Kernel Methods and Their Application to Structured Data

A dissertation

submitted by

Gabriel Wachman

In partial fulfillment of the requirements
for the degree of

Doctor of Philosophy

in

Computer Science

TUFTS UNIVERSITY

November 2009

ADVISER: Roni Khardon

For my parents

Kernel Methods and Their Application to Structured Data

Gabriel Wachman

ADVISER: Roni Khardon

Supervised Machine learning is concerned with the study of algorithms that take examples and their corresponding labels, and learn a general classification function that can predict the label of future examples. For example, an algorithm may take as input a set of molecules, each labeled "toxic" or "non-toxic" and try to predict the toxicity of new molecules based on the function learned from the input. In the astronomy domain, one might try to predict the type of a star given a series of measurements of the star's brightness, based on a set of known stars and measurements of their brightness. The thesis investigates three aspects of machine learning algorithms that use linear classification functions that work implicitly in feature spaces by using similarity functions known as kernels. The first aspect is robustness to noise, that is learning when some of the labels in the known examples are not reliable. An extensive experimental evaluation reveals a surprising result, that the Perceptron Algorithm with margin is an excellent algorithm in such contexts, and it is competitive or better than more sophisticated alternatives. The second aspect is producing estimates of the confidence of predictions from such classifiers, especially Support Vector Machines. We explore this topic by proposing new methods and comparing them experimentally to existing approaches to this challenge. Finally we investigate kernels for the two applications mentioned above, time series from astronomy and molecules from biochemistry, where the data are not initially

expressed in Euclidean space. In each case we provide an efficiently computable kernel function that captures a natural similarity between pairs of examples. An experimental evaluation shows that our kernels lead to excellent performance when used with Perceptron variants or Support Vector Machines. The contribution for the astronomy application goes beyond machine learning, providing a complete system for classifying stars from raw data taken in astronomy surveys, a task that typically requires a large amount of domain expert time. In this context the thesis investigates and evaluates several statistical tests and mechanisms for filtering and processing time series data.

Contents

List of Tables

List of Figures

Bibliographic Note

Significant portions of this thesis are from previously published work by the author. Most of the material in Chapter 3 can be found in Khardon and Wachman [2007]. Chapter 4 uses all of the material from Wachman and Khardon [2007] and adds significant new contributions. The material pertaining to kernel methods in Chapter 5 is published in Wachman et al. [2009].

Chapter 1

Introduction

This thesis presents research on supervised machine learning classification algorithms. *Machine learning* refers to the science of designing and analyzing algorithms that learn from experience. *Supervised* machine learning, the domain with which this thesis is concerned, is the study of algorithms that have access to labeled data; that is, the algorithms learn from experience by trying to label the data, and making corrections based on whether they have done so correctly. Surveys of supervised machine learning and the theory behind it can be found in Mitchell [1997], Kearns and Vazirani [1994], and Bishop [2006]. Prior to presenting the contributions of the thesis, we give a brief overview of the problem setting and associated algorithmic machinery.

In the supervised machine learning classification setting, the task is to group data into classes. For example, the data may be pictures of animals and we want to group the pictures of tigers together in one class, and pictures of all other animals into another class. The classification algorithm uses a set of examples, called a *training set*, to build a classifier that takes as input an image, and gives as output the predicted class. The examples in the training set are labeled according to their class so that the classification algorithm can check the class it predicts against the true class.

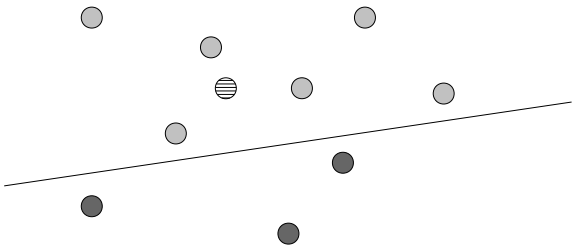

Figure 1.1: Illustration of a 2-dimensional linear classifier

In this thesis we concentrate on a subset of classification algorithms called *linear classifiers*. This is a class of functions that are linear in the instance space. In other words, linear classifiers find a separation boundary in the instance space between the examples from the training set that are in the target concept and those that are not. From here on, instead of referring to a target concept, we will use the term *class*. Unless otherwise noted, we assume that the dataset contains two classes represented by the labels 1 and 0, respectively. This is equivalent to saying that an example is either part of a concept (label 1) or not (label 0). In Figure 1 we show a simple example of a linear classifier where the instance space is 2-dimensional, and hence the separation boundary is a line. When the instance space is n-dimensional, we call the separation boundary a *hyperplane*. In the figure, the dark circles are members of one class, and the lighter circles are members of a second class (i.e., not the first class). The striped circle represents a future un-labeled example that the algorithm must classify.

The method a linear classifier uses to construct the separation boundary is the key differentiation among algorithms. There are many linear classification algorithms. One of the first, the Perceptron Algorithm [Rosenblatt, 1958], starts with some default separation boundary. It then looks at each training example, and assigns it a pre-determined label (i.e., 1) if it falls on one side of the separation boundary, and 0 if it falls on the other side. If it makes a mistake on an example, it moves the boundary towards the example, or past the example such that the mis-classified example is now on the correct side of the boundary. Under certain

conditions, the Perceptron is guaranteed to converge to a hyperplane that separates the two classes of data in the training set. There are many variants of the Perceptron Algorithm [Friess et al., 1998, Gentile, 2001, Li and Long, 2002, Crammer et al., 2005, Kivinen et al., 2004, Shalev-Shwartz and Singer, 2005, Tsampouka and Shawe-Taylor, 2005] that either change the updating criterion, or the update itself, or both. Perceptron variants are generally fast and are still top-performing algorithms.

Support Vector Machine (SVM) [Boser et al., 1992] is a linear classifier that optimizes a quantity called *margin*. The margin of a separation boundary is the distance between the boundary and the closest example in the training set. In Figure 1, one of the light circles is very close to the line. The margin of the boundary would be the distance between the boundary and this circle. Several theoretical results [Bartlett and Shawe-Taylor, 1999, Bartlett, 1998] show that by increasing the margin on a training set, we can decrease the upper bound on the error rate of the classifier.[1] Intuitively, it seems like a good idea to find the separation boundary that separates the two classes as much as possible; this is what SVM does. As with Perceptron, there are many variants of SVM and it has steadily become computationally faster since its adoption by the machine learning community through algorithmic and implementation improvements.

Many other linear classification algorithms exist: Adatron [Friess et al., 1998], Winnow [Littlestone, 1987], Naive Bayes, ALMA [Gentile, 2001], and ROMMA [Li and Long, 2002], for example. ALMA and ROMMA attempt to approximate the solution of SVM while using a Perceptron style implementation. Adatron actually computes the optimal separation boundary like SVM, except that the algorithm is now slower than most SVM implementations. Naive Bayes takes a loose probabilistic approach and computes the label of each example by computing the probability of its features based on how often those features occurred in the training set.

[1] For a summary of existing theoretical results see Cristianini and Shawe-Taylor [2000], Smola et al. [1999], and Shawe-Taylor et al. [1998].

Thus far in our discussion of linear classifiers we have only discussed classifying training examples in the original instance space. It may be that no linear separation boundary exists in this space, and hence the classification algorithm will perform poorly. We can try to change the classification algorithm to explore non-linear separation boundaries (i.e., quadratic, logarithmic, etc.) but this is computationally challenging. The other possibility is to embed the examples in a higher-dimensional space and run the linear classifier in the new space. For example, if the best separation boundary is a quadratic function in the original input space, we can embed the training examples in a new space such that a quadratic function in the original space is now a linear function. With this embedding no changes to the linear classifier are necessary. By using kernel methods, we can accomplish this exactly, but without incurring the cost of constructing the explicit feature space embedding. Kernel methods allow linear classifiers to work in much higher-dimensional (or even infinite-dimensional) spaces. Not all linear classifiers can take advantage of kernel methods, but SVM and Perceptron can and these are the algorithms we use primarily throughout this thesis.

It is often the case that the data are not linearly separable due to mis-labeled or noisy examples, that is, there is no hyperplane that separates one class of examples from the other. The problem of learning in a noisy setting has been a primary driving force behind the development of classification algorithms. When data are not noisy, optimal classification is possible, at least in principle. This is not the case in most settings, and data noise is an ever-present challenge.

For the first main contribution of the thesis, in Chapter 3, we examine variants of the Perceptron Algorithm in the setting where the data are not linearly separable. We make a thorough comparison among the algorithms, explore the reasons why certain algorithms fail or succeed, and present interesting questions arising from the experimental results. Below we give a brief summary of our experimental setup and main result.

Perceptron with Margins (PAM) is a Perceptron variant that uses a conservative update rule such that the algorithm adjusts not only when a training example is misclassified, but when it is *almost* misclassified. The notion of *almost* is captured formally in the algorithm and is explained in detail in Chapter 3. PAM is actually not designed to handle noise, but we show in Chapter 3 that surprisingly it out-performs all other methods. Another variant imposes an artificial separation between examples, preventing the Perceptron from making repeated mistakes on the same example, so as to avoid being dominated by noisy examples. A third variant, *longest survivor*, uses the separation boundary that gave the most correct consecutive responses during training (recall that each time the Perceptron makes a mistake, it updates the separation boundary). Finally the *voted perceptron* keeps track of all the separation boundaries computed during training and combines them to make a final separation boundary. In addition to comparing all variants to one another, we compare them to Support Vector Machines (SVM) [Boser et al., 1992]. SVM is a good point of comparison as it has become the standard for linear classifiers. Our conclusions are that PAM performs comparably to SVM and that it is a good choice due to its simplicity and efficiency. We also introduce a concrete parameter setting for PAM that leads to successful performance across different applications.

Our second main contribution is in Chapter 4, where we develop the first known kernels for hypergraphs and explore and compare them to other similar kernels in an experimental setting. The goal of this work is to address data that are described most naturally using a graph or hypergraph structure. A graph is a set of nodes, V, and edges, E, such that each edge (u, v) connects two nodes from V. Nodes and edges can have labels. A hypergraph is a graph in which edges have more than two nodes. A walk through the graph or hypergraph is a sequence of nodes and edges, and the walk is described by the sequence of labels of the nodes and edges. Our kernel computes the number of walks of length n that are shared by two graphs; to do this directly is exponential in n, however by using a dynamic programming

approach we can compute this in polynomial time (in n, $|V|$, and $|E|$).

A molecule is traditionally represented by a graph, and hence data naturally represented as graphs are found throughout the chemistry and biology domain. In the chemical setting, the labels on the nodes might be "carbon," "hydrogen," etc. and the labels on the edges would be "single-bond" or "double-bond." The classification of chemical data alone is enough to justify the investigation of learning from graphs, and is in fact the driving force behind much of the research on learning from structured objects. Capturing useful information about the graph is an important challenge, as much of what we may like to learn is intractable to compute [Gärtner et al., 2003]. Recent work on graph kernels [Gärtner et al., 2003, Kashima et al., 2003] makes use of labeled walks through two graphs as a basis for determining the similarity between the graphs. That means that if two graphs share many walks that have the same label sequence, they are considered similar.

Learning from graphs is a special case of learning from hypergraphs, a problem studied in Inductive Logic Programming (ILP) under the name of *learning from interpretations* [De Raedt and Dzeroski, 1994]. Ordered hypergraphs capture complex relational structures and are a powerful representation tool. Our kernel is the first kernel that is able to work directly on the hypergraph structure. We demonstrate that it performs comparably or better than state-of-the-art ILP systems [e.g., Muggleton, 1995, Quinlan, 1990] on some traditional ILP benchmark datasets. We also show that the ability to interpret hypergraphs directly improves performance in some cases. When learning from graphs, we show that our kernel maps the data into a significantly different feature space than previous graph kernels and gives state-of-the-art performance.

Another type of structured data is a time series in which a data point is a set of values, each with an associated time. In astronomy, a typical example is brightness measurements taken from a star: here each brightness measurement has an associated date indicating when the measurement was taken. The structure

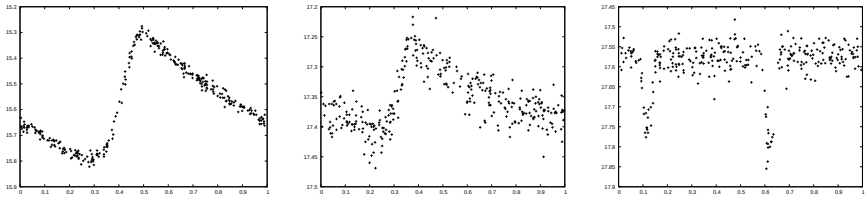

Figure 1.2: Typical astronomy time series for periodic stars.

inherent in a time series from the astronomy domain can be seen in Figure 1.2. Previous work on classifying time series extracts features based on the Fourier or Hermite basis representation [e.g., Vlachos et al., 2005, Osowski et al., 2004]. Other approaches use probabilistic models like Hidden Markov Models to classify time series [Ge and Smyth, 2000]. Finally, a similarity measure can be constructed that is informative for time series, such as Dynamic Time Warping [Berndt and Clifford, 1994, Lu et al., 2008].

In Chapter 5 we present our third contribution: a completely automatic system for classifying a certain type of astronomy data. The centerpiece of this contribution is a new kernel for classifying time series. We present theoretical insight into a successful non-kernel similarity measure for time series, and develop our method as an approximation. We show that our kernel performs as well or better than existing methods for periodic time series, in both the astronomy and shape-matching domains. We then use our kernel as the basis for an automatic system for classifying time series taken from stars. This work is among the first methods that can classify an astronomy survey completely automatically. We examine the various data mining challenges involved in processing a large, unfiltered dataset, presenting new algorithms, and we demonstrate our new system by classifying an entire astronomy survey.

In our final contribution, we analyze the problem of generating measurements of confidence of classifier output. As stated above, in the classification setting the primary problem is to build a classifier that finds labels for unknown data points. In many scenarios, however, we wish to know how confident the classifier is in its

predicted label. Most classification algorithms are designed only to minimize expected classification error, and hence do not give such confidence estimates. An open problem is whether it is possible to construct reliable confidences from the output of a classifier, or even to be able to reliably rank the output of a classifier in terms of confidence. A number of methods exist for estimating confidences from classifier output. These methods typically take the output of a classifier and run an optimization problem over the output that produces an estimate of the probability that the label produced by the classifier for a particular example is correct. In Chapter 6 we propose new solutions to this problem based on observed deficiencies in existing methods. We show that some of our methods perform well in an experimental setting, sometimes out-performing previous work. We analyze as well the cases where our methods do not perform well. We conclude that two of our methods show promise, and should be developed further.

To summarize, the thesis gives a thorough empirical analysis of variants of the Perceptron Algorithm designed for noise-tolerance, drawing the unintuitive conclusion that PAM is the best variant, comparable to SVM. We present the first kernel methods for learning from hypergraphs and demonstrate their effectiveness in an experimental setting. We give a new method for learning from periodic time series and build the first system that automatically classifies certain astronomical objects. Finally, we evaluate existing methods for estimating the confidence of a classifier in a predicted label, and suggest new methods, showing their performance, and suggesting new avenues for improvement.

Chapter 2

Background and Preliminaries

In this chapter we explain in detail several concepts related to the general problem of classification with kernel methods. The theory and concepts in this chapter are not original results of this thesis. For a thorough review of the material below, see Cristianini and Shawe-Taylor [2000], Bishop [2006], and Schölkopf and Smola [2002].

2.1 Kernel Functions

Definition 2.1.1. A *kernel*[1] is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that for all $m \in \mathbb{N}$ and all $x_1 \dots x_m \in \mathcal{X}$, k generates a positive semidefinite *Gram Matrix* G, where $G_{ij} = k(x_i, x_j)$.

Theorem 2.1.2 (Mercer[Cristianini and Shawe-Taylor, 2000])**.** *Let X be a compact subset of $\mathbb{R}^n$. Suppose K is a continuous symmetric function such that the integral operator $T_k : L_2(X) \to L_2(X)$,*

$$(T_K f)(\cdot) = \int_X K(\cdot, x) f(x) dx,$$

is positive, that is

$$\int_{X \times X} K(x, z) f(x) f(z) dx dz \geq 0,$$

[1] we use the term "kernel" to refer to a positive semidefinite kernel.

for all $f \in L_2(X)$. Then we can expand $K(x,z)$ in a uniformly convergent series (on $X \times X$) in terms of T_K's eigen-functions $\phi_j \in L_2(X)$, normalized in such a way that $\|\phi_j\|_{L_2} = 1$, and positive associated eigenvalues $\lambda_j \geq 0$,

$$K(x,z) = \sum_{j=1}^{\infty} \lambda_j \phi_j(x) \phi_j(z).$$

By Mercer's Theorem, stated formally above, a kernel is equivalent to taking the inner product in some feature space Φ of two objects from a domain $\mathcal{X}$. In our notation we use Φ to represent the feature space as well as the function that maps an object into the feature space. The explicit representation of an object $x \in \mathcal{X}$ is denoted $\Phi(x)$.

We next examine the practical consequences of the fact that a kernel is equivalent to taking an inner product. The first consequence is that we can take an inner product of two objects $x, y \in \mathcal{X}$ in the feature space Φ without explicitly representing $\Phi(x)$ or $\Phi(y)$; we need only compute the value $K(x,y) = \langle \Phi(x), \Phi(y) \rangle$. For example, consider the Boolean Kernel [Khardon et al., 2001]

$$K(x,y) = 2^{\langle x,y \rangle}$$

where $x, y \in \{0,1\}^n$ (i.e., x and y are binary feature vectors). It is easy to show that this kernel computes the inner product of x and y in a feature space indexed by all possible conjunctions. To represent this feature space explicitly, and to take an inner product in this feature space explicitly, would require $O(3^n)$ space and time. To compute K, on the other hand, takes $O(n)$ space and time.

The second consequence of the equivalence of kernels and inner products is that we can replace the inner product operator in an algorithm with a kernel function. This means that an algorithm that relies only on taking inner products can now work in a feature space without having to represent the data explicitly in that feature space. In other words, kernels enable an algorithm that depends only on

inner products to work in a feature space that would be intractable to represent explicitly. We will give specific examples as we discuss individual kernel methods.

Previous work [e.g., Cumby and Roth, 2003, Khardon et al., 2001] show that in certain cases, explicit representation of a feature space is sometimes preferable to using a kernel function, even if the feature space is intractable to represent in general. This stems from the trade-off between the "kernelization" of the classification algorithms (illustrated with examples in section 2.2), which incurs a significant performance penalty, and the explicit representation of a large feature space, which can be intractable. If the feature space is exponentially-sized with respect to the original data encoding but very sparse, for example, then it still may be possible to represent the data efficiently using a sparse-vector representation. We do not address optimizing run times in our experiments and hence we opt to use kernels throughout our work as they are tractable in all cases. Nevertheless, the analysis of whether kernels are the most efficient way to compute inner products in our experiments is a worthwhile endeavor for future work.

2.2 Kernel Methods

Definition 2.2.1. A *classifier* or *classification algorithm* is an algorithm that maps a objects from a domain $\mathcal{X}$ to a set of *labels* or *classes* $\mathcal{Y} \subset \mathbb{Z}$.

Definition 2.2.2. A kernel method is any classification algorithm that relies only on inner products of examples to make a prediction.

Examples of kernel methods are the Perceptron Algorithm [Rosenblatt, 1958], Support Vector Machine (SVM) [Boser et al., 1992], k-means, k-Nearest Neighbors, logistic regression, and Fisher Discriminants. For more information on these algorithms see Bishop [2006]. In the following we will describe in detail two classifiers, Perceptron and SVM, and illustrate how they can be used with kernels.

Input set of examples and their labels $\mathcal{Z} = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathbb{R}^n \times \{-1, 1\})^m$, η, θ_{Init}

- Initialize $\mathbf{w} \leftarrow \mathbf{0}$ and $\theta \leftarrow \theta_{Init}$
- for every training epoch:
- for every $x_j \in \mathcal{X}$:
  - $\hat{y} \leftarrow sign(\langle \mathbf{w}, x_j \rangle - \theta)$
  - if $(\hat{y} \neq y_j)$
    - $\mathbf{w} \leftarrow \mathbf{w} + \eta y_j x_j$
    - $\theta \leftarrow \theta + \eta y_j \theta_{Init}$

Figure 2.1: The basic Perceptron Algorithm

2.3 The Perceptron Algorithm

The Perceptron Algorithm [Rosenblatt, 1958] is an example of a *linear classifier*. That is, using data represented as feature vectors in $\mathbb{R}^n$, it constructs a *hyperplane* that separates one class of data from another. Formally, the Perceptron takes as input a set of training examples in $\mathbb{R}^n$ labeled $\{-1, 1\}$. Using a weight vector, $\mathbf{w} \in \mathbb{R}^n$, initialized to $\mathbf{0}$, and a threshold (or bias), θ, it predicts the label of each training example $\mathbf{x}$ to be $y = sign(\langle \mathbf{w}, \mathbf{x} \rangle - \theta)$. The algorithm adjusts $\mathbf{w}$ and θ on each misclassified example by an additive factor. The algorithm is summarized in Figure 2.1. The "learning rate" η controls the extent to which $\mathbf{w}$ can change on a single update. The initial choice of θ is important as it can be significant in early iterations of the algorithm.

The version of the algorithm as shown in Figure 2.1 is known as the *primal* form of the algorithm. Note that w is a linear combination of the input vectors, and is stored explicitly. Hence, in this setting it may be the case that we could not work in a feature space that is very large, as we would have to represent each example in the feature space. It is well known that the Perceptron can be re-written in "dual form" whereby it can be used with kernel functions [Cristianini and Shawe-Taylor, 2000]. We now show how this is done and how this allows us to work in high- or

Input as in primal form.

- Initialize $\alpha \leftarrow 0^m, k \leftarrow 0, \mathbf{w_0} \leftarrow \mathbf{0}$ and $\theta \leftarrow \theta_{Init}$
- for every training epoch:
- for every $x_j \in \mathcal{X}$:
  - $SUM \leftarrow \sum_{i|\alpha_i \neq 0} \eta \alpha_i y_i (\langle \Phi(x_i), \Phi(x_j) \rangle)$
  - $\hat{y} \leftarrow sign(SUM - \theta)$
  - if $(\hat{y} \neq y_j)$
    - $\alpha_j \leftarrow \alpha_j + 1$
    - $\theta \leftarrow \theta + \eta C y_j$

Figure 2.2: The Perceptron Algorithm in dual form

infinite-dimensional feature spaces.

In Figure 2.2, we show the dual form of the Perceptron Algorithm. This is exactly equivalent to the algorithm shown in Figure 2.1, and is based on the fact that $\mathbf{w}$ is a linear combination of the input and can be written as

$$\mathbf{w} = \sum_{i=1}^{m} \eta \alpha_i y_i x_i$$

where α_i is the number of mistakes that have been made on example i. Now re-writing $\langle \mathbf{w}, z \rangle$ with the alternate form of $\mathbf{w}$ we have

$$\langle \mathbf{w}, z \rangle = \sum_{i=1}^{m} \eta \alpha_i y_i \langle x_i, z \rangle \tag{2.1}$$

for any z.

The dual form of the algorithm relies only on taking inner products between examples. Because we do not store $\mathbf{w}$ explicitly, but only the coefficients α_i for each example, the algorithm is no longer dependent on the size of the feature space in which the inner products are taken. Using the results from Section 2.1 we replace

inner products with kernel functions, and so we can replace the LHS of (2.1) with

$$\sum_{i=1}^{m} \eta \alpha_i y_i K(x_i, z) \tag{2.2}$$

using any kernel K. To summarize, we have "kernelized" the Perceptron Algorithm by noting that its dual form relies only on inner products, and replacing those inner products with kernels. We can now run the Perceptron in any feature space for which we can efficiently compute the inner product, and we do not have to store the examples in that feature space.

2.4 Support Vector Machines

The Support Vector Machine (SVM), like the Perceptron Algorithm, is a linear classifier. Unlike the Perceptron, however, the SVM constructs a hyperplane that is optimal according to some conditions. Specifically, SVM finds the hyperplane that maximizes the *margin* of a hyperplane with respect to the training data. The margin of $\mathbf{w}$ with respect to $x_i \in \mathcal{X}$ is

$$\frac{y_i \langle \mathbf{w}, x_i \rangle + \theta}{\|\mathbf{w}\|} \tag{2.3}$$

and the margin of $\mathbf{w}$ with respect to the dataset $\{x_1, \ldots, x_m\}$ is

$$\min_i \frac{y_i \langle \mathbf{w}, x_i \rangle + \theta}{\|\mathbf{w}\|}.$$

The margin quantifies the extent to which the two classes in the data are separated by $\mathbf{w}$, and in fact the margin over the dataset represents the minimum distance of any point in the dataset to the separating hyperplane. The simple form of the SVM is normally given by:

$$\begin{aligned} \texttt{Minimize} \quad & \|\mathbf{w}\|^2 \\ \texttt{Subject to} \quad & y_i(\langle \mathbf{w}, x_i \rangle + \theta) \geq 1, i = 1, \ldots, m. \end{aligned} \tag{2.4}$$

Note that the problem is formulated to minimize the norm of $\mathbf{w}$ while bounding the decision value from below. This makes sense by looking at (2.3), where we see that just trying to increase the margin by scaling $\|\mathbf{w}\|^2$ by a factor < 1, will also decrease the numerator. Instead, the SVM problem looks to decrease the denominator in (2.3) while fixing the numerator to be at least 1.

Thus far we have assumed that the data are *linearly separable*, that is, there exists a hyperplane $\mathbf{w}$ such that $y_i(\langle \mathbf{w}, x_i \rangle + \theta) > 0$. In most cases, the data are not linearly separable. The Perceptron can classify such data without any modification, but SVM can not because the constraints in (2.4) can not be satisfied in the linearly inseparable case. To address this problem, we introduce *slack variables*, denoted ξ_i. In (2.4) we required $y_i(\langle \mathbf{w}, x_i \rangle + \theta) \geq 1$; now to account for data sets in which this is not possible for all i, we change the requirement to $y_i(\langle \mathbf{w}, x_i \rangle + \theta) \geq 1 - \xi_i$. In other words, ξ_i quantifies the extent to which x_i violates the requirement $y_i(\langle \mathbf{w}, x_i \rangle + \theta) \geq 1$. The new minimization problem is

$$\begin{aligned} \texttt{Minimize}_{\mathbf{w},\xi} \quad & \|\mathbf{w}\| + \tfrac{C}{2} \textstyle\sum_{i=1}^{m} \xi_i \\ \texttt{Subject to} \quad & y_i(\langle \mathbf{w}, x_i \rangle + \theta) \geq 1 - \xi_i, i = 1, \ldots, m \\ & \xi_i \geq 0, i = 1, \ldots, m \end{aligned} \tag{2.5}$$

Here we minimize the L_1-norm of ξ (the slack vector). An alternate formulation minimized the L_2-norm of ξ:

$$\begin{aligned} \texttt{Minimize}_{\mathbf{w},\xi} \quad & \|\mathbf{w}\| + \tfrac{C}{2} \textstyle\sum_{i=1}^{m} \xi_i^2 \\ \texttt{Subject to} \quad & y_i(\langle \mathbf{w}, x_i \rangle + \theta) \geq 1 - \xi_i, i = 1, \ldots, m \end{aligned} \tag{2.6}$$

Unless otherwise noted, we use the L_1-norm version (2.5) throughout the remainder of the thesis. Observe that in (2.6), $\xi_i \geq 0 \ \forall i$ because if $\xi_i < 0$ then no constraint is

violated by setting $\xi_i = 0$, and this change will decrease the value of the objective function. The parameter C quantifies the balance between the two terms in the objective function. A high value of C means that separation of the data is prioritized, since any change in the value of a ξ_i will have a large effect on the objective function. Likewise, if C is small, then many mis-classifications are allowed, and the priority shifts to the margin on the correctly classified examples.

As with the Perceptron, we can derive a dual form for (soft-margin) by calculating the dual of the optimization problem (2.5) which yields

$$\begin{array}{ll} \texttt{Maximize}_\alpha & \sum_{i=1}^m \alpha_i - \frac{1}{2}\sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle, \\ \texttt{Subject to} & 0 \leq \alpha_i \leq \frac{C}{2}, i = 1, \ldots, m \\ & \sum_{i=1}^m \alpha_i y_i = 0, i = 1, \ldots, m. \end{array} \tag{2.7}$$

In computing the dual form, θ is found to have the closed form:

$$\theta = \frac{1}{m} \sum_{i=1}^m \left(y_i - \sum_{j=1}^m y_j \alpha_j \langle x_i, x_j \rangle \right).$$

As with Perceptron, (2.7) depends only on inner products of examples, and hence we can "kernelize" SVM using the dual by replacing $\langle x_i, x_j \rangle$ with $K(x_i, x_j)$, where K is a kernel. This gives:

$$\begin{array}{ll} \texttt{Maximize}_\alpha & \sum_{i=1}^m \alpha_i - \frac{1}{2}\sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j K(x_i, x_j), \\ \texttt{Subject to} & 0 \leq \alpha_i \leq \frac{C}{2}, i = 1, \ldots, m \\ & \sum_{i=1}^m \alpha_i y_i = 0, i = 1, \ldots, m. \end{array} \tag{2.8}$$

We have reviewed SVM which we will use extensively throughout the thesis. We explained how it can be used with kernels by using the dual form of the algorithm and replacing the inner product operation with any kernel. As a consequence, we can now use SVM to classify data in feature spaces that we could not tractably represent explicitly.

2.5 Binary to Multiclass

The classifiers we have discussed thus far are built for binary classification problems, that is the case when the labels y can have two values. It is possible to extend any binary classifier to the multiclass case where there are $k \geq 2$ classes. A good summary of these methods for multi-class extensions of binary classifiers can be found in Allwein et al. [2000], Huang et al. [2006] and Wu et al. [2004]. The simplest extension is *one-versus-one*, which learns $k(k-1)/2$ binary classifiers, one to separate every pairing of labels. A simple decision rule for the one-versus-one extension is

$$\underset{i}{\operatorname{argmax}} \left[\sum_{j:j \neq i} I_{r_{ij} > r_{ji}} \right] \tag{2.9}$$

where the sum is over class labels, and the r_{ij} are derived from the raw classifier output (or are simply the raw classifier output). This rule simply chooses as a label the class that was chosen the most times by the binary classifiers.

Another classic binary-to-multiclass extension is known as *one-versus-all*. In this setting k classifiers are learned, one to distinguish every class from the other $k-1$ classes. A simple decision rule in this case is to simply use the maximum output of the k classifiers to make a decision in the multi-class cases.

There are many other possible decision rules for the one-versus-one case that are motivated by a probabilistic interpretation of the classifier output. We will discuss these in Chapter 6.

The binary-to-multiclass setting is generalized by Allwein et al. [2000] to any set of binary classifiers, each of which distinguishes any two subsets of the labels. The generalization of the setting in which the classifiers give probabilistic estimates of class membership is given by Huang et al. [2006]. Har-Peled et al. [2002] give a general framework that unifies binary-to-multiclass, ranking, and constraint classification.

Chapter 3

Noise-Tolerant Variants of the Perceptron Algorithm

The success of Support Vector Machines (SVMs) [Boser et al., 1992, Cristianini and Shawe-Taylor, 2000] has led to increasing interest in the Perceptron Algorithm [Rosenblatt, 1958]. Like SVM, the Perceptron Algorithm is a linear classifier and can be used with kernels, but unlike SVM, it is simple and easy to implement. Interestingly, despite a large number of theoretical developments, there is no result that explains why SVM performs better than Perceptron, and similar convergence bounds exist for both [Graepel et al., 2000, Cesa-Bianchi et al., 2004]. In practice, SVM is often observed to perform slightly better with significant cost in run time. Several on-line algorithms have been proposed which iteratively construct large margin hypotheses in the feature space, and therefore combine the advantages of large margin hypotheses with the efficiency of the Perceptron Algorithm. Other variants adapt the on-line algorithms to work in a batch setting choosing a more robust hypothesis to be used instead of the last hypothesis from the on-line session. There is no clear study in the literature, however, that compares the performance of these variants or the possibility of combining them to obtain further performance improvements. We believe that this is important as these algorithms have already

been used in applications with large datasets [e.g., Collins, 2002, Li et al., 2002, Punyakanok et al., 2008] and a better understanding of what works and when can have a direct implication for future use. This chapter provides such an experimental study where we focus on noisy data and more generally the "unrealizable case" where the data is simply not linearly separable. We chose some of the basic Perceptron variants and experimented with them to explore their performance both with hindsight knowledge and in a statistically robust setting.

More concretely, we study two families of variants. The first explicitly uses the idea of hard and soft margin from SVM. The basic Perceptron algorithm is mistake driven, that is, it only updates the hypothesis when it makes a mistake on the current example. The Perceptron Algorithm with margin [Krauth and Mézard, 1987, Li et al., 2002, Grove and Roth, 1997] forces the hypothesis to have some margin by making updates even when it does not make a mistake but where the margin is too small. Adding to this idea, one can mimic soft-margin versions of support vector machines within the Perceptron Algorithm that allow it to tolerate noisy data [e.g., Li et al., 2002, Kowalczyk et al., 2001]. The algorithms that arise from this idea constrain the update function of the Perceptron and limit the effect of any single example on the final hypothesis. A number of other variants in this family exist in the literature. Each of these performs margin based updates and has other small differences motivated by various considerations. We discuss these further in the concluding section of the chapter.

The second family of variants tackles the use of on-line learning algorithms in a batch setting, where one trains the algorithm on a dataset and tests its performance on a separate test set. In this case, because updates do not always improve the error rate of the hypothesis (e.g., in the noisy setting), the final hypothesis from the on-line session may not be the best to use. In particular, the longest survivor variant [Kearns et al., 1987, Gallant, 1990] picks the "best" hypothesis on the sequential training set. The Voted Perceptron variant [Freund and Schapire, 1999] takes a vote among

hypotheses produced during training. Both of these have theoretical guarantees in the PAC learning model [Valiant, 1984]. Again, other variants exist in the literature which modify the notion of "best" or the voting scheme among hypotheses and these are discussed in the concluding section of the chapter.

It is clear that each member of the first family can be combined with each member of the second. In this chapter we report on experiments with a large number of such variants that arise when combining some of margin, soft margin, and on-line to batch conversions. In addition to real world data, we used artificial data to check the performance in idealized situations across a spectrum of data types. Thus the main contribution of this chapter is an empirical study comparing several variants and their novel combinations and the resulting practical conclusions.

The experiments lead to the following conclusions: First, the Perceptron with margin is the most successful variant. This is surprising as among the algorithms experimented with it is only one not designed for noise tolerance. Second, the soft-margin variants on their own are weaker than the Perceptron with margin, and combining soft-margin with the regular margin variant does not provide additional improvements.

The third conclusion is that in most cases the Voted Perceptron performs similarly to the Perceptron with margin. The Voted Perceptron has the advantage that it does not require parameter selection (for the margin) that can be costly in terms of run time. Combining the two to get the Voted Perceptron with margin has the potential for further improvements but this occasionally degrades performance. Finally, both the Voted Perceptron and the margin variant reduce the deviation in accuracy in addition to improving the accuracy. This is an important property that adds to the stability and robustness of the algorithms.

The rest of the chapter is organized as follows. The next section reviews all the algorithms and our basic settings for them. Section 3.2 describes the experimental evaluation. We performed two kinds of experiments. In "parameter search"

we report the best results obtained with any parameter setting. This helps set the scope and evaluate the potential of different algorithms to improve performance and provides insight about their performance, but it does not give statistically reliable results. In "parameter optimization" the algorithms automatically select the parameters and the performance can be interpreted statistically. The concluding section further discusses the results and puts related work in its context leading to directions for future work.

3.1 Algorithms

In this section we describe the algorithms used in our study. All these algorithms have been previously introduced in the literature and citations are given throughout the text. Some minor variants (α-variant below) and combinations of options are new in this empirical study.

Recall from Section 2.3 that the Perceptron constructs a decision boundary parameterized by $(\mathbf{w}, \theta)$. When the data are linearly separable via some hyperplane $(\mathbf{w}, \theta)$, the *margin* is defined as

$$\gamma = \min_{1 \leq i \leq m} (y_i(\langle \mathbf{w}, x_i \rangle - \theta)). \tag{3.1}$$

When $\|\mathbf{w}\| = 1$, γ is the minimum Euclidean distance of any point in the dataset to $(\mathbf{w}, \theta)$. If the data are linearly separable, and θ is initialized to 0, the Perceptron algorithm is guaranteed to converge in $\leq (\frac{R}{\gamma})^2$ iterations [Novikoff, 1962, Cristianini and Shawe-Taylor, 2000], where $R = \max_{1 \leq i \leq m} \|x_i\|$.

In the case of non-separable data, the extent to which a data point fails to have margin γ via the hyperplane w can be quantified by a slack variable $\xi_i = \max(0, \gamma - y_i(\langle \mathbf{w}, \mathbf{x_i} \rangle + \theta))$. Observe that when $\xi_i = 0$, the example x_i has margin at least γ via the hyperplane defined by $(\mathbf{w}, \theta)$. The Perceptron is guaranteed to make no more than $(\frac{2(R+D)}{\gamma})^2$ mistakes on m examples, for any $\mathbf{w}, \gamma > 0$ where

$D = \sqrt{\sum_{i=1}^{m} \xi_i^2}$ [Freund and Schapire, 1999, Shalev-Shwartz and Singer, 2005]. Thus one can expect some robustness to noise.

All the Perceptron variants we discuss below have dual form representations. However, the focus of this chapter is on noise tolerance and this is independent of the use of primal or dual forms and the use of kernels. We therefore present all the algorithms in primal form.

3.1.1 The λ-trick

The λ-trick [Kowalczyk et al., 2001, Li et al., 2002] attempts to minimize the effect of noisy examples during training similar to the L_2 soft margin technique used with support vector machines [Cristianini and Shawe-Taylor, 2000]. We classify example x_j according to $sign(SUM - \theta)$ where

$$SUM = \langle \mathbf{w}, x_j \rangle + I_j y_j \lambda \|x_j\| \tag{3.2}$$

and where I_j is an indicator variable such that

$$I_j = \begin{cases} 1 & \text{if } x_j \text{ was previously classified incorrectly} \\ 0 & \text{otherwise} \end{cases} .$$

Thus during training, if a mistake has been made on x_j then in future iterations we increase SUM artificially by a factor of $\lambda\|x_j\|$ when classifying x_j but not when classifying other examples. A high value of λ can make the term $y_j\lambda\|x_j\|$ dominate the sum and make sure that x_j does not lead to an update, hence it is effectively ignored. In this way λ can help the algorithm avoid a large number of updates on noisy (as well as other) examples limiting their effect.

We observe that this technique is typically presented using an additive λ instead of $\lambda\|x_j\|$. While there is no fundamental difference, adding $\lambda\|x_j\|$ is more convenient in the experiments as it allows us to use the same values of λ for all datasets (because

the added term is scaled relative to the data).

3.1.2 The α-bound

This variant is motivated by the L_1 soft margin technique used with support vector machines [Cristianini and Shawe-Taylor, 2000]. The *α-bound* places a bound α on the number of mistakes the algorithm can make on a particular example, such that when the algorithm makes a mistake on some x_i, it does not update $\mathbf{w}$ if more than α mistakes have already been made on x_i. As in the case of the λ-trick, the idea behind this procedure is to limit the influence of any particular noisy example on the hypothesis. Intuitively, a good setting for α is some fraction of the number of training iterations. To see that, assume that the algorithm has made α mistakes on a particular noisy example x_k. In all subsequent training iterations, x_k never forces an update, whereas the algorithm may continue to increase the influence of other non-noisy examples. If the ratio of non-noisy examples to noisy examples is high enough the algorithm should be able to bound the effect of noisy examples in the early training iterations while leaving sufficient un-bounded non-noisy examples to form a good hypothesis in subsequent iterations. While this is a natural variant, we are not aware of any experimental results using it. Observe that in order for the α-bound to work effectively, the algorithm must perform a high enough number of training iterations.

3.1.3 Perceptron Using Margins

The classical Perceptron attempts to separate the data but has no guarantees on the separation margin obtained. The Perceptron Algorithm using Margins (PAM) [Krauth and Mézard, 1987, Dagan et al., 1997, Grove and Roth, 1997, Li et al., 2002] attempts to establish such a margin, τ, during the training process. Following work on support vector machines [Boser et al., 1992, Cristianini and Shawe-Taylor, 2000] one may expect that providing the Perceptron with higher margin will add to the

stability and accuracy of the hypothesis produced, and in fact PAM has been shown to perform well in experimental settings [Dagan et al., 1997, Grove and Roth, 1997, Punyakanok et al., 2008, e.g.,].

To establish the margin, instead of only updating on examples for which the classifier makes a mistake, PAM updates on x_j if

$$y_j(SUM - \theta) < \tau$$

where SUM is as in Equation (3.2). Notice that this includes the case of a mistake where $y_j(SUM - \theta) < 0$ and the case of correct classification with low margin when $0 \leq y_j(SUM - \theta) < \tau$. In this way, the algorithm "establishes" some minimal separation of the data. Notice that the weight vector $\mathbf{w}$ is not normalized during the learning process and its norm can increase with updates. Therefore, the constraint imposed by τ becomes less restrictive as learning progresses, and the normalized margin is not τ. When the data are linearly separable and $\tau < \gamma_{opt}$, PAM finds a separating hyperplane with a margin that is guaranteed to be at least $\gamma_{opt}\frac{\tau}{\sqrt{8}(\eta R^2+\tau)}$, where γ_{opt} is the maximum possible margin [Krauth and Mézard, 1987, Li et al., 2002].[1]

As above, it is convenient to make the margin parameter dataset independent in order that we can use the same values across datasets. To facilitate this we replace the above and update if

$$y_j(SUM - \theta) < \tau\theta_{Init} \tag{3.3}$$

in order that τ can be measured in units of θ_{Init}.

A variant of PAM exists in which a different value of τ is chosen for positive examples than for negative examples [Li et al., 2002]. This seems to be important when the class distribution is skewed. We do not study that variant in this chapter.

[1] Other variants [e.g., Gentile, 2001, Tsampouka and Shawe-Taylor, 2005, Shalev-Shwartz and Singer, 2005] do normalize the weight vector and thus have better margin guarantees.

3.1.4 Longest Survivor and Voted Perceptron

The classical Perceptron returns the last weight vector $\mathbf{w}$, i.e., the one obtained after all training has been completed, but this may not always be useful especially if the data is noisy. This is a general issue that has been studied in the context of using on-line algorithms that expect one example at a time in a batch setting where a set of examples is given for training and one hypothesis is used at the end to classify all future instances. Several variants exist that handle this issue. In particular Kearns et al. [1987] show that *longest survivor* hypothesis, i.e., the one who made the largest number of correct predictions during training in the on-line setting, is a good choice in the sense that one can provide guarantees on its performance in the PAC learning model [Valiant, 1984]. Several variations of this idea were independently proposed under the name of the *pocket algorithm* and empirical evidence for their usefulness was provided [Gallant, 1990].

The Voted Perceptron [Freund and Schapire, 1999] assigns each vector a "vote" based on the number of sequential correct classifications by that weight vector. Whenever an example is misclassified, the Voted Perceptron records the number of correct classifications made since the previous misclassification, assigns this number to the current weight vector's vote, saves the current weight vector, and then updates as normal. After training, all the saved vectors are used to classify future examples and their classifications are combined using their votes. At first sight the Voted Perceptron seems to require expensive calculation for prediction. But as pointed out by Freund and Schapire [1999], the output of the weight vector resulting from the first k mistakes can be calculated from the output of the weight vector resulting from the first $k-1$ mistakes in constant time. Therefore when using the dual Perceptron the prediction phase of the Voted Perceptron is not substantially more expensive than the prediction phase of the classical Perceptron, although it is more expensive in the primal form.

When the data are linearly separable and given enough iterations, both these

variants will converge to a hypothesis that is very close to the simple Perceptron Algorithm. The last hypothesis will predict correctly on all examples and indeed its vote will be the largest vote among all hypotheses used. When the data are not linearly separable the quality of the hypothesis may fluctuate during training as noisy examples are encountered.

3.1.5 Algorithms Summary

Figure 3.1 summarizes the various algorithms and prediction strategies in primal form. The algorithms have corresponding dual forms and kernelized versions that we address in the experimental sections.

3.1.6 Multi-Class Data

Some of the datasets we experiment with have more than two labels. In such cases we use the one-versus-all binary to multiclass extension explained in Section 2.5 on page 17, due to its simplicity and good performance. During testing we calculate the weight given by each classifier (before the sign function is applied) and choose the label of the classifier with maximum weight.

3.2 Experimental Evaluation

We ran two sets of experiments with the algorithms described above. In one set of experiments we searched through a pre-determined set of values of τ, α, and λ by running each of the classical, longest survivor, and Voted Perceptron using 10-fold cross-validation and parameterized by each value of τ, α, and λ as well as each combination of $\tau \times \alpha$ and $\tau \times \lambda$. This first set of experiments is called *parameter search*. The purpose of the parameter search experiments was to give us a comprehensive view of the effects of the various parameters. This can show whether a method has any chance of improving performance since the experiments give us hindsight knowledge. The experiments can also show general patterns and

TRAINING: Input set of examples and their labels
$\mathcal{Z} = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathbb{R}^n \times \{-1, 1\})^m$, θ_{Init}, η, α-bound, λ, τ

- Initialize $\alpha \leftarrow 0^m$, $k \leftarrow 0$, $\mathbf{w_0} \leftarrow \mathbf{0}$, $tally \leftarrow 0$, $best\_tally \leftarrow 0$, $\theta_0 \leftarrow \theta_{Init}, \theta_{l.s.} \leftarrow \theta_{Init}$
- for every training iteration
- for every $z_j \in \mathcal{Z}$:
  - $SUM \leftarrow \langle w_k, x_j \rangle + I_{(\alpha_j \neq 0)}\ y_j \lambda \|x_j\|$
  - Predict:
    - if $SUM < \theta_k - \tau$ then $\hat{y} = -1$
    - else if $SUM > \theta_k + \tau$ then $\hat{y} = 1$
    - else $\hat{y} = 0$ // This forces an update
  - if $(\hat{y} \neq y_j)$ AND $(\alpha_j < \alpha$-bound$)$
    - $w_{k+1} \leftarrow w_k + \eta y_j x_j$
    - $\alpha_j \leftarrow \alpha_j + 1$
    - Update ''mistakes'' data structure
    - $\theta_{k+1} \leftarrow \theta_k + \eta y_j \theta_{\texttt{init}}$
    - $vote_{k+1} \leftarrow 0$
    - $k \leftarrow k + 1$
    - $tally \leftarrow 0$
  - else if $(\hat{y} = y_j)$
    - $vote_k \leftarrow vote_k + 1$
    - $tally \leftarrow tally + 1$
    - if $(tally > best\_tally)$
      - $best\_tally \leftarrow tally$
      - $w_{l.s.} \leftarrow w_k$
      - $\theta_{l.s.} \leftarrow \theta_k$

PREDICTION: To predict on a new example x_{m+1}:

- CLASSICAL:
  - $SUM \leftarrow \langle w_k, x_{m+1} \rangle$
  - $\hat{y} \leftarrow sign(SUM - \theta_k)$
- LONGEST SURVIVOR:
  - $SUM \leftarrow \langle w_{l.s.}, x_{m+1} \rangle$
  - $\hat{y} \leftarrow sign(SUM - \theta_{l.s.})$
- VOTED:
  - for $i \leftarrow 1$ to k
    - $SUM \leftarrow \langle w_i, x_{m+1} \rangle$
    - $\hat{y} \leftarrow sign(SUM - \theta_i)$
    - VOTES $\leftarrow$ VOTES $+vote_i\ \hat{y}$
  - $\hat{y}_{final} \leftarrow sign($VOTES$)$

Figure 3.1: Illustration of All Algorithm Variants

trends in the parameter landscape again giving insight into the performance of the methods. Notice that parameter search cannot be used to indicate good values of parameters as this would be hand-tuning the algorithm based on the test data. However, it can guide in developing methods for automatic selection of parameters.

In the second set of experiments, we used the same set of values as in the first experiment, but using a method for automatic selection of parameters. In particular we used a "double cross validation" where in each fold of the cross validation (1) one uses parameter search on the training data only using another level of cross validation, (2) picks values of parameters based on this search, (3) trains on the complete training set for the fold using these values, and (4) evaluates on the test set. We refer to this second set of experiments as *parameter optimization*. This method is expensive to run as it requires running all combinations of parameter values in each internal fold of the outer cross validation. Hence if both validations use ten folds then we run the algorithm 100 times per parameter setting. This sets a strong limitation on the number of parameter variations that can be tried. Nonetheless this is a rigorous method of parameter selection.

Both sets of experiments were run on randomly-generated artificial data and real-world data from the University of California at Irvine (UCI) repository [Asuncion and Newman, 2007] and other sources. The purpose of the artificial data was to simulate an ideal environment that would accentuate the strengths and weaknesses of each algorithm variant. The other datasets explore the extent to which this behavior is exhibited in real world problems.

For further comparison, we also ran SVM on the datasets using the L_1-norm soft margin and L_2-norm soft margin. We used SVM Light [Joachims, 1999] and only ran *parameter search*.

Table 3.1: UCI and Other Natural Dataset Characteristics

Name	# features*	# examples	Baseline
Adult	105	32561	75.9
Breast-cancer-wisconsin	9	699	65.5
Bupa	6	345	58
Crx	46	690	55.5
Ionosphere	34	351	64.4
MNIST	784	70,000	N/A
sonar.all-data	60	208	53.4
USPS	256	9298	N/A
Wdbc	30	569	62.7
Wpbc	33	198	76.3
*after preprocessing			

3.2.1 Dataset Selection and Generation

We have experimented with ten real world datasets of varying size and 150 artificial datasets composed of 600 examples each.

We first motivate the nature of artificial data used by discussing the following four idealized scenarios. The simplest, type 1, is linearly separable data with very small margin. Type 2 is linearly separable data with a larger, user-defined margin. For type 3 we first generate data as in type 1, and then add random class noise by randomly reversing the labels of a given fraction of examples. For type 4, we generate data as in type 2, and then add random class noise. One might expect that the basic Perceptron algorithm will do fine on type 1 data, the Perceptron using Margins will do particularly well on type 2 data, that noise tolerant variants without margin will do well on type 3, and that some combination of noise tolerant variant with margin will be best for type 4 data. However, our experiments show that the picture is more complex; preliminary experiments confirmed that the expected behavior is observed, except that the Perceptron using Margins performed well on data of type 3 as well, that is, when the "natural margin" was small and the data was not separable due to noise. In the following, we report on experiments with artificial data where the margin and noise levels are varied thus we effectively span all four different types.

Concretely the data was generated as follows. We first specify parameters for

number of features, noise rate and the required margin size. Given these, each example x_i is generated independently and each attribute in an example is generated independently using an integer value in the range $[-10, 10]$. We generated the weight vector, w, by flipping a coin for each example and adding it to the weight vector on a head. We chose θ as the average of $\frac{1}{2}|\langle w, x_i \rangle|$ over all x_i. We measured the actual margin of the examples with respect to the weight vector and then discarded any examples x_i such that $|\langle w, x_i \rangle - \theta| < M * \theta$ where M is the margin parameter specified. For the noisy settings, for each example we randomly switched the label with probability equal to the desired noise rate. In the tables of results presented below, f stands for number of features, M stands for the margin size, and N stands for the noise rate. We generated datasets with parameters $(f, M, N) \in \{50, 200, 500\} \times \{0.05, 0.1, 0.25, 0.5, 0.75\} \times \{0, 0.05, 0.1, 0.15, 0.25\}$, and for each parameter setting we generated two datasets, for a total 150 datasets.

For real world data we first selected two-class datasets from the UCI Machine Learning Repository [Asuncion and Newman, 2007] that have been used in recent comparative studies or in recent papers on linear classifiers [Cohen, 1995, Dietterich et al., 1996, Dietterich, 2000, Garg and Roth, 2003]. Because we require numerical attributes, any nominal attribute in these datasets was translated to a set of binary attributes each being an indicator function for one of the values. As all these datasets have a relatively small number of examples we added three larger datasets to strengthen our conclusions statistically: "Adult" from UCI [Newman et al., 1998], and "MNIST"[2] and "USPS"[3], the 10-class character recognition datasets. Due to their size, however, for the "USPS," "MNIST," and "Adult" datasets, there is no outer cross-validation in *parameter optimization*. For ease of comparison to other published results, we use the 7291/2007 training/test split for "USPS", and a 60000/10000 split for "MNIST".

The datasets used and their characteristics after the nominal-to-binary feature

[2] `http://yann.lecun.com/exdb/mnist/`

[3] `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html`

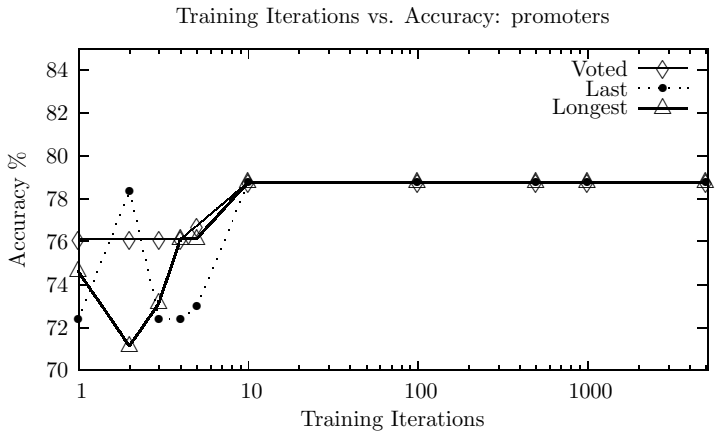

Figure 3.2: Example Learning Curve for 'UCI Dataset *promoters*

transformation are summarized in Table 3.1. The column labeled "Baseline" indicates the percentage of the majority class.

3.2.2 Exploratory Experiments and General Setup

The algorithms described above have several parameters that can affect their performance dramatically. In this chapter we are particularly interested in studying the effect of parameters related to noise tolerance, and therefore we fix the value of θ_{Init}, η, and the number of training iterations. Prior to fixing these values, we ran preliminary experiments in which we varied these parameters. In addition we ran experiments where we normalized the example vectors. The results showed that while the performance of the algorithms overall was different in these settings, the relationship between the performance of individual algorithms seems to be stable across these variations.

We set $\theta_{Init} = avg(\langle x_i, x_i \rangle)$, initializing the threshold at the same scale of inner products. Combined with a choice of $\eta = 0.1$, this makes sure that a few iterations should be able to guarantee that an example is classified correctly given no other changes to the hypothesis.

As explained above, the number of iterations must be sufficiently high to allow the α parameter to be effective, as well as to allow the weight vector to achieve

some measure of stability. Typically a small number of iterations is sufficient, and preliminary experiments illustrated by the curve in Figure 3.2 showed that 100 iterations are more than enough to allow all the algorithms to converge to a stable error rate. Therefore, except where noted below, we report results for 100 iterations.

For the large datasets, we reduced the number of training iterations and increased η accordingly in order to run the experiments in a reasonable amount of time; for "Adult" we set $\eta = 0.4$ with 5 training iterations and for "MNIST" we set $\eta = 1$ with 1 training iteration. For "USPS," we used $\eta = 0.1$ and 100 iterations for *parameter search*, and $\eta = 0.1$ and 10 iterations for *parameter optimization*.

The parameters of interest in our experiments are τ, α, λ that control the margin and soft margin variants. Notice that we presented these so that their values can be fixed in a way that is dataset independent. We used values for $\tau \in \{0, 0.125, 0.25, 0.5, 1, 2, 4\}$, $\alpha \in \{\infty, 80, 60, 40, 20, 10, 5\}$, and $\lambda \in \{0, 0.125, 0.25, 0.5, 1, 2, 4\}$. Notice that the values as listed from left to right vary from no effect to a strong effect for each parameter. We ran *parameter search* and *parameter optimization* over all of these values, as well as all combinations $\tau \times \lambda$ and $\tau \times \alpha$. Since *parameter optimization* over combined values is particularly expensive we have also experimented with a variant that first searches for a good τ value and then searches for a value of α (denoted $\tau \rightarrow \alpha$) and a variant that does likewise with τ and λ (denoted $\tau \rightarrow \lambda$).

We did not perform any experiments involving α on "MNIST" or "Adult" as their size required too few iterations to justify any reasonable α-bound.

Table 3.2: Noise Percentage vs. Dominance: V = Voted, C = Classical, LS = Longest Survivor

	V > C	V > LS	LS > C	LS > V	C > LS	C > V	V = LS = C
Noise = 0	0	0	0	0	0	0	30
Noise = 0.05	12	10	11	3	0	2	16
Noise = 0.1	15	15	14	3	3	3	12
Noise = 0.15	16	14	13	5	6	3	10
Noise = 0.25	16	12	13	8	7	4	10

Finally we performed a comparison of the classical Perceptron, the longest sur-

vivor and the Voted Perceptron algorithms without the additional variants. Table 3.2 shows a comparison of the accuracies obtained by the algorithms over the artificial data. We ignore actual values but only report the number of times one algorithm gives higher accuracy or whether they tie (the variance in actual results is quite large). One can see that with higher noise rate the Voted Perceptron and longest survivor improve the performance over the base algorithm. Over all 150 artificial datasets, the Voted Perceptron strictly improves performance over the classical Perceptron in 59 cases, and ties or improves in 138 cases. Using the distribution over our artificial datasets one can calculate a simple weak statistical test that supports the hypothesis that using the voted algorithm does not hurt accuracy, and can often improve it.

Except where noted, all the results reported below give average accuracy in 10-fold cross-validation experiments. To avoid any ordering effects of the data, the training set is randomly permuted before each run.

3.2.3 Parameter Search

The *parameter search* experiments reveal several interesting aspects. We observe that in general the variants are indeed helpful on the artificial data as the performance increases substantially from the basic version. The numerical results are shown in Tables 3.3 and 3.4 and discussed below. Before showing these we discuss the effects of single parameters. Figure 3.3 plots the effect of single parameters for several datasets. For the artificial dataset shown, the accuracy is reasonably well-behaved with respect to the parameters and good performance is obtained in a non-negligible region; this is typical of the results from the artificial data. Data obtained for the real world datasets show somewhat different characteristics. In some datasets little improvement is obtained with any variant or parameter setting. In others, improvement was obtained for some parameter values but the regions were not as large implying that that automatic parameter selection may not be

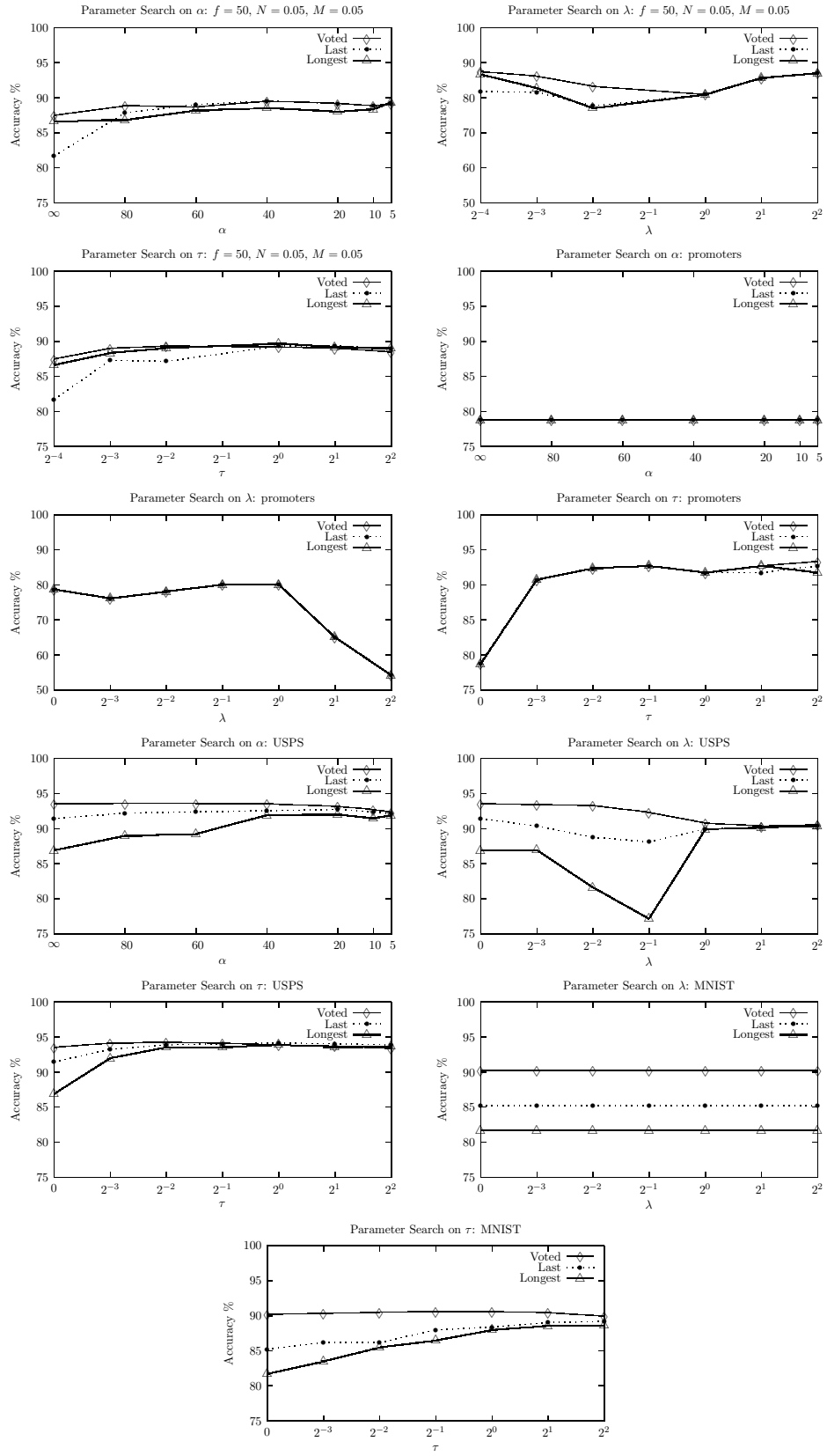

Figure 3.3: Parameter Search on Artificial and Real-world Data

easy. Nonetheless it appears that when improvement is possible, τ on it own was quite effective when used with the classical Perceptron. Notice that τ is consistently effective across all datasets; λ and α do not help on all datasets and λ sometimes leads to a drop in performance. As one might expect, our preliminary experiments also showed that very large values of τ harm performance significantly. These are not shown in the graphs as we have limited the range of τ in the experiments.

Tables 3.3 and 3.4 summarize the results of parameter search experiments on the real world data and some of the artificial data, respectively. For each dataset and algorithm the tables give the best accuracy that can be achieved with parameters in the range tested. This is useful as it can indicate whether an algorithm has a potential for improvement, in that for some parameter setting it gives good performance. In order to clarify the contribution of different parameters, each column with parameters among τ, α, λ includes all values for the parameter except the non-active value. For example, any accuracy obtained in the τ column is necessarily obtained with a value $\tau \neq 0$. The column labeled "Nothing" shows results when all parameters are inactive.

Several things can be observed in the tables. Consider first the margin parameters. We can see that τ is useful even in datasets with noise; we see this both in the noisy artificial datasets and in the real-world datasets, all of which are inseparable in the native feature space. We can also see that α and λ do improve performance in a number of cases. However, they are less effective in general than τ, and do not provide additional improvement when combined with τ.

Looking next at the on-line to batch conversions we see that the differences between the basic algorithm, the longest survivor and the Voted Perceptron are noticeable without margin based variants. For the artificial datasets this only holds for one group of datasets ($f = 50$), the one with highest ratio of number of examples to number of features ($12 : 1$).[4] The longest survivor seems less stable and degrades

[4]The results for data with $f = \{200, 500\}$ are omitted from the table. In these cases these was no difference between the basic, longest and voted versions except when combining with other

Table 3.3: Parameter Search on UCI and Other Datasets.

		Baseline	Nothing	τ	λ	α	$\tau \times \lambda$	$\tau \times \alpha$
breast-cancer-wisconsin	Last	65.5	90.6	96.8	97.2	96.9	97.3	97.2
	Longest		96.9	97.0	97.2	97.0	97.3	97.2
	Voted		96.9	96.8	97.2	96.9	97.3	97.2
bupa	Last	58	57.5	71.8	64.1	69.2	71.5	67.8
	Longest		64.1	58.2	64.8	68.4	60.9	61.2
	Voted		68.9	65.9	67.7	70.7	67.4	66.0
wdbc	Last	62.7	92.4	93.2	92.8	93.3	93.9	92.6
	Longest		92.4	93.2	92.6	92.4	92.8	92.8
	Voted		92.3	92.0	93.2	92.4	93.2	92.4
crx	Last	55.5	62.5	68.6	65.5	66.7	69.0	66.7
	Longest		55.1	63.6	65.5	63.5	65.5	65.2
	Voted		64.9	65.4	65.5	66.7	65.5	66.4
promoters	Last	50	78.8	92.8	82.1	78.8	94.4	93.8
	Longest		78.8	92.8	82.1	78.8	94.4	94.4
	Voted		78.8	93.4	82.1	78.8	94.4	93.8
ionosphere	Last	64.4	86.6	87.5	86.9	87.7	88.6	87.2
	Longest		87.2	87.5	87.8	88.0	88.3	87.7
	Voted		88.0	87.7	87.5	88.6	88.9	87.5
wpbc	Last	76.3	77.3	76.4	80.6	76.9	79.0	76.4
	Longest		77.2	76.4	77.4	78.5	76.9	76.4
	Voted		78.8	76.4	80.6	77.4	78.5	76.4
sonar.all-data	Last	53.4	71.9	74.6	72.5	77.1	75.8	79.1
	Longest		75.3	77.1	73.3	77.2	77.7	78.7
	Voted		75.1	77.2	73.8	77.1	78.7	77.7
USPS	Last	N/A	91.5	94.1	90.4	92.8	94.3	94.1
	Longest	N/A	86.9	93.9	90.4	92	93.9	93.9
	Voted	N/A	93.5	94.2	93.4	93.6	94.3	94.3
MNIST	Last	N/A	85.2	89.2	85.2		89.2	
	Longest	N/A	81.7	88.6	81.7		88.6	
	Voted	N/A	90.2	90.6	90.2		90.6	

Table 3.4: Parameter Search on Artificial Datasets with $f = 50$ and $M = 0.05$.

Noise Pctg. (N)	Nothing	τ	λ	α	$\tau \times \lambda$	$\tau \times \alpha$
Last, $N = 0$	95.4	97	95.6	96.1	96.8	97
Longest	95.4	96.6	95.6	95.9	97.3	96.8
Voted	95.4	96.6	95.4	96.1	96.8	97
Last, $N = 0.05$	81.7	89.4	87	89.5	89.7	89.9
Longest	86.6	89.7	87	89.3	89.9	89.9
Voted	87.5	89.4	87	89.5	89.7	90.2
Last, $N = 0.1$	77.8	84.6	81.9	85.1	84.9	85.8
Longest	77.2	84.9	81.9	85.1	84.6	85.5
Voted	81.6	84.9	81.9	85.1	84.9	85.6
Last, $N = 0.15$	71.2	81.6	79.9	80.6	81.6	81.7
Longest	72.9	80.7	79.9	79.2	80.7	81.7
Voted	75.6	81.6	79.9	80.6	82.1	81.6
Last, $N = 0.25$	59.9	70.7	68.2	70.7	71.1	71.1
Longest	65	70.7	68.2	70.1	70.4	70.6
Voted	68	70.4	69.4	70.7	70.6	71.1

performance in some cases.

Finally compare the τ variant with Voted Perceptron and their combination. For the artificial datasets using τ alone (with last hypothesis) gives higher accuracy than using the Voted Perceptron alone. For the real-world datasets this trend is less pronounced. Concerning their combination we see that while τ always helps the last hypothesis, it only occasionally helps voted, and sometimes hurts it.

Table 3.5 gives detailed results for parameter search over τ on the adult dataset, where we also parameterize the results by the number of examples in the training set. We see that Voted Perceptron and the τ variant give similar performance on this dataset as well. We also see that that in contrast with the performance on the artificial data mentioned above, Voted Perceptron performs well even with small training set size (and small ratio of examples to features).

The table adds another important observation about stability of the algorithms. Note that because we report results for concrete values of τ, we can measure the

variants.

Table 3.5: Performance on "Adult" dataset as a function of margin and training set size.

# examples:	10		100		1000		5000		10000		20000	
	Accuracy	Std. Dev.										
$\tau = 0$												
Last	75.6	3.5	70.2	9.2	77.8	3.1	78.3	4.1	80.3	2.8	76.5	11.1
Longest	75.7	3.5	76.5	3.6	79.9	3.5	81.8	1.8	82.9	0.7	82.3	1.3
Voted	75.8	3.4	79.4	1.8	82.5	0.5	84	0.7	84.2	0.6	84.5	0.5
$\tau = 0.125$												
Last	72.7	4.6	73.3	8.9	79.7	3.7	80.8	2.7	82.7	1.4	80.4	3.9
Longest	73.1	4.6	76.6	3.9	81	2.1	81.1	2.4	81.8	1.5	82.1	1.7
Voted	74.1	4.4	79.6	1.4	82.7	0.5	83.9	0.7	84.2	0.6	84.3	0.4
$\tau = 0.25$												
Last	74.5	1.8	75.1	7.4	79.3	5.1	80.5	2.8	81.5	2.2	80.6	3.7
Longest	73.9	3.9	77.2	2.1	80.3	2.6	81.8	1.8	81.7	1.7	82.3	2.4
Voted	74.1	4.7	79.5	1.5	82.7	0.5	83.8	0.7	84.1	0.7	84.3	0.5
$\tau = 0.5$												
Last	71.8	6.3	75.5	1.9	80.4	3.4	82.6	1	82.7	1	82.8	1.5
Longest	74	5.7	77.2	7.5	80.4	2.3	82.1	1.6	82.5	1.6	80.8	2.9
Voted	73.6	5.6	78.3	0.8	82.6	0.6	83.7	0.7	84	0.7	84.2	0.5
$\tau = 1$												
Last	72.9	6.6	78.4	2.5	81.9	1.2	83	1.1	82.7	1.5	83.5	0.8
Longest	75.9	0.3	75.9	0.9	78.3	2.8	82.5	1.1	83	1	81.9	2.7
Voted	74.8	3.4	76.4	0.9	82.4	0.6	83.5	0.7	83.7	0.7	84.1	0.5
$\tau = 2$												
Last	75.5	1.2	75.8	2.6	82.4	0.4	83.2	0.9	82.8	1.2	83.6	0.8
Longest	70.7	15.6	75.9	0.9	77.8	2.7	81.4	2.9	82.5	2	81.9	2.8
Voted	70.7	15.6	76.4	1.8	81.8	1	83.2	0.7	83.5	0.6	83.8	0.6
$\tau = 4$												
Last	75.9	0.3	75.7	1.4	81.8	1	83.1	2.1	83.3	0.6	83.6	0.6
Longest	70.7	15.6	75.9	0.9	75.9	0.5	81.8	0.7	79.4	3.4	80.8	3.1
Voted	70.7	15.6	75.9	0.9	78.4	2.1	83	0.7	83.2	0.6	83.5	0.6

standard deviation in accuracy observed. One can see that both the τ variant and the Voted Perceptron significantly reduce the variance in results. The longest survivor does so most of the time but not always. The fact that the variants lead to more stable results is also consistently true across the artificial and UCI datasets discussed above and is an important feature of these algorithms.

Finally, recall that the Average Algorithm [Servedio, 1999], which updates exactly once on each example, is known to tolerate classification noise for data distributed uniformly on the sphere and where the threshold is zero. For comparison, we ran this algorithm on all the data reflected in the tables above and it was not

competitive. Cursory experiments to investigate the differences show that, when the data has a zero threshold separator and when Perceptron is run for just one iteration, then (as reported in [Servedio, 1999]) Average performs better than basic Perceptron. However, the margin-based Perceptron performs better and this becomes more pronounced with non-zero threshold and multiple iterations. Thus in some sense the Perceptron variants are doing something non-trivial that is beyond the scope of the simple Average Algorithm.

3.2.4 Parameter Optimization

We have run the parameter optimization on the real-world datasets and the artificial datasets. Selected results are given in Tables 3.6 and 3.7. For these experiments we report average accuracy across the outer cross-validation as well as a 95% T-confidence interval around these, as suggested by Mitchell [1997].[5] No confidence interval is given for "USPS," "MNIST," or "Adult," as the outer cross-validation loop is not performed.

In contrast with the tables for parameter search, columns of parameter variants in Tables 3.6 and 3.7 do include the inactive option. Hence the search in the τ column includes the value of $\tau = 0$ as well. This makes sense in the context of parameter optimization because the algorithm can choose between different active values and the inactive value. Notice that the standard deviation in accuracies is high on these datasets. This highlights the difficulty of parameter selection and algorithm comparison and suggests that results from single split into training and test sets that appear in the literature may not be reliable.

Table 3.6 shows improvement over the basic algorithm in all datasets where parameter search suggested a potential for improvement, and no decrease in perfor-

[5] In more detail, we use the maximum likelihood estimate of the standard deviation σ, $s = \sqrt{\frac{1}{k}\sum_i (y_i - \bar{y})^2}$ where k is the number of folds (here $k = 10$), y_i is the accuracy estimate in each fold and $\bar{y}$ is the average accuracy. We then use the T confidence interval $y \in \hat{y} \pm t_{(k-1),0.975}\sqrt{\frac{1}{k(k-1)}\sum (y_i - \bar{y})^2}$. Notice that since $t_{9,0.975} = 2.262$ and $k = 10$ the confidence interval is 0.754 of the standard deviation.

Table 3.6: Parameter Optimization Results for UCI and Artificial Datasets.

	Nothing	τ	λ	α	$\tau \times \alpha$	$\tau \times \lambda$	$\tau \rightarrow \alpha$	$\tau \rightarrow \lambda$	SVM L1	SVM L2
Breast-cancer-wisconsin										
Last	90.6 +/- 2.3	95.2 +/- 2.1	95.2 +/- 3.1	94.7 +/- 3.3	95.3 +/- 2.6	96.3 +/- 2.1	95.1 +/- 2.3	96.9 +/- 1.3		
Longest	96.9 +/- 1.2	96.8 +/- 1.7	97.2 +/- 1.4	96.3 +/- 1.9	96.8 +/- 1.6	97 +/- 1.6	96.8 +/- 1.7	96.9 +/- 1.8	96.8 +/- 2.2	96.7 +/- 1.7
Voted	96.9 +/- 1.3	96.8 +/- 1.4	97 +/- 1.4	96.6 +/- 1.6	97 +/- 1.6	97.2 +/- 1.4	96.8 +/- 1.4	96.9 +/- 1.5		
Bupa										
Last	57.5 +/- 7.8	69.8 +/- 6.5	61.5 +/- 6.3	68.9 +/- 4.1	69.9 +/- 5.7	69.8 +/- 5.8	69.8 +/- 6.5	70.9 +/- 5.9		
Longest	64.1 +/- 7.1	64.1 +/- 7.1	64.1 +/- 6.6	65.3 +/- 4.3	65.3 +/- 4.3	64.1 +/- 6.6	65.3 +/- 4.3	64.1 +/- 6.6	66.5 +/- 8.6	63.2 +/- 5.2
Voted	68.9 +/- 5.8	68.9 +/- 5.8	67.5 +/- 6.4	68.9 +/- 6.3	68.9 +/- 6.3	67.5 +/- 6.2	68.9 +/- 6.3	67.2 +/- 6.0		
Wdbc										
Last	92.4 +/- 2.9	93 +/- 2.7	93.2 +/- 2.5	91.6 +/- 2.7	92.8 +/- 2.8	93.2 +/- 2.8	93 +/- 2.7	93.5 +/- 2.2		
Longest	92.4 +/- 2.0	91.7 +/- 2.5	92.1 +/- 2.5	92.3 +/- 2.3	93.2 +/- 1.7	92.5 +/- 2.1	92.6 +/- 2.1	91.4 +/- 2.9	92.8 +/- 4.4	94.7 +/- 2.1
Voted	92.3 +/- 2.3	92.3 +/- 2.6	92.8 +/- 2.7	91.7 +/- 1.9	91.6 +/- 2.2	92.4 +/- 2.4	91.6 +/- 2.6	92.8 +/- 2.3		
Crx										
Last	62.5 +/- 5.1	68.7 +/- 2.9	64.5 +/- 4.1	65.8 +/- 4.1	68.1 +/- 3.5	68.7 +/- 2.2	68.7 +/- 2.9	67.8 +/- 2.8		
Longest	55.1 +/- 7.3	60.4 +/- 6.5	62.6 +/- 4.8	62.6 +/- 4.4	64.5 +/- 4.9	60.9 +/- 5.7	62.9 +/- 4.6	59.4 +/- 6.7	76.6 +/- 13.7	65.9 +/- 22.8
Voted	64.9 +/- 4.0	64.2 +/- 2.7	65.4 +/- 3.2	66.2 +/- 3.8	66.4 +/- 3.7	64.9 +/- 2.6	65.7 +/- 3.3	64.3 +/- 3.1		
Ionosphere										
Last	86.6 +/- 3.8	87.2 +/- 3.4	86.3 +/- 3.5	85.7 +/- 4.8	86.9 +/- 3.4	86.9 +/- 2.6	86.6 +/- 3.0	86.6 +/- 2.6		
Longest	87.2 +/- 3.8	86.9 +/- 2.6	87.8 +/- 3.6	87.5 +/- 3.5	86.9 +/- 4.0	87.2 +/- 3.2	86.9 +/- 3.4	87.7 +/- 2.9	87.1 +/- 7.1	86.9 +/- 4.2
Voted	88 +/- 3.7	86.3 +/- 4.3	86.6 +/- 3.5	87.7 +/- 3.2	87.5 +/- 3.2	87.7 +/- 3.1	86 +/- 4.9	86 +/- 3.9		
Wpbc										
Last	77.3 +/- 4.7	76.7 +/- 4.4	76.4 +/- 4.4	75.1 +/- 6.1	75.1 +/- 6.1	77 +/- 5.2	75.7 +/- 6.2	78.3 +/- 5.1		
Longest	77.2 +/- 3.8	76.7 +/- 5.3	75.8 +/- 3.6	75.8 +/- 6.0	76.9 +/- 4.4	74.8 +/- 4.3	75.8 +/- 4.4	74.6 +/- 5.4	78.6 +/- 7.8	78.6 +/- 8.4
Voted	78.8 +/- 4.7	78.5 +/- 4.8	79 +/- 5.0	76.4 +/- 4.8	76.9 +/- 4.8	79 +/- 5.0	76.9 +/- 4.8	78.5 +/- 5.1		
Sonar										
Last	71.9 +/- 6.3	73.1 +/- 5.4	72.4 +/- 6.0	75.5 +/- 6.4	72.1 +/- 7.4	71.1 +/- 6.5	74.1 +/- 6.4	73.6 +/- 5.1		
Longest	75.3 +/- 5.1	77.6 +/- 6.2	73.3 +/- 6.0	74.3 +/- 6.9	73.5 +/- 5.9	74.1 +/- 7.5	74 +/- 5.5	78.1 +/- 6.6	57.2 +/- 11.9	55 +/- 11.8
Voted	75.1 +/- 6.0	75.6 +/- 6.9	74.3 +/- 5.6	77.5 +/- 7.0	78.1 +/- 7.0	77.1 +/- 7.2	76.1 +/- 7.9	75.6 +/- 6.9		
f=50,N=0.05,M=0.05										
Last	81.7 +/- 6.5	87.7 +/- 6.1	84.6 +/- 6.6	87.2 +/- 5.2	88 +/- 5.2	89.4 +/- 4	89 +/- 4.6	89.5 +/- 4.2		
Longest	86.6 +/- 4.3	88.2 +/- 4.2	85.6 +/- 4.7	87.3 +/- 6.1	89.4 +/- 4.5	88.2 +/- 3.9	89 +/- 5.4	88.8 +/- 3.6		
Voted	87.5 +/- 3.3	88.8 +/- 4.8	86.3 +/- 3.9	88.3 +/- 4.2	88.2 +/- 4.9	88.8 +/- 4.8	88.7 +/- 4.9	88.8 +/- 4.8		

Table 3.7: Parameter Optimization Results for Large Datasets

	Nothing	τ	λ	$\tau \times \lambda$	$\tau \rightarrow \lambda$
USPS					
Last	87.4	90.7	87.4	90.3	90.2
Longest	87.5	89.9	87.4	90.3	89.9
Voted	89.7	90.9	89.6	90.9	90.9
Adult					
Last	81.9	83.9	83	83.8	83.8
Longest	83.4	83.4	83.4	83.6	83.4
Voted	84.4	84.2	84.4	84.3	84.3
MNIST					
Last	85.6	87.1	85.5	87.1	87.1
Longest	79.8	86.8	79.8	86.8	86.8
Voted	88	88.4	88	88.4	88.4
USPS,degree 4 poly kernel					
Last	92.9	93.6			
Longest	91.6	92.9			
Voted	92.7	93.2			
MNIST,degree 4 poly kernel					
Last	95.2	95.4			
Longest	93.2	94.9			
Voted	94.8	95			

mance in the other cases, hence the parameter selection indeed picks good values. Both τ and the Voted Perceptron provide consistent improvement over the classical Perceptron; the longest survivor provides improvement over the classical Perceptron on its own, but a smaller one than voted or τ in most cases. Except for improvements over the classical Perceptron, none of the differences between algorithms is significant according to the T-intervals calculated. As observed above in *parameter search*, the variants with α and λ offer improvements in some cases, but when they do, τ and voted almost always offer a better improvement. Ignoring the intervals we also see that neither the τ variant nor the voted Perceptron dominates the other. Combining the two is sometimes better but may decrease performance in the high variance cases.

For further comparison we also ran experiments with kernel based versions of the algorithms. Typical results in the literature use higher degree polynomial kernel on the "MNIST" and "USPS" datasets. Table 3.7 includes results using τ with a

degree 4 polynomial kernel for these datasets. We can see that for "MNIST" the variants make little difference in performance but that for "USPS" we get small improvements and the usual pattern relating the variants is observed.

Finally, Table 3.6 also gives results for SVM. We have used SVMlight [Joachims, 1999] and ran with several values for the constants controlling the soft margin. For L_1 optimization the values used for the "-c" switch in SVMLight are $\{10^{-5}$, 10^{-4}, $10^{-3}, \ldots, 10^{4}\}$. For the L_2 optimization, we used the following values for λ: $\{10^{-4}, 10^{-3}, \ldots, 10^{3}\}$. The results for SVM are given for the best parameters in a range of parameters tried. Thus these are parameter search values and essentially give upper bounds on the performance of SVM on these datasets. As can be seen the Perceptron variants give similar accuracies and smaller variance and they are therefore an excellent alternative for SVM.

3.3 Discussion and Conclusions

The chapter provides an experimental evaluation of several noise tolerant variants of the Perceptron algorithm. The results are surprising because they suggest that the Perceptron with Margin is the most successful variant although it is the only one not designed for noise tolerance. The Voted Perceptron has similar performance in most cases, and it has the advantage that no parameter selection is required for it. The difference between voted and Perceptron with Margin are most noticeable in the artificial datasets, and the two are indistinguishable in their performance on the UCI data. The experiments also show that the soft-margin variants are generally weaker than voted or margin based algorithms and they do not provide additional improvement in performance when combined with these. Both the Voted Perceptron and the margin variant reduced the deviation in accuracy in addition to improving the accuracy. This is an important property that adds to the stability of the algorithms. Combining voted and Perceptron with Margin has the potential for further improvements but can harm performance in high variance cases. In terms

of run time, the Voted Perceptron does not require parameter selection and can therefore be faster to train. On the other hand its test time is slower especially if one runs the primal version of the algorithm.

Overall, the results suggest that a good tradeoff is obtained by fixing a small (relative to the average norm of the examples) value of τ. We suggest $\tau = 0.1$; this gives significant improvements in performance without the training time penalty for parameter optimization or the penalty in test time for voting. In addition, because we have ruled out the use of the soft margin α variant we no longer need to run the algorithms for a large number of iterations. While in our experiments we used 100 iterations as the default for thoroughness, fewer iterations may well be sufficient. We suggest at least 20 iterations with a learning rate of 0.1 (see Figure 3.2 and discussion). For very large datasets, even 20 iterations may present a challenge in terms of run-time, and in these cases we suggest scaling η inverse-proportionately to the number of iterations as we do in Section 4.4.

Our results also highlight the problems involved with parameter selection. The method of double cross-validation is time intensive and our experiments for the large datasets were performed using the primal form of the algorithms as the dual form is too slow. In practice, with a large dataset one can use a hold-out set for parameter selection so that run time is more manageable. Such results must be accompanied by estimates of the deviation to provide a meaningful interpretation.

As mentioned in the introduction a number of variants that perform margin based updates exist in the literature [Friess et al., 1998, Gentile, 2001, Li and Long, 2002, Crammer et al., 2005, Kivinen et al., 2004, Shalev-Shwartz and Singer, 2005, Tsampouka and Shawe-Taylor, 2005]. For example, aggressive ROMMA [Li and Long, 2002] explicitly maximizes the margin on the new example, relative to an approximation of the constraints from previous examples. NORMA [Kivinen et al., 2004] performs gradient descent on the soft margin risk resulting in an algorithm that rescales the old weight vector before the additive update. The Passive-Aggressive

Algorithm [Crammer et al., 2005] adapts η on each example to guarantee it is immediately separable with margin (although update size is limited per noisy data). The Ballseptron [Shalev-Shwartz and Singer, 2005] establishes a normalized margin and replaces margin updates with updates on hypothetical examples on which a mistake would be made. ALMA [Gentile, 2001] renormalizes the weight vector in order to establish a normalized margin. ALMA is distinguished by tuning its parameters automatically during the on-line session, as well as having "p-norm variants" that can lead to other tradeoffs improving performance, for instance when the target is sparse. The algorithms of Tsampouka and Shawe-Taylor [2005] establish normalized margin by different normalization schemes. Following Freund and Schapire [1999] most of these algorithms come with mistake bound guarantees (or relative loss bounds) for the unrealizable case, i.e., relative to the best hypothesis in a comparison class. Curiously, identical or similar bounds hold for the basic Perceptron algorithm so that these results do not establish an advantage of the variants over the basic Perceptron.

Another interesting algorithm, the Second Order Perceptron [Cesa-Bianchi et al., 2005], does not perform margin updates but uses spectral properties of the data in the updates in a way that can reduce mistakes in some cases.

Additional variants also exist for the on-line to batch conversions. Littlestone [1989] picks the best hypothesis using cross validation on a separate validation set. The Pocket algorithm with ratchet [Gallant, 1990] evaluates the hypotheses on the entire training set and picks the best, and the scheme of Cesa-Bianchi et al. [2004] evaluates each hypothesis on the remainder of the training set (after it made a mistake and is replaced) and adjusts for the different validation set sizes. The results in [Cesa-Bianchi et al., 2004] show that loss bounds for the on line setting can be translated to error bounds in the batch setting even in the unrealizable case. Finally, experiments with aggressive ROMMA [Li and Long, 2002, Li, 2000] have shown that adding a Voted Perceptron scheme can harm performance, just as we

observed for the margin Perceptron. To avoid this, Li [2000] develops a scheme that appears to work well where the voting is done on a tail of the sequence of hypotheses which is chosen adaptively [see also Dekel and Singer, 2005, for more recent work].

Consider the noise tolerance guarantees for the unrealizable case. Ideally, one would want to find a hypothesis whose error rate is only a small additive factor away from the error rate of the best hypothesis in the class of hypotheses under consideration. This is captured by the agnostic PAC learning model [Kearns et al., 1994]. It may be worth pointing out here that, although we have relative loss bounds for several variants and these can be translated to some error bounds in the batch setting, the bounds are not sufficiently strong to provide significant guarantees in the agnostic PAC model. Hence this remains a major open problem to be solved.

In light of the discussion above, several interesting questions arise from our experimental results. The first is whether the more sophisticated versions of margin Perceptron add significant performance improvements. In particular it would be useful to know what normalization scheme is useful and in what contexts in order that they can be clearly applied. We have raised parameter tuning as an important issue in terms of run time and the self tuning capacity of ALMA and related algorithms seems promising as an effective solution. Given the failure of the simple longest survivor it would be useful to evaluate the more robust versions of [Gallant, 1990, Cesa-Bianchi et al., 2004]. Notice, however that these methods have a cost in training time. Alternatively, one could further investigate the tail variants of the voting scheme [Li, 2000] or the "averaging" version of voting [Freund and Schapire, 1999, Gentile, 2001], explaining when they work with different variants and why. Finally, as mentioned above, to our knowledge there is no theoretical explanation to the fact that Perceptron with Margin performs better than the basic Perceptron in the presence of noise. Resolving this is an important problem.

Chapter 4

A New Kernel for Learning from Hypergraphs

In this chapter we investigate kernels for learning from graphs and hypergraphs. This is an example of learning from structured data, where the natural structure of the data cannot be trivially captured by explicit features. In Chapter 3, we explored the performance of several kernel methods in a general setting. Here we develop kernels that make it possible for kernel methods to achieve excellent performance when the input is a hypergraph.

There is significant recent work in the area of learning from graphs [Kramer and De Raedt, 2001, Deshpande et al., 2003, Gärtner et al., 2003, Fröhlich et al., 2005]. A prime application of this setting is learning to classify molecules. Here each molecule is a separate example labeled according to some property (e.g., carcinogenicity) and one would like to predict the labels of new examples. The atom-bond structure of the molecule is typically used as the underlying graph structure of the example, and the nodes and edges of the graph are annotated with atom and bond types.

Learning from graphs is a special case of a problem commonly studied in Inductive Logic Programming (ILP) under the name of Learning from Interpretations [De Raedt and Dzeroski, 1994]. Here each example is an interpretation from

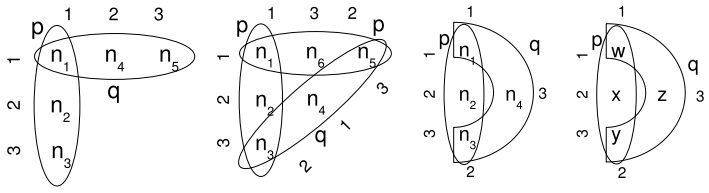

Figure 4.1: From left to right: H_1,H_2,H_3, and R. The letters are the hyperedge labels. The numbers represent each node's position within a hyperedge.

logic programming, which can be seen as a *labeled ordered hypergraph*. For example, the hypergraph H_1 with nodes $\{n_1, n_2, n_3, n_4, n_5\}$, hyperedge (n_1, n_2, n_3) labeled p, and hyperedge (n_1, n_3, n_4) labeled q can be compactly described as $H_1 = \{p(n_1, n_2, n_3), q(n_1, n_4, n_5)\}$. This generalizes the usual notion of a directed graph, in that edges have more than two endpoints and the order of nodes is important. A graphical representation of H_1 is given in Figure 4.1. Similarly hypergraphs $H_2 = \{p(n_1, n_2, n_3), p(n_1, n_5, n_6), q(n_4, n_3, n_5)\}$, and $H_3 = \{p(n_1, n_2, n_3), q(n_1, n_3, n_4)\}$ could be different examples in our problem domain. Typically ILP algorithms learn hypotheses represented as sets of first order logic rules and these are used to classify the interpretations [Muggleton, 1995, Quinlan, 1990, Arias et al., 2007]. For example, the rule $R = [\exists w, x, y, z, \quad p(w, x, y)q(w, y, z) \rightarrow Positive]$ classifies H_3 as positive and H_1 and H_2 as negative. As Figure 4.1 illustrates, a rule can be seen as a "hypergraph pattern" and its coverage of examples corresponds to hypergraph homomorphism. The search involved in ILP rule learning is complex and the matching problem, that is, checking whether a rule covers an example, is computationally hard. As a result such systems are typically slow and not easy to apply for large datasets.

The use of kernel methods over discrete structures, and in our case ordered hypergraphs, offers an attractive alternative. Recall from Section 2.1 that a kernel function calculates an inner product over some implicit feature space, and typically one uses a linear threshold classifier, such as Perceptron or SVM (see Section 2.2), to classify examples. A natural goal would be to capture each first-order logic rule as

one feature in order that the linear threshold function can combine the predictions of different rules. Notice that each rule corresponds to a potential sub-structure of the hypergraph. Therefore features in the implicit space correspond directly to substructures. Indeed, variants of this idea have already been studied for the special case of graphs, and are known as *graph kernels* [Gärtner et al., 2003, Kashima et al., 2003].

In this chapter we introduce a new kernel for ordered hypergraphs. To our knowledge this is the first kernel that applies to the general case of learning directly from interpretations, i.e., hypergraphs. We therefore benefit from the data setup of ILP and can use the kernel wherever learning from interpretations [De Raedt and Dzeroski, 1994] can be used, easily supporting relational data and background knowledge. The kernel generalizes graph kernels in that its features are based on walks in the hypergraph. The walk-based feature space of our kernel captures a large set of potential rules, so that our hypotheses can be seen as a weighted vote of ILP rules. The analogy to ILP breaks on two issues. First, as we discuss later in the chapter, not all ILP rules are expressible by walks, hence there is some trade-off when using the kernel as opposed to an ILP system. Second, while the kernel naturally takes input that is encoded as in ILP, the output of a kernel method using the kernel will not produce explicit rules that are useful in some applications. Despite these differences we show that the kernel does produce good results in terms of accuracy and is therefore useful when explicit rules are not a requirement.

Our kernel differs in important ways from previous graph kernel constructions and induces a new graph kernel when the data happen to be (non-hyper-)graphs. First, our kernel can recognize larger sub-structures than other walk-based kernels using the same length walk. This is a direct result of the way in which we extend walks. Second, unlike previous walk-based methods we consider only finite-length walks and therefore do not need to "discount" long walks in the kernel in order to guarantee convergence. Third, we provide a dynamic programming algorithm to

calculate the kernel, leading to reasonable run times even on large datasets.

In the experimental section we use real and artificial data. We use the Perceptron Algorithm with Margins [Krauth and Mézard, 1987] that has been shown in Chapter 3 to be competitive with Support Vector Machines in terms of accuracy but has a lower runtime. For real data, we choose several challenging chemical datasets: the estrogen binding dataset [Fang et al., 2001, Blair et al., 2000, Branham et al., 2002], the predictive toxicology challenge [Helma et al., 2001], and the HIV dataset from NCI/NIH.[1]

Our results demonstrate that the kernel outperforms existing ILP methods, and leads to performance comparable to other graph kernels when used on graph data. In investigating previous work with ILP and graph kernels, we found that ILP papers typically use a simple graph representation (we call this encoding 1 below) whereas graph kernel papers typically use a richer representation (we call this encoding 3 below). Therefore when comparing to ILP methods, we also compare across these representations. The results show that encoding 3 improves the performance of the hypergraph kernel significantly, but does not help the performance of the ILP methods tested.

We give insight into how the parameters of the kernel (walk length and discounting, explained below) affect performance. We present evidence that discounting walks as a function of their length, which is common in the literature, does not lead to a significant difference in performance. On the other hand the walk length is important and must be suitably chosen for every application.

We show that hyperedges lead to an improvement in performance using both artificial and natural data. The artificial data highlight the advantage of using the hypergraph representation directly in contrast with a translation capturing the same data using a graph. The experiments show that transforming the hypergraph into a graph has a direct negative impact on performance. We also show, using

[1] `http://dtp.nci.nih.gov/docs/aids/aids_data.html`

the Mutagenesis dataset, that incorporating background knowledge in the form of hyperedges (that exist for this dataset [Srinivasan et al., 1996]) indeed leads to performance improvements.

To summarize, the main contributions of this chapter is a new kernel suitable for the general case of learning from interpretations. Experimental results show that the kernel is effective both in terms of run time and in terms of classification performance. The experiments also highlight the crucial role of data encoding and identify an encoding that seems particularly suited to chemical applications. Finally we show that using our kernel and working directly with the hypergraph data can boost classification performance when compared to using a graph kernel on transformed hypergraph data.

The rest of the chapter is organized as follows. Section 4.1 gives some basic definitions and notation. In Section 4.2 we define our kernel and discuss potential variants to the basic kernel. This is followed by a more in-depth discussion of related work in Section 4.3. The experimental results are given in Section 4.4.

4.1 Definitions and Notation

Definition 4.1.1. A *labeled directed graph*, $G = (V, E)$, is a set of nodes V, and a set of edges $E \subseteq V \times V$. Every edge and every vertex is annotated with a label from a fixed set of labels L.

Hypergraphs are normally defined as a generalization of undirected graphs but here we define them as a generalization of directed graphs as follows.

Definition 4.1.2. A *labeled ordered hypergraph*, $G = (V, E)$ has a set of vertices V and a set of edges E. Each edge $e \in E$ is a tuple of vertices, $(v_1, \ldots, v_n)$ where $n \geq 1$ is the *arity* of the edge. Every edge is labeled with a label from L. We do not label vertices; instead we can use edges of arity 1. Furthermore, we allow parallel edges, that is, the same tuple can exist in E multiple times, but with different labels.

Example of ordered hypergraphs are given in the previous section (see Figure 4.1).

Definition 4.1.3. A *walk* in a directed graph is a sequence of vertices and edges $v_1, e_1, v_2, \ldots, e_{n-1}, v_n$ such that $e_i = (v_i, v_{i+1}) \in E$. We define a walk in an ordered hypergraph as a sequence of hyperedges where every two consecutive edges have at least one node in common, and no consecutive edges are identical. We represent a walk by explicitly specifying indices of the nodes shared by two edges. In particular, we use a string $P = p_1 i_1 j_1 p_2 i_2 j_2 p_3 \ldots i_{n-1} j_{n-1} p_n$ where $p_l \in E$, every i_k represents the exit position of p_k and every j_k represents the entry position of p_{k+1}. For example, $P = p(n_1, n_2, n_3), 1, 1, p(n_1, n_5, n_6), 2, 3, q(n_4, n_3, n_5)$ represents a walk in H_2. Notice that the ordering of edge arguments is important because we track entry and exit positions for the nodes.

Definition 4.1.4. A *walk type* is specified by a string $w = r_1 i_1 j_1 r_2 i_2 j_2 r_3 \ldots i_{n-1} j_{n-1} r_n$ where r_l is an edge label. For example, the walk type of P given above is $w = p, 1, 1, p, 2, 3, q$. Thus a walk identifies individual edges, whereas a walk type generalizes the walk and only includes edge labels. Although every walk in a hypergraph is unique, walk types are not; two walks are of the same type iff the strings representing them are identical.

In the following we define a kernel whose features correspond to walk types. Notice that walk types are less expressive than rules in that they make fewer distinctions. In our example, walk type $w = p, 1, 1, q$ captures hypergraphs H_1 and H_3, walk type $w = p, 3, 2, q$ captures H_2 and H_3, but there is no walk type equivalent to the rule R from the introduction which captures H_3 but neither of H_1, H_2.

We need the following additional notation. For edge p_i in hypergraph G, $\text{rel}(p_i)$ denotes its label, and p_i^j denotes the vertex at position j in the edge. The string $x.y$ denotes the string resulting from concatenating string y to x. Finally, for edge p_i in hypergraph G and walk type w we define $\#(G, p_i, w)$ to be the number of walks of type w starting at edge p_i in G. Note that if $\#(G, p_i, w) > 0$ then w begins with $\text{rel}(p_i)$.

4.2 A Hypergraph Kernel

We first define a kernel operating on hypergraphs that are "rooted" at particular edges. We then define a kernel operating on hypergraphs in general, and finally discuss variants and extensions of these kernels.

4.2.1 A Kernel Rooted at Specific Edges

The following kernel $K_n()$ operates on pairs of hypergraphs and edges so it should be written as $K_n((G_1, p_1), (G_2, p_2))$ but to simplify the presentation we omit G_1 and G_2 from the equations. We also omit the fact that p_1 and p_1' are always in G_1 and p_2 and p_2' are always in G_2.

Definition 4.2.1. The kernel $K_n()$ is defined recursively as follows:

$$\begin{aligned}
K_1(p_1, p_2) &= 1 \quad \text{iff } \mathrm{rel}(p_1) = \mathrm{rel}(p_2) \\
K_1(p_1, p_2) &= 0 \quad \text{otherwise} \\
K_n(p_1, p_2) &= K_1(p_1, p_2) \sum_{i=1}^{k} \sum_{j=1}^{\substack{\max \\ \text{arity}}} \sum_{p_1' : p_1^i = p_1'^j} \sum_{p_2' : p_2^i = p_2'^j} K_{n-1}(p_1', p_2').
\end{aligned}$$

where in the sum above $p_1' \neq p_1$ and $p_2' \neq p_2$, k is the arity of p_1 and *max arity* refers to the maximum arity of any edge in G_1 or G_2. The expression $p_1' : p_1^i = p_1'^j$ means "an edge p_1' such that the ith vertex of p_1 is the same as the jth vertex of p_1'."

The definition immediately gives a dynamic programming algorithm to calculate the kernel by incrementally calculating $K_\ell()$ for $\ell = 2$ to any desired n. It may seem that we need $(\text{max arity})^2 |E|^2$ steps to calculate each individual kernel value. One can do much better, however, for sparse hypergraphs (where the number of neighbors is small) by appropriately encoding the neighbors of each node. We next prove that $K_n()$ is indeed a kernel by showing explicitly that it is an inner product for a feature

space indexed by all walk types, and where the feature indexed by w takes value $\#(G, p, w)$.

Theorem 4.2.2.

$$K_n(p_1, p_2) = \sum_{\substack{\text{walk type w} \\ \text{of length n}}} \#(G_1, p_1, w) \cdot \#(G_2, p_2, w).$$

Proof. By induction on n. Base case for $n = 1$. Note that walk types of length 1 are simply edge labels. Hence, we need to show that

$$K_1(p_1, p_2) = \sum_{\substack{\text{edge} \\ \text{label } w}} \#(G_1, p_1, w) \cdot \#(G_2, p_2, w).$$

The sum is zero unless p_1 and p_2 have the same edge label and w is that label, in which case the sum is 1. Assume the claim is true for $n = \ell - 1$. Then

$$K_\ell(p_1, p_2) = K_1(p_1, p_2) \sum_{i=1}^{k} \sum_{j=1}^{\substack{\max \\ \text{arity}}} \sum_{p_1' : p_1^i = p_1'^j} \sum_{p_2' : p_2^i = p_2'^j} K_{\ell-1}(p_1', p_2') \tag{4.1}$$

$$= K_1(p_1, p_2) \sum_{i=1}^{k} \sum_{j=1}^{\substack{\max \\ \text{arity}}} \sum_{p_1' : p_1^i = p_1'^j} \sum_{p_2' : p_2^i = p_2'^j} \sum_{\substack{\text{w of} \\ \text{length } \ell-1}} \#(G_1, p_1', w) \cdot \#(G_2, p_2', w) \tag{4.2}$$

$$= K_1(p_1, p_2) \sum_{i=1}^{k} \sum_{j=1}^{\substack{\max \\ \text{arity}}} \sum_{\substack{\text{w of} \\ \text{length } \ell-1}} \left(\sum_{\substack{p_1': \\ p_1^i = p_1'^j}} \#(G_1, p_1', w) \right) \left(\sum_{\substack{p_2': \\ p_2^i = p_2'^j}} \#(G_2, p_2', w) \right) \tag{4.3}$$

$$= K_1(p_1, p_2) \sum_{i=1}^{k} \sum_{j=1}^{\substack{\max \\ \text{arity}}} \tag{4.4}$$

$$\sum_{\substack{w \text{ of} \\ \text{length } \ell-1}} \#(G_1, p_1, \text{rel}(p_1).i.j.w) \cdot \#(G_2, p_2, \text{rel}(p_2).i.j.w).$$

Where the transition from (4.1)-(4.2) is by the induction hypothesis; (4.2)-(4.3) by reordering summations; (4.3)-(4.4) follows by definition of $\#(.,.,.)$.

Consider a string w' representing a walk type of length $\ell - 1$. By adding $\text{rel}(p_1).i_1.j_1$ to the string we create a new walk type w of length ℓ. Now if we consider an arbitrary walk type w of length ℓ, if w does not start with $\text{rel}(p_1)$ then $\#(G_1, p_1, w)$ is 0. We can therefore replace (4.4) above with

$$K_1(p_1, p_2) \sum_{\substack{\text{w of} \\ \text{length } \ell}} \#(G_1, p_1, w) \cdot \#(G_2, p_2, w).$$

Finally, if p_1, p_2 do not have the same edge label then for every w at least one of $\#(G_1, p_1, w)$, $\#(G_2, p_2, w)$ is 0 and therefore the sum is 0 thus we can omit K_1 from the expression. Similarly, if p_1, p_2 do have the same relation symbol $K_1 = 1$ and we can omit K_1. Hence, as required, we have

$$K_\ell(p_1, p_2) = \sum_{\substack{w \text{ of} \\ \text{length } \ell}} \#(G_1, p_1, w) \cdot \#(G_2, p_2, w). \qquad \square$$

4.2.2 A Gappy Kernel for Hypergraphs

A modification to the hypergraph kernel allows us to compute the number of matching walks up to a certain length, and with a certain number of gaps.

Definition 4.2.3 (Gap)**.** Following Gärtner et al. [2003], we extend the alphabet of walk types to include the symbol '###'. The symbol takes the place of an entire segment of a walk (entry position, hyperedge label, and exit position). We say that a walk type contains g gaps if the walk type uses the ### symbol g times. For instance the walk $1e_1(n_1, n_2, n_3)12e_2(n_4, n_1, n_5)$, where $\text{rel}(e_1) = p$ and $\text{rel}(e_2) = r$ does not match walk type $1p1\mathbf{2q3}$ (since $\text{rel}(e_2) \neq q$) but matches walk type $1p12r3$, walk types $1p1$### and ###$2r3$ (that have one gap), and walk type ###### (that has two gaps).

The definition of a gap is motivated by the interpretation of a walk as a conjunc-

tion of predicates. Using this definition, a gap is equivalent to a single mismatched term in a conjunction of predicates. Because we are grouping walk types into segments of entry position, relation type, and exit position, we must think about what the first entry position in a walk type means. It is simply the starting node of the walk. Hence "1p2..." means the walk type starts in position 1 of an edge labeled p, and exits that edge through position 2.

The following notation is needed for the definition of the kernel. If a walk in the hypergraph takes a step from p to q via node n, we call n's position in p $exit(p,q)$, and n's position in q $entry(p,q)$. As hyperedges can have more than one node (or occurrence of a node) in common, to simplify notation we define

$$\sum_{\substack{p'_1 \\ \text{incident}}} \equiv \sum_{p'_1} \sum_{exit(p_1,p'_1)} \sum_{entry(p_1,p'_1)} .$$

Finally we use the previous kernel with $n = 1$:

$$K_1(p_1, p_2) = \text{as given in Definition 4.2.1.}$$

The kernel is defined as follows:

$$K_{n,g}(p_1, p_2, I, J) = 0, \forall g < 0 \tag{4.5}$$

$$K_{n,g}(p_1, p_2, I, J) = K_{n,n}(p_1, p_2, I, J), \forall g > n \tag{4.6}$$

$$K_{1,0}(p_1, p_2, I, J) = \delta(I, J) K_1(p_1, p_2) arity(p_1) \tag{4.7}$$

$$K_{1,1}(p_1, p_2, I, J) = arity(p_1) * arity(p_2) + K_{1,0}(p_1, p_2, I, J) \tag{4.8}$$

$$\begin{aligned} K_{n,g}(p_1, p_2, I, J) = \delta(I, J) K_1(p_1, p_2) \sum_{i=1}^{k} \sum_{p'_1 : p_1^i = p_1'^{j_1}} \sum_{p'_2 : p_2^i = p_2'^{j_2}} K_{n-1,g}(p'_1, p'_2, j_1, j_2) \\ + \sum_{\substack{p'_1 \\ \text{incident}}} \sum_{\substack{p'_2 \\ \text{incident}}} K_{n-1,g-1}(p'_1, p'_2, entry(p_1, p'_1), entry(p_2, p'_2)). \end{aligned} \tag{4.9}$$

The first line in (4.9) sums over all exit positions i in p_1 and p_2. For each such position we consider all potential entry positions j_1, j_2 in p_1', p_2' respectively. Hence the exit position from p_1, p_2 is the same and fixed to i; together with $\delta(I, J)K_1(p_1, p_2)$ we get a complete coordinated triple in a walk type. On the other hand the entry positions for the next triple, j_1, j_2, are not forced to be the same. The second line in (4.9) sums over all exit and entry positions and does not coordinate these for p_1, p_2. As we show next this yields a kernel with features indexed by walk types that include gaps:

Theorem 4.2.4. *Let*

$$\begin{aligned} \mathit{gaps}(w) &= \text{number of occurrences of \#\#\# in } w, \\ \#(G, p, i, w) &= \text{number of walks of type } w \text{ in } G \text{ starting at position } i \text{ of hyperedge } p, \end{aligned}$$

then

$$K_{n,g}(p_1, p_2, I, J) = \sum_{\substack{w | \mathit{gaps}(w) \leq g, \\ \mathit{length}(w) = n}} \#(G_1, p_1, I, w) \#(G_2, p_2, J, w) \tag{4.10}$$

Proof. We show by induction on n that for all $g \geq 0$, $K_{n,g}$ satisfies (4.10). The base case is for $n = 1$. To see that (4.7) is correct, note that the number of length one walk types starting at p_1 at position I is equal to the number of possible completions of the length one walk type beginning $I.\mathrm{rel}(p_1)$, i.e., any exit position in p_1. If p_1 and p_2 are of the same type and $I = J$, then for every possible position i in p_1 (or p_2 as they are the same type) there is a walk type $I.\mathrm{rel}(p_1).i$ that occurs once in G_1 starting at position I of p_1 and once in G_2 starting at position I of p_2. Now to see that (4.8) is correct, observe that for any $p_1 \in G_1$ and I, the length one walk type ### matches the walk Ip_1i for any possible i, hence the total number of times that ### matches in G_1 starting at p_1 in position I is the arity of p_1, and similarly for p_2. We add $K_{1,0}$ to the product of the arities to account for length one walk types

that are not ###. Because of (4.6), by showing (4.10) is true for $n = 1, g = 0$ and $n = 1, g = 1$, we have shown it is true for $n = 1$ and any $g \geq 0$.

Assume that for $\ell = n - 1$, $K_{\ell,g}$ satisfies (4.10) for all $g \geq 0$. For the inductive step consider first the case with $\ell = n$ and $g \geq 1$:

$$
\begin{aligned}
&K_{n,g}(p_1, p_2, I, J) = \\
&\quad \delta(I,J)K_1(p_1,p_2)\sum_{i=1}^{k}\sum_{p_1':p_1^i=p_1'^{j_1}}\sum_{p_2':p_2^i=p_2'^{j_2}} K_{n-1,g}(p_1', p_2', j_1, j_2) \\
&\quad + \sum_{\substack{p_1' \\ \text{incident}}}\sum_{\substack{p_2' \\ \text{incident}}} K_{n-1,g-1}(p_1', p_2', \mathit{entry}(p_1, p_1'), \mathit{entry}(p_2, p_2')) \qquad (4.11) \\
&= \delta(I,J)K_1(p_1,p_2)\sum_{i=1}^{k}\sum_{p_1':p_1^i=p_1'^{j_1}}\sum_{p_2':p_2^i=p_2'^{j_2}}\sum_{\substack{w|gaps(w)\leq g,\\ length(w)=n-1}} \#(G_1, p_1', w, j_1)\#(G_2, p_2', w, j_2) \\
&\quad + \sum_{\substack{p_1' \\ \text{incident}}}\sum_{\substack{p_2' \\ \text{incident}}}\sum_{\substack{w|gaps(w)\leq g-1,\\ length(w)=n-1}} \#(G_1, p_1', w, \mathit{entry}(p_1, p_1'))\#(G_2, p_2', w, \mathit{entry}(p_2, p_2')) \\
&\qquad\qquad (4.12) \\
&= \delta(I,J)K_1(p_1,p_2)\sum_{\substack{w|gaps(w)\leq g,\\ length(w)=n-1}}\sum_{i=1}^{k}\left(\sum_{p_1':p_1^i=p_1'^{j_1}} \#(G_1, p_1', w, j_1)\right)\left(\sum_{p_2':p_2^i=p_2'^{j_2}} \#(G_2, p_2', w, j_2)\right) \\
&\quad + \sum_{\substack{w|gaps(w)\leq g-1,\\ length(w)=n-1}}\left(\sum_{\substack{p_1' \\ \text{incident}}} \#(G_1, p_1', w, \mathit{entry}(p_1, p_1'))\right)\left(\sum_{\substack{p_2' \\ \text{incident}}} \#(G_2, p_2', w, \mathit{entry}(p_2, p_2'))\right) \\
&\qquad\qquad (4.13) \\
&= \delta(I,J)K_1(p_1,p_2)\sum_{\substack{w|gaps(w)\leq g,\\ length(w)=n-1}}\sum_{i=1}^{k}\#(G_1, p_1, I.\mathrm{rel}(p_1).i.w, I)\#(G_2, p_2, J.\mathrm{rel}(p_2).i.w, J) \\
&\quad + \sum_{\substack{w|gaps(w)\leq g-1,\\ length(w)=n-1}} \#(G_1, p_1, \#\#\#.w, I)\#(G_2, p_2, \#\#\#.w, J) \qquad (4.14)
\end{aligned}
$$

$$= \sum_{\substack{w|gaps(w)\leq g,\\ length(w)=n\\ w[0]\neq \#\#\#}} \#(G_1, p_1, w, I)\#(G_2, p_2, w, J) + \sum_{\substack{w|gaps(w)\leq g,\\ length(w)=n\\ w[0]=\#\#\#}} \#(G_1, p_1, w, I)\#(G_2, p_2, w, J) \tag{4.15}$$

For the transition (4.11)-(4.12) we apply the inductive hypothesis and (4.13) is obtained by rearranging the summations.

In the first line of (4.14) we extend each $n-1$ length walk type w by pre-pending $I.\text{rel}(p_1).i$ for G_1 and $J.\text{rel}(p_2).i$ for G_2. The first triple of the resulting walk type, $I.\text{rel}(p_1).i.w$ in G_1, is not a gap. Therefore, this guarantees that every incidence p_1' (and its entry position) via p_1's i'th node is counted exactly once for each walk continuation, just like in (4.13).

Next consider the transition to the first term in (4.15). In (4.14) if the two extensions to w are not of the same type, the expression $\delta(I, J)K_1(p_1, p_2) = 0$. In the first term of (4.15) we sum over all walk types w of length n but require that the first leg of w is not $\#\#\#$. We can omit $\delta(I, J)K_1(p_1, p_2)$ because we specify that the first leg of the walk is not a gap; hence if the walk type w does not begin with $I.\text{rel}(p_1)$ then $\#(G_1, p_1, w, I) = 0$ and similarly for G_2. Their identity is forced by the shared value in w. Similarly the joint exit position i is forced because the expressions for the two graphs share the exit position value from w.

In the second line of (4.14), we express the extension of walk type w of length $n-1$ with $\#\#\#.w$. In this summation we are fixing the first leg of the length n walk to be the gap symbol which will match the first leg of any walk. Hence we sum over all possible first legs starting at p_1 and p_2, that is, all incident edges. Since the walk type starts with a gap, we decrease the number of "available" gaps left in the rest of the walk type w of length $n-1$. This is equivalent to the second term in (4.15), in which we sum over all walk types of length n with up to g gaps such that the first leg of w is a gap.

As the two sums in (4.15) are over complementary walk types whose union is all

walk types, we can replace (4.15) with

$$\sum_{\substack{w|gaps(w)\leq g,\\ length(w)=n}} \#(G_1, p_1, w, I)\#(G_2, p_2, w, J).$$

Finally consider the case where $\ell = n$ and $g = 0$. Notice that the second lines of (4.11) - (4.14) and the second term of (4.15) are equal to 0 by (4.5). Hence the first term in (4.15) is summing exactly over walk types with no gaps. □

4.2.3 A General Kernel for Hypergraphs

We next define another kernel $K'_n()$ that extends the kernel from Definition 4.2.1, operating over entire hypergraphs:

Definition 4.2.5.

$$K'_n(G_1, G_2) = \sum_{p_1 \in E_1} \sum_{p_2 \in E_2} K_n(p_1, p_2) \tag{4.16}$$

One can show that (4.16) is a kernel by re-writing it as

$$\sum_{\substack{w \text{ of} \\ \text{length } \ell}} \left(\sum_{p_1 \in G_1} \#(G_1, p_1, w) \right) \left(\sum_{p_2 \in G_2} \#(G_2, p_2, w) \right)$$

The first inner sum is the total number of walks of type w in G_1 and likewise the second inner sum for G_2. In this representation it is easy to see that every element of the outer sum is the total number of walks of type w in G_1 times the total number of walks of type w in G_2.

4.2.4 Discounting and Normalizing

A general idea in string and graph kernels, where we consider an infinite number of features, is to discount the contribution of longer walks. Indeed, this discount factor is necessary in order to achieve convergence when summing contributions of

all possible walks on length 1 to ∞ [Gärtner et al., 2003]. Another standard variant is to normalize the norm of the examples in the kernel feature space.

These are implemented using our kernel as follows.

Definition 4.2.6 (Discounted Kernel)**.**

$$K_n^{\mathrm{D}}(G_1, G_2) = \sum_{i=1}^{n} \gamma^i K_i'(G_1, G_2) \tag{4.17}$$

$$K_n'^{\mathrm{D}}(G_1, G_2) = \frac{K_n^{\mathrm{D}}(G_1, G_2)}{\sqrt{K_n^{\mathrm{D}}(G_1, G_1) K_n^{\mathrm{D}}(G_2, G_2)}} \tag{4.18}$$

It follows from standard properties that (4.17) and (4.18) are kernels [Cristianini and Shawe-Taylor, 2000]. Notice that with $\gamma < 1$ we get discounting. However, for our kernel γ is not restricted in this way. In fact, we can emphasize the contribution of longer walks by using $\gamma > 1$. This is intuitively appealing because longer walks give more informative matches between the graphs.

4.3 Discussion and Related Work

Our work is closely related to graph kernels as well as several approaches in ILP and relational data mining. In the following, we discuss the relations to these, placing our work in context.

4.3.1 Kernels and Similarity functions for Graphs and Hypergraphs

The inspiration for our work comes largely from previous work on graph kernels [e.g., Gärtner et al., 2003, Kashima et al., 2003, Horváth et al., 2004] and the potential to extend it to be applicable to the entire range of problems addressed by ILP. One important basic result for graph kernels shows that it is computationally hard to calculate a kernel whose feature space corresponds to all possible subgraphs unique up to isomorphism, where each feature is binary-valued according to the existence of that particular subgraph [Gärtner et al., 2003]. Therefore one must compromise

and use a less expressive family of subgraphs as features. On the positive side, recent work on graph kernels uses various properties to create a similarity measure between two graphs: the number of labeled walks shared between graphs [Gärtner et al., 2003]; the probability of a random walk in both graphs [Kashima et al., 2003]; the number of a certain type of sub-structure present in both graphs [Kramer and De Raedt, 2001, Deshpande et al., 2003, Horváth et al., 2004, Ralaivola et al., 2005, Tsuda and Kudo, 2006].

From the graph kernel perspective, our work is most closely related to the walk-based kernels [Gärtner et al., 2003, Kashima et al., 2003]. The kernel of Gärtner et al. [2003] computes the number of walks of any length (with identical label sequences) that the two input graphs share. Kashima et al. [2003] present a *marginalized graph kernel* that computes the similarity of two graphs based on the probability that a random walk occurs in both graphs. Both kernels use walks of arbitrary length and sum their contributions thus both have some form of discounting to guarantee that the kernel value is not infinite. Both kernels are also expensive to compute; the kernel by Gärtner et al. [2003] must invert or diagonalize a matrix that is quadratic in the number of vertices of the direct product graph, and the kernel in Kashima et al. [2003] must solve a system of linear equations described by a matrix quadratic in the number of vertices in the direct product graph. Vishwanathan et al. [2006] present an alternative method for calculating the graph kernels in Kashima et al. [2003] and Gärtner et al. [2003] that is cubic in the size of the graph and show experimental evidence that the actual speedup is significant.

Although our kernel is also based on walks there are several important differences. First, we focus on a fixed finite length of walks. This helps avoid unintuitive discounting of the weights of long walks. We are not aware of a method for capturing kernels based on arbitrary length walks with hypergraphs. Second, we provide an efficient dynamic programming algorithm to calculate the kernel. This extends previous dynamic programming approaches to kernels for strings and

trees [e.g., Collins and Duffy, 2002]. Third, there are differences in the feature space that make our kernel more compact than other walk-based kernels. This is illustrated by the following example: consider a pattern capturing a star graph with one center v_0 and 4 outer nodes where each edges has a different label, i.e., $\{l_1(v_0, v_1), l_2(v_0, v_2), l_3(v_0, v_3), l_4(v_0, v_4)\}$. To capture this pattern with a walk one must consider an edge in each direction and go back and forth on each edge (except the first and last) thus we need a walk of length 6. In our case, because we match positions but do not consider the directionality of the edge, this can be captured by a walk of length 4 of type $l_1, 1, 1, l_2, 1, 1, l_3, 1, 1, l_4$ (hence we enter and exit the edge in the same node). Thus our kernel can be more expressive, capturing complex sub-graphs using shorter walks.

The only other work to give a kernel for multi-relational data we are aware of is the kernel for relational algebra of Woznica et al. [2005]. This kernel can be seen to take walks over the relations restricted by the notion of keys in the relational schema and some tree representation of the data. Our work differs in general applicability to interpretations, but more importantly in that our dynamic programming algorithm provides a polynomial time calculation of the kernel for arbitrary depths. We also explicitly provide the feature space of the kernel corresponding to walks (and rules) which makes for simple semantics.

Our kernel is also related to similarity functions used in relational instance-based learning (RIBL) [Horváth et al., 2001]. In particular, similarity in RIBL is calculated recursively where the similarity between atoms is defined through the similarity of each of the terms in its arguments, and the similarity of terms or objects can be defined through the similarity of the atoms they appear in. Thus a similar "walk based" similarity is defined. The emphasis in this work is on defining a similarity function that makes sense intuitively and works well in different contexts, whereas our focus is on a construction that yields an inner product kernel that can therefore be used with kernel methods.

4.3.2 Explicit Propositionalization for ILP

Relational representations are the natural data representations for ILP solvers. Two benefits of ILP solvers are that they naturally work in an expressive first-order data representation, and that they produce a set of rules that explain the hypothesis in a human-readable format. SVM and other discriminative classifiers (e.g., kNN, Perceptron) are generally computationally faster than ILP solvers while giving state of the art classification performance, yet they do not naturally handle relational (i.e., non-propositional) data nor produce rules. Our kernel bridges some of this gap — it enables discriminative classifiers to handle ILP data, including ease of incorporating relational background knowledge[2] but the final hypothesis is not human-readable.

Explicit propositionalization has been combined with SVM classifiers in recent work [e.g., Muggleton et al., 2005, Landwehr et al., 2006]. In particular, SVILP [Muggleton et al., 2005] and kFOIL [Landwehr et al., 2006] run an ILP solver to generate rules and use these rules as features. SVILP runs Progol [Muggleton, 1995] to statically generate rules in its search space, whereas kFOIL runs FOIL [Quinlan, 1990] in a wrapper based approach to dynamically select a small number of features for use with SVM. Cumby and Roth [2003] restrict the feature space using a *feature description language* and then explicitly generate the features. Our kernel can be seen as an implicit static propositionalization into the space syntactically defined using walks. Any explicit propositionalization in the hypergraph kernel's feature space would be computationally intractable in general.

It is important to clarify that walks in the hypergraph are not as expressive as rules in ILP. The example in Section 4.1 illustrates that we do not account for multiple shared nodes between adjacent edges on a walk, therefore a walk with multiple shared nodes will be represented in the features of more than one walk type. Designing kernels that do capture such complexity is an important open problem.

[2]See [Muggleton and De Raedt, 1994, De Raedt, 1997, Arias et al., 2007] for a discussion on the relationship between the normal ILP setting and learning from interpretations and how background knowledge is handled in each case.

Other approaches to propositionalization try to take advantage of specific sub-structures that may be important to the application, and are not directly comparable to the features of our kernel. Ralaivola et al. [2005] propose kernels that count fixed length common subtrees. In Horváth et al. [2004], a graph is decomposed into simple cycles and bridges connecting them and these correspond to the features in the kernel. Finally, some works do not restrict the format of the sub-structures corresponding to features. Instead informative sub-structures are computed automatically by identifying frequent subgraphs in the dataset similar to itemset data mining [Deshpande et al., 2003, Kramer and De Raedt, 2001].

Previous work [e.g., Cumby and Roth, 2003, Khardon et al., 2001] show that in certain cases explicit representation of a feature space is sometimes preferable to using a kernel function, even if the feature space is intractable to represent in general. This stems from the tradeoff between the "kernelization" of the classification algorithms (illustrated with examples in section 2.2), which incurs a significant performance penalty, and the explicit representation of a large feature space, which can be intractable. If the feature space is exponentially-sized with respect to the original data encoding but very sparse, for example, then it still may be possible to represent the data efficiently using a sparse-vector representation. We do not address optimizing run times in our experiments and hence we opt to use kernels throughout our work as they are tractable in all cases. Nevertheless, the analysis of whether kernels are the most efficient way to compute inner products in our experiments is a worthwhile endeavor for future work.

4.3.3 Translating Hypergraphs into Graphs

Due to the above restriction of our walks (linking one node at a time), one can translate the hypergraph into a quadratic size directed graph and capture similar walk-based information. This can be done by adding a new node for each hyperedge and connecting the new node to the nodes belonging to the hyperedge. We label

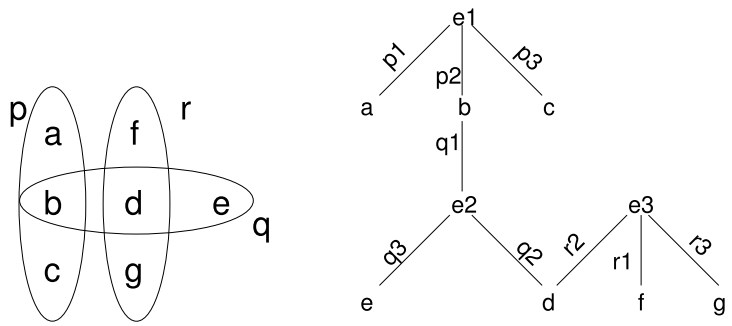

Figure 4.2: Left: target as a hypergraph. Right: target converted to a graph.

each edge with the original hyperedge label as well as the position of the node in the original hyperedge. For example, the hyperedge $p(a, b, c)$ becomes

$$p1(a, e1), p2(b, e1), p3(c, e1).$$

Figure 4.2 shows the hypergraph

$$\{p(a, b, c)q(b, d, e)r(f, d, g)\}$$

and its representation as a graph. Similarly, a single conjunction such as

$$p(a, b, c), q(b, d, e), r(f, d, g)$$

can be captured by a walk in the graph

$$p2(b, e1)q1(b, e2)q2(d, e2)r2(d, e3).$$

Hence we can simulate the hypergraph kernel using the graph representation though perhaps at increased complexity as the graph is larger and the walks are longer. However, as our experiments below show, the hypergraph-to-graph translation results in significantly reduced performance.

Table 4.1: Datasets Used in Experiments

Dataset	Examples	# Atoms	Majority Class
NCTRER	232	7-44	0.60
MUTAG	188	15-41	0.67
PTC(MM)	336	2-106	0.62
PTC(MR)	344	2-106	0.56
PTC(FM)	349	2-106	0.59
PTC(FR)	351	2-106	0.66
NCI-HIV	41606	2-438	0.99

4.4 Experiments and Results

We perform several sets of experiments using artificial data as well as real-world datasets for molecule classification. The number of examples, number of hypergraph nodes in examples, and label distribution in the chemical datasets are given in Table 4.1.

In all of the experiments, we used the Perceptron with Margin [Krauth and Mézard, 1987] as the learning algorithm (see Chapter 3). As we show in Chapter 3, this algorithm gives similar accuracy to SVM with often reduced run time. The algorithm has two parameters, the number of iterations used for training and the margin value. We did not optimize the parameters. For margin we follow our findings from Chapter 3 and use a small relative margin setting of 0.1. For iterations we used two settings depending on dataset size. On the artificial datasets and small chemical datasets (Mutagenesis, NCTRER, and PTC) we trained for 20 iterations with a learning rate of 0.1, and on the larger dataset (NCI-HIV) we trained for 2 iterations with a learning rate of 0.5. We ran all experiments with 10-fold cross validation. In all our experiments we used the kernel from Equation (4.18) with $\gamma = 1$ (i.e., no discounting) except when explicitly stated otherwise.

In the first set of experiments we use the National Center for Toxicological Research Estrogen Receptor Binding (NCTRER) dataset [Fang et al., 2001, Blair et al., 2000, Branham et al., 2002] and we explore the effect of dataset encoding on

performance. This dataset has been recently used in a study comparing several ILP systems and is therefore appropriate to address our question.

The dataset contains chemicals that are labeled based on how well they bind to estrogen receptors. We consider molecules labeled "active strong," "active medium," and "active weak" to be positive, and molecules labeled "slight binder," and "inactive" to be negative.[3] Each molecule in the dataset is represented as a set of predicates. As in previous studies [e.g., Deshpande et al., 2003, Horváth et al., 2004], we eliminate hydrogen atoms because this reduces the size of examples and hydrogens are implicit in the reduced representation.

Previous work on graph kernels has demonstrated that enriching the edge labels can help performance [Gärtner, 2005, Mahé et al., 2004]. We explore three methods of encoding the datasets in this set of experiments. In the first encoding bonds and their types are represented by edges: $bondtype1(x, y)$, for example. Atom type is encoded using unary edges with the type as the relation name. As bonds are not directed, we store bond relations twice in this encoding, once with each vertex ordering, as there is not enough information in the edge labels to imply the order; for example, using edges $(v1, v2)$ and $(v2, v3)$, both labeled bondtype1, we find the walk type "bondtype1,2,1,bondtype1" to be present. Two edges from another graph, $(w2, w1)$ and $(w2, w3)$ should generate the same walk type, but they will not unless we duplicate edges. In the second encoding we eliminate the original "bondtype" labels and create $bondXtoY$ predicates, where X is the type of the first atom in the bond, and Y is the type of the second. To get a compact representation, we impose an ordering on the $bondXtoY$ relation such that X is lexicographically smaller than Y. This ensures that $bondXtoY$ and $bondYtoX$ will not appear in the dataset together, avoiding the need to duplicate links (except in the case of $bondXtoX$ where we do duplicate). The final encoding follows the work of Gärtner [2005] and Mahé et al. [2004] and encodes even more information about endpoints

[3] We used the version of the dataset entitled `NCTRER_v3b_232_10Apr2006.sdf`.

of bonds. In particular each argument of the bond predicates encodes the types of its atom and the types of all its neighbors. This technique is also similar to the *neighborhood kernel* discussed by Fröhlich et al. [2005], however we do not use as detailed information, and we use the immediate neighborhood only. As in the second encoding we lexicographically order the arguments and duplicate the bond only if the arguments are identical.

The following example illustrates the three encodings. Consider a star graph similar to the one given above with labels as follows $\{bond(v_0, v_1)$, $bond(v_0, v_2)$, $bond(v_0, v_3)$, $bond(v_0, v_4)$, $A(v_0)$, $B(v_1)$, $B(v_2)$, $C(v_3)$, $D(v_4)\}$ (in the chemical domain A, B, C, D would be element names). Under encoding 1, the bond structure of the graph would be encoded as

$$bond(v_1, v_0), bond(v_0, v_1), bond(v_2, v_0), bond(v_0, v_2),$$
$$bond(v_3, v_0), bond(v_0, v_3), bond(v_4, v_0), bond(v_0, v_4)$$

and we add the node types to this structure. Under encoding 2 we get:

$$bondAtoB(v_0, v_1), bondAtoB(v_0, v_2),$$
$$bondAtoC(v_0, v_3), bondAtoD(v_0, v_4).$$

In encoding 3 for each argument we represent the atom type first, followed by the node types of its neighbors. Below we separate the node from its neighbors with a vertical bar. This yields

$$bond\underline{A|BBCD}to\underline{B|A}(v_0, v_1),$$
$$bond\underline{A|BBCD}to\underline{B|A}(v_0, v_2),$$
$$bond\underline{A|BBCD}to\underline{C|A}(v_0, v_3),$$
$$bond\underline{A|BBCD}to\underline{D|A}(v_0, v_4).$$

Table 4.2: Accuracy on the NCTRER dataset varying walk length and encoding.

Length	Encoding 1	Encoding 2	Encoding 3
1	0.64 ± 0.08	0.67 ± 0.08	0.79 ± 0.08
2	0.64 ± 0.10	0.68 ± 0.10	0.83 ± 0.05
3	0.65 ± 0.10	0.67 ± 0.10	0.84 ± 0.07
4	0.66 ± 0.06	0.62 ± 0.06	0.85 ± 0.09
5	0.66 ± 0.10	0.64 ± 0.07	0.85 ± 0.06
6	0.64 ± 0.10	0.62 ± 0.10	0.83 ± 0.07
7	0.66 ± 0.08	0.62 ± 0.08	0.80 ± 0.08
8	0.66 ± 0.10	0.59 ± 0.10	0.75 ± 0.07
16	0.67 ± 0.09	0.66 ± 0.12	0.68 ± 0.11
nFOIL	0.78 ± 0.091		0.56 ± 0.10
kFOIL	0.776 ± 0.094		

Encoding 3 will result in fewer walk matches between graphs, since a match requires both that the vertices are of the same type, and that all of their neighbors are of the same type. As a result the transfer between graphs is smaller and less generalization is possible from one feature. Another important property of encoding 3 is that it makes for faster computation as fewer matches means fewer items added in the dynamic programming formula.

Results are given in Table 4.2. Notice that for encoding 1 the kernel does not perform well but with encoding 3 the results are significantly improved. Encoding 3 illustrates that a substructure more complex than walks may be needed. Furthermore, the results suggest that very long walks do not perform well. For comparison, we give the best result (achieved using nFOIL) reported by Landwehr et al. [2006], who compare state-of-the-art ILP solvers using encoding 1. For reference we also show the result for kFOIL (reported by Landwehr et al. [2006]) whose representation described above is closer to ours than nFOIL. The kernel method with encoding 3 yields better performance than nFOIL (and the other ILP systems in that study) who use encoding 1.

While this is an interesting result and observation on previous work in this area, it raises an important question: would encoding 3 help ILP solvers as well? It is

Table 4.3: Accuracy on NCTRER varying walk length and discount factor γ.

Length	$\gamma = 0.1$	0.5	0.8	1	2	10
2	0.82	0.81	0.82	0.83	0.81	0.82
3	0.82	0.84	0.84	0.84	0.84	0.83
4	0.82	0.85	0.84	0.85	0.84	0.85
5	0.83	0.84	0.84	0.85	0.85	0.85
6	0.82	0.84	0.82	0.83	0.82	0.81
16	0.84	0.69	0.68	0.68	0.67	0.68
32	0.84	0.64	0.62	0.62	0.62	0.62

not clear that there is a unique answer to all ILP solvers, and a full investigation of this question is beyond the scope of this chapter. However, we performed some exploratory experiments described below and these suggest that encoding 3 does not help ILP solvers. In particular, we ran nFOIL with the settings[4] reported by Landwehr et al. [2007] using 10-fold cross validation. As can be seen in Table 4.2, the performance of nFOIL on the NCTRER dataset decreased dramatically under encoding 3. This may be due to the introduction of very specific features, which can present problems in a naïve Bayesian model. We also ran CProgol [Muggleton, 1995] and LogAn-H [Arias et al., 2007] using both encoding 1 and encoding 3. The results were worse than those reported for nFOIL and there was no significant difference in performance between encoding 1 and encoding 3.

Overall, the results show that encoding 3 improves the performance of the hypergraph kernel significantly, but does not help the performance of the ILP methods tested. When using encoding 3 the hypergraph kernel outperforms ILP solvers in terms of accuracy on this dataset.

In the next set of experiments we explore the effect of the hypergraph kernel parameters by varying the discount factor (γ) and walk length on the NCTRER dataset. We use γ values of $\{0.1, 0.5, 0.9, 1, 2, 10\}$ and walk lengths of $\{2, 3, 4, 5, 6, 16, 32\}$. The results are given in Table 4.3. While some performance variation with dis-

[4]Beam search width of 5, 25 max clauses, and 10 max literals.

Table 4.4: Accuracy on PTC and area under ROC curve for NCI-HIV. "HG" is the hypergraph kernel.

Dataset	HG short	HG medium	HG long	OA	MG
PTC(FM)	0.56 ± 0.09	$0.64 \pm .10$	0.64 ± 0.10	$0.64 \pm .03$	$0.62 \pm .03$
PTC(FR)	0.62 ± 0.09	0.62 ± 0.06	$0.67 \pm .07$	$0.67 \pm .02$	$0.67 \pm .02$
PTC(MM)	0.62 ± 0.12	$0.64 \pm .07$	0.59 ± 0.08	$0.68 \pm .02$	$0.67 \pm .02$
PTC(MR)	$0.52 \pm .07$	0.58 ± 0.10	$0.60 \pm .07$	$0.63 \pm .02$	$0.58 \pm .01$
Dataset		HG (l=5)	CPK	GK	
NCI-HIV		0.94 ± 0.02	0.91 ± 0.01	$0.94 \pm .01$	

Table 4.5: Accuracy on NCTRER and Area under ROC for NCI-HIV in the Same Experimental Conditions.

Dataset	HG (l=5)	CPK (Perceptron)
NCI-HIV	0.94 ± 0.02	0.92 ± 0.03
NCTRER	0.85 ± 0.06	0.80 ± 0.08

counting is noticeable it is statistically insignificant. Thus although incrementing long walks is intuitively attractive, our experiments show that the effect on performance is minimal, and it can be avoided. On the other hand, long walks can lead to overfitting. The exception is with the extreme discount of 0.1, which eliminates the overfitting caused by longer walks as they are scaled down so drastically. Nevertheless the walk length is a parameter that must be chosen carefully for each dataset. We illustrate this point below as well.

Next we compare our kernel to other graph kernels on the challenging datasets Predictive Toxicology Challenge (PTC)[5] and the National Cancer Institute's AIDS Anti-viral Screen Program (NCI-HIV)[6] that have been widely used in the literature. The PTC dataset contains 417 chemical descriptions labeled according to their carcinogenicity to rodents. Each chemical is evaluated based on whether it was carcinogenic to female rats, female mice, male rats and male mice. Following Kashima et al. [2003] and others, we treat any molecule labeled "CE," "SE," or "P"

[5] `http://www.predictive-toxicology.org/ptc/`
[6] `http://dtp.nci.nih.gov/docs/aids/aids_data.html`

as positive, "NE" and "N" as negative, and ignore examples labeled "EE," "IS," and "E" as these labels indicate an unsure classification. It is important to notice that state-of-the-art performance on the PTC dataset is not far above the majority class for the dataset. The NCI-HIV dataset contains chemical descriptions labeled based on the ability of the chemical to inhibit HIV in a specific experimental context. Each chemical is labeled "confirmed active" (CA), "confirmed inactive" (CI), and "moderately active" (CM). In our experiments we ignore the moderately active chemicals as their label is less reliable (these are compounds that yielded different results in multiple measurements). Notice that this is a large dataset and its class distribution is very skewed.

Given the conclusions from previous experiments we used encoding 3 and no discount ($\gamma = 1$). On the PTC dataset, due to its small size we were able to explore a short (2), medium (5), and long (16) walk length. The NCI-HIV dataset is relatively large and it includes large molecules. To reduce learning time without altering dataset statistics, in each fold we removed molecules with more than 200 atoms from the training set but kept the test set unmodified.[7] Overall this means we removed 4 positive and 79 negative molecules from the training data. The results for walk lengths 2-5 are very similar and we report only the result for length 5.

Results are given in Table 4.4 and in each case we we include for reference the best known results from the literature. On the PTC dataset, the results we give are competitive with the best performance recorded in Fröhlich et al. [2005], which is attained using their optimal assignment kernel (OA) and the marginalized graph kernel Kashima et al. [2003] (MG). On the HIV dataset, in order to compare with previously reported results we report the area under the ROC curve (although precision and recall may be more appropriate due to the skew in labels). Our results outperform the frequent substructure propositionalization approach [Deshpande et al., 2003] and are competitive with the results reported for the Cyclic Pattern Kernel

[7]Because the kernel calculation time is dependent on the size of the hypergraph, a dataset with many large molecules will take longer to classify than one with smaller molecules.

(CPK) by Horváth et al. [2004] and the approximation of the infinite walk graph kernel (GK) reported by Gärtner [2005].

The various comparisons show that the hypergraph kernel can be used to obtain state of the art performance comparable to that of graph kernels when used with graph data. We observe that these results are obtained by different learning algorithms and parameter settings for each of the kernels (and papers they were reported in). This is appropriate, however, as parameters are optimized separately in each case. Nonetheless, we augment this comparison with experiments using the CPK kernel [Horváth et al., 2004] and our hypergraph kernel in exactly the same experimental conditions using the PAM algorithm and same settings as above. Results for NCTRER and NCI-HIV are given in Table 4.5. These too show that the the kernels give comparable performance on these problems.

Finally we explore the effect of hyperedges on performance. We present two related sets of experiments. In the first, we generated artificial data in order to illustrate a specific case in which the target function of a dataset is much easier to learn using a hypergraph representation as opposed to a graph representation. In such an instance the hypergraph kernel has a distinct advantage over graph kernels. The positive labeled examples in the artificial data are examples in which a specific conjunction is true and negative examples do not satisfy the conjunction. In the second experiment we show that the multi-arity background information in the Mutagenesis dataset increases performance with the hypergraph kernel. This demonstrates that the ability to handle high arity background knowledge as in ILP is useful for kernel methods as well.

Our data-generation routine was parametrized by the number of hyperedges in the graph, the maximum number of hyperedges incident with a single node, the total number of intersections between hyperedges, and the number of nodes per hyperedge. Since the number of hyperedges in the target conjunction is constant, the number of hyperedges in the hypergraph partially determines what proportion

of the hypergraph is irrelevant to the target; increasing the number of hyperedges increases the number of irrelevant features in the feature space of the hypergraph kernel. Likewise, increasing the total number of hyperedge intersections has a similar effect, as this affects how much similarity between hypergraphs is unrelated to the target sub-graph. Too low of a number of intersections (for example 1/2 the number of total hyperedges) leads to graphs whose only sizable connected subgraph is the target conjunction, hence rendering it trivial for the hypergraph kernel with walk depth of 2 (less than that needed to traverse the target subgraph) to separate the dataset.[8] The maximum number of hyperedges incident to a single node helps control the structure of the graph: by setting it low enough, we encourage the edge intersections to be evenly dispersed as opposed to having many star-like topologies in the graph.

Our data generation routine was as follows: we chose a simple target concept composed of three hyperedges: $p(a, b, c, d, e)q(b, f, g, h, i)r(j, f, k, l, m)$. This is similar the target concept in Figure 4.2 except that each hyperedge is extended by two nodes. To make the learning task non-trivial, we made sure that any two graphs, regardless of label, would likely share many common walks. To accomplish this we chose hypergraphs with 50 hyperedges, a maximum of 2 hyperedges incident to any one node, 75 total intersections between hyperedges, and 5 nodes per hyperedge. We restrict hyperedges from self-intersecting, and we do not allow one hyperedge to intersect more than once with another hyperedge. These settings create hypergraphs that are fully connected or close so that there will be many matching walks of non-trivial length between two graphs, regardless of their label. The final setting, 5 nodes per hyperedge, was simply to ensure enough nodes to accommodate the total number of intersections with the restrictions, and to vary the types of walks.

To generate each example, we started with an empty hypergraph. Then we

[8] Consider for example if we allowed no other intersections between hyperedges other than those in the target concept, leading to examples composed of lists of unconnected hyperedges. Despite these irrelevant features, the target concept would be easily learnable as it would be the only walk of length > 1.

Table 4.6: Results for artificial data measured by accuracy.

Walk Length	Hypergraph Encoding	Graph Encoding
1	0.48 ± 0.04	0.47 ± 0.04
2	0.79 ± 0.04	0.65 ± 0.04
3	0.90 ± 0.03	0.75 ± 0.04
4	0.92 ± 0.02	0.73 ± 0.04
5	0.93 ± 0.02	0.77 ± 0.04
6	0.97 ± 0.02	0.80 ± 0.04
7	0.98 ± 0.01	0.80 ± 0.04
8	0.98 ± 0.01	0.80 ± 0.04
9	0.98 ± 0.01	0.84 ± 0.03
10	0.98 ± 0.01	0.82 ± 0.03

selected its class by flipping a fair coin. If it was positive, we added the target sub-graph to the example. We then generated the remaining hyperedges (47 if it was a positive example, 50 if negative) such that no hyperedges intersected, choosing each hyperedge label randomly from $\{p, q, r, s\}$. For each of the remaining intersections, we proceeded as follows until all 75 total intersections were in the hypergraph: we randomly chose two hyperedges and a position on each hyperedge. If it was possible for the two hyperedges to intersect at the selected positions, without violating any of the above constraints or introducing another target sub-graph into the hypergraph, the intersection was created by merging the nodes at the intersection positions.

In addition to the hypergraph dataset, we created the corresponding graph dataset by transforming each hypergraph as discussed in Section 4.3.3.

We ran the hypergraph kernel from Equation (4.18) with walk length of $1-10$ on the hypergraph data and $1-10$ on the graph data. Table 4.6 shows the full results. Using the hypergraph representation gives a clear performance benefit. In the table we also see that the graph kernel improves as the walk length increases, however we checked walk lengths of up to twenty and there was no improvement over length ten. Thus, the experiments show that while it is possible to represent the hypergraph as a graph, using the hypergraph representation directly is advantageous even in

Table 4.7: Accuracy on Mutagenesis Dataset. The leftmost column shows walk length. Top: encoding 1. Bottom: encoding 3.

Length	AB	AB+H	AB+LC	AB+H+LC
1	0.70 +/- 0.06	0.76 +/- 0.07	0.85 +/- 0.09	0.76 +/- 0.07
2	0.69 +/- 0.10	0.84 +/- 0.10	0.85 +/- 0.09	0.84 +/- 0.10
3	0.78 +/- 0.09	0.86 +/- 0.10	0.84 +/- 0.07	0.86 +/- 0.10
4	0.82 +/- 0.10	0.84 +/- 0.08	0.81 +/- 0.10	0.84 +/- 0.08
5	0.82 +/- 0.09	0.84 +/- 0.10	0.80 +/- 0.11	0.84 +/- 0.10
10	0.84 +/- 0.10	0.80 +/- 0.09	0.83 +/- 0.09	0.80 +/- 0.09
Length	AB	AB+H	AB+LC	AB+H+LC
1	0.85 +/- 0.09	0.86 +/- 0.08	0.89 +/- 0.09	0.90 +/- 0.10
2	0.84 +/- 0.10	0.84 +/- 0.13	0.89 +/- 0.10	0.87 +/- 0.09
3	0.83 +/- 0.08	0.86 +/- 0.13	0.89 +/- 0.10	0.86 +/- 0.10
4	0.85 +/- 0.10	0.85 +/- 0.13	0.87 +/- 0.10	0.86 +/- 0.10
5	0.85 +/- 0.09	0.82 +/- 0.12	0.87 +/- 0.11	0.84 +/- 0.10
10	0.86 +/- 0.08	0.74 +/- 0.10	0.86 +/- 0.09	0.73 +/- 0.12

simple cases as captured in the artificial data. As argued above the graph kernel may be less efficient because both the graph size and walk size are larger than the corresponding structures on the hypergraph.

To explore the performance benefit of hyperedges in real-world data we use the Mutagenesis (MUTAG) dataset [Srinivasan et al., 1996] The dataset is widely used in ILP literature and contains chemicals that are labeled based on their mutagenicity; we used the 188 example "regression friendly" portion of the data. One of the interesting points in the work of Srinivasan et al. [1996] is the inclusion of different levels of background knowledge showing that more knowledge, and in particular high-arity relational knowledge, can be used for classification. Therefore this dataset is useful in exhibiting another case where we get a performance increase due to the use of hyperedges.

We ran experiments using the Mutagenesis dataset under encodings 1 and 3, and with various combinations of binary edges (atom-bond information), hyperedges (ring structures, etc.), and discretized charge, lumo, and logp features encoded as

unary edges. Only binary edges are affected by the encoding; we did not change hyperedges or unary edges. Note that lumo and logp are global properties of the molecule hence they translate to isolated nodes in the graph. Our kernel gives flexibility to use the hyperedges and multiple unary predicates for the same node (these correspond to multiple labels for a node). For example, we can label a node with its type and its charge independently for this dataset. The results are reported in Table 4.7; "AB" is atom-bond information, "H" is hyperedge information, and "LC" is lumo, logp and charge information. Our results are competitive with the best reported graph kernel for this dataset [Kashima et al., 2003]. When using encoding 1, it is clear that adding the hyperedges to the dataset gives a substantial gain in performance. Observe that the best length walk is shorter when using the hyperedges. This may be due to the fact that larger substructure may be captured in fewer edges. It may also explain the fact that, as a long enough walk in the graph can capture a ring structure, these structures may not significantly increase performance. When using encoding 3, the benefit of using hyperedges is less pronounced; this is likely due to the fact that because binary edges under encoding 3 contain information about their neighbors, they perform a similar function to that captured by rings as in our hyperedges. Overall, these results show that high arity background knowledge can improve the performance obtained by hypergraph kernel. If hyperedges capture information that is not derived from the graph structure one might expect to attain significant improvements in performance.

To summarize, the experiments demonstrate that our kernel can outperform ILP methods, that high arity predicates are easily incorporated as hyperedges and that this can be useful, and that the kernel is competitive with graph kernels when used on graph data. Discounting of long paths appears to not have a large effect but walk length must be chosen using parameter selection for each application separately.

The kernel can be implemented reasonably efficiently. On the NCTRER, PTC, and Mutagenesis datasets, a typical run time for 10-fold cross validation was under

a minute, and often less than 20 seconds on a dual 2.8 GHz Intel Xeon machine with at most one other job scheduled on it. On the NCI-HIV dataset, the runtime varied significantly by the walk length: for a length 3 walk, the average run time per fold was about 11 hours, while for a length 5 walk the average run time per fold was about 18 hours.

4.5 Conclusion

The main contribution of this chapter is a new kernel (and associated variants) that is able to work directly with hypergraphs. Using artificial data, we showed that the translation of hypergraphs to graphs degrades performance and we also demonstrated that hyperedges are useful in real-world data. On chemical datasets we showed that the hypergraph kernel performs competitively with other graph kernels and ILP methods. As already noted, the hypergraph kernel is limited in that it can only capture certain types of conjunctions. An important open question is whether a hypergraph kernel can be designed to cover a larger subset of conjunctions, in particular, capturing multiple shared nodes between edges and nodes shared between non-adjacent edges in a walk. Another interesting questions concerns the optimal assignment similarity developed by Fröhlich et al. [2005]. In this work, instead of taking a sum over all pairs of edges as in Equation (4.16), a "best" alignment of the edges is used to calculate the similarity. In Chapter 5 we discuss a general construction of approximations to maximum alignment kernels. It would be interesting to investigate whether we can use a similar construction to formulate a kernel that computes an approximation to the maximum alignment of two hypergraphs under the kernel proposed in this chapter.

Chapter 5

Kernels for Periodic Time Series Arising in Astronomy

In this chapter we study another form of kernel methods for structured data. In this case, the data are *time series*, vectors of pairs of real values and time values. In the astronomy domain, the real values are brightness measurements of stars and the time values are the date of the measurement. As with hypergraphs from Chapter 4, time series data have an inherent structure that is informative. In astronomy, the structure of the time series can be interpreted as a "shape" that is indicative of star type. This shape is easy to see in Figure 5.1. The challenge in this domain is to capture the intuitive notion of "shape" in a concrete mathematical form. We accomplish this by defining and analyzing two useful similarity measures.

The concrete application motivating this research is the classification of stars into meaningful categories from astronomy literature. A major effort in astronomy research is devoted to sky surveys, where measurements of stars' or other celestial objects' brightness are taken over a period of time. Classification as well as other analyses of stars lead to insights into the nature of our universe, yet the rate at which data are being collected by these surveys far outpaces current methods to classify them. For example, microlensing surveys, such as MACHO [Alcock et al.,

1993] and OGLE [Udalski et al., 1997] followed millions of stars for a decade taking one observation per night. The next generation panoramic surveys, such as Pan-STARRS [Hodapp et al., 2004] and LSST [Starr et al., 2002], will begin in 2009 and 2013, respectively, and will collect data on the order of hundreds of billions of stars. It is unreasonable to attempt manual analysis of this data, and there is an immediate need for robust, automatic classification methods.

In the data sets taken from star surveys, each example is represented by a time series of brightness measurements. We are concerned with *periodic variable* stars, that is, stars whose brightness varies as a periodic function of time. Different types of periodic variables have different periodic patterns. Figure 5.1 shows several examples of such time series generated from the three major types of periodic variable stars: Cepheid, RR Lyrae, and Eclipsing Binary.

As our first contribution we present several insights into the use of the cross-correlation function proposed by Protopapas et al. [2006] as a similarity function for time series. Cross-correlation provides an intuitive mathematical analog of what it means for two time series to look alike: we seek the best phase alignment of the time series, where the notion of alignment can be captured by a simple Euclidean distance or inner product. We show that cross-correlation is "almost" a kernel in that it satisfies the Cauchy-Schwartz inequality and induces a distance function satisfying the triangle inequality. Therefore, fast indexing methods can be used with cross-correlation for example with the k-Nearest Neighbor algorithm [Elkan, 2003]. We further show that although every 3×3 similarity matrix is positive semidefinite, some 4×4 matrices are not and therefore cross-correlation is not a kernel and not generally applicable with kernel methods.

As our second contribution we introduce a positive semidefinite similarity function that has the same intuitive appeal as cross-correlation. We investigate the performance of our kernel on real and artificial data sets, showing excellent performance. We show instances where the kernel outperforms all other methods as well

as instances where a simple universal phasing algorithm, which aligns every star to some fixed phase, performs comparably. Our investigation reveals that our kernel performs better than cross-correlation and that the ability to use Support Vector Machines (SVM) [Boser et al., 1992] with our kernel can provide a significant increase in performance.

As our final contribution, we build on the methods and findings of our first two contributions and create a complete system that automatically processes an entire survey. We explain the several tiers of processing prior to the classification stage and show preliminary results on the entire MACHO survey, giving newly discovered periodic variable stars. In the process we develop several techniques for variability testing and period finding. In addition to classifying stars, we estimate the confidence of the classification to place some stars into a group of unknown periodic variables for further review. This part of the system motivates our research into class-membership probabilities from Chapter 6 and uses some of the conclusions therein.

The remainder of the chapter is organized as follows. Section 5.1 investigates properties of cross-correlation, and Section 5.2 introduces the new kernel function. Related work is discussed in Sections 5.3. We present our experiments and discuss results in Section 5.4. In Section 5.5 we introduce our complete system for classifying periodic variable stars and report our analysis of the MACHO Survey.

5.1 Cross-Correlation

Our examples are vectors in $\mathbb{R}^n$ but they represent an arbitrary shifts of periodic time series. We use the following notation: y_{+s} refers to the vector y shifted by s positions, where positions are shifted modulo n. We then use the standard inner

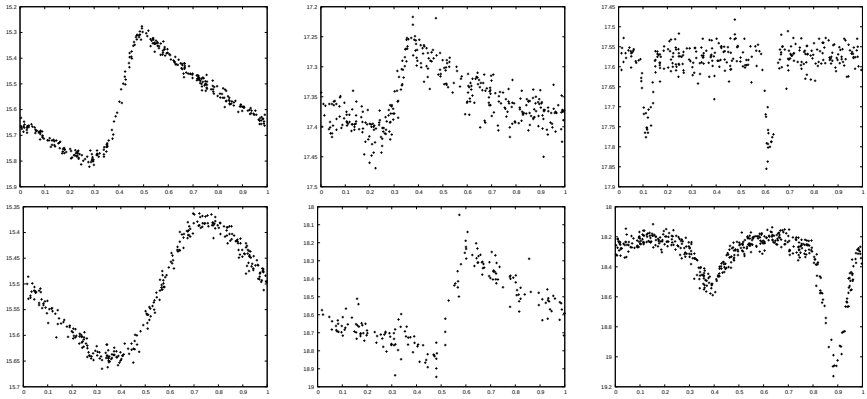

Figure 5.1: Examples of time series of periodic variable stars. Each column shows two stars of the same type. Left: Cepheid, middle: RR Lyrae, right: eclipsing binary. Examples of the same class have similar shapes but are not phase aligned. Examples are a result of folding a long sequence of observations leading to a noisy sample of one period of the light curve. The y-axis labels represent brightness in magnitude units, which is an inverse logarithmic scale (this is the convention in astronomy).

product between shifted examples

$$\langle x, y_{+s} \rangle = \sum_{i=1}^{n} x_i (y_{+s})_i.$$

We define the *cross-correlation* [Protopapas et al., 2006] between $x, y \in \mathbb{R}^n$ as

$$C(x, y) = \max_s \langle x, y_{+s} \rangle. \tag{5.1}$$

In the context of time series, computing the cross-correlation corresponds to aligning two time series such that their inner product, or similarity, is maximized.

5.1.1 Properties of Cross-Correlation

We first show that cross-correlation has some nice properties making it suitable as a similarity function:

Theorem 5.1.1.

(P1) $C(x, x) = \langle x, x \rangle \geq 0$.

(P2) $C(x,y) = C(y,x)$.

(P3) The Cauchy-Schwartz Inequality holds, i.e., $\forall x, y,\ C(x,y) \leq \sqrt{C(x,x)C(y,y)}$.

(P4) If we use the cross-correlation function to give a distance measure d *such that*

$$d(x,y)^2 = C(x,x) + C(y,y) - 2C(x,y) = \min_s \|x - (y_{+s})\|^2$$

then d *satisfies the Triangle Inequality.*

In other words cross-correlation has properties similar to an inner product, and can be used intuitively as a similarity function. In particular, we can use metric trees and other methods based only on the triangle inequality [Moore, 2000, Elkan, 2003] to speed up distance based algorithms using cross-correlation.

Proof. For (P1) observe that by definition $C(x,x) \geq \langle x, x\rangle$. On the other hand, $C(x,x) = \sum x_i x_{i+s}$, and by the Cauchy-Schwartz inequality,

$$\sum x_i x_{i+s} \leq \sqrt{\sum x_i^2}\sqrt{\sum x_{i+s}^2} = \sqrt{\sum x_i^2}\sqrt{\sum x_i^2} = \langle x, x\rangle. \tag{5.2}$$

Which means $\langle x, x\rangle \geq C(x,x) \geq \langle x, x\rangle$ or $C(x,x) = \langle x, x\rangle \geq 0$.

To prove (P2) observe that since

$$\langle x, y_{+s}\rangle = \langle x_{-s}, y\rangle = \langle x_{+(n-s)}, y\rangle,$$

maximizing over the shift for y is the same as maximizing over the shift for x.

(P3) follows from K1 of Theorem 5.1.2 below (see Proposition 2.7 of Schölkopf and Smola [2002]) but we give a direct argument here. Let

$$C(x,y) = \langle x, y_{+s}\rangle = \langle x, z\rangle$$

where s is the shift maximizing the correlation and where we denote $z = y_{+s}$. Then

by (P1),

$$\sqrt{C(x,x)C(y,y)} = \sqrt{\langle x,x\rangle\langle y,y\rangle} = \|x\|\|y\|.$$

Therefore the claim is equivalent to $\|x\|\|y\| \geq \langle x,z\rangle$, and because the norm does not change under shifting the claim is equivalent to $\|x\|\|z\| \geq \langle x,z\rangle = C(x,y)$. The last inequality holds by the Cauchy-Schwartz inequality for normal inner products.

Finally, for (P4) let $x,y,z \in \mathbb{R}^n$. Let τ_{ab} be the shift that minimizes $d(a,b)$.

$$\begin{aligned}
d(x,y) + d(y,z) &= \|(x_{+\tau_{xy}}) - y\| + \|(y_{+\tau_{yz}}) - z\| && (5.3)\\
&= \|(x_{+\tau_{xy}+\tau_{yz}}) - (y_{+\tau_{yz}})\| + \|(y_{+\tau_{yz}}) - z)\| && (5.4)\\
&\geq \|(x_{+\tau_{xy}+\tau_{yz}}) - (y_{+\tau_{yz}}) + (y_{+\tau_{yz}}) - z\| && (5.5)\\
&= \|(x_{+\tau_{xy}+\tau_{yz}}) - z\| && (5.6)\\
&\geq \|(x_{+\tau_{xz}}) - z\| && (5.7)\\
&= d(x,z) && (5.8)
\end{aligned}$$

Where (5.4) holds because shifting x and y by the same amount does not change the value of $\|x-y\|$, (5.5) holds because of the triangle inequality, and (5.7) holds because by definition τ_{xz} minimizes the distance between x and z. □

Since cross-correlation shares many properties with inner products it is natural to ask whether it is indeed a kernel function. We show that, although every 3x3 similarity matrix is positive semidefinite, the answer is negative.

Theorem 5.1.2.

(K1) Any 3×3 Gram matrix of the cross-correlation is positive semidefinite.

(K2) The cross-correlation function is not positive semidefinite.

Proof. Let $x_1, x_2, x_3 \in \mathbb{R}$, G a 3×3 matrix such that $G_{ij} = C(x_i, x_j)$, $c_1, c_2, c_3 \in \mathbb{R}$. We prove K1 by showing $Q = \sum_{i=1}^{3}\sum_{j=1}^{3} c_i c_j G_{ij} \geq 0$.

At least one of the products c_1c_2, c_1c_3, c_2c_3 is non-negative. Assume WLOG that $c_2c_3 \geq 0$ and shift x_2 and x_3 so that they obtain the maximum alignment with x_1,

calling the shifted versions $\tilde{x_1}, \tilde{x_2}, \tilde{x_3}$ noting that $\tilde{x_1} = x_1$. Now $C(x_i, x_j) = \langle \tilde{x_i}, \tilde{x_j} \rangle$ except possibly when $(i, j) = (2, 3)$, thus

$$\begin{aligned}
\sum_{i=1}^{3}\sum_{j=1}^{3} c_i c_j G_{ij} &= \sum_{i=1}^{3}\sum_{j=1}^{3} c_i c_j \langle \tilde{x_i}, \tilde{x_j} \rangle + 2c_2 c_3 (C(\tilde{x_2}, \tilde{x_3}) - \langle \tilde{x_2}, \tilde{x_3} \rangle) \\
&\geq \sum_{i=1}^{3}\sum_{j=1}^{3} c_i c_j \langle x_i, x_j \rangle \geq 0
\end{aligned}$$

since $c_2 c_3 \geq 0$ and $C(\tilde{x_2}, \tilde{x_3}) \geq \langle \tilde{x_2}, \tilde{x_3} \rangle$ by definition.

The negative result, K2, is proved is by giving a counter example. Consider the matrix A and the row-normalized A'

$$A = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 0 & 0 \\ 2 & 1 & 2 \\ 0 & 2 & 1 \end{pmatrix} \quad A' = \begin{pmatrix} 0 & 0.4472 & 0.8944 \\ 1 & 0 & 0 \\ 0.6667 & 0.3333 & 0.6667 \\ 0 & 0.8944 & 0.4472 \end{pmatrix}$$

where each row is a vector of 3 dimensions. This illustrates a case where we have 4 time series, each with 3 samples and the time series are normalized. Using the cross-correlation function on A', we would get the following Gram matrix

$$G = \begin{pmatrix} 1 & 0.8944 & 0.8944 & 0.8 \\ 0.8944 & 1 & 0.6667 & 0.8944 \\ 0.8944 & 0.6667 & 1 & 0.8944 \\ 0.8 & 0.8944 & 0.8944 & 1 \end{pmatrix}$$

G has a negative eigenvalue of -0.0568 corresponding to the eigenvector

$$c = (-0.4906, 0.5092, 0.5092, -0.4906)$$

and therefore G is not positive semidefinite. In other words

$$cGc' = \sum_{i=1}^{4}\sum_{j=1}^{4} c_i c_j G_{ij} = -0.0568. \qquad \square$$

5.2 A Kernel for Periodic time Series

As the cross-correlation function is not positive semidefinite, we propose an alternative kernel function that can be used in place of the cross-correlation function with kernel methods. To motivate our choice consider first the kernel

$$K(x,y) = \sum_{i=1}^{n}\sum_{j=1}^{n}\langle x_{+i}, y_{+j}\rangle.$$

Observe that here K iterates over all possible shifts, so that we no longer choose the best alignment but instead aggregate the contribution of all possible alignments. This seems to lose the basic intuition behind cross-correlation and it is indeed not a good choice. On closer inspection we can see that

$$\begin{aligned} K(x,y) &= (x_{+1} + x_{+2} + \ldots + x_{+n})y_{+1} + \ldots + (x_{+1} + x_{+2} + \ldots + x_{+n})y_{+n} \\ &= (\sum_{i=1}^{n} x_{+i})(\sum_{j=1}^{n} y_{+j}). \end{aligned}$$

Hence K simply calculates the product of the sums of the shifted vectors. For example, if the vectors are normalized to have a mean of 0, then K is identically 0.

Instead our kernel weights each shift with exponential function in order that shifts with high correlation are highly weighted and shifts with low correlation have smaller effect.

Definition 5.2.1. The kernel function $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is defined as

$$K(x,y) = \sum_{i=1}^{n} e^{\gamma\langle x, y_{+i}\rangle} \tag{5.9}$$

where $\gamma > 0$ is a constant.

Thus like cross-correlation the value of the kernel will be dominated by the maximizing alignment although the number of "good alignments" is also important. In this way we get positive semidefinite kernel while having the same guiding intuition as cross-correlation. Exponential weighting of various alignments of time series has been proposed previously in Cuturi et al. [2007]. Despite the similarity in the construction, the proof of positive semidefiniteness in Cuturi et al. [2007] does not cover our case as their set of alignments is all possible time warpings under a fixed phase and does not allow for circular shifting. Similar ideas to weight different matches exponentially have also been explored in kernels for multi-instance problems Gärtner et al. [2002].

Theorem 5.2.2. *K is a positive semidefinite kernel.*

Proof. Consider the following function

$$K'(x,y) = \sum_{i=1}^{n}\sum_{j=1}^{n} e^{\gamma\langle x_{+i}, y_{+j}\rangle}.$$

It follows from results in Haussler [1999] that $K'(x,y)$ is a convolution kernel. This can be directly shown as follows. First rewrite K' as

$$K'(x,y) = \sum_{a\in R^{-1}(x)}\sum_{b\in R^{-1}(y)} e^{\gamma\langle a,b\rangle} \tag{5.10}$$

where $R^{-1}(x)$ gives all shifts of x. It is well known that the exponential function $e^{\gamma\langle x,y\rangle}$ is a kernel [Schölkopf and Smola, 2002]. Let $\Phi(x)$ be the underlying vector representation of the this kernel so that $e^{\gamma\langle x,y\rangle} = \langle\Phi(x),\Phi(y)\rangle$. Then

$$K'(x,y) = \sum_{a\in R^{-1}(x)}\sum_{b\in R^{-1}(y)} \langle\Phi(a),\Phi(b)\rangle = \langle(\sum_{a\in R^{-1}(x)}\Phi(a)),(\sum_{b\in R^{-1}(y)}\Phi(b))\rangle \tag{5.11}$$

Thus K' is an inner product in the same vector space captured by Φ with the map

being the aggregate of all elements in $R^{-1}(x)$.

Observe that $K'(,)$ iterates over all shifts of both x and y, hence effectively counting each shift n times. For example, observe that for the identity shift, we have $\langle x, y\rangle = \langle x_{+1}, y_{+1}\rangle = \ldots = \langle x_{+(n-1)}, y_{+(n-1)}\rangle$. Hence we need to scale K' by $1/n$ in order to count each shift exactly once. This gives us

$$K(x,y) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n} e^{\gamma\langle x_{+i}, y_{+j}\rangle}.$$

Since scaling a kernel (i.e., K') is also a kernel, K is a kernel. □

Protopapas et al. [2006] showed that cross-correlation can be calculated in time $O(n \log n)$ where n is the length of the time series. In particular they show that

$$\langle x, y_{+s}\rangle = \mathcal{F}^{-1}(\mathcal{X} \cdot \hat{\mathcal{Y}})[s]$$

where $\cdot$ indicates point-wise multiplication, $\mathcal{X}$ is the Discrete Fourier Transform (DFT) of x, and $\hat{\mathcal{Y}}$ is the complex conjugate of the DFT of y. Therefore cross-correlation can be calculated as

$$C(x,y) = \max_s \mathcal{F}^{-1}(\mathcal{X} \cdot \hat{\mathcal{Y}})[s]$$

and using the DFT we get the claimed time bound. This easily extends to our kernel by calculating

$$K(x,y) = \sum_s e^{\mathcal{F}^{-1}(\mathcal{X}\cdot\hat{\mathcal{Y}})[s]}$$

implying:

Proposition 5.2.3. *$K(x,y)$ can be calculated in time $O(n \log n)$.*

Note that we need take the Fourier Transform of each example only once. This gives a significant practical speedup over the naïve quadratic time implementation.

5.3 Related Work

The current discoveries from the microlensing surveys such as OGLE and MACHO are predominantly transient objects such as gravitational microlensing, supernovae etc., and some periodic variable stars [Faccioli et al., 2007, Alcock et al., 1995]. Recent work on star surveys introduced the application of semi-automatic classification techniques for periodic variable stars based on simple selection criteria over the parameter space indexed by average brightness (magnitude), average difference in brightness between two spectral regions or passbands, and period [e.g., Geha et al., 2003, Howell et al., 2005]. We refer to these three parameters as *explicit features*. The semi-automatic methods require significant human intervention and hence pose an imperfect solution for a survey of even tens of millions of stars. An automatic approach has been proposed in Debosscher et al. [2007]. This approach extracts explicit features from the light curves and applies machine learning methods in the resulting parameter space. Despite the similarity in terms of automation, our approach is unique in that we use the shape of the periodic time series to derive a similarity measure. Furthermore our approach is not astronomy-specific and is applicable across a range of domains.

There are many existing approaches for processing and classifying time series. A classical approach is to extract features of the time series, such as the Fourier basis, wavelets, or Hermite basis representation, and then work directly in the resulting vector space, [e.g., Vlachos et al., 2005, Osowski et al., 2004]. Another major approach models the time series using a generative probabilistic model, such as Hidden Markov Models (HMM), and classifies examples using maximum likelihood or MAP estimates [Ge and Smyth, 2000]. Our work falls into a third category: using similarity functions or distance measures for time series data [Berndt and Clifford, 1994, Lu et al., 2008]. Various similarity functions for time series have been proposed. Notably, Dynamic Time Warping (DTW) has been shown to be very effective across a large number of applications [Berndt and Clifford, 1994, Keogh et al., 2006]. For

instance, a popular method for representing 2-d shapes is to create a time series from the contour of the shape.[1] Such similarity functions are not phase invariant, hence they rely on a good universal phasing of the data.

Cross-correlation has been proposed precisely as an effective phase-invariant similarity function for astronomy and has been used for anomaly detection [Protopapas et al., 2006]. It is faster in runtime, $O(n \log n)$, than other methods that compute a maximum phase-invariant alignment. The notion of phase-invariant similarity has also been explored in the context of time series classification, specifically for time series generated from 2-d shape contours. For example, Keogh et al. [2006] present a method for applying any distance measure in a phase-invariant context. This allows for the application of Dynamic Time Warping, for instance, to data that is phase-invariant. While in general the run-time ($O(n^3)$) is as bad as brute-force methods such as in Adamek and O'Connor [2004], they give experimental evidence that their heuristics lead to much faster run-times in practice. We extend the work in Protopapas et al. [2006] by investigating theoretical properties of cross-correlation and proposing a positive semidefinite alternative.

Several alternative approaches for working with non-positive semidefinite similarity measures exist in the literature. The simplest approach is just to use the (non-PSD) similarity function with SVM and hope for good results. Our experiments in the next section show that this does not always yield the desired performance. Another common alternative is to add a diagonal term λI to the Gram Matrix in order to render it positive semidefinite. More recent approaches reformulate the SVM optimization to account for the potential non-PSD kernel [Luss and d'Aspremont, 2007, Ong et al., 2004]. Finally, Balcan et al. [2008] show that a similarity function that meets some general requirements can be used to project examples into an explicit feature space indexed by their similarity to a fixed set of examples, and that

[1]Shape classification has its own domain-specific approaches and it is beyond the scope of this paper to examine them. Nevertheless we observe that shape matching is an example of a phase-invariant time series classification problem, and in fact we will present experiments from this domain.

this preserves some learnability properties. Unlike these generic methods, our work gives an explicit kernel construction that is particularly useful for the time series domain.

The general issue of "maximizing alignment" appears repeatedly in work on kernels for structured objects. Dynamic Time Warping is a classic (non-positive semidefinite) example where we maximize alignment under legal potential warping of the time axis. A general treatment of such alignments, characterizing when the result is a kernel, is developed by Shin and Kuboyama [2008]. Their results do not cover the case of cross-correlation, however. A similar alignment idea has been used for graph kernels in the application of classifying molecules, where each molecule can be seen as a graph of atoms and their bonds (see Chapter 4). Here a base kernel is introduced between pairs of nodes in the two graphs. Then one can define a convolution kernel between the graphs using an equation similar to (5.10) where the sum ranges over all nodes in the graph, analogous to (4.16). This approach does not maximize alignments, but sums over all possible alignments. A non-positive semidefinite alternative is to maximally align the two molecules by pairing their atoms in a one-to-one manner [Fröhlich et al., 2005]. A major question is whether one could define an efficiently computable exponentially weighted version of such a (non-maximizing but PSD) graph kernel (see Cuturi [2007]). One can show that this problem is closely related to calculating the permanent, a problem well known to be computationally hard [Valiant, 1979, Papadimitriou, 1993]. As it is a special case of the permanent problem, however, where edge weights are related through the kernel function, it may be possible to calculate efficiently.

5.4 Experiments

In this section we investigate the performance of our kernel as a general kernel for periodic time series. In particular we explore whether it is useful for the astronomy application, assuming appropriate pre-processing of the data. Once we have estab-

lished excellent performance in the astronomy domain, in the next section we build a fully automatic system for processing star surveys based on our kernel.

For real-world data we use time series from astronomy and time series generated from contours of 2-d images. For artificial data, we generate examples that highlight the importance of phase invariance in an intuitive fashion. We use the same preprocessing for all time series, unless otherwise noted. The time series are smoothed as in previous work [Protopapas et al., 2006, Gorry, 1990], linearly-interpolated to 1024 evenly spaced points, and normalized to have mean of 0 and standard deviation of 1.

In all experiments we use the LIBSVM [Chang and Lin, 2001] implementation of SVM [Boser et al., 1992] and k-Nearest Neighbors (k-NN) to perform classification. For LIBSVM, we choose the one-versus-one multiclass setting, and we do not optimize the soft-margin parameter, instead using the default setting. For k-NN, we choose $k = 1$ following Keogh et al. [2006], who have published results on the shape data used here.[2] For some of the experiments in the astronomy domain, we use explicit features. When we do so, we use a linear kernel. When we use cross-correlation or our kernel in addition to explicit features, we simply add the result of the inner product of the explicit features to the value of the cross-correlation or kernel.[3]

We use five different similarity functions in our experiments: Euclidean Distance (ED) returns the inner product of two time series. The Universal Phasing (UP) similarity measure uses the method from Protopapas et al. [2006] to phase each time series according to the sliding window on the time series with the maximum mean, and then behaves exactly like Euclidean Distance. We use a sliding window

[2]We reproduce their experiments as opposed to reporting their results in order to account for the different splits when cross-validating; our results do not differ significantly from those reported by Keogh et al. [2006].

[3]Another approach would be to perform multiple kernel learning (see for example Sonnenburg et al. [2006]) with one kernel being the cross-correlation and the other the inner product of the explicit features. However, this issue is orthogonal to the topic of the chapter hence we use the simple weighting.

size of 5% of the number of original points; the phasing takes place after the pre-processing explained above. In all experiments where we use K as in Equation 5.9, we do parameter selection by performing 10-fold cross-validation on the training set for each value of γ in $(1, 5, 10, 15, 25, 50, 80)$, then re-train using the value of γ that gave best average accuracy on the training set. When we use Dynamic Time Warping (DTW), we use the standard algorithm and do not restrict the warping window [Berndt and Clifford, 1994]. Finally we note that although cross-correlation is not positive semidefinite, we can in practice use it on some data sets with SVM.

In the first set of experiments we run on the OGLEII dataset [Soszynski et al., 2003]. This data set consists of 14087 time series (light curves) taken from the OGLE astronomical survey. Periods of each star in the data set are given and were selected by domain experts. For a more detailed description of this data set, see Section 5.5.1 in this chapter. Each light curve is from one of three kinds of *periodic variable star*: Cepheid, RR Lyrae (RRL), or Eclipsing Binary (EB). We run 10-fold cross-validation over the entire data set, using the cross-correlation (CC), our kernel (K), and Universal Phasing (UP). The results, shown in the left top three rows of Table 5.1, illustrate the potential of the different similarities in this application. We see significant improvements for both cross-correlation and the kernel over Universal Phasing. We also see that the possibility to run SVM with our kernel leads to significant improvement over cross-correlation.

While the results reported thus far on OGLEII are good, they are not sufficient for the domain of periodic variable star classification. Thus we turn next to improvements that are specific to the astronomy domain. In particular, the astronomy literature identifies three aggregate features that are helpful in variable star classification: the average brightness (magnitude) of the star, the *color* of the star which is the difference in average brightness between two different spectra, and the *period* of the star, i.e., the length of time to complete one period of brightness variation [Geha et al., 2003, Howell et al., 2005]. The right side of Table 5.1 gives the results when

Table 5.1: Accuracies with standard deviation reported from 10-fold cross-validation on OGLEII using various kernels and the cross-correlation

	1-NN	SVM		1-NN	SVM
CC	0.844 ± 0.011	0.680 ± 0.011	features + CC	0.991 ± 0.002	0.998 ± 0.001
K	0.901 ± 0.008	0.947 ± 0.005	features + K	0.992 ± 0.002	0.998 ± 0.001
UP	0.827 ± 0.010	0.851 ± 0.006	features + UP	0.991 ± 0.002	0.997 ± 0.001
			features	0.938 ± 0.006	0.974 ± 0.004

these features are added to the corresponding similarities. The features on their own yield very high accuracy, but there is a significant improvement in performance when we combine the features with cross-correlation or the kernel. Interestingly, while Universal Phasing on its own is not strong enough, it provides improvement over the features similar to our kernel and cross-correlation. Notice that a performance gain of 2% is particularly significant in the domain of astronomy where our goal is to publish such star catalogs with no errors or very few errors. The left confusion matrix in Table 5.2 (for SVM with our kernel plus features) shows that we can get very close to this goal on the OGLEII data. To our knowledge this is the first such demonstration of the potential of applying a shape matching similarity measure in order to automatically publish clean star catalogs from survey data.[4] In addition, based on our domain knowledge, some of the errors reported in the left of Table 5.2 appear to be either mis-labeled or borderline cases whose label is difficult to determine.

In addition to classification, we show in Table 5.2 that the confidences produced by the classifier are well ordered. Here we do not perform any calibration (akin to the normalization method from Section 6) and simply take the raw output of each of the three hyperplanes learned by the SVM. While we investigated the use of the methods from Section 6 here, in exploratory experiments we did not find the probability estimates to be more reliable than simply ordering the classifications

[4]Wyrzykowski et al. [2004] uses a shape-based similarity measure but only for EBs and not in an automatic classification setting.

Table 5.2: Four confusion matrices for OGLEII, using SVM with K and features. Left to right, top to bottom, we abstain from none, then the lowest 1%, 1.5% and 2%.

	Ceph	EB	RRL
Cepheid	3416	1	13
EB	0	3389	0
RRL	9	0	7259

	Ceph	EB	RRL
Cepheid	3382	1	3
EB	0	3364	0
RRL	1	0	7195

	Ceph	EB	RRL
Cepheid	3363	1	3
EB	0	3342	0
RRL	0	0	7166

	Ceph	EB	RRL
Cepheid	3352	1	0
EB	0	3312	0
RRL	0	0	7138

according to the raw output (also note that what we really want to do is rank the output, not assign probabilities). To calculate the confidence in label 1, we add the raw output of the $1v2$ (the classifier separating class 1 from class 2) and $1v3$ classifiers. To calculate the confidence in label 2 we add the negative output of the $1v2$ hyperplane and the output of the $2v3$ hyperplane, etc. We can then abstain from the examples that received the lowest confidences and set them aside for review. When we abstain from the lowest 1%, for example, we abstain from all but 5 errors, showing that almost all of our errors have low confidences. We now have reason to believe that, when we classify a new catalog, we can reliably abstain from a certain percentage of the predictions that are most likely to be errors. The examples on which we abstain can either be ignored or set aside for human review.

In the next set of experiments we use five shape data sets: Butterfly, Arrowhead, Fish, Seashells introduced in Keogh et al. [2006], as well as the SwedishLeaf data set from Söderkvist [2001].[5] These data sets were created by taking pictures of objects and creating a time series by plotting the radius of a line anchored in the center of the object as it rotates around the image [Keogh et al., 2006]. As all of the pictures have aligned each object more or less along a certain orientation, we randomly permute each time series prior to classification in order to eliminate any

[5]Detailed Information available via `www.cs.ucr.edu/~eamonn/shape/shape.htm`

Table 5.3: Number of examples in each data set. For those data sets that were filtered to include 20 examples of each class, the number of examples post-filtering appears after the '/'.

	Num Examples	Num Classes	Majority Class
Arrowhead	558/474	9	0.19
Butterfly	754/312	5	0.39
Intershape	2511	4	0.30
SwedishLeaf	1125	15	0.07

bias of the orientation. The identification of objects from various orientations is now cast as a phase-invariant time series problem.

A natural and relatively easy problem is to use a classifier to separate the different image types from each other. In this case we attempt to separate butterflies, arrowheads, seashells, and fish. We refer to this data set as *Intershape*.[6] We also investigate the potential to separate sub-classes of each shape type. The SwedishLeaf data has already been labeled before, and hence the sub-classes are already identified. For the other data sets that have not been explicitly labeled by class before, we generate labels as follows: for the Butterfly and Fish data set, we consider two examples to be the same class if they are in the same genus. For the Arrowhead data set, we consider two arrowheads to be the same type if they share the same type name, such as "Agate Basin," or "Cobbs." In order to make the results more statistically robust, we eliminate sub-types for which there exist fewer than 20 examples. Seashells and Fish have too few examples when processed in this way and are therefore only used in the Intershape data set. A summary of the data sets, including number of examples and majority class probability (that can be seen as a baseline) are given in Table 5.3.

For these experiments we calculate no explicit features. We run 10-fold cross-validation using 1-NN with cross-correlation (1-NN CC), the kernel (1-NN K), Dynamic Time Warping (1-NN DTW), Universal Phasing (1-NN UP) and SVM with the kernel (SVM K), Universal Phasing (SVM UP), and Euclidean distance (SVM

[6]We treat the SwedishLeaf set differently because it has a different resolution and is not part of the same overall shape data set.

Table 5.4: Performance on various shape data sets. All results are cross-validated. Data set names: A = arrowhead, B = butterfly, I = intershape, S = Swedish

	1-NN CC	1-NN K	1-NN DTW	1-NN UP	SVM ED	SVM UP	SVM K
A	0.54 ± 0.06	0.54 ± 0.08	0.33 ± 0.06	0.49 ± 0.05	0.2 ± 0.05	0.41 ± 0.05	0.63 ± 0.04
B	0.73 ± 0.04	0.73 ± 0.04	0.59 ± 0.08	0.70 ± 0.07	0.4 ± 0.1	0.65 ± 0.08	0.76 ± 0.08
I	0.98 ± 0.01	0.98 ± 0.01	0.84 ± 0.03	0.97 ± 0.02	0.47 ± 0.03	0.8 ± 0.02	0.91 ± 0.02
S	0.84 ± 0.03	0.82 ± 0.03	0.48 ± 0.06	0.78 ± 0.04	0.08 ± 0.03	0.18 ± 0.03	0.33 ± 0.04

ED). The results are given in Table 5.4. We also tried using 1-NN with Euclidean Distance, but the performance was not competitive with any of the other methods hence we do not include it in the comparison.

The results demonstrate that both cross-correlation and the kernel provide a significant performance advantage. It is not surprising that DTW does not do well because it only considers the one given random phasing of the data. Rather, it is surprising that it does not perform worse on this data. The only way it can expect to perform well with k-NN is if, by chance, for each example there is another example of the same class that happens to share roughly the same phase. In a large enough data set, this can happen, and this may explain why DTW does much better than random guessing. It is interesting that SVM does not always dominate k-NN and does very poorly on SwedishLeaf. It may be that the data are linearly inseparable but there are are enough examples such that virtual duplicates appear in the data allowing 1-NN to do well.

Another interesting observation is that while Universal Phasing never outperforms all methods it does reasonably well across the domains. Recall that this method phases the time series according to the maximum average magnitude of a sliding window. This finds a "maximum landmark" in the data for alignment and is obviously not guaranteed to be informative of the class in every case. Nevertheless, it works well on the Butterfly and Intershape data sets showing that this type of landmark is useful for them.

As we show in the next set of experiments with artificial data, it is easy to

construct examples where Universal Phasing will fail. We generate two classes of time series. Each example contains 1024 points. Class 1 is a multi-step function with one set of four steps beginning at time 0, as well as one spike placed randomly. Class 2 is also a multi-step function but with two sets of two steps, the first at time 0 and the second at time 665 (roughly 65% of the entire time series) and one random spike exactly as in class 1. We show two examples of each class in Figure 5.2. We generate 10 disjoint training sets containing 70 examples and test sets containing 30 examples for cross-validation. We keep the training set small to avoid clobbering the results by having near-identical examples. In these experiments we normalize as above, however we do not perform smoothing as the data contains no noise.

For this type of data the random spike will always be in the center of the largest magnitude sliding-window, and hence Universal Phasing will phase each time series according to the random location of the spike. In a real world setting, the random spike could be sufficiently wide noise period in the signal, or any irrelevant feature of the time series. This is key to understanding the strength of our method: if it is easy to find a global shifting such that each example is maximally correlated with every other, our method performs identically to Universal Phasing. On the other hand, when a global shift is not trivial to find, our method succeeds where a Universal Phasing algorithm fails. To illustrate further the performance potential of the kernel we create a second version of the data where we add noise to the labels by flipping the label of each example with probability of 0.1. When the data are completely or nearly separable, both k-NN and SVM should attain close to 100% accuracy. The noise changes the domain to make it harder to get this level of performance.

The results are shown in Table 5.5. As expected, Universal Phasing does quite poorly in this setting. With no noise, 1-NN with cross-correlation, 1-NN with our kernel, and SVM with our kernel attain almost 100% accuracy. The results with noisy data show that SVM with our kernel is more robust to noise than 1-NN with cross-correlation or our kernel.

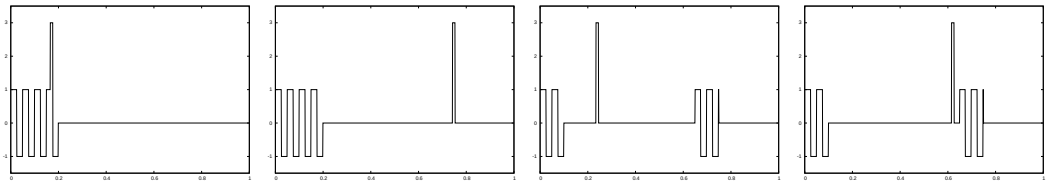

Figure 5.2: Examples of artificial data. The left two examples are from class 1, the right two example are from class 2.

To summarize, the experiments show that universal phasing and cross-correlation are useful in some contexts, but that our kernel combined with SVM can yield significant improvements in performance.

Table 5.5: Results on artificial data

	1-NN CC	1-NN K	1-NN UP	SVM UP	SVM K
Artificial	0.99 ± 0.02	1.00 ± 0.00	0.65 ± 0.04	0.50 ± 0.07	0.997 ± 0.001
Artificial w/ Noise	0.84 ± 0.14	0.84 ± 0.12	0.61 ± 0.09	0.53 ± 0.12	0.90 ± 0.05

5.5 A Fully Automated System for Classifying Periodic Variable Stars

Now that we have established a machine learning basis for classification of periodic time series, we can address the entire problem of processing, filtering, and classifying an astronomy catalog. This endeavor encompasses much more than just classification: for example, from ~ 25 million stars in the MACHO survey, we must eliminate all but approximately 50 thousand stars of interest[7] prior to running our classification routine from Section 5.4. To process a survey our system combines various statistical tests, machine learning algorithms, and signal-processing techniques. As proof-of-concept we use our system to find new RRLs, Cepheids, and Eclipsing Binaries in the MACHO survey, starting with no information other than the raw light

[7]Based on known distributions of periodic variable stars.

curves (time series). In addition to identifying new periodic variables, our system produces two useful categories of non-classified data: stars that are too much like two or more of RRL, Cepheid, or EB to make a confident decision, and stars that are periodic in some way but are not like any of RRL, Cepheid, or EB. The stars in the first category can be further processed to determine the best category; the stars in the second category serve as a reserve of potentially interesting astronomical events that can be reviewed by a domain expert. For example, we have found several stars whose period changes over time; this is a known astronomical phenomenon and a catalog of such stars would be useful to astronomers.

Using the OGLEII survey as a training set, we classify fields 1-82, 206, 207, 208, 211, 212, 213, 301-311, 401-403 in the MACHO survey, containing $25,309,792$ objects. Our system produces a list of $8,045$ new EBs, $6,792$ new Cepheids, $16,876$ new RRLs, and $3,787$ stars which may be one of the three types, and $24,944$ stars that are not one of the three types, but are periodic variable events of interest. To our knowledge, this is the first such system that is completely automatic. The system of Debosscher et al. [2007], for example, automatically classifies periodic variable stars, but only from among a set of known periodic variable stars; that is, first work must be done to identify periodic variable stars from within the dataset. Our system takes as input a complete, unfiltered survey, and automatically creates a catalog of periodic variables.

In Section 5.5.1 we describe the MACHO and OGLEII data. In Section 5.5.2 we define each part of our system and analyze its performance on the MACHO catalog. Finally in Section 5.5.5 we discuss ideas for future improvements to the system based on its performance on the MACHO catalog.

Table 5.6: Details of OGLEII dataset

	Num Stars in LMC	Num Stars in SMC	Total
Cepheid	1374	2051	3425
EB	2266	1124	3390
RRL	6812	460	7272
Total	10452	3635	14087

5.5.1 Description of Data & Initial Preprocessing

For each survey, we are given a time series for each star that was observed, in each *pass-band*[8] of that survey. As above, each time series consists of a list of magnitude measurements, the time each measurement was taken, and an estimated error of the measurement. Below, we refer to the magnitude measurement i of a time series as x_i, the associated time of the measurement as t_i, and the associated error of the measurement as σ_i^2. Figure 5.3 shows an example time series and illustrates the non-uniform sampling.

The OGLEII dataset [Soszynski et al., 2003] contains a total of 14087 light curves from periodic variable stars sampled in the standard V,B, and I passbands, from both the Small Magellanic Cloud (SMC) and Large Magellanic Cloud (LMC). There are 3425 Cepheids, 3390 Eclipsing Binaries, and 7272 RRLs in all. See Table 5.6 for details.

In the following we discuss some well-known characteristics of the *explicit features* from Section 5.4: average magnitude,[9] color,[10] and period that show their utility in predicting type of star, as well as the limitations in using only these features. Figures 5.17, 5.18, and 5.19 (pages 128 – 130) show histograms of the period, magnitude, and color for the OGLEII dataset. In Figure 5.17, we can see that there is a noticeable difference between the distribution of the RRL periods and the dis-

[8]Passbands refer to ranges of visible light; measurements of a star are taken using different filters on the telescope leading to measurements of brightness in different pass-bands.

[9]We use the standard V band for average magnitude, and subtract 0.52 from all magnitude measurements for stars in the SMC according to standard astronomy practice.

[10]The difference between average magnitudes in two passbands.

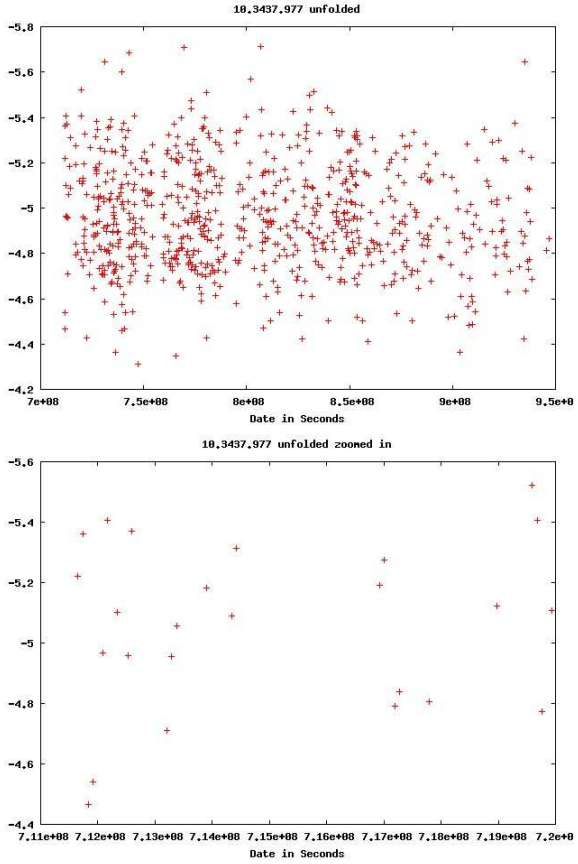

Figure 5.3: Unfolded time series from MACHO survey. Top: all data points. Bottom: expanded subsection.

tribution of the EB and Cepheid periods; while almost all of the RRL periods fall between $0.2d$ (d is days) and $0.8d$, the EB and Cepheid periods are spread out over a much larger range.

In Figure 5.18 (page 129) we can see that there is a lot of overlap in all three histograms. The peaks of the histograms are different, however, hence average magnitude does give a clue as to the type of star.

In Figure 5.19 (page 130) we show that all three types of stars have well defined narrow peaks, and there is some separation between EBs and Cepheids, with RRLs falling somewhere in the middle.

The histograms show us that we can expect well-defined ranges for each star type's color, magnitude, and period. In general, no one feature can be used to determine star type, but they do help to indicate which star type is more likely. When taken together, the features are even more informative, as we discuss below.

In the top of Figure 5.4, we examine the color and magnitude features together. There is a clear region for RRLs and Cepheids in this diagram, although the two regions overlap significantly; this space is sufficient to delineate many Cepheids and RRLs, but more information will be needed to distinguish those stars that fall in the overlapping region. EBs appear throughout the diagram, virtually encompassing the RRLs and Cepheids. Part of the EB region is distinct from the other two regions, and thus this feature space would be useful for data points that fall in that region. Overall, the color-magnitude space illustrates that the relationship between these two features can be used in many cases to distinguish star type.

In the bottom of Figure 5.4 (Color vs. Period), we see more overlap between the classes, but still there are autonomous regions for each star. The message here is that in the feature space indexed by period, magnitude, and color, we can distinguish star type for much of the data, however for the significant overlap regions we will have to analyze more carefully our predictive model. Furthermore, we can expect periodic variables to fall in a certain well-defined region of the feature space; this

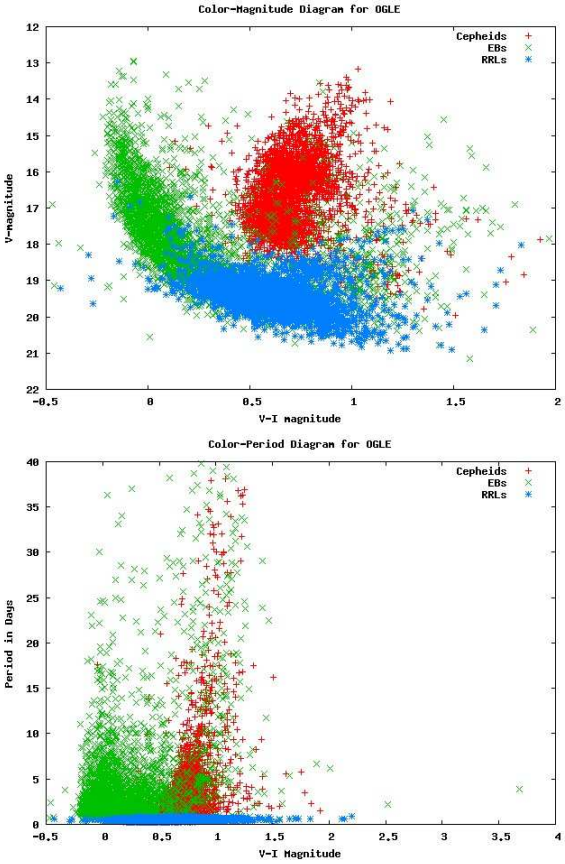

Figure 5.4: Top: Color-Magnitude diagram for OGLEII. Bottom: Color-Period diagram for OGLEII

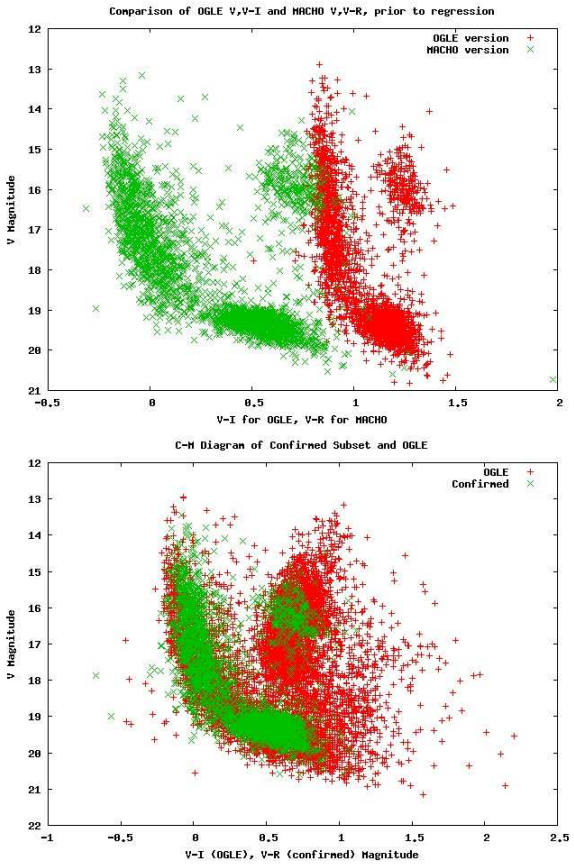

Figure 5.5: Top:Color-magnitude diagram for confirmed subset. The OGLEII version and MACHO version of each is shown here. OGLEII uses V-I and MACHO uses V-R; it is clear further calibration is warranted. Bottom: Calibrated color-magnitude diagram for confirmed subset. The MACHO version is shown here on top of the entire OGLEII dataset, after a regression model is learned and MACHO is calibrated accordingly.

will help us when we try to eliminate non-periodic variables from the raw MACHO data. As we illustrated above with the experiments on the OGLEII dataset, the features we list here are sufficient to be able to classify periodic variable stars quite well, although incorporating the shape of the time series improves performance. To be able to use the shape information, we must know the correct period of the star, which was determined for OGLEII by non-automatic methods. As we discuss below, determining period automatically is non-trivial, and we know of no system that accomplishes this task accurately.

The MACHO survey [Alcock et al., 1993] is a microlensing survey conducted in two non-standard passbands, "blue" and "red." After filtering out stars that have too many erroneous measurements, we are left with $25,309,792$ stars. The non-standard passbands present a challenge to making direct comparisons between stars in MACHO to stars in OGLEII. We use the formula given to us by P. Protopapas (2008, private communication) to convert the "blue" passband to Kron-Cousins V, and "red" to Kron-Cousins R:

$$V_k = B + 24.22 - 0.1804(B - R) \tag{5.12}$$

$$R_k = R + 23.08 + 0.1825(B - R) \tag{5.13}$$

where V_k, R_k are the Kron-Cousins V and R bands, and B, R are the non-standard MACHO bands. Additionally, we subtract 0.52 from the average magnitudes of stars in the SMC, as we did with OGLEII.

In order to evaluate our methodology in various stages, we use a subset of stars from the MACHO survey that also exist in the OGLEII catalog. We will refer to this throughout the document as the *confirmed subset*. To find these stars, we compare the RA and DEC[11] of the stars in OGLEII with stars in MACHO, and select stars such that:

$$\sqrt{(\mathrm{ra}_M - \mathrm{ra}_O)^2 + (\mathrm{dec}_M - \mathrm{dec}_O)^2} < 5s$$

where the units on the LHS are in seconds. Additionally, we require that the difference in V magnitude not exceed 0.5. This yields 3166 stars with 256 Cepheids, 1266 EBs and 1644 RRLs. These are very important because they allow us to track and validate different stages of our processing pipeline.

Even after the application of (5.12) we found that the V magnitudes and $V - R$ color in the confirmed subset were still too far off the V magnitude and $V - I$ color of their OGLEII counterparts; this is illustrated in the top plot of Figure 5.5.

[11] Right ascension (RA) and declination (DEC) are used to measure the position of stars in the sky.

We use a simple linear regression to map the MACHO V magnitudes to OGLEII V magnitudes, and MACHO $V-R$ colors to OGLEII $V-I$ color. In the bottom plot of Figure 5.5, we show the MACHO version of the confirmed subset on top of the entire OGLEII dataset; it is clear that the calibrated confirmed subset now sits in the same region as OGLEII in color-magnitude space. Note that we are not attempting for exact mapping, we simply want to ensure that any classification learned on OGLEII in color-magnitude space will transfer appropriately to the MACHO survey. As Figure 5.5 demonstrates, the mapping we choose should satisfy this requirement.

5.5.2 Classification Methodology

In this section we describe how we extract from the unlabeled MACHO catalog a set of labeled, periodic variable stars. We explain our methods and present experiments to show their effectiveness.

Our methodology is a pipeline of separate modules which we order as follows:

A Eliminate non-variables

- A1 sodset filter
- A2 check for sufficient number of points
- A3 check for sufficient variability

B Eliminate non-periodic variables

- B1 find periods
- B2 check for spurious periods due to sampling rate
- B3 refine period estimates
- B4 check for periodicity
- B5 check for symmetry

C Eliminate stars not of type Cepheid, EB, or RRL

C1 set aside based on nearest OGLEII star in C-M space

C2 set aside based on nearest OGLEII star using cross-correlation

D Classify

D1 train SVM on OGLEII dataset, classify remaining MACHO stars

D2 using confidences generated from the classifier output, abstain on any stars for which classification is indefinite.

The pipeline is illustrated in Figure 5.6. We explain each step in detail in the following sections.

SODSET (A1)

This is a pre-processing stage that does some basic filtering on the time series, eliminating completely any time series that have measurements out of a range determined by our collaborators.

Sufficient Number of Points (A2)

Before we perform any further testing, we eliminate light curves that have an insufficient number of observations. It is critical to eliminate light curves containing a low number of points because our tests for variability and periodicity will not give meaningful output for light curves that do not contain enough data to establish a pattern.

We start by eliminating points that have reported error more than 3σ away from the average error for that curve, i.e., if

$$e_i - \bar{e} > 3\sigma_e,$$

where e_i is the reported error of observation x_i, $\bar{e}$ is the average error of the time series, and σ_e is the standard deviation of the errors, then we throw away x_i. If the

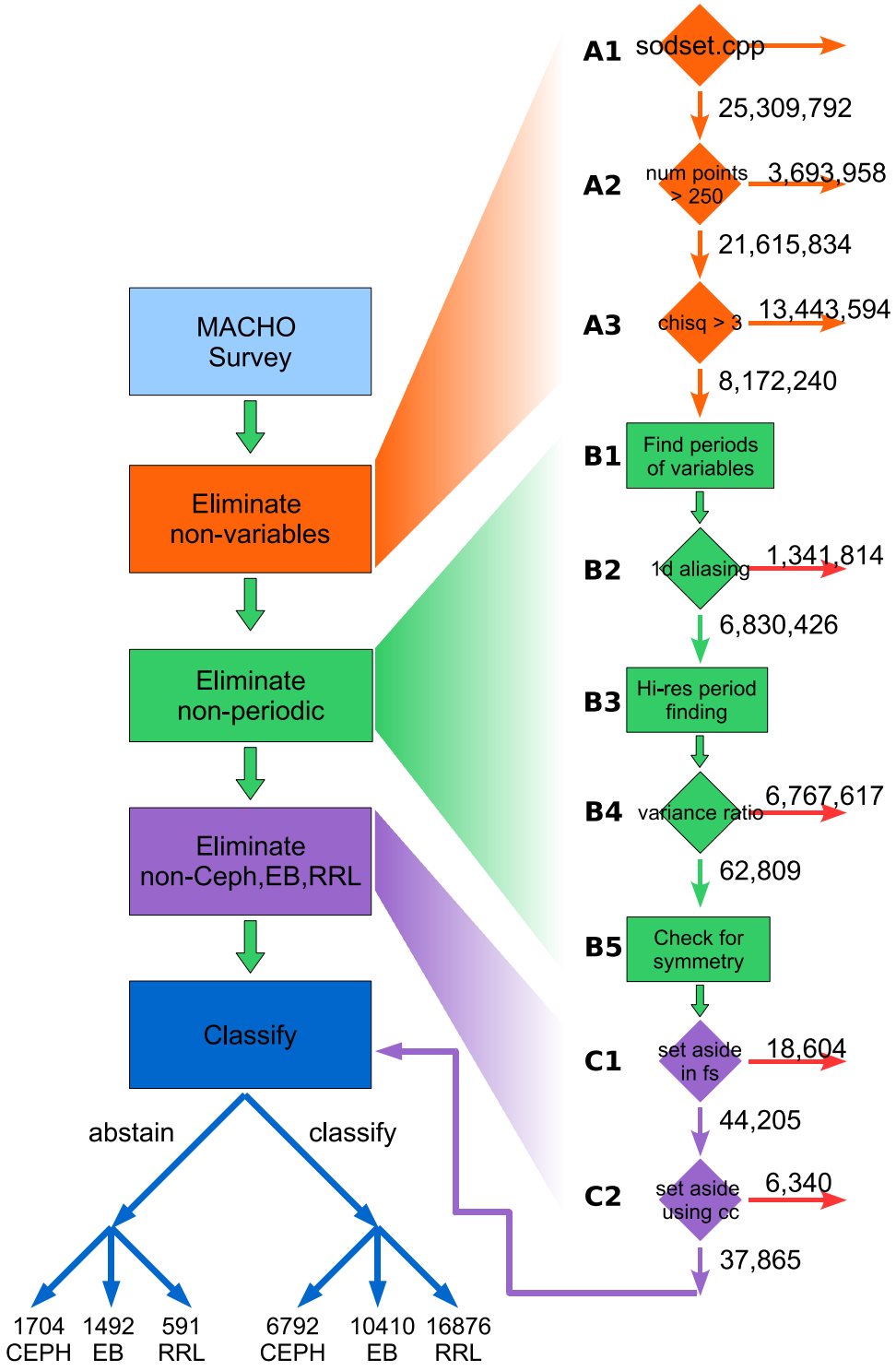

Figure 5.6: MACHO processing pipeline

either the red or blue band of the time series ends up with fewer than 250 points, we discard the star. This removes $3,693,958$ stars for future processing. Automated processing of stars with few and noisy measurements would require significantly improved techniques.

Chi-squared Test for Variability (A3)

In the next step, we use the chi-squared test to filter out non-variable stars. Specifically, we use a χ^2 test on the light curve for fit to the mean.

$$\chi^2 = \frac{1}{n-1} \sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{e_i^2}$$

where

$$\bar{x} = \frac{\sum_{i=1}^{n} (\frac{x_i}{e_i^2})}{\sum_{i=1}^{n} \frac{1}{e_i^2}},$$

i.e., the error-weighted average magnitude. If a star is variable, it will have a higher chi-squared value. We compute the χ^2 value for both red and blue bands in MACHO and use the maximum of the two for filtering purposes, that is, if only one of the bands is variable we want to include the star.

By discarding any star with $\chi^2 < 3$, we remove more than 50% of the survey which we believe are not variable. Furthermore, we do not eliminate any of the stars in the confirmed subset, indicating that, as intended, this test does not remove many variable stars. After this stage we have a total of $8,172,240$ stars remaining.

Finding Periods (B1-B5)

At this stage in the pipeline, we have established that the remaining stars are variable, however we have not yet determined if they are periodic. The problem of finding the period of a periodic, non-uniformly sampled time series has been studied extensively in general and in the astronomy literature [Reimann, 1994, Shin and Byun, 2004]. Yet while there are multiple methods for identifying the periodic com-

ponents of a light curve, there are no automatic methods for determining if a light curve is periodic. For example, if we give a non-periodic star to a period-finding algorithm, such as supersmoother [Friedman, 1984] or an analysis of variance (AoV) technique [Schwarzenberg-Czerny, 1989], it will return some period corresponding to a periodic component of the lightcurve. The fact that a periodic component is returned is no more indicative that the signal is periodic than is the existence of a Fourier decomposition of a uniformly sampled discrete signal. Methods must use some goodness-of-fit function to determine how well a given period describes the time series, however this is challenging in our context for the following reasons: the difference in goodness-of-fit between one period and an integer multiple of that period may be completely determined by scatter; the "true period" of an Eclipsing Binary may be impossible to describe mathematically with the data we have; semi-regular sampling times lead to spurious periodic components. These points and techniques to address them are discussed in detail below. The work of Reimann [1994] gives a thorough comparative study of the period finding methods in use in the context of periodic variable stars, and shows that while some perform better than others in some cases, no one method is perfect.

To summarize, state-of-the-art performance requires a human to verify that the star is in fact periodic, and that the period-finder has returned the true period; no currently available method performs a truly automatic, reliable period-selection, and as mentioned above, with EBs, such a method may not be possible with just the time series as input. The method we use is also imperfect and it is not as accurate as manual period selection, but it gives sufficient performance so that our classifier can make accurate predictions.

Methodology The period methods we examined operate using the same basic algorithm: for a given list of periods, use some statistic to rank the periods according to how well they fit the data. The Analysis of Variance (AoV) based techniques [Schwarzenberg-Czerny, 1989] bin the periodic signal and compare variance-

like statistics within each bin to variance-like statistics over the whole signal. Phase Dispersion Minimization (PDM) [Stellingwerf, 1978] reports the period minimizing the variance (scatter) of the folded signal. The Supersmoother Algorithm [Friedman, 1984] uses a sliding window of variable width, and calculates the squared error of each point with the average of the window around that point; the periods are then ranked according to this error. The entropy minimization method [Cincotta et al., 1995] forms a two-dimensional grid over the folded light curve, the resolution of which is specified by the user. The number of points in each square of the grid is used to estimate the "entropy" of the signal. In other words, if the signal is pure noise, the entropy will be high because the points will be distributed uniformly throughout the grid. If the signal is periodic, the entropy will be low because few squares will contain all of the points. The method we describe uses as its center the Lomb-Scargle (L-S) Periodogram [Lomb, 1976, Scargle, 1982]. The L-S Periodogram rates periods based on the sum-of-squares error of a sine wave at the given period. There are numerous studies comparing the period evaluation techniques we describe here as well as many others [e.g., Schwarzenberg-Czerny, 1989, Reimann, 1994]. No one method works perfectly in every situation. We choose L-S because it can be evaluated efficiently using the algorithm of Press and Rybicki [1989], and in preliminary experiments it has shown to perform reliably.

In our method, we first calculate a Lomb-Scargle Periodogram [Lomb, 1976, Scargle, 1982] (step B1 in Figure 5.6) for each light curve; we look for periods in the range of $0.1d - 20d$, where d is days, with an oversampling rate of 0.5, meaning the range of periods is broken into even segments of 0.5 times the reciprocal of the Nyquist frequency of an equivalent uniformly-sampled time series. The implementation of the L-S Periodogram is that of Hartman et al. [2008], and makes two important additions to the original algorithm. First, local maxima in the periodogram are represented by the peak period. In other words, periods immediately to the left and right of a local maxima will not be reported in the periodogram.

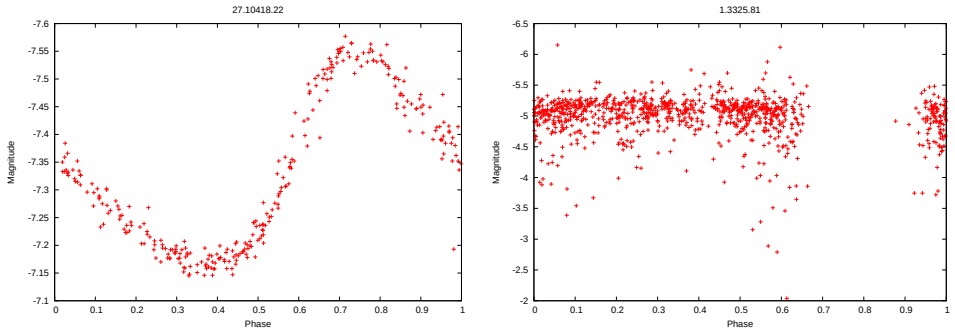

Figure 5.7: Two stars with predicted periods close to 1d, shown folded and plotted by phase. The left has a true period of about 1d, the right is not periodic but has a strong 1d frequency component due to the sampling frequency.

Second, when a period is reported, simple rational number multiples of that period are not reported. We use the highest power period as our first guess of the period of a light curve. We refer to the period giving the highest peak in the periodogram as the highest power period.

Many stars in the MACHO survey were observed with a sampling frequency around $1d$ because some stars are observed at roughly the same time of day for each observation. Therefore we expect these stars to have strong periodic components around $1d$. Looking at Figure 5.17, however, we expect many stars to have a true period close to one day, hence we need to find a way to separate the spurious $1d$ period guesses from the correct $1d$ period guesses (step B2 in Figure 5.6). We use a simple method to accomplish this that we illustrate with an example. In Figure 5.7 we show two light curves that have been folded to a period of $\sim 1d$, corresponding to the highest power period given by the Lomb-Scargle periodogram. It is clear that the right light curve has been folded to a sampling frequency by the gaps, or discontinuities in the folded light curve, but the left light curve actually has a true period close to $1d$. Therefore, we need some way to figure out if our period guess is the true period, or a result of the sampling frequency. To accomplish this, we first check if the guessed period, p, is within 1% of $1d$; if it is, we move a sliding window of width $0.1p$ (remember that the light curves are first folded to the period p) along

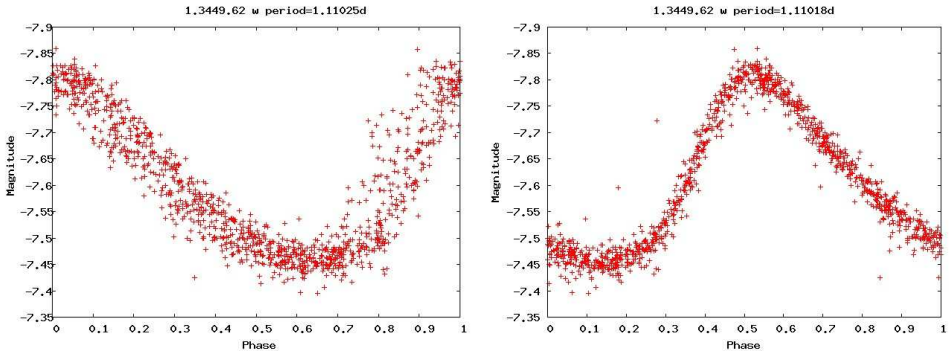

Figure 5.8: The same star folded according to low-resolution period search and high-resolution period search. Note the reduction in scatter in the high-resolution version.

the folded light curve, and record the number of points in each corresponding time window of width $0.1p$. We estimate the expected number of points in such a slice to be $0.1n \pm \sqrt{0.1n}$, where n is the total number of points. We say a light curve has been folded to a period corresponding to the sampling frequency if the window with the fewest number of points differs from $0.1n$ by more than $3\sqrt{0.1n}$, i.e., if the number of points in any window is more than 3 standard deviations from the expected mean. If this test indicates that the current period guess is in fact due to the sampling frequency, we check the next highest period given by the L-S Periodogram, and test the folded light curve as with the first period guess, otherwise we leave the current period guess as is. If we do test the second period and it also indicates it is a result of the sampling frequency, we reject the light curve entirely, otherwise we record the second frequency as the true frequency. This stage of processing eliminates $1,341,814$ stars, leaving $6,830,426$ (see Figure 5.6).

Now that we have filtered out erroneous periods derived from the sampling-frequency, we re-run the L-S Periodogram using a higher oversampling rate of 0.05 and a range of $p \pm 0.1p$ where p is the current period guess (step B3 in Figure 5.6). This is to eliminate as much scatter in the folded light curves as possible, as a slightly incorrect period guess can lead to a very noisy folded light curve. We could have run with this resolution initially, however it would have been too resource intensive

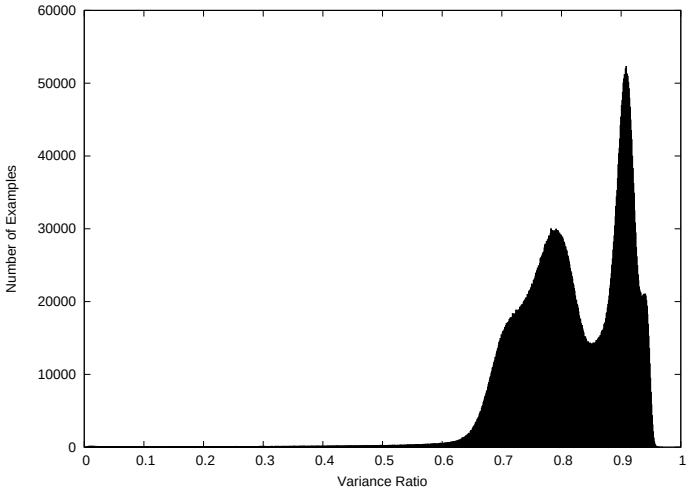

Figure 5.9: Histograms of variance ratios of MACHO stars.

to do so over the original period range. In Figure 5.8 we show the same light curve folded once to the period found initially, and then to the period resulting from the high-resolution period search. The right curve has significantly less scatter. As scatter tends to obscure any shape, this will help us in the next stage when we try to determine if a light curve has a "shape", and later on when we try to determine if two light curves have a similar shape.

The next challenge is determining if the light curve is actually periodic (step B4 in Figure 5.6). As we mentioned above, the L-S Periodogram will return a list of periods for any light curve, regardless of whether it is periodic, and the p-values (the probability that the signal is non-periodic) given by the L-S Periodogram for periodicity have not proved to be reliable in this context. We use a statistic we call the *variance ratio* as a method of determining whether a folded light curve is periodic. This statistic is the same as the one used in Stellingwerf [1978] to determine how well a period matches a given signal; hence we are using the L-S Periodogram to find candidate periods, and a part of Phase Dispersion Minimization to evaluate periodicity. The intuition is that if the folded light curve has a *shape*, the variance of a small window of points should be small compared to the variance of the entire

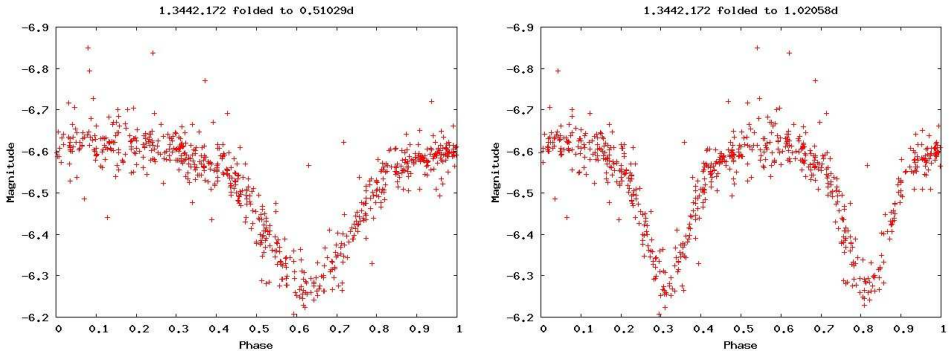

Figure 5.10: An Eclipsing Binary for which LS returns 1/2 the period, folded according to the LS period and twice the LS period. Here is it difficult to formalize that one is "better" than the other.

light curve, whereas if the light curve is just noise, the variance will be uniform regardless of the size of the window. We fold each light curve to the current period guess. We then calculate the average variance of a sliding window around the mean of the sliding window. Finally, we divide the average variance by the variance of the entire light curve. If this variance ratio is close to 1, we will tag the star as non-periodic and throw it away because this means the local variance is the same as the global variance. If this falls far enough below 1 we accept it as periodic.

Formally the variance ratio is defined as:

$$VR = \frac{\sum_{i=1}^{n}(x_i - \overline{x}_i)^2}{\sum_{i=1}^{n}(x_i - \overline{x})^2}$$

where n is the number of points in the folded light curve,

$$\overline{x_i} = \frac{1}{w}\sum_{j=1}^{w/2} x_{i-j \bmod n} + x_{i+j \bmod n},$$

and $w = \min\{10, 0.05n\}$ is the window size. In Figure 5.9 we give a histogram of the variance ratios of the MACHO stars that passed the previous filtering steps. By using a threshold of 0.7, the variance ratio test removes $6,767,617$ stars that we believe are variable but not periodic.

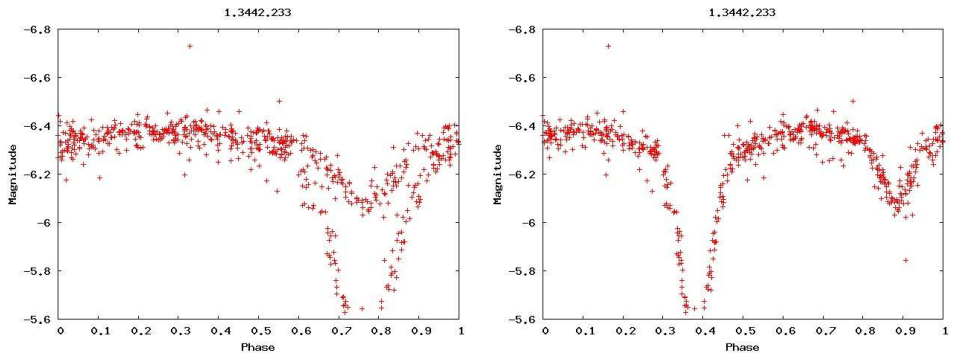

Figure 5.11: The same star folded first by using the output of the LS periodogram, then by using our method of checking for symmetry in twice the reported period.

We have the remaining problem of determining if the guessed period is the "true" period or a simple rational number multiple (harmonic). In our experience, the L-S Periodogram almost always gives the correct period for RRLs and Cepheids, but almost always gives 1/2 the true period for EBs. Finding the true period of an EB is particularly challenging. First, it is impossible to distinguish mathematically between the true period and half the true period of a symmetric Eclipsing Binary (see Figure 5.10). Second, methods that are better able to identify the true period of EBs, such as Supersmoother [Friedman, 1984], are prone to find periods that are integer multiples of single bump stars like RRLs and Cepheids; methods that fold RRLs and Cepheids correctly, such as the first period given by the Lomb-Scargle Periodogram [Lomb, 1976, Scargle, 1982] often give 1/2 the "true" period of EBs. We can compensate for this somewhat by using one extra check; this is step B5 of Figure 5.6. After the high-resolution period search, we fold each light curve to double the current period guess, smooth it by replacing each point with the mean of the 10 adjacent points, linearly interpolate the folded, smoothed curve to have 1024 equally spaced points, subtract the mean and divide by the standard deviation. We then take the Euclidean Distance between the vector of the first 512 points and the vector of the final 512 points. This is an estimate of how symmetric the curve is. If the true period is the original guess, then folding the light curve to double this

Table 5.7: Comparison of period finding methods on OGLEII. Here we assume that the periods reported in OGLEII are correct and show performance of other algorithms relative to the OGLEII periods.

	Correct	Integer Multiple	Error
B3	10108	972	1175
B5	10637	660	958

period should give two almost exactly symmetric halves, and hence there will be a very small distance between them. If the original guess was half the period of an EB with one large bump and one small bump, then this test will show that there is a significant distance between the two halves. By using this method we are able to correct many of the original period guesses for EBs, while making few errors on RRLs and Cepheids. In Figure 5.11 we show an example where the LS periodogram gave 1/2 the true period of an EB, and our method was able to correct for that. Observe that this method does not help when the EB is exactly symmetric, or when the L-S Periodogram returns a multiple of the true period other than 1/2. It is these cases in which our periods will be incorrect, yet as we show below our classification system is still able to perform well.

We can immediately evaluate our period-finding method by comparing it against the actual periods reported by OGLEII, and the periods we would use if we did not check for symmetry. The OGLEII dataset is first filtered through steps A1-B5 of the pipeline thus far, in order to remove any potentially noisy data. We denote our method by B5 (according to its position in Figure 5.6), and the method that does not check for symmetry by B3 (also see Figure 5.6). In Table 5.7 we report discrepancies between the actual periods and the two automatic methods on the OGLEII dataset. For each period reported, we check if it is within 1% of the actual period and if so record it as "correct." We then show two types of errors: if it is within 1% of an integer multiple of the period, and otherwise. B5 makes fewer errors than B3, and fewer mistakes between the true period and an integer multiple. This

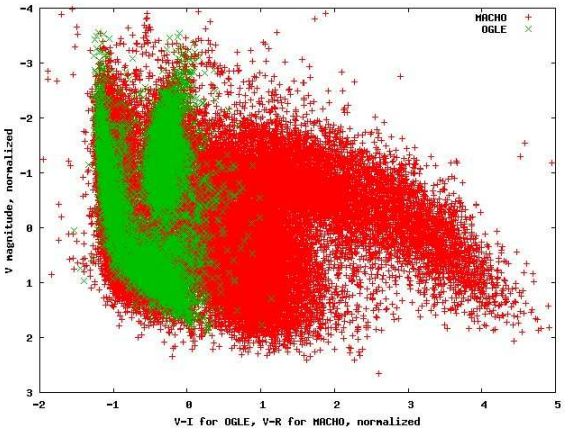

Figure 5.12: Color-magnitude diagram of stars in MACHO that have passed the periodic-variability test with known periodic-variables from OGLEII

is a small but significant improvement in the quality of the reported periods.

Selection (C1-C2)

The 62,809 remaining stars are periodic variables, and now we need to identify stars that we do not believe to be one of RRL, Cepheid, or EB. This is important because the classifier we use in the following section is trained to distinguish between these three classes but has no model of unknown types and thus cannot make good predictions on a star that is not Cepheid, EB or RRL. The stars that we identify as non-Cepheid EB or RRL will be interesting astronomical events, such as period-changing variable stars, and should be held for further review by a domain expert. We identify stars for further review in two ways: first we set aside any stars that do not have "near neighbors" from the OGLEII dataset in color-magnitude space (step C1 in Figure 5.6); second we set aside stars that do not "look like" any stars in the OGLEII dataset (step C2 in Figure 5.6).

In the first stage, (C1 in Figure 5.6), we simply embed each star in the two-dimensional space of average magnitude (denoted $\overline{V}$), and color (denoted $\overline{V} - \overline{I}$), where these features have been calibrated as mentioned in Section 5.5.1. Additionally, we join the OGLEII and MACHO datasets and normalized the features such

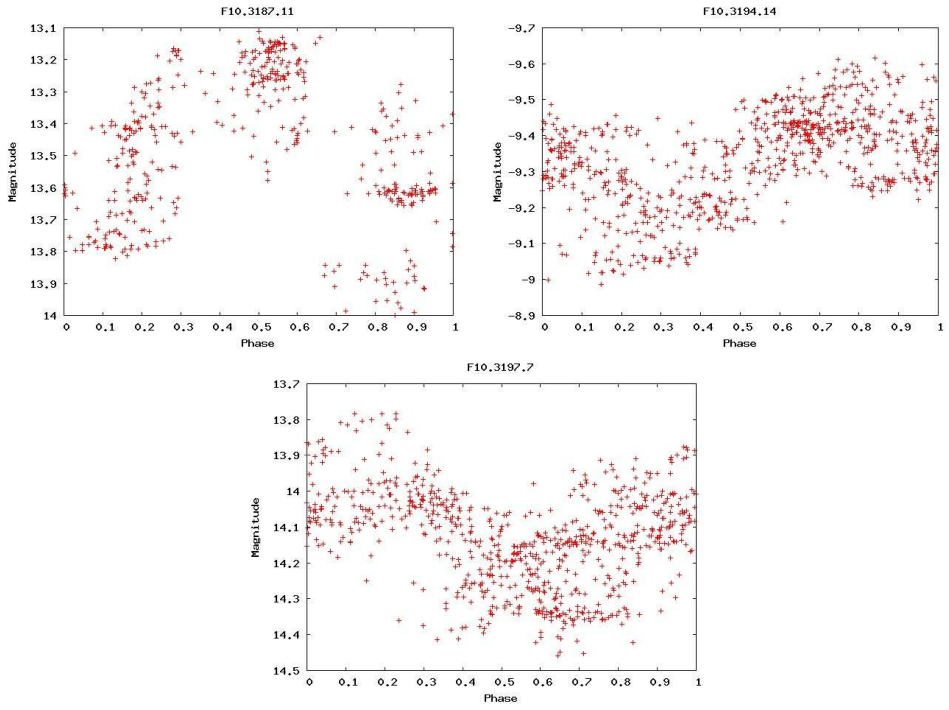

Figure 5.13: Selected for review due to distance in C-M space

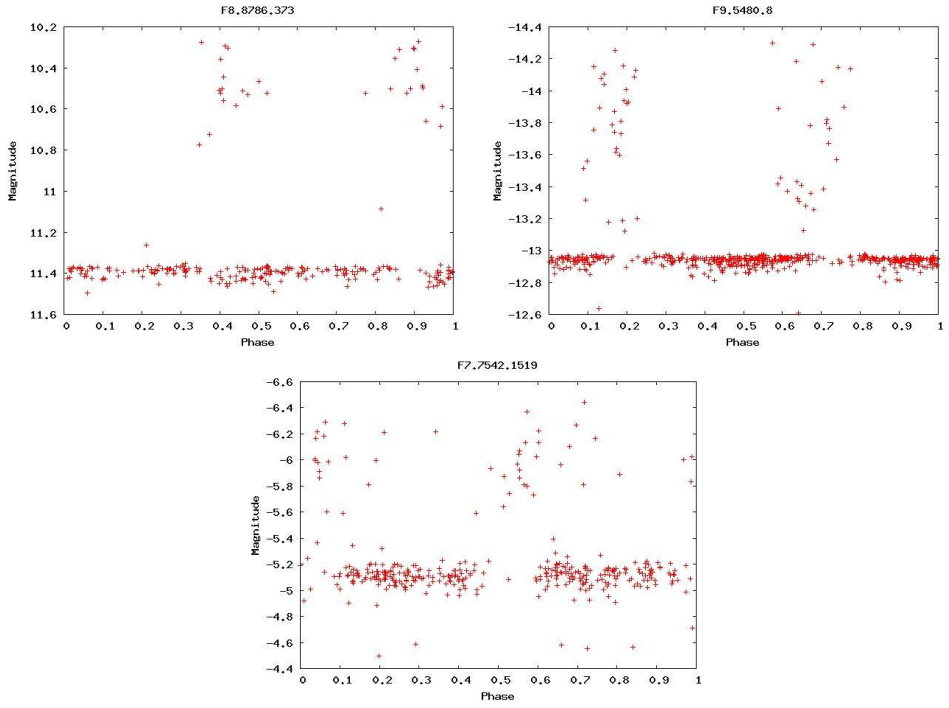

Figure 5.14: Selected for review due to cross-correlation

that they have standard deviation of 1 and mean of 0. We then use the Euclidean Distance of each star in MACHO to its nearest neighbor from the OGLEII dataset to generate a similarity measurement, e^{-d}, where d is the Euclidean Distance to the nearest neighbor. We then set aside any stars that are far away from any star in OGLEII, i.e., if $e^{-d} < 0.9$. As can be seen in Figure 5.12, it is clear that a large number of stars that have passed the periodic-variable test in MACHO are well outside the normal range of color and magnitude for EBs, Cepheids, and RRLs, as established by OGLEII. In this portion we remove an additional $18,661$ stars, including 82 from the confirmed subset. In Figure 5.13 we show three stars that are set aside in this phase; such stars are of interest but require future investigation before they can be classified.

In stage C2 of Figure 5.6, we use the cross-correlation similarity measure from Section 5.1 to quantify how "alike" two light curves are in shape. We first fold, smooth, and interpolate each light curve to 1024 points as in Protopapas et al. [2006], and scale each light curve to have a mean of 0 and standard deviation of 1. We then compute (5.1) for each light-curve in MACHO with each light curve in OGLEII. If an unknown star does not "look like" any star in OGLEII, we set it aside as it is most likely not a Cepheid, EB, or RRL.

In this phase we set aside star Y if $C(X, Y) < 0.97$, where X is the star in OGLEII giving the highest cross-correlation, $C(,)$ as in (5.1). In Figure 5.14 we show typical examples of periodic-variable stars that were set aside in this phase; these light curves clearly represent some event, but require a different method for classification. In total we remove $13,854$ stars among which 98 are in the confirmed subset.

Classification

With our classifier we wish to capture both the aggregate information about each star using the explicit features from Section 5.4, as well as the shape of the folded

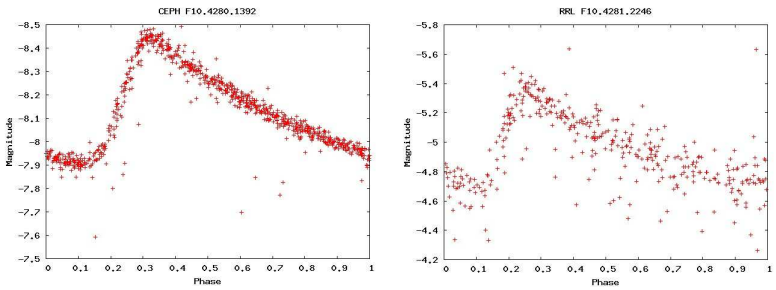

Figure 5.15: A Cepheid and RRL with similar shape.

light curve. This shape is critical in identifying stars, as we have shown in Section 5.4, and is one of the primary points of input for an astronomer. Recall the typical shape of the three types of stars from Figure 5.1. It is clear that shape can effectively delineate these prototypical examples of each class by shape. In Figure 5.15, however, we see a Cepheid and RRL that look very similar — in this case we would need to rely more on the explicit features in order to distinguish the stars. We illustrate two clean-cut examples here, but in general the decision is much harder and we use machine learning methods (primarily SVM) to induce a decision function.

5.5.3 Background and Preliminary Tests of OGLEII

We use *features* + K from Section 5.4 as our similarity measure of light curves. Recall this combines (5.9) from Section 5.2 with the explicit features.

We choose the value 15 for the constant γ in (5.9) by 10-fold cross-validating on the OGLEII dataset with each value of γ in $\{1, 5, 15, 20, 25, 50, 80\}$ and using the value that gives the highest accuracy.

Because we are building an automatic system, we cannot rely on human intervention to find periods. We can use the astronomer-found (OGLEII) periods on the training set, and the periods found by method B5 on the test set, or we can choose to use the B5 periods on both the training and test set. We evaluate both options by cross-validating on the OGLEII dataset. We compare the results to the

Table 5.8: Top Left: Cross validated results on OGLEII using known periods. Top Right: Cross-validated results on OGLEII using periods determined by method B5. Bottom: Cross-validated results on OGLEII, training on periods from OGLEII, testing on periods from method B5. Actual labels are the columns, predicted labels are the rows.

	Ceph	EB	RRL
Ceph	3095	16	7
EB	22	2069	3
RRL	4	3	7036

	Ceph	EB	RRL
Ceph	3078	64	13
EB	30	2018	6
RRL	13	6	7027

	Ceph	EB	RRL
Ceph	3105	340	10
EB	2	1440	7
RRL	14	308	7029

"ideal" case, i.e., using the astronomer-found periods on both training and testing. We show the results in Table 5.8. As expected, using the OGLEII periods for both training and testing (this is an identical experiment to that reported in Section 5.4) gives the best results but this method is not automatic. Using the method B5 on both the training and test set performs much better than training on the OGLEII periods and testing on the B5 periods, and almost as well as training and testing on OGLEII.[12] We will therefore use this method to classify MACHO.

Once we have classified each example, we would like to have some estimate of the confidence in the prediction. In Chapter 6, we explore in depth several methods that attempt to derive class-membership probabilities from classifier output. In this case, we do not need probabilities but a ranking of confidence in each example, nonetheless we can use a probability generating routine to give us this ranking. Although we found the raw confidences to be sufficient in Section 5.4, by running on the confirmed subset with our period estimates, we found that the method from (6.7) in Section 6.1.2 gave better ordering of the output than the raw confidences and thus we use that method here. It is possible that the less reliable period estimates

[12]Using the same uncertainty in both training and testing is related to Quinlan [1989], which gives an empirical study supporting the use of unknown attributes in both the training and test sets.

Table 5.9: Cross-validation results on OGLEII using periods from B5. Left to right, top to bottom: abstaining on none, lowest 1%, 5%, and 10%. The fourth row is the number abstained by category.

	Ceph	EB	RRL
Ceph	3078	64	13
EB	30	2018	6
RRL	13	6	7027
Abstain	0	0	0

	Ceph	EB	RRL
Ceph	3041	60	4
RRL	21	1994	4
EB	4	5	6999
Abstain	55	29	39

	Ceph	EB	RRL
Ceph	2823	49	1
RRL	18	1935	2
EB	0	1	6813
Abstain	280	103	230

	Ceph	EB	RRL
Ceph	2562	38	0
RRL	17	1863	2
EB	0	0	6547
Abstain	542	187	497

are causing the simple normalization to perform more poorly, justifying the use of a different method. We do not interpret the output directly as a probability, we use the output of this method only to rank the predictions. We can now opt to abstain from classifying the lowest, for example, 10% of examples. In Figure 5.9 we show confusion matrices for cross-validation on OGLEII using the B5 period finding method, and abstaining at various thresholds. The confusion matrices show that we can tune the abstention threshold so that we eliminate some classification errors without eliminating too many good classifications.

5.5.4 Classifying Stars from the MACHO survey

Prior to training the SVM on the OGLEII data, we filter it the same way we filtered MACHO, except that we do not set aside any stars as in Section 5.5.2. This is to ensure that we eliminate any noisy or potentially confusing data points, or stars with periods outside of our period search range. After filtering, we are left with 12255 stars of the original 14087.

Finally we train the Support Vector Machine (SVM) [Boser et al., 1992] using $K(X, Y)$ as in (5.9) and periods as determined by method B5 on the filtered OGLEII dataset and test on the remaining $37,865$ MACHO data points.

In Table 5.10 we show the confusion matrices for several abstention thresholds.

Table 5.10: Confusion Matrices for classification on MACHO using abstention thresholds of 1 (none), 0.99, 0.95, 0.9 going left to right, up to down.

	Cepheid	EB	RRL	Unknown
Cepheid	247	8	2	8136
EB	0	1107	1	10586
RRL	0	4	1637	16075
Set aside	9	167	4	32335
Abstained	0	0	0	0

	Cepheid	EB	RRL	Unknown
Cepheid	247	8	2	7974
EB	0	1106	1	10455
RRL	0	4	1637	15990
Set aside	9	167	4	32335
Abstained	0	1	0	378

	Cepheid	EB	RRL	Unknown
Cepheid	247	5	2	7313
EB	0	1103	0	9869
RRL	0	3	1637	15733
Set aside	9	167	4	32335
Abstained	0	8	1	1882

	Cepheid	EB	RRL	Unknown
Cepheid	246	1	1	6512
EB	0	1097	0	9134
RRL	0	3	1637	15391
Set aside	9	167	4	32335
Abstained	1	18	2	3760

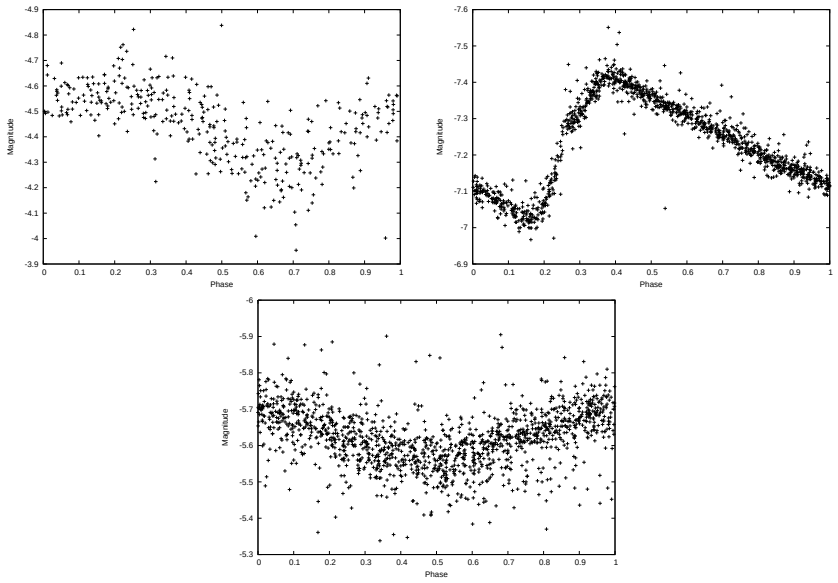

Figure 5.16: Light Curves with lowest confidence prediction among stars that pass all filtering stages.

The first three columns are the actual labels according to the confirmed subset; the first three rows are the labels produced by our classifier; the fourth column is the unknown MACHO stars; the fourth row is the number of stars we set aside in Section 5.5.2; the fifth row is the number of stars on which we abstain from making a prediction based on the given abstention threshold.

5.5.5 Discussion and Analysis of New MACHO Catalog

Through private communication with Dr. Protopapas at the Harvard-Smithsonian Center for Astrophysics, we gained access to a subset of labeled stars in the MACHO catalog. We do not present the details of the labels here, but claim the following new discoveries which are not in the set of labels. We present 8045 new Eclipsing Binaries, 6070 new Cepheids, and 5657 new RRLs. Because this is the first run of our automatic system, we had our domain-expert verify the classifications by examining each of the classified time series. His estimate was that our catalog is approximately 95% correct. As there is no unambiguous method for determining the true label, we cannot know the true error rate. To give an example of some of the

lowest confidence predictions, in Figure 5.16 we show light curves from stars that had the lowest confidence in their prediction among the stars that were classified (i.e., made it through all prior filtering steps).

The discoveries presented here have significant implications for the astronomy community. Our results are currently being investigated by astronomers and conclusions are forthcoming.

There are open problems originating in this research. It would be interesting to explore new ways of finding period automatically. Whether it is possible to do this as well as a domain expert in this context is unknown, and it would be useful to formalize this mathematically.

Every step of the filtering pipeline rejects some known good stars, while including some seemingly non-periodic variables. Future improvements to the various stages should focus on improving the model that rejects stars as either non-periodic or non-variable.

5.5.6 Additional Figures

This section includes large figures discussed previously in this chapter that have been placed here to minimize disruption of the written material.

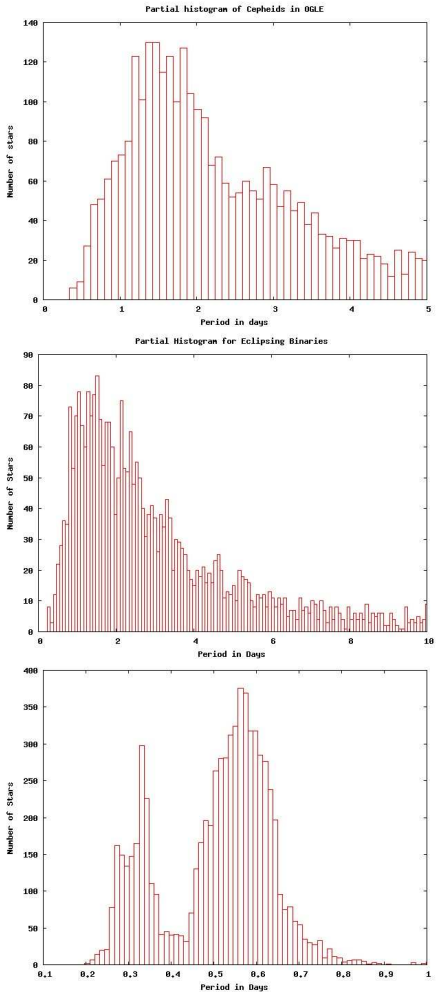

Figure 5.17: Histograms of periods for stars in OGLEII dataset. From top to bottom: Cepheids, EBs, RRLs.

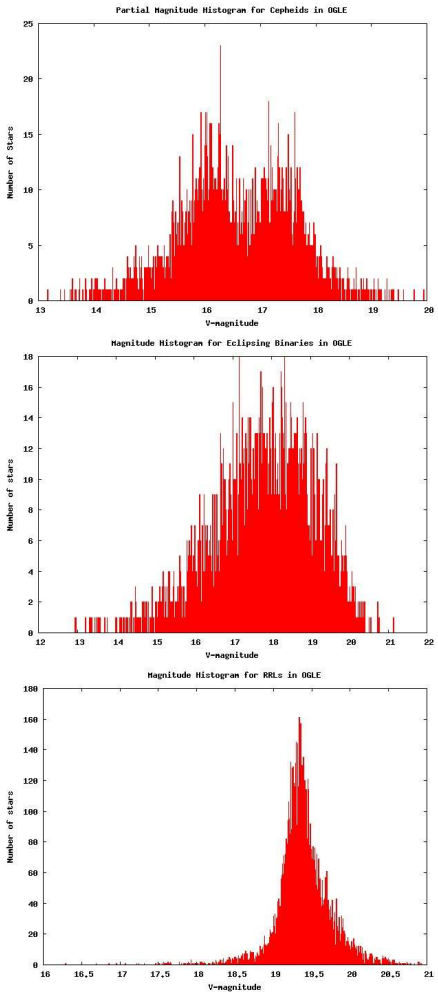

Figure 5.18: Histograms of V-mag for stars in OGLEII dataset. From top to bottom: Cepheids, EBs, RRLs. Magnitudes for stars in the SMC have been corrected by subtracting 0.52.

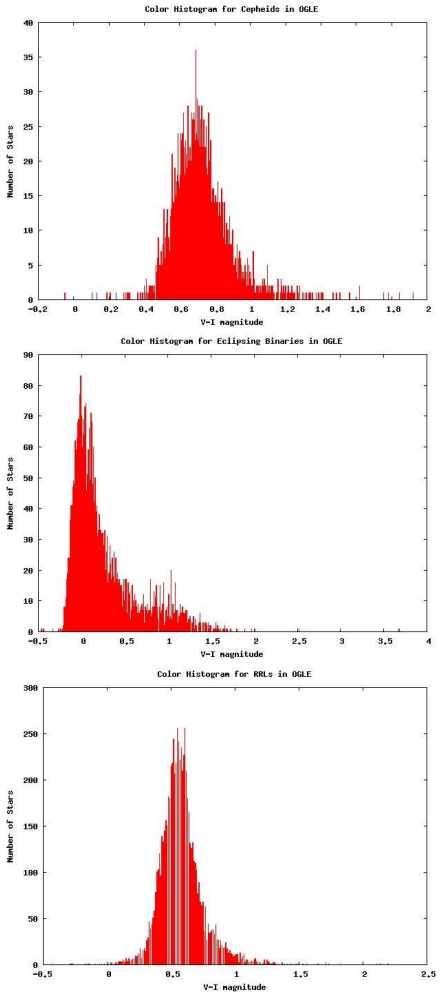

Figure 5.19: Histograms of V-I for stars in OGLEII dataset. From top to bottom: Cepheids, EBs, RRLs.

Chapter 6

Generating Confidences from Classifier Output

In the previous chapters, we have discussed various methods for classifying data, specifically structured data, in a variety of settings. The primary focus there was to overcome obstacles inherent in the classification process: capturing information inherent in structure (Chapters 4 and 5), or overcoming noise in the data set (Chapter 3). Now we turn to another problem related to classification that begins when the classification is finished: how to estimate the reliability of classifier output.

In Chapter 5, during the "abstention" phase (Section 5.5.2), we were concerned with finding a ranking of the output examples according to how confident the classifier was with the predicted label. In that context, we used raw output in Section 5.4 and estimates of probabilities in Section 5.5.3 together with a cutoff based on the training set. It is clear that it would be useful in such a setting to be able to estimate accurately the *probability* that an unknown example belongs to a certain class, that is, the probability that a data point x has the label i:

$$P(y = i|x) \tag{6.1}$$

where as in Chapter 1 we use y to indicate the label of a data point x. We will often refer to (6.1) as *class conditional probability* or CCP. Then, instead of estimating the threshold, we could simply reject examples based on a concrete estimate of the reliability of the label.

Some classification algorithms, such as logistic regression [Bishop, 2006], Random Forests [Breiman, 2001], and various forms of Boosting [Freund and Schapire, 1996], do in fact produce estimates of class-conditional probabilities, that is a value for (6.1) based on an assumed model. Others, such as Support Vector Machines [Boser et al., 1992], Perceptron [Rosenblatt, 1958], and Winnow [Littlestone, 1987], estimate only the decision boundary between to classes and do not provide probabilistic output. There is ample recent work, however, that addresses the problem of estimating class-conditional probability from, e.g., SVM output [Mease et al., 2007, Wu et al., 2004, Platt, 1999, Niculescu-Mizil and Caruana, 2005]. For the remainder of this chapter we will use SVM as the representative of classifiers not generating class-conditional probabilities.

In this chapter we explore the effectiveness of several methods for generating class-conditional probabilities in an experimental setting. We use logistic regression as an example of a classifier naturally producing class-conditional probabilities. In addition, we compare these to various methods, some new, for producing class-conditional probabilities from SVM and kNN output. In our analysis we attempt to discern what, if any, performance difference exists between different algorithms, and in what settings. Finally, we highlight new avenues of research based on the questions raised by the experimental results.

6.1 Background and Related Work

6.1.1 Classifiers Giving Probabilistic Output

Probabilistic generative classifiers attempt to model all the aspects of the distributions governing the classification task namely $P(x|y)$, the probability of an example being generated by a specific class, and $P(y)$, the probability that a given label will occur. Using Bayes' Rule, $P(y|x)$ can be calculated, and hence we have our class-conditional probabilities. *Probabilistic discriminative* classifiers, such as logistic regression, attempt to model $P(y|x)$ directly by specifying a parametric form of $P(y|x)$ and then finding a maximum likelihood or MAP solution (see for example [Bishop, 2006]). In this chapter we discuss only probabilistic discriminative models, and leave a discussion of generative models and comparisons thereof for future work.[1]

Logistic regression (LR) is a probabilistic discriminative classifier. That is, LR does not attempt to build a full probabilistic model, but constructs a discriminative classifier that has some probabilistic motivation or interpretation. In the case of LR, the assumption is that

$$a = \log \frac{p(x|y=1)p(y=1)}{p(x|y=2)p(y=2)}$$

is a linear function in terms of the input data.[2] As a linear function is equivalent to a hyperplane in the input space, LR uses the same class of underlying functions to assign class labels as SVM (and all other linear classifiers). The difference is in the motivation governing how the hyperplanes are built.

The probabilistic interpretation does not always lead to the best classification performance. In theory, perfect probabilistic knowledge would give optimal classi-

[1] For a detailed discussion of generative and probabilistic discriminative see Chapter 4 of Bishop [2006].

[2] This follows from a the assumption that the $P(x|y)$ are exponential in form. For a detailed explanation see Chapter 4 of Bishop [2006].

fication, but this is not the case in practice due to the assumptions made in the probabilistic models. Hence in the rest of the thesis, we use SVM and kNN because of their excellent classification ability. We would like to have the best of both worlds and in this chapter we explore the possibility of generating good estimates of (6.1) from SVM and kNN output. We will continue to use LR as a point of comparison, but will focus on SVM and kNN.

6.1.2 Methods for Generating CCPs from SVM

In this section we describe various methods for generating class-conditional probabilities from multi-class SVM output. In addition, we discuss some relevant theoretical results pertaining to probabilistic interpretations of SVM output. We focus on SVM here in the interest of keeping the discussion coherent and concrete, and because most of the work we reference does likewise, yet it is easy to see that many of the methods can be applied in general to other algorithms. We will use the notation in Table 6.1 and we assume $r_{ij} + r_{ji} = 1$.

Table 6.1: Notation for probability discussion.

Variable	Meaning
p_i	$P(y = i\|x)$
$\mathbf{p}$	$(p_i)_{i=1...k}$
μ_{ij}	$P(y = i\|y = i \vee y = j)$
r_{ij}	Estimate of μ_{ij}
k	Number of classes
I_e	Indicator variable for event e

Methods for Binary Classification

We first review methods for binary classification problems that are not trivially extensible to the multi-class case. In a later section of this chapter we discuss various binary-to-multiclass extensions for probabilistic classifiers and show how to apply the methods in this section in the multi-class setting.

Platt Scaling [Platt, 1999], like logistic regression, is founded on the assumption that the output of the classifier is the log-odds of class membership. In this case, however, the output is from SVM, and the assumption is not theoretically justified [Bartlett and Tewari, 2004, Hérault and Grandvalet, 2007].[3] Platt [1999] justifies the assumption in part based on empirical evidence that a parameterized sigmoid of the form

$$\frac{1}{1 + e^{Af(x)+B}} \tag{6.2}$$

where $f(x)$ is the output of SVM on example x, fits the output of SVM well. This method estimates A and B using a maximum likelihood solution where the true $p(y = 1|x)$ is estimated by 1, and $p(y = -1|x)$ by 0. No multi-class extension is given by Platt [1999], however as we discuss below it is still possible to apply Platt Scaling in the multi-class setting.

Isotonic regression [Zadrozny and Elkan, 2002], like Platt Scaling, takes as input the output of an already-learned classifier such as SVM. The output is sorted according to the value of $f(x)$ given by the SVM (i.e., the raw output), and each example is given a preliminary value of 0 for $p(y = 1|x)$ if $y = 0$, and 1 for $p(y = 1|x)$ if $y = 1$. The algorithm then looks for regions in which the $p(y = 1|x)$ are not sorted, and uses the average value of $p(y = 1|x)$ in the un-sorted region as the new value of $p(y = 1|x)$ for all data in the region. New values of $f(x)$ for an unknown point are then estimated by binning the new value to the calibrated values. As with Platt Scaling, isotonic regression is inherently a two-class method, however it can be extended to the multi-class setting using the methods we discuss below. Zadrozny and Elkan [2002] show that isotonic regression performs better than Platt Scaling in some experimental contexts.

Caruana and Niculescu-Mizil [2006] gives an extensive empirical comparison of Platt Scaling and Isotonic Regression applied to various types of classifiers in the

[3]Other work has shown that SVM makes accurate estimates of class membership only in the region of the decision boundary [Bartlett and Tewari, 2004].

binary classification setting. In their experiments both calibration methods work well for some classifiers such as SVM and Naive Bayes, but not others such as neural nets and logistic regression.

Methods Using One-versus-one Extension to Multiclass

Some methods [Hastie and Tibshirani, 1998, Refregier and Vallet, 1991, Wu et al., 2004, Price et al., 1995] are concerned only with the one-versus-one multi-class extension of a binary classifier. These methods assume that a probability estimate r_{ij} already exists for each of the binary classifiers, and they do not concern themselves with the initial creation of this probability estimate, only on combining the r_{ij} to estimate p_i. Although we are concerned with the generation of the probabilities, it is worth discussing these algorithms for combining the probabilities; first, they assume an estimate of the probability in question, not the true probability and hence share a common thread with our methods which assume no knowledge of the true probability. Second, the methods they use to combine probabilities introduce ideas and mathematical machinery that can be adapted to our problem.

The probability-combining methods typically use properties of probabilities to derive equations with p_i, and then solve the equations either exactly or approximately. These equations express a relationship between μ_{ij} and p_i, and then using estimates r_{ij} of μ_{ij}, solve for p_i. Wu et al. [2004] compare several existing methods to their original methods. The methods differ significantly in how they estimate p_i; they are all similar in that they require that the r_{ij} be proper probability estimates. This means that, in the case of SVM for example, the raw output of each binary classifier needs to be processed somehow to generate the r_{ij}, such as by using Platt Scaling or isotonic regression, prior to running any methods to solve for p_i. This processing is classifier-dependent and can have a significant effect on the output.

The simplest method is to normalize (2.9):

$$p_i = \frac{2}{k(k-1)} \sum_{j \neq i} I_{r_{ij} > r_{ji}}. \tag{6.3}$$

Note that in this case, the r_{ij} could be raw output from SVM and need not be normalized, as we only want to know if $r_{ij} > r_{ji}$. In Hastie and Tibshirani [1998], the authors wish to minimize the *Kullback-Liebler* (KL) distance between μ_{ij} and r_{ij}, the estimate of μ_{ij}:

$$\sum_{i \neq j} n_{ij} r_{ij} \log \frac{r_{ij}}{\mu_{ij}}. \tag{6.4}$$

Price et al. [1995] use the identity

$$\left(\sum_{i \neq j} P(y = i \vee y = j | x) \right) - (k-2) P(y = j | x) = \sum_{i=1}^{k} P(y = i | x) = 1 \tag{6.5}$$

and then substituting r_{ij} for μ_{ij} and noting that

$$\mu_{ij} = \frac{P(y = i | x)}{P(y = i \vee y = j | x)},$$

solve for p_i directly:

$$p_i = \frac{1}{\sum_{i \neq j} \frac{1}{r_{ji}} - (k-2)}.$$

Refregier and Vallet [1991] use the following identity to create a system of equations:

$$\frac{\mu_{ij}}{\mu_{ji}} = \frac{p_i}{p_j}. \tag{6.6}$$

Then substituting the known value r_{ij} for μ_{ij}, there are $k(k-1)/2$ equations with k unknowns (the p_i). Because the system is overly constrained, it cannot be solved

in general. Wu et al. [2004] suggest the following two minimization problems:

$$\min_{\mathbf{p}} \sum_{i=1}^{k} \left(\sum_{j \neq i} r_{ji} p_i - \sum_{j \neq i} r_{ij} p_j \right)^2 \tag{6.7}$$

and

$$\min_{\mathbf{p}} \sum_{i=1}^{k} \sum_{j \neq i} (r_{ji} p_i - r_{ij} p_j)^2. \tag{6.8}$$

Wu et al. [2004] point out that (6.8) is similar in spirit to (6.6) except that it is solvable. Both (6.7) and (6.8) are solvable using a linear system.

In their analysis of (6.7), (6.8), (6.5), and (6.3) using synthetic data (i.e., when the true p_i are known), Wu et al. [2004] show that in their specific artificial settings all methods perform comparably, both in terms of MSE and accuracy, except (6.3) which performs worse. In these settings they do not use an underlying classifier, but generate the r_{ij} by perturbing the actual μ_{ij} with Gaussian Noise. Wu et al. [2004] give an additional analysis on real-world data, using SVM as the underlying classifier. To generate the r_{ij}, however, they use Platt's method [Platt, 1999]. The results seem to show that with the exception of (6.3) which performs worse, the methods perform comparably.

Other Methods

Huang et al. [2006] extend the method in Hastie and Tibshirani [1998] (6.4) to the general binary-to-multiclass setting. In other words, this new method minimizes an analog to (6.4) in the general setting of training binary classifiers to distinguish any two subsets of classes (see Chapter 1) and combining the output to extract class membership probabilities. This general setting includes both one-versus-one and one-versus-rest. In their experimental analysis, Huang et al. [2006] show that one-versus-one and one-versus-rest perform similarly in terms of MSE and accuracy on real-world data. Their artificial data is generated according to a function that is parameterized by the particular binary-to-multiclass setting, and the authors

acknowledge that this may give an unfair advantage to one method or another. The results on the artificial data are not conclusive as to whether performance differences are due to the binary-to-multiclass setting or to the generation of the artificial data.

A study in Duan and Keerthi [2005] compares the classification accuracy of one-versus-rest SVM using Platt Scaling and the method from Hastie and Tibshirani [1998] (6.4), logistic-regression using the method from Hastie and Tibshirani [1998],[4] one-versus-all SVM, and one-versus-rest SVM. They claim that one-versus-rest SVM with Platt Scaling and the method from Hastie and Tibshirani [1998] are much better than the rest, but the experimental results show that in general the accuracies of all four methods are within standard deviations of one another and it is hard to draw a firm conclusion.

6.2 New Methods

The methods we presented in Section 6.1.2 solve equations for each test example separately, and do not directly preserve the structure inherent in the raw output of the classifier. For instance, the ordering that a classifier gives to the examples may be worth using as a constraint on the p_{ij}, or the fact that one classifier ranks an example higher than another classifier may be worth preserving (this second relationship is indeed preserved indirectly by several methods). In this section, we develop several methods that are motivated by the desire to preserve the structure given by the output of the classifier.

We present several new methods that attempt to estimate the p_i from the raw SVM output. In these methods, no assumption is made that the r_{ij} are known. No algorithm to combine the r_{ij} is needed as the p_i are estimated directly. The methods are given by the following optimization problems which are explained next.

[4]Logistic regression can be used in the multi-class case and generates probabilities, hence it does not require a probability-generating function or a binary-to-multiclass method.

$$\begin{aligned} \texttt{Minimize} \quad & \textstyle\sum_{i=1}^{n}\sum_{j=1}^{k}(\hat{p}_{ij}-p_{ij})^2 \\ \texttt{subject to} \quad & \textstyle\sum_{j=1}^{k}\hat{p}_{ij}=1, i=1\ldots n \\ & c_{ij}\le c_{i'j}\rightarrow \hat{p}_{ij}\le \hat{p}_{i'j}, i,i'=1\ldots n, j=1\ldots k \end{aligned} \tag{6.9}$$

$$\begin{aligned} \texttt{Minimize} \quad & \textstyle\sum_{i=1}^{n}\sum_{j=1}^{k}(\hat{p}_{ij}-p_{ij})^2+\|\xi\|^2 \\ \texttt{subject to} \quad & \textstyle\sum_{j=1}^{k}\hat{p}_{ij}=1, i=1\ldots n \\ & c_{ij}\le c_{i'j}\rightarrow \hat{p}_{ij}\le \hat{p}_{i'j}-\xi_{ij}, i,i'=1\ldots n, j=1\ldots k \end{aligned} \tag{6.10}$$

$$\begin{aligned} \texttt{Minimize} \quad & \textstyle\sum_{i=1}^{n}\sum_{j=1}^{k}(\hat{p}_{ij}-p_{ij})^2 \\ \texttt{subject to} \quad & \textstyle\sum_{j=1}^{k}\hat{p}_{ij}=1, i=1\ldots n \\ & c_{ij}\le c_{ij'}\rightarrow \hat{p}_{ij}\le \hat{p}_{ij'}, i,i'=1\ldots n, j=1\ldots k \end{aligned} \tag{6.11}$$

$$\begin{aligned} \texttt{Minimize} \quad & \textstyle\sum_{i=1}^{n}\sum_{j=1}^{k}(\hat{p}_{ij}-p_{ij})^2+\|\xi\|^2 \\ \texttt{subject to} \quad & \textstyle\sum_{j=1}^{k}\hat{p}_{ij}=1, i=1\ldots n \\ & c_{ij}\le c_{ij'}\rightarrow \hat{p}_{ij}\le \hat{p}_{ij'}-\xi_{ij}, i,i'=1\ldots n, j=1\ldots k \end{aligned} \tag{6.12}$$

where $p_{ij}=P(y_i=j|x_i)$ (as opposed to an analog of r_{ij} used above), $\hat{p}_{ij}$ are the (unknown) estimates of p_{ij} that we are solving for, and ξ is a vector of slack variables. The value of p_{ij} is of course not known in general, hence for the training set we assign $p_{ij}=1$ if $y_i=j$ and 0 otherwise. The c_{ij} are generated from the raw output of the SVM; in evaluating the methods we will use the one-versus-rest setting, and thus the c_{ij} will be the output on example x_i of the classifier distinguishing class j from the rest. We defer the extension of these methods to the general multi-class setting for future work.

These methods use the output of SVM to generate constraints on the $\hat{p}_{ij}$, and then try to get the $\hat{p}_{ij}$ as close to the idealized p_{ij} as possible under these constraints.

In (6.9) we preserve *column order*. Imagine the matrices $C, P \in \mathbf{R}^{n \times k}$ such that $C_{ij} = c_{ij}, P_{ij} = \hat{p}_{ij}$. Then by preserving column order, we mean that if the classifier distinguishing class 1 from the rest "ranks" example m higher than example m' (that is the output of the classifier is higher on m than on m'), i.e., $c_{m1} > c_{m'1}$, then $p_{m1} > p_{m'1}$. The intuition is that the ordering of the examples by each classifier is worth preserving, and our estimates of class membership probabilities should be constrained appropriately. In (6.10), we allow this constraint to be violated, with an associated penalty term ξ. SVM is not a ranking algorithm, and in fact its only goal is to estimate the decision boundary, hence it is not clear whether the constraint on column order is a good one. To throw away column order however, is almost akin to re-learning the classifier, as the c_{ij} represent the distance to the decision boundary and if the p_{ij} do not respect column order, we are effectively ignoring the decision boundary.

In (6.11) we use a different constraint imposed by the decision boundaries: *row order*. By this we mean that if the classifier distinguishing class 1 from the rest gives a higher score to example m than the classifier distinguishing class 2 from the rest, i.e., $c_{m1} > c_{m2}$, then $\hat{p}_{m1} > \hat{p}_{m2}$. Like above, this is akin to preserving the row order in the matrix C in P. Instead of trying to preserve the ordering of distances to the decision boundary, we just want to make sure that the if an example is closer to one decision boundary than another, the $\hat{p}_{ij}$ are ordered accordingly. This means that (6.11) will not change the labels of the SVM. Again we introduce a version with slack variables in (6.12) that allows the constraint to be violated.

Notice that the process just described uses a training or validation set to provide probability estimates for the same set. Therefore we still need to address how to use the probabilities given by Methods 1-4 to assign probabilities to new examples. We propose the following simple algorithm to address this:

- Split the training set into t training examples and c calibration examples.
- Train the binary classifiers on the t training examples.

- Compute the raw output of the trained classifiers on the c calibration examples and use the output as input to the quadratic program.
- Embed each of the c calibration examples in a k-dimensional feature space, where feature j is the raw output of the binary classifier distinguishing class j from all others.
- Embed any future test example in the same feature space and find its nearest neighbor from the calibration examples in this space.
- Assign the test example the probability vector that was calculated for its nearest neighbor.

6.3 Experiments and Results

In the following experiments we use Methods 1-4 with input from both SVM and kNN. We compare these methods to normalization of the one-versus-rest SVM (NORM-SVM), Normalization of k-NN output (NORM-kNN) and logistic regression (LR).

The methods summarized in Wu et al. [2004] and in Section 6.1.2 are for combining existing probability estimates from a one-versus-one multi-class classifier. The exception is Huang et al. [2006] who generalize the method from Hastie and Tibshirani [1998] (6.7) to the general binary-to-multiclass extension [Allwein et al., 2000]. For reference we do compare the above methods to the generalized method of Huang et al. [2006] (6.7), which uses Platt Scaling prior to applying the method. We refer to this method as 1v1 when a one-versus-one SVM is used and 1vR when a one-versus-rest SVM is used. This allows us to examine what, if any, performance difference may be attributed to the particular binary-to-multiclass extension of SVM.

For every method that takes input from SVM, we run the SVM using a linear kernel and a RBF kernel:

$$k(x, y) = e^{-C\|x-y\|^2}$$

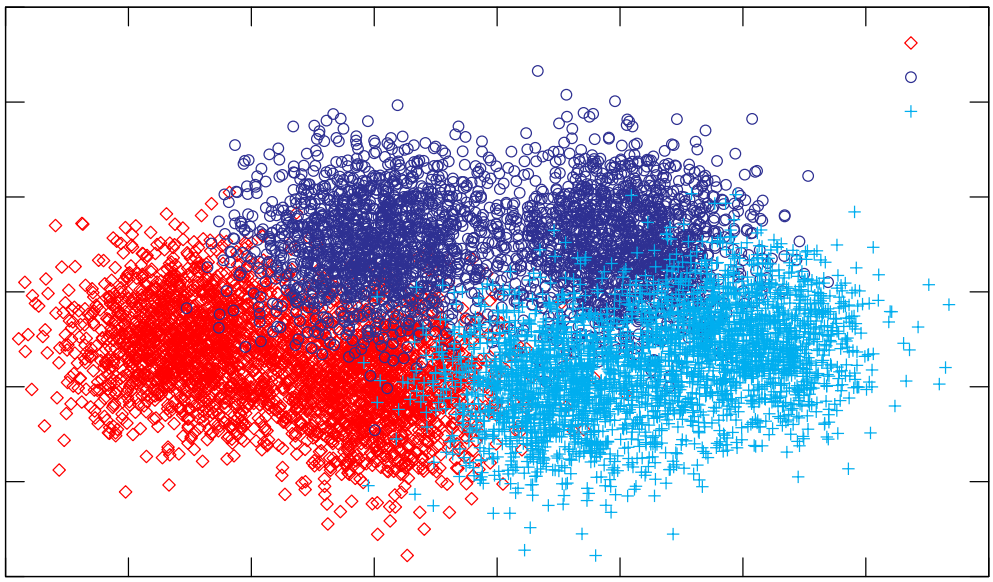

Figure 6.1: Training dataset of 3-classes each generated from a mixture of 2 Gaussians

where $C = 1/2$. This gives us an additional point of comparison and allows us to examine what performance differences may be due to the feature space.

To evaluate the quality of the probability estimates, we use the quantity

$$\frac{1}{n}\sum_{i=1}^{n}\frac{1}{k}\sum_{j=i}^{k}(p_{ij} - \hat{p}_{ij})^2$$

which we refer to as *MSE*. Although we are concerned primarily with the ability of the methods to estimate probabilities, we also examine how each method affects classification accuracy. As a baseline we will use a constant probability estimate of $1/k$ when computing MSE, and the mode of the labels in the training set when computing accuracy.

To evaluate the methods we run on both real and artificial data. For real data, we use the datasets *dna* and *segment* from the Statlog Collection [Michie et al., 1994], *waveform* [Asuncion and Newman, 2007], *USPS* [Hull, 1994], and *MNIST* [LeCun

Table 6.2: Description of Datasets

	Number of Classes	Number of Dimensions	Number of Examples
dna	3	180	3186
waveform	3	21	5000
satimage	6	36	6435
segment	7	19	2310
MNIST	10	784	70000

et al., 1998]. We chose these datasets as they are used in other work on generating probabilities from classifier output [Wu et al., 2004, Duan and Keerthi, 2005, Huang et al., 2006]. We use the first 10 pre-determined cross-validation splits from Wu et al. [2004].[5] Table 6.2 shows information on the real-world data. For artificial data, we create two 3-class datasets, each containing a fixed training and test set with 3000 examples. First a 2-dimensional dataset in which each class is generated from a mixture of two Gaussians, class 1 with means $(1,1),(4,0)$, class 2 with $(4,3),(8,3)$ and class 3 with $(7,0),(10,0)$. The means form a triangle-like pattern with each class having two means on a side, and each class having one mean close to a mean from each of the other two classes (see Figure 6.1). Second, we generate another 2-dimensional dataset, again generating each class according to two means, and again in a triangle-like pattern, except that this time we separate the means even further by scaling each mean by a factor of 2.

For Methods 1-4 the training process is as outlined above with 300 calibration examples held out from the training set prior to training the classifiers. For example, for a training set of size 3000 as for the artificial data, we train the classifier on 2700 examples, withholding 300 as input to the quadratic programs for Methods 1-4. We use the same training set on all examples, so methods that do not require the calibration set will train on the 2700 examples and ignore the 300 calibration examples.

We first run all methods on a subset of the real-world data: dna and segment.

[5]These were formed by randomly selecting 1000 testing and 800 training examples from the entire dataset.

Table 6.3: Accuracy for datasets dna and segment. Numbers are averages over 10-fold cross-validation with standard deviations. No results were obtained for LR on segment due to numerical precision issues with the software implementation.

	dna	segment
baseline	0.509100 ± 0.006437	0.151500 ± 0.003440
Method 1	0.509100 ± 0.006437	0.148100 ± 0.003604
Method 3	0.889600 ± 0.014841	0.920300 ± 0.008056
Method 4	0.888800 ± 0.015469	0.934700 ± 0.007243
Method 1 (RBF)	0.509100 ± 0.006437	0.148100 ± 0.003604
Method 3 (RBF)	0.897800 ± 0.010993	0.905000 ± 0.006912
Method 4 (RBF)	0.901000 ± 0.009718	0.924400 ± 0.007947
Method 1 (kNN)	0.745100 ± 0.023634	0.641900 ± 0.117678
Method 3 (kNN)	0.8955 ± 0.011919	0.6988 ± 0.063261
Method 4 (kNN)	0.755100 ± 0.017842	0.751500 ± 0.043887
1v1	0.897700 ± 0.008125	0.922900 ± 0.006064
1vR	0.890600 ± 0.012322	0.915700 ± 0.009661
1v1 (RBF)	0.908500 ± 0.007044	0.881500 ± 0.007261
1vR (RBF)	0.906200 ± 0.007465	0.898700 ± 0.010678
NORM	0.888700 ± 0.014103	0.894600 ± 0.010895
NORM (RBF)	0.893900 ± 0.011455	0.860100 ± 0.011259
NORM (kNN)	0.728900 ± 0.024164	0.678400 ± 0.063755
LR	0.895500 ± 0.011919	N/A

Table 6.4: MSE for datasets dna and segment. Numbers are averages over 10-fold cross-validation with standard deviations. If no kernel/algorithm is shown in parenthesis, the classification method is SVM with a linear kernel. No results were obtained for LR on segment due to numerical precision issues with the software implementation.

	dna	segment
baseline	0.222222 ± 0.000000	0.122449 ± 0.000000
Method 1	0.228730 ± 0.007894	0.122400 ± 0.000000
Method 3	0.056320 ± 0.006139	0.015300 ± 0.001055
Method 4	0.056750 ± 0.006189	0.013200 ± 0.000799
Method 1 (RBF)	0.228770 ± 0.007910	0.122400 ± 0.000000
Method 3 (RBF)	0.050320 ± 0.003108	0.018530 ± 0.001013
Method 4 (RBF)	0.049330 ± 0.002591	0.015570 ± 0.000987
Method 1 (kNN)	0.123270 ± 0.011234	0.046440 ± 0.009484
Method 3 (kNN)	0.05672 ± 0.006228	0.053420 ± 0.008930
Method 4 (kNN)	0.117980 ± 0.008177	0.046190 ± 0.006578
1v1	0.052470 ± 0.003651	0.018385 ± 0.000360
1vR	0.055827 ± 0.004296	0.022013 ± 0.000900
1v1 (RBF)	0.044986 ± 0.002991	0.025856 ± 0.000952
1vR (RBF)	0.048177 ± 0.002134	0.021521 ± 0.001004
NORM	0.054601 ± 0.005239	0.030808 ± 0.001462
NORM (RBF)	0.081478 ± 0.003051	0.050608 ± 0.001102
NORM (kNN)	0.168180 ± 0.015340	0.089530 ± 0.018423
LR	0.056720 ± 0.006228	N/A

Table 6.5: Accuracies on remaining datasets.

Dataset	1v1	Method 3 (RBF)	Method 4 (RBF)
satimage	0.835200 ± 0.006356	0.800400 ± 0.009709	0.822800 ± 0.008108
waveform	0.849300 ± 0.009878	0.850800 ± 0.008791	0.852100 ± 0.010214
MNIST	0.833900 ± 0.007651	0.888900 ± 0.007937	0.824500 ± 0.016257

Table 6.6: MSE on remaining datasets.

Dataset	1v1	Method 3 (RBF)	Method 4 (RBF)
satimage	0.037821 ± 0.001123	0.043820 ± 0.002287	0.039250 ± 0.002076
waveform	0.072216 ± 0.002977	0.071710 ± 0.003142	0.069700 ± 0.003106
MNIST	0.023295 ± 0.001542	0.016490 ± 0.000843	0.026750 ± 0.001921

This first round of experiments is meant to give a general idea of the performance of the algorithms. In Tables 6.3 and 6.4 we notice that Method 1 performs poorly on the data set. In fact, it appears Method 1 may be an ill-formed quadratic program as it seems to be converging to the same solution regardless of input; it is easy to see instances in which the constraints will be forced to equality. For example, consider a two example dataset such that $c_{11} > c_{21}$ and $c_{12} > c_{22}$. The only way to satisfy all constraints is to set $c_{11} = c_{21}$ and $c_{12} = c_{22}$. With a larger dataset and hence more constraints the picture is not as simple, however the 2 example case gives some insight as to why preserving column-order is not on its own sufficient. We do not report results for Method 2 as they are equally bad. Methods 3 and 4, however, perform quite well, in most cases comparably than 1v1 and 1vR, which seem to be the overall best performing methods. On the segment data using the RBF kernel Methods 3 and 4 are the best both in terms of accuracy and MSE. Interestingly, Method 3 (no slack variable) has a slight drop in performance on the dna dataset when using the RBF kernel instead of linear, but the RBF kernel gives an increase in performance for Method 4 (with slack variables). In general, the RBF kernel helps all methods.

Next we run on the remaining real-world datasets with the methods that did

Table 6.7: Accuracy on artificial datasets. Art 2 is generated from the means of Art 1 scaled by a factor of 2.

Dataset	1v1	Method 3 (RBF)	Method 4 (RBF)
Art 1	0.907	0.907	0.9063
Art 2	0.997	0.997	0.997

Table 6.8: MSE on artificial datasets. Art 2 is generated from the means of Art 1 scaled by a factor of 2.

Dataset	1v1	Method 3 (RBF)	Method 4 (RBF)
Art 1	0.00023	0.00028	0.00028
Art 2	0.0045	0.007	0.0065

well in the experiments on dna and segment: we concentrate on Methods 3 and 4, and compare them to 1v1 (the overall best performer so far). The results for accuracy are in Table 6.5 and for MSE in Table 6.6. We show clear wins on MNIST for Methods 3 and 4, but the results in general are similar to the first round of experiments: 1v1 and Method 3 and 4 are competitive, each one sometimes beating the other, but there is no clear overall winner.

We show the results for the artificial data in Table 6.7 and Table 6.8. We see that as in the experiments with the real-world data, Methods 3 and 4 perform comparably to 1v1, with a slight edge going to 1v1. This setting gives us a better idea of how good the probability estimates are, because we have actual values for $p(y = i|x)$ with which to evaluate the estimates. The MSE values are extremely low for all algorithms, indicating that this setting may not be challenging enough. This is a worthwhile issue to address in future experiments.

6.4 Conclusions and Future Work

We conclude the chapter with the following observations and ideas for future work.

Observe that Methods 1-4 all have the following term in their objective functions:

$$\sum_{i=1}^{n} \|\hat{\mathbf{p}}_\mathbf{i} - \mathbf{p}_\mathbf{i}\|^2$$

where $\hat{\mathbf{p}}_\mathbf{i}$ is the norm of the probability vector for example i. Expanding we have

$$\sum_{i=1}^{n} \left(\|\hat{\mathbf{p}}_\mathbf{i}\|^2 + \|\mathbf{p}_\mathbf{i}\|^2 - 2\langle \hat{p}_i, p_i \rangle \right)$$

and because $\mathbf{p}_\mathbf{i}$ is a constant, it can be eliminated from the objective function

$$\sum_{i=1}^{n} \left(\|\hat{\mathbf{p}}_\mathbf{i}\|^2 - 2\langle \hat{p}_i, p_i \rangle \right)$$

Setting all $p_{ij} = 1/k$ minimizes the term $\|\hat{\mathbf{p}}_\mathbf{i}\|^2$. It is possible that this term dominates and we assign $p_{ij} = 1/k$ in some cases leading to poor probability estimates. The inner product term, $-2\langle \hat{p}_i, p_i \rangle$ pushes the $\hat{p}_{ij}$ to be similar to the p_{ij}, which are $0-1$ valued on the training set. This means that any value in $\hat{\mathbf{p}}_\mathbf{i}$ in a dimension other than the class label does not affect this term, as $p_{ij} = 0$ for all j other than the class label. The inner product term and the norm term hence drive the $\mathbf{p}_\mathbf{i}$ to 1 in dimension j and to $1/k$ in all other dimensions. If we remove the norm term, we end up with a linear program with an objective function similar to that in the primal of SVM:

$$\begin{array}{rl} \texttt{Maximize} & \sum_{i=1}^{n} \langle \hat{p}_i, p_i \rangle \\ \texttt{subject to} & \sum_{j=1}^{k} \hat{p}_{ij} = 1, i = 1 \ldots n \\ & \text{and constraints on row/column ordering} \end{array} \tag{6.13}$$

Here we replace the minimization of the negative inner product with the maximization of the inner product. This is similar to the SVM primal objective function in a nm dimensional feature space with $\mathbf{w}$ set to the vector of all the $\hat{p}_i$, and each training example p_i has all 0-valued features except at indices $(i-1)k+1 \ldots ik$. The

constraints are similar to constraining the L1-norm of $\mathbf{w}$, except it is a piecewise constraint. This may present another way of looking at this problem and deserves further investigation.

The inequality constraints in the objective functions are not strict, and hence it is possible to set all p_{ij} (along a row or column) equal to each other and satisfy the constraints. Future work should explore how to change these constraints or add additional constraints so that such a trivial solution is not available.

Row and column ordering could be replaced by some other measure of how SVM ranks examples. Perhaps a more complex relationship is useful, such as ensuring that if $c_i > c_j$, then $p_i > p_j$ where c_i is the difference between the output of the classifier giving the highest output on example i and the classifier giving the second highest output on example i (and likewise for the p_i).

The computational resources required by the quadratic program are substantial and restrict the number of examples and classes in the calibration set. It may be possible to formulate linear programs that accomplish the same or similar goals as Methods 1-4.

Using the nearest-neighbor approach to map new examples to probability vectors is a good start, however in future work we will want to explore other mapping techniques.

Data generated by different distributions is a priority for future experiments, as we failed to truly differentiate 1v1 and Methods 3 and 4 with our artificial data. A good artificial setting will highlight strengths and weaknesses which we have not done here.

Finally, it may be the case that SVM simply ranks examples poorly and the methods presented here do as well as can be expected. It is not the objective of SVM to rank examples, and theoretical results have shown that SVM (and other classifiers) only estimates $P(y|x)$ properly in the region immediately around the decision boundary, i.e., when $P(y|x) = 0.5$ [Bartlett and Tewari, 2004]. A better

source of input to the quadratic programs (and to other methods) might be the the output of a ranking SVM or other ranking algorithm. These algorithms attempt only to rank the examples according to how likely they are to be in a given class, but they do not estimate probabilities. In future experiments, it would be interesting to see if Methods 1-4 can perform well when combined with a ranking versus classification algorithm.

Chapter 7

Conclusion

We have presented several interesting results in both analyzing and developing machine learning classification algorithms. We gave a thorough empirical study of Perceptron Algorithm variants in the noisy data setting, showing the surprising result that Perceptron with Margins outperforms variants specifically designed for the noisy setting, and even performs comparably with SVM. We studied problems where the data are graphs or hypergraphs, and designed a kernel that works directly on hypergraphs; we showed that it performs as well as, and sometimes better than, ILP systems on hypergraph data, and that it uses a different feature space than existing graph kernels. We illustrated several interesting properties of the cross-correlation function, showing that it is very useful for classifying periodic time series, but that it is not positive semidefinite. As a solution, we developed a kernel that is an intuitive analog to the cross-correlation and demonstrated its good performance. We then built a completely automatic system for classifying periodic variable stars and proved its performance by classifying the entire MACHO survey. Finally we studied existing methods for generating classifier confidence estimates, and proposed several new methods, comparing them in an experimental setting.

There are several avenues for future work that follow from the results of this thesis. The automatic classification system for periodic variable stars has several

steps in its pipeline that can be improved. Specifically period finding has shown to be a very interesting problem. The hypergraph kernel we developed does not account for certain types of conjunctions and it is not clear whether it is possible to compute them with a similar kernel. The methods we developed for estimating confidence are competitive, but leave room for improvement. The question of whether SVM can generate better confidences remains a key component of that work.

Bibliography

T. Adamek and N. O'Connor. A multiscale representation method for nonrigid shapes with a single closed contour. *IEEE Trans. Circuits Syst. Video Techn.*, 14 (5):742–753, 2004.

C. Alcock et al. The MACHO Project - a Search for the Dark Matter in the Milky-Way. In B. T. Soifer, editor, *Sky Surveys. Protostars to Protogalaxies*, volume 43 of *Astronomical Society of the Pacific Conference Series*, pages 291–+, January 1993.

C. Alcock et al. The MACHO project LMC variable star inventory. 1: Beat Cepheids-conclusive evidence for the excitation of the second overtone in classical Cepheids. *Astronomy Journal*, 109:1653–+, April 1995. doi: 10.1086/117392.

E.L. Allwein, R. Schapire, and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–141, 2000.

M. Arias, R. Khardon, and J. Maloberti. Learning Horn expressions with logan-h. *Journal of Machine Learning Research*, 8:549–587, 2007.

A. Asuncion and D.J. Newman. UCI machine learning repository, 2007. URL `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

M-F. Balcan, A. Blum, and N. Srebro. A theory of learning with similarity functions. *Machine Learning*, 72(1-2):89–112, 2008.

P. Bartlett and J. Shawe-Taylor. Generalization performance of support vector machines and other pattern classifiers. In B. Schölkopf, C.J.C. Burges, and A.J. Smola, editors, *Advances in Kernel Methods – Support Vector Learning*, pages 43–54. MIT Press, Cambridge, MA, 1999.

P. Bartlett and A. Tewari. Sparseness versus estimating conditional probabilities: Some asymptotic results. In J. Shawe-Taylor and Y. Singer, editors, *COLT*, volume 3120 of *Lecture Notes in Computer Science*, pages 564–578. Springer, 2004. ISBN 3-540-22282-0.

Peter L. Bartlett. The sample complexity of pattern classification with neural networks: The size of the weights is more important than the size of the network. *IEEE Transactions on Information Theory*, 44(2):525–536, 1998.

D. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *Knowledge Discovery and Data Mining*, pages 359–370, 1994.

C.M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, NY, 2006.

R.M. Blair, H. Fang, W.S. Branham, B.S. Hass, S.L. Dial, C.L. Moland, W. Tong, L. Shi, R. Perkins, and D.M. Sheehan. The estrogen receptor relative binding affinities of 188 natural and xenochemicals: Structural diversity of ligands. *Toxicol. Sci.*, 54:138–153, 2000.

B. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 144–152, 1992.

W.S. Branham, S.L. Dial, C.L. Moland, B.S. Hass, R.M. Blair, H. Fang, L. Shi, W. Tong, R.G. Perkins, and D.M. Sheehan. Binding of phytoestrogens and mycoestrogens to the rat uterine estrogen receptor. *J. Nutr.*, 132:658–664, 2002.

L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In Cohen and Moore [2006], pages 161–168. ISBN 1-59593-383-2.

N. Cesa-Bianchi, A. Conconi, and C. Gentile. On the Generalization Ability of On-line Learning Algorithms. *Information Theory, IEEE Transactions on*, 50(9): 2050–2057, 2004.

N. Cesa-Bianchi, A. Conconi, and C. Gentile. A Second-order Perceptron Algorithm. *SIAM Journal on Computing*, 34(3):640–668, 2005.

C-C. Chang and C-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

P. M. Cincotta, M. Mendez, and J. A. Nunez. Astronomical Time Series Analysis. I. A Search for Periodicity Using Information Entropy. *Astrophysical Journal*, 449: 231–+, August 1995. doi: 10.1086/176050.

W. Cohen. Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123, 1995.

W. Cohen and A. Moore, editors. *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, 2006. ACM. ISBN 1-59593-383-2.

M. Collins. Ranking algorithms for named entity extraction: Boosting and the voted perceptron. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 489–496, 2002.

M. Collins and N. Duffy. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *ACL*, pages 263–270, 2002.

K. Crammer, O. Dekel, S. Shalev-Shwartz, and Y. Singer. Online passive aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585, 2005.

N. Cristianini and J. Shawe-Taylor. *An introduction to support vector machines*. Cambridge University Press, 2000.

C. Cumby and D. Roth. On kernel methods for relational learning. In Fawcett and Mishra [2003], pages 107–114. ISBN 1-57735-189-4.

M. Cuturi. Permanents, transport polytopes and positive definite kernels on histograms. In M. Veloso, editor, *IJCAI*, pages 732–737, 2007.

M. Cuturi, J.-P. Vert, O. Birkenes, and T. Matsui. A kernel for time series based on global alignments. *IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP) 2007.*, 2:413–416, April 2007. ISSN 1520-6149. doi: 10.1109/ICASSP.2007.366260.

I. Dagan, Y. Karov, and D. Roth. Mistake-driven learning in text categorization. *CoRR*, cmp-lg/9706006, 1997.

L. De Raedt. Logical settings for concept learning. *Artificial Intelligence*, 95(1): 187–201, 1997. See also relevant Errata (forthcoming).

L. De Raedt and S. Dzeroski. First order jk-clausal theories are PAC-learnable. *Artificial Intelligence*, 70:375–392, 1994.

J. Debosscher, L. M. Sarro, C. Aerts, J. Cuypers, B. Vandenbussche, R. Garrido, and E. Solano. Automated supervised classification of variable stars. i. methodology. *Astronomy and Astrophysics*, 475:1159–1183, December 2007.

O. Dekel and Y. Singer. Data driven online to batch conversions. In *Advances in Neural Information Processing Systems 18 (Proceedings of NIPS 2005)*, 2005.

M. Deshpande, M. Kuramochi, and G. Karypis. Frequent sub-structure-based approaches for classifying chemical compounds. In *ICDM*, pages 35–42. IEEE Computer Society, 2003. ISBN 0-7695-1978-4.

T. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40:139–158, 2000.

T. Dietterich, M. Kearns, and Y. Mansour. Applying the weak learning framework to understand and improve C4.5. In *Proceedings of the International Conference on Machine Learning*, pages 96–104, 1996.

K. Duan and S. Keerthi. Which is the best multiclass SVM method? An empirical study. In N. Oza, R. Polikar, J. Kittler, and F. Roli, editors, *Multiple Classifier Systems*, volume 3541 of *Lecture Notes in Computer Science*, pages 278–285. Springer, 2005. ISBN 3-540-26306-3.

C. Elkan. Using the triangle inequality to accelerate k-means. In Fawcett and Mishra [2003], pages 147–153. ISBN 1-57735-189-4.

L. Faccioli, C. Alcock, K. Cook, G. E. Prochter, P. Protopapas, and D. Syphers. Eclipsing Binary Stars in the Large and Small Magellanic Clouds from the MACHO Project: The Sample. *Astronomy Journal*, 134:1963–1993, November 2007. doi: 10.1086/521579.

H. Fang, W. Tong, L.M. Shi, R. Blair, R. Perkins, W. Branham, B.S. Hass, Q. Xie, S.L. Dial, C.L. Moland, and D.M. Sheehan. Structure-activity relationships for a large diverse set of natural, synthetic, and environmental estrogens. *Chem. Res. Tox.*, 14(3):280–294, 2001.

T. Fawcett and N. Mishra, editors. *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*, 2003. ISBN 1-57735-189-4.

Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *ICML*, pages 148–156, 1996.

Y. Freund and R. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37:277–296, 1999.

J.H. Friedman. A variable span smoother. Technical report, Stanford University CA Lab for Computational Statistics, 1984.

T. Friess, N. Cristianini, and C. Campbell. The kernel adatron algorithm: A fast and simple learning procedure for support vector machines. In *Proceedings of the International Conference on Machine Learning*, pages 188–196, 1998.

H. Fröhlich, J. Wegner, F. Sieker, and A. Zell. Optimal assignment kernels for attributed molecular graphs. In *ICML*, pages 225–232, 2005.

S. Gallant. Perceptron-based learning algorithms. *IEEE Transactions on Neural Networks*, 1(2):179–191, 1990.

A. Garg and D. Roth. Margin distribution and learning algorithms. In *Proc. of the International Conference on Machine Learning (ICML)*, pages 210–217, 2003.

T. Gärtner. Predictive graph mining with kernel methods. In *Advanced Methods for Knowledge Discovery from Complex Data*. Springer, 2005.

T. Gärtner, P. Flach, A. Kowalczyk, and A.J. Smola. Multi-instance kernels. In *International Conference on Machine Learning*, pages 179–186, 2002.

T. Gärtner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In B. Schölkopf and M. Warmuth, editors, *COLT*, volume 2777 of *Lecture Notes in Computer Science*, pages 129–143. Springer, 2003. ISBN 3-540-40720-0.

X. Ge and P. Smyth. Deformable markov model templates for time-series pattern matching. In *Knowledge Discovery and Data Mining*, pages 81–90, 2000.

M. Geha et al. Variability-selected Quasars in MACHO Project Magellanic Cloud Fields. *Astronomy Journal*, 125:1–12, January 2003. doi: 10.1086/344947.

C. Gentile. A new approximate maximal margin classification algorithm. *Journal of Machine Learning Research*, 2:213–242, 2001.

Z. Ghahramani, editor. *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvalis, Oregon, USA, June 20-24, 2007*, volume 227 of *ACM International Conference Proceeding Series*, 2007. ACM. ISBN 978-1-59593-793-3.

P. Gorry. General least-squares smoothing and differentiation by the convolution (savitzky-golay) method. *Analytical Chemistry*, 62(6):570–573, 1990. doi: 10.1021/ac00205a007. URL \url{http://pubs.acs.org/doi/abs/10.1021/ac00205a007}.

T. Graepel, R. Herbrich, and R. Williamson. From margin to sparsity. In *Advances in Neural Information Processing Systems 13 (Proceedings of NIPS 2000)*, pages 210–216, 2000.

A. Grove and D. Roth. Linear concepts and hidden variables: An empirical study. In M. Jordan, M. Kearns, and S. Solla, editors, *NIPS*. The MIT Press, 1997. ISBN 0-262-10076-2.

S. Har-Peled, D. Roth, and D. Zimak. Constraint classification for multiclass classification and ranking. In S. Becker, S. Thrun, and K. Obermayer, editors, *NIPS*, pages 785–792. MIT Press, 2002. ISBN 0-262-02550-7.

J. D. Hartman, B. S. Gaudi, M. J. Holman, B. A. McLeod, K. Z. Stanek, J. A. Barranco, M. H. Pinsonneault, and J. S. Kalirai. Deep MMT Transit Survey of the Open Cluster M37. II. Variable Stars. *Astrophysical Journal*, 675:1254–1277, March 2008. doi: 10.1086/527460.

T. Hastie and R. Tibshirani. Classification by pairwise coupling. *The Annals of Statistics*, 26(2):451–471, 1998.

D. Haussler. Convolution kernels for discrete structures. Technical Report UCSC-CRL-99-10, Department of Computer Science, University of California at Santa Cruz, July 1999.

C. Helma, R. King, S. Kramer, and A. Srinivasan. The predictive toxicology challenge 2000-2001. *Bioinformatics*, 17(1):107–108, 2001.

R. Hérault and Y. Grandvalet. Sparse probabilistic classifiers. In Ghahramani [2007], pages 337–344. ISBN 978-1-59593-793-3.

K. W. Hodapp et al. Design of the Pan-STARRS telescopes. *Astronomische Nachrichten*, 325:636–642, October 2004. doi: 10.1002/asna.200410300.

T. Horváth, S. Wrobel, and U. Bohnebeck. Relational instance-based learning with lists and terms. *Machine Learning*, 43(1/2):53–80, 2001.

T. Horváth, T. Gärtner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In *KDD*, pages 158–167, 2004.

D. A. Howell et al. Gemini Spectroscopy of Supernovae from the Supernova Legacy Survey: Improving High-Redshift Supernova Selection and Classification. *Astrophysical Journal*, 634:1190–1201, December 2005. doi: 10.1086/497119.

T-K. Huang, R.C. Weng, and C-J. Lin. Generalized Bradley-Terry models and multi-class probability estimates. *Journal of Machine Learning Research*, 7:85–115, 2006.

J. Hull. A database for handwritten text recognition research. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16(5):550–554, 1994.

T. Joachims. Making large-scale SVM learning practical. In B. Scholkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1999.

H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *ICML*, pages 321–328, 2003.

M. Kearns and U. Vazirani. *An Introduction to Computational Learning Theory.* The MIT Press, Cambridge, Massachusetts, 1994.

M. Kearns, M. Li, L. Pitt, and L. G. Valiant. Recent results on boolean concept learning. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 337–352, 1987.

M. Kearns, R. Schapire, and L. Sellie. Toward efficient agnostic learning. *Machine Learning*, 17(2/3):115–141, 1994.

E. Keogh, L. Wei, X. Xi, S-H. Lee, and M. Vlachos. Lb keogh supports exact indexing of shapes under rotation invariance with arbitrary representations and distance measures. In U. Dayal et al., editors, *International Conference on Very Large Databases*, pages 882–893. ACM, 2006. ISBN 1-59593-385-9.

R. Khardon and G. Wachman. Noise tolerant variants of the perceptron algorithm. *Journal of Machine Learning Research*, 8:227–248, 2007.

R. Khardon, D. Roth, and R.A. Servedio. Efficiency versus convergence of boolean kernels for on-line learning algorithms. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *NIPS*, pages 423–430. MIT Press, 2001.

J. Kivinen, A. J. Smola, and R. C. Williamson. Online Learning with Kernels. *IEEE Transactions on Signal Processing*, 52(8):2165–2176, 2004.

A. Kowalczyk, A. Smola, and R. Williamson. Kernel machines and boolean functions. In *Advances in Neural Information Processing Systems 14 (Proceedings of NIPS 2001)*, pages 439–446, 2001.

S. Kramer and L. De Raedt. Feature construction with version spaces for biochemical applications. In *ICML*, pages 258–265, 2001.

W. Krauth and M. Mézard. Learning algorithms with optimal stability in neural networks. *Journal of Physics A*, 20(11):745–752, 1987.

N. Landwehr, A. Passerini, L. De Raedt, and P. Frasconi. kfoil: Learning simple relational kernels. In *AAAI*. AAAI Press, 2006.

N. Landwehr, K. Kersting, and L. De Raedt. Integrating naïve bayes and foil. *Journal of Machine Learning Research*, 8:481–507, 2007.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998.

Y. Li. Selective voting for perception-like online learning. In *ICML*, pages 559–566, 2000.

Y. Li and P. Long. The relaxed online maximum margin algorithm. *Machine Learning*, 46:361–387, 2002.

Y. Li, H. Zaragoza, R. Herbrich, J. Shawe-Taylor, and J. Kandola. The perceptron algorithm with uneven margins. In *International Conference on Machine Learning*, pages 379–386, 2002.

N. Littlestone. From on-line to batch learning. In *Proceedings of the Conference on Computational Learning Theory*, pages 269–284, 1989.

N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, 1987.

N. R. Lomb. Least-squares frequency analysis of unequally spaced data. *Astrophysics and Space Science*, 39:447–462, February 1976. doi: 10.1007/BF00648343.

Z. Lu, T.K. Leen, Y. Huang, and D. Erdogmus. A reproducing kernel hilbert space framework for pairwise time series distances. In William W. Cohen, Andrew

McCallum, and Sam T. Roweis, editors, *International Conference on Machine Learning*, pages 624–631, 2008.

R. Luss and A. d'Aspremont. Support vector machine classification with indefinite kernels. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Neural Information Processing Systems*. MIT Press, 2007.

P. Mahé, N. Ueda, T. Akutsu, J. Perret, and J. Vert. Extensions of marginalized graph kernels. In *ICML*, 2004.

D. Mease, A. Wyner, and A. Buja. Boosted classification trees and class probability/quantile estimation. *JMLR*, 8:409–439, 2007.

D. Michie, D.J. Spiegelhalter, and C.C. Taylor. Machine learning, neural and statistical classification, 1994. URL `http://www.amsta.leeds.ac.uk/~charles/statlog/`.

T.M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

A. Moore. The anchors hierarchy: Using the triangle inequality to survive high dimensional data. In C. Boutilier and M. Goldszmidt, editors, *Uncertainty in Artificial Intelligence*, pages 397–405. Morgan Kaufmann, 2000. ISBN 1-55860-709-9.

S. Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13 (3&4):245–286, 1995.

S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 20:629–679, 1994.

S. Muggleton, H. Lodhi, A. Amini, and M. Sternberg. Support vector inductive logic programming. In A. Hoffmann, H. Motoda, and T. Scheffer, editors, *Discovery Science*, volume 3735 of *Lecture Notes in Computer Science*, pages 163–175. Springer, 2005. ISBN 3-540-29230-6.

D.J. Newman, S. Hettich, C.L. Blake, and C.J. Merz. UCI repository of machine learning databases, 1998. URL \url{http://www.ics.uci.edu/~mlearn/MLRepository.html}.

A. Niculescu-Mizil and R. Caruana. Predicting good probabilities with supervised learning. In L. De Raedt and S. Wrobel, editors, *ICML*, pages 625–632. ACM, 2005. ISBN 1-59593-180-5.

A. B. Novikoff. On convergence proofs on perceptrons. *Symposium on the Mathematical Theory of Automata*, 12:615–622, 1962.

C.S. Ong, X. Mary, S. Canu, and A.J. Smola. Learning with non-positive kernels. In C. Brodley, editor, *International Conference on Machine Learning*. ACM, 2004.

S. Osowski, L.T. Hoai, and T. Markiewicz. Support vector machine-based expert system for reliable heartbeat recognition. *IEEE Transactions on Biomedical Engineering*, 51:582–589, 2004.

C.H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1993.

J.C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In A.J. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 61–74. MIT Press, Cambridge, MA, 1999.

W. H. Press and G. B. Rybicki. Fast algorithm for spectral analysis of unevenly sampled data. *Astrophysical Journal*, 338:277–280, March 1989. doi: 10.1086/167197.

D. Price, S. Knerr, L. Personnaz, and G. Dreyfus. Pairwise neural network classifiers with probabilistic outputs. In G. Tesauro, D. Touretzky, and T. Leen, editors, *NIPS*, pages 1109–1116. MIT Press, 1995.

P. Protopapas, J. M. Giammarco, L. Faccioli, M. F. Struble, R. Dave, and C. Alcock. Finding outlier light curves in catalogues of periodic variable stars. *Monthly Notices of the Royal Astronomical Society*, 369:677–696, June 2006. doi: 10.1111/j.1365-2966.2006.10327.x.

V. Punyakanok, D. Roth, and W-T. Yih. The importance of syntactic parsing and inference in semantic role labeling. *Computational Linguistics*, 34(2):257–287, 2008.

J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5: 239–266, 1990.

J. R. Quinlan. Unknown attribute values in induction. In Alberto Maria Segre, editor, *ML*, pages 164–168. Morgan Kaufmann, 1989. ISBN 1-55860-036-1.

L. Ralaivola, S. J. Swamidass, H. Saigo, and P. Baldi. Graph kernels for chemical informatics. *Neural Networks*, 18(8):1093–1110, 2005.

P. Refregier and F. Vallet. Probabilistic approaches for multiclass classification with neural networks. In *International Conference on Artificial Neural Networks*, pages 1003 – 1006, 1991.

J.D. Reimann. *Frequency Estimation Using Unequally-Spaced Astronomical Data*. PhD thesis, University California at Berkeley, 1994.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–407, 1958.

J. D. Scargle. Studies in astronomical time series analysis. II - Statistical aspects of spectral analysis of unevenly spaced data. *Astrophysical Journal*, 263:835–853, December 1982. doi: 10.1086/160554.

B. Schölkopf and A.J. Smola. *Learning with Kernels*. The MIT Press, 2002.

A. Schwarzenberg-Czerny. On the advantage of using analysis of variance for period search. *MNRAS*, 241:153–165, November 1989.

R.A. Servedio. On PAC Learning Using Winnow, Perceptron, and a Perceptron-like Algorithm. In *Proceedings of the twelfth annual conference on Computational learning theory*, pages 296–307, 1999.

S. Shalev-Shwartz and Y. Singer. A new perspective on an old perceptron algorithm. In *Proceedings of the Sixteenth Annual Conference on Computational Learning Theory*, pages 264–278, 2005.

J. Shawe-Taylor, P. Bartlett, R. Williamson, and M. Anthony. Structural risk minimization over data-dependent hierarchies. *IEEE Transactions on Information Theory*, 44(5):1926–1940, 1998.

K. Shin and T. Kuboyama. A generalization of Haussler's convolution kernel: mapping kernel. In W. Cohen, A. McCallum, and S. Roweis, editors, *International Conference on Machine Learning*, pages 944–951, 2008.

M.-S. Shin and Y.-I. Byun. Efficient Period Search for Time Series Photometry. *Journal of Korean Astronomical Society*, 37:79–85, June 2004.

A.J. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, editors. *Advances in Large Margin Classifiers*. MIT Press, Cambridge, MA, 1999.

O. Söderkvist. Computer vision classification of leaves from Swedish trees. Master's thesis, Linköping University, SE-581 83 Linköping, Sweden, September 2001.

S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf. Large scale multiple kernel learning. *Journal of Machine Learning Research*, 7:1531–1565, 2006.

I. Soszynski, A. Udalski, M. Szymanski, M. Kubiak, G. Pietrzynski, P. Wozniak, K. Zebrun, O. Szewczyk, and L. Wyrzykowski. The Optical Gravitational Lens-

ing Experiment. Catalog of RR Lyr Stars in the Large Magellanic Cloud. *Acta Astronomica*, 53:93–116, June 2003.

A. Srinivasan, S. Muggleton, M. Sternberg, and R. King. Theories for mutagenicity: A study in first-order and feature-based induction. *Artificial Intelligence*, 85(1-2): 277–299, 1996.

B. M. Starr et al. LSST Instrument Concept. In J. A. Tyson and S. Wolff, editors, *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 4836 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 228–239, December 2002. doi: 10.1117/12.457331.

R. F. Stellingwerf. Period determination using phase dispersion minimization. *Astrophysical Journal*, 224:953–960, September 1978. doi: 10.1086/156444.

P. Tsampouka and J. Shawe-Taylor. Analysis of generic perceptron-like large margin classifiers. In *Proceedings of the European Conference on Machine Learning*, pages 750–758, 2005.

K. Tsuda and T. Kudo. Clustering graphs by weighted substructure mining. In Cohen and Moore [2006], pages 953–960. ISBN 1-59593-383-2.

A. Udalski, M. Szymanski, M. Kubiak, G. Pietrzynski, P. Wozniak, and Z. Zebrun. Optical gravitational lensing experiment. photometry of the macho-smc-1 microlensing candidate. *Acta Astronomica*, 47(431), 1997.

L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11): 1134–1142, 1984.

L.G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

S. Vishwanathan, K. Borgwardt, and N. Schraudolph. Fast computation of graph kernels. In *NIPS 19*, 2006.

M. Vlachos, Z. Vagena, P. Yu, and V. Athitsos. Rotation invariant indexing of shapes and line drawings. In O. Herzog, H-J. Schek, N. Fuhr, A. Chowdhury, and W. Teiken, editors, *Conference on Information and Knowledge Management*, pages 131–138. ACM, 2005. ISBN 1-59593-140-6.

G. Wachman and R. Khardon. Learning from interpretations: a rooted kernel for ordered hypergraphs. In Ghahramani [2007], pages 943–950. ISBN 978-1-59593-793-3.

G. Wachman, R. Khardon, P. Protopapas, and C. Alcock. Kernels for periodic time series arising in astronomy. In *European Conference on Machine Learning*, 2009.

A. Woznica, A. Kalousis, and M. Hilario. Kernels over relational algebra structures. In *PAKDD*, pages 588–598, 2005.

T-F. Wu, C-J. Lin, and R.C. Weng. Probability estimates for multi-class classification by pairwise coupling. *Journal of Machine Learning Research*, 5:975–1005, 2004.

L. Wyrzykowski, A. Udalski, M. Kubiak, M. K. Szymanski, K. Zebrun, I. Soszynski, P. R. Wozniak, G. Pietrzynski, and O. Szewczyk. The Optical Gravitational Lensing Experiment. Eclipsing Binary Stars in the Small Magellanic Cloud. *Acta Astronomica*, 54:1–, March 2004.

B. Zadrozny and C. Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *KDD*, pages 694–699. ACM, 2002. ISBN 1-58113-567-X.