theta_*$. (A stronger notion of consistency can be shown in the non-parametric setting where the parameter is the collection of all measurable functions $f(x,y)$.)

We assume that the true density $p(x,y)$ and base model $q_0(x,y)$ are strictly positive: $p(x,y) > 0$ and $q_0(x,y) > 0$. In addition, we assume that the parameter set $\Theta \subset \mathbb{R}^m$ is a bounded subset of Euclidean space, and that the features $f(x,y) \in \mathbb{R}^m$ are uniformly bounded.

First note that $M_\infty(\theta)$ is strictly convex, since

$$p(x,y) > 0 \tag{5.24}$$

$$\nabla^2 M_\infty(\theta) = \mathbb{E}_p\left[\exp(-\theta^T f(x,y))\|f(x,y)\|_2^2\right] > 0. \tag{5.25}$$

Thus,

$$\inf_{\theta:\, \|\theta-\theta_*\|>\epsilon} M_\infty(\theta) > M_\infty(\theta_*) \tag{5.26}$$

for every $\epsilon > 0$. Moreover, it follows from convexity of $m_\theta(x,y)$ in θ that

$$|m_{\theta_1}(x,y) - m_{\theta_2}(x,y)| \le D\|e^{C^T f(x,y)} f(x,y)\|\, \|\theta_1 - \theta_2\| \tag{5.27}$$

for some constant D and $C = \sup_{\theta\in\Theta}|\theta_j|$. Thus, the class of functions $m_\theta(x,y)$ is Glivenko-Cantelli. Together, these results imply consistency (e.g., [113], Theorem 5.7).

Consider now the gradient of the population version of the objective function, $M_\infty(\theta)$:

$$\nabla M_\infty(\theta) = -\mathbb{E}_p[\exp(-\theta^T f(X,Y))] + \mathbb{E}_{q_0}[f] \tag{5.28}$$

If the true density $p(x,y)$ takes the form

$$p(x,y) = q_0(X,Y)\exp(\theta_0^T f(x,y)) \tag{5.29}$$

then $\nabla M_\infty(\theta_0) = 0$. In this case the estimator converges to the true parameter, $\widehat{\theta}_n \to \theta_0$. Since

$$\theta_0^T f(X,Y) = \log p(X,Y) - \log q_0(X,Y), \tag{5.30}$$

the linear model $\widehat{\theta}_n^T f(x,y)$ is an estimate of the log likelihood ratio $\log\left(p(x,y)/q_0(x,y)\right)$ of the true density to the base model.

5.2.3 Parameter Estimation

We estimate $\widehat{\theta}_n$ by minimizing the expression in (5.13). Since $M_n(\theta)$ is both differentiable and convex, optimization algorithms such as conjugate gradient and L-BFGS efficiently find the global optimum. Moreover, for small feature sets, Newton's method is an attractive alternative since the Hessian is readily computed; we cannot use Newton's method for CRFs because the Hessian is not readily computed in the CRF case.

Numerical Issues

Evaluating the objective function poses challenges from a numerical standpoint. The first term computes a sum over $\exp(-\theta^T F(X_i, Y_i))$. The contribution of the second term is linear with respect to theta. Due to the mixed scales of the values, i.e. theta is exponentiated in one term but not the other, we perform all computations using *signed* log-domain arithmetic. As usual, we store the logarithm of each value v rather than the value itself. Unlike our previous discussion of log-domain arithmetic, v can take on negative values. Therefore, we track $\log(|v|)$ as well as a separate bit that stores $\text{sign}(v)$. Multiplication and division with log-domain values is straight forward addition and subtraction. As before, we use

$$\text{add}(a, b) = \begin{cases} a + \log(1 + \exp(b - a)) & \text{if } a > b \\ b + \log(1 + \exp(a - b)) & \text{if } b \geq a \end{cases} \tag{5.31}$$

to perform addition with minimal loss of precision. Adding signed values to this framework merely results in some extra book keeping and special cases to keep track of the signs, but does not change the major principles.

Regularization

We found that regularization plays an important role in parameter estimation. When we applied no penalty or too small of a regularization penalty to the objective function (5.13), the objective function was unbounded from below and the minimization did not terminate except due to numerical overflow. To understand why this is the case, consider the partial derivatives of an *unsupported*, i.e. a feature that never takes on a non-zero value over the training set, feature f_k in the conditional likelihood loss function for a CRF in (5.6) compared with the loss function for the M-estimator.

$$\frac{\partial M_n(\theta)}{\partial \theta_k} = -\frac{1}{n} \sum_{i=1}^{n} \exp\left(-\theta^T F(X_i, Y_i)\right) F_k(X_i, Y_i) + \mathbb{E}_{q_0}\left[F_k\right] \tag{5.32}$$

The first term in both equations is 0 because f_k is zero and therefore F_k is also zero. The second term depends on either the feature expectation under the CRF or, in the case of the M-estimator, the expectation of the feature under q_0. In both cases, the corresponding weight, θ_k will tend towards negative infinity. In the case of the CRF, the expectation of f_k under the model will approach zero as θ_k grows more negative, meaning that the partial with respect to θ_k will also approach zero and the optimization will be well behaved in the CRF case. In the M-estimator, we compute $\mathbb{E}_{q_0}[F_k]$ once at the start of the optimization and that expectation is unaffected by θ_k. The corresponding term in the gradient is constant. As a result, the optimization will not converge to a finite value without regularization to penalize large negative values of θ_k. The unregularized M-estimator assigns infinite negative weight to features that are deemed "impossible" by their absence from the training set, which can be viewed as an extreme form of over-fitting. The model learns that features that lack support in the training set can never take on non-zero values. We therefore regularized the M-estimator with an ℓ_2 penalty in all of our experiments.

5.2.4 Inference

Inference in the M-estimator, via the forward-backward algorithm, is essentially the same as inference in a conditional random field. In particular, computing the likelihood of a new sequence is simply a matter of running forward-backward, and then multiplying by q_0. In cases where q_0 factors over the edges in the label sequence, as is the case for the HMM, logistic regression, and CRF base models, the base model can be simply treated as a feature in the CRF with a weight of 1. When q_0 takes such a form, the computation of the marginals $p(y_t \mid X)$ using forward-backward, or computing the most probable labeling using the Viterbi algorithm is efficient. In cases where q_0 does not factor neatly over the labels, inference may be intractable, as we discussed in section 5.2.1.

5.3 Experiments

We present an empirical evaluation of the behavior and performance of conditional random fields trained using the proposed M-estimator. We use synthetic data, the robot tag domain described in chapter 2, recorded RoboCup robot soccer games, and a text chunking domain. We included this last domain to validate the M-estimator on a non-activity recognition data set as well as to use a publicly available data set to validate the accuracy of our models against the results of others. We designed our experiments to test several aspects of the M-estimator.

- We used synthetic data to validate that the M-estimator can use rich features to learn concepts that cannot be represented in the base model q_0.
- We consider the issue of training efficiency. The M-estimator is asymptotically consistent. Given an infinite number of training sequences, the model will converge to the true density. In practice, we work with finite amounts of training data. We again use a mixture of synthetic and real data to explore the issues of data efficiency.
- Parameter estimation in the M-estimator requires computing feature expectations under the base model. We use sampling in order to generate estimates of $\mathbb{E}_{q_0}[F]$. We experiment to see how many samples are required in order to achieve good learning performance.
- Regularization is required in the presence of unsupported features and, in general, is important for high classification accuracy. We examine the role of regularization in training CRFs with the M-estimator.
- Our principle motivation for using the M-estimator is to reduce the training time of our models. We use accuracy versus training time as a metric to compare the trade offs between classification accuracy and training time.

5.3.1 Synthetic Data

Traditional CRF training maximizes the conditional likelihood of the training set. The first term of the conditional log-likelihood $w^T F(X, Y)$ is linear and grows without bound. The normalization constant is a penalty which bounds the conditional log-likelihood. Training balances the increase in the objective function from first term due to large absolute values of weights w_i against the increased penalty charged by the normalization term. The gradient of the log-likelihood, the sum of the features over the training set minus the expected values of the features under the model, shows the trade-off between choosing large weights for the features frequently seen in the training set and the penalty charged for those weights over all possible label sequences. When the gradient is zero at the maximum log-likelihood, the feature sums are exactly equal to the feature expectations under the model. Any increase in the objective function by increasing the weight associated with a particular feature is offset by the increased penalty charged by the normalization constant.

The M-estimator uses feature expectations under the base model q_0 in place of the traditional normalization constant. Improvements in the M-function due to the sums of features over the training set are offset by penalties to the objective function due to the expectations of features under q_0. In the M-estimator, there is no longer a direct link between the expected

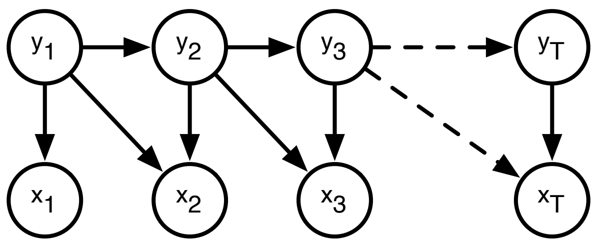


Figure 5.1: *Second-order HMM Structure:* The graphical structure for a second-order HMM. We used a second-order HMM as a benchmark for our synthetic data because it accurately captures the second-order dynamics of the generated sequences.

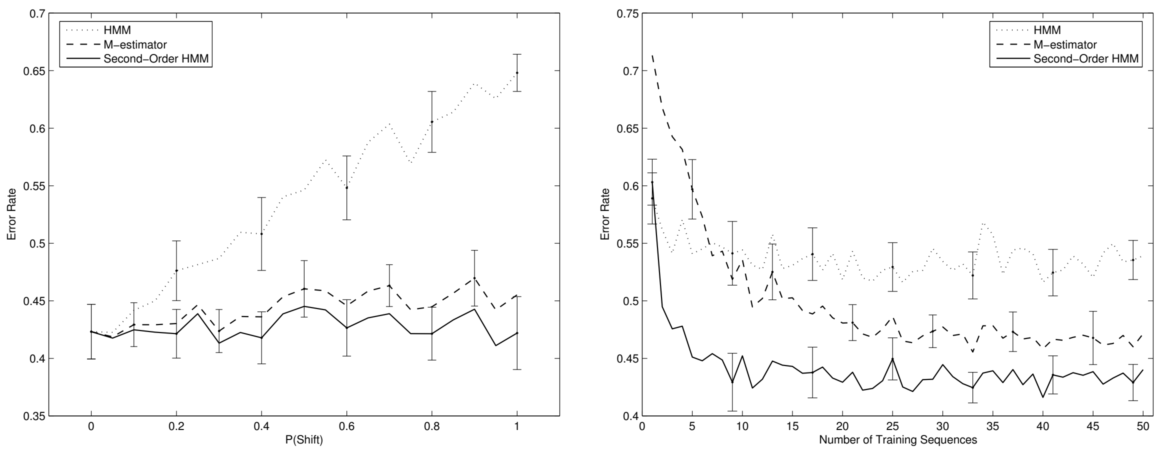
values of features under the model and the size of the "penalty" due to the normalization term. We explore this issue in our first set of experiments. We test whether or not the M-estimator can produce a model that accurately classifies data that the base model cannot effectively classify. Our rational for these experiments is that when q_0 cannot accurately capture the concepts represented by a feature f_k, i.e. f_k is required for classification and q_0 lacks any analog to f_k, then $\mathbb{E}_{q_0}[F_k]$ is a poor approximation to $\mathbb{E}_p[F_k]$, the normalization term in the gradient of traditional log-likelihood. When q_0 is a poor approximation of p, q_0 is a poor approximation of the traditional normalization term.

We created synthetic data with second-order Markov dynamics. A traditional HMM is defined in terms of $p(y_t|y_{t-1})$ and $p(x_t|y_t)$. We randomly generated parameters for these two conditional probability tables and then sampled sequences in the usual way, with one important difference. When drawing observations x_t from the observation model $p(x_t|y_t)$, we used the label from the previous time step y_{t-1} in the place of y_t with probability p_{shift}. In other words,

$$\mathrm{x_t} \sim \begin{cases} p(x_t|y_t = \mathrm{y_t}) & \text{with probability } p_{\text{shift}} \\ p(x_t|y_t = \mathrm{y_{t\text{-}1}}) & \text{with probability } 1 - p_{\text{shift}} \end{cases} \tag{5.33}$$

When $p_{\text{shift}} = 0$, the sampled sequences are drawn from a traditional HMM. To add on a new time step at t, we draw the label $\mathrm{y_t}$ from $p(y_t|y_{t-1})$. And then we use the true value $\mathrm{y_t}$ to draw an observation from $p(x_t|y_t)$. When $p_{\text{shift}} = 1$, it is as if the observations in the HMM are all moved one time step to the right. To add a new time step at t, we draw $\mathrm{y_t}$ from $p(y_t|y_{t-1})$ in the usual way. Then we use the value of the *previous label* $\mathrm{y_{t\text{-}1}}$ to sample from $p(x_t|y_t = \mathrm{y_{t\text{-}1}})$. The parameter p_{shift} can be viewed as a mixing constant that determines which regime is used to generate observations.

We used a hidden Markov model as q_0. As p_{shift} varies from 0 to 1, the sampled data ranges



a) Varying p_{shift} b) Varying n

Figure 5.2: *M-estimator Synthetic Results:* a) The parameter p_{shift} controls the degree to which second-order effects in the data make a first-order HMM inadequate. As p_{shift} increases, the first-order HMM's error rate increases linearly, but a second-order HMM and an M-estimated model with second-order features have consistently low error rates. b) The data contain second-order effects that a first-order HMM cannot model accurately ($p_{\text{shift}} = .5$). As the number n of training sequences increases, the M-estimated model achieves a lower error rate than the first-order HMM (which serves as the base model for the M-estimator). The improvement is apparent with fewer than ten examples. The lower error rate in the second-order HMM may be due in part to the effects of regularization, which makes M-estimation inconsistent.

from data that q_0 can capture perfectly, to data where q_0 cannot model the observations at all. The base model is not entirely useless, as it can model the Markov transition dynamics even when $p_{\text{shift}} = 1$, but its accuracy is severely impaired. We use a second order HMM, with the structure shown in figure 5.1, as a comparison because it can accurately model the second order dynamics of the synthetic data.

Figure 5.2 shows classification accuracy versus p_{shift} for an HMM, the M-estimator, and a second order HMM. The M-estimator included features that made it equivalent to a discriminatively trained second-order HMM. We trained the models on 1,000 training sequences where each sequence had a length of 100 time steps. When p_{shift} is close to zero, the HMM is an acceptable model for the data and all three models exhibit equivalent performance. However, as p_{shift} increases, the performance of the HMM decreases dramatically as less and less observation information is available to it. The second-order HMM and the M-estimator do not suffer nearly as much. The former did not suffer because it contains a link from y_{t-1} to x_t and the latter did not suffer because it included features of the form: $f(t, y_{t-1}, y_t, X) = (y_{t-1} \overset{?}{=} i)(y_t \overset{?}{=} j)(x_t \overset{?}{=} v)$ as well as the usual HMM-equivalent observation features $f(t, y_{t-1}, y_t, X) = (y_t \overset{?}{=} j)(x_t \overset{?}{=} v)$.

The M-function operates on feature sums over entire sequences and provides an asymptotic guarantee of consistency. We require an infinite number of training sequences to be sure that the parameters produced by the M-estimator truly encode $p(X, Y)$ in terms of q_0 and the features f_k (to the extent that the model is capable of representing the data). In our second set of experiments, we examine the relationship between classification accuracy and the number of training sequences used during parameter estimation. We use the same second order model to generate training data with the fixed parameter $p_{\text{shift}} = .5$. Figure 5.2 shows classification for an HMM, the M-estimator with an HMM base model, and a second-order HMM. When the number of training sequences n is small, the HMM outperforms the M-estimator. However, as n is increased, the M-estimator quickly comes to dominate the HMM and its error rate approaches the error rate of the second-order HMM.

5.3.2 Robot Tag

We return to the robot tag domain of chapter 2 as a source of data that requires complex features for accurate classification, but where the level of complexity is manageable. We have *a priori* knowledge that a small set of key features capture the properties of the domain and produce high classification accuracy. In other words, the robot tag domain is challenging enough to require complex features, but due to our previous experience, we know which features to include in the model.

We use the robot tag domain to investigate the issues surrounding our choice of the base model q_0. In particular:

- We compare the classification accuracy of the M-estimator to the accuracy of a CRF that contains the same features.
- We compare the training time required for parameter estimation in the M-estimator versus the CRF.
- We explore how the properties of q_0 affect the M-estimator. We vary the number of samples that we use to estimate $\mathbb{E}_{q_0}[F]$, we consider smoothing in the base model, and we explore different models for q_0.

The Tag Domain and a Simplified Feature Set

To briefly review the domain, the robot tag domain is inspired by the children's game of Tag. Three robots move freely in an approximately 3x3 meter playing area. Two of the robots take on passive roles and navigate to a series of points on the field. The two passive robots are oblivious to other robots, except they will avoid direct collisions. The third robot, called the seeker, takes an active role. The seeker robot attempts to *tag* its closest neighbor and thereby transfer the seeker role to the tagged robot. The seeker role is transferred whenever the seeker approaches within 50 cm of another robot. Once the seeker role is transferred, the new seeker pauses in place briefly to allow the former seeker to escape without immediately being tagged in return.

The observations in the Tag domain are the real valued positions, in millimeters, of the three robots on the field; the observation vector x_t is a 6-tuple that contains 2 coordinates per robot. The label at each time step y_t identifies which of the three robots is currently executing the seeker role. The set of labels is then $\{1, 2, 3\}$, where the label is the ID number of the current seeker.

In our experiments in chapter 2, we identified a small number of key concepts. In particular, we found that distance threshold tests, which evaluate when one robot moves to within a fixed distance of another, are useful for capturing the act of tagging. We also found that the velocities of the robots and the notion of one robot "chasing" another, i.e. moving towards it, are important. We give the full set of features and the number of weights associated with each prototype in table 5.1. Overall, the model contained 210 weights and the features were binary tests of relevant properties. The model contained only highly relevant features.

An Auto-regressive HMM Base Model

We used an auto-regressive HMM, as shown in figure 5.3, for the base model q_0 in our tag experiments. An auto-regressive HMM has the same structure as a traditional HMM, with an added link between the observation at time $t-1$ and the observation at time t. The additional dependency enhances the model's performance in domains where the observation at time t is determined both by the current label y_t and the previous observation x_{t-1}. Models with this type of auto-regressive structure perform well at detecting discrete states from data in robotics (e.g. [62]). In robot domains, quantities like position and orientation vary smoothly over time. The role or behavior of the robot affects how it changes its position, but that change is relative to its previous position, so an auto-regressive HMM is a good model. Because the observation sequence is always known during inference, adding the additional

Prototype	Size	Description
$(y_t \overset{?}{=} \text{label})$	3	Standard intercept feature
$(y_{t-1} \overset{?}{=} \text{label})(y_t \overset{?}{=} \text{label})$	9	First-order Markov dynamics
$(y_t \overset{?}{=} \text{label})(v_s^T(p_{c_s} - p_s) \overset{?}{>} 0)$	9	Is robot s "chasing" (moving toward) its closest neighbor c_s? p_x represents the position vector for robot x and v_x represents the velocity vector for robot x.
$(y_t \overset{?}{=} \text{label})(v_s^T(p_{c_s} - p_s) \overset{?}{\leq} 0)$	9	Not "chasing"
$(y_{t-1} \overset{?}{=} \text{label})(y_t \overset{?}{=} \text{label})(d_{i,j} \overset{?}{\leq} k)$	81	Tests if the distance between robots i and j is less than a threshold k for $k \in \{24, 35, 50\}$ and all possible pairs of i, j.
$(y_{t-1} \overset{?}{=} \text{label})(y_t \overset{?}{=} \text{label})(d_{i,j} \overset{?}{>} k)$	81	Tests if the distance between robots i and j is more than a threshold k centimeters for $k \in \{24, 35, 50\}$ and all possible pairs of i, j.
$(y_t \overset{?}{=} \text{label})(v_i \overset{?}{\leq} k)$	9	Tests if robot i is moving at less than k mm/s for $k = 160$.
$(y_t \overset{?}{=} \text{label})(v_i \overset{?}{>} k)$	9	Tests if robot i is moving at more than k mm/s for $k = 160$.

Table 5.1: Features in the M-estimator for the Tag domain

link does not increase the computational cost of classification using the model. Similarly, we can efficiently sample sequences from an auto-regressive HMM in order to estimate $\mathbb{E}_{q_0}[F]$. While an auto-regressive HMM will outperform a regular HMM, as our experiments show, a conditional random field outperforms both models in terms of classification accuracy because neither variety of HMM incorporates complex, high level features. Our choice of an auto-regressive HMM for q_0 is motivated by the training efficiency of the base model rather than its classification accuracy.

Comparing the M-estimator to a CRF

We investigated the classification accuracy and training time of the M-estimator, based around an auto-regressive HMM q_0, versus a conditional random field. Figure 5.4 (a) shows the error rate of the M-estimator versus the error rate of a conditional random field. The

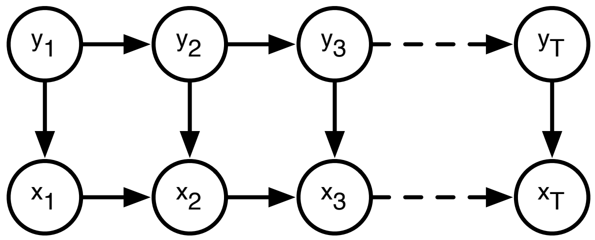
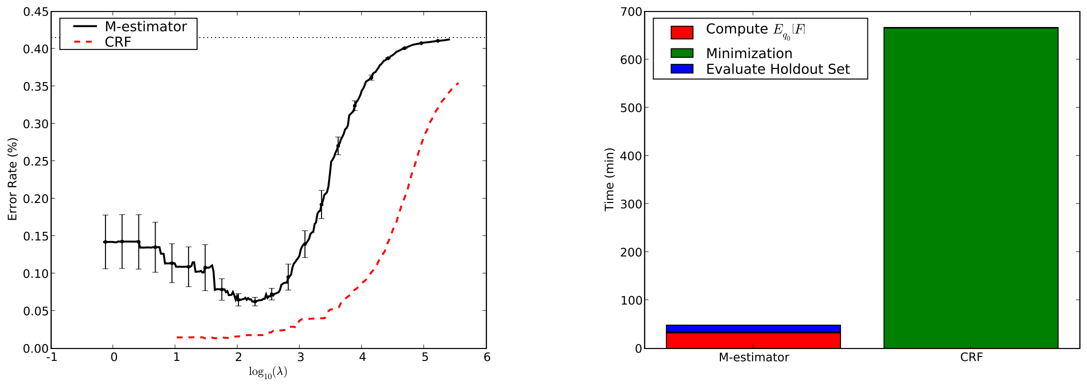


Figure 5.3: *Auto-regressive HMM Structure:* An HMM augmented with a link between successive observations. The additional link aids the model in the presence of auto-regressive data where the observation at time t, x_t, is heavily influenced by the previous observation x_{t-1} in addition to the label variable. Robot positions, which vary smoothly as the robot moves through space, have this characteristic; the position of the robot changes according to its role, but the robots new position depends on its previous position as well.

error rate for the auto-regressive HMM, shown by the dotted horizontal line, is much higher than the error rate of the CRF. The labels are distributed uniformly among the three possible values, so random prediction yields an error rate of 66%. The M-estimator closes a significant portion of the gap between the auto-regressive HMM and the CRF. The CRF retains a performance advantage, but the gap between the CRF and the M-estimator is smaller than the gap between the M-estimator and q_0, which demonstrates that the M-estimator can repair a somewhat mediocre model to yield high classification accuracy.

The M-estimator cannot completely close the gap to a properly trained conditional model and shows evidence of over-fitting. We used an ℓ_2 penalty to smooth the models during training. Even under weak smoothing, the CRF shows no signs of over-fitting. The error rate for the M-estimator increases for low values of λ, which suggests that the M-estimator is more prone to over-fitting than a CRF.

While the difference in accuracy between the M-estimator and the CRF is relatively small, there is a large difference in the training times for the two models. Figure 5.4 (b) shows the training times for the two models. We break training time in the M-estimator down into the time required to estimate $\mathbb{E}_{q_0}[F]$, by drawing samples from q_0; the time required to estimate the model parameters, via repeated rounds L-BFGS with successively lower regularization penalties; and the time required to evaluate the parameter vectors against a holdout set to choose the final model. For the CRF, we only break training time down into repeated rounds of L-BFGS as the regularization parameter λ is varied and evaluation against the holdout set. Computing the feature expectations under q_0 is the most computationally expensive part of training the M-estimator. The feature expectations need only be computed once and



a) Classification Accuracy

b) Training Time

Figure 5.4: *M-estimator Accuracy:* Error rates for the M-estimator and a conditional random field with identical features versus the regularization parameter λ when both models were trained under an ℓ_2 penalty. The M-estimator used an auto-regressive HMM as q_0. The error rate of the base model is shown as a dotted horizontal line.

can be reused with any number of different values λ for ℓ_2 regularization. Furthermore, as we will see below, we were very generous in sampling from q_0 for these experiments. We drew 250 samples for each of 500 training sequences. In practice, the M-estimator performs well when we draw many fewer sequences during estimation of $\mathbb{E}_{q_0}[F]$. The time required to train the CRF was dominated by repeated iterations of L-BFGS under different values of λ. Again, our experimental setup favored the CRF. Because, as we discuss below, the M-estimator is very sensitive to regularization, we evaluated 250 candidate values of λ with the M-estimator. After all, training with different values of λ is fairly cheap. We limited the CRF to 100 iterations. The bar for the CRF is only $\frac{2}{5}$ths of the length it would be if we'd considered 250 λs for the CRF as well. As a final point, we note that evaluation, while negligible in the CRF, accounts for a larger relative percentage of the M-estimators training time. Inference in the M-estimator is slower than in a CRF because we need to compute $q_0(X, Y)$ during inference. For some types of base models, we can transform the model into a CRF and remove this slow down, but for these experiments we simply evaluated $q_0(y_t|y_{t-1})$ and $q_0(x_t|y_t, x_{t-1})$ during belief propagation in the CRF.

Computing Feature Expectations

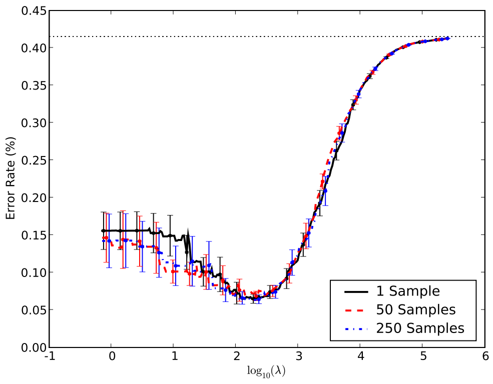
To estimate parameters using the M-estimator, we require feature expectations under the base model q_0. In the case where q_0 is a generative model, such as an HMM or an auto-regressive HMM, estimating $\mathbb{E}_{q_0}[F]$ by drawing sample sequences from the base model is trivial. We sample new sequences from the base model and then compute the average feature sums over the sampled sequences.

We make a trade-off when we use sampling to estimate feature expectations. Averaging over many samples produces a more accurate estimate of $\mathbb{E}_{q_0}[F]$ at the expense of increased training time. Conversely, averaging over few samples yields a less accurate estimate in less time. However, even in cases where many samples are required, feature expectations are only computed once at the start of training. The expectations under q_0 do not depend on the parameter vector and can be reused, even across different rounds of ℓ_2 penalized minimization.

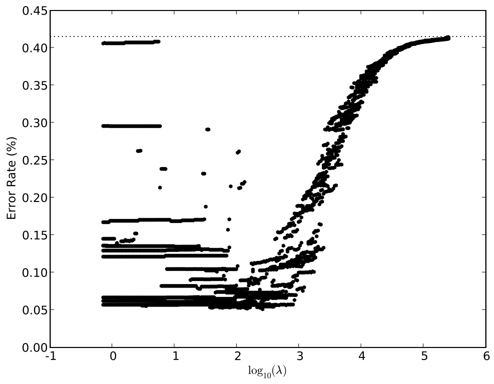
We tested the sensitivity of the M-estimator to the number of samples drawn to estimate $\mathbb{E}_{q_0}[F]$. For each sequence in the training set, we sampled k sequences with the same length as the original training sequence. We sampled new values for the observations and labels in each of the k sequences, but we used the length from the same sequence in the training set for all k sampled sequences. So, for a training set with n sequences, we averaged $F(X_i, Y_i)$ over nk samples to estimate $\mathbb{E}_{q_0}[F]$. The sequences in the nk long sampled set were divided into n blocks of k sequences where each sequence in a block has the same length as a particular sequence in the training set. The observation and label values in all nk sequences were sampled directly from q_0 and *not* copied from the training sequences.

Figure 5.5 (a) shows error rate versus the normalization parameter λ for $n = 500$ training sequences and $k \in \{1, 50, 250\}$. We show the average error rate and .95 confidence intervals for 25 trials. Varying the number of samples that we use to estimate $\mathbb{E}_{q_0}[F]$ does not significantly affect the error rate of the final model or its regularization curve. In all of the experiments, the .95 confidence intervals increase in size for low values of the regularization parameter, i.e. when there is little smoothing. The increase in the size of the confidence intervals is due to increased variance in the error rates of the models. Figure 5.5 (b) shows the unaveraged points for the $k = 250$ experiments. For large values of λ, i.e. under strong regularization, the distribution of the error rates has low variance and lies close to the mean. Once λ falls too low, i.e. there is too little regularization, the variance increases and the distribution of error rates is more widely distributed. The M-estimator is poorly behaved with too little regularization.

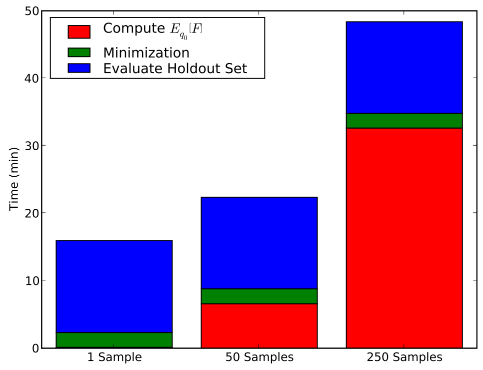
The number of samples that we draw when estimating feature expectations has a significant



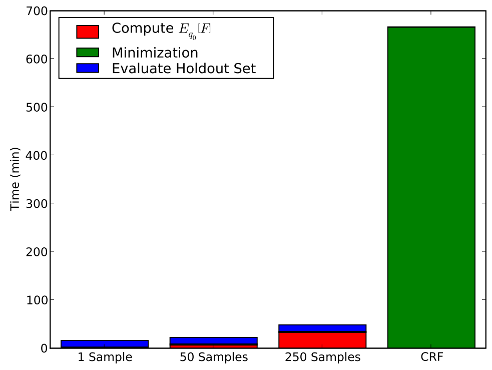
a) Error Rate versus λ



b) 250 Samples – Individual Error Rates



c) Training Time



d) Training Time (with CRF for scale)

Figure 5.5: *Feature Expectations under q_0:* To compute $\mathbb{E}_{q_0}[F]$, we sampled 500, 25,000, or 125,000 new sequences from q_0. We sampled both observations and labels, but used the lengths from the training set. For $n = 500$ sequences in the training set, we drew 1, 50, and 250 samples *per* sequence in the training set. We use these smaller values for the labels in the plots. a) Mean accuracy versus the smoothing parameter λ with .95 confidence intervals. b) Accuracy versus λ for $k = 250$ samples per training sequence over 25 trials. c) Training time for the M-estimator. d) Also training time for the M-estimator, but with the CRF's training time included for scale.

impact on the training time of the M-estimator. Figure 5.5 (c and d) break the training time for the M-estimator down into its component parts. For small values of k, the cost of sampling all but disappears from the training time and evaluating candidate parameter vectors on the holdout set dominates the training times. For large values of k, sampling is the dominant term. Since accuracy does not seem to suffer from small values of k (e.g. when we only draw n samples for $k = 1$), it seems that the dominant cost for using the M-estimator for training is evaluating the candidate models that it produces. We already evaluate on a smaller holdout set than training set (50 versus 500) sequences. We could further reduce the training time by only evaluating a subset of the candidate models and "zooming in" on regions of the regularization path with low error rates.

Smoothing the Base Model

The M-estimator over-fits to the training data under weak regularization. We use ℓ_2 regularization to smooth the M-estimator during training to achieve high classification accuracy. To understand why smoothing is required with the M-estimator, we consider whether the base model q_0 is peaked or smoothed. In particular, we can rearrange (5.15) to show that the M-estimator represents the log-odds ratio of the true distribution to q_0.

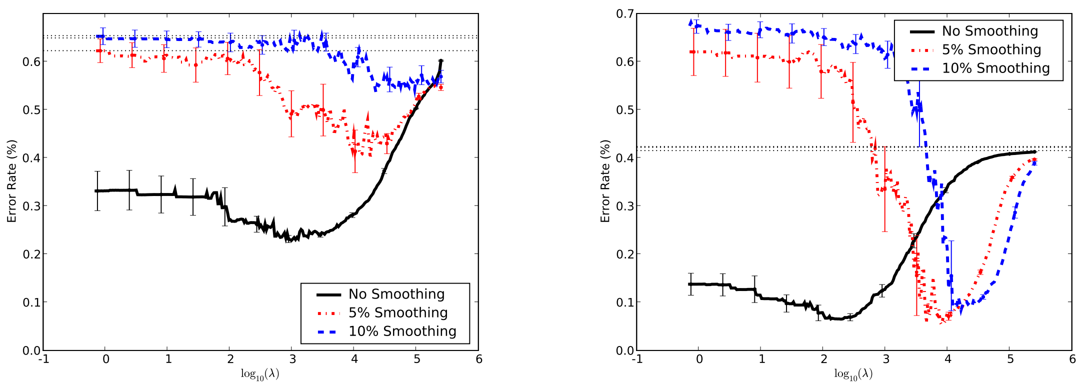
$$\theta^T F(X, Y) = \log \frac{p(X, Y)}{q_0(X, Y)} \tag{5.34}$$

Because q_0 appears in the denominator of the log-odds ratio, q_0 must support the true distribution p. The base model must report non-zero probability for any sequence that might arise from the true distribution. More formally,

$$p(X, Y) > 0 \rightarrow q_0(X, Y) > 0 \tag{5.35}$$

The relationship between q_0 and p may explain why the M-estimator performs poorly under weak regularization. If the training set contains sequences for which $q_0(X, Y)$ is small relative to $p(X, Y)$ then elements in θ must take on large values in order to represent the ratio of p to q_0. In the extreme case, where $q_0(X, Y)$ is zero $p(X, Y)$ is non-zero, only the regularization term prevents weights in θ from growing without bound. The accuracy of the final model may suffer when q_0 offers poor support for p and too little regularization allows weights in θ to take on large values. When predicting labels for sequences drawn from p, the artificially large weights in q_0 my move probability away from the sequences that are unsupported or under-supported by q_0.

We tested the hypothesis that a peaked base model can harm classification accuracy in two ways. First, we applied smoothing to the transition matrix of our auto-regressive HMM to increase the probability of state transitions. We also used a different model as q_0; we ran a set of experiments using an HMM rather than an auto-regressive HMM as q_0. The "smoothing" in this case comes from the fact that an HMM is a more parsimonious model



a) HMM q_0 b) auto-regressive HMM q_0

Figure 5.6: *Smoothed* q_0 *Results:* Error rate versus λ for an HMM and an auto-regressive HMM. The smoothing parameter λ was used during parameter estimation in the M-estimator, not the base model q_0. We smoothed the base models by mixing a uniform distribution into the maximum likelihood parameter estimates for the transition matrices of the HMMs.

than an auto-regressive HMM. The regular HMM contains fewer parameters, which makes it less likely to over-fit to the training data. We also experimented with smoothing the transition matrix of the traditional HMM.

To smooth the transition matrices, we performed the usual maximum likelihood parameter estimation for the models (essentially counting), and then mixed in values from a uniform distribution over transitions. We used a scalar α to control the amount of smoothing according to:

$$p_{smoothed}(y_t|y_{t-1}) = (1-\alpha)p(y_t|y_{t-1}) + \frac{\alpha}{|y_t|} \tag{5.36}$$

We used values $\alpha \in \{0, .05, .1\}$, which correspond to no smoothing, 5%, and 10% smoothing of the transitions.

The results shown in figure 5.6 suggest that smoothing q_0 *harms* classification accuracy. Switching to an HMM from an auto-regressive HMM significantly increases the error rate. The HMMs, with various amounts of smoothing, achieve approximately the same error rate as random guessing. The M-estimator, with an HMM as q_0, shows significantly higher error rates than when an auto-regressive HMM is used for q_0. Smoothing the transition matrix of the HMM increases the error rate of the M-estimator, although it does not appreciably shift the performance of the base model. Smoothing the auto-regressive HMM does not yield as dramatic a decrease in classification accuracy, but it does change the regularization curve of the models. When we smooth transitions in the auto-regressive HMM, more smoothing of the M-estimator is required for low error rates. The error rate for the M-estimator that uses an unsmoothed auto-regressive HMM as q_0 does not climb above the error rate of the base model for any value of λ that we tested. The error rates for the smoothed auto-regressive HMMs do rise above the error rate of the base model. With too little regularization, the M-estimator performs worse than q_0. As with the HMM, the error rate of the auto-regressive HMM base model does not change appreciably with the smoothing. The issue with smoothing is not that it affects the error rate of the base model, it is that smoothing changes the shape of q_0.

Conditional Base Models

We used a conditional random field, with parameters estimated via pseudo-likelihood, with the same features as the M-estimator for q_0. I.e., we use the M-estimator to patch a different fast, approximate training method (pseudo-likelihood) for the full model. We also experimented with a smaller set of features; we eliminated all of the transition-linked features to create a logistic regression model for q_0. Logistic regression makes the same independence assumptions across time steps as pseudo-likelihood, except it does not use the training set to fill in the labels for adjacent time steps because it only uses the current time step and observation sequence to define its features.

To compute feature expectations under q_0, we decompose the joint probability $q_0(X, Y)$ into $\tilde{p}(X)q_o(Y|X)$ using the identity $p(a, b) = p(a)p(b|a)$. The quantity $\tilde{p}(X)$ represents the empirical distribution of the observations in the training set. We compute feature expectations under q_0 either by Gibbs sampling or exactly via the same dynamic programming approach that we use to compute the gradient of the CRF log-likelihood.

To generate a labeled sequence by Gibbs sampling, we copy one of the labeled training sequences (both labels and observations). And then we use repeated rounds of Gibbs sampling to draw a new sequence of labels by holding the variables in the Markov blanket for each label y_t fixed and drawing a new label $y_t \sim p(y_t|X, y_{t-1}, y_{t+1})$. As in the generative q_0 case, we average the feature expectations over several sampled sequences for each training sequence.

Figure 5.7 shows that the CRF trained by pseudo-likelihood (our first base models) performs poorly. The parameters estimated via pseudo-likelihood are as accurate as randomly guessing the labels with an error rate of around 66%. The M-estimator that uses a CRF with the pseudo-likelihood parameters does not improve on the pseudo-likelihood parameters as all. In our particular data set, the labels at y_{t-1} and y_t are highly correlated; the state labels change relatively infrequently compared to the 60 sample per second data rate. Pseudo-likelihood uses the true values of adjacent labels during training and, in our setting, simply learns to predict that transitions never occur. (This is a known problem with pseudo-likelihood training, e.g. [29]) The resulting q_0 is a terrible model of the true distribution p and the M-estimator cannot improve upon it in any appreciable way.

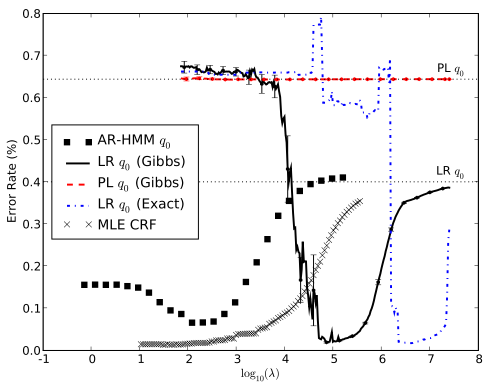
We also used logistic regression as a base model. Logistic regression does not fill in correct labels from adjacent time steps during training and it produced an error rate of approximately 40% after training. We used feature expectations from Gibbs sampling as well as exact expectations from dynamic programming to train M-estimators around the LR base model. In both cases, the error rate dropped below the best error rate for the auto-regressive HMM q_0 to approximately the same level as the error rate in a CRF – around 2%. The exact expectation version reached its minimum error rate under larger values of the regularization constant λ than the M-estimator that used Gibbs sampling to compute $\mathbb{E}_{q_0}[F]$. Both M-estimators showed high error rates under too little regularization.

The training times displayed in figure 5.7 (b) show that the M-estimator, with conditional q_0, offers significant reduction in training time over a CRF that uses MLE for parameter estimation. Both conditional base models required more training time than the auto-regressive HMM. The training time for the later is too low to be visible in the plot. The logistic regression style q_0 required less training time than the full featured version simply because the logistic regression model contained fewer features; we used the same pseudo-likelihood training with both models. Gibbs sampling to compute feature expectations in the conditional models required a significant amount of time. Computing feature expectations exactly was much more efficient. The logistic regression base model, with exact feature expectations under q_0 required approximately the same training time as the AR-HMM q_0 and produced error rates comparable to the full CRF.

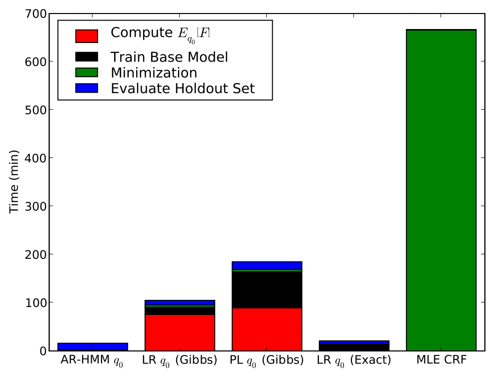
Scaling to Larger Feature Vectors

We used the M-estimator for parameter estimation with larger feature vectors. Specifically, we used the set of 1,200 real valued features for the tag domain from chapter 3. Our previous experiments showed that M-estimation can correct the logistic regression base model to achieve nearly the same classification accuracy as a CRF with simple, binary features. Here, we test if M-estimation can correct the same base model using a larger and more complex set of features.

Figure 5.8 shows the error rate on test data versus the regularization constant λ. The M-estimator, using both Gibbs sampling and exact expectations under q_0, improves upon the accuracy of the base model. However, the error rate achieved with the larger feature vector



a) Error Rates



b) Training Times

Figure 5.7: *Conditional Base Models:* a) The ℓ_2 regularization paths for the M-estimator when a logistic regression model and a CRF, trained by pseudo-likelihood, are used as q_0. Regularization paths for a CRF, trained by maximum likelihood, and the M-estimator, with an auto-regressive HMM base model, are provided for comparison. The M-estimator performs poorly, with an error rate equivalent to random guessing, with the pseudo-likelihood base model. When a logistic regression model is used as q_0, the error rate of the M-estimator approaches the error rate of the MLE CRF and outperforms the error rate of the M-estimator with an auto-regressive HMM as q_0. The error rates for the two conditionally trained base models are shown as horizontal dashed lines. The lines for the base models are labeled on the right of the plot rather than in the key. b) The training time for the conditional q_0 M-estimators. We include the MLE CRF and AR HMM q_0 M-estimator for scale. For the conditional q_0 M-estimators, we include the time required to train q_0 as a separate category. Training time for q_0 is either not applicable or negligible (and accounted for in the "Minimization" category because it is a form of parameter estimation) in the other two models.

is significantly higher than the error rate that we achieved using simple features to correct q_0, even though the base model was identical in both situations.

To investigate why the accuracy of the M-estimator decrease with the larger feature vector, we trained a CRF by maximum likelihood with the set of 1,200 tag features and q_0 as an additional feature. In our previous experiments, we initialized this type of CRF with parameters from the M-estimator in order to classify the test set. Here, we applied regular maximum likelihood training to the augmented model to compute the MLE parameters for a CRF that includes q_0. We do not expect the MLE parameters to exactly match the parameters produced by M-estimation because the M-estimator is a joint model and a CRF is, of course, a conditional model. However, the MLE parameters achieve an extremely low error rate and provide a point of comparison.

Figure 5.9 (a) plots the MLE parameter estimates for each weight versus the M-estimated parameter estimate for that same weight. We ran 25 trials using Gibbs sampling for feature expectations, since that worked better than computing the expectations exactly. The MLE parameters do not vary, so we estimated the MLE parameters once. We divide the weights in plot (a) into four categories:

- We used an X to mark weights that take on relatively large negative values in both models. These parameters have the same sign in both models and their relative magnitudes are large compared to the other features in each model. Intuitively, these weights correspond to useful features that the M-estimator "found".
- We used a horizontal dash to mark weights that are non-zero in the CRF but close to zero in the M-estimator. The features associated with these weights carry information about the labels, otherwise MLE would not produce non-zero weights. However, the M-estimator did assign significant weight to them. Intuitively, the weights correspond to useful features that the M-estimator "missed".
- We used a vertical dash to indicate features with non-zero weights in the M-estimator but near zero weights in the CRF. These weights lie near zero in the MLE parameter vector for a CRF (that includes q_0), which indicates that they are relatively unimportant for predicting the label sequence. If we had used a generative q_0, it would be possible that the features associated with these weights helped model the observations, because the M-estimator models $P(X, Y)$. However, we used a conditional model for q_0 with the empirical distribution $\tilde{p}(X)$, i.e. we copied the observations verbatim and used $p(Y|X)$ to sample from q_0. Intuitively, we can say that these are "junk" weights that the M-estimator assigns non-zero values to these parameters, but maximum likelihood training does not.
- The final group of weights is a catch-all "other" category that includes weights that do not fall into our first three categories. We drew these weights as small circles.

We plotted the feature expectations under the base model $\mathbb{E}_{q_0}[F_i]$ versus the average empirical feature sums $\bar{F}_i$ (figure 5.9 (b)). Most of the weights fall along the line $y = x$. I.e. the feature expectations match the feature sums over the training set. However, there is a notable outlier. The feature expectations for the first group of parameters, the ones with large negative values in both models, are far larger than the empirical sums for those features in the training set. The most significant weights (and the ones that are "right") found

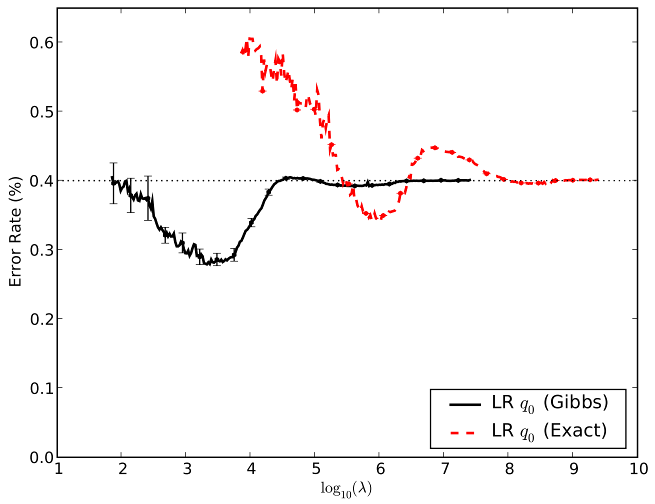


Figure 5.8: *Rich Features in the M-estimator:* The error rate versus the regularization parameter λ for the M-estimator that uses the same logistic regression base model as our previous experiments.

by the M-estimator correspond to the largest gap between feature expectations and feature sums. The expectations for some of the "junk" also show marked differences between the empirical and expected counts, which suggests the (not surprising) conclusion that the M-estimator produces larger parameter estimates for features that produce different values over the training set versus under q_0.

Figures 5.9 (c) and (d) show parameter estimates versus $\bar{F}_i$ for the MLE and M-estimated parameters respectively. The MLE parameter estimates take on non-zero values across a wide range of values for $\bar{F}_i$, but the most significant parameters in the M-estimator fall close to $\bar{F}_i = 0$. Many of the significant parameters that the M-estimator missed, shown by horizontal dashes, fall far from $\bar{F}_i = 0$. Plots (e) and (f) show parameter values versus $\mathbb{E}_{q_0}[F_i]$ for the two models. The "junk" features (vertical dashes) with the largest weights have the largest expectations under q_0. Those same features have the smallest empirical sums, which suggests that a large ratio of $\mathbb{E}_{q_0}$ to F_i results in large weights in the M-estimator. The "good" features that the M-estimator missed (horizontal dashes) have larger F_i and smaller $\mathbb{E}_{q_0}[F_i]$, which may explain why the M-estimator missed those weights – they were masked by the first group of "good" features (X marks) and the "junk" features (vertical dashes).

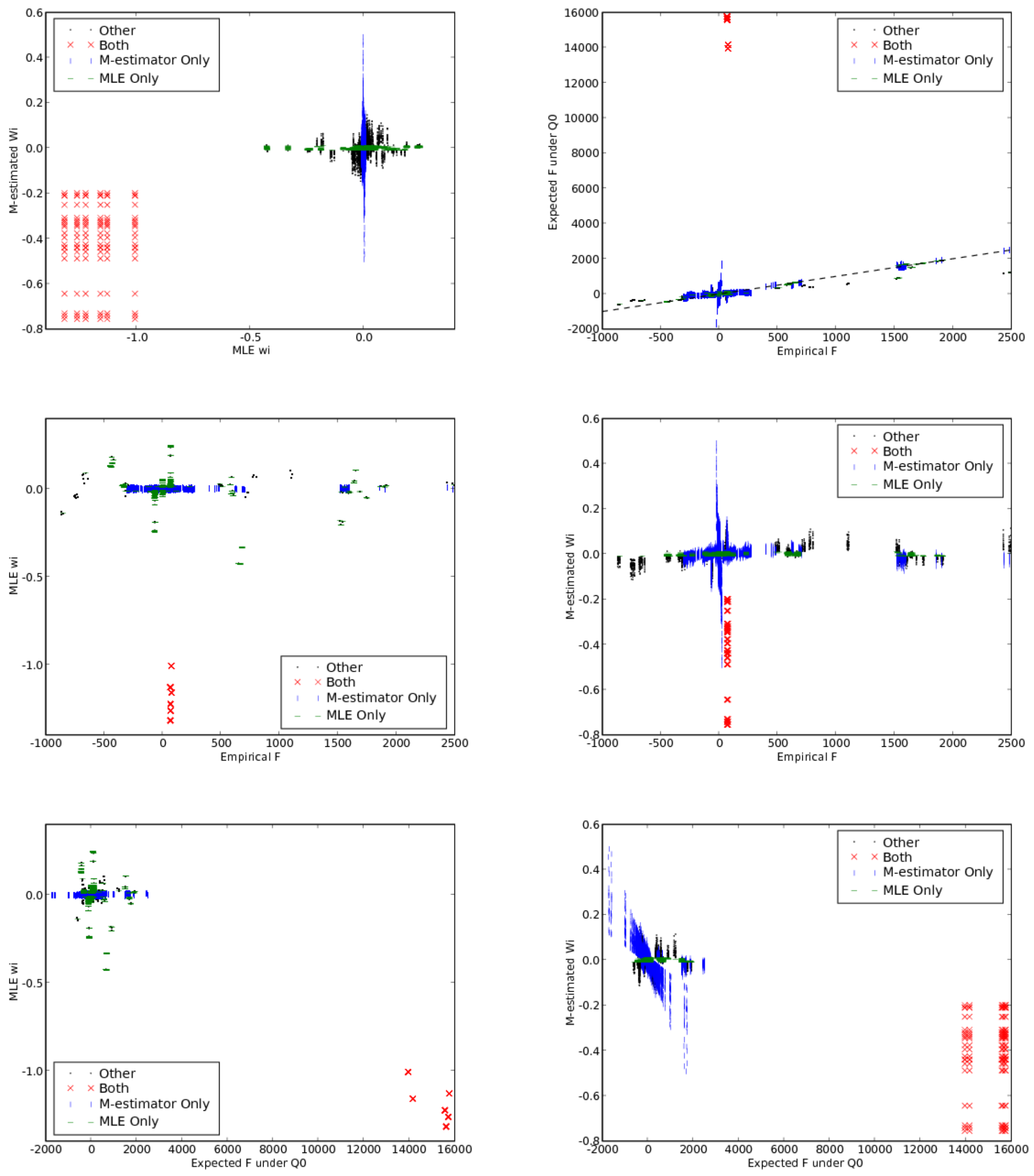


Figure 5.9: *M-estimated Parameters:*

5.3.3 Shallow Parsing

Natural language processing domains feature extremely large feature vectors. Unlike in our activity recognition data, the feature vectors usually contain only binary features and

	Base Model		M-Estimator		CRF	
Iterations	F_1	Time	F_1	Time	F_1	Time
25	70.84	00:07	89.62	00:31 (00:38)	78.08	05:22
50	86.85	00:13	90.85	00:15 (00:28)	90.53	11:05
100	90.19	00:25	91.65	00:14 (00:38)	92.19	20:02
150	90.70	00:33	91.83	00:13 (00:46)	92.64	28:02
200	90.73	00:46	91.80	00:12 (00:58)	92.88	35:56
250	90.85	00:54	91.66	00:12 (01:06)	92.95	44:02
500	90.69	01:47	91.13	00:11 (01:58)	92.96	84:04

Table 5.2: *Shallow Parsing Results:* F_1 accuracy and training time from the logistic regression base model, the M-estimator, and a CRF. Each row contains results from when the base model and CRF were limited to a maximum number of iterations of L-BFGS during training. (The M-estimation was always allowed up to 500 L-BFGS iterations, though it frequently converged in fewer.) Training times are given in hours:minutes. Parenthesized values under the M-estimator column include the time required to train the base model.

the vectors are extremely sparse. We experimented in a text chunking domain in order to apply the M-estimator with large vectors of simple features. We previously reported similar experiments that were joint work with Noah Smith in [98].

In this set of experiments, we applied the M-estimator model to the CoNLL 2000 shared task, which is a shallow parsing task that requires identifying phrases in text documents [107]. The data for this task came from English language newspaper articles [94]. Given observations consisting of an English word and its part of speech tag, the goal is to produce a series of labels identifying the start of non-recursive linguistic phrases, e.g. "Begin Noun Phrase," as well as labels denoting "Inside Noun Phrase," etc.

In total, there are 22 distinct labels and 45 part of speech tags, which make up the second observed variable. We smoothed the first observed variable, the word in the sentence, by replacing the first occurrence of each word in the training set with an out-of-vocabulary marker. Second, and subsequent, appearances of words in the training set were treated as observations of that word. This smoothing reduced the cardinality of the first observed variable to slightly more than 9,000 values.

Conditional random fields have previously been applied to the identification of noun phrases, one of the phrase types to be identified in this task [97]. We used Sha and Pereira's feature functions over labels and pairs of words/tags and triples of tags in our CRF and M-estimator. We did not include the unsupported features, which Sha and Pereira reported were used in their highest scoring experiments. In the logistic regression base model, we used a subset of the features (those confined to a single label) to speed parameter estimation in q_0; we used features over labels and words/tags as well as over pairs of tags, but the total number of features was less than a tenth of the CRF and M-estimator.

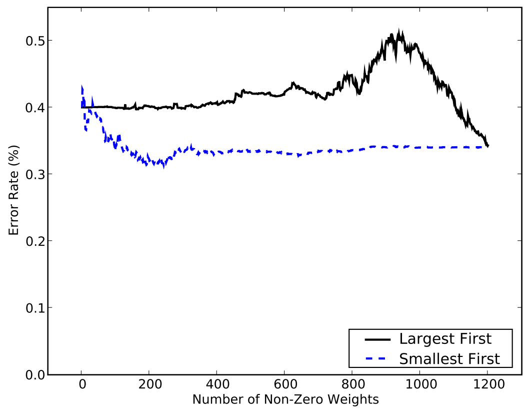
Table 5.2 shows the results of training the M-estimator using logistic regression as the base model. During optimization, we limited the number of iterations of L-BFGS when estimating the model parameters of the base model and the CRF to range from 25 to 500. Parameter estimation in the M-estimator, using cached feature sums and constant expectations from the base model, was limited to 500 iterations in all trials and typically converged before this limit. In general, the base model performed well, and the M-estimator performed at an intermediate level between the base model and the CRF. In the trial where the base model was limited to 25 iterations of training, there is a larger gap between the base model and the CRF. In this case, the M-estimator performs substantially better than the base model, but not as well as the best trials of the CRF.

5.4 Feature Selection

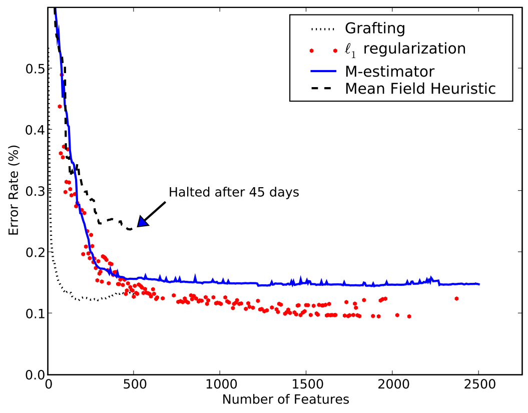
Grafting, which relies on a simple gradient calculation to choose new features, is the most computationally efficient method of feature selection in CRFs. However, grafting tends to over-fit and produces less accurate models than less greedy algorithms like the mean field heuristic and ℓ_1 regularization. In chapter 4, where we discussed scaling feature selection to very large sets of candidate features, we said that the mean field heuristic does not scale to large candidate sets because computing the approximate gain for individual features required too much computational effort. In this section, we explore using the M-estimator as an intermediate between an algorithm that uses discriminative training to compute the approximate gain for adding individual features (the mean field heuristic) and a purely greedy algorithm (grafting) that uses the gradient of the log-likelihood to select features.

M-estimation uses discriminative training to discover a parameter vector that moves the distribution under the base model $q_0(X, Y)$ closer to the true distribution $p(X, Y)$. If follows then that large weights in the M-estimator correspond to important features for closing the gap between q_0 and p. To perform feature selection with the M-estimator we:

- Take a CRF with no features as q_0
- Repeat:



a) Pruning Weights



b) Error Rate vs. Model Size

Figure 5.10: a) We estimated parameters for a CRF with 1,200 features using the M-estimator and data from the robot tag domain. We removed parameters one by one, by setting them to zero, and computed the loss of the trimmed models on test data. The x-axis represents the size of the model, so pruning moved from right to left. We pruned the parameters starting with the largest absolute weight and moving toward the smallest absolute weight. We also pruned in the opposite direction from the smallest absolute weights to the largest absolute weights. b) Error rate versus model size for feature selection via grafting, ℓ_1 regularization, the mean field heuristic, and using the M-estimator. We limited the mean field heuristic to at most 500 features and allowed the M-estimator to consider up 2500 features because the former took 45 days of CPU time to reach the 500 feature mark. We show the average over three different training sets (two games per training set) for all of the algorithms except the mean field heuristic. We show performance for the mean field heuristic over a single training set due to the high computational cost of that algorithm.

– Estimate parameters on over the candidate features via M-estimation
– Add the candidate with the largest absolute weight to q_0
– Retrain q_0 by MLE

Figure 5.10 (a) provides the rational for our feature selection algorithm. We used the M-estimator to discover parameters in a CRF with 1200 complex features. In the previous section, we saw that the resulting model performed relatively poorly compared to the model with simple binary features. We then zeroed out the weights one by one and computed the error rate on test data after removing each parameter. We started from the largest absolute weights and moved towards smaller absolute weights. We also zeroed out the weights in the opposite order from the smallest absolute weight to the largest. The purpose of these experiments was to verify that the largest parameters in the M-estimator correspond to the most useful features (for lowering the error rate). These experiments show that our assumption is valid; when we prune the largest absolute weights first, the error rate immediately increases. Once the largest weights are gone, removing the weights with intermediate values increases the error rate above that of the base model. Finally, once all the weights are zeroed out, the model is equivalent to q_0.

Moving in the opposite direction, when we prune the smallest absolute weights first, we do not see a significant change in the error rate until we begin to remove features with intermediate weights. The error rate decreases slightly as these harmful features are removed and then increases as the useful features (with larger weights) are pruned. These experiments show that useful features take on large weights under the M-estimator.

Figure 5.10 (b) shows the results of applying the M-estimator to feature selection in full RoboCup domain with 92,310 candidate features. We compare against grafting, the mean field heuristic, and ℓ_1 regularization. We stop grafting after is chooses 500 features because the error rate begins to increase due to over-fitting for larger models. We stop the mean field heuristic after choosing 500 features (and limit it to one data set rather than using three different training sets as with the other algorithms), because it takes the mean field heuristic approximately 45 days to select 500 features. We allow ℓ_1 regularization to choose any number of features but restrict it to 250 different values of the regularization parameter λ and we allow the M-estimator to consider up to 2500 features. The M-estimator produces results that are intermediate between grafting and the mean field heuristic. However, grafting and ℓ_1 regularization both produce more accurate models.

Table 5.3 gives a summary of our feature selection experiments and considers the case where we retrain the models with ℓ_2 smoothing after feature selection. ℓ_1 regularization produces relatively large models with the lowest error rate. Grafting produces smaller models with higher error rates than ℓ_1 regularization. The mean field heuristic requires by far the most computation and produces the least accurate models. Our feature selection algorithm based on the M-estimator is less accurate than ℓ_1 regularization and grafting, but more accurate than the mean field heuristic. The M-estimator based algorithm is significantly more efficient than the mean field heuristic and requires approximately the same amount of time *per feature* as grafting. We allow the M-estimator to select five times as many features as grafting because it did not show the same tendency to over-fit, which increased the over-all run time for the M-estimator based algorithm. We selected the "best" candidate model using a holdout set, which produced artificially large models with the M-estimator; a human choosing a model would probably choose to cut off the long, flat tail in the models produced by the M-estimator based feature selection algorithm.

Training	Error Rate	Model Size	Time (hours)
ML / ℓ_2	10.5%	92310	11.2
ℓ_1	10.3%	1823	6.9
Grafting	12.3%	220	1.9
Grafting / ℓ_2	12.0%	220	0.6 (2.5)
Mean Field Heuristic	23.8%*	483*	1091.0*
MF Heuristic / ℓ_2	21.8%*	483*	1.0 (1092.0)*
M-estimator	14.9%	1694	59.8
M-estimator / ℓ_2	14.0%	1694	1.3 (61.1)

Table 5.3: We trained on data from three earlier games and tested with data from the final. We compared maximum likelihood training with ℓ_2 smoothing (the no feature selection case); ℓ_1 regularization; grafting by itself; grafting followed by ℓ_2 smoothing; the mean field heuristic; the mean field heuristic followed by ℓ_2 smoothing; our M-estimator feature selection algorithm; and our M-estimator based algorithm followed by ℓ_2 smoothing. We show average error rates on test data for the chosen subset of features, the size of the final model, and the training time required to discover the final model. When feature selection was followed by smoothing, we show the time required for smoothing followed by the cumulative time for feature selection and smoothing in parentheses. (*) For the mean field heuristic, we used a single training set rather than averaging over three data sets due to the computational cost of that algorithm.

Our proposed feature selection algorithm, that uses M-estimation to choose new features in a forward-selection algorithm and traditional MLE parameter estimation with the selected features, provides a viable intermediate between greedy algorithms like grafting and more computationally intensive algorithms like the mean field heuristic. However, with our data, the M-estimator based algorithm required more computation and produced less accurate models than either grafting or ℓ_1 regularization, so the practical utility of the algorithm seems limited.

5.5 Chapter Summary

We have shown how a computationally efficient M-estimator, which was recently proposed by [42] in the context of certain non-parametric ANOVA models, can be used for fast training of log-linear models for structured domains. The proposed estimator offers a computational advantage over the traditional method of parameter estimation in such models, i.e., maximizing the conditional likelihood, by eliminating the need to compute a normalizing constant over all labelings. It does this by estimating parameters that represent the difference between the true density p and a base model q_0 rather than estimating p directly.

We adapted the M-estimator for feature selection and showed that the resulting feature selection algorithm can identify important features from a large candidate set in an activity recognition domain. The resulting feature selection algorithm requires significantly less computation than algorithms such as the mean field heuristic, although in our domain, grafting and ℓ_1 regularization proved to be more suitable choices for feature selection.

In summary, we:

- Adapted a computationally efficient M-estimator for fast, approximate parameter estimation in undirected graphical models.
- Explored the properties of the M-estimator using a variety of real and synthetic data.
- Introduced a feature selection algorithm based on the M-estimator that offers an intermediate approach between greedy algorithms like grafting and more discriminative approaches such as the mean field gain heuristic.

Chapter 6

Related Work

We organize our discussion of the related work into three categories. We begin with a discussion of the various models that are used for activity recognition with a focus on activity recognition from sensor data using graphical models. In the second section, we turn to feature selection. There is an extensive feature selection literature and we focus on feature selection in log-linear models in general and in conditional random fields in particular. Since one of our focuses is on efficient training for conditional random fields, we conclude our discussion of the related work with an overview of fast training methods for CRFs.

6.1 Activity Recognition

We organize our discussion of activity recognition according to the models that were used for classification. We begin with the most simple models and more towards more complex models that are more closely related to the work in this thesis.

6.1.1 Traditional Classifiers

A variety of non-temporal classification methods, such as decision trees and support vector machines, have been used to classify sensor data, e.g. [5, 95]. In particular, accelerometer data is often used for activity recognition. In general, non-temporal approaches use features such as means, variances, and Fourier transforms computed over windows of the data to perform ad-hoc temporal smoothing. Indeed, although we do not discuss it in this thesis, we used decision trees trained on window statistics to classify states, such as detecting when legged robots are entangled or the type of surface that a legged robot is walking across, from accelerometer data [110].

Discriminative training is straight forward with non-temporal appraoches because adjacent time steps are treated as independent, so the presense of features computed over windows of sensor data does not result in over-counting of the evidence. However, such approaches cannot smooth the label sequence over time because, which is why we focus on a temporal model (i.e. conditional random fields) in this thesis.

6.1.2 Hidden Markov Models

Hidden Markov models [25, 91] are the canonical models for sequential or time series classification. Hidden Markov models have been applied extensively in the domain of activity recognition. Particularly relevant to our work, Han and Veloso used hidden Markov models, built using hand selected observations, to classify behaviors in the Small Size League of RoboCup [38]. In a less closely related multi-agent military domain, Sukthankar and Sycara used a hidden Markov model to classify team behaviors. In particular, they used spatial relationships between the agents on teach team as well as the spatial relationships between agents and landmarks in the environment as the input features to the HMM [99].

As we showed in chapter 2, generative models, such as HMMs, cannot robustly incorporate complex, non-independent features of their inputs. The complex spatial relationships that we use for activity recognition are non-independent, which is one of our main motivations for using conditional models. Lester et al. used a hybrid model to avoid this issue and obtain the benefits of discriminative training as well as the temporal smoothing that comes from using an HMM [63]. They used boosting to discriminatively train ensembles for human activity recognition from data collected by a wearable sensor board. They used ensembles of boosted decision stumps to compute marginal probabilities over the activity states and passed these marginal probabilities to a hidden Markov model for temporal smoothing.

6.1.3 Augmented Markov Models

Augmented Markov models use the behavior or activity labels of agents as a representational substrate to model the environment [32–34]. On a practical level, augmented Markov models are similar to HMMs with noise-free observations. Each state emits a single symbol, but the same symbol may be emitted by multiple states. Augmented Markov models maintain auxilliary statistics about state duration and the frequency of link traverals on a per state or per transition basis. They provide a first-order Markov representation that allows for reasoning about higher order [Markov] behavior. Augmented Markov models use the joint role of a team of robots as input to classify states of the world. They abstract away the noise and uncertainty of real sensors by operating at a higher level of abstraction. The focus of this thesis is on selecting features to classify individual robot roles and behaviors. Our work can be viewed as enabling more abstract reasoning over robot roles and behaviors.

6.1.4 Dynamic Baysian Networks

Dynamic Bayesian networks [76] extend Bayes nets [84] to temporal domains. They can be viewed as a superset of hidden Markov models that allow for a much richer set of relationships among random variables. For example, they allow for factored state, e.g. factorial hidden Markov models [30], and can represent complex relationships, e.g. hierarchy, among model variables. DBNs are popular models for activity recognition from sensor data.

For example, Patterson et al. used a DBN to track human subjects on a map using noisy GPS traces [83]. The representational power of DBNs allowed them to explicitly model factors such as bus stop and parking lot locations while simultaneously estimating the position and mode of transportation for the subject. In follow-on work, Liao et al. increased the depth of the hierarchy to include factors such as the end destination of the user. They used a hierarchical hidden Markov model, based on the abstract hidden Markov model of Bui et al. [17], to model the users position and to detect deviation from his or her normal routines [66, 68].

In general, adding hierarchy improves classification accuracy, e.g. [27, 80], although it results in models where exact inference is intractable. In particular, methods such loopy belief propagation [75] or Rao-Blackwellized particle filters [24] are required to perform approximate inference. Raj et al. describe inference via a Rao-Blackwellized partical filter for activity recognition using a hierarchical DBN in detail [92].

6.1.5 Conditional Random Fields

As newer models, conditional random fields have received less attention than HMMs or DBNs, but they are used for activity recognition as well as in related domains that require rich, continuous features such as image segmentation [54], object recognition [88], and gesture recognition [116]. In activity recognition domains, Truyen et al. found that discriminative models such as CRFs and maximum entropy Markov models [74] significantly outperform generative models like HMMs for recognizing human activities from images in videos [108].

In closely related work, Liao et al. used a hierarchical conditional random field to recognition human activities from GPS traces [65, 69]. The key contribution of their work is that they show how to build hierarchical CRFs for activity recognition. They perform a first pass of classification over the data in order to predict activity labels for the sequence. They use the predicted activities from the first pass to identify *signficant places*, such as home, work, and friends' houses, for the person whose activities are being labeled. They then perform a second round of prediction in the hierarchical model, which now includes the signficant places as unobserved random variables. They show that the label sequence from the hierarchical model is more accurate than the label predictions from the first pass in the flat model, even though the flat model was used to determine the structure of the hierarchical model. When training their models, they used pseudo-likelihood for parameter estimation. For inference in the complete model, where exact inference is not tractable due to cyclic structure, they used loopy belief propagation to predict labels over test sequences.

6.1.6 Relational Models

Relational Markov networks are an extension of conditional random fields that use a *relational language* to specify the cliques in the graph [105]. Like CRFs, RMNs are undirected graphical models and therefore their clique potentials are the functions of the variables that comprise the cliques in the graph. Rather than using a fixed functional form for their cliques, as in CRFs, which specify their clique potentials as $\psi_t(y_{t-1}, y_t, X)$, RMNs define their cliques using a collection of *schema* or templates. Schema are defined in terms of *entities* and *attributes*. For example, in natural language domains, e.g. [2], individual documents might be the entities and a possible attribute is *refers to*. For a given data set, a Markov random field is unrolled or *instantiated* according to the rules embodied in the schema and the usual inference technques are applied to the unrolled graph. Typically, this graph is densly interconnected and exact inference is intractable [105].

Liao et al. used relational Markov networks for human activity recognition [64]. In their earlier work with CRFs [66], they dynamically created a model structure based on a preliminary classification using a flat model. In this work, they formally defined the clique structures to include notions such as signfificant places using relational Markov networks. In chapter 4, we make extensive use of relational *features* for activity recognition. We define a relational language for feature specification within conditional random fields. This is different than defining the model *structure* using a relational language. The models in our work are true conditional random fields. In particular, they retain the simple clique structure for linear-chain CRFs $\psi_t = \exp(w^T f(t, y_{t-1}, y_t, X))$ and we perform fast, exact inference in our models. Liao et al. defined rich model structures in terms of a relational language and then used advanced, approximate inference techniques to predict labels within the complex model structures. We defined rich *features* in a simple model structure and performed fast inference over that simple structure.

6.1.7 Non-Parametric Models

We focus on selecting appropriate features for activity recognition. Non-parameteric methods avoid the need to define features by using the data itself as the model representation. Lenser and Veloso used non-parametric statistics to detect discrete environmental states from streams of robot sensor data [60–62]. They used an auto-regressive HMM with a rich, non-parametric observation model $p(x_t|x_{t-1}, y_t)$, where x_{t-1} and x_t represent the current and previous observations (or model emmissions) and y_t represents the label at the current time t. They used raw sensor data from robots to *detect and adapt* to properties of the environment such as the type of carpet under a legged robot and to changes in lighting conditions. Kernel conditional random fields are a non-parametric CRF representation [55] and non-parametric methods are applicable to activity recognition with CRFs. Our focus is on using rich features of the observations rather than the observations themselves to build our models.

6.2 Feature Selection

There is an extensive feature selection literature dating back more than 40 years, e.g. via feature selection via forward-selection [72]. Traditional feature selection methods focus on relatively small collections of features (hundreds) and it is only in recent years that models with many thousands or millions of features have received wider attention [36]. In our discussion, we focus on features that are useful for activity recognition and feature selection in the specific case of generalized linear models [77], which includes CRFs and logistic regression. We provide a more general discussion of feature selection in chapter 3. Further general discussion is available from Guyon and Elisseeff [36]; Blum and Langley [10]; as well as Kohavi and John [51].

6.2.1 Features for Activity Recognition

The first advice that Guyon and Elisseeff offer in their survey is to build domain specific features where possible [36]. In the case of activity recognition, spatial features, that capture the relationships between agents or between agents and important objects in their environment are particularly helpful, e.g. the work of Sukthankar and Sycara, discussed above [99]. Yan and Mataric also used spatial features for robot activity recogntition. They used features computed over clusters of robots, such as the size of the clusters and the area taken up by the cluster in a multi-robot domain [117]. We designed our relational feature specification language so that it is easy to specify complex spatial relationships in light of the fact that spatial features are important for accurate recognition.

Huynh and Schiele used features such as fast Fourier transform coefficients computed over windows of accelerometer data to recognize human activities such as walking or riding the bus [41]. They found that while they could pick out a general class of features, e.g. FFT coefficients, as important for the classification, individual activities depended on different scales of those features. This motivated our decision to select features at the granularity of individual edges (i.e. by individual weights) rather than activating features for all combinations of label pairs.

6.2.2 Training under an ℓ_1 penalty

Training models under an ℓ_1 penalty offers simultaneous smoothing and feature selection [39]. In case of linear regression, an efficient algorithm, LARS, exists for recovering the entire ℓ_1 regularization path, i.e. parameter values across all possible values of the scalar penalty term λ, because the regularization path is piece-wise linear [26,106]. Training a generalized linear model with an ℓ_1 penalty is equivalent to solving a quadratic optimization problem and we cannot efficiently compute the full regularization path. Despite the difficulty of training, ℓ_1 regularization is an appealling method for feature selection because it can effectively prune irrelevant features; in [79], Ng shows that the sample complexity of ℓ_1 regularization grows logarithmically with the number of irrelevant features. In other words, the number of training sequences required to select the correct features is logarithmic in the total number of features and ℓ_1 is an efficient method for feature selection.

A wide variety of methods have been proposed for ℓ_1 training. Grafting, if care is taken to explicitly deactivate features when their weights cross zero, allows training with an ℓ_1 penalty [86]. Riezler and Vasserman [96] used the scalar penalty parameter λ as a threshold to determine which features to activate during each iteration of grafting. They explicitly removed features as their weights become zero. The coordinate-wise optimization approach that we describe in chapter 4 is similar in that we threshold the set of active features using λ, except that we do not deactivate features that cross zero. The same feature can enter and exit the model an arbitrary number of times under our scheme and the orthant-wise L-BFGS method that we use for training is more computationally efficient.

The gradient LASSO method of Kim and Kim provides a simple, constraint based algorithm for ℓ_1 penalized training based on the dual of the penalized objective function. The dual version of the problem constrains the ℓ_1 norm of the weight vector rather than charging a penalty proportional to the ℓ_1 norm [46]. Empirically, we found that numerical optimization with projected conjugate gradient or L-BFGS to be much more efficient.

Park and Hastie use a path following algirhtm based on the predictor-corrector method to trace the full regularization path of ℓ_1 regularized generalized linear models [82]. In our applications, we do not require the full regularization path. We merely require parameter vectors sampled from along the regularization path. Following the full regularization path for the model tends to be computationally expensive, which is why we apply other methods to estimate parameters for a discrete set of λ values along the full path.

Koh et al. used a log-barrier method [11] to train logistic regression under an ℓ_1 penalty. Their method does not produce weights that are exactly equal to zero. They prune the weight vector based on the duality gap between the log-barrier version of the problem and the true penalized objective function [50]. In practice, we had difficultly implementing their method to efficiently recover sprase models.

An easier to implement method for ℓ_1 penalized training is due to Kazama and Tsujii [45]. They present a method for training generalized linear models with inequality constraints. Their method is very similar to the reparameterization that we used in our early ℓ_1 experiments where the weight vector w is reparameterized into w^+ and w^- and we constrain the two new vectors to the non-negative portion of the parameter space.

The most efficient method of ℓ_1 penalized training that we found is due to Andrew and Gao. They propose orthant-wise L-BFGS for training generalized linear models under an ℓ_1 penalty. Their key insight is that the second order term in L-BFGS does not depend on the penalty term. They omit the penalty term from the second order updates and only use it to update the first order terms when computing the search direction [3].

6.2.3 Forward Selection

There are a variety of feature selection methods based around the idea of incremental forward selection, where we start with an empty model and add features one at a time using a scoring heuristic to evaluate which feature to add next to the existing model. In the context of maximum entropy models for text classification, Berger et al. proposed an incremental feature selection algorithm based on an approximate gain computation [6]. To compute the approximate gain for adding a given feature, they fixed the weights of all other features at their current values and performed a one-dimensional minimization over the weight of the candidate feature. The resulting improvement in the likelihood of the training data served as the gain. McCallum's mean field based gain heuristic [73] is similar to Berger et al.'s scoring function. In both cases, the computational requirements for scoring large sets of candidate features are prohibitive. Zhou et al. introduced a modification to Berger et al.'s algorithm to reduce the computational requirements of feature selection [118]. They note that the feature gain estimates are relatively constant between iterations. After adding a new feature, they re-evaluate the features from the second best feature downwards. They stop re-evaluating features when they encounter a new gain that dominates the remaining [old] gain scores to reduce the number of candidates that they consider each iteration. In the setting of Markov random fields, Della Pietra et al. incrementally added features to Markov random fields for text classification [21].

6.2.4 Virtual Evidence Boosting

Virtual evidence boosting [67] is an extension to McCallum's feature induction algorithm [73] that flexibly addresses continuous observations and provides additional smoothing during training. VEB uses decision stumps as weak learners in an extension of LogitBoost in order to flexibly model continuous observations. It provides additional temporal smoothing by marginalizing over adjacent labels rather than making a pseudo-likelihood assumption or mean field assumption where only the most likely label is used. VEB also allows for semi-supervised training. Rather than labeling data to include fixed beginnings and endings for labeled segments, VEB allows for probabilistic labels that smoothly transition between states to avoid fixed labels for intermediate periods when the system is transitioning from one state to another [71].

6.3 Fast Training for CRFs

In chapter 5, we describe a method for fast, approximate training in CRFs. There are a wide variety of other approaches for fast CRF training, which we outline in this section.

6.3.1 Maximum Pseudo-Likelihood

Maximum pseudo-likelihood training is a long used method for fast, approximate parameter estimation in undirected graphs [8, 9]. Pseudo-likelihood breaks the model down into a collection of independent variables by conditioning each variable on the values of its neighbors. In other words, pseudo-likelihood fills in the values for each variable's Markov blanket using the observed values from the training set. Far less computational effort is required for inference under such strict independence assumptions because belief propagation is no longer required.

6.3.2 Piece-wise Training

Recent work on accelerating training in conditional random fields for natural language processing includes piece-wise training. In piece-wise training, undirected graphs that contain loops are broken down into a series of disjoint models. The parameters for the disjoint models are estimated independently and then recombined to form the full model. Piece-wise training offers improved accuracy over pseudo-likelihood, which makes stricter independence assumptions, and offers substantial computational savings over multiple rounds of believe propagation (i.e. loopy belief propagation) [81]. Further computational gains can be had, at the expense of accuracy, but using pseudo-likelihood for training within the disjoint sections of the model, as in [103].

6.3.3 Sparse Belief Propagation

Maximum pseudo-likelihood and piece-wise training simplify the structure of the model to quickly estimate approximate parameters for the model. And alternative to simplifying the model is to change belief propagation in the model. Pal et al. investigated beam search to speed inference in linear-chain CRFs by limitting the number of messages between time steps [81, 104] and analyzed their technique in terms of variational methods [44]. In graphs that contain loops, Sutton and McCallum investigated dynamic belief propagation schedules to improve convergence rates for loopy belief propagation [102].

6.3.4 Stochastic Gradient Methods

Stochastic gradient methods have been extensively applied in convex optimization settings. In particular, Vishwanathan et al. have used a stochastic gradient method to accelerate parameter estimation in conditional random fields [114]. The key property of these methods is that they compute the gradient over a subsample of the training set to approximate the gradient in terms of the entire training set. In domains with large amounts of redundent training data, stochastic gradient methods can converge orders of magnitude faster than MLE training on the full data set.

Chapter 7

Conclusion

Our goal in this thesis is to explore methods for fast, accurate activity recognition. We are interested in methods that perform well when faced with the complex, noisy observations available from robot sensors and we desire classifiers that are applicable in the computationally limited environments found on most robots.

Conditional random fields are particularly well suited to classification based on noisy sensor data. Conditional random fields can robustly incorporate arbitrary functions (features) of the observations. Complex features are essential for extracting meaningful information from sensor data that often contains little information relative to the volume of data being processed. Complex features allow us to inject domain knowledge into the classification process and pull out the key information from the sensor data. Furthermore, as conditional models, CRFs do not waste modeling effort on the observations themselves. This is particularly important in robot domains where the sensor observations are continuous and we cannot easily model the true distribution of the observations.

Conditional random fields are well suited to activity recognition because of their ability to incorporate complex features, but that still leaves the issue of choosing *which* features to include in the final model. We investigated three algorithms for feature selection: grafting, ℓ_1 regularization, and forward selection using a mean field heuristic. We applied these feature selection algorithms to a variety of real and synthetic data to illustrate the properties of the three algorithms.

Our focus was on computational efficiency. While inference in a CRF, equipped with a suitably small set of relevant features, chosen by feature selection, is extremely efficient, the offline process of feature selection and parameter estimation is extremely computationally intensive. We investigated the use of a novel M-estimator for fast, approximate parameter estimation for CRFs.

In our feature selection work, we found that there is a trade off between the sparsity of the final model and the accuracy of the final model. Greedy algorithms, such as grafting produced very sparse, but less accurate, models. Less greedy algorithms, like ℓ_1 regularization, produced larger but more accurate models. Forward selection algorithms, like McCallum's mean field scoring heuristic, offer a middle ground. But that particular heuristic is too slow for large sets of candidate features. We investigated how to apply the M-estimator for feature selection to yield an algorithm potentially less greedy than grafting, because it optimized over candidate weights rather than simply using the gradient to score them, but with approximately the same computational requirements as grafting, because the computationally expensive part of the M-estimator is evaluating the feature expectations under the base model, i.e. a CRF with $n-1$ features in the case of feature selection.

7.1 Contributions

The main contributions of this thesis are:

- We provided a tutorial introduction to conditional random fields for activity recognition. We used synthetic and real data to highlight the properties that make CRFs well suited to activity recognition, namely discriminative training and the ability to incorporate complex, non-independent features of the observations.

- We performed an empirical comparison of CRFs and HMMs to highlight the differences between generative and discriminative models using a robot tag activity recognition domain.

- We introduced the use of conditional random fields for activity recognition in robot domains, such as RoboCup, and explored the types of features that are required for high classification accuracy. We found that spatial features performed well in our models and we presented a relational feature specification language as a succinct and intuitive means of generating large sets of candidate features in a spatial domain with continuous observations.

- We empirically compared three existing feature selection algorithms using a variety of real and synthetic data. We showed that greedy approaches that use a fast heuristic to evaluate candidate features, such as grafting require less computation during feature selection and produce smaller models than more robust approaches like ℓ_1 regularization, at the price of decreased accuracy.

- We investigated the scalability of feature selection to conditional random fields with extremely large sets of candidate features. ℓ_1 regularization, by virtue of the fact that we can use the regularization parameter θ as a threshold to inactive features and perform coordinate-wise optimization scales to extremely large candidate sets. Grafting, which relies on a fast heuristic (the gradient) to choose features scales to relatively large numbers of features, but in situations where the final model legitimately needs many features, the number of rounds of grafting required (linear in the size of the final model) becomes prohibitive. More expensive heuristics, such as McCallum's mean field scoring metric have trouble scaling to large candidate sets because of the time required to score all of the candidate features.

- We explored using an M-estimator, due to Jeon and Lin, for fast, approximate training in CRFs. We found that the M-estimator performs well with relatively small sets of binary features. For small feature sets, the M-estimator yielded comparable accuracy to a CRF while requiring a small fraction of the training time. The M-estimator did not scale as well to larger sets of more complex features.

- We adapted the M-estimator for feature selection in conditional random fields. The most expensive part of training the M-estimator is computing expected values for the feature vector under the base model. If we use a CRF as our base model, computing the feature expectations requires approximately the same computational effort as computing the gradient. We used optimization in the M-estimator to identify the most relevant features in a slightly-less-greedy fashion than grafting, which uses the gradient directly, with approximately the same computational requirements.

7.2 Future Directions

There are a number of promising directions for future work:

- In our experiments with real data from RoboCup robot soccer games, we showed that we can build accurate models for activity recognition using a readily acquired volume of training data. Running feature selection on these models yields light weight classifiers that we can easily run at the full 60 hz frame rate required for robot soccer. A clear next step is to use conditional random fields for online classification of robot behaviors and to use the classifier output to affect the course of the match. In short, we need to close the loop and use the output of activity recognition to make online decisions that affect the world (and future sensor readings). Work remains to create classifiers that predict labels across different teams, but integrating the classifier into a RoboCup team and using it for online classification is a practical next goal.

- In our work to date, we have selected features from a large, but fixed, set of candidate features. We would like to actively induce new features during the course of feature selection, e.g. by creating combinations of previously selected features. Our relational feature specification language lends itself to this type of inductive feature generation because we can build up complex trees of selection and evaluation operators to encode complex relationships between the objects in our domains.

- Greedy forward selection algorithms performed well using the handful of scoring functions that we evaluated. Further work can be done to evaluate a wider collection of scoring metrics drawn from the feature selection literature. A straight forward experiment that we could try immediately would be to use the pruning method of [118], discussed in related work, to reduce the number of candidate features that we evaluate each iteration during forward selection.

- Currently, the M-estimator does not scale to large, complex feature sets. Further characterization could reveal why it does not scale well and help remedy the problem, which would allow us to use it for fast, accurate parameter estimation in CRFs. Further work with the M-estimator for feature selection can clarify the differences between using the M-estimator, more greedy approaches such as grafting, and other heuristic scoring metrics.

- The idea of using an M-estimator for computational efficiency, rather than statistical robustness, creates new possibilities. For example, it may be effective to apply the proposed estimator in the context of graphical models where computing the normalizer is intractable, e.g., in lattices. A variational approximation might be used as q_0 in such a scenario and the resulting M-estimator would estimate the difference between that approximation and the true density of interest.

- Another opportunity for improved classification with the M-estimator comes from the inclusion of a generative base model. Incorporating a generative q_0 allows for semi-supervised learning. The base model can be trained on a large, partially labeled data set. Such a base model may allow the M-estimator to outperform the same M-estimator, or even a CRF, trained on only the labeled portion of the data set. Such a scenario allows models with rich, overlapping features to be trained on partially labeled data without repeated dynamic programming calls.

7.3 Concluding Remarks

This thesis empirically demonstrates that rich, complex models from machine learning, such as conditional random fields, are practical in real world robot domains. While we have not yet shown how much benefit online activity recognition can provide in domains like robot soccer, we feel that machine learning and classification have much to offer for improving the decisions that robots make. We focused on making activity recognition and feature selection methods work on real world data. While it is possible to become bogged down in implementation details, we would emphasize that implementation is important. By the third or fourth major re-write of your CRF implementation, you have a deep understanding of the structure of the computation. That deep understanding is useful not only for making existing algorithms run faster (and it is certainly useful for optimization), it also provides the insight needed to extend existing algorithms and move in new directions, e.g. using the M-estimator for feature selection. Machine learning has a great deal to offer robotics in terms of advanced algorithms for processing sensor data and determining the state of the world. And the robotics perspective of building complete systems that work in the real world can enrich machine learning as well.

Bibliography

[1] Srinivas M. Aji and Robert J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343, March 2000.

[2] C.R. Anderson, P. Domingos, and D.S. Weld. Relational Markov models and their application to adaptive web navigation. *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 143–152, 2002.

[3] Galen Andrew and Jianfeng Gao. Scalable training of L1-regularized log-linear models. *Proceedings of the 24th International Conference on Machine learning*, pages 33–40, 2007.

[4] Arthur Asuncion and David J. Newman. UCI machine learning repository, 2007.

[5] L. Bao and S.S. Intille. Activity Recognition from User-Annotated Acceleration Data. *Pervasive Computing: Second International Conference, Pervasive 2004, Linz/Vienna, Austria, April 18-23, 2004: Proceedings*, 2004.

[6] A.L. Berger, V.J. Della Pietra, and S.A. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, 1996.

[7] Dimitri P. Bertsekas. Projected newton methods for optimization problems with simple constraints. *SIAM Journal on Control and Optimization*, 20(2):221–246, 1982.

[8] J. Besag. Spatial interaction and the statistical analysis of lattice systems (with discussion). *Journal of the Royal Statistical Society*, Series D, 36:192–236, 1974.

[9] J. Besag. Statistical Analysis of Non-Lattice Data. *The Statistician*, 24(3):179–195, 1975.

[10] Avrim Blum and Pat Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(1-2):245–271, 1997.

[11] S.P. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[12] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J Stone. *Classification and regression trees.* Wadsworth, Belmont, CA, 1984.

[13] B. Browning, J. Bruce, M. Bowling, and M. Veloso. STP: skills, tactics, and plays for multi-robot control in adversarial environments. *IEEE Journal of Systems and Control Engineering*, 219(1):33–52, 2005.

[14] James Bruce, Michael Bowling, Brett Browning, and Manuela Veloso. Multi-robot team response to a multi-robot opponent team. In *Proceedings of ICRA'03, the 2003 IEEE International Conference on Robotics and Automation*, Taiwan, May 2003.

[15] James Bruce and Manuela Veloso. Fast and accurate vision-based pattern detection and identification. In *Proceedings of ICRA'03, the 2003 IEEE International Conference on Robotics and Automation*, Taiwan, May 2003.

[16] James Robert Bruce, Manuela Veloso, and Stefan Zickler. CMDragons: Dynamic Passing and Strategy on a Champion Robot Soccer Team. In *Proceedings of ICRA'2008*, Pasadena, CA, 2008.

[17] Hung H. Bui, Svetha Venkatesh, and Geoff West. Policy recognition in the abstract hidden markov model. *Journal of Artificial Intelligence Research*, 17:451–499, 2002.

[18] R. Caruana and D. Freitag. How useful is relevance, 1994.

[19] Michael Collins. Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms. In *EMNLP '02: Proceedings of the ACL-02 conference on Empirical methods in natural language processing*, pages 1–8, Morristown, NJ, USA, 2002. Association for Computational Linguistics.

[20] Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1-2):41–85, 1999.

[21] S. Della Pietra, V. Della Pietra, and J. Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393, 1997.

[22] Keith Diefendorff, Pradeep K. Dubey, Ron Hochsprung, and Hunter Scales. Altivec extension to powerpc accelerates media processing. *IEEE Micro*, 20(2):85–95, 2000.

[23] Thomas G. Dietterich. Machine learning for sequential data: A review. In *Lecture Notes in Computer Science.* Springer Verlag, 2002.

[24] A. Doucet, N. de Freitas, K. Murphy, and S. Russell. Rao-Blackwellised particle filtering for dynamic Bayesian networks. *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 176–183, 2000.

[25] Richard O Duda, P.E. Hart, and David G. Stork. *Pattern Classification*. Wiley, New York, 2nd edition edition, 2001.

[26] Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least angle regression. *Ann. Statist.*, 32(2):407–499, 2004.

[27] Shai Fine, Yoram Singer, and Naftali Tishby. The hierarchical hidden markov model: Analysis and applications. *Machine Learning*, 31(1):41–62, 1998.

[28] Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. *Machine Learning: Proceedings of the Thirteenth International Conference*, 148:156, 1996.

[29] C.J. Geyer and E.A. Thompson. Constrained Monte Carlo maximum likelihood for dependent data. *Journal of the Royal Statistical Society. Series B. Methodological*, 54(3):657–699, 1992.

[30] Z. Ghahramani and M.I. Jordan. Factorial hidden Markov models. *Machine Learning*, 29(2):245–273, 1997.

[31] W.R. Gilks, S. Richardson, and DJ Spiegelhalter. *Markov Chain Monte Carlo in Practice*. Chapman & Hall/CRC, 1996.

[32] Dani Goldberg and Maja J. Mataric. Augmented markov models. Technical Report IRIS-99-367, University of Southern California, 1999.

[33] Dani Goldberg and Maja J. Mataric. Coordinating mobile robot group behavior using a model of interaction dynamics. In *The Third International Conference on Autonomous Agents ('Agents '99)*, 1999.

[34] Dani Goldberg and Maja J. Mataric. Maximizing reward in a non-stationary mobile robot environment. *Autonomous Agents (FIXME - not sure)*, 2000.

[35] Samuel S. Gross, Olga Russakovsky, Chuong B. Do, and Serafim Batzoglou. Training conditional random fields for maximum labelwise accuracy. In *Advances in Neural Information Processing Systems 19 (NIPS 2006)*, 2006.

[36] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3:1157–1182, 2003.

[37] J. Hammersley and P. Clifford. Markov fields on finite graphs and lattices, 1971.

[38] K. Han and M. Veloso. Automated robot behavior recognition applied to robotic soccer. In *Proceedings of IJCAI-99 Workshop on Team Behaviors and Plan Recognition*, 1999.

[39] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, August 2001.

[40] P.J. Hubert. *Robust Statistics*. Wiley, 1981.

[41] Tâm Huynh and Bernt Schiele. Analyzing features for activity recognition. In *sOc-EUSAI '05: Proceedings of the 2005 joint conference on Smart objects and ambient intelligence*, pages 159–163, 2005.

[42] Yongho Jeon and Yi Lin. An effective method for high-dimensional log-density ANOVA estimation, with application to nonparametric graphical model building. *Statistica Sinica*, 16(2):353–374, 2006.

[43] G.H. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. *Proceedings of the Eleventh International Conference on Machine Learning*, 129, 1994.

[44] M.I. Jordan, Z. Ghahramani, T.S. Jaakkola, and L.K. Saul. An introduction to variational methods for graphical models. *Learning in Graphical Models*, 1998.

[45] Junichi Kazama and Junichi Tsujii. Evaluation and extension of maximum entropy models with inequality constraints. *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, pages 137–144, 2003.

[46] Y. Kim and J. Kim. Gradient LASSO for feature selection. *ICML 2004*, 2004.

[47] K. Kira and L.A. Rendell. A practical approach to feature selection. *Proceedings of the ninth international workshop on Machine learning*, pages 249–256, 1992.

[48] K. Kira and L.A. Rendell. The feature selection problem: Traditional methods and a new algorithm. *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 129–134, 1992.

[49] Hiroaki Kitano, editor. *RoboCup-97: Robot Soccer World Cup I*. Springer-Verlag, London, UK, 1998.

[50] K. Koh, S.J. Kim, and S. Boyd. An interior-point method for large-scale l1-regularized logistic regression. *Under Submission*, 2006.

[51] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[52] Igor Kononenko. Estimating attributes: Analysis and extensions of RELIEF. In *European Conference on Machine Learning*, pages 171–182, 1994.

[53] F.R. Kschischang, B.J. Frey, and H.A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001.

[54] S. Kumar and M. Hebert. Discriminative random fields: a discriminative framework for contextual interaction in classification. *Ninth IEEE International Conference on Computer Vision*, pages 1150–1157, 2003.

[55] J. Lafferty, X. Zhu, and Y. Liu. Kernel conditional random fields: representation and clique selection. *ACM International Conference Proceeding Series*, 2004.

[56] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*, pages 282–289. Morgan Kaufmann, San Francisco, CA, 2001.

[57] S.L. Lauritzen. *Graphical Models.* Clarendon Pr, 1996.

[58] Su-In Lee, Honglak Lee, Pieter Abbeel, and Andrew Y. Ng. Efficient l1 regularized logistic regression. In *Proceedings of AAAI 2006*, 2006.

[59] EL Lehmann and G. Casella. *Theory of Point Estimation.* Springer, 1998.

[60] Scott Lenser and Manuela Veloso. Classification of robotic sensor streams using non-parametric statistics. In *Intelligent Robots and Systems (IROS) 2004*, volume 3, pages 2719–2724, 2004.

[61] Scott Lenser and Manuela Veloso. Non-parametric time series classification. In *Proceedings of ICRA 2005*, 2005.

[62] Scott R. Lenser. *On-line Robot Adaptation to Environmental Change.* PhD thesis, Carnegie Mellon University, 2005.

[63] J. Lester, T. Choudhury, N. Kern, G. Borriello, and B. Hannaford. A hybrid discriminative/generative approach for modeling human activities. *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI 2005)*, 2005.

[64] L. Liao, D. Fox, and H. Kautz. Location-based activity recognition using relational Markov networks. *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2005.

[65] L. Liao, D. Fox, and H. Kautz. Extracting Places and Activities from GPS Traces Using Hierarchical Conditional Random Fields. *The International Journal of Robotics Research*, 26(1):119, 2007.

[66] L. Liao, D.J. Patterson, D. Fox, and H. Kautz. Learning and inferring transportation routines. *Artificial Intelligence*, 171(5-6):311–331, 2007.

[67] Lin Liao, Tanzeem Choudhury, Dieter Fox, and Henry Kautz. Training conditional random fields using virtual evidence boosting. In *IJCAI*, pages 2530–2535, 2007.

[68] Lin Liao, Dieter Fox, and Henry Kautz. Learning and inferring transportation routines. In *Proceedings of AAAI 2004*, 2004.

[69] Lin Liao, Dieter Fox, and Henry Kautz. Hierarchical Conditional Random Fields for GPS-based activity recognition. In *Proc. of the International Symposium of Robotis Research (ISRR 2005)*. Springer Verlag, 2005.

[70] D. C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Math. Program.*, 45(3):503–528, 1989.

[71] Maryam Mahdaviani and Tanzeem Choudhury. Fast and scalable training of semi-supervised crfs with application to activity recognition. In *Advances in Neural Information Processing Systems 20*, 2007.

[72] T. Marill and D. Green. On the effectiveness of receptors in recognition systems. *Information Theory, IEEE Transactions on*, 9(1):11–17, 1963.

[73] A. McCallum. Efficiently inducing features of conditional random fields. *Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI03)*, 2003.

[74] A. McCallum, D. Freitag, and F. Pereira. Maximum entropy Markov models for information extraction and segmentation. *Proc. 17th International Conf. on Machine Learning*, pages 591–598, 2000.

[75] K. Murphy, Y. Weiss, and M.I. Jordan. Loopy belief propagation for approximate inference: An empirical study. *Proceedings of Uncertainty in AI*, pages 467–475, 1999.

[76] Kevin Murphy. *Dynamic Bayesian Networks: Representation, Inference, and Learning.* PhD thesis, University of California, Berkeley, 2002.

[77] J.A. Nelder and P.P. McCullagh. *Generalized Linear Models.* CRC Press, 1989.

[78] Andrew Ng and Michael Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in Neural Information Processing Systems (NIPS) 14*, 2002.

[79] Andrew Y. Ng. Feature selection, l1 vs. l2 regularization, and rotational invariance. In *International Conference on Machine Learning (ICML)*, 2004.

[80] Nam T. Nguyen, Dinh Q. Phung, Svetha Venkatesh, and Hung Bui. Learning and detecting activities from movement trajectories using the hierarchicial hidden markov model. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR-2005)*, 2005.

[81] Chris Pal, Charles Sutton, and Andrew McCallum. Sparse forward-backward using minimum divergence beams for fast training of conditional random fields. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2006.

[82] Mee-Young Park and Trevor Hastie. An L1 regularization path algorithm for generalized linear models. *JRSSB*, To Appear.

[83] Donald J. Patterson, Lin Liao, Dieter Fox, and Henry Kautz. Inferring high-level behavior from low-level sensors. In *Proceedings of the International Conference on Ubiquitous Computing 2003 (UBICOMP-03)*, 2003.

[84] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.

[85] S. Perkins, K. Lacker, and J. Theiler. Grafting: fast, incremental feature selection by gradient descent in function space. *The Journal of Machine Learning Research*, 3:1333–1356, 2003.

[86] S. Perkins and J. Theiler. Online feature selection using grafting. *International Conference on Machine Learning*, 2003.

[87] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1992.

[88] A. Quattoni, M. Collins, and T. Darrell. Conditional random fields for object recognition. *Advances in Neural Information Processing Systems*, 17:2004, 2004.

[89] JR Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[90] J.R. Quinlan. *C4.5 Programs for machine learning*. Morgan Kaufmann, San Francisco, CA, 1993.

[91] L.R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, February 1989.

[92] A. Raj, A. Subramanya, J. Bilmes, and D. Fox. Rao-Blackwellized particle filters for recognizing activities and spatial context from wearable sensors. *Experimental Robotics: The 10th International Symposium, Springer Tracts in Advanced Robotics (STAR). Springer Verlag*, 2006.

[93] SK Raman, V. Pentkovski, and J. Keshava. Implementing streaming SIMD extensions on the Pentium III processor. *Micro, IEEE*, 20(4):47–57, 2000.

[94] Lance A. Ramshaw and Mitchell P Marcus. Text chunking using transformation-based learning. In David Yarovsky and Kenneth Church, editors, *Proceedings of the Third ACL Workshop on Very Large Corpora*, pages 82–94, 1995.

[95] N. Ravi, N. Dandekar, P. Mysore, and M.L. Littman. Activity recognition from accelerometer data. *Proceedings of the Seventeenth Conference on Innovative Applications of Artificial Intelligence (IAAI)*, 2005.

[96] S. Riezler and A. Vasserman. Incremental feature selection and l1 regularization for relaxed maximum-entropy modeling. *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing (EMNLP04)*, 2004.

[97] Fei Sha and Fernando Pereira. Shallow parsing with conditional random fields. In *NAACL '03: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pages 134–141, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[98] Noah A. Smith, Douglas L. Vail, and John D. Lafferty. Computationally Efficient M-Estimation of Log-Linear Structure Models. In *ACL*, 2007.

[99] G. Sukthankar and K. Sycara. Robust recognition of physical team behaviors using spatio-temporal models. *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 638–645, 2006.

[100] C. Sutton, A. McCallum, and K. Rohanimanesh. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. *The Journal of Machine Learning Research*, 8:693–723, 2007.

[101] Charles Sutton and Andrew McCallum. An introduction to conditional random fields for relational learning. In Lise Getoor and Ben Taskar, editors, *Introduction to Statistical Relational Learning*. MIT Press, 2006. To appear.

[102] Charles Sutton and Andrew McCallum. Improved dynamic schedules for belief propagation. In *Uncertainty in Artifical Intelligence (UAI)*, 2007.

[103] Charles Sutton and Andrew McCallum. Piecewise pseudolikelihood for efficient crf training. In *International Conference on Machine Learning (ICML)*, 2007.

[104] Charles Sutton, Chris Pal, and Andrew McCallum. Sparse forward-backward for fast training of conditional random fields. In *NIPS 2005 Workshop on Structured Learning for Text and Speech Processing*, 2005.

[105] B. Taskar, P. Abbeel, and D. Koller. Discriminative probabilistic models for relational data. *Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI02)*, pages 895–902, 2002.

[106] Robert Tibshirani. Regression shrinkage and selection via the lasso. *J. Roy. Statist. Soc. Ser. B*, 58(1):267–288, 1996.

[107] Erik Tjong Kim Sang and Sabine Buchholz. Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of CoNLL-2000*, 2000.

[108] T. Truyen, H. Bui, and S. Venkatesh. Human activity learning and segmentation using partially hidden discriminative models. *Workshop on Human Activity Recognition and Modelling HAREM2005*, pages 87–95, 2005.

[109] Douglas L. Vail, John D. Lafferty, and Manuela M. Veloso. Feature selection in conditional random fields for activity recognition. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2007.

[110] Douglas L. Vail and Manuela M. Veloso. Learning from accelerometer data on a legged robot. *Proceedings of the 5th IFAC/EURON Symposium on Intelligent Autonomous Vehicles*, 2004.

[111] Douglas L. Vail and Manuela M. Veloso. Feature selection for activity recognition in multi-robot domains. In *Proceedings of AAAI 2008*, 2008.

[112] Douglas L. Vail, Manuela M. Veloso, and John D. Lafferty. Conditional random fields for activity recogntion. In *AAMAS*, 2007.

[113] A. W. van der Vaart. *Asymptotic Statistics.* Cambridge University Press, 1998.

[114] S. V. N. Vishwanathan, Nicol N. Schraudolph, Mark W. Schmidt, and Kevin P. Murphy. Accelerated training of conditional random fields with stochastic gradient methods. In *ICML 2006*, pages 969–976, 2006.

[115] Hanna Wallach. Efficient training of conditional random fields. Master's thesis, University of Edinburgh, 2002.

[116] S.B. Wang, A. Quattoni, L.P. Morency, D. Demirdjian, and T. Darrell. Hidden Conditional Random Fields for Gesture Recognition. *IEEE Conference on Computer Vision and Pattern Recognition*, 2, 2006.

[117] H. Yan and MJ Mataric. General spatial features for analysis of multi-robot and human activities from raw position data. *Intelligent Robots and System, 2002. IEEE/RSJ International Conference on*, 3, 2002.

[118] Y. Zhou, F. Weng, L. Wu, and H. Schmidt. A fast algorithm for feature selection in conditional maximum entropy modeling. *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, pages 153–159, 2003.