Metric Learning and Scale Estimation in High Dimensional Machine Learning Problems with an Application to Generic Object Recognition

by

Nayyar A. Zaidi, BCompSc (Hons)

Thesis

Submitted by Nayyar A. Zaidi

for fulfillment of the Requirements for the Degree of

Doctor of Philosophy (0190)

Supervisor: Dr. David McG. Squire

Associate Supervisor: Prof. David Suter

Clayton School of Information Technology

Monash University

July, 2011

Metric Learning and Scale Estimation in High Dimensional Machine Learning Problems with an Application to Generic Object Recognition

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Nayyar A. Zaidi
July 1, 2011

In the loving memory of my mother

Acknowledgments

I am thankful to my supervisor Dr. David Squire for being an excellent supervisor, a great mentor and a prolific collaborator. David has an excellent ability to balance giving me independence to pursue my ideas while offering insights and a fresh perspective at critical junctures. I am thankful to my associate supervisor Prof. David Suter for all his help especially during the first two years of my candidature.

I am extremely grateful to the Faculty of Engineering Monash University and Monash Research Graduate School for providing me the scholarships for my PhD. I would particularly like to thank Dr. Nallasamy Mani with whom I got in contact when I heard about the opportunity of a PhD position in Monash. I am grateful to Prof. Geoff Webb for his help during my Masters to PhD conversion exam and for working in the role of my supervisor during my transfer from Engineering to Information Technology Faculty. Special thanks to Shiranthi Ponniah for being a wonderful post-graduate coordinator. I do not remember the time, I walked into her office and came out with my problem still unsolved. I am thankful to Prof. Arthur Lowery for providing me the facilities in Electrical Engineering Robotics Lab for all my experiments. I am thankful to Dr. Peter Tischer for chairing the committee of my pre-submission seminar and most importantly for his extremely useful comments on the structure of my thesis and guiding me to interesting new directions and future work.

I am extremely thankful to my sister Mizhgan, my brother-in-law Ali and my gorgeous nephew Aatir for providing me with a home during my four years in Melbourne. I couldn't have asked for a more supportive sister and I thank her for that. Numerous ideas in this dissertation came to me in the vline train which I had to take frequently to visit her in Bendigo.

Last but not least, I would like to thank my mother (ami) and my father (abu). Their love has kept me going in extremely difficult and excruciating times. For their support, well wishes and prayers, a mere 'thank you' can not do any justice. I owe all my successes to them.

Nayyar A. Zaidi

Monash University
July 2011

Metric Learning and Scale Estimation in High Dimensional Machine Learning Problems with an Application to Generic Object Recognition

Nayyar A. Zaidi, BCompSc (Hons)
nayyar.zaidi@monash.edu
Monash University, 2011

Supervisor: Dr. David McG. Squire
david.squire@monash.edu
Associate Supervisor: Prof. David Suter
david.suter@adelaide.edu

Abstract

A typical machine learning algorithm takes advantage of training data to discover patterns among observed variables. For example, a learning algorithm can predict the label of a test data point by measuring its similarity with the training data points. Since the data is constituted of features which are not necessarily related by any evident relation, the notion of similarity is not trivially defined. When measuring similarity, one should not necessarily treat all features as being equally important. The problem of learning in high-dimensional machine learning data is actually the problem of estimating the relative importance of the features. The relevance determination tunes a data-dependent similarity measure. Most machine learning algorithms either implicitly or explicitly learn this similarity measure on which their performance is critically dependent. In this thesis, various metric learning techniques are analyzed and systematically studied under a unified framework to highlight the criticality of data-dependent distance metric in machine learning. The metric learning algorithms are categorized as naive, semi-naive, complete and high-level metric learning, under a common distance measurement framework. The connection of feature selection, feature weighting, feature partitioning, kernel tuning, etc. with metric learning is discussed and it is shown that they are all in fact forms of metric learning. Novel naive, semi-naive, complete and high-level metric learning algorithms are proposed to improve classification performance. Also, it has been shown that the realm of metric learning is not limited to k-nearest neighbor classification, and that a metric optimized in the k-nearest neighbor setting is likely to be effective and applicable in other kernel-based frameworks, for example Support Vector Machines (SVM) and Gaussian Processes (GP).

Contents

List of Tables

List of Figures

List of Algorithms

Symbols and Notations

Table 1: List of Symbols and Notations with their description.

Symbols	Description	Symbols	Description
$\vec{x}$	Column vector.	$\vec{x}^T$	Transpose of the vector $\vec{x}$.
x_p	Value of p-th feature of the vector $\vec{x}$.	$\bar{x}$	Mean feature-vector.
$\langle \vec{x}, \vec{x_i} \rangle$	Dot product of $\vec{x}$ and $\vec{x_i}$.	$\\|\vec{x}\\|^2$	$\langle \vec{x}, \vec{x} \rangle$.
X	Matrix whose columns are the training feature-vectors.	Y	Column vector containing the labels of the training data.
X^*	Matrix whose columns are the testing feature-vectors.		
$y(\vec{x})$	Label of the training point $\vec{x}$.	$\hat{y}(\vec{x})$	Predicted label of the testing point $\vec{x}$.
$\mathcal{X}$	Input space.	$\mathcal{H}$	Transformed space.
$\boldsymbol{\Phi}$	Transformation function.	$\mathbf{x}$	$\vec{x}$ in $\mathcal{H}$.
$k(\vec{x}, \vec{x_i})$	Kernel function.	$d(\vec{x}, \vec{x}_i)$	Euclidean distance between $\vec{x}$ and $\vec{x}_i$.
k	Number of nearest neighbors.	K	Some function of k.
N	Number of training data.	P	Number of features.
C	Number of classes.	$\vec{\mu}$	Same as $\bar{x}$.
Σ	Covariance of the data.	σ^2	Variance of the feature.
$P()$	Posterior probability.	$p()$	Likelihood function.
$\pi()$	Prior probability.	$\delta()$	Discriminant function.
$N_k(\vec{x})$	Neighborhood of $\vec{x}$ containing k data points.	λ	Characteristic length scale parameter of a Gaussian kernel.
I	Identity matrix.	A	$P \times P$-dimensional distance matrix in distance measurement framework.
L	$A = L^T L$.	a_{ij}	Elements of matrix A.
ω	Weight of each feature-vector in high level metric learning.	$\mathcal{FS}(\vec{x})$	Feature-set associated with data point $\vec{x}$.
χ^2	Chi-squared.	$E[.]$	Expected value.
φ	Matrix whose columns are Eigenvectors.	Λ	Diagonal matrix of the Eigenvalues.
$g(.)$	Any objective function.	α	Stepping size in gradient descent algorithm. Also, Lagrange coefficient in SVM formulation.

Continued on next page

Table 1 – continued from previous page

Symbols	Description	Symbols	Description
$f(.)$	An unknown function.	$\hat{f}(.)$	Predicted function.
$\mathcal{E}$	Mean-square-error.	w	Weights of features to be optimized.
V	Votes casted by k neighbors using Gaussian Kernel with Euclidean distance.	W	Votes casted by k neighbors using Gaussian Kernel with Adaptive distance.
e_m	Error rate of any method m.	b_m	Ratio of the error rate of method m and the smallest error rate of all other compared methods.
$\#$	Number of.	M	Confusion matrix.
γ	Weight assigned to the objective function.	ϵ	Random Gaussian noise.
c_p	Index measuring the relevance of feature p.	I_p	Same as c_p.
κ	Small number added for numerical tractability.	$\mathcal{R}$	Region in the input space
Q	Maximum pyramid level.		
F	Strong classifier obtained as a result of Boosting algorithm.	d	Weights associated to each sample in Boosting framework.
D	Normalization factor in Boosting framework.	$h(.)$	Probabilities in Boosting framework.
$\tilde{X}$	Partitions in Boosting framework.	$\tilde{C}_p$	Cost associated with feature p in CRAB framework.
$\mathcal{C}$	miss-classification penalty in SVM formulation.	ξ	Slack variables in SVM formulation.
$\vec{\beta}$	Weights in SVM framework.	β_0	Intercept value in SVM framework.
$\mathcal{L}$	Lagrange primal.		
$\mathcal{N}()$	Gaussian distribution	J	Maximum likelihood estimation objective function.
σ_f^2	Signal variance in GP framework.	σ_n^2	Noise variance in GP framework.
$\mathbf{K}$	Gram matrix in GP framework.		

Chapter 1

Introduction

Machine learning is the study of algorithms that allow machines to find meaningful patterns in the data for inference, prediction or classification purposes. A learning algorithm typically takes advantage of the example data to discover patterns among the observed variables. A serious problem faced by machine learning methods is that the example data (also known as the training data) given to the algorithm as an input is just one set of all the possible inputs. The learning algorithm needs to generalize from the given example data, so as to perform well on unknown data (Bishop, 2006). Various methods and techniques used in machine learning have been studied by researchers in related fields, for example applied statistics, functional approximation, computational neuroscience, data mining, etc. One of the closest fields to machine learning is that of Artificial Intelligence (AI). There are subtle differences between the two fields. Unlike typical AI methods of specifying and hard coding a pattern, machine learning methods learns a pattern from the data. That is, rather than defining the rules and specifying the logic for solving the problem, machine learning methodologies aim to learn the rule, logic or pattern from the training data. Machine learning techniques have turned out to be more fruitful than artificial intelligence methods in many applications. Over the last two decades, machine learning algorithms have been applied successfully to a huge range of applications in engineering, medicine and sciences. From weather prediction to medical diagnosis, from human visual tracking to malware detection in computer network security, from online gambling to stock prediction, the list of applications of machine learning is exhausting.

Most machine learning algorithms deal with the learning of a pattern from examples (training data) that are in high dimensions. The dimensions are typically known as features in machine learning. Though it is difficult to visualize beyond two or three dimensions, one can think intuitively of the training data as points in some high-dimensional space (also referred as a feature space). The problem of learning in machine learning is in fact the problem of estimating the right properties of the space in which the data lies. Mathematicians have frequently generalized their theories to P dimensional spaces, so one might assume that the mathematics of high dimensions can be easily applied to machine learning data sets. This assumption is not true. Calculations in lower dimensional spaces can only be generalized to higher dimensional spaces if we have a clear understanding of the properties of space we are working in. For example Euclidean space, Hilbert space, etc. have well defined mathematical properties (Bachman et al., 2000; Bourbaki, 1986). The trouble with machine learning data sets is that the data often lies in a space where the dimensions are not related. That is, features are not equally important, often come from different sources, and are of totally different natures. One of the consequence of such a space is that the distances between the data points are arbitrary. For example, if one of the features represents a temperature value, the distance between two points is different depending on whether the temperature is measured in degrees Celsius or degrees Fahrenheit. As a result the notion of similarity is not well defined. That is, we can not take two vectors and determine their similarity simply by measuring the distance between them, or by calculating the projection of one vector over the other as we do in the Euclidean space. How to learn patterns in the data that lies in such a space is the crux of machine learning, and finding this relationship (pattern) among different features is the goal of much machine learning research.

1.1 Thesis Overview

As described, machine learning deals with data in a space where features may be unrelated, there is a need to deal with these features in a systematic and principled manner so that distances between the data points are not affected and a suitable measure of similarity exists for learning patterns in the data. My main hypothesis is:

> Due to the inherent nature of machine learning data sets, any learning algorithm relying on similarity measurement to predict the class labels needs to estimate or learn the similarity measure from the training data. In fact, most learning algorithms either implicitly or explicitly estimate this similarity measure, and their performance depends critically on how well such a measure of similarity is estimated. The estimation of a similarity measure can be referred to as 'learning of a data-dependent distance metric' or 'metric learning'. Several techniques such as feature selection, feature scaling, feature relevance, feature partitioning, kernel tuning, kernel scaling, kernel fusion, distance fusion, etc., that are used for making learning algorithms more efficient, are in fact metric learning methods. That is, they are doing nothing but estimating a measure of similarity that is suitable for the data.

The need to estimate a similarity metric arises because of the heterogeneous structure of the input space in which the data lies. Therefore, the estimation of a data-dependent similarity measure is in fact estimation of the properties of an input space. One can deduce that the performance of learning algorithms will depend on how well these properties are estimated. It is important to make a distinction between the properties of the learning algorithm and the properties of the input space. The properties of an input space depends on data distribution and are independent of any learning algorithm.

My secondary hypothesis is:

> A similarity measure optimized in the framework of one learning algorithm is likely to be applicable and effective in those of other learning algorithms.

1.1.1 Contributions

The main contributions of the thesis are as follows.

A novel categorization scheme is introduced to categorize metric learning methods. Such a categorization leads to the unification of different terms used for metric learning. The methods are categorized as naive, semi-naive, complete, and high-level metric learning categories under a common distance measurement framework. The categorization depends on the form of the distance matrix[1]. For example naive and semi-naive metric learning is the case of learning a diagonal distance matrix, complete metric learning is the case of learning a full distance matrix (i.e. diagonal and off-diagonal terms) whereas, high-level metric learning is the case of learning a block partition distance matrix. As mentioned above, several different terms are used for metric learning in the literature, for example feature selection, feature weighting, feature partitioning, scale estimation, kernel tuning, etc. By analyzing and categorizing these methods under a common framework, it has been shown that all these methods are in fact special cases of metric learning. The details of this categorization scheme are discussed in chapter 3.

A novel algorithm for semi-naive and complete metric learning has been proposed. The Mean-Square-Error Gradient Minimization (MEGM) metric learning algorithm is based on minimizing the gradient of the Mean-Square-Error (MSE) in a nearest neighbor framework. The motivation behind the algorithm is to reduce the bias in high dimensions that originates due to the curse of dimensionality. It has been shown that the proposed algorithms perform better than most state of the art metric learning algorithms in the k-nearest neighbor framework. The details of the algorithm are discussed in chapter 4, and 5 (Zaidi et al., 2010a,c).

The connection between kernel learning in the SVM and GP classification frameworks and metric learning in the k-nearest neighbor framework has been studied. It has been shown that metric learning algorithms in the k-nearest neighbor framework can be used to tune kernel parameters for SVM and GP classification. To the best of our knowledge, this is the first work that deals with learning of a full data-dependent distance metric (complete metric learning) for kernel tuning in the SVM and GP classification frameworks. On a wide range of data sets, it has been shown that the classification performance of both SVM and GP classifiers can be significantly improved by using kernel parameters learned in a k-nearest neighbor framework, as compared to the kernels obtained by other state of the art methods, such as cross-validation. The details of the method are discussed in chapter 7 and 9 (Zaidi and Squire, 2010b).

An algorithm for local adaptive SVM (LASVM) classification has been proposed. The motivation behind such a formulation is that, unlike k-nearest neighbor classification, the SVM framework is based on binary classifiers and hence does not scale well with the number of classes. Despite its classification efficiency, the SVM framework is not always computationally efficient. To combine the computational efficiency of a k-nearest neighbor classifier and the classification efficiency of an SVM classifier, one can use

[1]It should be noted that if distance matrix is the covariance matrix of the data, it will lead to 'Mahalanobis' distance. An identity matrix on the other hand, will lead to a simple Euclidean distance.

an SVM locally, i.e. train an SVM classifier in the neighborhood of the query point. The size of the neighborhood in which an SVM classifier is to be trained should be kept small for computational efficiency, but the performance of the local SVM is degraded by decreasing the neighborhood size. In this work, it has been shown that by training an SVM in an adaptive neighborhood (altering the neighborhood by learning a data-dependent distance metric), excellent classification performance can be achieved even in a small neighborhood. The details of the algorithm are discussed in chapter 8 (Zaidi and Squire, 2010a).

A novel algorithm for local adaptive metric learning for k-nearest neighbor classification has been proposed. The idea is that, rather than learning a global distance metric, one can learn a metric in the neighborhood of the query point. This will result in an adaptive neighborhood for k-nearest neighbor classification. The feature relevance index in the proposed algorithm is inspired by the Boosting classification algorithm. It has been shown that the proposed algorithm performs better than the other competing local adaptive metric learning algorithms. The details of the algorithm are discussed in chapter 6 (Zaidi et al., 2010b).

The problem of combining different features for object recognition in a systematic way has been addressed under the proposed metric learning categorization scheme. Two information fusion schemes, classifier fusion and distance fusion, have been studied with object categorization and object detection tasks. A novel Boosting learning algorithm, which is a modification of Adaboost algorithm, named Confidence-Rated-Adaptive-Boosting (CRAB) for generic object classification has also been proposed. CRAB is a domain partitioning adaptive Boosting algorithm in which each weak classifier partitions the input space and predicts the label of each partition. The details of the proposed information fusion schemes and CRAB algorithm are discussed in chapter 10 (Zaidi and Suter, 2008a,b).

The algorithms listed in the 'list of algorithms' at the start of the thesis are novel.

1.2 Roadmap

The thesis is structured into nine chapters excluding the introduction and conclusion chapters. The nine chapters can be divided into four parts. The chapters, algorithms and the relevant appendices referred to from each part are listed in table 1.1. The parts are introduced in this chapter only to give a broad overview of the thesis and to depict the dependencies between the chapters (figure 1.1). Parts 2, 3 and 4 are dependent on part 1. However, there are no dependencies between parts 2, 3 and 4.

Part	**Chapter**	**Algorithm**	**Appendix**	**Reference**
Part 1	2, 3		A, B, C	
Part 2	4, 5, 6	1, 2, 3, 4, 5	F	(Zaidi et al., 2010a,c,b)
Part 3	7, 8, 9	6, 7, 8, 9, 10	D, E, F	(Zaidi and Squire, 2010b,a)
Part 4	10	11	A, C	(Zaidi and Suter, 2008a,b)

Table 1.1: List of chapters, algorithms, relevant appendices, and relevant references for each part.

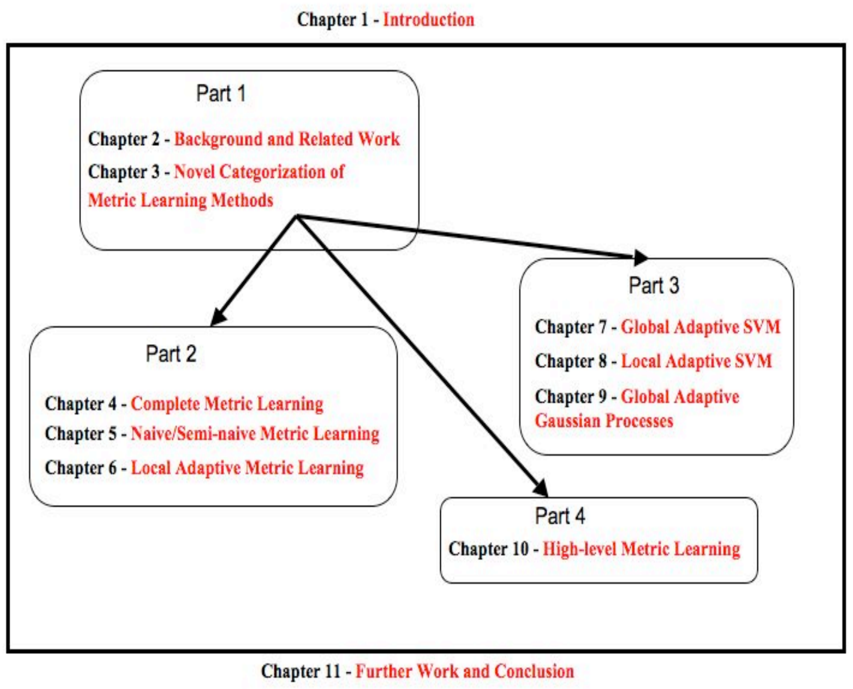

Chapter 1 - Introduction
Part 1
Chapter 2 - Background and Related Work
Chapter 3 - Novel Categorization of Metric Learning Methods
Part 2
Chapter 4 - Complete Metric Learning
Chapter 5 - Naive/Semi-naive Metric Learning
Chapter 6 - Local Adaptive Metric Learning
Part 3
Chapter 7 - Global Adaptive SVM
Chapter 8 - Local Adaptive SVM
Chapter 9 - Global Adaptive Gaussian Processes
Part 4
Chapter 10 - High-level Metric Learning
Chapter 11 - Further Work and Conclusion

Figure 1.1: Figure depicting the roadmap of the thesis.

In the following, the details of each part are given.

Part 1 constitutes chapters 2 and 3 and builds the foundation for the research conducted in the thesis. This part introduces the terminologies and sets the stage on which the algorithms in the later parts are proposed. Chapter 2 formally introduces the background and the related work and deals with the problems addressed in this dissertation, why they are significant, how other researchers have handled them, what the issues are that needs to be addressed, and how these issues will be addressed in this work. In chapter 3, a novel categorization of metric learning methods is introduced, and naive, semi-naive, complete, and high-level metric learning methods are illustrated with a simple classification example.

Part 2 constitutes chapter 4, 5 and 6 and deals with metric learning in the k-nearest neighbor framework. Metric learning algorithms are proposed in both global and local settings. The algorithms proposed in this part are: MEGM (algorithm 1), GML (algorithm 2), MEGM-SNML (algorithm 3), BoostML1 (algorithm 4) and BoostML2 (algorithm 5).

Part 3 constitutes chapters 7, 8 and 9 and deals with the application of metric learning methods in the k-nearest neighbor framework to the SVM and GP classification frameworks. The algorithms proposed in this part are: GASVM1 (algorithm 6), GASVM2 (algorithm 7), LASVM (algorithm 8), GPML1 (algorithm 9) and GPML2 (algorithm 10).

Part 4 constitutes chapter 10 only and illustrates high-level metric learning with scene categorization and object detection scenarios.

Chapter 2

Background and Related Work

In this chapter, the foundation for the research is laid and the related work is discussed. A general overview of the basic strategies employed by most machine learning algorithms is provided and the issues such as similarity measurement and distance calculation are highlighted. The goal here is not to give a comprehensive summary of these learning algorithms, but to introduce the building blocks that will be used in the later chapters. After explaining typical learning strategies, metric learning will be introduced and its impact on the performance of learning algorithms, for example k-nearest neighbor classification, SVM classification, etc., will be explained. In the light of the hypothesis given in section 1.1, various known issues in the current formulations of learning algorithms and metric learning algorithms will be discussed. Proposals to address these issues will be provided in the conclusion of the chapter.

Let us make a distinction about how the word 'feature' will be used, as it can lead to confusion. Throughout this thesis, a set of values concatenated together in a vector is denoted as a 'feature-vector' (multi-variate case). Each element of the 'feature-vector' is denoted as a 'feature'. If there is only one feature, it will be addressed as a 'feature' (univariate case). For example, let us suppose that we are given the task of classifying two kinds of fish. For this classification task, we can use attributes such as the length, weight and color of the fish. The 'length', 'weight' and 'color' of the fish are example of features and constitute a 3-dimensional feature-vector representing each data object i.e., each instance of fish. Based on these feature-vectors, we want to learn the kind of fish. It should be noted that we have considered three features (attributes) only but there can be countless different features for example smell, presence of fins, place trapped, etc. Determining which features are more correlated with each other, and how much features are correlated with the class labels, are questions that goes deep into the heart of machine learning.

A collection of different *types* of 'feature-vector' is denoted as a 'feature-set'. For example shape, color, texture, etc. of an object represent possible *types* of feature-vector for object recognition problems. The *types* of feature-vector means: that either feature-vector represents different sort of information for example color, shape, etc. or it represents a different measurement of the same sort of information for example a feature-vector based on RGB (red, green, blue) and a feature-vector based on HSV (hue, saturation, value), etc.

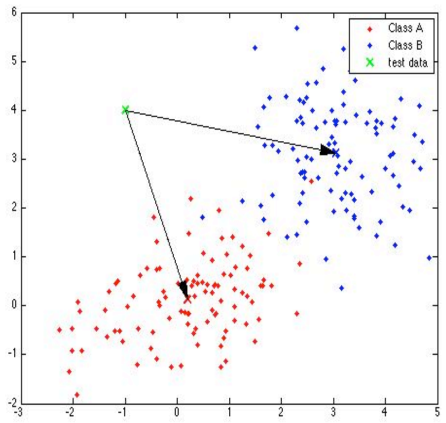
are two different feature-vectors representing the color information. As will be discussed in section 2.4, a feature-vector can be pre-processed, for example it can be re-scaled to compensate for the fact that the learning algorithm is assuming data to be isotropic, or it can be projected on some lower dimensional subspace for efficiency purposes. Any pre-processing function has some tunable parameters. Different values of these parameters can result in generating different feature-vectors. In such cases, a feature-set is constituted of feature-vectors which are in fact multiple measurements of the same information.

Based on the previous discussion of features, feature-vectors and feature-sets, one can say that there are two types of problem in machine learning based on the nature of data. The first kind of problem represents the case where there is a natural partitioning among features and each data object is represented as a feature-set. For example, features can be partitioned into multiple feature-vectors based on their units, source, etc. Let us consider the object recognition problem (Varma and Ray, 2007; Leibe et al., 2007a; Opelt and Pinz, 2006). As discussed before, many cues, for example shape, color, texture, context, etc., can be used to discriminate objects from each other. Let us suppose that shape, color, texture and context information is represented by scale-invariant-feature-transform (SIFT), HSV, Textons and Gabor filters based feature-vectors respectively, as will be discussed in section 3.4 (Lowe, 1999; Varma and Zisserman, 2005; Oliva and Torralba, 2001). Hence, each data object (or image of an object) is represented as a feature-set. It may be natural to treat feature-vectors in the feature-sets separately. In such a kind of problem, where data is represented as feature-sets, we can consider the training data to exist in more than one input space. Since we do not know the relation between the features in the feature-vectors and also between the input spaces, perhaps the best strategy is to deal with them separately. That is, consider them as different learning problems. A motivation behind such a strategy is that features in the same input space might be more correlated with each other or with the class labels and, therefore, may exhibit similar behavior.

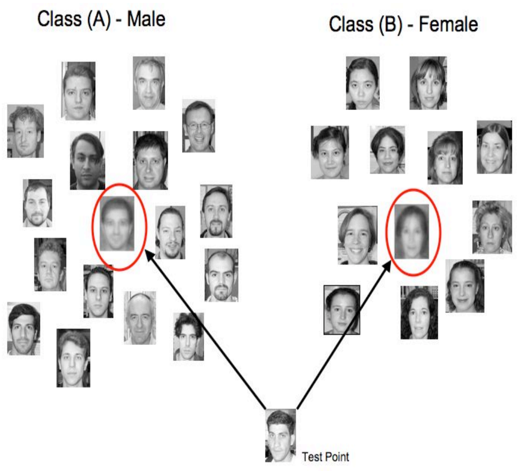
The second kind of problem represents the case where no such partitioning among features exist and each data object is represented as a feature-vector (Mangasarian et al., 1995; Iba et al., 1988). In this case, the features can not be treated separately. For example length, weight and color features might seem to be highly correlated for fish classification, and should be treated together as a feature-vector (Duda et al., 2006). Although one could always argue for treating these features separately, they may not be able to discriminate between the two kinds of fish if that is done. Such is the delicacy and intricacy of machine learning data. We do not know in advance the relevance of each feature or the relevance of subsets of features. Trying to determine the relevance of each subset of features may lead to combinatorial problems that are very difficult to solve due to computational issues. A detailed analysis and examples of each of these two kinds of problem is given in appendix A.

2.1 Basic Machine Learning Strategies

No matter how the examples (data points) are represented, that is either in the form of a feature-vector or feature-set, the goal of a learning algorithm is to learn a pattern



(a)



(b)

Figure 2.1: Examples demonstrating the fundamental problem in machine learning. Figure 2.1(a): a trivial example of classification in two dimensions. Figure 2.1(b): a complicated face classification problem.

among variables from these examples. In this section, let us assume that each example is represented as a feature-vector. In a typical machine learning scenario of supervised learning, we are provided with a matrix X whose columns are the feature-vectors ($\vec{x}$) representing each training example, and a column vector Y (consisting of either $+1$ or -1) representing the class labels of each feature-vector in X. The fundamental problem in machine learning is to infer the labels of feature-vectors in matrix X^* containing testing data, from X and Y. A distinction can be made between generative and discriminative learning. In generative learning, a model representing the distribution of the data is sought (Blei et al., 2003; Hoffmann, 2001). It is called generative as one can generate the synthetic data points in the input space by sampling from the model. Discriminative learning, on the other hand, models the posterior class distribution and uses decision theory to assign a query point to one of the classes (Friedman et al., 2000).

To illustrate the basic strategies that most machine learning algorithms employ, two simple problems representing the fundamental problem of classification in machine learning are considered (see figure 2.1). Figure 2.1(a) represents a case of 2-dimensional training data. Each training data point belongs to one of the two classes, either A or B. The goal is to predict the label of a test data point based on this information. Figure 2.1(b) represents the problem of classifying male and female faces. We are given sets of male and female faces as the training data. The goal again is to predict the label of the testing image (that is categorize it as male or female). Let us suppose that the faces are represented as a 16×16 matrix of gray-scale intensity values. This means that each face is actually a point in a 256-dimensional feature space. We can not visualize this space as we did for the 2-dimensional problem in figure 2.1(a). However, for comparison, faces in figure 2.1(b) are represented just like the 2-dimensional data in figure 2.1(a). The mean of each class is also shown in both examples. The distance between the test point and the mean of each class is shown as an arrow from the test point to the class mean. In summary, in both examples, we are given some training data, and, based on this information, the goal is to predict the label of a test data point. A basic strategy to assign a label to the test point is:

- Calculate the mean of each class.
- Calculate the *similarity* of the test point to the mean of each class.
- Assign test point to the class to whose mean it is more *similar*.

This is a an extremely simple strategy but it forms the bedrock of many sophisticated machine learning algorithms. The only problem with the above formulation is the concept of *similarity.* The notion of similarity is a deep question that lies at the core of machine learning (ten Brinke, 2010). What choices of similarity measure do we have? In the following, some very general similarity measures are discussed:

- One can use the famous distance measurement class known as the 'Minkowski metric' shown below:

  $$d(\vec{x}_1, \vec{x}_2) = \left(\sum_{p=1}^{P} |x_{1p} - x_{2p}|^q \right)^{1/q} \tag{2.1}$$

  Setting $q = 2$ gives the familiar Euclidean metric which is a very common measure of similarity. A Minkowski metric is appropriate only if the feature space is isotropic. This measure is invariant to translations and rotations of the feature space, but it is not invariant to transformations that distort the feature space, such as sheer or scaling. One way to achieve such invariance is to normalize the data. For example, to obtain invariance to scale changes, one can scale the axes such that all the features have unit variance. An alternative to scaling the axis is to change the metric. Both processes are exactly equivalent. That is, the effects of metric modification are the same as those of data transformation.

- Since each training data point is represented as a feature-vector, it is useful to look at the similarity issue from a geometric point of view. Such a formulation is beneficial as it allows one to analyze the problem of similarity from the perspective of linear algebra and analytical geometry. For example, the dot-product is a simple form of a similarity measure between the feature-vectors.

- Another alternative is to use a metric based on the data itself, for example, the Mahalanobis distance (Mahalanobis, 1936). A metric can also be learned from the data. This is typically known as metric learning. As will be discussed in section 2.4, such a form of data-dependent distance metric can be extremely useful.

The abovementioned measurement methods constitute some of the important classes of similarity measurement. There are numerous other measures that can be used. In the following, three basic machine learning strategies that employ these similarity measures to predict class labels are illustrated.

2.1.1 Template Matching

To make a prediction about the label of the test point using the training data (X,Y), we have to make assumptions about the process that generated the training data. Let us assume that the training data for the problems in figure 2.1 is generated from two Gaussian distributions (one for each class). Since a Gaussian distribution is specified by the mean and the covariance matrix, there are several possibilities when it comes to specifying the distributions:

- We can assume that the features are statistically independent and each feature has the same variance.
- We can assume that the features are not statistically independent but all classes have the same covariance matrix.
- And third, we can assume that the features are not statistically independent and each class has a separate covariance matrix.

Let us consider the implications for distance measurement of each of these choices. If the features are assumed to be statistically independent and have the same variance σ^2, the covariance matrix is actually a diagonal matrix, merely being σ^2 times the identity matrix I. To classify a point $\vec{x}$, we can measure its squared distance to the mean of each class. The mean $\vec{\mu}_c$ is simply the mean of data points in the class c. The distance between $\vec{x}_i$ and $\vec{\mu}_c$:

$$\frac{1}{\sigma^2}(\vec{x}_i - \vec{\mu}_c)^T(\vec{x}_i - \vec{\mu}_c) \tag{2.2}$$

The testing point $\vec{x}_i$ is assigned the label of the class whose mean is closest to it. This is an extremely simple approach and with some modifications it can become very effective.

The problem with above formulation is the same that arises due to the use of Euclidean distance. That is, the distance is not invariant to scale and assumes isotropy. In most machine learning data sets, this is not the case. Features are not independent and variance is different for each each dimension. Let us consider the case of a common covariance matrix Σ for all classes. If c is the index running over the classes, we have $\Sigma_c = \Sigma\ \forall c$. We can reiterate the solution to the problems in figure 2.1 as: to classify a test point $\vec{x}_i$, measure the Mahalanobis distance from $\vec{x}_i$ to the mean of each class, that is:

$$(\vec{x}_i - \vec{\mu}_c)^T\Sigma^{-1}(\vec{x}_i - \vec{\mu}_c) \tag{2.3}$$

and assign $\vec{x}_i$ to the class whose mean feature-vector is the closest. The distance formulation $(\vec{x}_i - \vec{\mu}_c)^T\Sigma^{-1}(\vec{x}_i - \vec{\mu}_c)$ between $\vec{x}_i$ and $\vec{\mu}_c$ is called the Mahalanobis distance[1] between

[1]The Mahalanobis distance is needed because the features are incommensurate in most machine learning data sets. For example consider salary and height of the person in dollars and meters respectively as two features constituting the feature-vector. Clearly a unit change in the height (one meter) is greatly significant as opposed to a unit change (one dollar) in the salary of a person. We need a metric that takes into account the dispersion of the data. This is the motivation behind the Mahalanobis distance, where Σ^{-1} is the inverse covariance matrix and actually determines the shape of the data cluster.

$\vec{x}_i$ and $\vec{\mu}_c$. Such classification is typically called a 'minimum distance classifier' and if each mean is an ideal prototype or a template pattern for its class, then this is essentially a 'template matching' procedure. It should be noted that we have not considered the effect of the prior probabilities of each class, assuming that each class is equally probable. If we consider the prior probabilities of each class in the above formulation of distance measurement, the strategy is known as Linear Discriminant Analysis (LDA) (Fischer, 1936; Duda et al., 2006). After some simplification, the confidence in the prediction of class c can be written as:

$$\delta_c(\vec{x}) = \vec{x}^T \Sigma^{-1} \vec{\mu}_c - \frac{1}{2} \vec{\mu}_c^T \Sigma^{-1} \vec{\mu}_c \tag{2.4}$$

Similarly if we suppose that each class in the training data has a different covariance matrix, the confidence in the prediction of class c can be written as:

$$\delta_c(\vec{x}) = -\frac{1}{2} \log |\Sigma_c| - \frac{1}{2} (\vec{x} - \vec{\mu}_c)^T \Sigma^{-1} (\vec{x} - \vec{\mu}_c) \tag{2.5}$$

The above formulation is known as 'Quadratic Discriminant Analysis' (QDA) (Duda et al., 2006). Again, the effect of prior probabilities in equations 2.4 and 2.5 have been ignored for simplification. A complete derivation and analysis of linear and quadratic discriminant analysis in the light of distance measurement is provided in appendix B.

2.1.2 Kernel-based Strategies

The second strategy employs the use of the dot-product as a similarity measure (Shawe-Taylor and Cristianini, 2004; Scholkopf and Smola, 2004). The template matching strategy discussed in section 2.1.1 relies on either Euclidean or Mahalanobis distance to predict the class labels. In this section, the template matching strategy is interpreted in terms of a dot-product similarity measure. It should be noted that we have switched from the distance measure, which is small for similar data points, to the similarity measure, which is large for similar data points.

It is important to note that in order to consider the dot-product as a similarity measure, we have to assume that the training data lies in a dot-product space. Such an assumption is not always true. Therefore, it is imperative to transform data from input space to some dot-product space, let us say $\mathcal{H}$. If the input space is denoted as $\mathcal{X}$, we are interested in the following mapping:

$$\begin{aligned} \mathbf{\Phi} : \mathcal{X} &\rightarrow \mathcal{H} \\ \vec{x} &\rightarrow \mathbf{x} \equiv \mathbf{\Phi}(\vec{x}) \end{aligned} \tag{2.6}$$

The transformed space $\mathcal{H}$ is a space where using the dot-product as a similarity measure between the data points is permissible. It can be seen that the problem of learning has been transformed into the problem of finding an appropriate transformation ($\mathbf{\Phi}$) of the input space $\mathcal{X}$ to a space $\mathcal{H}$ where similarity is defined effectively. As a different transformation function will lead to a different dot-product space having a different similarity measure, the transformation step is extremely important.

The use of the dot-product as a similarity measure has gained a lot of interest in machine learning. A strong reason for using dot-products as similarity measures actually stems from the 'kernel trick' (Scholkopf and Smola, 2004). The kernel trick suggests that we do not have to actually transform the data into a dot-product space $\mathcal{H}$ and then compute the dot-product to calculate similarity. By virtue of the kernel trick, we can bypass the whole transformation ($\mathbf{\Phi}$) step by finding a suitable similarity function k (known as the kernel) that when given two data points $\vec{x}_1$ and $\vec{x}_2$ returns a real number characterizing their similarity. The kernel (k) is generally symmetric, that is $k(\vec{x}_1, \vec{x}_2) = k(\vec{x}_2, \vec{x}_1)$. It gives the ability to process the data in the input space $\mathcal{X}$ rather than finding the transformation function $\mathbf{\Phi}$. Using a kernel function to measure similarity in $\mathcal{X}$ is exactly equivalent to computing the similarity by the dot-product in the transformed space $\mathcal{H}$. The need to find a transformation function $\mathbf{\Phi}$ has been transformed into finding an appropriate kernel function $k(.,.)$.

Let us analyze the template matching strategy of section 2.1.1 in terms of dot-products and kernels. It will be assumed in the following discussion that we have an appropriate kernel function $k(.,.)$ and therefore, we have access to the desired transformation $\mathbf{\Phi}$. This means that we can measure the similarity in the transformed space $\mathcal{H}$ using the dot-product. We can use the template matching strategy in the transformed space $\mathcal{H}$ as we did in the input space $\mathcal{X}$ in section 2.1.1. Let us represent the means of the two classes as:

$$\begin{aligned} \mu_{c1} &= \frac{1}{N_{c1}} \sum_{(i \mid y_i=+1)} \mathbf{x_i} \\ \mu_{c2} &= \frac{1}{N_{c2}} \sum_{(i \mid y_i=-1)} \mathbf{x_i} \end{aligned} \tag{2.7}$$

where N_{c1} and N_{c2} are the number of data points in classes $c1$ and $c2$ respectively and $\mathbf{x}$ denotes the feature-vector in the transformed space $\mathcal{H}$. Following the template matching strategy, we can assign a new data point $\mathbf{x}$ to the class whose mean is closest to $\mathbf{x}$. Let us analyze this nearest mean strategy (template matching) in terms of the dot-product $\langle .,. \rangle$. In the transformed space $\mathcal{H}$, we can predict the class label of the testing point $\mathbf{x}$ by computing the difference between the dot-product of $\mathbf{x}$ with μ_{c1} and the dot-product of $\mathbf{x}$ with μ_{c2}. Therefore, we can write:

$$\begin{aligned} \delta(\vec{x}) &= \operatorname{sign}(\langle \mathbf{x} - (\mu_{c1} + \mu_{c1})/2, (\mu_{c1} - \mu_{c2}) \rangle) \\ &= \operatorname{sign}(\langle \mathbf{x}, \mu_{c1} \rangle - \langle \mathbf{x}, \mu_{c2} \rangle \ + b) \end{aligned} \tag{2.8}$$

where b is the offset term and is equal to:

$$b = \frac{1}{2}(\|\mu_{c2}\|^2 - \|\mu_{c1}\|^2) \tag{2.9}$$

If the class means have the same distance from the origin, then b will vanish. We can write equation 2.8 in terms of the class means as:

$$\begin{aligned}\delta(\vec{x}) &= \operatorname{sign}\left(\frac{1}{N_{c1}}\sum_{(i\mid y_i=+1)}\langle \mathbf{x},\mathbf{x_i}\rangle - \frac{1}{N_{c2}}\sum_{(i\mid y_i=-1)}\langle \mathbf{x},\mathbf{x_i}\rangle \ +b\right)\\ &= \operatorname{sign}\left(\frac{1}{N_{c1}}\sum_{(i\mid y_i=+1)}k(\vec{x},\vec{x}_i) - \frac{1}{N_{c2}}\sum_{(i\mid y_i=-1)}k(\vec{x},\vec{x}_i) \ +b\right) \qquad (2.10)\end{aligned}$$

Equation 2.10 depicts a simple nearest mean (template matching) based strategy for predicting the labels of the testing data using kernels. The kernel $k(\vec{x},\vec{x}_i)$ plays a pivotal role in a classification scheme employing such a strategy. There are different forms of kernels, for example linear ($\langle\vec{x},\vec{x}_i\rangle$), polynomial ($\langle\vec{x},\vec{x}_i\rangle^d$), Gaussian, etc., and kernel construction is the point of much research in machine learning. A simple and a widely used Gaussian kernel takes the form:

$$k(\vec{x},\vec{x}_i) = \exp\left(-\frac{(\vec{x}-\vec{x}_i)^T(\vec{x}-\vec{x}_i)}{2\lambda^2}\right) \qquad (2.11)$$

λ in equation 2.11 is a tunable scaling parameter. As λ influences the transformation of $\mathcal{X}$ into $\mathcal{H}$, it is an extremely important variable that controls the classification effectiveness. Kernels are at the core of many state of the art machine learning algorithms, e.g., SVM, GP classifiers, etc.

2.1.3 Nearest neighbor strategies

Nearest neighbor methods for pattern recognition have proven to be very useful in machine learning (Cover and Hart, 1967). Despite the simplicity, their performance is comparable to other sophisticated classification and regression techniques such as SVM and GP, and they have been applied to a wide variety of problems (Shakhnarovich et al., 2006; Goldberger et al., 2005). Computer vision research has benefited greatly from advancements in nearest neighbor methods, for example some state of the art techniques for object recognition are based on nearest neighbor analysis (Frome et al., 2006; Nilsback and Zisserman, 2006). A simple demonstration of the k-nearest neighbor method on the male versus female classification example from figure 2.1(b) is shown in figure 2.2.

For a given query point $\vec{x}_0$, a k-nearest neighbor classifier predicts the label of $\vec{x}_0$ by finding an average of the class labels in the neighborhood of $\vec{x}_0$. The neighborhood is represented as $N_k(\vec{x}_0)$ which denotes the k closest neighbors of $\vec{x}_0$. The term "closest" in the specification of the neighborhood $N_k(\vec{x}_0)$ implies that we have access to a measure of similarity, i.e. an appropriate distance metric. Let us suppose that the metric used is the Euclidean distance. We can write k-nearest neighbor prediction $\hat{y}$ as:

$$\hat{y}(\vec{x}_0) = \frac{1}{k}\sum_{\vec{x}_i\in N_k(\vec{x}_0)} y_i \qquad (2.12)$$

Since the predicted function $\hat{y}$ is discontinuous in $\vec{x}$, the decision boundary obtained by equation 2.12 changes abruptly. Due to this discontinuity, the above formulation of k-nearest neighbors is rarely used in machine learning. A common strategy to overcome such discontinuity is: rather than assigning each point in the neighborhood equal weight, we can assign weights that die off smoothly with the distance from the query point. This leads to a smooth predicted function:

$$\hat{y}(\vec{x}_0) = \frac{\sum_{i=1}^{K} k_\lambda(\vec{x}_0, \vec{x}_i) y_i}{\sum_{i=1}^{K} k_\lambda(\vec{x}_0, \vec{x}_i)} \tag{2.13}$$

where $k_\lambda(\vec{x}_0, \vec{x}_i)$ is a kernel parameterized by λ that specifies the similarity between $\vec{x}_0$ and $\vec{x}_i$. A well known example of a kernel in the nearest neighbor framework is the Epanechinikov quadratic kernel (Hastie et al., 2001). An Epanechinikov quadratic kernel between $\vec{x}_0$ and $\vec{x}_i$ is defined as:

$$k_\lambda(\vec{x}_0, \vec{x}_i) = D\left(\frac{|\vec{x}_0 - \vec{x}_i|}{\lambda}\right) \tag{2.14}$$

where

$$D(z) = \begin{cases} 3/4(1 - z^2) & \text{if } |z| \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

Another example of a widely used kernel for nearest neighbor methods is the Gaussian kernel in equation 2.11. As will be discussed in section 2.4, the λ parameter of the Gaussian kernel plays an important role in the classification performance of nearest neighbor methods.

One of the real advantages of the k-nearest neighbor classifier is its simplicity. A k-nearest neighbor classifier deals with the multi-class classification scenario almost effortlessly. On the other hand, we must resort to one-versus-one and one-versus-all techniques to deal with the multi-class scenario in the case of binary classifiers such as SVM and GP classifiers, which makes them computationally expensive. In the one-versus-all scenario, each classifier distinguishes between one of the classes and the rest. Classification of a query point is done by a winner-takes-all strategy, in which the classifier with the highest output function assigns the class label. In the one-versus-one strategy, each classifier distinguishes between every pair of classes. Classification of a query point is done by a max-wins voting strategy, in which every classifier assigns the instance to one of the two classes, then the vote for the assigned class is increased by one vote, and finally the class with most votes assigns the class label. As k-nearest neighbor classification involves no training, it is computationally efficient. The second advantage of k-nearest neighbor methods stems from their asymptotic properties. The asymptotic analysis of k-nearest neighbor methods suggests that k-nearest neighbor classification achieves the accuracy of a Bayes optimal classifier provided the number of training data is not too small (Cover, 1968; Fix and Hodges, 1951; Snapp and Venkatesh, 1998). This suggests that if we have a large number of training data, a k-nearest neighbor classifier can perform as well as any other sophisticated alternative classifier, such as the SVM.

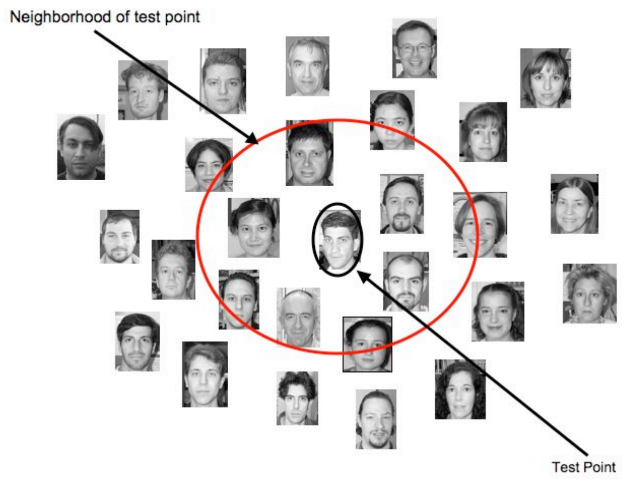


Figure 2.2: A simple demonstration of the k-nearest neighbor strategy for predicting the label of the test point. The problem depicted is of determining the label of a face image (either male or female). Refer to figure 2.1(b) for details.

2.2 Scaling Parameters

All of the learning strategies discussed in section 2.1 rely on similarity measurement or distance calculation. Scaling parameters appear in all of these learning strategies. For example, the scaling parameter Σ in the template matching strategy (equation 2.4), scaling parameter λ in the kernel-based strategy (equation 2.11), and nearest neighbor classification (equation 2.13). It is clear that the goal of Σ in the case of template matching is to compensate for different scales across the axes.

The λ parameter in equations 2.11 and 2.13 actually controls the size of the neighborhood. It defines a metric window size. For example, $\lambda = 1$ or $\lambda = 30$ defines neighborhood sizes of one or 30 units respectively for each query point. By specifying the neighborhood size, λ actually controls the smoothness of the predicted function. Smoothness is highly desirable for achieving generalization. We can always train a classifier that classifies the training data perfectly. Such classification behavior is not sufficient because we want the classifier to perform well on unseen data too. A classifier performing well on the training data might be over-fitted. Intuitively, it can be seen that if the predicted function is over-fitted, rather than deducing a pattern in the data it has learned that particular instance of the data. Over-fitting leads to poor generalization and can be avoided if we adjust our prediction to do some averaging around the neighborhood of each query point. This simple procedure goes by the name of smoothing, and is one of the lynch-pins of the techniques to make machine learning algorithms more effective. The idea is simple. We just assume that the predicted function is smooth at least in the neighborhood of the query point, i.e. the function does not change its value rapidly and exhibits a regular behavior in the neighborhood.

More formally, we can analyze λ from the bias and variance perspective. Since MSE:

$$\begin{aligned} \text{MSE} &= \text{bias}^2 + \text{variance} \\ E[(\hat{y} - y)^2] &= [E(\hat{y} - y)]^2 + E[\hat{y} - E[\hat{y}]] \end{aligned}$$

consists of bias and variance terms, any learning algorithm needs to reduce the bias as well as the variance of its classification. The bias term of MSE measures the accuracy or quality of the match, high bias implies poor match. The variance term of MSE measures the precision of the match, again a high variance implies a poor match. Generally an increase in the variance causes a decrease in the bias and vice-versa (Hastie et al., 2001). The size of the neighborhood λ influences the bias and variance of classification. There is a need to choose λ such that both the bias and the variance terms are minimized. A large λ implies lower variance (average over more observations) but it implies higher bias (true function is assumed to be constant within the window). In other words, if the window is narrow, $\hat{y}(\vec{x}_0)$ is an average of a small number of y_i close to $\vec{x}_0$, and its variance will be relatively large, almost equal to the variance of the individual y_i. The bias will tend to be small because $E(\hat{y}_i) = y_i$. If the window is wide, the variance of $\hat{y}(\vec{x}_0)$ will be small relative to the variance of any y_i, because of the effect of averaging. The bias will be higher, because we are now using observations $\vec{x}_i$ further from $\vec{x}_0$. As the width goes to 0, the estimates approach a piece-wise linear function that interpolates the training data (high variance), as the width gets infinitely large, the fit approaches the global linear least-square fit to the data (high bias) (Duda et al., 2006).

2.3 Analysis of the Scaling Parameter (λ) in High Dimensions

Until now we have been assuming a circular neighborhood and hence assuming that the function varies equally in all directions. Such a case is called isotropic. This may not be the case in typical machine learning scenarios. We can not assume that the function changes its value uniformly across all directions. Catering for anisotropic cases involves adapting the neighborhood by elongating in some directions and constricting across the others. As will be discussed such an adaptive neighborhood, as a result of using an adaptive metric, is extremely desirable for efficient implementation of most machine learning algorithms in high dimensions.

Figure 2.1 depicts a trivial example of data in 2D and a complicated problem of face category recognition. Let us analyze the two problems from the point of view of the scaling parameter (λ). The relative importance of each feature in figure 2.1(a) is explicit. It is easy to see that class labels are correlated to both the x and y axes. As the class labels change equally in both directions, an isotropic neighborhood parameterized by λ can be used. Such a luxury of eyeballing the correlation of labels with features is not common. For example, consider the data in figure 2.1(b). Since each image is represented by a 256-dimensional feature-vector, we can not easily find the extent of correlation of each feature with the output label. Similarly, we can not assume that the class labels varies equally across all dimensions and, therefore, can not use an isotropic neighborhood. There is thus a need to choose an adaptive neighborhood (tuning a metric with 'metric learning') to scale different features in high dimensions in a systematic way. Incorporation of an anisotropic case will result in the parameterization of λ as a vector or a diagonal matrix, where each element of the diagonal is the scaling parameter for the relevant feature.

The effect of the neighborhood size (λ) on bias and variance of classification has been indicated in section 2.2. In the following, it will be explained that, due to the curse of dimensionality (COD), it is impossible to simultaneously maintain localness (low bias) and a large number of training data in the neighborhood (low variance) in high dimensions. To illustrate, let us consider a simple k-nearest neighbor classification example from Hastie et al. (2001). We assume that all the training data is uniformly distributed in a p-dimensional unit hypercube. Now, let us consider a hypercubical neighborhood of a query point $\vec{x}_0$ and let us suppose it spans a fraction r of the training data. Since this corresponds to a certain fraction r of the unit volume, the expected edge length can be calculated as: $r^{1/p}$. Let us suppose that $p = 20$ and the neighborhood of query point $\vec{x}_0$ spans only 4% of the training data. The edge length will be equal to $(4/100)^{1/20} = 0.85$. The edge lengths if the neighborhood of $\vec{x}_0$ spans 5%, 10% and 20% of data are 0.86, 0.89 and 0.92 respectively. This implies that to capture only 4% of the data, we must cover 85% of the range of each dimension. The neighborhoods are no longer local and are in fact global. This results in extremely high bias. Reducing the neighborhood size by spanning a smaller fraction of the training data will not help, because we need sizable number of training data in the neighborhood to keep the variance low. A second manifestation of the curse of dimensionality is that any feasible number of training data only sparsely populate the input space. Due to this, we will never have enough training data in high dimensions to make nearest neighbor analysis robust.

Intuitively, one can see that to keep the variance low, we need an average over a sizable number of training data. Let us assume that we have a large neighborhood size such that variance is low. What can we do to reduce bias but still keeping the size of neighborhood large? A natural thing to do is to modify the shape of the neighborhood. Rather than specifying a neighborhood that is isotropic, one can elongate the neighborhood in some dimensions and constrict across the others in the hope that the data points in the resulting neighborhood will be close to the query point, hence ensuring localness (low bias) but still maintaining a sizable number of data points in the neighborhood (low variance). We can take advantage of the fact that, at least locally the class probability function does not vary with equal strength or in the same manner in all directions in the measurement space around the query point $\vec{x}_0$. We can choose a metric that gives more weight to important features and less weight to others. The resulting neighborhood will be thereby elongated along de-emphasized directions and constricted along the influential ones. This will result in reduced bias without increasing the variance.

2.4 Metric Learning

Metric learning is a method for learning a metric in a distance measurement framework to improve the performance of a k-nearest neighbor classifier. An example of a distance measurement framework is:

$$d^2(\vec{x}_1, \vec{x}_2) \quad = \quad (\vec{x}_1 - \vec{x}_2)^T A (\vec{x}_1 - \vec{x}_2) \tag{2.15}$$

The Mahalanobis distance between $\vec{x}_1$ and $\vec{x}_2$ is a special case of distance measurement framework and can be written as:

$$d^2(\vec{x}_1, \vec{x}_2) \quad = \quad (\vec{x}_1 - \vec{x}_2)^T \Sigma^{-1} (\vec{x}_1 - \vec{x}_2) \tag{2.16}$$

where the matrix Σ is the covariance matrix of the data (Mahalanobis, 1930, 1936). The idea behind Mahalanobis distance formulation is to normalize the effect of different features when measuring distance between the two points. The matrix A in equation 2.15 can take any form and of course is not restricted to the inverse covariance matrix. We can modify equation 2.15 as:

$$\begin{aligned} d_A^2(\vec{x}_1, \vec{x}_2) &= \|\vec{x}_1 - \vec{x}_2\|_A^2 \\ &= (\vec{x}_1 - \vec{x}_2)^T A (\vec{x}_1 - \vec{x}_2) \\ &= (\vec{x}_1 - \vec{x}_2)^T L^T L (\vec{x}_1 - \vec{x}_2) \quad \text{where} \quad A = L^T L \\ &= (L\vec{x}_1 - L\vec{x}_2)^T I (L\vec{x}_1 - L\vec{x}_2) \quad \text{where} \quad I = \text{identity matrix} \\ &= d^2(L\vec{x}_1 - L\vec{x}_2) \\ &= \|L\vec{x}_1 - L\vec{x}_2\|_2^2 \end{aligned} \tag{2.17}$$

Note, the decomposition of matrix A into $L^T L$ imposes a constraint that matrix A is symmetric positive semidefinite. It can be seen from equation 2.17 that the effect of matrix A in distance measurement framework is nothing but a linear transformation of the data by the matrix L. This is where metric learning algorithms come into play. Metric learning algorithms aim to learn a data-dependent distance metric (matrix A) such that in the transformed space induced by the matrix A, some desirable behavior is expected from the data. Some examples of desirable behavior in the transformed space are: points belonging to the same class tends to be close together, the predicted function changes isotropically, the transformed data lie in the Euclidean space[2], etc. Let us suppose that the matrix A is the inverse covariance matrix. It is desirable in some scenarios to transform the data such that the covariance matrix in the transformed space is the identity matrix. If φ is a matrix whose columns are the eigenvectors of A, and Λ is a diagonal matrix of

[2]The Euclidean space is equipped with a norm ($\|.\|$) which can be used to define a metric (Euclidean distance). The Euclidean distance between $\vec{x}_1$ and $\vec{x}_2$ is:

$$d(\vec{x}_1, \vec{x}_2) = \|\vec{x}_1 - \vec{x}_2\| = \sqrt{(\vec{x}_1 - \vec{x}_2)^T (\vec{x}_1 - \vec{x}_2)}.$$

the corresponding eigenvalues, then we can write matrix L as:

$$L = \varphi \Lambda^{-1/2}$$

The resulting transformation of data induced by the matrix L is typically known as the whitening transformation. The data in the transformed space is uncorrelated. Its covariance matrix is the identity matrix. Another motivation for metric learning is to transform data into a subspace such that the variance of the data is preserved in that subspace. This is again motivated from the dimensionality reduction point of view.

Using equation 2.15 the kernel in equation 2.11 can be modified and the similarity between the two feature-vectors $\vec{x}_1$ and $\vec{x}_2$ can be defined as:

$$k(\vec{x}_1, \vec{x}_2) = \left(\frac{(\vec{x}_1 - \vec{x}_2)^T A (\vec{x}_1 - \vec{x}_2)}{\lambda^2} \right) \tag{2.18}$$

Note the above kernel is similar to the one in equation 2.11 except for the exponential function and the constant in the denominator. Let us suppose that the matrix A is the inverse covariance matrix of the data and λ is the smoothing parameter. If the matrix A take the form:

$$A = \begin{pmatrix} \sigma_1^{-2} & 0 & \cdots & 0 \\ 0 & \sigma_2^{-2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_P^{-2} \end{pmatrix} \tag{2.19}$$

we can write equation 2.18 as:

$$k(\vec{x}_1, \vec{x}_2) = (\vec{x}_1 - \vec{x}_2)^T \begin{pmatrix} \sigma_1^{-2}\lambda^{-2} & 0 & \cdots & 0 \\ 0 & \sigma_2^{-2}\lambda^{-2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_P^{-2}\lambda^{-2} \end{pmatrix} (\vec{x}_1 - \vec{x}_2) \tag{2.20}$$

There are two issues that need to be discussed about equation 2.20. First, in equation 2.20, we have assumed the kernel to be isotropic, that is $\lambda = \lambda_1 = \lambda_2 = \cdots = \lambda_P$. This is not necessarily the case and the neighborhood can be adapted in any directions by choosing suitable λ_p. Multiple scaling parameters λ_p can be treated as a vector and equation 2.20 can be written as:

$$k(\vec{x}_1, \vec{x}_2) = (\vec{x}_1 - \vec{x}_2)^T \begin{pmatrix} \sigma_1^{-2}\lambda_1^{-2} & 0 & \cdots & 0 \\ 0 & \sigma_2^{-2}\lambda_2^{-2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_P^{-2}\lambda_P^{-2} \end{pmatrix} (\vec{x}_1 - \vec{x}_2) \tag{2.21}$$

Secondly, the σ_p and λ_p can be merged into a single value resulting in:

$$\begin{aligned} k(\vec{x}_1,\vec{x}_2) &= (\vec{x}_1-\vec{x}_2)^T \begin{pmatrix} a_1 & 0 & \cdots & 0 \\ 0 & a_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_P \end{pmatrix} (\vec{x}_1-\vec{x}_2) \\ &= (\vec{x}_1-\vec{x}_2)^T A (\vec{x}_1-\vec{x}_2) \end{aligned} \tag{2.22}$$

So far we have assumed that the matrix A is diagonal. This does not have to be the case. We can learn diagonal and off-diagonal terms of the matrix. The metric learning algorithms studied in nearest neighbor framework (Xing et al., 2002; Goldberger et al., 2005; Weinberger et al., 2009; Bar-Hillel et al., 2003) learn both the diagonal and off-diagonal terms of the matrix A. Therefore, we can rewrite equation 2.22 as:

$$\begin{aligned} k(\vec{x}_1,\vec{x}_2) &= (\vec{x}_1-\vec{x}_2)^T \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1P} \\ a_{21} & a_{22} & \cdots & a_{2P} \\ \vdots & \vdots & \ddots & \vdots \\ a_{P1} & a_{P2} & \cdots & a_{PP} \end{pmatrix} (\vec{x}_1-\vec{x}_2) \\ &= (\vec{x}_1-\vec{x}_2)^T A (\vec{x}_1-\vec{x}_2) \end{aligned} \tag{2.23}$$

The learning of matrix A in equations 2.22 and 2.23 is the goal of metric learning algorithms. As discussed, the kernels are used to transform data to a dot-product space $\mathcal{H}$ (using the kernel trick) where the similarity between the two points can be measured by the dot-product. Just as λ in equation 2.11 controls the transformation of the input space $\mathcal{X}$ into $\mathcal{H}$, the matrix A in equation 2.22 controls a similar transformation. Therefore, the performance of any classifier leveraging transformation of data by a kernel will depend critically on the choice of the distance matrix A.

Hastie et al. (2001, section 6.4.1) also mentioned the use of metric learning for tuning the kernel:

> One line of approach is to modify the kernel. The default spherical kernel gives equal weight to each coordinate, and so a natural default strategy is to standardize each variable to unit standard deviation. A more general approach is to use a positive semi-definite matrix A to weigh the different coordinates:
>
> $$k_{\lambda,A}(\vec{x}_0,\vec{x}) = D\left(\frac{(\vec{x}-\vec{x}_0)^T A(\vec{x}-\vec{x}_0)}{\lambda}\right)$$
>
> Entire coordinates or directions can be downgraded or omitted by imposing appropriate restrictions on A. For example, if A is diagonal, then we can increase or decrease the influence of individual predictors X_j by increasing or decreasing A_{jj}. Often the predictors are many and highly correlated, such as those arising from digitized analog signals or images. The covariance function

of the predictors can be used to tailor a metric A that focuses less, say, on high-frequency contrasts.

2.4.1 Why Learn matrix A?

The motivations behind metric learning from the bias reduction point of view were discussed in section 2.3. In the following, some further motivations are discussed.

Features in most machine learning data sets are not commensurate, coming from different sources and having different scales. The influence of each feature on the distance is proportional to the dispersion of its values over the training data. For example, if we change the scale of a feature by measuring it in a different unit, the contribution of this feature to distance measurement will change. This will in turn affect classification performance. In the absence of any other information (for example class labels) the contribution of each feature towards distance measurement can be normalized by dividing it by its variance. Therefore, if σ_i^2 is the variance of the i'th feature, matrix A in equation 2.15 will take the form in equation 2.19 which is actually the covariance matrix of the data.

The relevance of each feature in predicting the class labels may differ. There is a need to give more weight to those features that are more important and less weight to unimportant features. It can be seen that the influence of an individual feature can by controlled through matrix A. For example, if A is modeled as a diagonal matrix, the influence of feature i can be increased or decreased by modifying a_i (equation 2.22). Having a zero on the diagonal results in that feature being completely ignored and hence will result in feature selection. This will remove the feature from the problem and will reduce the dimensionality. Reducing the dimensionality will also help to alleviate the curse that comes with it.

As discussed in section 2.3, metric learning has the effect of neighborhood adaptation. Such an adaptive neighborhood is absolutely essential for reducing bias in higher dimensions and helps in alleviating the effects of the curse of dimensionality. A small value for a_i will result in a neighborhood that is elongated in the direction of the i'th feature. And similarly, a large value of a_i will result in neighborhood being constricted in the direction of the i'th feature. Making the value of a_i very small (note this will result in a_i^{-2} to be very large) will result in extending the neighborhood to the entire training data along the i'th feature. As a result of this, all the training data along i'th feature will be assigned equal weights in predicting the class label. This will result in removing the i'th feature from consideration as the model has become global in i'th feature's direction (Friedman, 1994).

2.4.2 k-Nearest Neighbor and Metric Learning

In the current research on metric learning for nearest neighbor classification, a dichotomy exists in terms of the goals of metric learning algorithms. Most metric learning algorithms are aimed at finding a metric that results in a transformation of data such that, in the transformed space, the intra-class distances are small and inter-class distances are

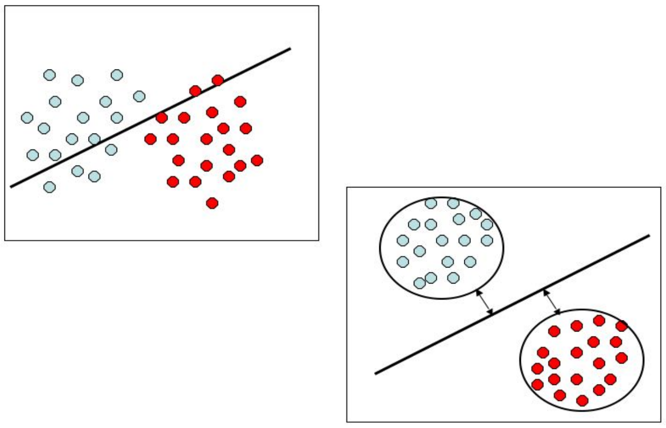


Figure 2.3: (Left) data in the original space, (Right) data in the transformed space. Contrived example demonstrating the impact of data-dependent distance metric on margin.

large (Goldberger et al., 2005; Davis and Dhillon, 2008; Weinberger et al., 2005; Sriperumbudar et al., 2008; Globerson and Roweis, 2005; Xing et al., 2002). This results in maximizing the margin. The margin of a point is defined as the distance between the point and the closest point on the classification boundary. A classifier which results in maximizing the collective margin, i.e. the margin of all the data points, is desirable for generalization purposes. Alternatively, metric learning can also be aimed at minimizing the MSE due to bias-related problems in high dimensions. Such a strategy was introduced by Friedman (1994).

Figure 2.3 depicts a simple contrived example of data belonging to two classes represented by red and blue dots. The classes are linearly separable, and a separating hyperplane is represented by a black line. It should be noted that the margin of the training data can be maximized in two ways. First, we can modify the hyperplane such that the margin is maximized. For example, SVM classifiers maximize the margin by finding an optimal hyperplane to separate the classes. They are designed to minimize empirical risk with a bound on the generalization error. Secondly, we can transform the training data such that the margin is maximized. Rather than showing the margin of each data point in the right figure 2.3, the margin of the whole cluster is shown instead. As can be seen, in the transformed space, the margin is maximized and classes are well separated. As described, margin maximization is the goal of most metric learning algorithms, resulting in a better k-nearest neighbor classification.

2.4.3 Support Vector Machines and Metric Learning

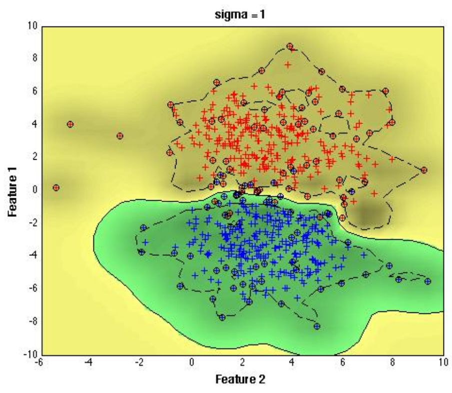
SVM classifiers have been shown to give state of the art performance on a wide range of classification data sets. The SVM formulation finds an optimal hyperplane in high-dimensional space to classify data into two classes. Given a training set $\{(\vec{x}_i, y_i)_{i=1}^N\}$, the decision function is found by solving the following convex optimization problem (Lagrangian Dual):

$$\begin{aligned}
\max_{\alpha} f(\alpha) \quad & = \textstyle\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j k(\vec{x}_i, \vec{x}_j) \\
\text{subject to} \quad & 0 \leq \alpha_i \leq \mathcal{C} \quad \text{and} \quad \textstyle\sum_{i=1}^{n} \alpha_i y_i = 0 \\
\text{where} \quad & k(\vec{x}_i, \vec{x}_j) = \exp\left(- \frac{\|\vec{x}_i - \vec{x}_j\|^2}{2\sigma^2} \right)
\end{aligned} \tag{2.24}$$

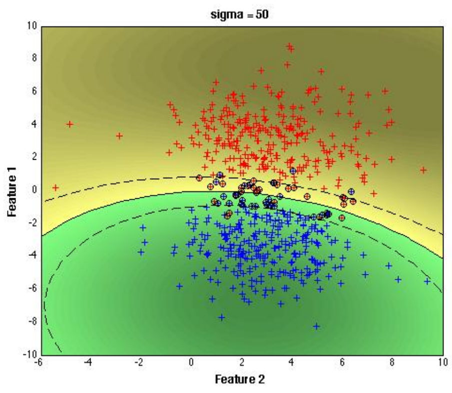
where α are the Lagrange coefficients, $\mathcal{C}$ controls the misclassification penalty and $k(.,.)$ is the kernel function. A detailed analysis of the SVM classification framework is given in appendix D. An SVM classifier is in fact an optimal hyperplane which is computed to classify data in the space induced by the kernel. As discussed, different types of kernel (also, the same kernel with different parameters) embed data in different high-dimensional spaces. The linear decision boundary which the SVM formulation finds thus depends on the choice of kernel and its parameters, in turn affecting the classification performance. The performance of an SVM classifier is thus critically dependent on the choice of kernel and its parameters (Scholkopf and Smola, 2004).

The choice of kernel in the SVM formulation is a regularization choice. The scaling parameters of the kernel controls the bias and variance of the classifier. Scholkopf and Smola (2004, section 7.8) have discussed this issue in detail. Having a large value of σ results in a loosely fitted function, whereas a small value may result in over-fitting. The impact of σ on synthetic data is shown in figure 2.4. It can be seen from the figure that Feature 2 is redundant and that data can be well separated with Feature 1 only. Figure 2.4(a) depicts the case of $\sigma = 1$. The classifier is over-fitted and has memorized the training data. The predicted function is extremely non-smooth and there are a large number of support vectors. Figure 2.4(b) depicts the case of $\sigma = 50$. The classifier is loosely fitted to the training data. There are few support vectors and the classifier is more likely to generalize better than the classifier in figure 2.4(a) where $\sigma = 1$. It should be noted that in most cases, for sheer simplicity, the kernel is assumed to be isotropic, that is $\sigma_1 = \sigma_2 = ... = \sigma_P$ and the σ parameter is tuned through some cross-validation procedure[3]. Typically in computer vision research (Varma and Ray, 2007; Zhang, Marszalek, Lazebnik and Schmid, 2006) to avoid expensive cross-validation, σ is set to be the average value of the squared distance between all the data points as shown

[3]The cross-validation procedure in machine learning is a technique for measuring how well the results of a statistical analysis generalize to unseen data set. Common types of cross-validation are N-fold cross-validation, leave-one-out cross validation, etc. The N-fold cross-validation works by dividing input data into N sub-samples. Of the N sub-samples, only one sample is used for testing, whereas, the other $N-1$ sub-samples are used for training. The process is repeated N times with each of the sub-sample used as testing data once. The N results from the rounds are averaged to produce a single estimate. The 2-fold cross-validation has been used in this thesis unless specified. It works by dividing the data into two mutually exclusive sub-samples, usually known as the training and the validation sets. The performance of the model (trained on the training set) is tested on the validation set. Leave-one-out cross-validation, on the other hand, uses a single data point as the validation set and the remaining data as the training set.



(a)



(b)

Figure 2.4: Demonstration of the impact of varying the σ parameter of a Gaussian kernel in the SVM formulation using synthetic data (continued in figure 2.5). The support vectors are shown in circles.

in the following equation:

$$\sigma^2 = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} d^2(\vec{x}_i, \vec{x}_j) \tag{2.25}$$

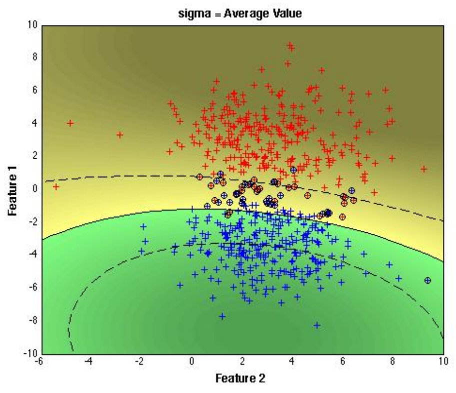
This strategy ensures that numeric range of the kernel is within a bounded interval and results in achieving a trade-off between the bias and variance of a classifier. In figure 2.5(a), this average value of σ is used to find the classification boundaries. As can be seen, it results in much better classification boundaries than those in figure 2.4. Setting σ to be the average value of the distances between the training data seems like a sensible strategy if no prior information is present about the features.

It has been shown that SVM classifiers are not immune to the curse of dimensionality. Hastie et al. (2001, section 12.3.4) have stated this in the following manner:

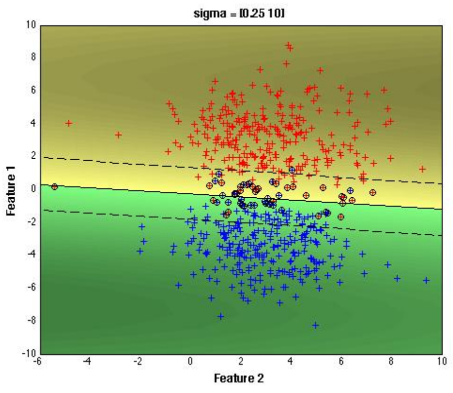
> In the early literature on support vectors, there were claims that the kernel property of the SVM is unique to it and allows one to finesse the curse of dimensionality. Neither of these claims are true, ...

Assuming an isotropic kernel and tuning the value of σ through cross-validation may not be computationally effective. Also using an averaged value of the distances between the entire training data as σ may not be effective as it may not be optimal locally. An alternative is to train an SVM classifier in the local neighborhood. Local metric learning and local SVM classification will be discussed in section 2.4.5. It will be seen that choosing the size of the neighborhood in which an SVM has to be trained is not trivial.

Consider the impact of an anisotropic kernel on the contrived data in figure 2.5(b) where an anisotropic Gaussian kernel with $\sigma = [0.25\ \ 10]'$ is used. The classifier has ignored Feature 2 from consideration and the resulting boundaries are a smooth fitting of the training data.



(a)



(b)

Figure 2.5: Demonstration of the impact of varying the σ parameter of a Gaussian kernel in the SVM formulation using synthetic data (continued from figure 2.4). The support vectors are shown in circles.

The kernel in equation 2.24 can be written as:

$$k_A(\vec{x}_i, \vec{x}_j) = \exp\left((\vec{x}_i - \vec{x}_j)^T A(\vec{x}_i - \vec{x}_j)\right) \tag{2.26}$$

Again, we seek a matrix A that will embed the input data in a high-dimensional feature space such that the linear decision boundary found by the SVM classifier separates data well. In the SVM framework, the matrix A is usually assumed to be diagonal and cross-validation is the most popular and effective method for tuning its elements. A few methods have been proposed for learning the elements of matrix A for improving SVM classification performance (Chapelle et al., 2002; Keerthi, 2001; Amari and Wu, July 1999). These methods will be discussed in detail in section 7.1.

2.4.4 Gaussian Processes and Metric Learning

The Gaussian Processes (GP) is a non-linear nonparametric technique that has proven to be very effective for a wide range of classification and regression tasks. We can define GP as a collection of random variables, any finite number of which have a joint Gaussian distribution (Rasmussen and Williams, 2006). The GP is completely specified by its mean and covariance function. We can write GP as:

$$f(\vec{x}) \sim \mathrm{GP}(m(\vec{x}), k(\vec{x}, \vec{x}')) \tag{2.27}$$

where $m(\vec{x})$ and $k(\vec{x}, \vec{x}')$ are the mean and covariance functions respectively. Usually, for simplicity, $m(\vec{x})$ is taken to be zero. The GP can thus be specified completely in terms of its covariance function. The problem of learning with GP is exactly the problem of finding a suitable covariance function (also known as the kernel). A detailed introduction to GP is given in appendix E.

One of the most widely applied covariance functions in GP setting is the isotropic Squared Exponential (SE) covariance function:

$$k(\vec{x_i}, \vec{x_j}) = \sigma_f^2 \ \exp\left(-\frac{\|\vec{x_i} - \vec{x_j}\|^2}{2\lambda^2}\right) + \sigma_n^2 \Delta_{ij} \tag{2.28}$$

σ_f^2 and σ_n^2 denotes the signal and noise variance in the data and λ is a parameter specifying the characteristic length scale[4]. Informally, the characteristic length scale is the distance one can move in the input space before the function value changes significantly. Of the three parameters $\{\sigma_f^2, \sigma_n^2, \lambda\}$, λ is the most important, as the classification performance of GP classifier depends a great deal on the characteristic length scale parameters. During the course of this work, σ_f^2 and σ_n^2 will not be taken into account. Therefore, the kernel we are interested in has the following form:

$$k(\vec{x_i}, \vec{x_j}) = \exp\left(-\frac{\|\vec{x_i} - \vec{x_j}\|^2}{2\lambda^2}\right) \tag{2.29}$$

The kernel in equation 2.29 can be generalized as in equation 2.26. This suggests that, as for the SVM, the problem of learning with GP is actually finding the right specification of matrix A in equation 2.26. The representation of length scale parameters as the elements of a matrix has been mentioned by Rasmussen and Williams (2006, sec. 5.1) as:

> Covariance functions such as the squared exponential can be parameterized in terms of hyperparameters. For example
>
> $$k(\vec{x}_p, \vec{x}_q) \quad = \quad \sigma_f^2 \exp(-\frac{1}{2}(\vec{x}_p - \vec{x}_q)^T M(\vec{x}_p - \vec{x}_q) + \sigma_n^2 \delta_{pq} \tag{2.30}$$
>
> where $\theta = (\{M\}, \sigma_f^2, \sigma_n^2)^T$ is a vector containing all the hyperparameters, and $\{M\}$ denotes the parameters in the symmetric matrix M. Possible choices for the matrix M include
>
> $$M_1 = l^{-2}I, \quad M_2 = \text{diag}(l)^{-2}, \quad M_3 = \Lambda\Lambda^T + \text{diag}(l)^{-2} \tag{2.31}$$
>
> where l is a vector of positive values, and Λ is a $D \times k$ matrix, $k < D$. The properties of functions with these covariance functions depend on the values of the hyperparameters. For many covariance functions it is easy to interpret the meaning of the hyperparameters, which is of great importance when trying to understand your data. For the squared exponential covariance function eq. (2.30) with distance measure M_2 from eq. (2.31), the $l_1, \ldots, l_D$ hyperparameters play the role of characteristic length-scales; loosely speaking, how far do you need to move (along a particular axis) in input space for the function values to become uncorrelated. Such a covariance function implements automatic relevance determination (ARD) [Neal, 1996], since the inverse of the length-scale determines how relevant an input is: if the length-scale has a

[4]The λ in equation 2.28 is equivalent to σ in equation 2.24. It is typically denoted as λ in GP settings to avoid confusion with symbols denoting the signal and noise variance.

very large value, the covariance will become almost independent of that input, effectively removing it from the inference.

Neal (1996), Zhou and Suter (2008), Snelson et al. (2003), Schmidt and O'Hagan (2003) and others have also proposed methods for learning the elements of matrix A in GP setting. These methods will be discussed in detail in section 9.1.

2.4.5 Local and Global Methods of Metric Learning

The metric learning methods discussed so far are global[5]. That is, a data-dependent distance metric is optimized using all training data and the same distance metric is used for all locations of the query point in the input space. There is an alternative to learning a globally optimized distance metric. One can learn a data-dependent distance metric that is optimized only in the (small) neighborhood of a query point. Such metric learning approaches are called local adaptive metric learning. Both global and local methods of metric learning share the same motivations. Intuitively, one can think of local methods of metric learning in terms of the feature relevance at a particular point in the input space. It may be the case that a feature is more relevant at one location but is less relevant at another location in the input space. For example, consider a simple gender classification example using features such as height, weight, complexion, etc. If height is measured in feet, one can notice that the height feature is more important in predicting gender when it is above six (a woman is less likely to have height above six feet). This suggests the local relevance of the feature height in the input space.

Local adaptive metric learning methods have often been studied in the k-nearest neighbor framework. A nearest neighbor classifier assumes that the predicted function exhibits a regular behavior at least in the neighborhood of the query point and hence the function can be predicted by simply averaging the class labels in the neighborhood. In other words, class conditional probabilities are assumed to be smooth in the neighborhood. This assumption of locally smooth class conditional probabilities is invalid at least near class boundaries, where the class conditional probabilities can vary differently across different dimensions. Learning a metric in the neighborhood of a query point can not only result in smoother class-conditional probabilities in the neighborhood, but also in reduction of the bias introduced due to the curse of dimensionality. Consider the example in figure 2.6. The class-conditional probabilities vary only in the horizontal direction. There is a need to stretch local neighborhood near the class boundary in the vertical direction. This strategy reduces the bias of our estimate and leaves the variance the same, as discussed in section 2.3.

Any global metric learning algorithm can be made local by adjusting the size of the neighborhood in which the metric is learned. Though local adaptive metric learning methods are more computationally expensive, they are a natural approach to adapting neighborhoods so that the class-conditional probabilities are smooth. As, unlike global

[5]Global methods in machine learning encompass the use of the entire range of training data. In terms of neighborhood size, one can imagine the size of the neighborhood so large that it covers the entire training data. On the other hand, local methods encompass the use of the training data in a small neighborhood.

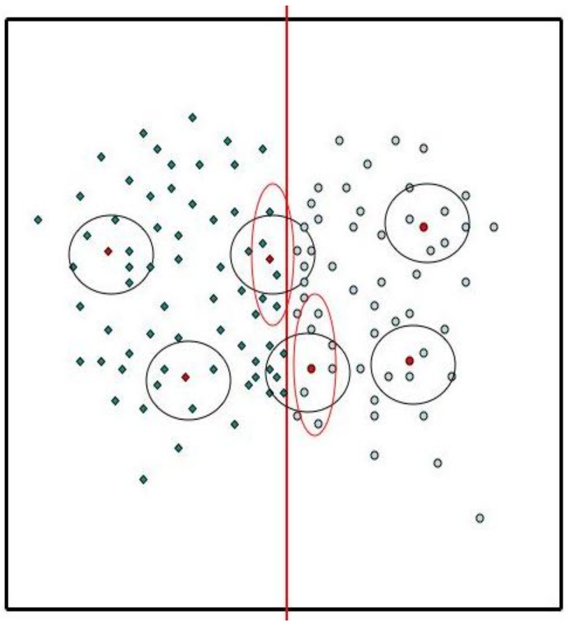


Figure 2.6: Illustration of adaptive metric learning on 2-dimensional data. The data belongs to two classes which are well separated by a linear classifier (shown as bold red line). Round circles (black) depicts the isotropic neighborhoods (calculated with a Euclidean metric). As can be seen, this assumption of isotropy is not valid near class boundaries and a modified neighborhood stretched in vertical direction is more effective. This adaptive neighborhood is depicted as ellipses (red).

methods, local metric learning fits naturally in the lazy learning framework[6] of nearest neighbor classifiers, i.e. rather than learning a global distance metric beforehand, once a query data point is encountered, its neighbors are determined and a local metric is learned. The need to determine the size of the neighborhood in which the metric is to be learned is one of the biggest disadvantage of local methods.

The scope of local metric learning is not limited to k-nearest neighbor classification. As hinted in section 2.4.3, an SVM classifier can be trained in the local neighborhood of the query point. Since the tuning of SVM kernel parameters is actually a form of metric learning, learning a kernel in the neighborhood of a query point will have almost the same effect as training an SVM classifier in the modified neighborhood of a query point. This is the exact equivalent of the local adaptive metric learning discussed so far, except that rather than training a nearest neighbor classifier in the neighborhood, an SVM classifier is trained and used to predict the label of the query point. This equivalence can be extremely useful. As learning the kernel parameters for a local SVM classifier actually results in a local adaptive SVM classifier, it will be shown in chapter 8 that learning a data-dependent distance metric and training a local SVM classifier without learning any kernel parameters also results in a local adaptive SVM classifier and somewhat alleviates the need to tune kernel through cross-validation procedures.

[6]Lazy learning denotes those techniques in which learning is delayed until a query data is encountered. 'Eager learning' on the other hand denotes techniques which learn from the training data and apply the learned model to the query data.

2.4.6 Metric Learning with Feature-sets

So far we have assumed that the data is represented as feature-vectors and hence requires the tuning of the scaling parameters. It may be the case that the data is represented as feature-sets (appendix A), that is each data point is represented as a set of feature-vectors. In the case of learning a data-dependent distance metric where data is represented in the form of feature-sets, not only will a metric learning algorithm have to learn a metric for each feature-vector in the feature-sets, but also it will need to find the appropriate weights to combine the different feature-vectors.

Processing feature-sets is actually an example of information fusion, as we need to decide how different feature-vectors in the feature-sets should be combined. There are several possibilities when it comes to learning from feature-sets. Either we can concatenate all the feature-vectors in the feature-set and process them (learn a distance metric, train a classifier, etc.) as a single feature-vector, or we can process them separately and combine the output of the processing at a later stage. There are two further possibilities. We can combine the different feature-vectors in the original input spaces. That is, after a metric is learned for each feature-vector, we concatenate the different feature-vectors with an appropriate weighting scheme (distance fusion). The second possibility is that, after a metric has been learned for each feature-vector, a separate classifier is trained for each feature-vector and the outputs of the classifiers are combined with an appropriate weighting scheme (classifier fusion).

2.5 Discussion

There are at least two issues that have become evident from the previous discussion of metric learning and machine learning algorithms. First, although conventionally metric learning is usually applied in nearest neighbor settings only, the performance of any learning algorithm that depends on a kernel to measure the similarity between two data points can be improved through metric learning. Therefore, the realm of metric learning extends beyond nearest neighbor classification to, for example SVM and GP, etc. Although various techniques have been proposed that learn a data-dependent distance metric for kernel-based methods, these techniques are generally restricted to learning the diagonal terms of the distance matrix A only (Chapelle et al., 2002; Williams and Rasmussen, 1996).

Secondly, metric learning, by controlling the size of the neighborhood, actually implements feature relevance and scaling. Also, it can result in feature selection and control the bias and variance of the classification. Similarly, various techniques such as feature selection, feature relevance, feature scaling, and kernel tuning can be defined in terms of metric learning and can be viewed as learning the elements of the matrix A in equation 2.22. Such a line of thinking has also been mentioned by Chapelle et al. (2002, page 133):

> Indeed, when no a priori knowledge is available about the meaning of each of the attributes, the only choice is to use spherical kernels (i.e. give the same

> weight to each attribute). But one may expect that there is a better choice for the shape of the kernel since may real-world database contain attributes of very different natures. There may thus exist more appropriate scaling factors that give the right weight to the right feature. For example, we will see how to use radial basis function kernels (RBF) with as many different scaling factors as input dimensions:
>
> $$k(\vec{x}, \vec{z}) \quad = \quad \exp\left(-\sum_i \frac{(x_i - z_i)^2}{2\sigma_i^2}\right)$$
>
> The usual approach is to consider $\sigma = \sigma_1 = \cdots = \sigma_n$ and to try to pick the best value for σ. However, using the proposed method, we can choose automatically good values for the scaling factors σ_i. Indeed, these factors are precisely the parameters of the kernel.
>
> Moreover, we will demonstrate that the problem of *feature selection* can be addressed with the same framework since it corresponds to finding those attributes which can be rescaled with a zero factor without harming the generalization.

They go on to say:

> We thus see that tuning kernel parameters is something extremely useful and a procedure that allows to do this would be a versatile tool for various tasks such as finding the right shape of the kernel, feature selection, finding the right tradeoff between error and margin, etc. All this gives a rationale for developing such techniques.

Therefore, in the light of the above discussion we propose the following:

1. There is a need to unify the different terms used for metric learning — for example feature selection, feature weighting, feature scaling, and kernel tuning — under a common framework for systematic study of these techniques.

2. There is a need to investigate and propose novel global and local metric learning methods aimed at reducing bias in high dimensions for improving classification performance.

3. There is a need to investigate metric learning in kernel-based frameworks. More importantly, there is a need to investigate the utility of current metric learning methods, proposed in the nearest neighbor framework, when applied in kernel-based classification frameworks such as SVM and GP. Also, there is a need to study the effects of tuning a kernel by learning a full distance matrix (diagonal and off-diagonal terms of matrix A in equation 2.26) on such techniques.

4. There is a need to investigate local metric learning methods in kernel-based frameworks. For example, a kernel is learned that is optimized in the neighborhood of the query point and a local SVM classifier is then trained using the learned kernel.

5. Learning with feature-sets actually deals with information fusion. Distance fusion and classifier fusion methods represent a high-level form of metric learning and, therefore, there is a need to investigate such methods in the context of metric learning.

Addressing the first proposal

The work in this thesis deals with the unification of the different terms used for various forms of metric learning. Different metric learning methods are categorized under a common distance measurement framework into naive, semi-naive, complete, and high-level metric learning. This novel categorization is introduced in chapter 3 and explained in chapters 4, 5 and 10. This categorization is desirable for the following reasons:

- Categorizing these approaches as forms of metric learning demonstrates the prevalence of metric learning algorithms. For example, it is well known that most learning algorithm rely on scale estimation, feature selection, feature scaling, etc. Studying these approaches as metric learning algorithms reveals that most learning algorithms rely on some sort of metric learning.
- Studying these approaches under the framework of metric learning can lead not only to a better understanding of these approaches but also of the learning algorithms that rely on metric learning.
- As metric learning methods are aimed at learning the properties of the input space, an analysis of these methods in the light of a common framework encourages their use across different learning algorithms, because every learning algorithm has to deal with the inherent properties of the space in which the data lies.

Addressing the second proposal

To address the second proposal, novel metric learning algorithms in local and global setting, have been proposed in chapters 4, 5 and 6. The performance of the proposed algorithms is compared with state of the art methods. They are shown to perform well on a variety of data sets.

Addressing the third proposal

The connection between kernel and metric learning was discussed in section 2.4. It was also discussed that most common methods of metric learning have been studied in the k-nearest neighbor framework only. Since, kernel and metric learning are very closely related, it is natural to use existing methods of metric learning to tune the kernel parameters for SVM and GP classifiers. There are at least two reasons why such a strategy should work:

- Even though the formulations of nearest neighbor methods and kernel-based learning methods are different, both frameworks share their dependence on a similarity measure. For example both use the kernel in equation 2.22 and their performance depends how well they can optimize matrix A in equation 2.22.

- Feature scaling methods proposed in the SVM and GP framework have been successfully applied in k-nearest neighbor settings for feature selection and feature weighting (Chapelle et al., 2002; Guyon et al., 2004, chapter 11). If a metric tuned in SVM and GP formulation is effective in the nearest neighbor framework, one can argue that a metric tuned in nearest neighbor formulation might also be effective in the SVM and GP settings.

An analysis of kernel tuning for SVM and GP in terms of metric learning in section 2.4.3 and 2.4.4 has revealed that most methods only estimate a diagonal distance matrix (Chapelle et al., 2002; Williams and Rasmussen, 1996). The effect of learning a full distance matrix has not been investigated, perhaps for the following reasons:

- In the SVM and GP frameworks, the Gram matrix (produced from the kernel) has to be positive semi-definite (PSD). It may be the case that a global linear transformation learned via a metric learning approach could violate this condition.

- A metric optimized for nearest neighbor classification may not be optimized for SVM and GP classification. Researchers are more likely to attempt to tune scaling parameters along with non-diagonal terms of the matrix A directly in either SVM or GP setting. Efforts have been made in this direction but are limited to the learning of the diagonal terms of matrix A, for example automatic relevance determination (ARD) (Williams and Rasmussen, 1996) in GP framework and Chapelle et al.'s (2002) method of scale estimation in SVM framework.

The first issue can be alleviated by learning a metric that results in a positive semi-definite Gram matrix. As to the second, there does not exist a proof that a metric optimized in a nearest neighbor framework will be effective in SVM or GP settings. The application of a learned metric can be considered as a pre-processing step[7] which aims to improve GP classification performance leading to a better classification performance. Several pre-processing techniques for GP classifiers have been proposed and shown to improve GP classification performance (Snelson et al., 2003; Zhou and Suter, 2008). Schmidt and O'Hagan (2003) provided a proof that the off-diagonal terms of transformations to isotropic spaces can not be learned in some GP formulations. If we can not learn a full metric characterizing the linear transformation of data in GP settings, and following the insight that pre-processing of data can improve GP performance, one can learn a distance metric in nearest neighbor framework in hope that the transformed data will be better modeled by GP.

In chapters 7 and 9, the effects of tuning a kernel by learning a full distance matrix on SVM and GP classification performance are studied. Algorithms for learning the kernel

[7]The motivation behind pre-processing (transformation) is to rectify the effects of the often unreasonable assumption of the GP that the raw data will have Gaussian noise and will be well modeled by GP. Significant performance gains can be achieved by transforming data such that it is well modeled by GP. One particular example of such a transformation is taking the log of the data (for example, when quantities vary over many order of magnitude, it make little sense to model these quantities directly assuming isotropic Gaussian noise. A log transformation is required). Many other transformations can be applied to the data to bring it into a form well modeled by GP.

parameters for improving SVM and GP classification performance using metric learning methods are also proposed.

Addressing the fourth proposal

In chapter 8, building on the insights from local methods and motivated by Zhang, Berg, Maire and Malik (2006), local metric learning is investigated in kernel-based settings and a local SVM classification scheme is proposed. That is, rather than training a single global SVM classifier, an SVM classifier is trained in the neighborhood of each query point. The motivations behind such a formulation are discussed in detail. Also, a local adaptive SVM is proposed for achieving the computational efficiency of a k-nearest neighbor classifier and the classification efficiency of an SVM classifier.

Addressing the fifth proposal

Chapter 10 deals with learning from feature-sets. Distance and classifier fusion approaches are studied in the context of metric learning using scene categorization and object detection examples.

Chapter 3

Novel Categorization of Metric Learning Methods

In this chapter a novel categorization scheme for metric learning methods will be introduced. The use of these categories will be demonstrated on a simple classification task.

There are different ways in which we can categorize metric learning algorithms. On a broader level, metric learning can be categorized into supervised and unsupervised learning. In unsupervised metric learning, class labels are not used to learn the distance matrix A (equation 2.15). This form of metric learning algorithm has been extensively studied and is often motivated from a dimensionality reduction and scale normalization point of view. For example, Principal Component Analysis (PCA) is a well known unsupervised metric learning technique aimed at dimensionality reduction while preserving maximal variance of the data (Pearson, 1901; Jolliffe, 2002). Other notable unsupervised metric learning algorithms include Multidimensional Scaling (MDS) and Local Linear Embedding (LLE) (Cox and Cox, 2001; Roweis and Saul, 2000; Saul and Roweis, 2000).

On the other hand, supervised metric learning tunes a metric using side information such as the class labels. A motivation behind this form of metric learning is to increase the classification performance (Xing et al., 2002; Goldberger et al., 2005; Weinberger et al., 2009). The novel metric learning algorithms proposed in chapters 4 and 5 are supervised.

Since metric learning deals with the learning of matrix A in equation 2.15, we can categorize metric learning methods based on the form of matrix A, for example diagonal matrix, full matrix, block partition matrix, etc. Such a categorization is novel and is explained in the following section.

3.1 Naive/Semi-naive Metric Learning

Feature selection and feature relevance are well studied and effective techniques for ignoring or down-weighting irrelevant or redundant features and making relevant features more explicit. Feature selection can be viewed as a form of metric learning. For example learning a diagonal matrix in equation 2.15 with some diagonal elements as zero results in those features being ignored, i.e. feature selection. The case of learning a diagonal matrix in equation 2.15 is categorized as either naive metric learning or semi-naive metric learning.

The naive metric learning case arises when we learn the diagonal terms of matrix A in equation 2.15 and assume that features are independent from each other. The relevance of each feature is estimated separately. The problem with naive metric learning is that features that are irrelevant when analyzed separately may become relevant when analyzed together with other features. Therefore, there is a need to estimate the relevance of a feature in combination with the other features. This is the motivation behind semi-naive metric learning. Semi-naive metric learning is a modification of naive metric learning where a diagonal matrix is learned by learning the relevance of each feature in combination with the other features.

Naive and semi-naive metric learning are actually forms of feature selection. Feature selection and weighting has been extensively studied in machine learning (Guyon et al., 2004), but usually it is not explicitly specified that feature selection is in fact learning of a distance metric such that the measurement of distances across certain features is ignored. It should be noted that any feature selection technique can be viewed as a naive or semi-naive metric learning method and vice-versa. Though the two techniques (feature selection and metric learning) have been studied separately, the motivations behind them are exactly the same.

3.2 Complete Metric Learning

As discussed in section 2.4, the matrix A learned in equation 2.15 does not have to be diagonal and a full distance matrix can be learned. Complete metric learning deals with the learning of both the diagonal and the off-diagonal terms of the matrix A in the form shown in equation 2.23. It should be noted that this form of matrix A is more reasonable, in that we can not simply assume that the function changes only along the directions of axes only. Typical metric learning algorithms studied in the k-nearest neighbor classification framework generally estimate the full distance matrix A and are in fact complete metric learning methods (Weinberger et al., 2009; Goldberger et al., 2005; Bar-Hillel et al., 2003).

3.3 High-level Metric Learning

High-level metric learning deals with learning a data-dependent distance metric in cases where data is represented as feature-sets. As discussed in chapter 2 and appendix A, data is represented in the form of feature-sets in those problems when there is a natural partitioning among features, for example object recognition. High-level metric learning is concerned with how to deal with feature-vectors in the feature-sets. Let us suppose that m and n are two types of feature-vector used for object recognition problem (say m representing the 'shape' information and n representing the 'color' information). Let $\vec{x}_m$ and $\vec{x}_n$ be the feature-vectors representing the m and n types of information about the data point $\vec{x}$ respectively. We can denote the feature-set representing the data point $\vec{x}$ as:

$$\mathcal{FS}(\vec{x}) = \{\vec{x}_m, \vec{x}_n\} \tag{3.1}$$

The two high-level metric learning schemes will be discussed in detail in chapter 10. In the following, a brief introduction of the two schemes is given. In the first scheme, we can concatenate the two feature-vectors into one, learn a naive/semi-naive/complete metric with the resulting feature-vector, and train a single classifier.

In the second scheme, we can treat $\vec{x}_m$ and $\vec{x}_n$ separately. A separate naive/semi-naive, complete distance metric is learned for each type of feature-vector. Using the learned distance matrices we can train a separate classifier for each type of feature-vector and combine the outputs of the classifiers using some weighting scheme to reach consensus about the label of the object (classifier fusion). Alternatively, we can combine the distances computed using the learned distance matrices for each type of feature-vector (distance fusion) and learn a single classifier. Note that this will have the same effect as measuring distances using a block-diagonal matrix A in equation 2.15 such that:

$$A = \begin{pmatrix} A_m & 0 \\ 0 & A_n \end{pmatrix} \tag{3.2}$$

where A_m and A_n are the distance matrices learned for feature-vector of type m and n.

3.4 Example Illustrating Forms of Metric Learning

The goal of this section is to demonstrate with a simple example how different forms of metric learning can be used in a learning scenario. Let us consider the problem of object recognition where the objects are different categories of flowers. The database considered is the Oxford Flowers database which consists of 17 categories of flowers (Nilsback and Zisserman, 2007b, 2006, 2007a). Some example images of flowers are shown in figure 3.1. From the recognition point of view, it is a difficult database as different flowers have huge pose, scale, and light variation. Color seems to be a discriminative feature, but many flower categories share the same color. Not only are there huge inter-class differences, there are huge intra-class variations among the flowers that make this categorization problem extremely challenging. Note the background of each flower. Different flower categories have very different but peculiar backgrounds. Therefore, background information can be used for classification purposes.

Designing a classification algorithm for such a problem generally has two phases. The first deals with the identification of feature-vectors (or feature-set). As can be seen from figure 3.1, the learning algorithm can not rely on one type of cue. For example some categories can be discriminated by their shape, some categories have a peculiar color, others can effectively be categorized by their texture, etc. Therefore, each image should be represented as a feature-set (each feature-set consists of different feature-vectors representing cues like shape, color, texture, etc.). The second phase deals with the learning of a classifier from the feature-vectors (or feature-set). In the following, the different types of feature-vectors used in this flower categorization example are explained.

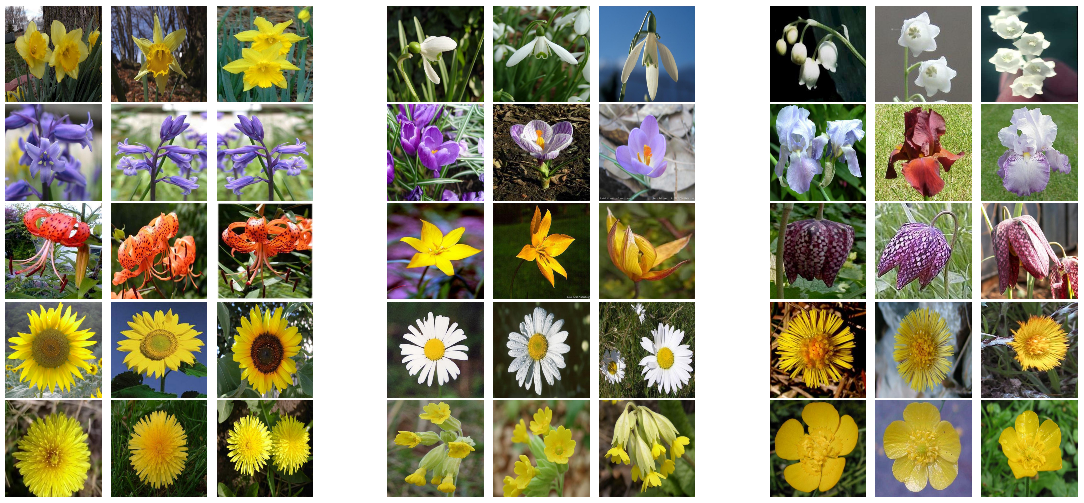


Figure 3.1: Example images (three images per category) from the Oxford Flowers database.

Features

Four different types of feature-vector based on shape, color, texture and context (gist) of the flowers have been used. The feature-set representing image $\vec{x}$ can be written as:

$$\mathcal{FS}(\vec{x}) = \{\vec{x}_{\text{shape}}, \vec{x}_{\text{color}}, \vec{x}_{\text{texture}}, \vec{x}_{\text{gist}}\} \tag{3.3}$$

In the following, each feature-vector in $\mathcal{FS}$ is explained:

- **Shape:** As can be seen from figure 3.1, the shape of the flowers (shapes of petals and their configurations) is a discriminatory trait. To capture the shape of a flower, SIFT features extracted on a regular grid of size 20×20 with a spacing of ten pixels were used (Lowe, 1999). The features extracted were vector quantized with a code-book of size 200 obtained via k-means clustering The resulting histogram was then normalized to give a 200-dimensional feature-vector to represent the shape of the object. Note that this is an example of a dense feature. An alternative is to calculate the SIFT features at some points of interest on the image. It has been shown that dense features perform better than sparse features on most categorization tasks (Dance et al., 2004; Willamowski et al., 2004). For a comparative analysis of dense and sparse features for object categorization, refer to appendix C.

- **Color:** As can be seen from figure 3.1, color can be a powerful cue for flower categorization. To represent color, the Hue, Saturation, and Value (HSV) value of each pixel was used. Therefore, each pixel was represented as a 3-dimensional vector. These 3-dimensional vectors representing pixels were vector quantized with a code-book of size 200 obtained via k-means clustering. The resulting histogram was normalized to represent the color of the object.

- **Texture:** Texture is represented by convolving images with an MR8 filter bank, introduced by Varma and Zisserman (2005). The filter bank contains filters at multiple orientations. Rotation invariance was obtained by choosing the maximum response over orientations. The texture of an image was represented by a normalized histogram obtained by vector quantizing the response of filter bank using a code-book size of 800. This gave an 800-dimensional feature-vector representing the texture of an object.

- **Context (gist):** Since most flowers exists in their natural habitats, there seems to be some correlation between flowers and their backgrounds. To exploit this information, the gist features based on Gabor filters as introduced by Oliva and Torralba (2001) were used. This gave a 512-dimensional feature-vector representing the gist of an object.

For each category, 40 images per category were used for training and 20 images per category were used for testing. The results shown in tables 3.1 and 3.2 are the means of ten runs on different sets of training and testing data.

Experiment 1

In this experiment, the feature-vectors in the feature-set were processed separately. The classification performance in terms of the correctness rate of each of the following metric learning methods is shown in figure 3.2.

- **Level 1 - Euclidean Distance:** A 1-nearest neighbor classifier was trained for each feature-vector in the feature-set.

- **Level 2 - Unsupervised Metric Learning:** As each feature in every feature-vector is different, a natural preprocessing was to re-scale all features to have a zero mean and unit variance so that they impact the distance measurement equally[1]. Also, since all feature-vectors in $\mathcal{FS}$ are of high dimension, PCA was used to reduce the dimensionality. It should be noted that the goal of this level of metric learning is not to improve the classification efficiency but only to reduce dimensionality. The original shape, color, texture and context feature-vectors were reduced to 20, 20, 160 and 100 dimensions respectively. A 1-nearest neighbor classifier was trained after re-scaling and dimensionality reduction for each feature-vector.

- **Level 3 - Complete metric learning:** Feature-vectors were transformed by learning a data-dependent distance metric (the MEGM complete metric learning algorithm that will be introduced in chapter 4 was used) and a 1-nearest neighbor classifier was applied in the transformed space.

Note that the Level 2 metric (PCA) did not result in any significant performance gain whereas Level 3 metric (MEGM) lead to some improvement in the performance of Level 1 metric (Euclidean). Since the feature-vectors were not combined, high-level metric learning was not applicable.

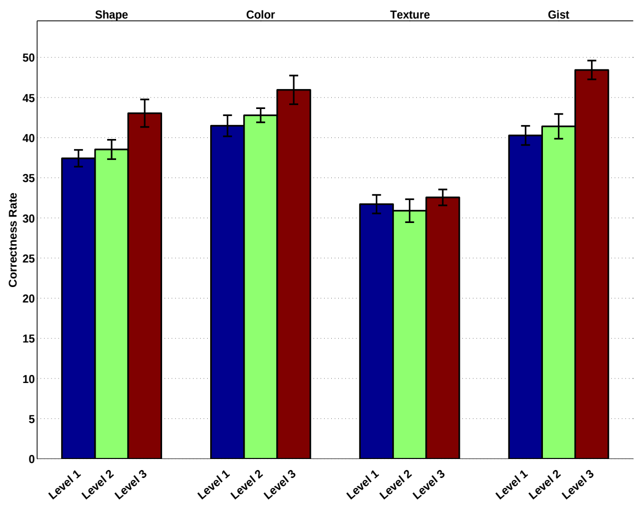


Figure 3.2: Experiment 1 results (correctness rate) for training a separate 1-nearest neighbor classifier for each feature-vector in $\mathcal{FS}$. The means and standard deviations of the results are shown. Bar group 1 = Shape feature-vector, group 2 = Color feature-vector, group 3 = Texture feature-vector, group 4 = Gist feature-vector.

Experiment 2

It can be seen from figure 3.2, the color and the gist features performed best, with a classification performance of around 48% but these feature-vectors alone do not have the discriminative power to classify all the objects successfully. Therefore, there is a need to combine different feature-vectors. For example, the feature-vectors can be combined by concatenating them together. When the feature-vectors are concatenated, we can run the metric learning methods from experiment 1 on the concatenated feature-vectors.

In the second experiment, the feature-vectors in $\mathcal{FS}$ were combined and processed together. The classification performance in terms of the correctness rate of each of the following metric learning methods is shown in table 3.1.

- **Level 1 - Euclidean Distance:** The feature-vectors were concatenated and a 1-nearest neighbor classifier was trained.
- **Level 2 - Unsupervised Metric Learning:** Same as in experiment 1 except the feature-vectors were concatenated before the unsupervised metric learning algorithm was applied.
- **Level 3 - Complete metric learning:** Same as in experiment 1 except the feature-vectors were concatenated before the complete metric learning algorithm was applied.
- **Level 4 - High-level Metric Learning:** Feature-vectors which were already transformed via unsupervised metric learning (Level 2) and complete metric learning (Level 3) were combined with a high-level metric learning algorithm. That is, the

[1]This step is not very critical for histogram-based features as histograms are already normalized. But non-histogram based feature-vectors must be normalized.

	Level 1	Level 2	Level 3	Level 4
Shape + Color	47.89 ± 0.08	48.95 ± 0.50	54.11 ± 0.02	56.73 ± 0.21
Shape + Texture	39.85 ± 1.22	40.08 ± 0.20	44.82 ± 0.45	46.23 ± 0.19
Shape + Gist	45.02 ± 0.58	46.27 ± 0.11	54.94 ± 0.25	56.61 ± 0.39
Color + Texture	46.67 ± 0.11	47.79 ± 0.67	49.32 ± 0.65	52.23 ± 0.12
Color + Gist	51.58 ± 0.31	52.36 ± 0.05	60.35 ± 0.16	61.69 ± 0.20
Texture + Gist	45.48 ± 0.80	45.45 ± 0.25	47.47 ± 0.95	49.82 ± 0.79
Shape + Color + Texture	48.95 ± 0.50	54.51 ± 0.26	56.56 ± 0.19	58.69 ± 0.43
Shape + Color + Gist	53.61 ± 0.27	53.48 ± 0.45	$\mathbf{62.33 \pm 0.33}$	$\mathbf{64.2 \pm 0.10}$
Color + Texture + Gist	50.98 ± 0.24	52.73 ± 0.45	57.86 ± 0.15	60.11 ± 0.09
Shape + Texture + Gist	46.52 ± 0.59	47.86 ± 0.50	53.42 ± 0.28	56.23 ± 0.32
Shape + Color + Texture + Gist	$\mathbf{54.22 \pm 0.13}$	$\mathbf{54.94 \pm 0.10}$	61.01 ± 0.19	–

Table 3.1: Experiment 2 results (correctness rate) for a 1-nearest-neighbor classifier on Oxford flower database obtained using high-level metric learning for $\mathcal{FS}$. The results were obtained with the combination of feature-vectors listed in the first column of the table. The best result for each level of metric learning is highlighted.

distances measured across different feature-vectors were combined with distance fusion version of high-level metric learning scheme 2 (HML2-DF) that will be discussed in section 10.1.2.

The best result of 64.2% was obtained by combining the shape, color and the gist feature-vectors. Also, noteworthy is the performance when only color and gist feature-vectors were combined with a classification performance of 61.69% suggesting that the color and the gist feature-vectors can be extremely effective when considered together. There is not a huge difference in the classification performance of the 1-nearest-neighbor classifier at Level 1 and Level 2. Since the goal of Level 2 is only to reduce dimensionality for computational efficiency, the Level 2 results are encouraging as they are similar to Level 1 results but are obtained with data with vastly reduced dimensionality. The optimal combination weights of each feature-vector for distance fusion version of high-level metric learning scheme 2 (Level 4) are given in table 3.2. To obtain the optimal ω values, ten images per category from the training images were used as a validation set. The weight of each feature-vector was selected from the set $\{0.2, 0.4, 0.6, 0.8, 1\}$. The set of weights giving the best results on the validation set was used to train a 1-nearest-neighbor classifier to obtain the Level 4 results in table 3.1.

Comparison of Results with State of the Art

Nilsback and Zisserman (2006) have reported results on Oxford flowers databases with a k-nearest neighbor formulation. That work is similar to this work, with some subtle differences. Their goal is to form an optimize code-book of visual words for flower categorization, whereas the goal here is to demonstrate the working of different forms of metric learning. Unlike Nilsback and Zisserman (2006), no variable affecting the classification performance has been optimized (except for weights ω in high-level metric learning scenario). Nilsback and Zisserman (2006) optimize various performance variables, for example the grid size,

Feature Combination	Weights
S + C	$\omega_s = 1$, $\omega_c = 0.6$
S + T	$\omega_s = 1$, $\omega_t = 0.4$
S + G	$\omega_s = 1$, $\omega_g = 0.4$
C + T	$\omega_c = 1$, $\omega_t = 0.6$
C + G	$\omega_c = 1$, $\omega_g = 0.8$
T + G	$\omega_t = 0.8$, $\omega_g = 1$
S + C + T	$\omega_s = 0.6$, $\omega_c = 0.6$, $\omega_t = 1$
S + C + G	$\omega_s = 0.3$, $\omega_c = 1$, $\omega_g = 0.6$
C + T + G	$\omega_c = 0.6$, $\omega_t = 1$, $\omega_t = 1$
S + T + G	$\omega_s = 0.6$, $\omega_t = 0.6$, $\omega_g = 1$

Table 3.2: Optimal combination weights (ω) for each feature-vector in the $\mathcal{FS}$ for Level 4 (high-level metric learning) on the Oxford flower database.

code-book size, etc. using the validation set. They report a classification performance of 81.3% with 40 training images and 20 validation images. The best result in this work is 64.2% with 40 training images (note ten images from the 40 training images were used as a validation set for determining weights for high-level metric learning). A second major difference is the distance measure used. Nilsback and Zisserman (2006) used the chi-squared (χ^2) distance. The chi-squared distance between point $\vec{x}_1$ and $\vec{x}_2$ is defined as:

$$d_{\chi^2}(\vec{x}_1, \vec{x}_2) = \sum_p \frac{(x_{1p} - x_{2p})^2}{(x_{1p} + x_{2p})} \tag{3.4}$$

Since we are using histogram-based features, the χ^2 measure is a more natural distance measure than the Euclidean measure. Even though the results reported in this section are encouraging in highlighting the efficacy of metric learning methods, it should be note that Euclidean distance may not be most suitable with histogram-based feature-vectors. This is due to the nature of histogram-based features. For example we can see from figure 3.1 that most flowers have a yellow color. The largest value in the color histogram will correspond to the yellow color. A small shift in the color (for example, due to some clutter or occlusion) which does not affect the nature of the object to be recognized will result in a large difference in Euclidean distance. Therefore, with histogram-based features, a more general and simple chi-squared distance is an effective distance measure (Chapelle et al., 1999).

Chapter 4

Complete Metric Learning

This chapter deals with *complete* metric learning. As discussed in chapter 3, complete metric learning deals with the learning of both the diagonal and the off-diagonal terms of the matrix A in equation 2.23. The conventional metric learning algorithms which are proposed in nearest neighbor classification settings are in fact complete metric learning algorithms. A novel complete metric learning algorithm (MEGM) is introduced in this chapter which is based on the minimization of the gradient of the MSE in the nearest neighbor framework (section 4.2). The performance of MEGM is compared with other metric learning approaches, for example Neighborhood Component Analysis (NCA) which aims to learn a metric to maximize the margin of a classifier. It is shown that the proposed algorithm not only results in significant improvement in the performance of the k-nearest neighbor classifier, but also outperforms other metric learning algorithm on most data sets. It should be noted that the MEGM algorithm minimizes the empirical risk only. As discussed, a generalization term can be included in the formulation. Current methods for complete metric learning are presented in section 4.1. The comparison of the performance of the proposed algorithm with other state of the art metric learning algorithms on major UCIML data sets (Appendix F), face, digits and object databases is given in section 4.3. The experimental results are discussed in section 4.3.

4.1 Related Work

The proposed MEGM metric learning algorithm in this chapter is similar to the method proposed by Lowe (1995), where a gradient-based technique is used to estimate the relevancy of each feature. That is, only the diagonal terms of the matrix A are estimated. In other words, the method proposed by Lowe (1995) is a semi-naive metric learning algorithm. MEGM, on the other hand, is a complete metric learning algorithm, as it learns a

Algorithms	Description
MEGM	A complete metric learning algorithm based on the minimization of the gradient of mean-square-error's.

Table 4.1: List of Algorithms proposed in chapter 4

full distance matrix and hence is potentially superior to the technique proposed by Lowe (1995). The semi-naive metric learning method by Lowe (1995) is one of the first metric learning algorithms applied to k-nearest neighbor methods. To our knowledge, they for the very first time actually viewed feature selection and feature weighting as a metric learning problem. It is somewhat strange that the most later work in metric learning (especially complete metric learning) did not acknowledge Lowe's (1995) work. David Mackay, however, mentioned Lowe's metric learning algorithm as a variant of GP modeling (MacKay, 2003, chapter 45).

The work of Xing et al. (2002) is considered to be one of the first proposed (complete) metric learning methods. As mentioned in section 2.4.1, typical metric learning algorithms aim to reduce intra-class and increase inter-class distances. The technique proposed in Xing et al. (2002) has the same goal. It should be noted that the work of Xing et al. (2002) turned out to be a trend setter. Subsequent methods proposed in metric learning shared the same goal and the idea of metric learning which Lowe (1995) proposed was somewhat sidelined. The method presented in Xing et al. (2002) used the class label information in terms of pairwise constraints. For example, any two points having the same label is an equivalence constraint and two points having a different label is an inequivalence constraint. Let us say that we denote all equivalence constraints by $\mathcal{S}$ and in-equivalence constraints by $\mathcal{D}$. Xing et al. (2002) formulated the metric learning problem as:

$$\begin{aligned} & \min\nolimits_A \sum_{(\vec{x}_i,\vec{x}_j)\in\mathcal{S}} \|\vec{x}_i - \vec{x}_j\|_A^2 \\ \text{such that} \quad A \succeq 0, & \sum_{(\vec{x}_i,\vec{x}_j)\in\mathcal{D}} \|\vec{x}_i - \vec{x}_j\|_A^2 \leq 1 \end{aligned} \tag{4.1}$$

The positive-semi-definite constraint $A \succeq 0$ is needed to ensure that the matrix A results in a valid metric. The main disadvantage of the approach is the presence of the positive-semi-definite constraint, which is difficult and expensive to maintain. Even though the problem is posed as a convex optimization problem, it may not be solved efficiently as it does not fall under any category of quadratic or semi-definite programming. Nevertheless, Xing et al. (2002) presented their metric learning approach in a very clear and concise way: a metric parameterized by the matrix A is to be learned such that the intra-class distances are minimized and inter-class distances are maximized. The approach also inspired various other metric learning algorithms (Goldberger et al., 2005; Weinberger et al., 2009; Nguyen and Guo, 2008).

NCA, proposed by Goldberger et al. (2005), maximizes the margin by minimizing the probability of error under stochastic neighborhood assignment. In particular, each point i selects another point j as its neighbor with some probability p_{ij}. In other words, p_{ij} denotes the probability that point $\vec{x}_i$ has the same label as point $\vec{x}_j$. The p_{ij} is defined as a softmax over Euclidean distances in the transformed space, parameterized by the matrix L:

$$p_{ij} = \frac{\exp(-\|L\vec{x}_i - L\vec{x}_j\|^2)}{\sum_{k\neq i}\exp(-\|L\vec{x}_i - L\vec{x}_k\|^2)} \tag{4.2}$$

where $A = L^T T$ (section 2.4). If C_i denotes the set of all the data points that share the same class label with $\vec{x}_i$, the objective function of NCA can be written as:

$$g(L) = \sum_{i}^{N} \log \left(\sum_{j \in C_i} p_{ij} \right) \tag{4.3}$$

The maximization of the objective function in equation 4.3 actually maximizes the expected number of data points correctly classified. The NCA metric learning algorithm was a motivation behind our proposed MEGM metric learning algorithm. Both algorithms minimize/maximize an objective function using stochastic gradient (descent/ascent) methods. The two algorithms, however, represents two very different approaches. NCA is a margin maximization metric learning algorithm whereas MEGM is a MSE minimization metric learning algorithm. The empirical results in section 4.3 suggest that MEGM performs better than NCA on most data sets. In section 4.4, an approach to combine both MEGM and NCA to improve the generalization capacity of MEGM will be described.

Weinberger et al. (2009) proposed a classical margin maximization metric learning algorithm. The proposed Large Margin Nearest Neighbor (LMNN) metric learning algorithm is posed as a convex optimization problem, and thus convergence to the global solution is guaranteed. The algorithm can be considered as a modification of Xing et al.'s (2002) and Goldberger et al.'s (2005) metric learning algorithm. The algorithm aims to learn a matrix parameterizing the linear transformation of the data. The cost function of the LMNN algorithm can be defined as:

$$g(L) = \sum_{ij} \eta_{ij} \|L(\vec{x}_i - \vec{x}_j)\|^2 + c \sum_{ijl} \eta_{ij}(1 - y_{il})[1 + \|L(\vec{x}_i) - \vec{x}_j\|^2 - \|L(\vec{x}_i) - \vec{x}_l\|^2]_+ \tag{4.4}$$

where $[.]_+$ denotes the standard hinge loss. The c parameter in equation 4.4 is some positive quantity and is set by cross-validation. The $\eta_{ij} \in \{0, 1\}$ indicates if data point $\vec{x}_i$ is the neighbor of $\vec{x}_j$. The target neighbors are identified as the data points in the neighborhood sharing the same class labels (Euclidean distance is used). The first term in equation 4.4 penalizes large distances between each data point and its target neighbors, whereas the second term penalizes small distances between each data point and all other data points having a different label. It can be seen that LMNN builds on the idea of Xing et al. (2002) and seeks a transformation such that in the transformed space data points belonging to the same class are close together and vice-versa. Similarly, it also builds on the idea of Goldberger et al. (2005) and seeks a transformation such that the margin is maximized. The LMNN method shares the disadvantage of method proposed by Xing et al. (2002). It optimizes for matrix A instead of matrix L and must maintain the constraint $A \succeq 0$. Therefore, a special optimization solver is needed for efficient implementation.

The Relevant Component Analysis (RCA) metric learning algorithm proposed by Bar-Hillel et al. (2003) is designed on an information theoretic basis and uses only closed form expressions of the data. The RCA algorithm is a simple and efficient algorithm for learning a full rank distance metric. It constructs a distance metric from a weighted sum of in-class covariance matrices. It is similar to PCA and LDA in its reliance on second order statistics. RCA applies a global linear transformation to assign large weights to relevant dimensions and low weights to irrelevant dimensions. The relevant and irrelevant dimensions are estimated using chunklets. A chunklet is defined as a subset of data points that are known to belong to the same class. In the following, the steps taken by RCA algorithm for each chunklet to learn a metric will be explained:

- Subtract the mean of chunklet from all the points in the given chunklet.
- Compute the covariance matrix of all the centered data points in the chunklets. If there are q points in r chunklets, and each chunklet j contains the data points $\{\vec{x}_{ji}\}_{i=1}^{N_j}$ with mean $\vec{m}_j$, the covariance matrix in RCA is computed as:

$$A = \frac{1}{q}\sum_{j=1}^{r}\sum_{i=1}^{N_j}(\vec{x}_{ji}-\vec{m}_j)(\vec{x}_{ji}-\vec{m}_j)^T \tag{4.5}$$

- The matrix $\hat{A}$ such that $\hat{A} = A^{-\frac{1}{2}}$ is the resulting matrix parameterizing the transformation (note that $\hat{A} = L$).

Relief-based feature selection algorithms will be explained in chapter 5 (Kira and Rendell, 1992; Yijun and Dapeng, 2008). Recently, the family of Relief-based feature selection and weighting algorithms have been modified for metric learning (Chang, 2010; Yijun, 2007; Yijun and Jian, 2006). Relief is a simple but effective feature selection algorithm and some good results have been reported regarding their metric learning capability.

Another notable metric learning algorithm is the information theoretic metric learning algorithm ITML (Davis et al., 2007; Davis and Dhillon, 2008). The problem is formulated as that of minimizing the differential relative entropy between two multivariate Gaussian distributions under constraints of the distance function.

4.2 MEGM Metric Learning

In a typical regression setting, an unknown function $f : R^D \to R$ is learned from the training data $\{(\vec{x}_1, y_1), (\vec{x}_2, y_2), ...(\vec{x}_N, y_N)\}$, where $\vec{x}_i$ is a data point and y is the corresponding target value. The function $\hat{f}$ is chosen to be the one that minimizes some loss function such as MSE, etc. We can define the MSE of $\hat{f}$ as:

$$\mathrm{MSE}(\hat{f}) = \frac{1}{2}\sum_{i=1}^{N}(f(\vec{x}_i) - \hat{f}(\vec{x}_i))^2 \tag{4.6}$$

For classification tasks having C classes, we can write the error function in equation 4.6 as:

$$\mathrm{MSE}(\hat{y}) = \frac{1}{2}\sum_{t=1}^{C}\sum_{i=1}^{N}(y_{ti} - \hat{y}_{ti})^2 \tag{4.7}$$

where $\hat{y}_{ti}$ denotes the predicted probability of point $\vec{x}_i$ to be belonging to class t and y_{ti} denotes the actual label (either zero or one) of point $\vec{x}_{ti}$. For brevity, $\hat{y}(\vec{x}_{ti})$ has been denoted with $\hat{y}_{ti}$ and $y(\vec{x}_{ti})$ with y_{ti}. In the following discussion, it will be assumed that there are only two classes to make the derivations simple. As discussed in section 2.1.3, for any query point $\vec{x}_i$, nearest neighbor methods predict the label $\hat{y}_i$ of $\vec{x}_i$ by considering the labels of the k-nearest neighbors of $\vec{x}_i$. In order to have a smooth boundary, each of the k neighbors vote for the label of $\vec{x}_i$ based on its distance from $\vec{x}_i$. For simplicity, let us replace the kernel $k(.,.)$ with V_j in equation 2.13 and rewrite it as:

$$\hat{y}(\vec{x}_i) = \frac{\sum_j y_j V_j}{\sum_j V_j} \tag{4.8}$$

The vote V_j cast by each neighbor around the query point $\vec{x}_i$ is usually chosen to be a function that decays exponentially as the distance from the query point increases. For example we can define V_j as:

$$V_j \quad = \quad \exp\left(\frac{-d^2(\vec{x}_i, \vec{x}_j)}{2\sigma^2}\right) \tag{4.9}$$

$d^2(\vec{x}_i, \vec{x}_j)$ in equation 4.9 is often the Euclidean metric, but it can be replaced by a more general metric: that is $d_A^2(\vec{x}_i, \vec{x}_j)$. Again as discussed in section 2.4, if $A = L^T L$, then $d_A^2(\vec{x}_i, \vec{x}_j) = (L\vec{x}_i - L\vec{x}_j)^T(L\vec{x}_i - L\vec{x}_j)$. Since MSE is a function of $\hat{y}$, and $\hat{y}$ depends on $\|\vec{x}_i - \vec{x}_j\|_A^2$, the MSE can be minimized by selecting an optimal value of matrix A. In other words, a change in A induces a change in the distances, which can alter the votes V_j. As learning the matrix A requires the maintenance of the expensive constraint that A be positive-semi-definite, a simple alternative is to learn L rather than A. Obviously trying all possible values of L is not feasible. Some sort of search mechanism is required to find an optimal value of L. The votes V_j in equation 4.9 can be replaced by W_j. Like V_j, W_j is the vote of each neighbor and is based on its distance from the query point, but unlike V_j, the metric used to measure distances is parameterized by the matrix A. Therefore, W_j can be written as:

$$W_j = \exp\left(\frac{-\|L\vec{x}_i - L\vec{x}_j\|_2^2}{2\sigma^2}\right) \tag{4.10}$$

The proposed gradient-based technique (MEGM) is based on a gradient descent algorithm to minimize the MSE in equation 4.7. Let us denote the MSE of the predicted function as $\mathcal{E}$. Since a gradient descent based method is used for optimization, convergence to the global minimum is not guaranteed. The risk of local minima can be reduced by running the algorithm several times and choosing the output with minimum $\mathcal{E}$. We can write the gradient with respect to matrix matrix L as:

$$\begin{aligned}
\frac{\partial \mathcal{E}}{\partial L} &= \sum_{i=1}^{N}(y_i - \hat{y}_i)\left(-\frac{\partial \hat{y}_i}{\partial L}\right) \\
&= \sum_{i=1}^{N}(y_i - \hat{y}_i)\left(-\sum_j y_j W_j(-1)(\sum_j W_j)^{-2}\frac{\partial W_j}{\partial L} + (\sum_j W_j)^{-1}\sum_j y_j \frac{\partial W_j}{\partial L}\right) \\
&= \sum_{i=1}^{N}(y_i - \hat{y}_i)\left(-(\sum_j W_j)^{-1}\left(\frac{-\sum_j y_j W_j \frac{\partial W_j}{\partial L}}{\sum_j}\right) + (\sum_j W_j)^{-1}\sum_j y_j \frac{\partial W_j}{\partial L}\right) \\
&= \sum_{i=1}^{N}(y_i - \hat{y}_i)\left(-\frac{\sum_j (y_j - \hat{y}_j)\frac{\partial W_j}{\partial L}}{\sum_j W_j}\right) \qquad (4.11)
\end{aligned}$$

The size of the Gaussian kernel centered at the query point (σ in equation 4.10) is set proportional to the distance of the k-nearest neighbors. Generally the average distance of half of the nearest neighbors is used, as this measure is more stable under a varying distance metric (Lowe, 1995). Another motivation behind using only $\frac{k}{2}$ neighbors for calculating σ^2 is to reduce the effect of outliers during the calculation of the gradient. We can write σ^2 as:

$$\sigma^2 = \frac{1}{k/2}\sum_{m=1}^{k/2}\|\vec{x}_i - \vec{x}_m\|^2 \qquad (4.12)$$

$\frac{\partial W_j}{\partial L}$ in equation 4.11 can be derived as:

$$\frac{\partial W_j}{\partial L} = -W_j L\left(\frac{(\vec{x}_i - \vec{x}_j)(\vec{x}_i - \vec{x}_j)^T}{\sigma^2}\right) \qquad (4.13)$$

Combining equations 4.11 and 4.13 we can write the gradient of $\mathcal{E}$ with respect to matrix L as:

$$\frac{\partial \mathcal{E}}{\partial L} = L\sum_{i=1}^{N}(y_i - \hat{y}_i)\frac{1}{\sum_j W_j}\sum_j (y_j - \hat{y}_j)W_j\left(\frac{(\vec{x}_i - \vec{x}_j)(\vec{x}_i - \vec{x}_j)^T}{\sigma^2}\right) \qquad (4.14)$$

The Polack-Ribiere flavor of conjugate gradients is used to compute search directions, and a line search using quadratic and cubic polynomial approximations and the Wolfe-Powell stopping criteria is used together with the slope ratio method for guessing the initial step sizes (Press et al., 1988). An outline of MEGM metric learning algorithm is given in algorithm 1.

Algorithm 1 MEGM: A complete metric learning algorithm based on the minimization of the gradient of MSE.

```
Require:
  - $\vec{x}_0$: Testing data.
  - $\{\vec{x}_n, y_n\}_{n=1}^{N}$: Training data.
  - $k$: Number of nearest neighbors, $k$ is set to be floor(log2($N$))).
  - $L$: $P$-dimensional unit matrix.

  - Calculate matrix $L$ by using conjugate gradient method as described in the text
    (gradient of the objective function is given in equation 4.14).
  - Use matrix $L$ for 1-nearest neighbor classification.
```

4.2.1 Generalization Issues

Since MEGM minimizes an objective function which is actually a MSE function, one is likely to question its generalization capacity. It should be noted that the MEGM algorithm has at least two peculiar properties which may not lead to an over-fitted solution. First, there is a tunable parameter k, i.e. the number of nearest neighbors used when calculating the gradient of $\mathcal{E}$ with respect to L. As long as k is set reasonably (not too large, not too small, e.g. k = floor(log2(N))), the solution matrix L will not lead to over-fitting. Though the value of k is absolutely critical to k-nearest neighbor classification and should be optimized, the neighborhood size k is fixed in the MEGM formulation.

Secondly, it can be seen that there is no need for a σ^2 in equation 4.10, as when calculating the weights W_j, σ^2 can be incorporated in the matrix L. One can rewrite equation 4.10 as:

$$W_j = \exp\left(-\|L\vec{x}_i - L\vec{x}_j\|_2^2\right) \tag{4.15}$$

Though perfectly acceptable, such formulation is not used. Instead, σ^2 in equation 4.10 is set to the average distance of $k/2$ neighbors (equation 4.12). The experimental evaluation of the MEGM metric learning algorithm is given in section 4.3, where it is shown that MEGM performs extremely well on many data sets. However, the MEGM metric learning algorithm has two issues that needs to be addressed.

- First, the algorithm may not scale well to large numbers of dimensions. For example, consider data with 4000 dimensions. A gradient decent search in a space spanning 4000 dimensions may not be feasible. This issue can be resolved by reducing the dimensionality of the data by projecting to a lower dimensional subspace, for example, preprocessing data using the PCA technique. Of course, this strategy assumes that features are highly correlated and there is a minimal loss of information when dimensionality is reduced.

- The second issue is that of local minima. Since MEGM is based on a simple gradient descent on its objective function, some care has to be taken so that it does not suffer from local minima.

4.3 Experimental Evaluation of MEGM metric learning

In this section, the classification results of various metric learning algorithms on different UCIML, face, object and digit databases are presented (Frank and Asuncion, 2010). A short description of the various UCIML databases used is given in appendix F. The classification performance of MEGM is discussed and compared with other state of the art metric learning algorithms for example NCA, RCA and LMNN as described in section 4.2.

In all the experiments, MEGM is used with its standard settings as shown in the algorithm 1. That is, a 1-nearest neighbor classifier is used in the transformed space induced by the learned matrix L and the size of the neighborhood is consistently set equal to floor(log2(N)) for all databases, where N is the cardinality of the data.

4.3.1 UCIML Repository Databases

The number of data, features and classes for each UCIML database used is reported in table F.1. The error rate of each method is obtained using 40 rounds of 2-fold cross-validation. Prior to training, all features were normalized to have zero mean and unit variance. The classification performance in terms of the error rates of each of the following methods for different databases is shown in figures 4.1 and 4.2.

- **KNN:** A simple 1-nearest neighbor classifier using Euclidean distance.
- **RCA:** 1-nearest neighbor classification using the metric learned via the relevant component analysis metric learning algorithm (Bar-Hillel et al., 2003).
- **LMNN:** 1-nearest neighbor classification using the metric learned via the large margin nearest neighbor metric learning algorithm (Weinberger et al., 2009).
- **MEGM:** Algorithm 1.
- **NCA:** 1-nearest neighbor classification using the metric learned via the nearest component analysis metric learning algorithm (Goldberger et al., 2005).

As can be seen from figures 4.1 and 4.2, MEGM performed better than the others on 13 of the 19 UCIML databases. NCA, on the other hand, performed best on four. No metric learning algorithm improved k-nearest neighbor performance on one of the databases. Though MEGM performed well on most of the UCIML databases used, the databases (in terms of the number of data) are not particularly large (table F.1). The number of data in most databases also varies, for example *hayesroth* has only 135 data points, whereas *tic-tac-toe* has 958. This should be encouraging. Usually techniques relying on gradient descent methods require more training data to converge (Duda et al., 2006). From the results it seems that MEGM is capable of converging even with a small number of training data. In terms of the number of classes, there is not a huge variance. For example, most databases have either two or three classes with the exception of *dermatology* and *vowel* databases which have got six and ten classes respectively. On both of these databases, the performance of MEGM is better than the other metric learning algorithms. This suggests that MEGM is capable of handling a large number of classes better than the competing algorithms.

Though MEGM performed better than other approaches on most databases as shown in figure 4.1, NCA performance is also noteworthy especially on *monks1* and *statlog heart*. NCA performed marginally better than other techniques on *ionosphere*, *house vote* and *hepatitis* database. On *monks3* both MEGM and NCA performed best. Note, on *parkinson*, no metric learning algorithm resulted in any improvement on the k-nearest neighbor classifier.

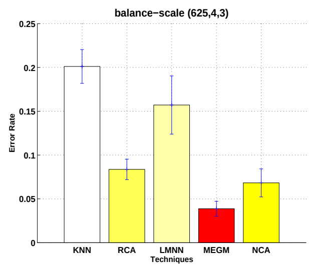
A few design issues in the MEGM formulation were discussed in section 4.2.1. One of them was the operation of MEGM in very high dimensions. The biggest databases in terms of the number of features are *sonar*, *dermatology* and *ionosphere* with 60, 34 and 34 features respectively. As can be seen, MEGM performed better on two (*sonar*, *dermatology*) out of these three databases. This suggests that MEGM is capable of learning a metric in high dimensions and is more efficient than other methods.

To compare the performance of various algorithms, a robustness measurement test introduced by Friedman (1994) is used. The test basically compares the robustness of one algorithm with others. It measures how well a particular method (say m) performs on average in situations that are most favorable to other methods. The robustness of a method is measured by computing the ratio b_m of its error e_m and the smallest error of all the other competing methods. That is:

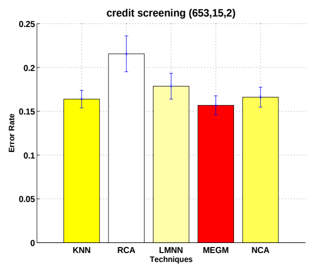
$$b_m = \frac{e_m}{\min_i e_i} \tag{4.16}$$

The best method m^* will have $b^*_m = 1$ and all other methods will have values larger than one. The larger the value of b_m, the worse the performance is of the m^{th} method in relation to the best one for that data set. In figure 4.3, the distribution of b_m for each method over all 19 databases is shown in the form of boxplots[1]. As can be seen, MEGM is the most robust of all. LMNN and NCA are also very robust, except in the case of a few outliers.

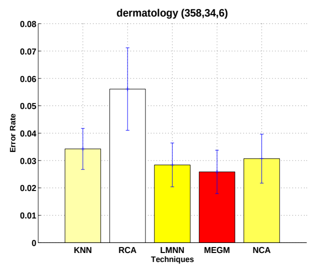
[1]Boxplots have been used in statistics for depicting the group of data in terms of a five number summary. That is, the minimum sample, the maximum sample, lower quartile, median and upper quartile of data is displayed. A boxplot may also indicate which observations, if any, might be considered outliers.



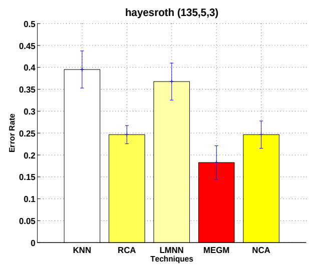
(a)



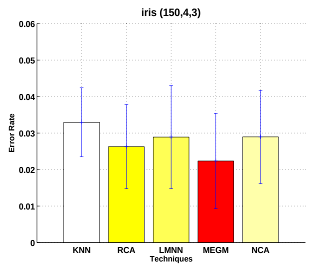
(b)



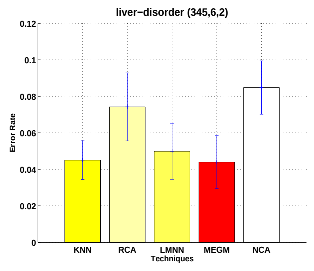
(c)



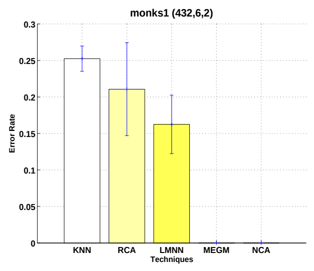
(d)



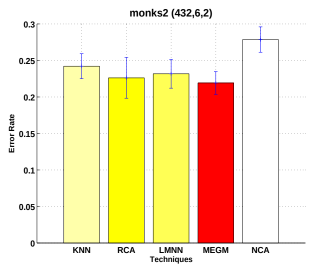
(e)



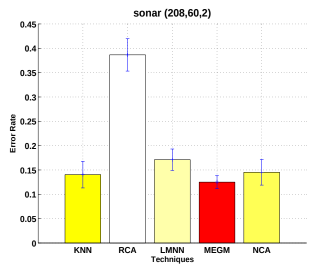
(f)



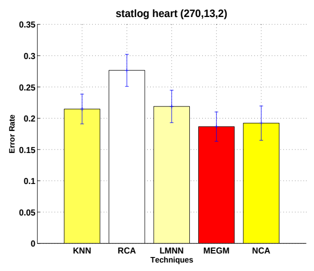
(g)



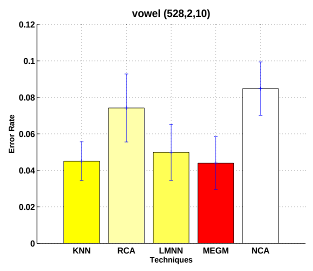
(h)



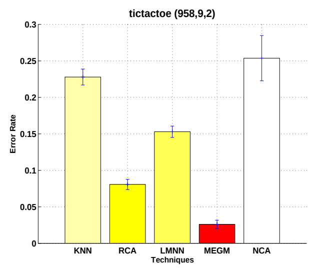
(i)



(j)

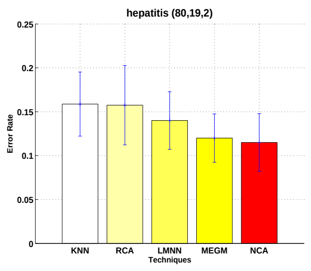


(k)

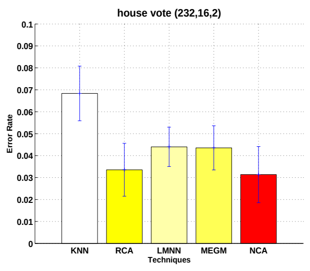


(l)

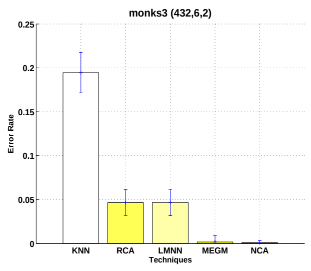
Figure 4.1: Error rate comparison of KNN, RCA, LMNN, MEGM and NCA on various UCIML databases (continued in figure 4.2). MEGM not only improved on k-nearest neighbor classification performance, but in all cases resulted in better performance than other competing metric learning methods of NCA, RCA and LMNN.



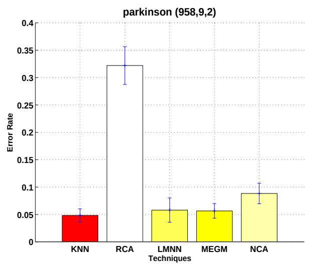
(a)



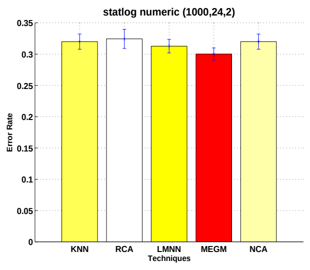
(b)



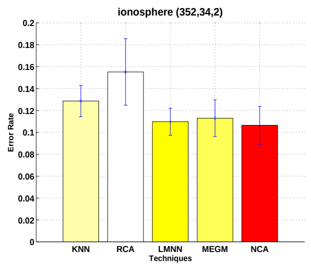
(c)



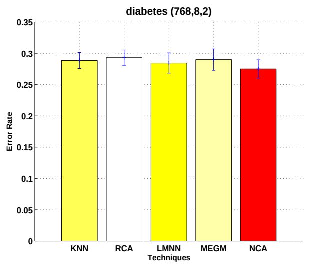
(d)



(e)



(f)



(g)

Figure 4.2: Error rate comparison of KNN, RCA, LMNN, MEGM and NCA on various UCIML databases (continued from figure 4.1).

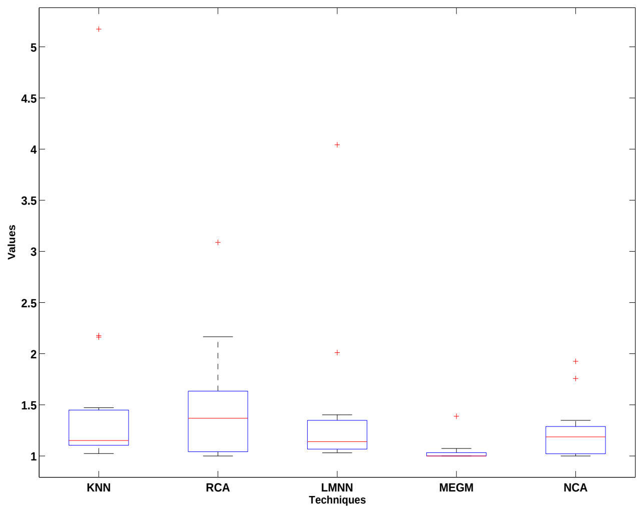


Figure 4.3: Boxplots are computed for KNN, RCA, LMNN, MEGM and NCA on the results of all UCIML databases which are reported in figure 4.1 and 4.2 using the method of Friedman (1994).

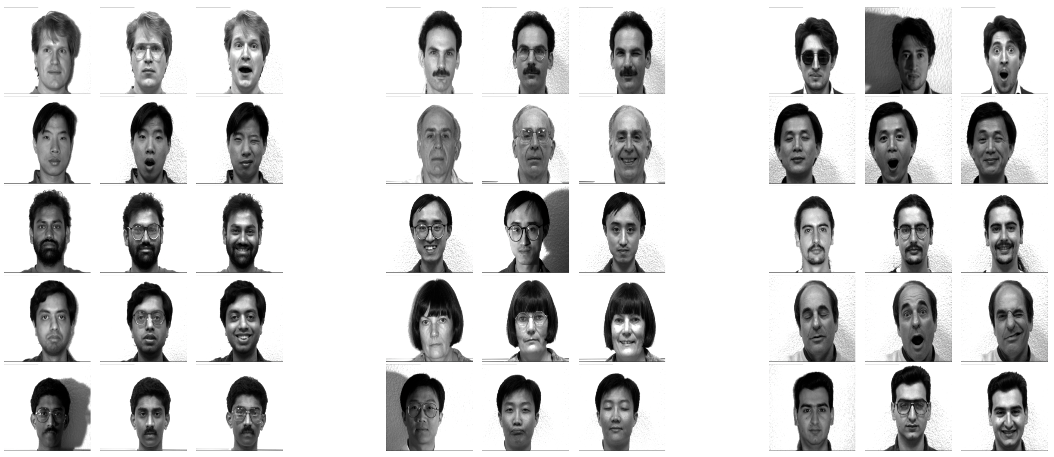


Figure 4.4: Example images (three images per category) from the yalefaces database.

4.3.2 Face Databases

In this section MEGM is tested in the setting of a simple retrieval task using face recognition databases. For a given query image, the task is to retrieve the image from the database that is the most similar. The details of various face recognition databases used are given in table 4.2. The yalefaces (figure 4.4), AT&Tfaces (figure 4.5) and yalefacesB (figure 4.4) databases are well known in face recognition research.

The yalefaces database constitutes images from *The Yale Face Database* (1997) which contains 165 grayscale images of 15 individuals. There are 11 images per subject, one per different facial expression or configuration: center-light, with glasses, happy, left-light, with no glasses, normal, right-light, sad, sleepy, surprised, and wink. The yalefacesB database constitutes images from *The Yale Face B Database* (2001) which is a much bigger database than the yalefaces. It contains 5760 single light source images of ten subjects, each seen under 576 different viewing conditions (9 poses $\times$ 64 illumination conditions). For every subject in a particular pose, an image with ambient illumination is also captured. The AT&Tfaces constitutes images from *The AT&T Face Database* (2002) which has ten different images of each of 40 distinct subjects. For some subjects, the images were taken at different times, varying the lighting, facial expressions (open, closed eyes, smiling, not smiling) and facial details (glasses, no glasses). All the images were taken against a dark homogeneous background with the subjects in an upright, frontal position (with tolerance for some side movement). The caltechfaces and caltechfacesB constitutes images from the

Database	#Data	#Features	#Classes	#Eigen-vectors	#Train/Class	#Test/Class
yalefaces	165	77760	15	50	4	7
yalefacesB	5850	307200	10	20	10	20
caltechfaces	435	47500	29	30	5	10
caltechfacesB	435	47500	2	30	50	100
AT&Tfaces	400	10304	40	50	5	5
Coil100	7200	16384	100	50	10	10
USPS	9298	256	10	50	20	10
Isolet	6238	617	26	30	20	10

Table 4.2: Details of Face, Object and Digit Databases used for classification

Figure 4.5: Example images (three images per category) from the AT&T face database.

Figure 4.6: Example images (three images per category) from the caltechfaces and caltechfacesB databases.

face category in the *Caltech-101 Object Database* (2006) . The face category in *Caltech-101 Object Database* (2006) has 435 images of around 20 people. The caltechfaces database is based on splitting the face category of *Caltech-101 Object Database* (2006) in 19 distinct categories, each belonging to a different person (refer to figure 4.7). On the other hand, caltechfacesB is based on splitting the face category of *Caltech-101 Object Database* (2006) in only two classes, male and female.

The feature-vectors representing an image constitutes the intensity value of the pixels. As can be seen from table 4.2, due to the resolution of images, the number of features is extremely large (around 30000 for yalefacesB). When discussing the known issues in the MEGM formulation in section 4.2.1, it was mentioned that since MEGM algorithm may not scale well to the number of features, a reasonable approach is to reduce the

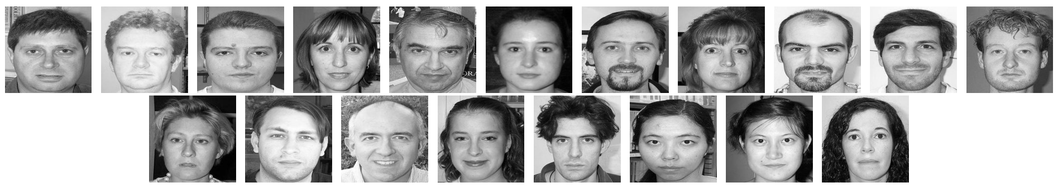


Figure 4.7: For illustration, names have been assigned to each category of caltechfaces. The names of the categories are: (First row, left to right) David, Alan, Peter, Sarah, Matthew, Alice, John, Michelle, Kevin, Tom, Ali. (Second row, left to right) Kate, Michael, Tim, Meg, Denial, Fei, Reeta, Allison. Refer to figure 4.13.

dimensionality of the data before applying MEGM. Therefore, due to the high dimensionality of face databases, all images are pre-processed using PCA technique for efficiency[2]. Pre-processing images using PCA is a common approach in object recognition research to reduce dimensionality for computational efficiency (Grauman and Darrell, 2005). For illustration, the first 16 Eigen-faces (or Eigen-vectors) of yalefaces are shown in figure 4.8 (Turk and Pentland, 1991). Also the reconstruction of some of the sample images using the first 100 Eigen-faces is shown in figure 4.9. It can be seen that even the first 100 Eigen-vectors contains very discriminative information and images can be well discriminated in the subspace. Rather than working in a space of 77760 features, due to a PCA transformation, we are working in a space of only 100 or fewer features. This results in a huge gain in computational efficiency. The number of Eigen-vectors is an important factor. Although, this parameter should be optimized, the number of Eigen-vectors was set to be as small as possible to make the learning problem more challenging. The number of Eigen-vectors for each database is given in table 4.2. After the PCA transformation, the coefficients of the Eigen-vectors constitutes the feature-vectors which are used for training the classifier.

The classification performance in terms of the correctness rate of each of the following methods for different face databases is shown in figure 4.10.

- **KNN:** A simple 1-nearest neighbor classifier with the Euclidean distance.
- **MEGM:** Algorithm 1.
- **NCA:** 1-nearest neighbor classification using the metric learned via nearest component analysis metric learning algorithm (Goldberger et al., 2005).
- **SVM:** SVM classifier (appendix D).

To obtain the SVM classification results, a multi-class SVM classifier with a Gaussian kernel is used. Refer to appendix D for a detailed derivation of SVM classifiers. SVM classifiers have two tunable parameters, $\mathcal{C}$ and σ. For the experiments in this section

[2]PCA introduced by Pearson (1901) is a widely used technique for exploratory data analysis. It involves the Eigenvalue decomposition of the covariance matrix of the data, usually after normalizing the data, such that mean of the data across each feature is zero. Since learning a metric actually results in linear transformation of data (section 2.4), PCA can be viewed as an orthogonal linear transformation that transforms the data to a space such that the greatest variance by any projection of the data in that space lies on the first feature (called the first principal component), the second greatest variance on the second feature (second principal component), and so on.

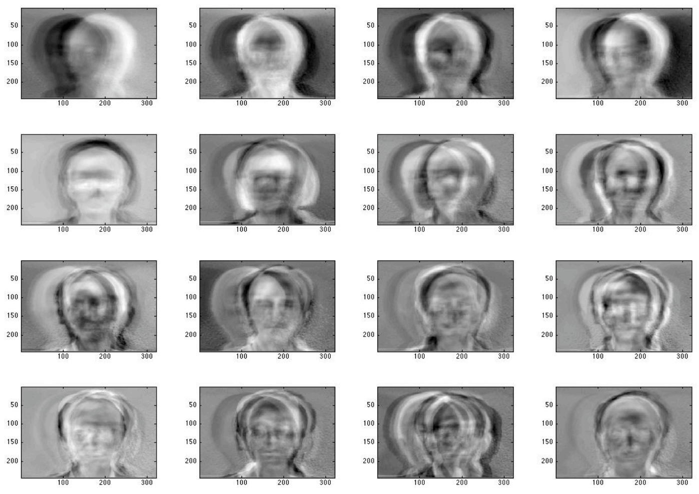
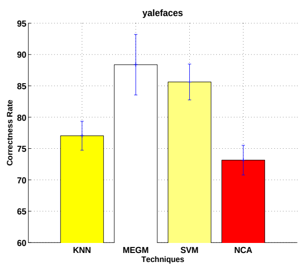


Figure 4.8: The first 16 Eigen-vectors of yalefaces database. Note the discriminative information that each Eigen-vector carries for example the face area, eyes, etc. The background of each image seems to be ignored.

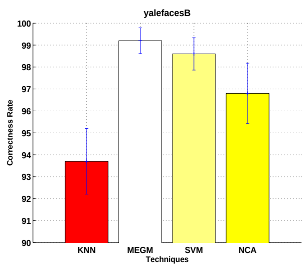
Figure 4.9: Reconstruction of some sample images from yalefaces database from first 100 Eigen-faces. Note the images are distinct from each other even though they are represented with far less information.

only, the $\mathcal{C}$ parameter is tuned through cross-validation by selecting its value from the following set: $\{1, 10, 100, 1000\}$. The σ parameter is not optimized. It is set equal to the average of the distances between the training data. This approach to setting σ alleviates the need to tune the σ parameter and has been shown to be effective for object related databases (Zhang, Marszalek, Lazebnik and Schmid, 2006). Since the SVM is a binary classifier, to obtain multi-class results a one-versus-all strategy is used.

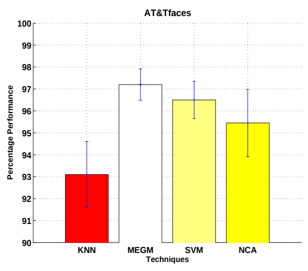
The results of the various methods on the face databases are shown in figure 4.10. The number of training images per category, number of testing images per category and the number of Eigen-vectors used for each database is given in table 4.2. As for the UCIML databases, MEGM resulted in significant improvement over the performance of k-nearest neighbor classification, despite using a small number of Eigen-vectors. It performed better than all other methods for all five databases. From figures 4.10(a) and 4.10(d), it can be seen that NCA did not result in any performance improvement over 1-nearest neighbor classifier on the yalefaces and caltechfaces databases. Though the performance gain of MEGM algorithm over SVM is not significant, given that the training of an SVM classifier is expensive, as it involves optimizing the $\mathcal{C}$ parameter through cross-validation, the MEGM results are encouraging. Also, an SVM is a binary classifier, which means that



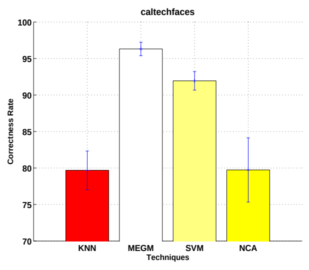
(a)



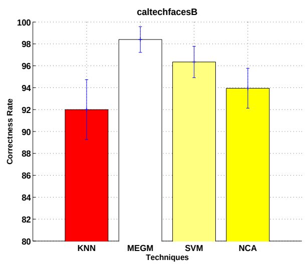
(b)



(c)



(d)



(e)

Figure 4.10: Percentage performance (correctness rate) of KNN, MEGM, SVM and NCA on various face databases. The mean and standard deviation of the correctness rate over ten runs (each run with a different training data) is reported.

each query image is evaluated by all trained SVM classifiers (in a one-versus-all strategy). The classifier with the highest confidence in its decision predicts the class label. On the other hand, MEGM is based on a simple linear transformation of the data followed by a simple 1-nearest neighbor classification. This renders MEGM favorable in those cases where there are a large number of classes and a decision has to be made in real time[3].

A demonstration of a simple nearest neighbor classification strategy (retrieval task) is shown in figure 4.11. The first four nearest neighbors of example query images from the caltechfaces database obtained via KNN and MEGM algorithms are compared. It can be seen that the retrieval efficiency of a 1-nearest neighbor classifier appears much better in the transformed space induced by MEGM than in the original space.

In the following, the procedure that was followed (and is followed in the later chapters) to obtain the results shown in figure 4.10 is explained. The results for a particular method in figure 4.10 are obtained by calculating the confusion matrix. The confusion matrix M is a $C \times C$ dimensional matrix (C is the number of classes). The M_{ij} element of the confusion matrix denotes the number of testing images that have been classified as belonging to category i but actually belong to category j. One hopes for a diagonal confusion matrix, which indicates that all the testing images have been classified to the categories to which they actually belong. The confusion matrix is evaluated using the

[3]This is a general argument and applies equally to the debate concerning the use of any binary classifier (for example boosting (Schapire and Singer, 1998), SVM, etc.) versus nearest neighbor methods (for example KNN, NCA or MEGM, etc.).

following formula to obtain the results in figure 4.10:

$$\text{perf}(M) = \sum_{i=1}^{C} \left(\frac{M_{ij}}{N_c} \right) \tag{4.17}$$

where N_c denotes the total number of testing data points belonging to category c. For example, visualization of the confusion matrices obtained with KNN and MEGM for caltechfaces are shown in figure 4.13. An alternative is to calculate the average of the number of all the testing images correctly classified. Since the number of testing points belonging to each category may vary, it is more reasonable to use the formula given in equation 4.17.

It can be seen from figure 4.13(a) that the confusion matrix for KNN has many confusions. For example, Michelle is confused with Sarah and Alice, David is confused with Peter, Tim is confused with Matthew, etc. The source of some of these confusions can be seen in figure 4.7. For example, it can be seen that that Michelle and Sarah shares may facial features. In the confusion matrix of MEGM, there are fewer confusions. The notable cases are: Tim is confused with Matthew; the resemblance can also be seen in figure 4.7, Meg is confused with Alice. But overall, MEGM results in a classification efficiency of around 98% compared with 94% for KNN.

Confusion matrices are a natural way to represent and evaluate the classification results for databases consisting of more than two classes. For a database that consists of only two classes, a natural way to plot the results is in the form of receiver operating characteristic (ROC) curves, which are generally used in signal detection theory (Fawcett, 2006). In machine learning, ROC curves plot the sensitivity (true positives) versus the false positives (1 - specificity) for a binary classifier as the discrimination threshold is varied. For example the ROC curves for KNN and MEGM for caltechfacesB are shown in figure 4.12.

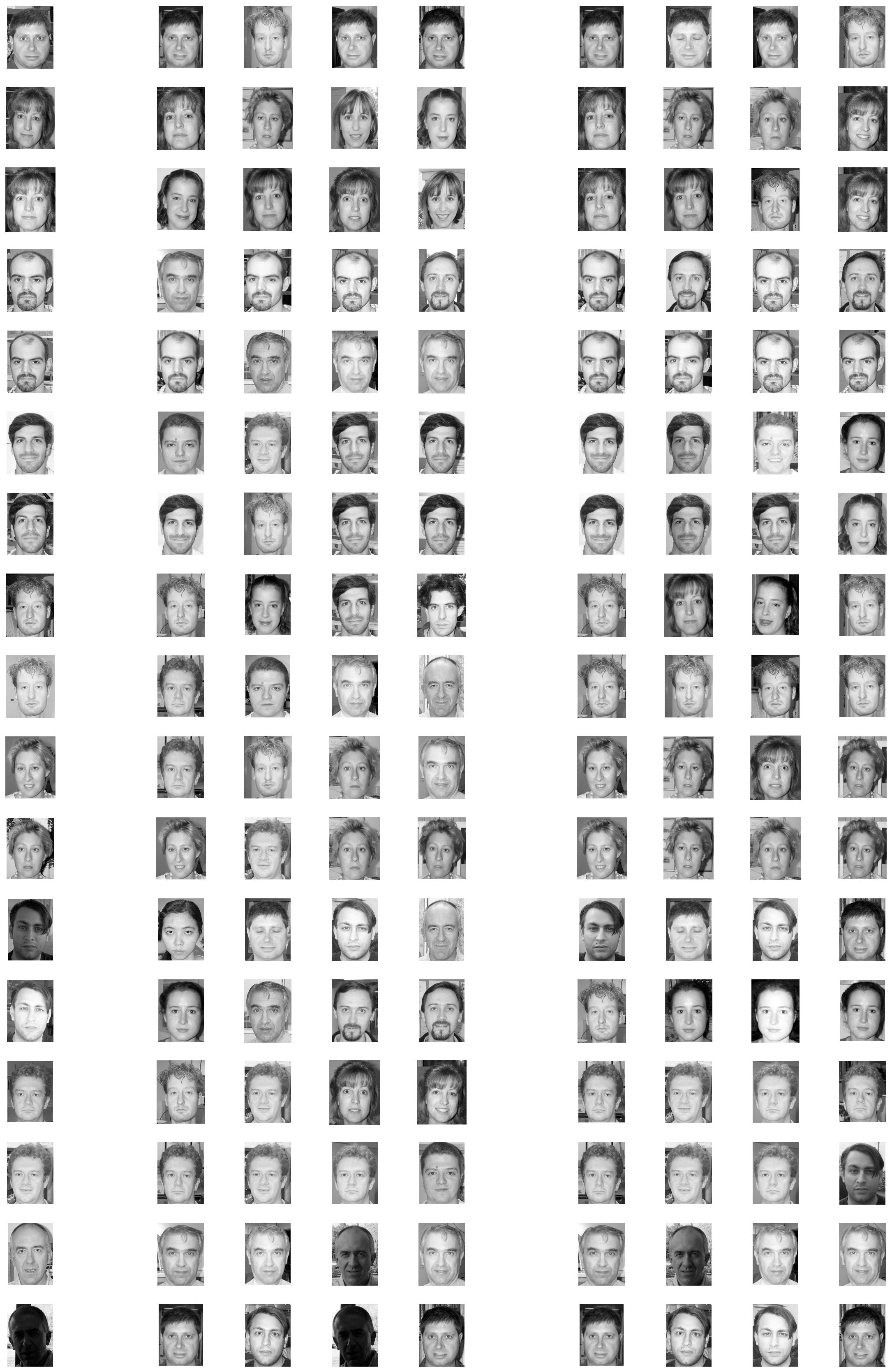


Figure 4.11: Example of retrieval results for the caltechfaces database. Each row represents one example. Column 1 shows the query image. Columns 2-4: show the first 4 nearest neighbors obtained with the Euclidean distance. Column 5-9 show the first 4 nearest neighbors obtained with MEGM.

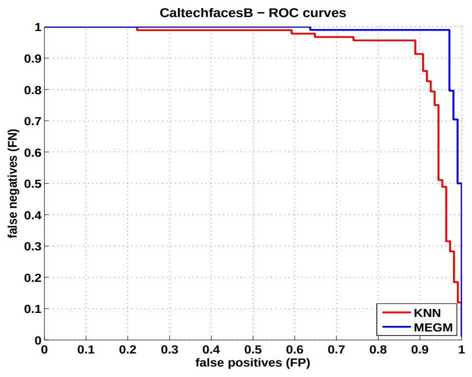
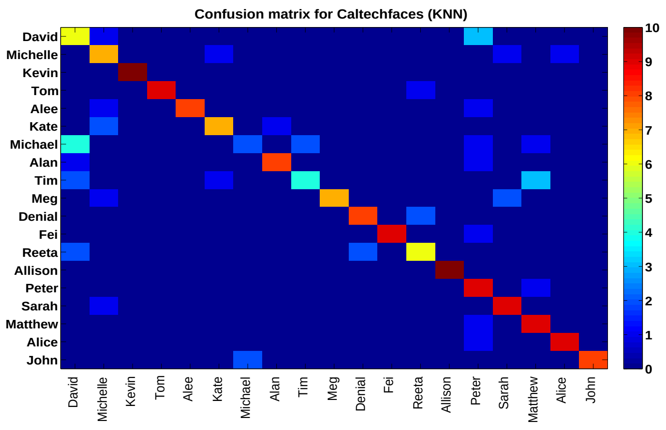
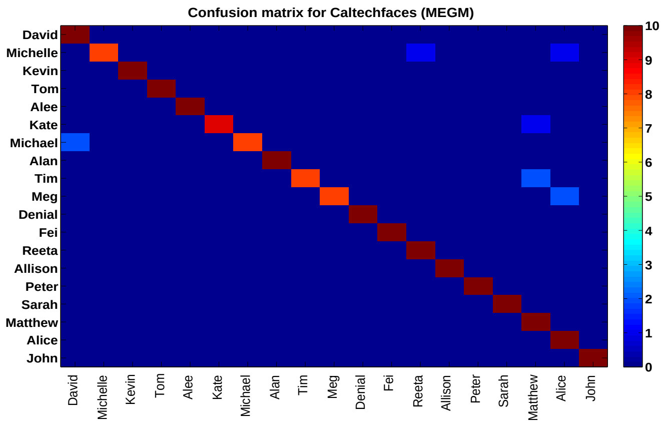


Figure 4.12: ROC curves of KNN and MEGM methods for caltechfacesB database.



(a)



(b)

Figure 4.13: Figure 4.13(a): Confusion matrices obtained with KNN (results labeled as KNN in figure 4.10(d)), Figure 4.13(b): KNN followed by MEGM for caltechfaces (results labeled as MEGM in figure 4.10(d)).

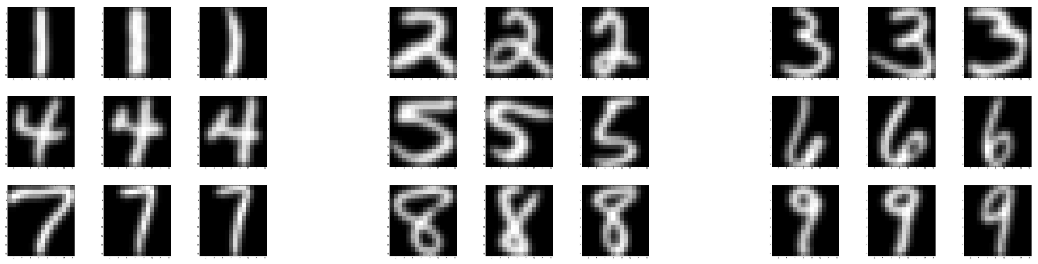


Figure 4.14: Example images (three images per category) from the USPS digit database.

4.3.3 USPS Digit, Isolet and Coil100 databases

In this section, the classification performance in terms of the correctness rate of KNN, MEGM, NCA and SVM methods (section 4.3.2) for USPS, Isolet and Coil100 databases is discussed. Both the USPS digit database (figure 4.14) and the Coil100 (figure 4.15) object database are well known in object recognition research.

The USPS digits data was gathered at the Center of Excellence in Document Analysis and Recognition (CEDAR) at State University of Buffalo, as part of a project sponsored by the US Postal Service (Hull, 1994). The human error rate on this database has been estimated to be 2.5%. This shows that this is a hard recognition task. It contains normalized grey scale images of size 16×16. The database has traditionally been divided into a training set of 7291 images and a test set of 2007 images. This setting has been used as a benchmark for comparing the performance of various machine learning algorithms. It should be noted that these two sets (training and testing images) were actually collected in slightly different ways and the images in the test set are much harder to classify than the images in the training set. This property of the database makes it a less desirable choice for comparing learning algorithms. That is why, in this work, the training and testing sets are merged and the data is randomly shuffled and divided into new training and testing sets. The Coil100 object database consists of 100 object categories and is one of the few extensive object databases existing in object recognition research (Nene et al., 1996). Each object in the image is present uncluttered. Each category has 72 images, where each image is taken at a different angle of the object. The Isolet (Isolated Letter Speech Recognition) database consists of 26 categories. 120 subjects spoke the name of each letter of the alphabet twice (Frank and Asuncion, 2010). Hence, we have 52 training examples from each speaker. The goal is to classify letters into one of the 26 categories. The database has been used as a benchmark for testing the scalability of various machine learning algorithms. It has also been used to test the performance of learning algorithms in the presence of noise (Frank and Asuncion, 2010).

The results on the USPS, Isolet and Coil100 databases are shown in figure 4.16. The MEGM method performed well on all three databases, resulting in improvement in the performance of k-nearest neighbor classifier. Also the results of MEGM are better than NCA. SVM, however, performed best on the USPS and Isolet databases. A strange outcome can be seen in figure 4.16(c). On the Coil100 database, a simple k-nearest neighbor classifier has performed better than a standard SVM classifier. The comparative results

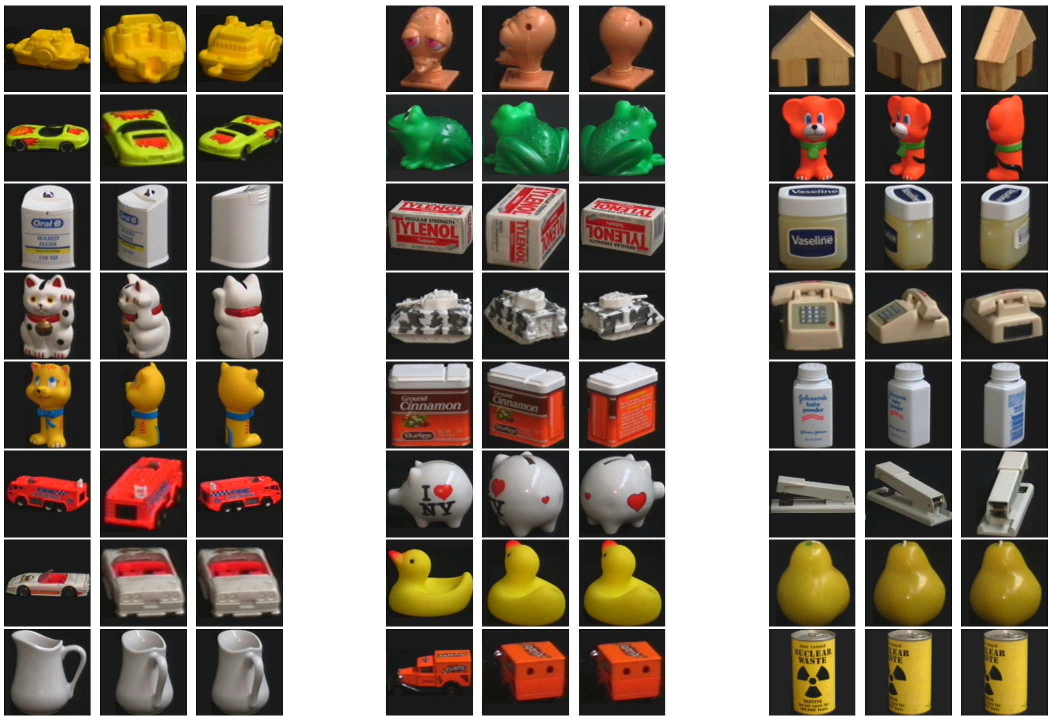


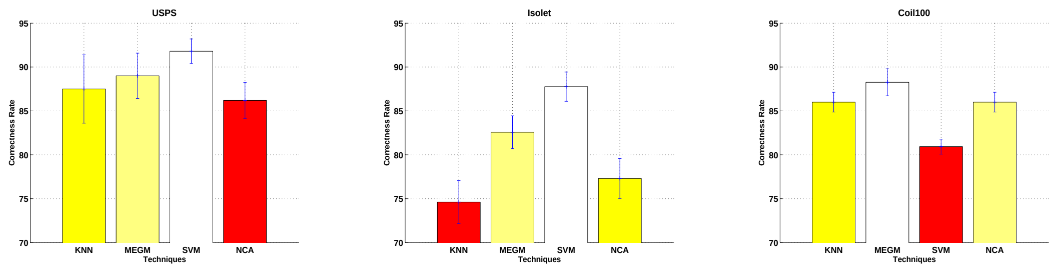
Figure 4.15: Some example images (three images per category) from the Coil100 object database.

on the Coil100 database suggest that the SVM policy of training a multitude of classifiers in a one-versus-all strategy may not always be optimal. This will be discussed in chapters 7 and 8. The neighborhood of any query point can contain valuable information for predicting the label of that query point. This is evident from some good results with nearest neighbor methods such as KNN, MEGM and NCA on the Coil100 database in figure 4.16(c).

4.4 Summary

In this chapter, complete metric learning algorithms were discussed and a novel complete metric learning algorithm (MEGM) was proposed. On various UCIML, face, digit and object databases, it was shown that MEGM metric learning algorithm not only results in classification improvement of the k-nearest neighbor classifier, but it also performed better than other complete metric learning algorithms. The performance was also compared with a standard SVM classifier.

The main advantage of the proposed MEGM algorithm is its simplicity. It deals with multi-class problems effortlessly, as opposed to binary classifiers such as SVM, where either a one-versus-one or a one-versus-all strategy is used. SVM training and testing is computationally expensive. For example, as training involves training 100 classifiers, and to classify an image all classifiers vote before reaching a consensus on the prediction, it takes a very long time to train and test an SVM classifier on the Coil100 database. On the other hand, once a metric is learned using MEGM, only a simple nearest neighbor classification is required. In typical object recognition tasks, where the number of classes is



(a) (b) (c)

Figure 4.16: Percentage performance (correctness rate) of KNN, MEGM, SVM and NCA on USPS, Isolet and Coil100 object databases. The mean and standard deviation of the correctness rate over ten runs (each run with different training data) is reported. The number of training images per category, number of testing images per category and the number of Eigen-vectors used for each of these databases is given in table 4.2.

very large, nearest neighbor methods should be preferred for their computational efficiency. Therefore, k-nearest neighbor methods equipped with a proper distance metric can be extremely useful.

A disadvantage of the proposed MEGM algorithm is the absence of a regularization term in its objective function. One could investigate modifying the objective function of MEGM algorithm to include an explicit regularization term. One could also investigate combining the objective functions of MEGM and NCA to improve classification results. Since MEGM is based on the minimization of MSE, and NCA is based on maximizing the margin explicitly, combining the two approaches would be an interesting idea. That is, one could investigate learning a metric by simultaneously maximizing the margin and minimizing the MSE. There has been a lot of work in multi-objective optimization which deals with simultaneously optimizing two or more (often conflicting) objective functions subject to some constraints[4] (Steuer, 1986; Deb, 2002). There are two possibilities when combining the MEGM and NCA objective functions under a multi-objective function framework. First, we can create a single aggregate function by combining all the objective functions[5]. A simple combination strategy is a linearly weighted sum of the objective functions. The weights of the combination will have to be specified. However, it should be noted that the final solution obtained will be highly dependent on the weights. For example, the objective functions of MEGM and NCA are combined as:

$$\mathcal{E}_L = \sum_{i=1}^{N} \left[\gamma_1 \left(y_i - \exp\left(\frac{-\|L\vec{x}_i - L\vec{x}_j\|_2^2}{2\sigma^2} \right) \right) - \gamma_2 \left(\frac{\exp(-\|L\vec{x}_i - L\vec{x}_j\|^2)}{\sum_{k \neq i} \exp(-\|L\vec{x}_i - L\vec{x}_k\|^2)} \right) \right] \tag{4.18}$$

[4]It should be noted that in multi-objective optimization problems, there is not a single solution that simultaneously minimizes each objective function. The solution to multi-objective function is generally a set of Pareto points. Pareto solutions are those for which improvement in one objective function can only be achieved with the worsening of at least one other objective function.

[5]This method goes by the name of AOF (Aggregate Objective Function).

Note that we are minimizing $\mathcal{E}$ by finding an optimal matrix L, and γ_1 and γ_2 are the weights assigned to the objective functions of MEGM and NCA respectively. Since the NCA objective function is supposed to be maximized, we are minimizing the negative of it in equation 4.18. Any method for solving a single objective function can be used. Obviously, one could investigate gradient-based methods to optimize for L in equation 4.18, that is, a simple gradient descent based optimization strategy as employed in MEGM could be used.

The second possibility when combining the MEGM and NCA objective functions is to use genetic algorithms[6]. For example, multi-objective optimization methods such as the non-dominated sorting genetic algorithm (NSGA-II) (Deb et al., 2002) and the strength Pareto evolutionary approach two (SPEA-2) (Coello et al., 2007) have been proven to be very effective. This points to an interesting new direction for learning a data-dependent distance metric.

[6]This strategy goes by the name of multi-objective optimization using evolutionary algorithms (MOEA).

Chapter 5

Naive/Semi-naive Metric Learning

This chapter deals with *naive* and *semi-naive* metric learning. As discussed in section 3.1, naive and semi-naive metric learning actually result in feature selection and feature weighting. This chapter provides an alternative perspective on the current methods of feature selection and feature weighting by viewing them as metric learning algorithms. Two new metric learning algorithms are proposed. The first proposed algorithm is a generic metric learning algorithm (GML) that can incorporate any feature weighting criteria and can be used for naive or semi-naive metric learning. The second proposed algorithm is a modification of the MEGM complete metric learning algorithm proposed in chapter 4. The algorithm is modified for semi-naive metric learning. The list of algorithms proposed in this chapter is given in table 5.1.

5.1 Related Work

As discussed in section 4.1, Lowe (1995) viewed feature selection as metric learning and proposed an algorithm for semi-naive metric learning. The proposed semi-naive metric learning algorithm in this chapter (algorithm MEGM-SNML) is similar to the one proposed by Lowe, but with a few small differences. Lowe's (1995) method seems to be over-parameterized. Therefore, the algorithm proposed in this chapter is a simpler version of Lowe's method. Modifying complete metric learning algorithms for semi-naive metric learning is common in metric learning research. For example, Xing et al. (2002) proposed a semi-naive metric learning algorithm by modifying their complete metric learning algorithm (section 4.1).

The feature selection techniques can be categorized into three forms: filter, wrapper and embedded.

Algorithms	Description
GML	A general naive and semi-naive metric learning.
MEGM-SNML	A modification of algorithm 1 in section 4.2 for semi-naive metric learning.

Table 5.1: List of Algorithms proposed in chapter 5

- The filter methods of feature selection are applied as a pre-processing step (Press et al., 1988; Robnik-Sikonja and Kononenko, 2003; Kononenko, 1995). That is, the relevance of each feature is determined before any classification algorithm is applied.
- The wrapper-based methods for feature selection score features based on the performance of a classifier trained using the learning algorithm with those features (Kohavi and John, 1997; Kohavi and Sommerfield, 1995). In other words, the actual output of the classifier is used to tune an index that gauges the relevance of each feature. Once the relevance of each feature is determined, any classification algorithm can be used for prediction. Note that the algorithm used for prediction can be different from the one used to measure the feature relevance.
- Like wrapper methods, embedded methods for feature selection rank features based on the performance of a classifier trained using some learning algorithm with those features (Guyon et al., 2002; LeCun et al., 1990; Chapelle et al., 2002; MacKay, 1994). Unlike wrapper methods, however, the learning algorithm used for measuring the relevance of the feature is also used for classification.

As feature selection is actually naive or semi-naive metric learning, we can also categorize naive/semi-naive metric learning methods as either filter, wrapper or embedded. Also, complete metric learning algorithms (from chapter 4) can be viewed as either filter, wrapper or embedded. For example, the MEGM algorithm can be viewed as an embedded method as the metric is tuned based on the score of a k-nearest neighbor classifier and the actual classification is also carried out by k-nearest neighbor classifier.

5.2 Naive and Semi-naive Metric Learning

As discussed in section 3.1, both naive and semi-naive metric learning correspond to learning the diagonal elements of the distance matrix in a distance measurement framework. But there is a difference between the two. Naive metric learning learns the diagonal terms by measuring the relevance of each individual feature separately. The features are assumed to be independent from each other. On the other hand, semi-naive metric learning measures the relevance of an individual feature in combination with the other features.

Individual feature relevance analysis has been studied extensively in statistics and machine learning. Different techniques have been proposed that assign scores to features based on their individual relevances (Rodgers and Nicewander, 1988; Peng et al., 2005; Cover and Thomas, 2006). Such feature ranking techniques are relatively fast and efficient. However, these techniques rely on the naive assumption of feature independence, which is not always true (Guyon and Elisseeff, 2003; Guyon et al., 2004). Although these formulations have been named 'naive' and 'semi-naive', it should be noted that such an optimistic assumption of feature independence should not be considered a major drawback. There exists classification techniques in machine learning which make such an assumption of feature independence and perform well. For example, Naive-Bayes[1] classifiers have been

[1]A two class posterior probability function in terms of the likelihood function and the prior probability is given in equation B.2. The naive Bayes classifier assumes that the likelihood function in equation B.2

shown to outperform other sophisticated alternatives in some cases (Huang et al., 2003). In the following, some common individual feature relevance measures are discussed.

Correlation Coefficient: The Pearson correlation coefficient is a well known relevance measuring index for individual features (Rodgers and Nicewander, 1988). It can be used for regression problems and binary classification problems. For multi-class classification problems one can use the closely related Fisher coefficient or normalize the class labels to be used in regression settings. It is defined as the covariance of two variables divided by the product of their standard deviation. For example, the Pearson coefficient for feature p is defined as:

$$c_p = \frac{|\sum_{n=1}^{N}(x_{np} - \bar{x}_p)(y_n - \bar{y})|}{\sqrt{\sum_{n=1}^{N}(x_{np} - \bar{x}_p)^2 \sum_{n=1}^{N}(y_n - \bar{y})^2}} \tag{5.1}$$

where $\bar{x}_p$ denotes the average value of the data points across the p'th feature and N is the number of training data.

Mutual Information: In probability and information theory, mutual information measures the mutual dependence of two random variables (Cover and Thomas, 2006; Peng et al., 2005). We can define the mutual information between two discrete random variables as:

$$I(X, Y) = \sum_{\vec{y} \in Y} \sum_{\vec{x} \in X} p(\vec{x}, \vec{y}) \log \left(\frac{p(\vec{x}, \vec{y})}{p(\vec{x})p(\vec{y})} \right) \tag{5.2}$$

The relevance of a feature p can be measured by calculating the mutual information between X_p (that is all data points across the p'th feature) and Y (class labels).

Both the Pearson correlation coefficient and mutual information are useful measures of the relevance of an individual feature. There is also a need to determine the relevance of an individual feature in combination with the others. This corresponds to the case of semi-naive metric learning. One justification of such strategy is that features that are individually irrelevant may become relevant when used in combination with other features. In the following, a simple but extremely effective feature relevance index is discussed which measures the relevance of an individual feature in combination with the other features.

Relief is an excellent technique for ranking individual features according to their relevances in combination with the others (Kira and Rendell, 1992). The key idea behind Relief is to iteratively estimate the relevance of a feature based on its ability to discriminate between neighboring data points. In each iteration, a data point $\vec{x}$ is randomly selected

can be written as:

$$p_j(X) = \prod_{k=1}^{P} p_{jk}(X_k)$$

where j is the class index, k is the index running over all features and P is the total number of features. Though this assumption is not necessarily true, but it simplifies the estimation process manifold. For example, the individual class probability densities can be estimated by separately using one-dimensional kernel density estimates.

along with its two nearest neighbors, one from the same class as $\vec{x}$ (referred to as nearest-hit (NH)) and another from the opposite class (referred to as nearest-miss (NM)). In each iteration, the relevance of the p-th feature is updated as:

$$c_p = c_p + |x_p - [NM(\vec{x})]_p| + |x_p - [NH(\vec{x})]_p| \tag{5.3}$$

Relief originally was designed for two class problems (Kira and Rendell, 1992). It is extended to deal with multi-class problems, noise and missing data in Kononenko (1994). Numerous other modifications of the Relief algorithm have been proposed. For example, Yijun and Jian (2006) proposed an iterative version of Relief. Also, the Simba algorithm which is a modification of Relief where at each iteration the features score used in the previous iterations is used to calculate the nearest-hit and nearest-miss, was proposed in Bachrach et al. (2004).

An outline of a general metric learning algorithm that can be used for naive and semi-naive metric learning is given in algorithm 2, where the learned metric is used in a k-nearest neighbor framework. Algorithm 2 demonstrates that any feature relevance measure can be used to train a distance metric. Depending on the relevance measure used, the algorithm can be categorized as either a naive or a semi-naive metric learning method.

Algorithm 2 GML: Generic algorithm for naive/semi-naive metric learning.

```
Require:
  - $\vec{x}_0$: Testing data.
  - $\{x_n, y_n\}_{n=1}^N$: Training data.
  - L: $P$-dimensional unit matrix.

  for each feature $p$ do
    - Calculate the relevance score $c_p$ (either from equation 5.1, 5.2 or 5.3).
    - Set $L(p,p) = c_p$.
  end for
  - Use matrix $L$ for 1-nearest neighbor classification.
```

5.2.1 MEGM-based Semi-naive Metric Learning

The MEGM metric learning algorithm can be used for semi-naive metric learning. That is, rather than learning the diagonal and off-diagonal terms of the matrix L in section 4.2, the algorithm can be easily modified to learn only its diagonal terms. The MEGM based semi-naive metric learning algorithm (MEGM-SNML) is proposed in this section. Since we are interested in the diagonal form of the matrix L, we can replace it by the vector $\vec{w}$ which defines the weight associated with each feature or the diagonal element of the matrix L. Therefore, we can replace equation 4.10 in section 4.2 with:

$$W_j = \exp\left(\frac{-\sum_{p=1}^{P} w_p^2 (x_{ip} - x_{jp})^2}{2\sigma^2}\right) \tag{5.4}$$

where w_p is the weight associated with the feature p and x_{ip} denotes the value of the p-th feature of $\vec{x}_i$. As can be seen from equation 5.4, W_j is the vote cast by the neighbors based on their distances from the query point. Based on equations 4.7, 4.8 and 5.4, we can write the gradient of MSE ($\mathcal{E}$) in equation 4.7 with respect to $\vec{w}$ as:

$$\frac{\partial \mathcal{E}}{\partial \vec{w}} = \sum_{i=1}^{N} (y_i - \hat{y}_i) \left(- \frac{\sum_j (y_j - \hat{y}_j) \frac{\partial W_j}{\partial \vec{w}}}{\sum_j W_j} \right) \tag{5.5}$$

The discussion from section 4.2 related to the size of the Gaussian kernel (σ in equation 5.4) centered at each query point is also relevant here. The partial derivative $\frac{\partial W_j}{\partial \vec{w}}$ can be calculated the same way as in equation 4.13 and will take the form:

$$\frac{\partial W_j}{\partial w_p} = -W_j \left(\frac{w_p (x_{ip} - x_{jp})^2}{\sigma^2} \right) \tag{5.6}$$

Combining equations 5.5 and 5.6 we can write the gradient of $\mathcal{E}$ with respect to vector $\vec{w}$ as:

$$\frac{\partial \mathcal{E}}{\partial \vec{w}} = \sum_{i=1}^{N} (y_i - \hat{y}_i) \frac{1}{\sum_j W_j} \sum_j (y_j - \hat{y}_j) W_j \left(\frac{\sum_{p=1}^{P} w_p^2 (x_p - x_{pj})^2}{\sigma^2} \right) \tag{5.7}$$

The MEGM-SNML algorithms employs the same gradient-based optimization strategy of MEGM algorithm as described in section 4.2. An outline of the MEGM-SNML algorithm is given in algorithm 3.

Algorithm 3 MEGM-SNML: Modification of MEGM (algorithm 1) for semi-naive metric learning.

```
Require:
  - $x_0$: Testing data.
  - $\{x_n, y_n\}_{n=1}^{N}$: Training data.
  - $k$: Number of nearest neighbors, $k$ is set to be floor(log2($N$)))
  - $\vec{w}$: $P$-dimensional unit vector.
  - $L$: $P$-dimensional unit matrix.

  - Calculate $\vec{w}$ by using conjugate gradient method as described in the text (gradient of
    the objective function is given in equation 5.7).
  - Update matrix $L$ such that $L(p, p) = w_p$.
  - Use matrix $L$ for 1-nearest neighbor classification.
```

5.3 Experimental Results

In this section, the performance of naive and semi-naive metric learning on synthetic data sets is compared. For naive metric learning, the generic metric learning algorithm (algorithm 2) with the correlation coefficient as the feature relevance measure is used (the results are labeled as Corrcoef-NML in the performance graphs). For semi-naive metric learning, the MEGM-SNML algorithm (algorithm 3) is used (the results are labeled as MEGM-SNML in the performance graphs). The data is generated from the following three functions:

- **function 1:** $y = x_1^2 + \epsilon$.
- **function 2:** $y = \sin(2\pi x_1 + \pi/2) + \epsilon$.
- **function 3:** $y = \sin(2\pi x_1)\sin(2\pi x_2) + \epsilon$.

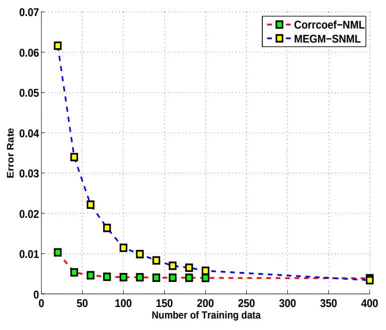
where ϵ denotes Gaussian random noise with a mean of zero and variance of five. 1000 data points were generated from each function. The data was embedded in 49 dimensions of uniform random noise for the first two functions and 48 dimensions of uniform random noise for the third function. Hence each data point was represented as a 50-dimensional feature-vector. It can be seen that that functions 1 and 2 depends on the first feature only whereas function 3 depends on the first and second feature.

The performance of naive and semi-naive metric learning on these synthetic data sets is shown in figure 5.1 and 5.2. Figure 5.1 shows the effect of learning a naive and semi-naive metric on the MSE of a k-nearest neighbor classifier. The results are shown for varying numbers of training data. It can be seen that both naive and semi-naive metric learning result in significant reduction of MSE for function 1 as the number of training data is increased. But for function 2 and function 3, naive metric learning algorithm did not perform well (in fact it failed miserably) and MSE is quite high even for large numbers of training data (figures 5.1(b) and 5.1(c)). Semi-naive metric learning, on the other hand, resulted in a significant decrease of MSE for both functions 2 and 3 as the number of training data was increased (figures 5.1(b) and 5.1(c)).

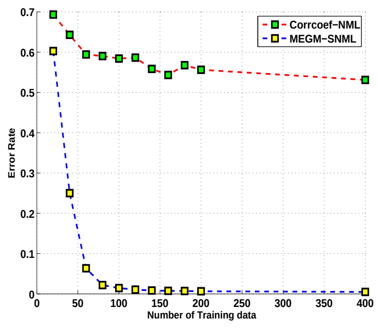
The feature selection performance of the naive and semi-naive metric learning algorithms is shown in figure 5.2. The bar graphs denotes the average correctness rate over 100 experiments (an experiment is successful if the metric learning algorithm selects the most relevant features). The result follows the trend from figure 5.1. Both the naive and semi-naive metric learning algorithms performed well for function 1. The naive metric learning algorithm again failed completely in selecting the relevant feature for the data generated from functions 2 and 3 (figure 5.2(b) and 5.2(c)). The performance of semi-naive metric learning (for feature selection) improved significantly as the number of training data was increased (figures 5.2(b) and 5.2(c)).

5.3.1 UCIML Databases

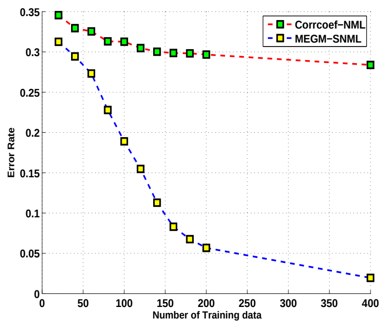
In this section, the performance of various naive and semi-naive metric learning methods on the UCIML databases from section 4.3.1 is compared. The details of these databases are given in table F.1. The error rate of each method is obtained using 40 rounds of 2-fold cross-validation. Prior to training, all features were normalized to have zero mean and unit variance. The classification performance in terms of the error rates of each of the following methods is shown in figures 5.3 and 5.4:



(a)



(b)

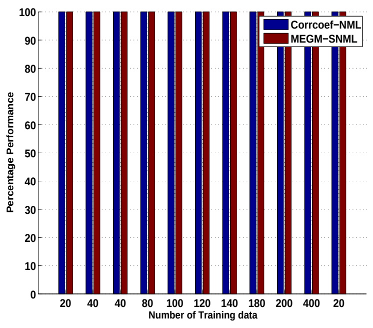


(c)

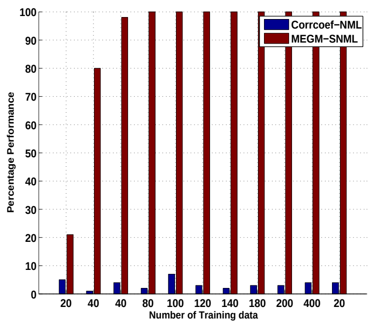
Figure 5.1: Comparison of the classification performance of naive and semi-naive metric learning algorithm, in terms of the error rates, on the synthetic data, The change in MSE of a k-nearest neighbor classifier trained for the data generated from function 1 (figure 5.1(a)), function 2 (figure 5.1(a)), and function 3 (figure 5.1(a)) by varying the number of training data is shown.

- **KNN:** A simple 1-nearest neighbor classifier with the Euclidean distance.
- **Corrcoef:** A 1-nearest neighbor classifier based on a metric tuned through the GML algorithm (algorithm 2) with the correlation coefficient (equation 5.1) as the relevance index.
- **Infogain:** A 1-nearest neighbor classifier based on a metric tuned through the GML algorithm (algorithm 2) with the mutual information measure (equation 5.2) as the relevance index.
- **Relief:** A 1-nearest neighbor classifier based on a metric tuned through the GML algorithm (algorithm 2) with relevance score based on the Relief algorithm for feature selection (equation 5.3).
- **MEGM-SNML:** A 1-nearest neighbor classifier based on a metric tuned through the MEGM-SNML algorithm (algorithm 3).

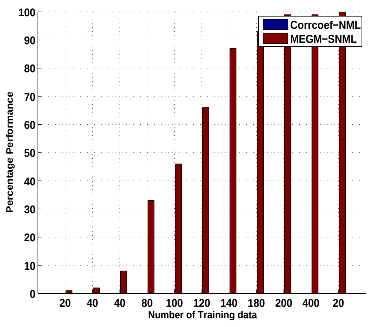
As can be seen from figures 5.3 and 5.4, the MEGM-SNML algorithm performed best on ten out of 19 UCIML databases. Corrcoef (naive), Infogain (naive) and Relief (semi-naive) performed best on two databases each. No method resulted in any performance gain of the KNN classifier on three databases (*hepatitis*, *monks2*, *liver-disorder*). It should be noted



(a)



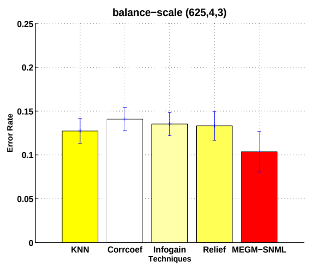
(b)



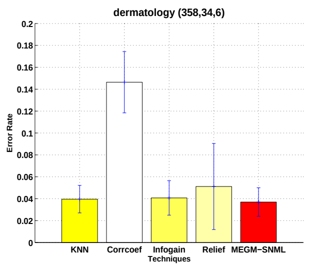
(c)

Figure 5.2: Comparison of the feature selection capability of naive and semi-naive metric learning algorithm on the synthetic data, Figure 5.2(a): function 1, Figure 5.2(b): function 2, Figure 5.2(b): function 3.

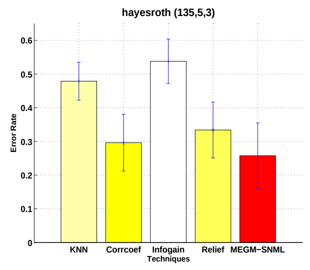
that both naive and semi-naive based metric learning methods resulted in performance improvement of the k-nearest neighbor classification on 16 out of 19 databases, even though both forms of metric learning assume feature independence. The results in figures 5.3 and 5.4 suggest that perhaps the features in most data sets are independent from each other. This may be the likely reason why Naive Bayes classifiers have performed well despite their assumption of feature independence.



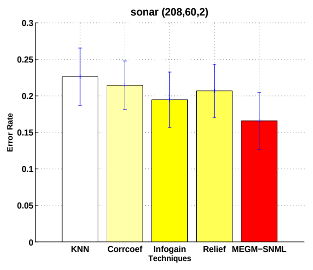
(a)



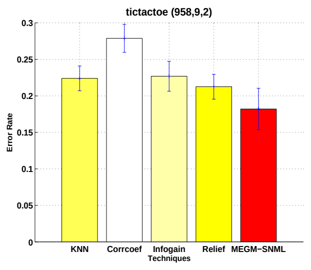
(b)



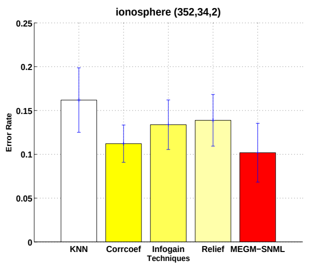
(c)



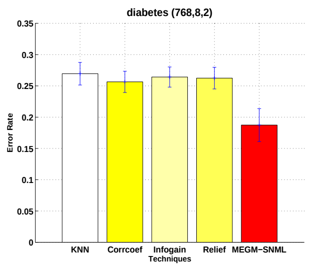
(d)



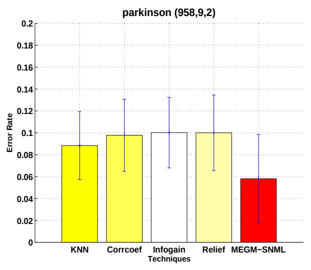
(e)



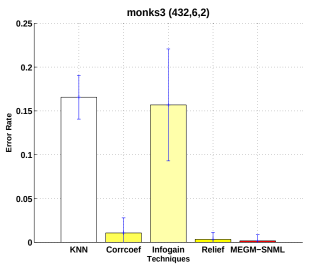
(f)



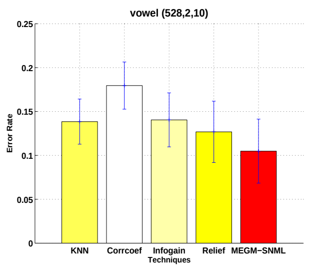
(g)



(h)

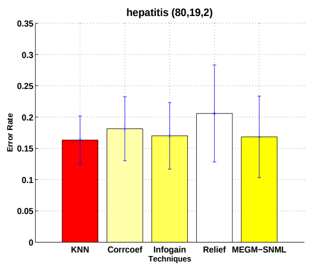


(i)

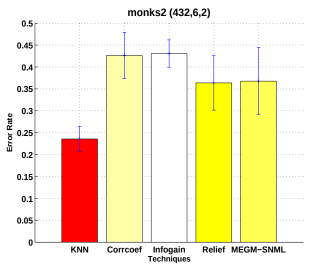


(j)

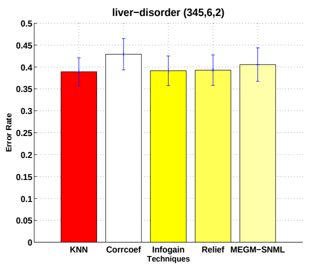
Figure 5.3: Error rate comparison of KNN, Corrcoef, Infogain, Relief and MEGM-SNML on various UCIML databases (Continued in figure 5.4).



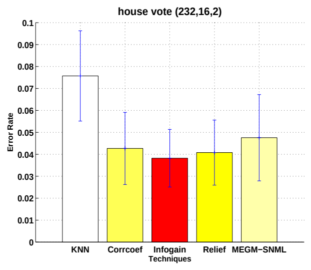
(a)



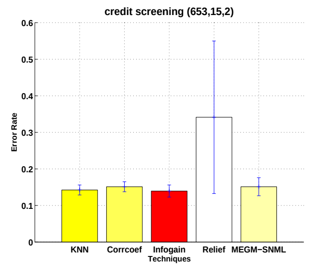
(b)



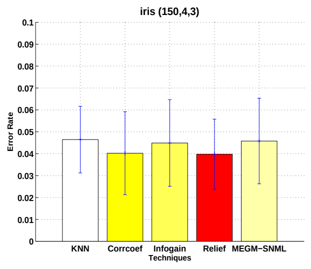
(c)



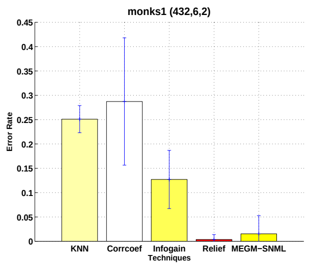
(d)



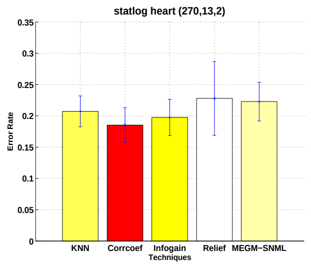
(e)



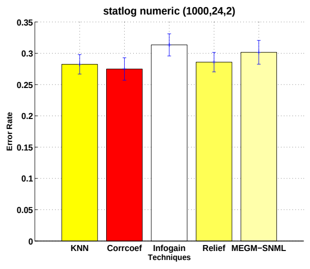
(f)



(g)



(h)



(i)

Figure 5.4: Error rate comparison of KNN, Corrcoef, Infogain, Relief and MEGM-SNML on various UCIML databases (Continued from figure 5.3).

Chapter 6

Local Adaptive Metric Learning

This chapter introduces a local adaptive metric learning method and analyzes naive, semi-naive and complete metric learning methods in the local setting. In this chapter, two new algorithms for local adaptive metric learning will be proposed which are based on the application of naive metric learning in the neighborhood of a query point. The proposed algorithms will be explained in section 6.2 and the experimental evaluation of the proposed algorithms will be provided in section 6.3. Current work in local adaptive metric learning determines the relevance of each feature at a query point using some numerical index. This index gauges the relevancy of the feature and controls the form of neighborhood around the query point. The proposed index in this work is inspired by work in the area of boosting (section 10.3), where at each iteration the data is partitioned across the most discriminative dimension (Schapire and Singer, 1998). The index is based on the logit-transform of the class probability estimate. Using this index the dimension is selected that is the most discriminative. This is similar to 'boosting classifiers' where at each iteration a feature is selected with which the data can be classified best (Friedman et al., 2000). The list of algorithms proposed in this chapter is given in table 6.1.

In the following discussion, the query point will be denoted by $\vec{x}_0$ and training points by $\vec{x}_n$, where $n = [1, \ldots, N]$, N is the number of training data, P denotes the number of features and $\vec{x}_{0p}$ and $\vec{x}_{np}$ denote the value at the p-th feature of $\vec{x}_0$ and $\vec{x}_n$ data points respectively.

6.1 Related Work

An early inspirational work in the area of local adaptive metric learning is by Friedman (1994) in which he discussed in detail various issues that need to be addressed for nearest

Algorithms	**Description**
BoostML1	An iterative adaptive metric learning algorithm which divides an input space at each step based on the most relevant feature.
BoostML2	Modification of BoostML1, divides an input space at each step based on the learned metric.

Table 6.1: List of Algorithms proposed in chapter 6

neighbor classification in high dimensions. Local metric learning algorithms are proposed for reducing bias in high dimensions. The two proposed algorithms in Friedman (1994), Machete and Scythe, were the major inspiration for the algorithms proposed in this chapter. Both Machete and Scythe are based on the learning of a local metric around the query point by recursively splitting the neighborhood across the feature which is most relevant. The relevance of feature p at query point $\vec{x}_0$ is calculated as:

$$\begin{aligned} I_p^2(\vec{x}_0) &= \sum_{j=1}^{C}(E[y_j] - E[y_j|\vec{x}_{0p} = \vec{x}_{np}])^2 \\ &= \sum_{j=1}^{C}\left(\frac{1}{C} - E[y_j|\vec{x}_{0p} = \vec{x}_{np}]\right)^2 \end{aligned} \qquad (6.1)$$

Equation 6.1 represents a measure of the impurity of the class labels conditioned on the term $\vec{x}_{0p} = \vec{x}_{np}$. It is also related to the well known 'Gini' (entropy-based) index. The main difference of the proposed algorithms in this chapter from Friedman (1994) is the feature relevance determination. In the proposed methods a feature is considered more relevant if it is more discriminatory, whereas in Friedman (1994) a feature is considered relevant if the class label varies the most.

Hastie and Tibshirani (1996) have proposed an adaptive metric learning algorithm based on linear discriminant analysis (LDA). A distance metric is computed as a product of properly weighted within (W) and between sum-of-square (B) $P \times P$ dimensional matrices. The proposed algorithm 'Discriminative Adaptive Nearest Neighbor' (DANN) estimates the W and B matrices locally and uses them to form a local distance metric in an iterative fashion. The matrix A takes the following form:

$$A = W^{-1/2}(W^{-1/2}BW^{-1/2} + \epsilon I)W^{-1/2} \qquad (6.2)$$

where I is the identity matrix and ϵ is a tuning parameter. Though sound in theory, the method has limitations. The major limitation is that in high dimensions we may not have sufficient data to fill in the $P \times P$ within class sum-of-square matrix (due to sparsity). Hastie and Tibshirani (1996) also found it more effective to estimate only the diagonal terms of within class sum-of-square matrix and assume that the off-diagonal terms are zero. This is especially true if the dimensionality of the input space is large, as there will be insufficient data locally to estimate the $\Theta(P^2)$ elements of the within class sum-of-square matrix.

Domeniconi et al. (2000) proposed an adaptive metric learning method based on Chi-squared distance analysis. A weighted Chi-squared distance between two points $\vec{x}_0$ and $\vec{x}_1$ in terms of their class posterior probabilities is defined as:

$$d(\vec{x}_0, \vec{x}_1) = \sum_{j=1}^{C}\left(\frac{[P(j|\vec{x}_1) - P(j|\vec{x}_0)]^2}{P(j|\vec{x}_0)}\right) \qquad (6.3)$$

Note, $P(j|\vec{x}) = P(j|\vec{x}_0)$ suggests that $P(j|\vec{x})$ can be approximated at $\vec{x}_0$. Their proposed algorithm ADAMENN measures the relevance of the feature by replacing $P(j|\vec{x}_0)$ in equation 6.3 in the following way:

$$I_p(\vec{x}_0) \quad = \quad \sum_{j=1}^{C} \left(\frac{[P(j|\vec{x}) - P(j|\vec{x}_{0p} = \vec{x}_{np})]^2}{P(j|P(j|\vec{x}_{0p} = \vec{x}_{np})} \right) \tag{6.4}$$

where I_p represents the relevance of feature p at query point $\vec{x}_0$.

Some other notable techniques for local adaptive metric learning are Peng et al. (2001); Janusz (2005); Domenciconi et al. (2005). The algorithm proposed in Domenciconi et al. (2005) used an SVM classifier for determining feature relevance. The decision function of an SVM is used to determine the most discriminant direction in the neighborhood of a query point. Such a direction provides a local feature weighting scheme. For example, if the decision boundary traced by the SVM is $f(\vec{x}) = 0$ in the input space, the gradient vector $\vec{g}_i = \nabla_i f$ computed at any point i on the decision boundary f points to the direction perpendicular to the decision boundary in the input space at point i. The vector $\vec{g}_i$ identifies the orientation in the input space along which the projected training data are all well separated in the neighborhood around i. Therefore, the orientation given by $\vec{g}_i$, and any orientation close to it, carries highly discriminant information for classification. This information can be used to define a local measure of feature relevance. A similar but slightly modified method for metric learning based on an SVM is proposed in Peng et al. (2001).

Local adaptive metric learning methods are very close in nature to query-sensitive metric learning (Zhan et al., 2009; Zhou and Dai, 2006). Zhou and Dai (2006) have investigated a query-sensitive metric for content-based image retrieval.

6.2 Local Adaptive Metric Learning

In this section, two proposed algorithms, BoostML1 and BoostML2, for local adaptive metric learning are described. Measuring feature relevance is actually the key to designing efficient local, as well as global, metric learning algorithms. Before describing the details of the proposed algorithms, the relevance measure used by the proposed algorithms is described.

6.2.1 Feature Relevance

For any query point $\vec{x}_0$, the relevance (c) of the feature p is estimated as:

$$c_p(\vec{x}_0) = \frac{I_p(x_{0p})}{\sum_{p=1}^{P} I_p(x_{0p})} \tag{6.5}$$

where $I_p(\vec{x}_0)$ is defined as:

$$I_p(\vec{x}_0) = \sum_{c=1}^{C} \text{abs}\left(\frac{1}{2} \ln \left(\frac{P(c|x_{np} = x_{0p}) + \kappa}{P(c|x_{np} \neq x_{0p}) + \kappa} \right) \right) \tag{6.6}$$

The κ in equation 6.6 is a small quantity introduced for numerical tractability. A small I_p (close to zero) implies that there is an equal split of positive and negative training data points in the neighborhood of $\vec{x}_0$ across the feature p. A large value of I_p implies that one class dominates the other class across feature p, therefore, feature p is more relevant and should be given more weight.

The computation of probabilities $P(c|x_{np} = x_{0p})$ and $P(c|x_{np} \neq x_{0p})$ in equation 6.6 is not trivial, as we may not have sufficient data in the neighborhood of the query point to accurately define them. The probability $P(c|x_{np} = x_{0p})$ is computed as:

$$P(c|x_{np} = x_{0p}) = \frac{\sum_{\vec{x}_n \in N(\vec{x}_0)} 1(|x_{np} - x_{0p}| \leq \delta_p) 1(y_n = c)}{\sum_{\vec{x}_n \in N(\vec{x}_0)} 1(|x_{np} - x_{0p}| \leq \delta_p)} \tag{6.7}$$

A small neighborhood around query point $\vec{x}_0$ denoted by $N(\vec{x}_0)$ is defined and a value of δ_p is chosen to make sure that the neighborhood contains L points, such that:

$$\sum_{n=1}^{N} 1(|x_{np} - x_{0p}| \leq \delta_p) = L \tag{6.8}$$

In other words we look for at least L points that are close to the query point on feature p and compute the probabilities in equation 6.6 using these points. The output of feature relevance analysis is a $P \times P$ diagonal matrix whose terms are the estimated relevances of the features. Based on equation 6.6, we can specify a local distance metric at point $\vec{x}_0$ using matrix $A(\vec{x}_0)$, where:

$$A(\vec{x}_0) = \begin{pmatrix} c_1 & 0 & \cdots & 0 \\ 0 & c_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & c_p \end{pmatrix} \tag{6.9}$$

6.2.2 Details of the Algorithm

Given the training data $\{(\vec{x}_n, y_n)\}_{n=1}^{N}$, the goal is to estimate the label of the query point $\vec{x}_0$. The proposed BoostML1 algorithm starts by initializing the neighborhood of the query point to be the entire measurement space ($\mathcal{R}_0$). The relevance of each feature is determined in this neighborhood of K points (initially $K = N$) using equation 6.5. The feature having the highest relevance score is selected, that is:

$$p^*(\vec{x}_0) = \mathrm{argmax}_{1 \leq p \leq P}\ c_p(\vec{x}_0) \tag{6.10}$$

The neighborhood is divided into two based on the feature p^*. The extent of division is controlled by the parameter ζ (algorithm 4). It should be noted that different features can be selected for splitting the neighborhood for different query points $\vec{x}_0$ based on the relevance of the features at that location in the input measurement space. The split divides the input measurement space into two regions: $\mathcal{R}_1(\vec{x}_0)$, that contains the query point and the H training points that are closest to the query point $\vec{x}_0$ on the chosen feature p^*, and the other (complement) region $\mathcal{R}_2(\vec{x}_0)$ that contains $K - H$ points that are farthest from $\vec{x}_0$ across feature p^*. $\mathcal{R}_2(\vec{x}_0)$ is removed from further consideration. Thus the result of the split is just one region, $\mathcal{R}_1(\vec{x}_0)$. The above procedure is then applied recursively on region $\mathcal{R}_1(\vec{x}_0)$ until some k number of data points are left in $\mathcal{R}_1$. The neighborhood represented by these k points is used to determine the label of the query point $\vec{x}_0$. The method is named BoostML1 and its outline is given in algorithm 4. Refer to figure 6.1 for an illustration of BoostML1 algorithm.

Algorithm 4 BoostML1: Local adaptive metric learning algorithm.

```
Require:
  - $\vec{x}_0$: Testing data.
  - $\{\vec{x}_n, y_n\}_{n=1}^N$: Training data.
  - $k$ : Number of elements in the final neighborhood.
  - $\zeta$: Stepping size.
  - Initialize $K = N$ and $A$ as a $p$ dimensional unit matrix.
  - $N_K(\vec{x}_0)$ denotes neighborhood of $\vec{x}_0$ consisting of $K$ points.

  while flag do
    - Get Feature Relevance index $c_p(\vec{x}_0)$ at $\vec{x}_0$ (equation 6.5), update $A$ (equation 6.9).
    - Choose feature $r = \text{argmax}_p \, c(\vec{x}_0)$.
    - Modify neighborhood size parameter $K$, that is, $K = \zeta K$.
    - Update $N_K(\vec{x}_0)$ by finding $K$ data points in the neighborhood of $\vec{x}_0$ ($N_K(\vec{x}_0)$) across
    the feature $r$ only.
    if $N_K(\vec{x}_0) < k$ then
      flag = false.
    end if
  end while
  - Perform $k$-nearest neighbor classification in the final neighborhood of $\vec{x}_0$.
```

Algorithm 5 BoostML2: Local adaptive metric learning algorithm, variant of BoostML1

```
Require:
  - $\vec{x}_0$: Testing data.
  - $\{\vec{x}_n, y_n\}_{n=1}^{N}$: Training data.
  - $k$ : Number of elements in the final neighborhood.
  - $\zeta$: Stepping size.
  - Initialize $K = N$ and $A$ as a $p$ dimensional unit matrix.
  - $N_K(\vec{x}_0)$ denotes neighborhood of $\vec{x}_0$ consisting of $K$ points.

  while flag do
    - Get Feature Relevance index $c_p(\vec{x}_0)$ at $\vec{x}_0$ (equation 6.5), update $A$ (equation 6.9).
    - Modify neighborhood size parameter $K$, that is, $K = \zeta K$.
    - Update $N_K(\vec{x}_0)$ by finding all $K$ data points in the neighborhood of $\vec{x}_0$ ($N_K(\vec{x}_0)$)
    using the matrix $A$.
    if $N_K(\vec{x}_0) < k$ then
      flag = false.
    end if
  end while
  - Perform $k$-nearest neighbor classification in the final neighborhood of $\vec{x}_0$.
```

As can be seen in algorithm 4, the splitting procedure in BoostML1 is recursively applied until there are only k training observations left in the final neighborhood. At each step a region is split based on the feature (p^*) that is estimated to be the most relevant in terms of capturing the variation of the target functions within that region (equation 6.10). All diagonal terms of the matrix A (equation 6.9) are ignored except the one with the maximum value, which is retained to split the region at each step. This is a greedy approach which is not necessarily effective all the time. BoostML2 is a similar to BoostML1 but has a different splitting procedure, i.e. updating procedure for $N_K(\vec{x}_0)$. At every iteration it splits the region based on all the terms in the matrix A (equation 6.9) as computed in the current iteration. It will be shown in section 6.3 that BoostML2 is an improvement over BoostML1 (algorithm 4). An outline of BoostML2 algorithm is given in algorithm 5.

6.3 Experimental Results

In this section, the results of the proposed adaptive metric learning algorithms on UCIML databases are shown (Frank and Asuncion, 2010). The details of these databases are given in table F.1 and appendix F. Databases were selected such that the competing techniques perform best on at least one of the databases. To obtain the error rates, leave-one-out cross-validation was used for the *Iris*, *Ionoshphere*, *Dermatology*, *Echocardiogram* and *Heart* data sets, whereas ten rounds of 2-fold cross-validation were used for the *Credit*

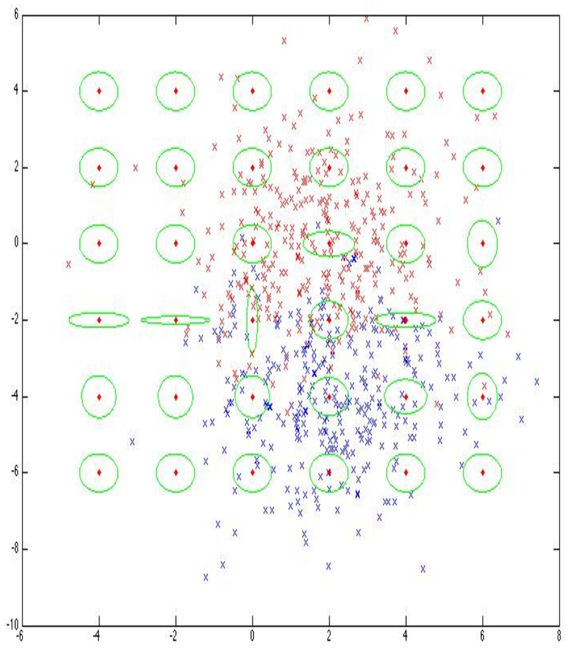


Figure 6.1: Demonstration of BoostML1 on synthetic 2-dimensional data. The classes are represented as red and blue crosses. Green ellipses show the learned distance metric at different points in the input measurement space.

and *Diabetes* databases[1]. Prior to training, all features were normalized to have zero mean and unit variance. The classification performance in terms of the error rates of each of the following methods is shown in figure 6.3.

- **KNN:** A simple k-nearest neighbor classifier with the Euclidean distance.
- **DANN:** Discriminative Adaptive Nearest Neighbor classifier as described in section 6.1 (Hastie and Tibshirani, 1996).
- **ADAMENN:** Adaptive metric nearest neighbor classification technique based on chi-squared analysis (Domeniconi et al., 2000).
- **Machete:** Recursive partitioning algorithm (Friedman, 1994).
- **Scythe** This is a generalization of the Machete algorithm in which features influence each split in proportion to their estimated local relevance, in contrast to the 'winner-takes-all' strategy of Machete (Friedman, 1994).
- **BoostML1:** Algorithm 4. The implementation details are described in the following discussion. The results are denoted as 'BML1' in the performance graphs.

[1]These experimental settings were inspired by Domeniconi et al. (2000); Friedman (1994) as the comparison with ADAMENN, Machete and Scythe metric learning algorithms was one of the motivation behind the proposed algorithms.

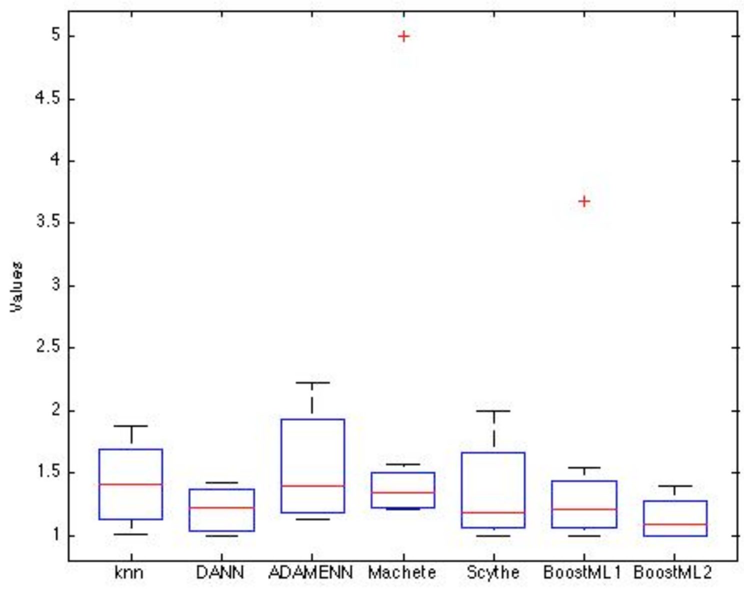
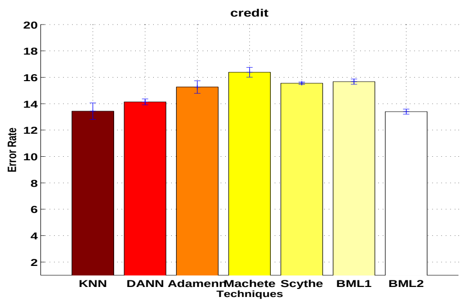


Figure 6.2: Boxplots depicting the robustness of KNN, DANN, ADAMENN, Machete, Scythe, BoostML1 and BoostML2 adaptive metric learning algorithms tested on eight UCIML databases.

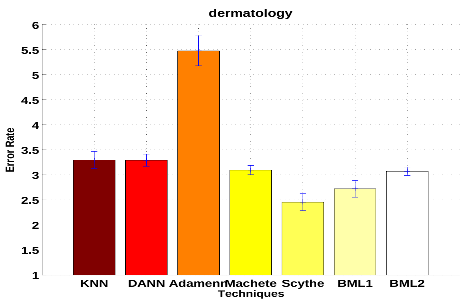
- **BoostML2:** Algorithm 5. Variant of BoostML1 as described in section 6.2. The results are denoted as 'BML2' in the performance graphs.

As can be seen from figure 6.3, both the BoostML1 and BoostML2 metric learning algorithms resulted in the improvement of k-nearest neighbor classification. This improvement comes at an extra cost. BoostML1 and BoostML2 have introduced two new tuning parameters, L and ζ. The value of L in equation 6.8 must be specified to compute the probabilities in equation 6.7. It can be seen that L controls the size of the neighborhood and, therefore, determines the bias-variance trade-off. In the experiments, it was found that L does not affect the classification performance provided it is neither too small nor too large. $L = 20$ was used for BoostML1 and BoostML2 in all experiments. There was not any significant improvement in the results for setting the value of L less than or greater than 20. The ζ parameter, which controls the splitting of the region at each step, is critical to the classification performance. A large value of ζ will result in a better performance, but at an increased computational cost. On the other hand, a small value of ζ will result in poorer performance but it will be faster. A tradeoff has to be achieved between computational cost and performance. $\zeta = 0.8$ was used in all the experiments.

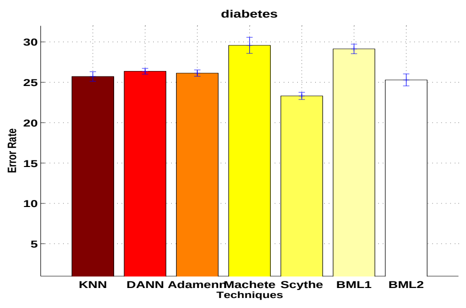
To compare the robustness of BoostML1 and BoostML2 with other local adaptive metric learning algorithms, the same procedure introduced in section 4.3.1 was used. The distribution b_m for each method over all eight databases is shown in terms of the boxplots in figure 6.2. As can be seen, BoostML2 turned out to be the most robust of all the methods, with DANN coming second. Machete and BoostML1 also performed well, with the exception of an outlier in both cases.



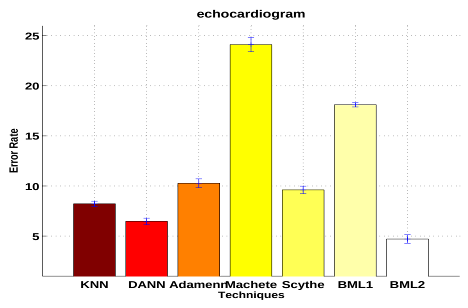
(a)



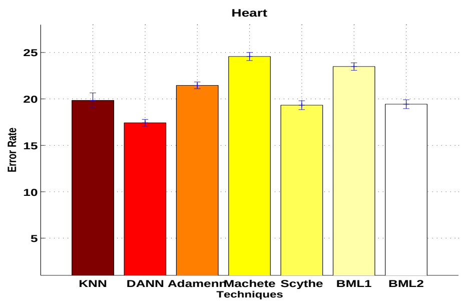
(b)



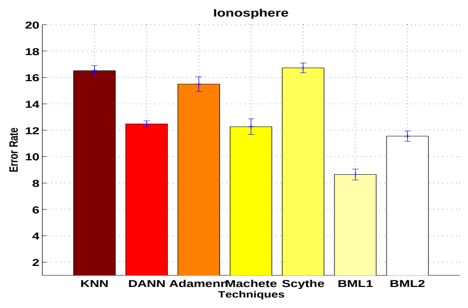
(c)



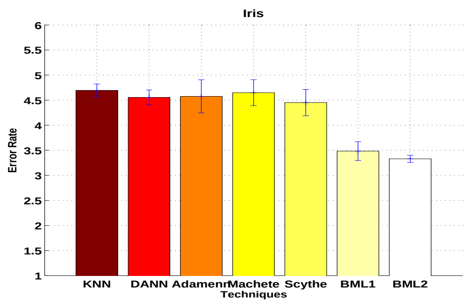
(d)



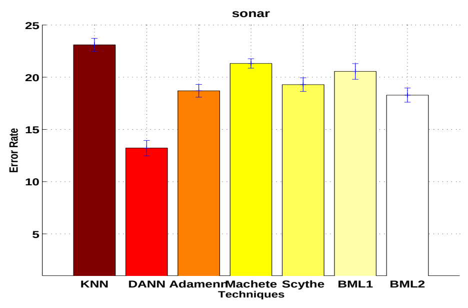
(e)



(f)



(g)



(h)

Figure 6.3: Error rate comparison of KNN, DANN, ADAMENN, Machete, Scythe, BoostML1 and BoostML2 on various UCIML databases. Figure 6.3(a): *credit screening*, Figure 6.3(b): *dermatology*, Figure 6.3(c): *diabetes*, Figure 6.3(d): *echocardiogram*, Figure 6.3(e): *statlog heart*, Figure 6.3(f): *ionosphere*, Figure 6.3(g): *Iris*, Figure 6.3(h): *sonar*.

Chapter 7

Global Adaptive SVM

As discussed in section 2.4.3, learning kernel parameters is actually the learning of a data-dependent distance metric. This chapter deals with the learning of kernel parameters in an SVM formulation by learning a data-dependent distance metric using the MEGM algorithm that was discussed in chapter 4. The related work on metric learning and kernel tuning in the context of SVM classifiers will be discussed in section 7.1. Two algorithms that aim to improve SVM classification performance by learning a data-dependent distance metric will be presented in section 7.2. The experimental evaluation of the proposed algorithms will be given in section 7.3, where the algorithms will be tested on various UCIML, digit and face databases. The list of algorithms proposed in this chapter is given in table 7.1.

7.1 Related Work

Metric learning (or kernel tuning) algorithms proposed in the SVM framework can be categorized as either naive or semi-naive metric learning methods. To the best of our knowledge, complete metric learning methods have not been investigated in the SVM framework. In the following some well known techniques for tuning SVM kernel parameters are explained.

From equation D.16 we can write the decision function of an SVM classifier as:

$$y(\vec{x}) = \text{sign}\left(\sum_{i}^{N} \alpha_i^* y_i k_A(\vec{x}_i, \vec{x}) + \beta_0^*\right) \tag{7.1}$$

Algorithms	Description
GASVM1	Train a support vector machine classifier using a data-dependent distance metric (a metric is learned for all C classes).
GASVM2	Train a support vector machine classifier using a data-dependent distance metric (a separate metric is learned for each class).

Table 7.1: List of Algorithms proposed in chapter 7

Chapelle et al. (2002) proposed an algorithm for estimating SVM hyper-parameters, that is α and k_A in equation 7.1. The matrix A is assumed to be a diagonal matrix. The values of A and α are chosen such that the f in equation 2.24 is maximized by minimizing some model selection criterion, for example radius-margin bound, etc. The algorithm proposed by Chapelle et al. (2002) is:

1. Initialize A to some random value.
2. Find the α in equation 2.24 using the standard SVM algorithm, that is:
$$\alpha^* = \arg\max_{\alpha} f(\alpha, A) \tag{7.2}$$
3. Using the gradient-based model selection criterion of Bengio (2000) for optimizing the model parameters, the matrix A is updated such that the radius-margin bound (T) is minimized.
$$A^* = \arg\min_{A} T(\alpha^*, A) \tag{7.3}$$
4. Go to step 2 until convergence.

Chapelle et al.'s (2002) algorithm for tuning the parameters of an anisotropic kernel is applied to the classification of high resolution computed tomography images in Shamsheyeva and Sowmya (2004). Inspired by Chapelle et al. (2002), Keerthi (2001) proposed some techniques for efficient tuning of SVM hyper-parameters using iterative algorithms. The radius/margin bound is taken as an index to be minimized and gradient-based methods from Chapelle et al. (2002) are used. The algorithm proposed in Keerthi (2001) is also modified for feature selection by Chapelle and Keerthi (2008).

Amari and Wu (July 1999) proposed the idea of learning a data-dependent kernel based on the Riemannian geometrical structure induced in the input space. The idea is to enlarge the spatial resolution around the separating hyperplane (decision boundary) by modifying the kernel such that the separability between the two classes is increased. Since, the decision boundary is unknown, their algorithm works by computing an SVM decision boundary first by using an initial kernel $k(.,.)$. The kernel is then modified as:

$$\tilde{k}(\vec{x}, \vec{x}_i) \quad = \quad c(\vec{x})c(\vec{x}_i)k(\vec{x}, \vec{x}_i) \tag{7.4}$$

where $c(\vec{x})$ is a positive scalar function and represents the conformal transformation of the kernel $k(\vec{x}, \vec{x}_i)$ by factor $c(\vec{x})$. It is defined as:

$$c(\vec{x}) \quad = \quad \sum_{i \in \text{SV}} \alpha_i \text{exp}\left(\frac{-\|\vec{x} - \vec{x}_i\|^2}{2\tau^2}\right) \tag{7.5}$$

where τ is a tunable parameter and SV represents all support vectors. A second SVM classifier is trained with the modified kernel $\tilde{k}(\vec{x}, \vec{x}_i)$. This algorithm inspired the local adaptive metric learning algorithm by Domenciconi et al. (2005) discussed in chapter 6.

An approach based on learning the parameters of the Gaussian kernel by maximizing the kernel polarization using a gradient-based method is proposed in Xu and Liu (2009). The kernel polarization is defined as the Frobenius inner product between the kernel and the ideal kernel matrix (yy^T) which compares the similarity of the kernel with the ideal kernel. Parameters of the Gaussian kernel are sought that result in the maximization of the kernel polarization.

Even though a lot of methods have been proposed to tune kernel parameters for SVM, cross-validation is still the most popular approach. It may not be computationally efficient, but it is effective in terms of the classification performance. For example the proposed method of kernel tuning by Chapelle et al. (2002) was tested on five UCIML databases and compared with the cross-validation approach. There was not a great gain in the classification performance reported on the UCIML databases and the performance was quite similar to cross-validation. The proposed method, however, has advantage of being computationally efficient. Similarly the method in Amari and Wu (July 1999) was tested on only two UCIML databases and again the results were comparable to cross-validation. One can say that the cross-validation method is state of the art when it comes to learning the kernel parameters in the SVM framework.

7.2 Global Adaptive SVM

In this section, algorithms for training an SVM classifier with a data-dependent distance metric are proposed. A complete metric learning algorithm in the k-nearest neighbor framework (algorithm 1) is used for learning the parameters of the Gaussian kernel. The outline of the proposed algorithms 'globally adaptive SVM' (GASVM1 and GASVM2) is given in algorithms 6 and 7. Although the algorithms rely on MEGM to learn the kernel parameters, any naive, semi-naive or complete metric learning algorithm that gives a linear transformation of the data by learning a transformation matrix A can be used.

The proposed GASVM1 algorithm tunes the kernel by learning a data-dependent distance metric using MEGM before training an SVM classifier. Since an SVM is a binary classifier, C SVM classifiers need to be trained for C classes (with the one-versus-all strategy). Once the matrix A is learned, the kernel k_A is used by all C SVM classifiers when classifying a test point ($\vec{x}_0$). GASVM2 is a slight variant of GASVM1. Rather than learning one kernel for all the categories, GASVM2 learns a different kernel using MEGM for each category. The learned kernel k_{Ac} is saved with the classifier for category c. Whenever the classifier c is used to classify the test point ($\vec{x}_0$), it uses the learned kernel (metric) k_{Ac} to measure the similarity. As will be shown in section 7.3, in terms of the classification performance, both GASVM1 and GASVM2 perform equally well on different data sets. Since GASVM1 uses only one kernel for all classifiers, it is more computational efficient than GASVM2. It should be noted that, though other kernels can be incorporated in global adaptive SVM formulation, the scope of the work is limited to Gaussian kernels. Since Gaussian kernels have been shown to perform extremely well on a huge variety of data sets and are widely used, this should not be considered as a drawback (Cover and Thomas, 2006; Chapelle et al., 2002). However, in those cases where similarity is more efficiently measured by some other kernel, for example linear or polynomial, the use of global adaptive SVM algorithms may not be beneficial.

Algorithm 6 GASVM1: Train an SVM classifier using a data-dependent distance metric.

```
Require:
  - $\vec{x}_0$: Testing data.
  - $\{\vec{x}_n, y_n\}_{n=1}^N$: Training data.

  - Get a data-dependent distance metric (matrix $A$) using MEGM such that $A = L^T L$.
  for $c = 1, 2, \ldots C$ do
    - Train an SVM classifier for category $c$ using kernel:
          $k_A(\vec{x}_i, \vec{x}_j) = \exp\left(-\|L\vec{x}_i - L\vec{x}_j\|_2^2\right)$
  end for
  - Use $C$ SVM classifiers in one-versus-all way to classify $\vec{x}_0$ using the kernel $k_A$.
```

Algorithm 7 GASVM2: Train an SVM classifier using a data-dependent distance metric.

```
Require:
  - Testing data: $\vec{x}_0$.
  - Training data: $\{\vec{x}_n, y_n\}_{n=1}^N$.

  for $c = 1, 2, \ldots C$ do
    - Get a data-dependent distance metric (matrix $A$) using MEGM for category $c$ such
    that $A = L^T L$.
    - Train an SVM classifier for category $c$ using kernel:
          $k_{Ac}(\vec{x}_i, \vec{x}_j) = \exp\left(-\|L\vec{x}_i - L\vec{x}_j\|_2^2\right)$
  end for
  - Use $C$ SVM classifiers in one-versus-all way to classify $\vec{x}_0$ using the pool of kernels
  $\{k_{Ac}\}_{c=1}^C$.
```

7.3 Experimental Results

In this section, the performance of the proposed globally adaptive SVM algorithms is compared with other standard SVM and nearest neighbor formulations on various UCIML, face and digit databases. The details of UCIML databases is given in appendix F.

7.3.1 UCIML Databases

The number of data, features and classes for each UCIML database used is reported in table F.1. The error rate of each method is obtained using 40 rounds of 2-fold cross-validation. Prior to training, features in all the databases were normalized to have zero mean and unit variance. The classification performance in terms of the error rate of each of the following methods for different databases is shown in figures 7.1, 7.2, 7.3 and 7.4.

- **KNN:** Simple 1-nearest neighbor classification with the Euclidean distance.
- **SVM:** Standard SVM formulation that is, a multi-class SVM with a Gaussian kernel is used. The $\mathcal{C}$ parameter for the SVM is tuned through cross-validation (that is chosen from the set: $\{1, 10, 100, 1000\}$). The value of σ is set to be the average distance of k-nearest neighbors following the insight from Zhang, Marszalek, Lazebnik and Schmid (2006); Varma and Ray (2007). A one-versus-all strategy is employed for multi-class classification.
- **OSVM:** Similar to standard SVM, but both the $\mathcal{C}$ and σ parameters are optimized using cross-validation. This formulation is called optimized SVM (OSVM). The $\mathcal{C}$ and σ parameters are chosen from the sets: $\mathcal{C} = \{1, 10, 100, 1000\}$ and $\sigma = \{0.1, 0.5, 1, 2, 3, 5\}$ respectively.
- **GASVM1:** Global adaptive SVM algorithm (algorithm 6).
- **GASVM2:** Global adaptive SVM algorithm (algorithm 7).

The value of $\mathcal{C}$ for both GASVM1 and GASVM2 is not optimized and was set equal to ten in all experiments.

It can be seen from figures 7.1, 7.2, 7.3 and 7.4, that out of 21 UCIML databases, GASVM1 performed best on 11. On the other hand, GASVM2 performed best on four databases. OSVM and KNN performed best on three databases each. GASVM1 performance is very close to the standard and optimized SVM in most cases. These results are encouraging because, as mentioned before, SVM and OSVM requires the tuning of kernel parameters through an expensive cross-validation procedure, whereas the global adaptive SVM algorithms have no such tuning involved. It should be noted that the results obtained with GASVM2 have particularly high variance on most databases, for example *balance* and *hayesroth* (figures 7.1(a) and 7.1(c)). One likely reason for such high variance may have to do with its reliance on MEGM for learning the kernel parameters. Since GASVM2 learns a different kernel for each category, there are more chances that local minima will affect its performance. Although the global adaptive SVM methods performed well on a variety of databases, it can be seen from figure 7.3 that on the *ionosphere*, *monks2* and *satimage* databases, the optimized SVM (OSVM) outperformed global adaptive SVM methods.

Surprisingly, the KNN method performed better than standard SVM, OSVM and global adaptive SVM methods on the *pageblock*, *parkinson* and *vowel* databases. As discussed in section 7.2, the formulation of global adaptive SVM methods assumes a Gaussian kernel. The success of global adaptive SVM methods on 15 out of 21 UCIML databases, suggests that the assumption of Gaussian kernel is safe and reasonable on most data sets. However, there are some exceptions, as depicted in the results in figure 7.4. The likely reason for the poor performance of SVM methods on these databases might be the form of the kernel. An SVM trained with a linear or polynomial kernel may perform better on these databases.

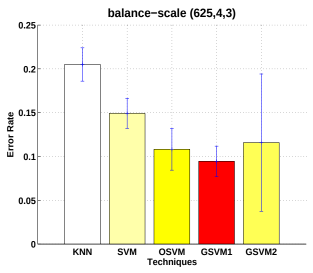
7.3.2 Faces, USPS and Isolet databases

This section deals with the performance evaluation of the global adaptive SVM algorithms on the face, digit and Isolet databases from section 4.3. The detail of databases used in this section is given in table 4.2. The images in all databases are pre-processed for efficiency as described in section 4.3. That is, the dimensionality of the feature-vector representing each image is reduced by using PCA.

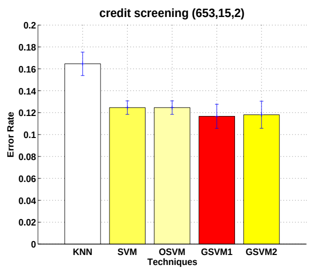
The classification performance in terms of the correctness rate of each of the following methods is shown in figure 7.5 and 7.6:

- **KNN:** Simple 1-nearest neighbor classification with the Euclidean distance.
- **SVM:** Same as in section 7.3.1.
- **MEGM-KNN:** MEGM (algorithm 1) as described in chapter 4.
- **GASVM1:** Same as in section 7.3.1.
- **GASVM2:** Same as in section 7.3.1.

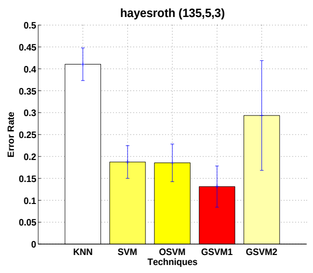
It can be seen from figures 7.5 and 7.6 that the global adaptive SVM methods perform better than the other competing methods on all except caltechfacesB where MEGM-KNN performed best. The results reveal that learning the kernel parameters by employing metric learning algorithms from the k-nearest neighbor framework, for example MEGM, can result in significant improvement in standard SVM performance. Also the results are better than the standard MEGM-KNN formulation, where a 1-nearest neighbor classifier is trained using the metric learned using MEGM.



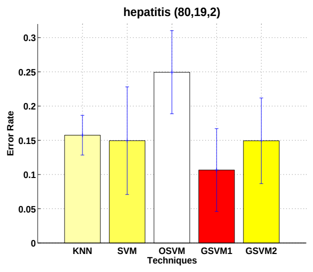
(a)



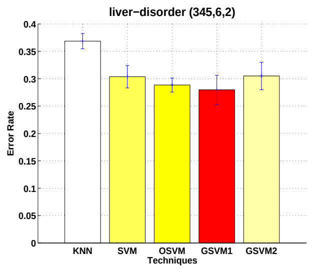
(b)



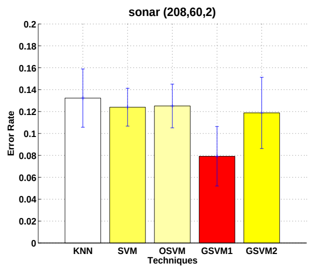
(c)



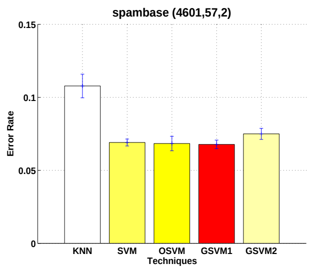
(d)



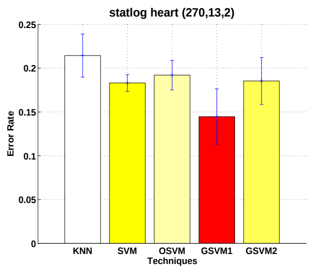
(e)



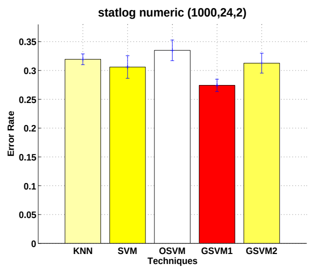
(f)



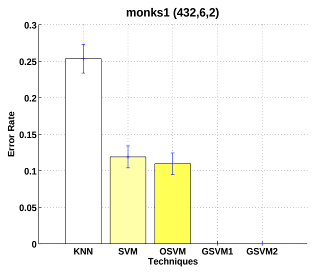
(g)



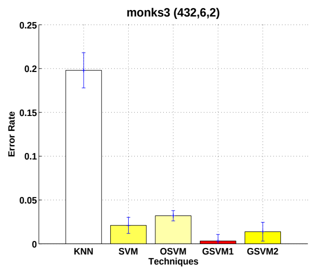
(h)



(i)

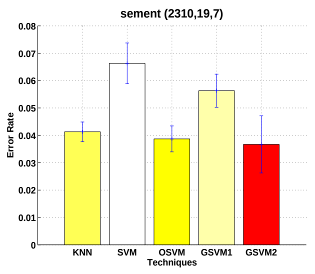


(j)

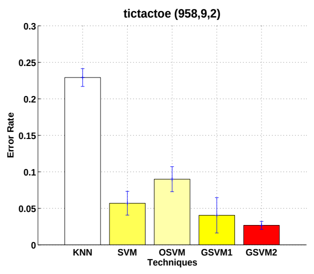


(k)

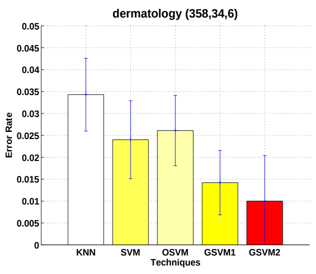
Figure 7.1: Comparison of the error rates of global adaptive SVM methods with KNN, SVM, OSVM on various UCIML databases. GASVM1 performed better than the other methods.



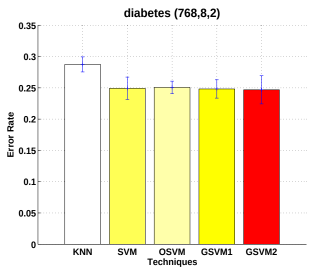
(a)



(b)

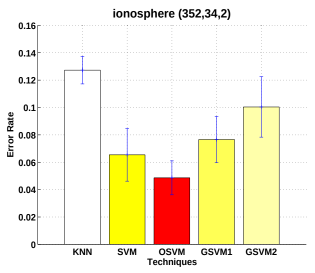


(c)

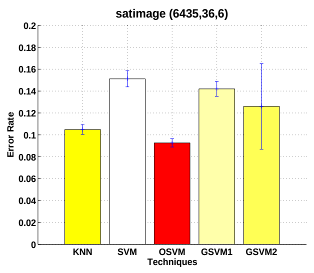


(d)

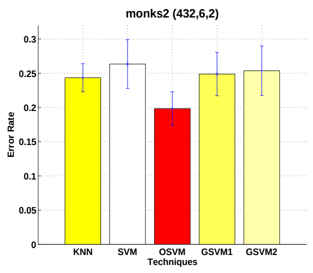
Figure 7.2: Comparison of the error rates of global adaptive SVM methods with KNN, SVM, OSVM on various UCIML databases. GASVM2 performed better than the other methods.



(a)

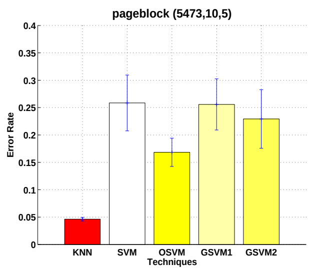


(b)

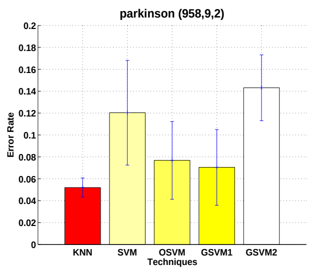


(c)

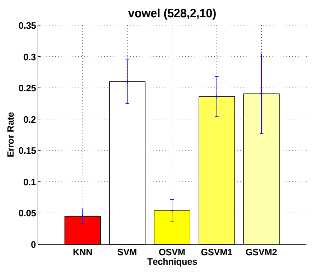
Figure 7.3: Comparison of the error rates of global adaptive SVM methods with KNN, SVM, OSVM on various UCIML databases. OASVM performed better than the other methods.



(a)

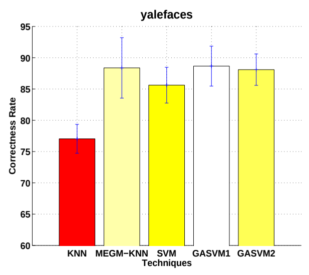


(b)

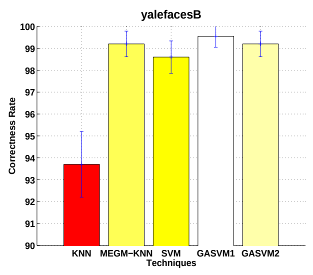


(c)

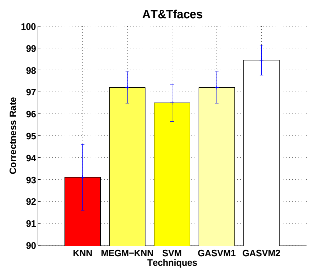
Figure 7.4: Comparison of the error rates of global adaptive SVM methods with KNN, SVM, OSVM on various UCIML databases. KNN performed better than the other methods.



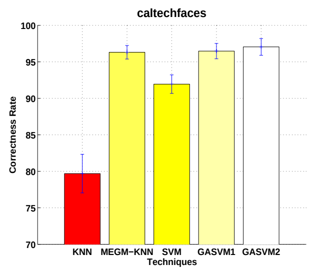
(a)



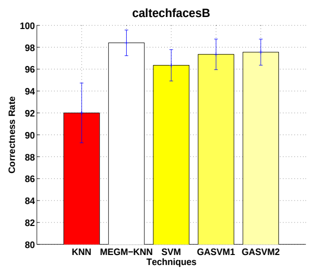
(b)



(c)

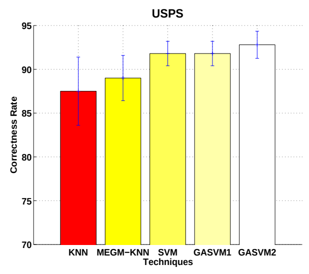


(d)

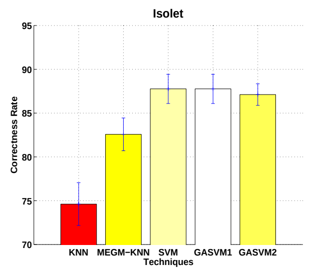


(e)

Figure 7.5: Comparison of the correctness rate of global adaptive SVM methods with KNN, MEGM-KNN and standard SVM on various face databases. The mean and standard deviation of the correctness rate over ten runs (each run with different training data) is reported. The number of training images per category, number of testing images per category and the number of Eigen-vectors used for each database is given in table 4.2.



(a)



(b)

Figure 7.6: Comparison of the correctness rate of global adaptive SVM methods with KNN, MEGM-KNN and standard SVM on various USPS and digit database. The mean and standard deviation of the correctness rate over ten runs (each run with different training data) is reported. The number of training images per category, number of testing images per category and the number of Eigen-vectors used for each database is given in table 4.2.

Chapter 8

Local Adaptive SVM

Recently, local SVM (LSVM) classifiers have been proposed that aim to combine the computational efficiency of a k-nearest neighbor classifier and the classification efficiency of an SVM classifier (Zhang, Berg, Maire and Malik, 2006). The idea is that, rather than training a single global SVM classifier for the entire training data, a separate SVM classifier can be trained in the neighborhood of each query point. One can extend the idea of the global adaptive SVM classification from chapter 7 to LSVM classification. In this chapter local adaptive SVM (LASVM) classifiers will be introduced which are based on learning a global data-dependent distance metric using a complete metric learning algorithm and training a LSVM classifier in the modified neighborhood. As discussed in section 2.4.5, the motivations behind LASVM classification are the same as those behind local adaptive k-nearest neighbor classification. The goal of this chapter is to put local SVM classifiers in the context of kernel parameter tuning. The related work in the area of LSVM classification will be discussed in section 8.1. In section 8.2, a novel LASVM classification algorithm, which is aimed at improving LSVM performance, will be proposed. The experimental results comparing the performance of the proposed algorithm will be provided in section 8.3. The computational efficiency of LSVM classifiers will be discussed in section 8.4.

Though extremely popular and efficient, there are at least two issues that need to be addressed when training SVM classifiers. First, the SVM is designed for binary classification problems. For multi-class classification, one has to resort to one-versus-all or one-versus-one classification strategies. This means that the model becomes more and more complicated as the number of classes increases. Also, any time a new category is added, all classifiers have to be retrained. This results in very long training and testing times, rendering them computationally inefficient. The second issue is related to the tuning of SVM hyper-parameters for example $\mathcal{C}$ and kernel parameter σ in equation 2.24. As discussed in section 2.4.3, these parameters are typically tuned through cross-validation schemes that are computationally expensive.

Local methods, such as local logistic regression and k-nearest neighbor methods, provide an alternative strategy to training a complex global model. As discussed in chapter 2, k-nearest neighbor methods in particular have been shown to be very effective for classification and, with the right distance measure, they can out-perform far more sophisticated alternatives (Berg et al., 2005; Belongie et al., 2005; Hastie and Tibshirani, 1996; Sriperumbudar et al., 2008). Still, the fundamental strength of the k-nearest neighbor classifier stems from its simplicity. Unlike SVM, the k-nearest neighbor classifier deals with multi-class problems effortlessly. The success of k-nearest neighbor methods, however, has taught two valuable lessons: first, good recognition performance is achieved by defining the right similarity measure for the prototypes, which corroborates early human perception studies (Rosch, 1973); second, most of the information required to make a decision about the label of a query is present in its local neighborhood.

Building on these insights from local methods and motivated by Zhang, Berg, Maire and Malik (2006), we propose to use SVM classifiers locally in this chapter. That is, rather than training a single global SVM classifier, an SVM classifier is trained in the local neighborhood of each query point. LSVM classification in some ways alleviates the problems (described above) associated with SVM classifiers. Some motivations behind LSVM formulations are described below.

A major motivation for using LSVM classification is to improve the classification performance (Zhang, Berg, Maire and Malik, 2006). As will be shown in section 8.3, an LSVM classifier results in performance gain in some databases, but performance deteriorates on most databases. The LSVM classifier, however, has an advantage over standard SVM as it involves no training[1], and extension to greater numbers of classes is more feasible. As an SVM classifier has to be trained for every query point, the testing time is likely to increase. Despite this, in most databases having C classes, a query point is surrounded by instances of only c classes, and usually $c \ll C$. This can expedite the recognition phase rather than slowing it down.

Another motivation (which is not reported by Zhang, Berg, Maire and Malik (2006)) for training SVM locally has to do with the multiple kernel scaling parameters. It is hoped that, at least locally, class conditional probabilities vary similarly across all features and using an isotropic kernel may not hurt much. This avoids the need to tune multiple kernel parameters in the SVM. The only parameter that needs to be tuned for LSVM formulation is the size of the neighborhood in which the SVM classifier is to be trained. In section 2.4.3, it was discussed that in much computer vision research, to avoid expensive cross-validation, the scaling parameter σ of the kernel is set to be the average value of the squared distance between entire training data, that is: $\sigma^2 = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} d^2(\vec{x}_i, \vec{x}_j)$ (Varma and Ray, 2007; Zhang, Marszalek, Lazebnik and Schmid, 2006). Using σ as an averaged value of distances between the entire training data set might not be effective. Since in LSVM settings SVM classifiers are trained locally, using σ as the average distance of local training data might be locally optimal and might improve classification performance.

Local SVM classifiers provide a mechanism for incorporating non-stationary kernels in the SVM framework.

[1]As LSVM falls under lazy learning paradigm, like k-nearest neighbor it postpones all training till testing. When a query point is encountered, it trains C classifiers based on the number of classes present in the neighborhood. As a result, testing time of LSVM is likely to be more as compared to standard SVM. This will be discussed in section 8.4.

One drawback of the LSVM formulation is its reliance on the parameter characterizing the size of the neighborhood. As will be shown in section 8.3, there is a trend that the classification performance of LSVM improves as the neighborhood size increases. As the neighborhood size increases, however, LSVM classification becomes less computationally efficient. There is a trade-off necessary between computational and classification efficiency. In this chapter, LASVM classification is proposed. That is, rather than simply training an SVM classifier in the local neighborhood, one can first adapt the neighborhood and train an SVM classifier in the adapted neighborhood. As will be shown in section 8.3, such local adaptation of the neighborhood will render SVM less sensitive to the neighborhood size.

Various local adaptive metric learning techniques were discussed in chapter 6, for example BoostML1 (algorithm 4), DANN (Hastie and Tibshirani, 1996), Machete (Friedman, 1994), etc. Any of these local metric learning techniques can be used to adapt the local neighborhood in which an SVM classifier is trained. As mentioned in chapter 6, as we may not have enough training data in the local neighborhood of a query point, adapting a neighborhood using a complete metric learning algorithm, for example MEGM (algorithm 1), may not be efficient. Therefore, the LASVM formulation proposed in this chapter learns a *global* distance metric using a complete metric learning algorithm. This results in adapting the local neighborhoods as well. The list of algorithms proposed in this chapter is given in table 8.1.

Algorithms	**Description**
LSVM	Train an SVM classifier in the local neighborhood of a query point.
LASVM	Train an SVM classifier in the local adapted neighborhood of a query point.

Table 8.1: List of Algorithms proposed in chapter 8

8.1 Related Work

LSVM classification methods have been explored in some detail in Cheng et al. (2010); Zhang, Berg, Maire and Malik (2006). Cheng et al. (2010) proposed a local SVM classification technique called ‘Profile SVM’ (PSVM). The data is first clustered and a separate SVM classifier is trained for every cluster. A query point is first assigned to its nearest cluster and the corresponding SVM is used to predict its label. A modified form of k-means clustering algorithm ‘MagKmeans’ is proposed which modifies the k-means clustering criterion to control the class distribution of the training data within each cluster. Given the training data X and the corresponding class labels Y, the objective function of MagKmeans can be written as:

$$\min_{Z,C} \sum_{j}^{C} \sum_{i}^{N} Z_{i,j} \|X_i - \mu_j\|_2^2 + O \sum_{j}^{C} | \sum_{i}^{N} Z_{i,j} Y_i | \tag{8.1}$$

where μ_j denotes the mean of cluster j, Z is the cluster membership matrix, whose (i, j)-th element is one if the i-th training data point is assigned to j-th cluster and zero otherwise and O is the scaling parameter. As can be seen, the first term in the objective function is similar to objective function of k-means. Its minimization will lead to compact clusters. The second term measures the class imbalance within clusters. This term is minimized when every cluster contains equal numbers of positive and negative training examples. Though interesting, a major drawback of the approach is its reliance on MagKmeans. Similarly, its is not trivial to determine the number of clusters before training. The results may vary depending on the clustering scheme. Though results are reported on UCIML databases where PSVM is shown to outperform standard SVM, the performance of PSVM on larger databases (in terms of the number of data, number of features and number of classes) is not reported.

Similarly Zhang, Berg, Maire and Malik (2006) proposed a method to train an SVM classifier in the local neighborhood of a query point. The size of the neighborhood in which the SVM is trained is tuned through cross-validation. They used custom-designed similarity measures for efficient classification, for example chi-squared, shape context, geometric blur, tangent distance measures, etc. with different databases. Though promising results have been achieved for a variety of object recognition databases, for example USPS, Caltech101, etc., LSVM classification is not discussed in the context of tuning kernel parameters. Also, the effect of the local neighborhood size on LSVM classification performance is not described. In this work, gaps existing in Zhang, Berg, Maire and Malik's (2006) work are filled, and the effects of variation of the neighborhood size on LSVM performance are studied.

8.2 Local Adaptive SVM

In this section, the proposed LASVM algorithm is described. For any query point $\vec{x}_0$, LSVM classification works in the following way:

- Find the K nearest neighbors of $\vec{x}_0$. If N is the cardinality of training data set then $K = k \times \log2(N)$, typically $k \in [1, 2, 3, 5, 10]$[2].

- If the labels of all K points are the same, $\vec{x}_0$ belongs to the corresponding class and the procedure exits.

- If the labels are different, a one-versus-all SVM is trained for each class present and $\vec{x}_0$ is labeled accordingly.

The LASVM formulation extends LSVM by training LSVM in a transformed space, where the transformation is specified by the matrix L learned using the MEGM complete metric learning algorithm. An outline of the LASVM algorithm is given in algorithm 8. Though this formulation of LASVM incorporates MEGM, it should be noted that other complete metric learning algorithms from chapter 4, for example NCA (Goldberger et al., 2005), LMNN (Weinberger et al., 2009), etc. or naive/semi-naive metric learning algorithms from chapter 5, can be used.

[2]Any function altering the neighborhood size can be used.

Algorithm 8 LASVM: Train an SVM classifier in the adapted neighborhood of a query point.

```
Require:
  - $\vec{x}_0$: Testing data.
  - $\{\vec{x}_n, y_n\}_{n=1}^{N}$: Training data.
  - $K = k \times \log2(N)$, typically $k \in [1, 2, 3, 5, 10]$.

  - Learn a data-dependent distance matrix $A$ such that $A = L^T L$ using the MEGM
    complete metric learning algorithm (algorithm 1).
  - Transform both training and testing data using the matrix $L$.

  - Apply the LSVM procedure as described in section 8.2 on the transformed training
    and testing data.
```

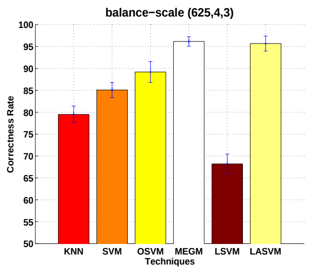
8.3 Experimental Results

In this section, the performance of the proposed LSVM and LASVM algorithms is compared with the standard SVM formulation and k-nearest neighbor methods on various UCIML, faces, USPS digits, Isolet and Coil100 object databases.

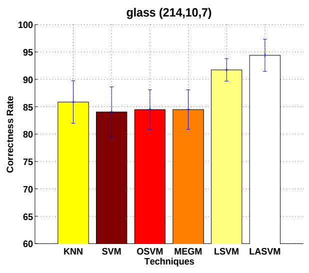
8.3.1 UCIML Databases

The number of data, features and classes for each UCIML database used in this section is reported in table F.1. The correctness rate of each method is obtained using 40 rounds of 2-fold cross-validation. Prior to training, features in all the databases were normalized to have zero mean and unit variance. The classification performance in terms of the correctness rate of each of the following methods for different databases is shown in figure 8.1:

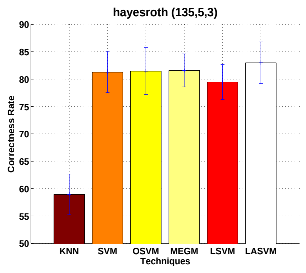
- **KNN:** Simple 1-nearest neighbor classification with the Euclidean distance.
- **SVM:** Standard SVM formulation that is, a multi-class SVM with a Gaussian kernel is used. The $\mathcal{C}$ parameter for the SVM is tuned through cross-validation (that is selected from the set: $\{1, 10, 100, 1000\}$). The value of σ is set to be the average distance of the k-nearest neighbors. A one-versus-all strategy is employed for multi-class classification.
- **MEGM:** MEGM complete metric learning algorithm (algorithm 1, chapter 4).
- **OSVM:** Similar to standard SVM, but both the $\mathcal{C}$ and σ parameters are optimized using cross-validation. This formulation is called ‘optimized SVM’ (OSVM). The $\mathcal{C}$ and σ parameters are selected from the sets: $\mathcal{C} = \{1, 10, 100, 1000\}$ and $\sigma = \{0.1, 0.5, 1, 2, 3, 5\}$ respectively.
- **LSVM:** LSVM classification as described in section 8.2.



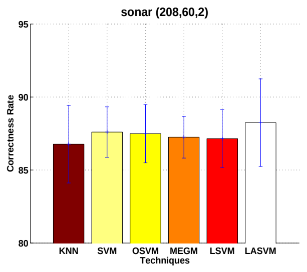
(a)



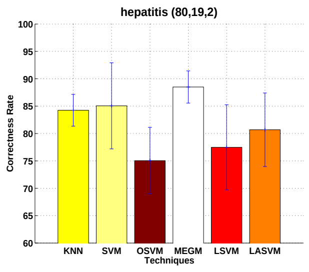
(b)



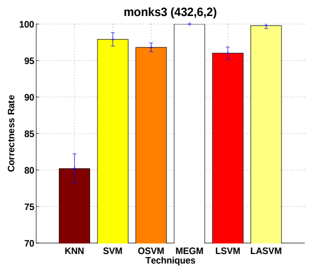
(c)



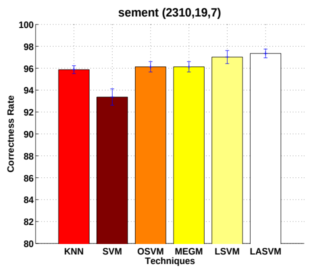
(d)



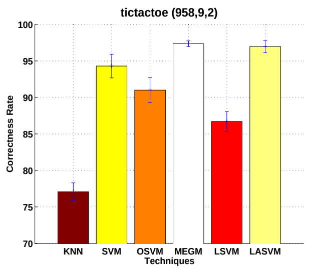
(e)



(f)



(g)



(h)

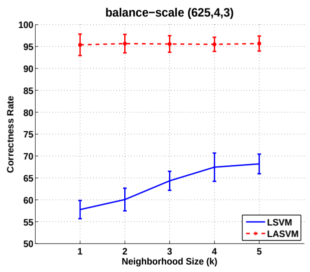
Figure 8.1: Comparison of the correctness rate of LSVM and LASVM with KNN, SVM and MEGM methods on UCIML databases. The LSVM and LASVM results with the best neighborhood size (k) from figure 8.2 are reported for comparison.

.

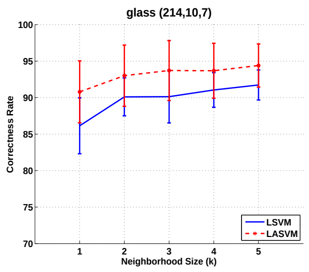
- **LASVM:** LASVM classification (algorithm 8).

The value of $\mathcal{C}$ for both LSVM and LASVM is not optimized and is set equal to ten in all experiments.

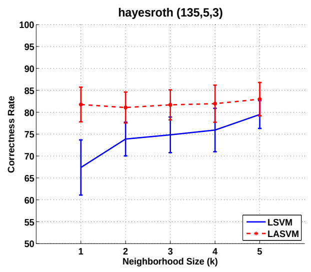
As can be seen from figure 8.1, both LASVM and MEGM performed equally well on most of the databases. It should be noted that LASVM performed better than LSVM on all databases. The comparison of LSVM and LASVM performance when the neighborhood size is varied is shown in figure 8.2. It can be seen from figure 8.2 that LASVM performance is not affected by the neighborhood size (k). The robustness of LASVM with respect to the neighborhood size (k), as compared to LSVM, emphasizes the importance of using the right neighbors for prediction. Since in the original space the neighborhood of the query point does not hold enough discriminatory information for prediction, a larger neighborhood is required. It appears that as LASVM finds a local boundary in the transformed space, its performance is as good in the bigger neighborhood as it is in the smaller one. This highlights the efficacy of the LASVM formulation.



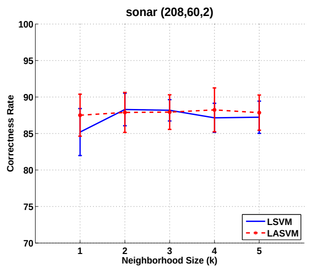
(a)



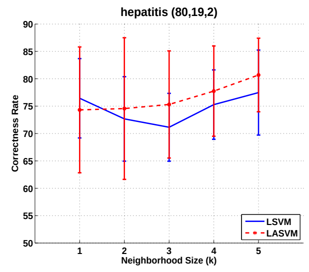
(b)



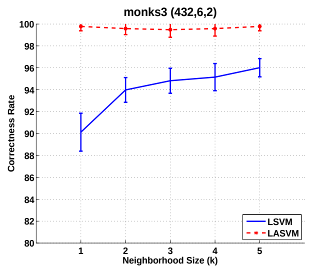
(c)



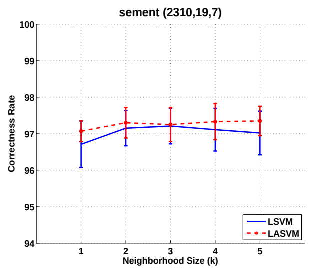
(d)



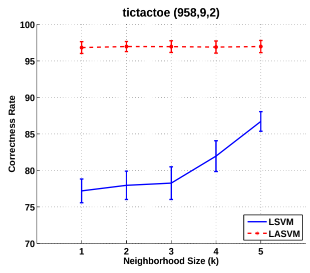
(e)



(f)



(g)



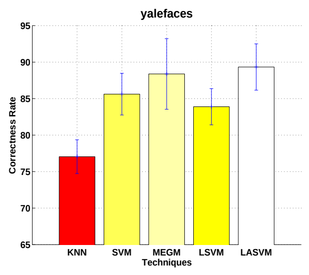
(h)

Figure 8.2: Comparison of the correctness rate of LSVM and LASVM on UCIML databases as the neighborhood size (k) is varied.

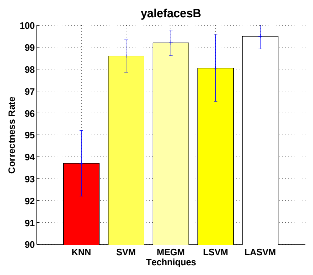
8.3.2 Faces, USPS, Isolet and Coil Databases

To compare the performance of LSVM and LASVM on large databases, five face databases (yalefaces, AT&T, yalefacesB, caltechfaces and caltechfacesB), USPS digit database, Isolet database and Coil100 object database from section 4.3 are used. The details of the databases used in this section are given in table 4.2. The images in all databases are pre-processed for efficiency as described in section 4.3. That is, the dimensionality of the feature-vector representing each image is reduced by using PCA. The classification performance in terms of the correctness rate of each of the following methods for different databases is shown in figure 8.3.

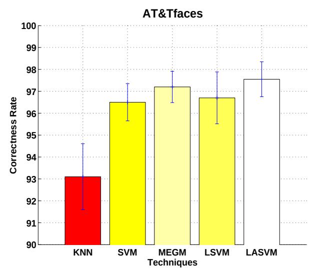
- **KNN:** Simple 1-nearest neighbor classification with Euclidean distance.
- **SVM:** Same as in section 8.3.1.
- **MEGM:** MEGM complete metric learning algorithm (algorithm 1, chapter 4).
- **LSVM:** Same as in section 8.3.1.
- **LASVM:** Same as in section 8.3.1.



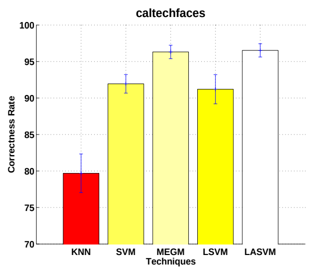
(a)



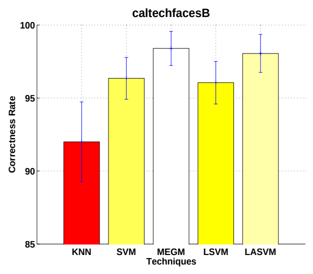
(b)



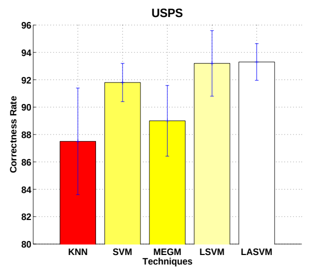
(c)



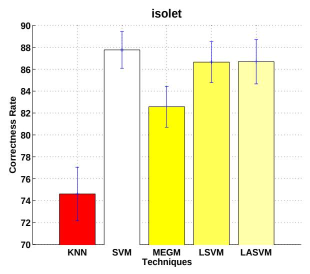
(d)



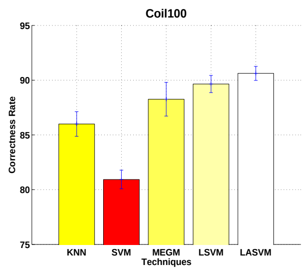
(e)



(f)



(g)

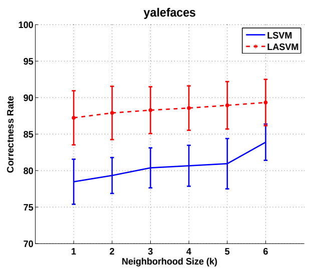


(h)

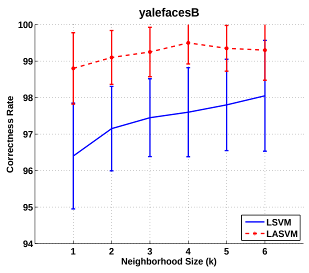
Figure 8.3: Comparison of the correctness rates of LSVM and LASVM method with KNN, SVM and MEGM methods on faces, USPS, Isolet and Coil100 database. The mean and standard deviation of the correctness rate over ten runs (each run with different training data) is reported. The number of training images per category, number of testing images per category and the number of Eigen-vectors used for each database is given in table 4.2. LSVM and LASVM results with the best neighborhood size (k) from figure 8.4 are reported for comparison.

The value of $\mathcal{C}$ for both LSVM and LASVM is not optimized and is set equal to ten in all experiments.

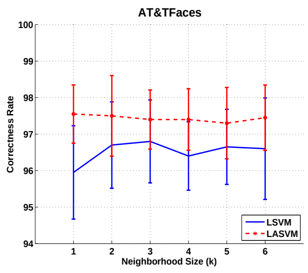
As can be seen from figure 8.3, the LASVM results followed the trend from figure 8.1. That is, LASVM results not only in improving standard SVM performance, but also performed better than the competing MEGM method. LASVM performed best on six out of eight databases, MEGM and standard SVM performed best on one database each (figures 8.4(e), 8.4(g)). LSVM, on the hand, resulted in performance deterioration of standard SVM except on the USPS (figure 8.4(f)) and Coil100 (figure 8.4(h)) databases. This suggests that LSVM, though computationally efficient, may not be the best choice when classification efficiency is of utmost importance. Figure 8.4 compares LSVM and LASVM performance with varying neighborhood size (k). An interesting pattern can be seen. The performance of LSVM increases with the neighborhood size (k), whereas



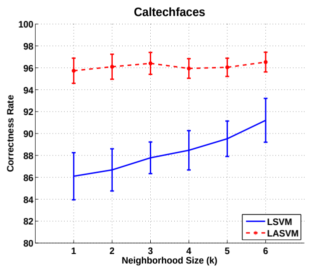
(a)



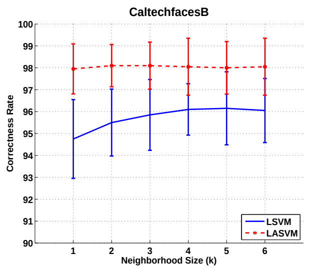
(b)



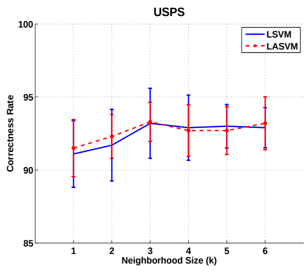
(c)



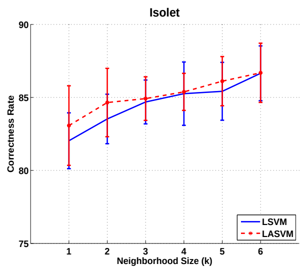
(d)



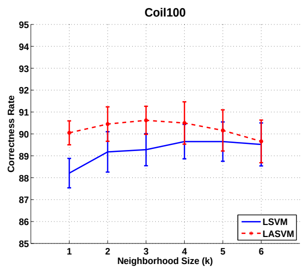
(e)



(f)



(g)



(h)

Figure 8.4: Comparison of the correctness rate of LSVM and LASVM on faces, USPS, Isolet and Coil100 databases as the neighborhood size (k) is varied.

LASVM performance in most cases is not affected by (k). The dependence of LSVM on the neighborhood size may have motivated Zhang, Berg, Maire and Malik (2006) to optimize the neighborhood size through cross-validation.

There are three things that needs to be mentioned. First, as stated, no SVM parameters are optimized in the cases of LSVM and LASVM. Therefore, the results in figures 8.1 and 8.3 are encouraging not only for LASVM but also for LSVM, as LSVM performance is comparable in some cases to standard SVM and OSVM. For example, in the case of the Coil100 database (figure 8.4(h)), LSVM results in huge performance gain over standard SVM. Secondly, it was found that training LSVM and LASVM is far more computationally efficient than standard SVM, provided we keep the neighborhood size reasonably small. Therefore, training LSVM and LASVM can be attractive from a computational efficiency point of view. Third, as depicted in the results, LSVM though improving SVM performance in some cases, can worsen SVM performance in others. This suggests that the LSVM should not be used with Euclidean distance. Indeed Zhang, Berg, Maire and Malik (2006) proposed to use LSVM with a specifically designed distance measure. The proposed local adaptive formulation (LASVM) resulted in far superior performance as compared with LSVM.

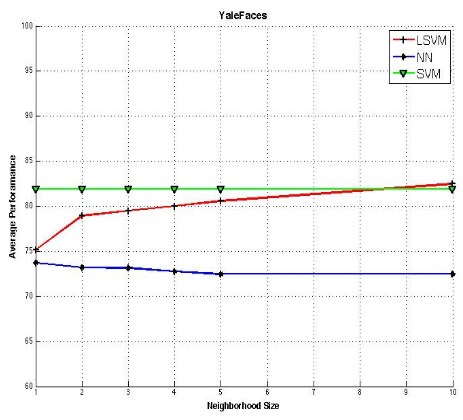
8.4 Analysis of Computational Efficiency

The results on UCIML, faces, USPS, Isolet, and Coil databases in figures 8.1 and 8.3 reveal that LASVM results in significant performance gain over standard SVM, LSVM and also KNN methods. On the other hand, the performance gain of LSVM is marginal. On most databases, in fact, it resulted in deterioration of standard SVM performance. Even though LSVM results in performance degradation, it should be noted that it is more computationally efficient than standard SVM. As mentioned before, when training LSVM, there is a trade-off involved between the computational efficiency and the classification efficiency. This trade-off involved in the training of LSVM is demonstrated in figures 8.5 for yalefaces (figure 8.5(a) and 8.5(b)), AT&faces (figure 8.5(c) and 8.5(d)) yalefacesB (figure 8.5(e) and 8.5(f)).

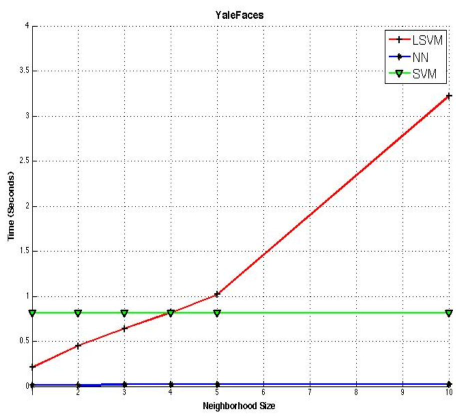
It can be seen from figure 8.5, that though the classification performance of LSVM improves as the neighborhood size (k) increases, the computational time also increases manyfold. There is a need to keep the neighborhood size small without compromising the classification efficiency of LSVM. The LASVM classification framework proposed in this chapter is a remedy for such a problem. It can be seen from figures 8.2 and 8.4 that LASVM results in significant performance improvement of LSVM, and the encouraging aspect of LASVM formulation is that the performance is as good for large neighborhoods as it is for the smaller one. Therefore, rather than aiming to achieve an optimal trade-off between classification and computational efficiency by adjusting neighborhood size (k), LASVM provides a sophisticated and efficient alternative.

8.5 Summary

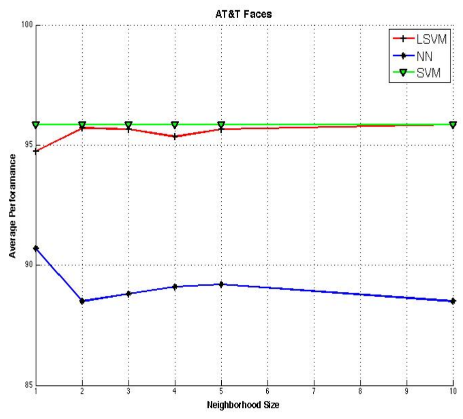
In this chapter, the LSVM and LASVM classification frameworks were introduced. The motivations behind LSVM classification were presented and it was discussed that, apart from classification and computational efficiency, the LSVM should be studied and analyzed from the perspective of kernel tuning. Perhaps when an SVM classifier is trained locally, tuning of the kernel parameters is not as critical as it is when training a global SVM classifier. This is the likely reason for good performance of LSVM on some databases. The classification performance of LSVM was evaluated on a large set of databases. It was found that LSVM can improve standard SVM classification performance when the number of classes is very large. For example, on the Coil100 database LSVM performed better than standard SVM. The size of the neighborhood (k) in which LSVM is to be trained is a critical parameter. It was discussed that k actually controls the trade-off between classification and computational efficiency. A large k results in better classification performance, but it becomes extremely computational inefficient. The LASVM classification algorithm, which is based on LSVM formulation, was proposed. The idea is to train an LSVM classifier in the adapted neighborhood of the query point. On various UCIML, faces, digit and object databases, it was shown that LASVM results in significant improvement over LSVM, standard SVM, and other k-nearest neighbor methods. Also, LASVM classification performance is not strongly dependent on the neighborhood size k. As a result, LASVM can perform extremely well in a small neighborhood. Hence it can gain the computational efficiency of a k-nearest neighbor classifier and the classification efficiency of an SVM classifier.



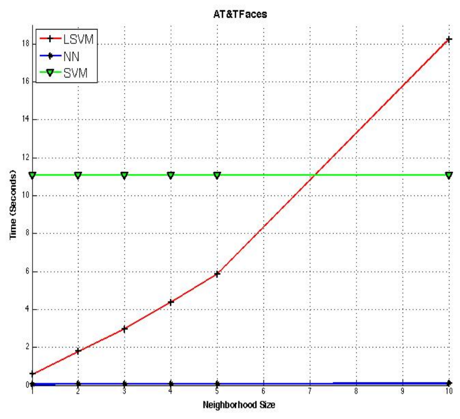
(a)



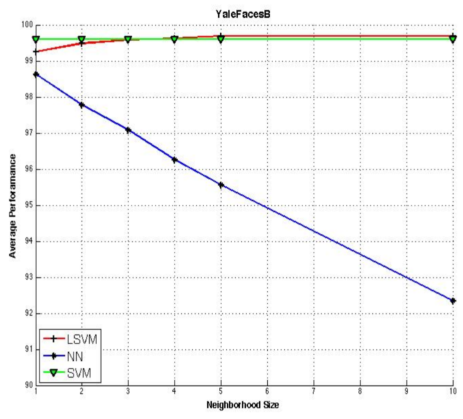
(b)



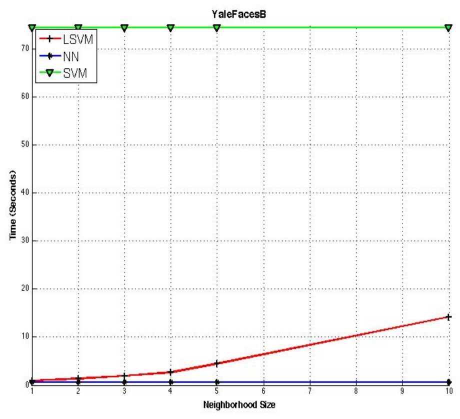
(c)



(d)



(e)



(f)

Figure 8.5: Demonstration of trade-off when training LSVM between computational and classification efficiency on yalefaces (figures 8.5(a) and 8.5(b)), AT&Tfaces (figures 8.5(c) and 8.5(d)) and yalefacesB (figures 8.5(e) and 8.5(f)) databases. The classification efficiency (correctness rate) with varying neighborhood size of LSVM (local SVM), NN (nearest neighbor classifier) and SVM (standard SVM classifier) is shown in the first column. The computational efficiency (time in seconds) with varying neighborhood size of LSVM, NN and SVM classifiers is shown in the second column for yalefaces, AT&Tfaces and yalefacesB respectively. Standard SVM formulation is not dependent on the neighborhood size but is included in the graphs for comparison purpose.

Chapter 9

Global Adaptive Gaussian Processes

Chapters 7 and 8 dealt with the learning of kernel parameters for SVM classification by learning a data-dependent distance metric. This chapter extends the idea of kernel learning by learning a distance metric for GP classification. It will be shown in this chapter that, just like SVM, the kernel parameters of a GP classifier can be optimized by learning a data-dependent distance matrix. The work related to GP classification in the context of metric learning will be discussed in section 9.1. In section 9.2, two algorithms that aim to improve GP classification performance by learning a data-dependent distance matrix will be proposed. The experimental evaluation of the proposed algorithms will be given in section 9.3. A list of the algorithms proposed in this chapter is given in table 9.1.

9.1 Related Work

The problem of optimizing the length scale parameters of an anisotropic Gaussian kernel in the GP framework (learning of a naive/semi-naive metric) was investigated by Neal (1996). The log-likelihood function of a GP model can be written as:

$$\log\ p(\vec{y}|\vec{\theta}) = -\frac{1}{2}\vec{y}^T\mathbf{K}^{-1}\vec{y} - \frac{1}{2}\log\ |\mathbf{K}| - \frac{n}{2}\log\ 2\pi \tag{9.1}$$

where $\vec{\theta} = (A, \sigma_f^2, \sigma_n^2)$, $\vec{y}$ is the prediction of the GP, and $\mathbf{K}$ is the Gram matrix of the input data[1]. We can optimize for the matrix A (also σ_f^2, σ_n^2) by taking the partial derivative of

[1]Refer to appendix E for more details on Gram matrix and its connection with matrix A.

Algorithms	Description
GPML1	GP classification using data-dependent distance metric (a metric is learned for all C categories).
GPML2	GP classification using data-dependent distance metric (a separate metric is learned for each category).

Table 9.1: List of Algorithms proposed in chapter 9

the log likelihood function in equation 9.1 with respect to the matrix A and maximizing it using gradient-based methods. Note that the matrix A is a diagonal matrix. It is also straightforward to introduce priors over $\vec{\theta}$ and maximize the log posterior. Maximizing the log posterior to determine the diagonal elements of the matrix A (length-scale parameters) allows the relative importance of different dimensions to be inferred from the data. This represents an example of ARD in GP classification as discussed in section 2.4.4 (Williams and Rasmussen, 1996).

Zhou and Suter (2008) used spectral techniques to determine the relevance of each feature (hence trained a naive metric). They proposed a measure called 'Major Bandwidth' to measure the spectral properties of each feature and tuned a metric based on this. By learning such a metric they claim that the data is transformed into a space where isotropic behavior is expected. The authors have reported good results for human action recognition (Zhou and Suter, 2008). A major drawback of their method is the assumption that features are independent. Similarly, Snelson et al. (2003); Schmidt and O'Hagan (2003) proposed a transformation of data so that it is well modeled as a GP. The technique proposed by Snelson et al. (2003) learns a transformation as a part of probabilistic modeling rather than as a pre-processing stage.

The methods proposed by Zhou and Suter (2008); Neal (1996); Schmidt and O'Hagan (2003) have been the major motivations for the work in this chapter. The methods proposed in this chapter differ from those described above as the data is pre-processed by learning a full data-dependent distance metric (complete metric learning) whereas previous works in the GP framework have only considered optimizing the diagonal terms (naive/semi-naive metric learning).

9.2 Global Adaptive Gaussian Process

In this section, two algorithms to train a GP classifier with a data-dependent distance metric are proposed. The algorithms are similar to algorithms 6 and 7 from chapter 7 in their reliance on the MEGM complete metric learning algorithm for learning the parameters of a Gaussian kernel.

The outline of the proposed algorithms 'Gaussian process metric learning' (GPML) is given in algorithms 9 and 10. For multi-class GP classification a one-versus-all strategy is used. GPML1 algorithm learns a different kernel using MEGM for each category. Whenever the classifier for category c is used to classify a test point $\vec{x}_0$, the kernel k_{Ac} is used to measure the similarity. GPML2 is a slight variant of GPML1. Instead of learning a separate kernel for each category, GPML2 learns only one kernel k_A for all categories. The learned kernel k_A is used by all c GP classifiers. The two algorithms will be denoted as GPML in the subsequent discussion.

Algorithm 9 GPML1: GP classification using a data-dependent distance metric.

```
Require:
  - $\vec{x}_0$: Testing data.
  - $\{\vec{x}_n, y_n\}_{n=1}^{N}$: Training data.

  for $c = 1, 2, \ldots, C$ do
    - Get a data-dependent distance metric (matrix $A$) using MEGM for category $c$ such
    that $A = L^T L$. The kernel learned for category $c$ is:
        $k_{Ac}(\vec{x}_i, \vec{x}_j) = \exp\left(-\|L\vec{x}_i - L\vec{x}_j\|^2\right)$
  end for
  - Predict the label of the query point $\vec{x}_0$ using the pool of learned kernels $\{k_{Ac}\}_{c=1}^{C}$ in
  the GP formulation.
```

Algorithm 10 GPML2: GP classification using a data-dependent distance metric.

```
Require:
  - Testing data: $\vec{x}_0$.
  - Training data: $\{\vec{x}_n, y_n\}_{n=1}^{N}$.

  - Get a data-dependent distance metric (matrix $A$) for all $C$ categories using MEGM
  such that $A = L^T L$. The kernel learned is:
        $k_A(\vec{x}_i, \vec{x}_j) = \exp\left(-\|L\vec{x}_i - L\vec{x}_j\|^2\right)$
  - Predict the label of the query point $\vec{x}_0$ using the learned kernel $k_A$ in the GP formu-
  lation.
```

9.3 Experimental Results

In this section, the performance of the proposed GPML algorithms is compared with other standard GP methods, the standard SVM classifier, and the k-nearest neighbor classifier on various UCIML, faces and digit databases. The following methods are compared:

- **KNN:** Simple 1-nearest neighbor classification with the Euclidean distance..
- **SVM:** Standard SVM formulation that is, a multi-class SVM with a Gaussian kernel is used. The $\mathcal{C}$ and σ parameters for the SVM are tuned through cross-validation, that is they are selected from the sets: $\mathcal{C} = \{1, 10, 100, 1000\}$ and $\sigma = \{0.1, 0.5, 1, 2, 3, 5\}$ respectively. A one-versus-all strategy is employed for multi-class classification.
- **GP:** Standard GP classifier with an isotropic Gaussian kernel. The value of σ is optimized through cross-validation. σ is selected from the following values: $\{0.1, 0.5, 1.3, 2.0, 2.0\}$.

- **ISO:** Standard GP classifier with an isotropic Gaussian kernel whose length scale value is optimized through the automatic relevance determination procedure as described in section 9.1.

- **ARD:** Standard GP classifier with an anisotropic Gaussian kernel. The values of the length scale parameters are tuned through automatic relevance determination procedure.

- **GPML1:** GP with metric learning (algorithm 9). Apart from the distance matrix, no other parameter is learned.

- **GPML2:** GP with metric learning (algorithm 10). Apart from the distance matrix, no other parameter is learned.

9.3.1 UCIML Repository Databases

The number of data, features and classes for each UCIML database used is reported in table F.1. The correctness rate of each method is obtained using 40 rounds of 2-fold cross-validation. Prior to training, features in all the databases were normalized to have zero mean and unit variance.

The comparative performance (correctness rate) of the different methods for various UCIML databases is shown in figure 9.1. As can be seen from figure 9.1, GPML2 performed better than GPML1. It appears that training one metric for all classes is more effective than training a separate metric for each class. Out of the 12 databases, GPML2 performed best on nine whereas GPML1 performed best on only three (figures 9.1(i), 9.1(j), 9.1(k)). There are two possible reasons: first, due to MEGM, since the MEGM algorithm suffers from local minima problems. There are more chances that local minima will affect the results in the case of GPML1, as the algorithm has to be run C times. Secondly, possibly training a separate metric for each class leads to over-fitting and hence affects GPML1 performance.

9.3.2 Face Databases

This section deals with GPML performance evaluation on large databases. Five face (yalefaces, AT&T, yalefacesB, caltechfaces and caltechfacesB) and one (USPS) digit database from section 4.3 are used. The details of databases used in this section are given in table 4.2. The images in all databases are pre-processed for efficiency as described in section 4.3. That is, the dimensionality of the feature-vector representing each image is reduced by using eigen-faces.

The comparative results (correctness rate) are shown in the figure 9.2. GPML2 performed best on yalefaces, yalefacesB, caltechfaces and caltechfacesB, whereas GPML1 performed best on AT&Tfaces and USPS digit database. On all six databases, the GP classifier trained with metric learning algorithm performed not only better than the standard GP, but also better than ISO and ARD formulations of GP, where parameters are tuned through the computationally expensive automatic relevance determination procedure (section 9.1). As mentioned, there is no tuning of parameters in the GPML algorithms. It only requires a data-dependent distance metric. This highlights the efficacy of the proposed approach, as the results are comparable with, and in most cases better than, the ISO and ARD formulations. GPML performance is also comparable with SVM performance. It should be noted, however, that SVM is optimized by grid searching over $\mathcal{C}$ and σ values.

9.3.3 Comparison with MEGM

In this section, GPML results are compared with the following two metric learning methods in k-nearest neighbor settings:

- **MEGM-KNN:** MEGM complete metric learning (algorithm 1) proposed in chapter 4.
- **NCA:** NCA algorithm discussed in chapter 4 (Goldberger et al., 2005).

The comparison of the results of GPML algorithms with MEGM and NCA on the face and digit databases is given in figure 9.3. The GPML and MEGM-KNN methods each performed best on three databases. The results on the UCIML databases are given in figure 9.4. GPML methods results in significant improvement over the classification accuracy of both MEGM-KNN and NCA. It performed better than all other methods on all but the *tictactoe* data set.

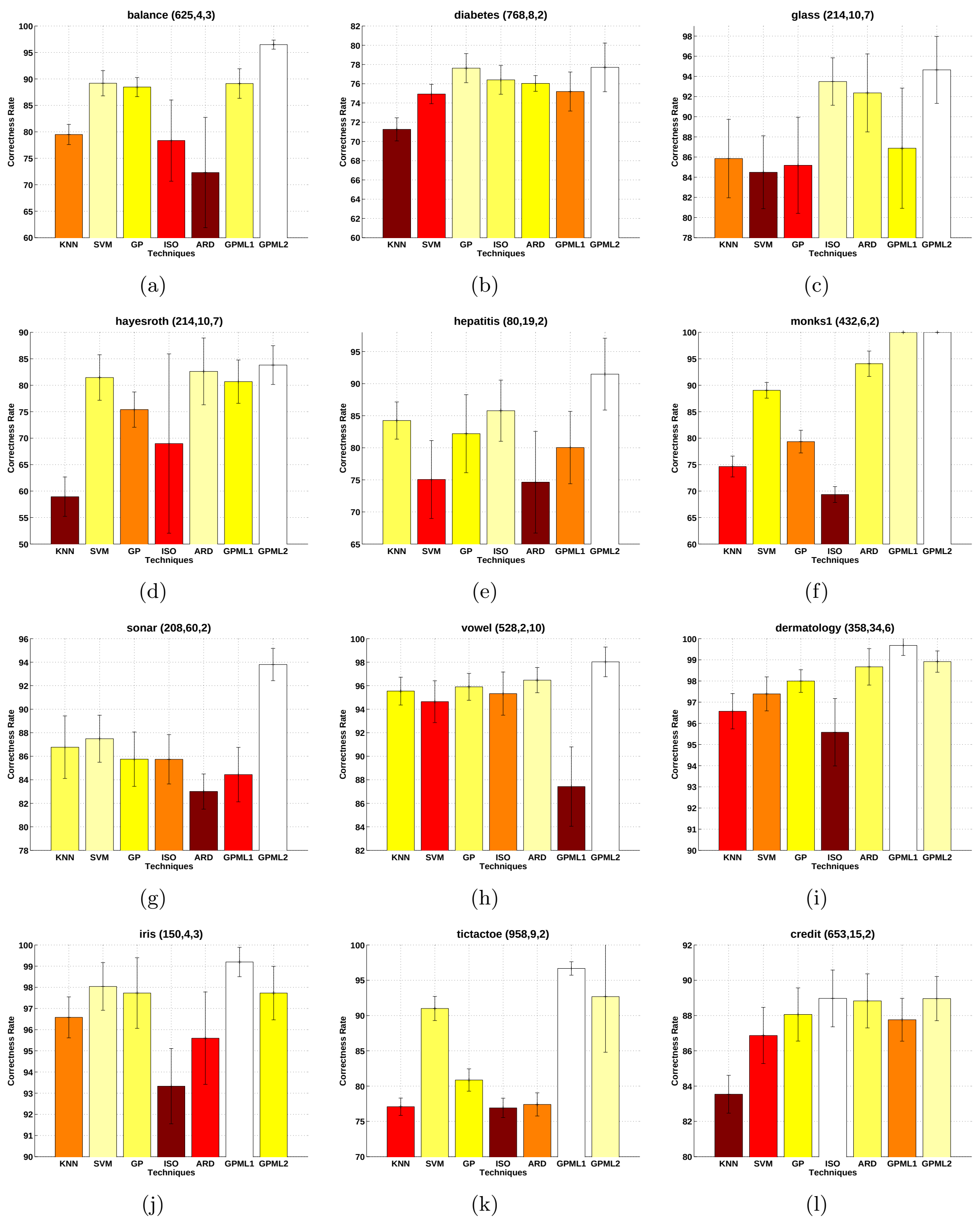
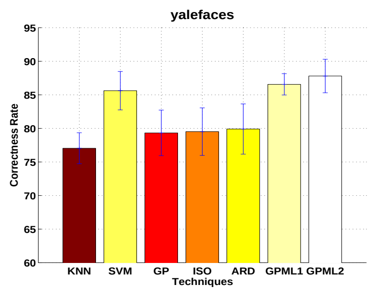
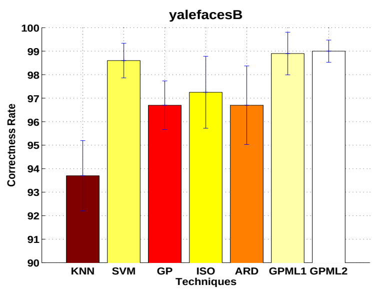


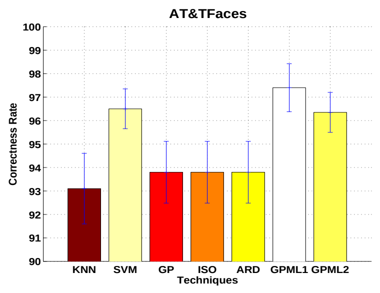
Figure 9.1: Comparison of the correctness rate of GPML1 and GPML2 with the standard GP, the ISO and ARD formulation of GP, KNN and standard SVM on various UCIML databases. The mean and standard deviation of the correctness rate over ten runs (each run with different training data) is reported.



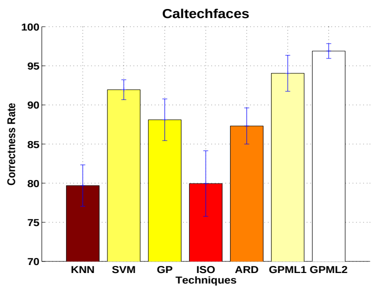
(a)



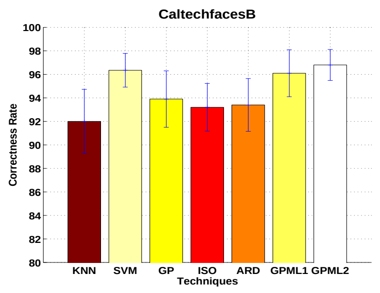
(b)



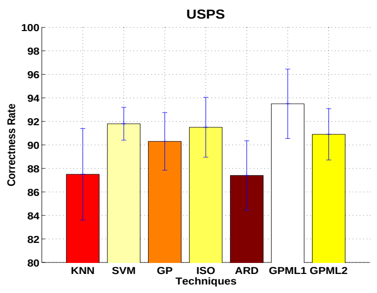
(c)



(d)

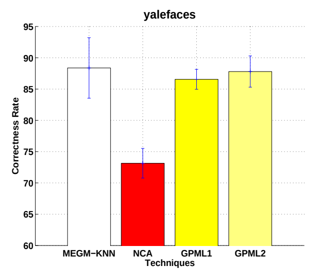


(e)

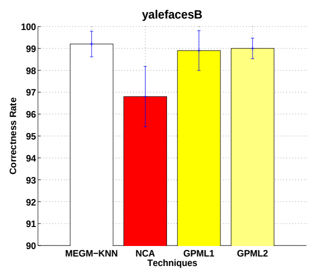


(f)

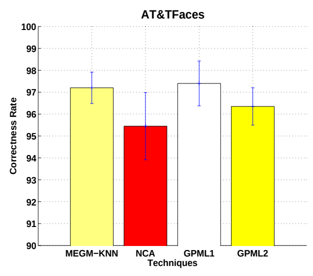
Figure 9.2: Comparison of the correctness rate of GPML1 and GPML2 with the standard GP, the ISO and ARD formulation of GP, KNN, and standard SVM on various faces and USPS databases. The mean and standard deviation of the correctness rate over ten runs (each run with different training data) is reported. The number of training images per category, number of testing images per category and the number of Eigen-vectors used for each database are given in table 4.2.



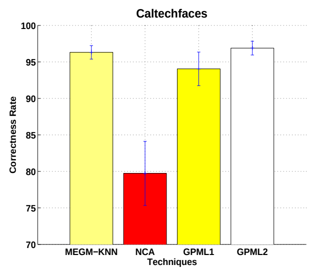
(a)



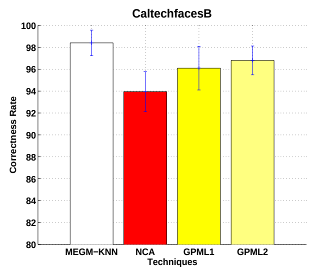
(b)



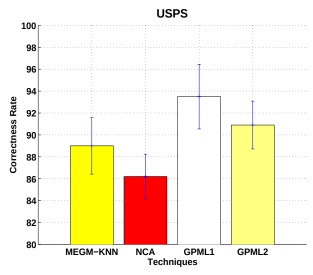
(c)



(d)

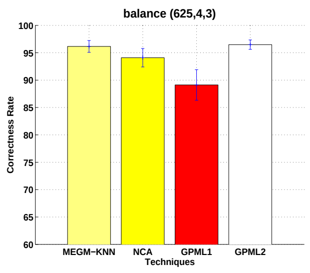


(e)

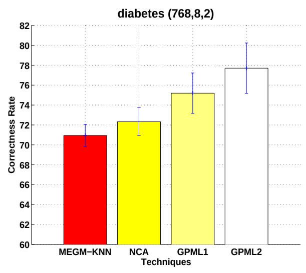


(f)

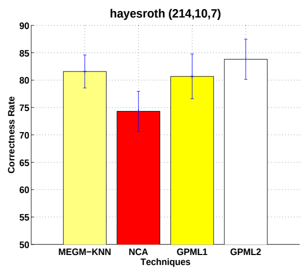
Figure 9.3: Comparison of the correctness rate of MEGM-KNN, NCA, GPML1 and GPML2 on faces and USPS databases. The mean and standard deviation of the correctness rate over ten runs (each run with different training data) is reported. The number of training images per category, number of testing images per category and the number of Eigen-vectors used for each database are given in table 4.2.



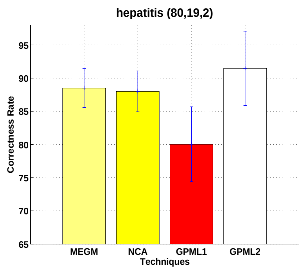
(a)



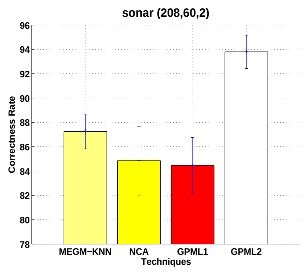
(b)



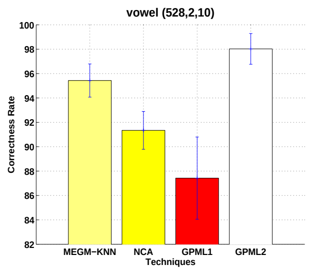
(c)



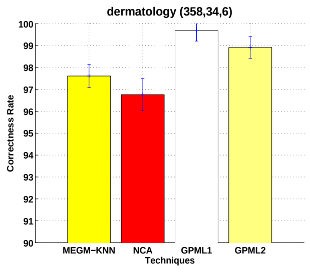
(d)



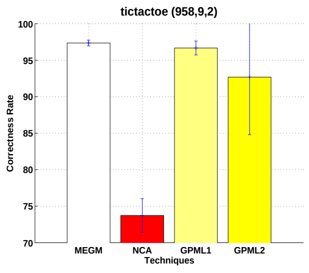
(e)



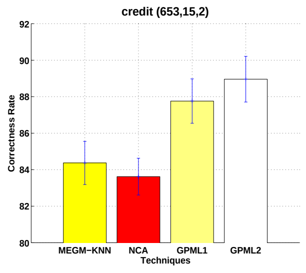
(f)



(g)



(h)



(i)

Figure 9.4: Comparison of the correctness rate of MEGM-KNN, NCA, GPML1 and GPML2 on UCIML databases. The mean and standard deviation of the correctness rate over ten runs (each run with different training data) is reported.

Chapter 10

High-level Metric Learning

This chapter deals with high-level metric learning, which was introduced in section 3.3. In the following various high-level metric learning schemes will be discussed in detail. Related work will be discussed in section 10.2 and a detailed experimental analysis of various forms of high-level metric learning will be given in section 10.4.

10.1 Introduction

Let us consider the two High-level Metric Learning (HML) schemes in the same settings as in section 3.3. We can process the feature-set $\mathcal{FS}(\vec{x})$ in equation 3.1 in the ways described in the following sections.

10.1.1 HML Scheme 1 (HML1)

The two feature-vectors $\vec{x}_m$ and $\vec{x}_n$ can be concatenated into a single feature-vector $\vec{x}$:

$$\vec{x} \;=\; \begin{pmatrix} \vec{x}_m \\ \vec{x}_n \end{pmatrix} \tag{10.1}$$

If $\vec{x}_m$ and $\vec{x}_n$ have lengths of s and c respectively, any metric learning algorithm (naive, semi-naive, complete) can be used to learn the $(s + c) \times (s + c)$-dimensional matrix A in equation 2.15.

10.1.2 HML Scheme 2 (HML2)

An alternative to concatenating feature-vectors $\vec{x}_m$ and $\vec{x}_n$ into a single feature-vector ($\vec{x}$) is to process these feature-vectors separately. That is, we can learn a separate distance metric for each feature-vector. Once a distance metric is learned for each feature-vector, there are two alternatives. We can either combine the distances measured across each feature-vector and train a single classifier (distance fusion), or we can train a separate classifier for each feature-vector and combine the outputs of the classifiers (classifier fusion). In the following these two formulations of distance and classifier fusion are discussed.

Distance Fusion (HML2-DF)

Distance fusion deals with combining the distances measured across different feature-vectors in an optimized way. Once the distances are fused together, any learning algorithm can be used. This strategy also goes by the name of kernel fusion or kernel combination. The idea is simple. Let us suppose that A_m and A_n are the two distance matrices learned for feature-vectors of type m and n respectively. The distance between the two data points $\vec{x}_1$ and $\vec{x}_2$ can be defined in the following way:

$$d^2_{A_{mn}}(\vec{x_1}, \vec{x_2}) = \|\vec{x}_{1m} - \vec{x}_{2m}\|^2_{A_m} + \|\vec{x}_{1n} - \vec{x}_{2n}\|^2_{A_n} \tag{10.2}$$

It can be seen that matrices A_m and A_n induce a linear transformation of data in the subspaces spanned by feature-vectors of type m and n respectively. Combining the distances as shown in equation 10.2 will have the same effect as transforming the data by matrix A_{mn} in the combined space (feature-vectors concatenated together) and measuring distances in that space. The matrix A_{mn} takes the form of a block diagonal matrix and can be written as:

$$A_{mn} = A_m \oplus A_n \tag{10.3}$$

Equation 10.2 can be modified to incorporate a weighting scheme by introducing the parameter $\vec{\omega}$ as:

$$d^2_{A_{mn}}(\vec{x}_1, \vec{x}_2) = \omega_m\|\vec{x}_{1m} - \vec{x}_{2m}\|^2_{A_m} + \omega_n\|\vec{x}_{1n} - \vec{x}_{2n}\|^2_{A_n} \tag{10.4}$$

where the parameter $\vec{\omega}$ controls the relative weighting of each feature type. In summary, HML2 with distance fusion (HML2-DF) involves the following steps:

1. (Optional) Find the distance matrices for each type of feature-vector in the feature-set, for example learn A_m and A_n in equation 10.4.
2. Define the weighting scheme ($\vec{\omega}$).
3. By measuring the similarity between data points using equation 10.4, train a classifier (for example k-nearest neighbor, SVM, GP classifiers).

Note that HML2-DF is exactly equivalent to HML1 when working in the original space (i.e. use of Euclidean distance for all feature-vectors) and the weights specified by $\vec{\omega}$ are equal to one in equation 10.2.

Classifier Fusion (HML2-CF)

Classifier fusion deals with combining the outputs of classifiers trained for each feature-vector in the feature-set. Let us suppose that A_m and A_n are the distance matrices learned for feature-vectors of type m and n respectively. Rather than combining the distances using equation 10.4 and training a single classifier, a separate classifier for each type of feature-vector is trained. The outputs of these classifiers are combined to predict the class label. If f_m and f_n denotes classifiers trained for feature-vectors of type m and n respectively, we can combine these two classifiers as:

$$f(\vec{x}) \quad = \quad \omega_m f_m(\vec{x}_m) + \omega_n f_n(\vec{x}_n) \tag{10.5}$$

where $f_m(\vec{x}_m)$ denotes the output of the classifier f_m for feature-vector of type m and $\vec{\omega}$ controls the relative weighting of each feature type. In summary, HML2 with classifier fusion (HML2-CF) involves the following steps:

1. (Optional) Find the distance matrices for each type of feature-vector in the feature-set, for example A_m and A_n in equation 10.4.

2. Define the weighting scheme ($\vec{\omega}$).

3. Train a different classifier (for example SVM, GP classifiers) for each type of feature-vector using the corresponding distance matrix.

4. Combine the outputs of each classifier by using equation 10.5.

Estimating the distance matrix for each type of feature-vector in the cases of distance and classifier fusion is optional. One can choose to combine distances or classifiers either in the original space (using Euclidean distance) or in the transformed space induced by the distance matrices. In summary, a typical HML2 has to deal with the following issues:

- For each feature-vector, deciding whether to work in the original space or in the transformed space. This step basically determines the distance measure used for each type of feature-vector. If using the transformed space, learn the distance matrix for each feature-vector in the feature-set using a naive, semi-naive or complete metric learning algorithm, otherwise use the Euclidean distance for each feature-vector.

- Deciding whether to use distance fusion or classifier fusion.

- Defining (or learning) the weighting scheme to optimize the distance or classifier combination.

This chapter deals with the abovementioned aspects of high-level metric learning. For illustration and comparison, two very different type of classifiers have been considered in this chapter: SVM (Friedman et al., 2000) and Boosting (Schapire and Singer, 1998) classifiers. The kernel in SVM provides a general framework for distance (kernel) fusion. For example, different types of kernels can be combined to improve SVM classification efficiency. Boosting, on the other hand, does not have an inherent kernel or explicit similarity measure and does not provide a natural framework for fusing distances. Boosting is discussed in section 10.3, where a novel Boosting algorithm is proposed. In short, SVM classification can incorporate HML1, HML2-DF and HML2-CF. Boosting classification on the other hand can only incorporate HML1 and HML2-CF.

10.2 Related Work

The two versions of high-level metric learning (distance fusion and classifier fusion) have been investigated in some detail in machine learning. For example, Boosting classification is an example of classifier fusion (Friedman et al., 2000). Boosting algorithms aim to produce a single strong classifier based on a set of weak classifiers which are combined together with appropriate weights. HML2-CF has been investigated under different names for example mixture of experts, multi-modal data fusion, etc. (Jordan and Jacobs, 1994; Kittler, 1998). The hierarchical mixture of experts architecture was first described by Jordan and Jacobs (1992, 1994). The mixture of experts architecture is actually HML2-CF. It consists of t component classifiers or experts. For any query point $\vec{x}_0$, each component classifier or expert gives an estimate of its category. The outputs are weighted by the 'gating sub-system' governed by the parameter $\vec{\omega}$ (equation 10.5) and are pooled for the final estimation.

With the success of kernel-based methods such as SVM and GP classification, there has been a growing trend in combining distances measured across different types of feature-vectors for optimal classification. This case corresponds to HML2-DF (section 10.1.2). For example Nilsback and Zisserman (2006); Lin et al. (2007); Varma and Ray (2007); Kumar and Sminchisescu (2007); Lazebnik et al. (2006); Bosch et al. (2007) have dealt with learning an optimal combination of distance measurements for object recognition problems. Flower categorization was discussed in section 3.4 and the results were compared with Nilsback and Zisserman (2006). Nilsback and Zisserman (2006) have proposed a method for flower categorization that employs HML2-DF. Three type of feature are used (shape, color and texture) and a brute force search is performed over the combination weights to optimize the distance fusion. As an alternative strategy to distance fusion, the authors have also alluded to classifier fusion in their paper.

Another way of doing distance fusion is by kernel alignment. The idea of kernel alignment was first proposed by Cristianini and Shawe-Taylor (2002) and Pekalskai et al. (2002). It aims at modifying the Gram kernel matrix ($\mathbf{K}$) such that its similarity to the target Gram kernel matrix ($\mathbf{G}$) is maximized. The target Gram kernel can take any form, for example:

$$\mathbf{G}(i,j) = \begin{cases} +1 & \text{if } y_i = y_j \\ -1 & \text{otherwise} \end{cases}$$

Either the parameters used for generating the Gram kernel matrix ($\mathbf{K}$) are modified to maximize some alignment score or the Gram kernel matrix ($\mathbf{K}$) is obtained from a pool of Gram matrices. Weights are sought to combine matrices such that some alignment score is maximized. Therefore, HML2-DF can be written as the following optimization problem:

$$\begin{aligned} & \max_{\vec{\omega}} \mathcal{A}(\mathbf{K}, \mathbf{G}) \\ \text{subject to} \quad & \mathbf{K} = \textstyle\sum_{t=1}^{T} \omega_t \mathbf{K}_t \\ & \operatorname{trace}(\mathbf{K}) = 1 \end{aligned} \tag{10.6}$$

where $\mathcal{A}$ is some function calculating the alignment score (Lanckriet et al., 2002). The kernel alignment strategy has been used for object recognition by Lin et al. (2007), for general optimodal multimedia data fusion by Wu et al. (2004), and for gene prioritization by Bie et al. (2007). Inspired by Lin et al. (2007), Varma and Ray (2007) proposed a method for domain-specific kernel learning based on the combination of the base kernels corresponding to different features. The problem is posed as a convex optimization problem and some state of the art results are shown on various object databases.

An alternative strategy to learning the combination weights $\vec{\omega}$ (for both HML2-DF, HML2-CF) is to hardcode the weights. There exist many problem domains where the relative importance of each type of feature is known and one can specify the weights associated with each type of feature-vector, e.g. spatial pyramid matching algorithms (Lazebnik et al., 2006). Spatial pyramid matching is a simple and computationally efficient extension of orderless bag of features image representation and has been proven to be very effective for object recognition. The technique works by partitioning the image into increasingly fine sub-regions and computing the histograms of local features found inside each sub-region. The resulting histograms at different levels of partitioning (pyramid) are used as feature-vectors. It can be seen that, in this formulation, the histogram matches at the finer levels should be given more weight, or in other words the histogram matches at each region should be weighted inversely proportional to the width of region. The idea of the pyramid matching kernel was first introduced by Grauman and Darrell (2005). Bosch et al. (2007) have also explored the idea of combining kernels computed at different spatial pyramid level for object classification.

10.3 Confidence-Rated Adaptive Boosting

Recently Boosting-based learning algorithms have been shown to be very effective discriminative classification techniques. Torralba et al. (2007); Opelt et al. (2006); Viola and Jones (2001) have used them for object recognition tasks. Boosting is a general learning framework. The idea is to combine a number of weak classifiers to produce a monolithic strong classifier, whose performance is better than all the combined weak classifiers. Boosting initializes every training data point with a weight d. Initially all data points are given the same weight, i.e. for $n = 1, \ldots, N$

$$d_t(n) = \frac{1}{N} \tag{10.7}$$

where $d_t(n)$ denotes the weight associated with data point n at iteration t. During training iterations, the samples are re-weighted according to the training error of the individual weak classifiers. The weight associated with the misclassified samples is increased, and decreased for the samples which are correctly classified. The weak classifiers trained at later iterations, therefore, concentrate on the harder training samples (having higher weights). A number of variants of Boosting exist in the literature for example AdaBoost, GentleBoost, etc. (Schapire, 2003; Friedman et al., 2000). It is better to view Boosting as a framework of learning from weak classifiers. Any weak classifier can be used and any weight alteration strategy can be employed.

The discrete version of Boosting, AdaBoost, defines a strong binary classifier F as:

$$F(\vec{x}) \quad = \quad \text{sign}(\sum_{t=1}^{T} f_t(\vec{x})) \tag{10.8}$$

where T is the number of Boosting rounds and $f_t(\vec{x})$ is the weak learner trained at iteration t. The weak learner in each round identifies the most discriminative feature along which to classify the training data. The sign of $f_t(\vec{x})$ indicates the class label, and its value gives the confidence of the prediction. In this work, a variant of such a Boosting strategy is proposed. A domain partitioning confidence-rated Boosting algorithm (CRAB) is introduced. In CRAB, each weak classifier partitions the input space into finite bins and gives the prediction related to each partition along with the confidence values of each prediction. That is, rather than proposing a single threshold and giving prediction signs, domain partitioning confidence-rated prediction gives predictions for the full range of the input space.

Algorithm 11 CRAB: Confidence-Rated Adaptive Boosting Algorithm

```
Require:
  - $\vec{x}_0$: Testing data.
  - $\{\vec{x}_n, y_n\}_{n=1}^{N}$: Training data.
  - Initialize the distribution using equation 10.7.

  for $t = 1, 2, \ldots, T$ do
    - Normalize the weights $d_t(n)$.
    for $p = 1, 2, \ldots, P$ do
      - Train a weak classifier $f_p(\vec{x}|d_t)$ on feature $p$ using equation 10.11.
      - Evaluate the cost of weak classifier $f_p(\vec{x}|d_t)$ using equation 10.15.
      - Find the best feature $p$ using equation 10.16.
    end for
    - Update weights as: $d_{t+1} = d_t \exp[-y_i f_t(\vec{x}|d_t)]$
  end for

  - Output: Final strong classifier
```

$$F(\vec{x}_0) = \text{sign}\left(\sum_{t=1}^{T} f_t(\vec{x}_0|d)\right)$$

An outline of the proposed algorithm is given in algorithm 11. The CRAB algorithm at each iteration identifies the most discriminative feature p. This feature is used to classify the training data such that it results in the lowest training error (based on the weights associated with each data point at the current iteration). For each feature p the weighted distribution of the positive and negative samples is defined as

$$h_p^+(\vec{x}|d) = (p(\vec{x}_p|y=+1) \times d(\vec{x}|y=+1))/D^+ \quad (10.9)$$

$$h_p^-(\vec{x}|d) = (p(\vec{x}_p|y=-1) \times d(\vec{x}|y=-1))/D^- \quad (10.10)$$

Here D^+ and D^- are the normalization factors to make $h_p^+(\vec{x}|d)$ and $h_p^+(\vec{x}|d)$ probability distributions. Based on equations 10.9 and 10.10, we can define a weak classifier based on feature p as:

$$f_p(\vec{x}|d) = \frac{1}{2}\log\left(\frac{h_p^+(\vec{x}|d)+\epsilon}{h_p^-(\vec{x}|d)+\epsilon}\right) \quad (10.11)$$

where ϵ is a small constant added for numerical tractability. In the following, the computation of the probabilities in equation 10.11 will be explained. The probabilities $h_p^{\pm}(\vec{x}|d)$ in equation 10.11 are computed by partitioning the input domain. That is, across each feature p, we partition the input space $[\min(\vec{x}_j), \ldots, \max(\vec{x}_j)]$ into K disjoint bins denoted as $\tilde{X}_p^1, \tilde{X}_p^2, \ldots, \tilde{X}_p^K$. The probability of a category being present or not present in the partition k is computed as:

$$h_p^+(k) = \sum_{\vec{x}_{ip} \in X_p^k \wedge y_i=+1} \left(\frac{d_i}{D^+}\right) \quad (10.12)$$

$$h_p^-(k) = \sum_{\vec{x}_{ip} \in X_p^k \wedge y_i=-1} \left(\frac{d_i}{D^-}\right) \quad (10.13)$$

Based on equations 10.12 and 10.13, we can modify the probabilities $h_p^{\pm}(\vec{x}|d)$ in equation 10.11 for each partition k as:

$$f_p(k) = \frac{1}{2}\log\left(\frac{h_p^+(k)+\epsilon}{h_p^-(k)+\epsilon}\right) \quad (10.14)$$

The cost of each feature p, motived from Schapire and Singer (1998), is calculated as:

$$\tilde{C}_p = 2\sum_k \sqrt{h_p^+(k)h_p^-(k)} \quad (10.15)$$

The best feature is selected as

$$\operatorname*{argmin}_p \tilde{C}_p \quad (10.16)$$

A simple demonstration of algorithm 11 is shown in figure 10.1 on a 2-dimensional synthetic data.

Four categories from the *Caltech-101 Object Database* (2006) are used for the performance evaluation of the proposed CRAB algorithm. This subset is typically denoted as Caltech4. The results for CRAB are labeled as ‘cAda’ in the ROC curves. The performance is compared with the the GentleBoost version of Boosting. The results for

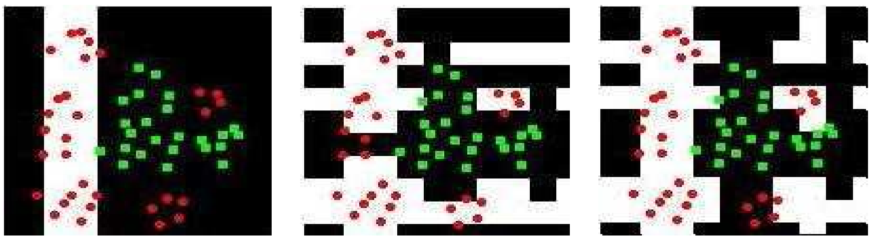


Figure 10.1: Result of applying CRAB classifier (algorithm 11) on the synthetic data after first, third and fifth iteration. The red (class A) and green (class A) blocks are randomly generated 2-dimensional data. White areas are classified as belonging to class A and black as belonging to class B (green blocks). It can be seen that the first few weak classifiers can classify the data quite efficiently.

GentleBoost are labeled as ‘Gentle’ in the ROC curves. Example images from Caltech4 are given in figure 10.2. In this experiment sparse features are used. SIFT features are extracted around some interest points (Lowe, 1999). The interest points are detected using the Harris-Affine interest point detector (Mikolajczyk et al., 2005). Around $200,000$ patches (feature-vectors) are extracted from the training images of four categories and clustered into codebooks of size 250, 500 and 1000. The features are vector-quantized using the codebooks. Each image is represented as a histogram of features contained within the image (the bag of words model) thus representing each image as 250, 500 and 1000 dimensional feature-vector. The classification results in the form of ROC curves are given in figure 10.3. As can be seen from figure 10.3, for the airplanes and cars categories, both CRAB and GentleBoost algorithms have the same performance with codebook sizes 250, 500 and 1000. For faces and motorbikes categories, both GentleBoost and CRAB have very similar performance in the case of a codebook of size 250, but CRAB outperformed GentleBoost when the size of the codebook is increased to 500 or 1000. This suggests that CRAB can handle high dimensional feature-vectors better than GentleBoost.

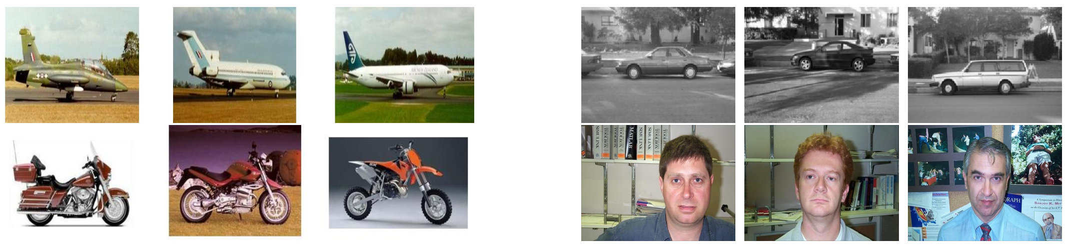


Figure 10.2: Example images (3 images per category) from Caltech4 database.

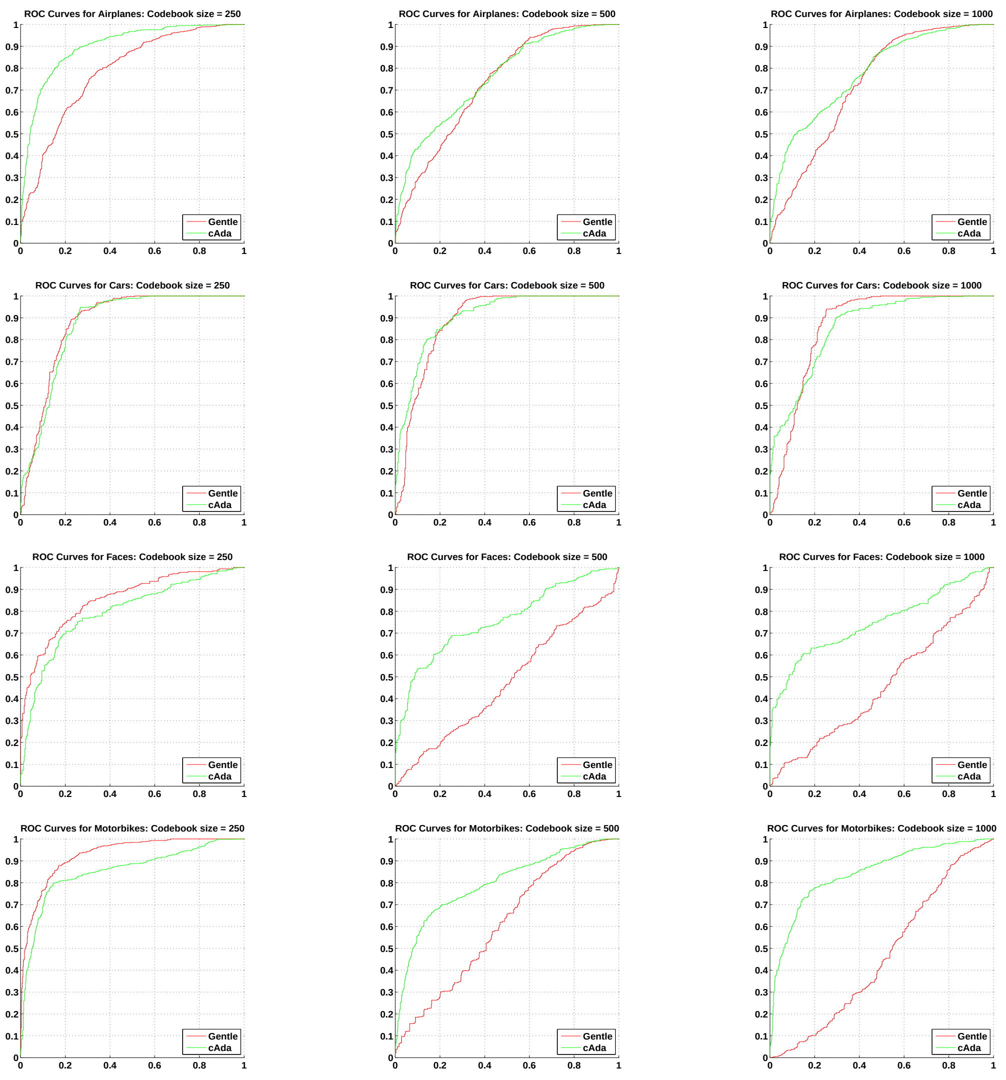


Figure 10.3: Comparison of the classification performance of the CRAB algorithm and GentleBoost on four categories of Caltech4 database. First row: ROC curves for airplanes, second row: ROC curves for cars, third row ROC curves for faces and fourth row: ROC curves for motorbikes. First column: codebook of size 250, second column: codebook of size 500, third column: codebook of size 1000. True Positive on Y-axis and False Positive on X-axis

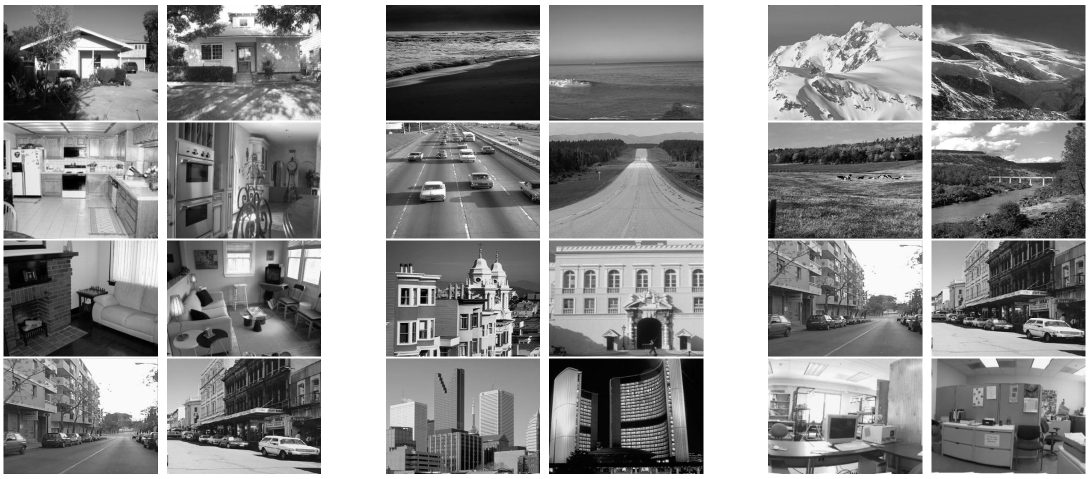


Figure 10.4: Example images (2 images per category) from the Stanford scene database. Fei-Fei and Perona (2005) provided 13 categories from the Stanford scene database, eight of these were originally collected by Oliva and Torralba (2001). Each category has 200 to 400 images. The major sources of the pictures in the database include COREL collection, personal photographs and Google image search. This is one of the most complete scene category database used in computer vision research.

10.4 Experimental Results

In this section, scene classification and object detection are considered to illustrate high-level metric learning.

10.4.1 Scene Categorization

For scene classification problem, the Stanford scene recognition database is used (figure 10.4). Each image is represented by using dense features. SIFT features are extracted from an intensity image (Lowe, 1999). The features are extracted at points on a regular grid of size 20×20. At each grid point, the features are extracted at scales of 4, 8, 16, 24 and 32 pixels. The motivations behind using dense features comes from the comparative study of dense versus sparse features which is provided in appendix C. Once the features are extracted, they are clustered into a codebook of size 200 using the k-means clustering algorithm. The reason for using codebook of size 200 is that it has been used widely in previous work on object recognition (Bosch et al., 2007; Lazebnik et al., 2006) and has been shown to give excellent results. Also no improvement was noticed in the results when using a codebook of a different size. Once the codebook is computed, the features are vector-quantized. A pyramid-based approach is used to compute the feature-vector representing the image[1]. An image at a pyramid level i has 4^i cells. An histogram is

[1]Bag of features methods, while represent an image as an order less collection of local features, have been shown to give excellent performance for object categorization tasks (Dance et al., 2004; Willamowski et al., 2004; Grauman and Darrell, 2005). Though effective, they have some short-comings. Apart from their inability to capture the shape of an object or segment an object from its background, they are disadvantaged because of their disregard for the spatial layout of the features inside the image. These issues with the bag of features model are addressed by Lazebnik et al. (2006), where a pyramid-based approach to compute the histogram has been proposed. Pyramid-based histogram methods use global non-invariant representations based on aggregating statistics of local features over fixed sub-regions.

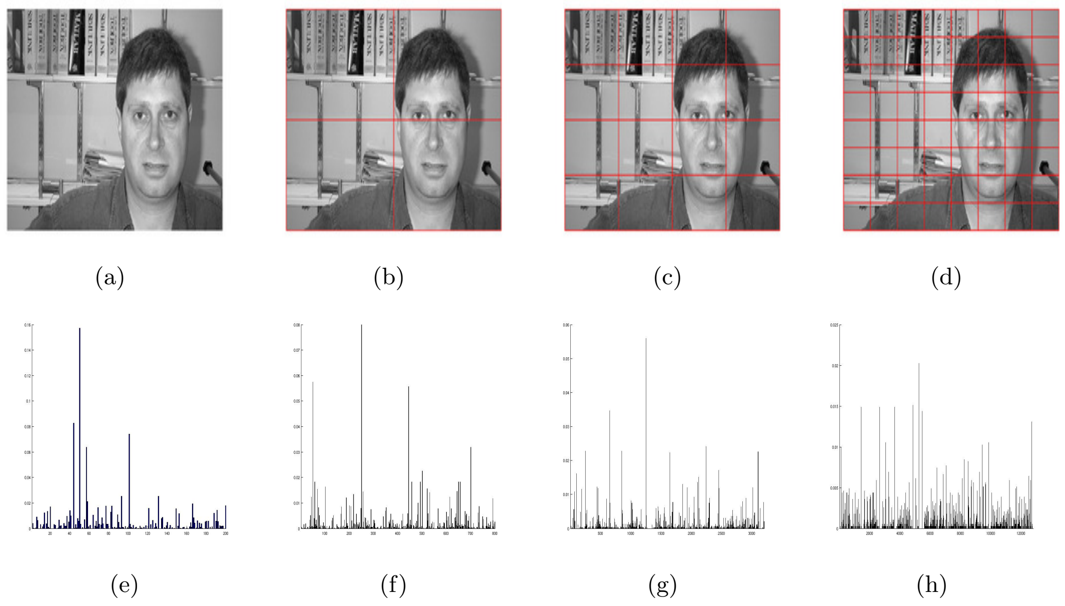


Figure 10.5: Figure showing the first four pyramid levels and resulting feature-vectors. Figure 10.5(a): Pyramid Level 0 (1 cell), Figure 10.5(b): Pyramid Level 1 (4 cells), Figure 10.5(c): Pyramid Level 2 (8 cells), Figure 10.5(d): Pyramid Level 3 (64 cells). Normalized histograms of an image at pyramid level 0,1,2 and 3, having length of 200 (figure 10.5(e)), 800 (figure 10.5(f)), 3200 (figure 10.5(g)) and 12800 (figure 10.5(h)) respectively.

computed for each cell for all pyramid levels. The histograms across each pyramid resolution are concatenated and normalized to form a single feature-vector representing the image for that pyramid level. This is demonstrated in figure 10.5. In this experiment, a maximum pyramid level of 2 is used, therefore, the following resulting features are used for high-level metric learning:

- **f200:** A feature-vector obtained at pyramid level 0 as shown in figure 10.5(a) and 10.5(e).
- **f800:** A feature-vector obtained at pyramid level 1 as shown in figure 10.5(b) and 10.5(f).
- **f3200:** A feature-vector obtained at pyramid level 2 as shown in figure 10.5(c) and 10.5(g).

The classification performance of the following HML schemes on Stanford scene recognition database is given in table 10.1:

- **HML1:** As discussed in section 10.1.1, the feature-vectors f200, f800 and f3200 are concatenated. SVM and Boosting classifiers are trained on the resulting feature-vectors to get the results. [f200 f800]′ denotes concatenation of feature-vectors f200 and f800 whereas [f200 f800 f3200]′ denotes the concatenation of feature-vectors f200, f800 and f3200. The classification results obtained by training a classifier separately with each feature-vector, that is f200, f800 and f3200, are also shown.

- **HML2-DF:** This scheme only incorporates SVM classification. The results are obtained by adding the distances for the three feature-vectors f200, f800 and f3200 and a resulting SVM classifier is trained as discussed in section 10.1.2. The weights ω associated with each feature-vector are set by using the scheme motivated from Lazebnik et al. (2006). Equation 10.4 can be written as:

$$d^2(\vec{x}_1, \vec{x}_2) = \sum_{i=0}^{Q} \frac{1}{2^{Q-i+1}} \|\vec{x}_{1i} - \vec{x}_{2i}\|_2^2 \tag{10.17}$$

  where Q is the number of pyramid levels and is set equal to two. A multi-class SVM is trained in a one-versus-all fashion with a Gaussian kernel whose scaling parameter is set to the average value of the distances between the training points.

- **HML2-CF:** This scheme incorporates both SVM and Boosting classifiers. The results are obtained by training a separate classifier for the three feature-vectors and combining the outputs of each individual classifier. The weights ω associated with each feature-vector are set by using the scheme motivated from Lazebnik et al. (2006). Equation 10.5 can be written as:

$$f(\vec{x}) = \sum_{i=0}^{Q} \frac{1}{2^{Q-i+1}} f_i(\vec{x}_i) \tag{10.18}$$

  where Q is the number of pyramid levels and is set equal to two. A multi-class SVM is trained in a one-versus-all fashion with a Gaussian kernel whose length scale parameter is set to the average value of the distances between the training points. CRAB algorithm, as described in section 10.3 is used to obtain the Boosting results.

Let us concentrate on the SVM results first. It can be seen that HML2 (both with classifier and distance fusion) is more effective than HML1. HML2 achieved performances of 84.90% and 85.50% with distance and classifier fusion respectively, as compared to a best of 83.16% in case of HML1. Surprisingly, feature-vector f3200 alone resulted in a performance of 83.82% which is better than the performance we obtained when concatenating the three feature-vectors i.e. with HML1. Though the difference is not substantial, the results indicate that SVM results are not immune to the curse-of-dimensionality as discussed in chapter 2 and the addition of extra features may lead to performance deterioration unless they are properly weighted. It should be noted that the only difference between the SVM results of HML1 when trained with feature-vector [f200 f800 f3200]$'$ and HML2-DF is the presence of an appropriate weight ($\vec{\omega}$) associated with each feature-vector. Such a simple weighting formulation actually resulted in increasing the performance to 84.90%. Another significant result is that HML2-CF gave better results than HML2-DF.

The results obtained with Boosting followed a different trend from those for SVM. Like SVM, HML2-CF resulted in significant performance improvement over HML1 and a classification performance of 83.14% is achieved. Unlike SVM, with HML1, significant performance gain is achieved when features are concatenated together, that is a performance

	f200	**f800**	**f3200**	**[f200 f800]′**	**[f200 f800 f3200]′**
HML1					
SVM	79.09 ± 0.70	82.62 ± 0.72	83.82 ± 0.79	83.44 ± 0.16	83.16 ± 1.04
Boosting	74.80 ± 0.91	76.62 ± 0.74	73.93 ± 0.77	77.63 ± 0.90	78.18 ± 1.38
HML2-DF					
SVM	-	-	-	84.02 ± 0.20	84.90 ± 0.51
HML2-CF					
SVM	-	-	-	84.10 ± 0.41	85.50 ± 0.79
Boosting	-	-	-	81.75 ± 1.33	83.14 ± 0.71

Table 10.1: Results of high-level metric learning on Stanford scene database. Results for HML scheme 1 and HML scheme 2 (distance and classifier fusion) obtained with SVM and Boosting classification is shown. Mean correctness rate and standard deviation are reported over ten experiments with different training and testing data. 100 images per category were used for testing and training.

of 78.18% is achieved with feature-vector [f200 f800 f3200]′ as compared to the performance of 76.62% achieved with feature-vector f800 only. Like SVM, HML2-CF resulted in significant performance improvement over HML1 and a classification performance of 83.14% is achieved.

In summary, it can be seen from table 10.1 that HML2-CF performed best with an average performance of 85.50%. HML2 for both classifiers resulted in a much better classification performance as compared to HML1. For SVM, the classifier fusion version of HML2 turned out to be more effective as compared to distance fusion version HML2.

10.4.2 Object Detection

For the object detection problem, three categories from the TUDarmstadt database (*The PASCAL Object Recognition Database Collection*, 2005), and one category from the Caltech101 database (*Caltech-101 Object Database*, 2006) are used. The details of these categories are given in figure 10.6. The TUDarmstat database consist of approximately 100 images of three categories: cows, bikes and cars. The bikes and cars categories are quite challenging, as there are significant scale and viewpoint changes. The cows category is easy compared to other two, as there is not much scale and rotation variation. The faces category in the Caltech101 database is excellent for testing detection tasks, as each image is rich in background detail. In the bikes category there were some images containing more than one bike. Since multi-object detection is not incorporated, these images are not used as part of testing or training set. All categories are fully annotated (bounding box present around the object).

Dense SIFT features are used, as in section 10.4.1. The features are extracted at points on a regular grid of size 20×20. At each grid point, the features are extracted at scales of 4, 8, 16, 24 and 32 pixels. Typical object and a non-object grid points, along with the demonstration of scales at which features are computed, are given in figure 10.7. Features extracted at all scales are clustered into a codebook of size 200 using the k-means clustering algorithm and are vector-quantized. Once the features are vector-quantized, a sliding window-based approach is used, where each window is represented by the histogram of features contained within that window (bag of words model). The size of the sliding

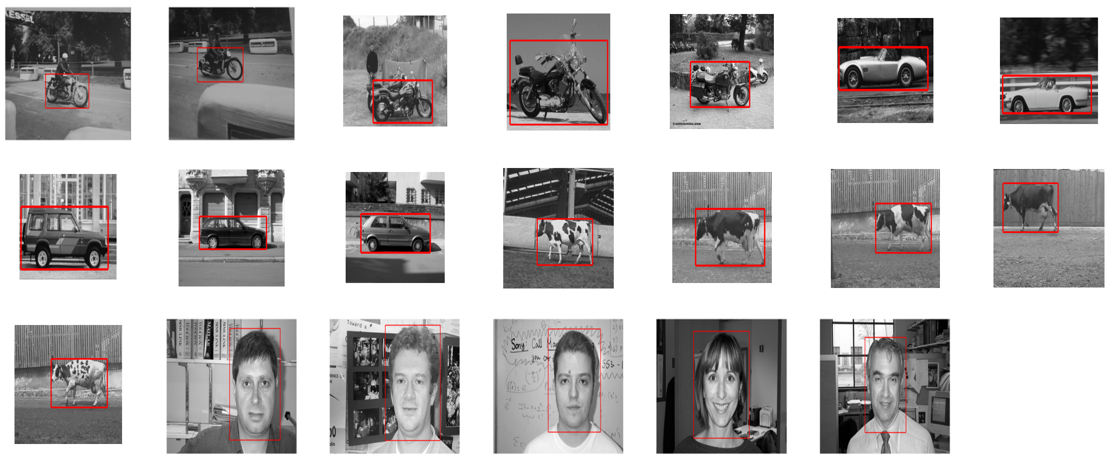


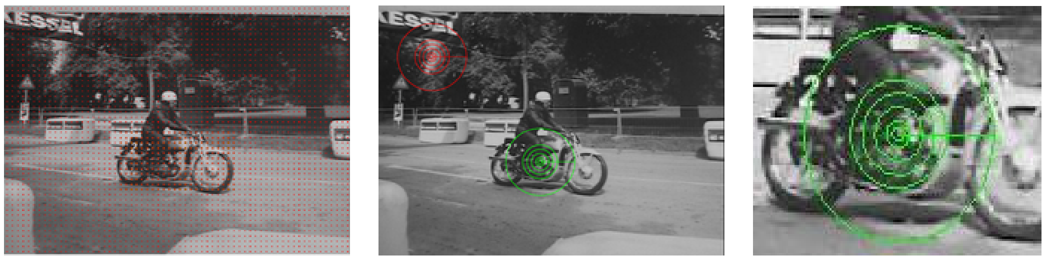
Figure 10.6: Examples images from the TUDarmstadt and Caltech101 database along with their annotations.

window is not fixed (since images vary in sizes). The histograms are computed over the window equal to the size of the region at pyramid level 3 with a spacing of 50 pixels. Each window is represented as the following 200-dimensional feature-vectors.

- **f4:** Feature-vector extracted at scale 4.
- **f8:** Feature-vector extracted at scale 8.
- **f16:** Feature-vector extracted at scale 16.
- **f24:** Feature-vector extracted at scale 24.
- **f32:** Feature-vector extracted at scale 32.

Typical object detection approaches requires background images to train the classifier for any certain object. The idea is to extract features from the object image and from the background images and to train a classifier using object and non-object features. No background images were used in this work. All the categories used are annotated with a bounding box and hence each window of the image can be distinguished as either an object or a non-object window. Hence the feature-vectors extracted for each window are known to belong either to an object or a non-object which are used to train classifiers using one-versus-all scheme. 30, 30, 30 and 100 images are used for training the bikes, cars, cows and faces category respectively. Similarly, 67, 70, 81 and 350 images are used for testing the bikes, cars, cows and faces category respectively.

In order to detect an instance of an object, each window in the testing image is classified as object or non-object with a certain confidence. The output of the classifier for each window acts as a vote in a Hough voting space. Votes are accumulated in a circular search window with a radius of three windows around the center of the window (Comaniciu and Meer, 2002). Accumulated votes above a certain threshold t_{det} are taken as detections of an object instance. An object is deemed correctly detected if the overlap of the bounding boxes (detection vs ground truth) is greater than 70%.



(a) (b) (c)

Figure 10.7: Figure 10.7(a): grid points across the image at which features are extracted, Figure 10.7(b): demonstration of typical scales at which features are extracted (scales depicted in green belong to object whereas scales depicted in red belongs to non-object), Figure 10.7(c): An object point in figure 10.7(b) is magnified.

	f4	**f8**	**f16**	**f24**	**f32**	**[f4 f8 f16 f24 f32]′**
HML1						
SVM	50.23	69.19	75	78.43	79.43	84.11
Boosting	47.75	72.75	74.25	78.25	73.50	84.25
HML2-DF						
SVM	-	-	-	-	-	84.71
HML2-CF						
SVM	-	-	-	-	-	85.20
Boosting	-	-	-	-	-	84.95

Table 10.2: Results of high-level metric learning on the cows, bikes and cars category from the TUDarmstadt database and the faces category from the Caltech101 database. Results for HML1 and HML2 (distance and classifier fusion) obtained with SVM and Boosting classification are shown. Mean correctness rates are reported over ten experiments with different training and testing data.

The classification performance of the following HML schemes on object detection database is given in table 10.2.

- **HML1:** Same as in section 10.4.1.
- **HML2-DF:** Same as in section 10.4.1.
- **HML2-CF:** Same as in section 10.4.1.

The object detection results in table 10.2 follow a somewhat similar pattern as the scene classification results in table 10.1. Let us analyze the Boosting results. As for scene classification, HML2 achieved a performance of 84.95% which is better than 84.25% obtained with HML1. Also, within HML1, significant performance gain is achieved when all feature-vectors are concatenated together.

Similarly, with SVM, HML2-CF performed better than HML1 and HML2-DF. Unlike the scene classification results, within HML1, significant performance gain is achieved when all feature-vectors are concatenated together.

In summary, as for scene classification, HML2 SVM classifier fusion performed best. HML2 for both classifiers resulted in much better classification performance than HML1. Similarly, for SVM, HML2-CF turned out to be more effective than HML2-DF.

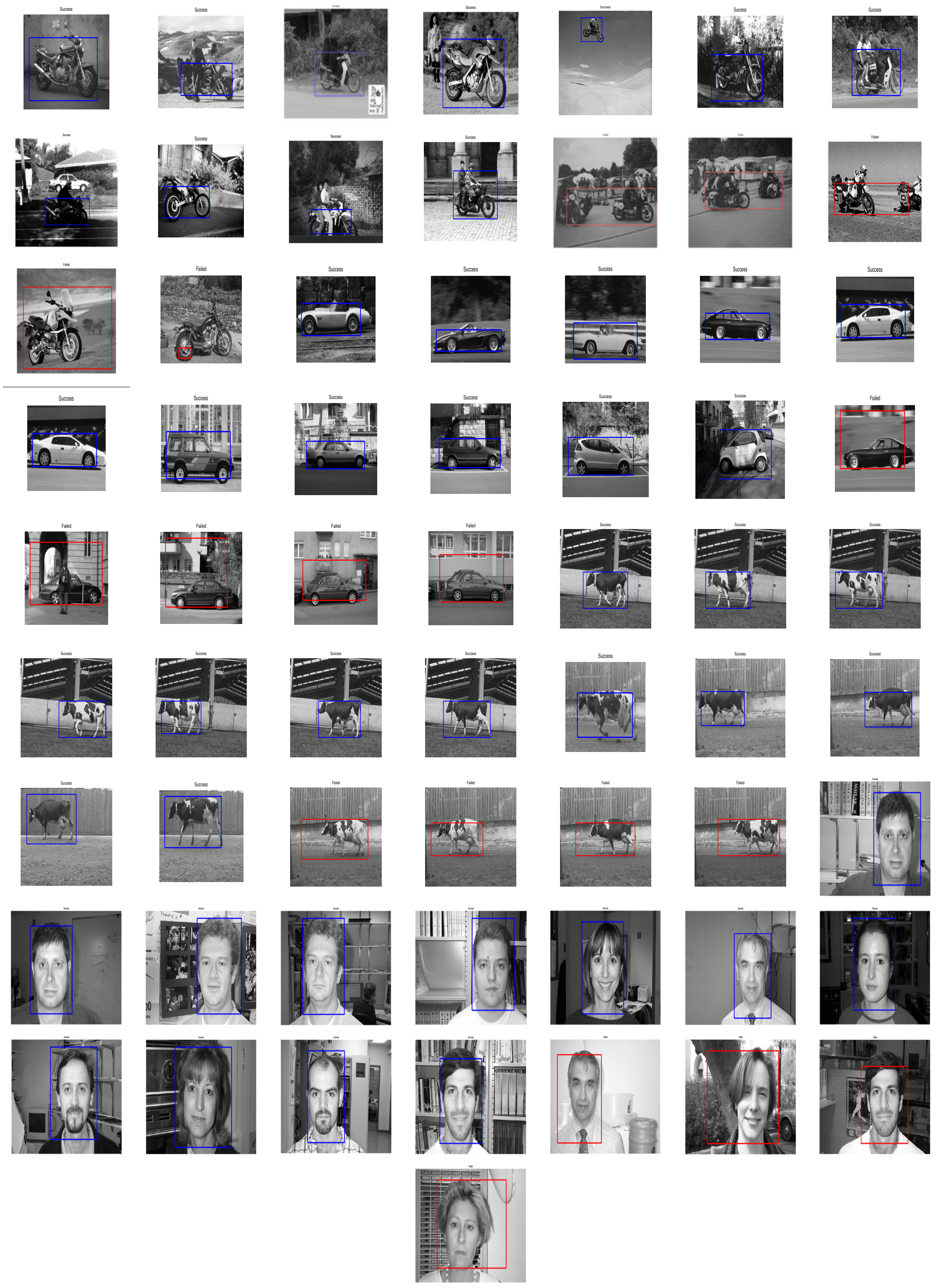


Figure 10.8: Examples of the detection results obtained with Boosting using HML2-CF scheme on the cows, bikes and cars category from the TUDarmstadt database and the faces category from the Caltech101 database. Blue box: correct detection, red box: false detection.

Chapter 11

Further Work and Conclusion

11.1 Further Work

In this thesis, the scope of the work has been limited to k-nearest neighbor, SVM and GP classification frameworks. We acknowledge the fact that fully supporting broader claims such as

> "The performance of most machine learning algorithms depends on their implicit or explicit metric learning approach."

and

> "A distance metric optimized in one learning framework can be applicable across others."

requires an extensive survey, literature review and a deep analysis of machine learning algorithms beyond the k-nearest neighbor, SVM and GP frameworks. We are currently investigating other machine learning techniques. Though our empirical results in this thesis suggest that a metric effective in the framework of one learning algorithm is effective across other frameworks, one needs to work on a mathematical proof for such effectiveness. There is also a need to investigate the comparative performance of different learning algorithms in the presence of an appropriate distance metric.

The proposed categorization scheme for metric learning techniques was based on the form of the distance matrix in distance measurement framework, for example diagonal, block-partition, etc. There is need to further investigate feature selection and feature weighting from the metric learning perspective. This will provide an alternative analysis of feature selection problems.

There is a need to further investigate metric learning methods for kernel tuning. For example, we were limited to the MEGM metric learning algorithm for kernel tuning, but a comparative study of the performance of metric learning algorithms for example NCA, LMNN, RCA, ITML, etc. for tuning SVM kernel parameters is required.

Several research directions can be pursued for local SVM and local adaptive SVM. First, they need to be further examined from the perspective of kernel tuning. Secondly, local SVM scales extremely well with the number of classes and, therefore, is worth investigating for real-time object recognition tasks. Finally, one can investigate combining local and global SVM classifiers for improving classification performance.

A comprehensive analysis of distance and classifier fusion schemes of high-level metric learning is required with classification frameworks such as SVM and GP. Also, there is a need to investigate why classifier fusion results in better performance than distance fusion in the case of SVM.

11.2 Conclusion

In this dissertation, we articulated the importance of an appropriate distance metric for machine learning algorithms and discussed the fact that most learning algorithms rely on tuning a suitable measure of similarity for effective classification. The concept of similarity has deep philosophical meaning and, in a nut shell, we conjectured that the learning capacity of an algorithm is dependent on how well the similarity between data points is measured. We claimed that a data-dependent and a problem-specific measure of similarity is required because the features in most machine learning data sets are incommensurate. Hence all features can not be treated as equally important. We noted that several techniques exist for taking into account the different nature of features, for example feature selection, feature weighting, kernel tuning, etc. We build an argument that these techniques have the same goal as that of learning a distance metric and, therefore, can be viewed as forms of metric learning. To formalize this idea, we introduced a novel categorization of metric learning methods. The existing metric learning methods were categorized into naive, semi-naive, complete and high-level metric learning methods. The scheme provided a powerful way of comparing and improving existing metric learning methods.

We discussed the prevalence of metric learning by showing that most learning algorithms either implicitly or explicitly tune a data-dependent distance metric on which their performance is critically dependent. However, we noted that metric learning in its conventional form is usually restricted to k-nearest neighbor methods and this should not have to be the case. Metric learning is general and actually learns the properties of the input space. In most cases it is independent of the learning algorithm. Therefore, we conjectured that a metric effective in the framework of one learning algorithm can be effective and applicable in the other frameworks. We supported our hypothesis by proposing a novel metric learning algorithm in k-nearest neighbor settings (MEGM) and using it to learn the kernel in SVM and GP frameworks. Encouraging results were reported on a wide range of machine learning data sets, not only highlighting the importance of the metric in various learning algorithms but also showing that a metric trained in k-nearest neighbor settings can be applicable and effective in SVM and GP settings. We provided a unified view of kernel and metric learning, as both are very much related and extended the idea of local metric learning to SVM for computational and classification efficiency.

The work in this thesis is an attempt to view machine learning algorithms from a distance measurement and metric learning perspective. Our encouraging results on most data sets, achieved by tuning a distance metric with the proposed metric learning algorithms, suggest that the performance of learning algorithms can be increased greatly by using a suitable distance metric. Also, various methods such as feature selection, feature weighting, scale estimation, distance fusion, etc. are in fact different forms of metric learning. Similarly, since estimating a data-dependent distance metric is estimation of the properties of the data, it is very likely that a metric optimized in the framework of one learning algorithm is applicable and effective in those of others. In a nutshell, metric learning is an important and interesting direction and one of the many promising emerging areas of machine learning.

Appendix A

Typical Problems in Machine Learning

In this appendix, two kinds of problem prevalent in machine learning are explained. The first kind of problem represents the case when there is a natural partitioning among features, for example object recognition, object detection, etc. A brief history of the object recognition problem is also provided, as well as an explanation of how machine learning helps in solving such a complicated problem. The second kind of problem represents the case where there is no natural partitioning among features. Cancer and mushroom classification are considered for illustration.

Object Recognition

One of the long-awaited goals of computer vision has been to create programs that can recognize objects just as we humans do. Though significant success has been achieved in low-level computer vision, typically known as 'Machine Vision' which generally requires efficient image processing algorithms (as most of the images are taken in controlled and calibrated environments), much ground is yet to be broken for computer vision. Object recognition encompasses much more than mere image processing. Incorporating the fundamental property of learning a new category of an object has to do with cognitive sciences, psychology and animal behavioral studies. Object recognition for computer vision dates back to the mid twentieth century. The research was primarily motivated by advancements in robotics and related areas.

A true object recognition system should be able to recognize thousand of objects from many different categories. Recent efforts to incorporate the ease and comfort with which we humans and some animals recognize objects into machines have not been fruitful. One of the problem hampering these efforts has to do with view-point variance (figure A.1(e)). An object viewed from one angle may look completely different if viewed from another. Also there are huge intra-class variations, for example, different types of chairs such as stool, bean bag, bench, couch, etc., have a different appearance and structure. Taking all intra-class and view-point variations into consideration, training and learning across

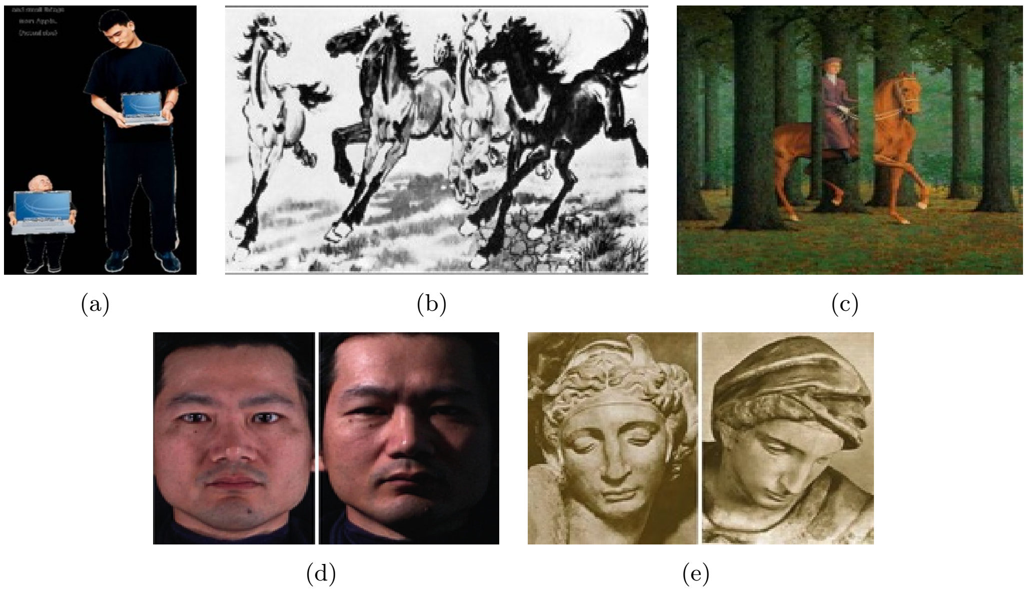


Figure A.1: Example images depicting problems like scale (A.1(a)), deformation (A.1(b)), clutter and occlusion (A.1(c)), brightness variance (A.1(d)) and view-point variance (A.1(e)) for object recognition system. Images courtesy of Fei-Fei et al. (2007).

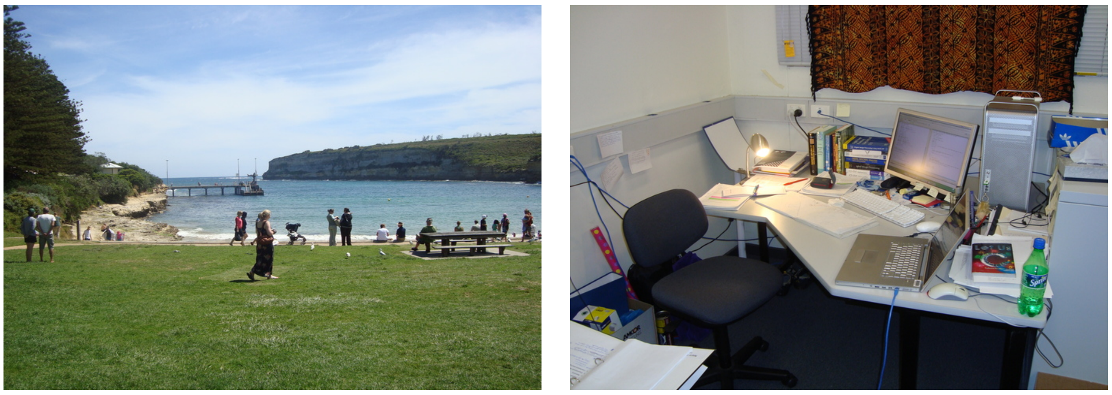
all poses is a task that is not computationally feasible. Apart from view-point and intra-class variation problems, there can be issues due to inherent properties of an image, for example brightness or color variations (figure A.1(d)). Serious problems arise when there is more than one object in an image. For example, consider the extent of information in figures A.2(a) and A.2(b). The two figures represent an outdoor and an indoor scene. Some possible objects of interests are sky, grass, trees, humans, bench, laptop, computer, chair, bottle, lamp, etc. The two images represents an extremely challenging recognition problem. Note the different postures of people in figure A.2(a), that is sitting, walking, running, lying, etc. Similarly note how the books are organized in figure A.2(b) that is vertically, horizontally, stacked together, etc. Also, issues such as scale, deformation, clutter and occlusion make the recognition problem very difficult (figure A.1).

Any viable object recognition system has to deal with all the problems as shown in figure A.1. As a consequence, the recognition process is extremely difficult. Note the deformation of the horses in figure A.1(b). Using a 2-D (2-dimensional) model or a 3-D (3-dimensional) model to recognize objects will not be effective under such deformations. The fundamental problem when it comes to recognizing an object in an image is that of segmentation. Segmentation deals with dividing an image into its constituent regions or objects. The fact that we do not know objects in advance, means that we do not know the extent to which segmentation should be applied. Perhaps, the recognition process is not difficult. But, to recognize an object, we need to segment it first. And similarly we can segment an object only if it can be recognized in the image. Maybe segmentation and recognition should be treated as a same task. Recent object recognition methods are shown to be more effective if recognition and segmentation are considered simultaneously or two stages of a single process (Leibe et al., 2007a; Leibe and Schiele, 2003).

Since images are 2-D projections of real world objects, early research in object recognition focussed on estimating 3-D models of objects and fitting these models to the objects in images (Lowe, 1987). The research in this area of object recognition spans more than three decades (Marr, 1982, chapter 5). Similarly a lot of work was done to define a structural grammar for each object (Leyton, 1988; Evans et al., 1993). Due to view-point variations and huge intra-class variations, fitting a 2-D or a 3-D model of an object to the image or defining the structural grammar for each object turned out to be very computationally expensive. Though much was learned in the development of early computer vision systems, none of them were particularly successful, mainly because of heavy use of hand-tuned heuristics which did not generalize well to unseen data. Early computer vision researchers were very ambitious and it seems that the early pioneers were simply ahead of their time (Yakimovsky and Feldman, 1974; Yakimovsky, 1975, 1964; Feldman and Yakimovsky, 1974; Shafer et al., 1993). They had no choice but to rely on heuristics because they lacked the large amount of data and the computational resources to learn the relationships governing the structure of the objects.

The failure of early researchers to provide robust solutions for object recognition and the related tasks led to a new paradigm in computer vision. Since machine learning deals with the problem of learning from examples, the application of machine learning methods was a natural step forward for creating viable object recognition systems. Rather than specifying a structural model or grammar for each object, machine learning encouraged computer vision researchers to analyze images as data patterns. In the last 15 years, there has been a huge trend towards applying machine learning techniques to object recognition problems. This has turned out to be extremely useful. For example, real-time robust object detection by Viola and Jones (2001), object recognition from local scale-invariant features by Lowe (1999), etc. have been shown to be very robust and effective and are widely used.

Machine learning deals with problems in object recognition tasks in at least three ways. First, machine learning encourages the development of novel features to represent images in such a way that they are not affected by the abovementioned problems (figure A.1). As a learning algorithm relies on features to classify objects, researchers are likely to concentrate on computing effective features. Recently a number of such features have been introduced, for example, scale-invariant-feature-transform (SIFT) features (Lowe, 1999), shape contexts (Belongie et al., 2005), geometric blur (Berg and Malik, 2001), textons (Varma and Zisserman, 2005), etc. It should be noted that the problem of specifying a model for each object has actually been transformed into specifying the appropriate features for objects. Secondly, it leads to the development of novel learning algorithms to recognize objects from their features (Torralba et al., 2007; Xiao et al., 2006). And third, machine learning leads to the division of the difficult problem of generic object recognition into smaller more manageable subsets of problems such as object categorization, scene recognition, object detection, multi-object detection, face recognition, digit recognition, visual tracking, etc. In the following, some of these research directions are briefly explained.



(a) (b)

Figure A.2: Figure A.2(a): A beautiful day at Apollo Bay, VIC, Australia. Figure A.2(b) A cluttered desk in Digital Perception Lab, Monash University, Clayton VIC, Australia.

Object categorization, which also goes by the name of 'object instance recognition', deals with categorizing images (Fei-Fei and Perona, 2007; Dance et al., 2004; Zhang, Berg, Maire and Malik, 2006). The goal here is to categorize an image based on its contents. It is a simple task and easy to solve as the whole image represents only one class, for example, classifying an image of a car versus an image of a toy, etc. Face recognition also falls in the category of object categorization. Of course, the face of each person represents one class. 'Scene understanding' is also a form of object categorization (Oliva and Torralba, 2001; Lazebnik et al., 2006). For example categorizing figure A.2(a) as an outdoor scene and figure A.2(b) as an office instance, etc. Object detection, on the other hand, deals with detecting a certain object in the image (Leibe et al., 2007b; Opelt et al., 2006; Shotton et al., 2005), for example finding the location of the laptop in figure A.2(b), determining if figure A.2(b) contains a car, determining the number and location of the cars in figure A.2(b), etc. Two notable applications of object detection are face (Turk and Pentland, 1991; Viola and Jones, 2001) and pedestrian detection (Leibe and Schiele, 2003).

As described, with the emergence of machine learning techniques, a great deal of success has been achieved in computer vision, but the dream of a major breakthrough in object recognition research is still to be seen. Machine learning research has also benefited from computer vision, as much research in machine learning has been motivated by problems in computer vision. A typical object recognition process using machine learning approaches consists of two main steps. The first step is the representation of an image in a form so that the learning algorithm can be applied. This step is generally known as feature computation or feature extraction, and results in a feature-vector or feature-set representing the image. This results in the transformation of an input image into a P-dimensional feature-vector. A simple example of a feature-vector is the vectorial form of the gray-scale values of each pixel in the image. The second phase of the recognition process deals with choosing and customizing machine learning algorithms to better suit object recognition problems. Though both phases are equally important in creating a robust recognition system, it should be noted that the ability of learning algorithm to perform effectively is dependent on the way images are represented in feature-vectors or feature-sets. It is often desirable to modify the learning algorithm to better fit the feature-vectors or feature-sets and modify the feature-vectors or feature-sets (for example re-scaling, partitioning, etc.) to better fit the learning algorithms. In short, the problem of learning new objects is actually the problem of learning appropriate discriminative features describing the object.

Cancer and Mushroom Classification

Let us consider an example of a medical diagnosis where physicians have to determine if a certain type of cancer (breast-cancer) exists or not (Mangasarian et al., 1995). Physicians have access to previous examples, where each example represents certain symptoms and the outcome the symptoms represent, i.e. a benign or malign case. Let us suppose that each example represents the following symptoms: clump thickness, uniformity of cell size, uniformity of cell shape, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli, mitoses, and the outcome if the symptoms represent a cancer or not. In this case, the training data constitutes a 9-dimensional feature-vector. Based on the examples, the problem is to determine if a new case, that is, some set of symptoms, represents a case of cancer or not. As can be seen, in this case there is no natural partitioning among features. We can not treat two or more features together as one feature-vector and the remaining features as another feature-vector. The relationship among features has to be inferred by the learning algorithm.

Similarly, let us consider another example that of mushroom classification into edible and non-edible categories (Iba et al., 1988). To classify mushrooms, let us suppose that we have the following information about mushrooms: cap-shape, cap-surface, cap-color, bruises, odor, gill-attachment, gill-spacing, gill-size, gill-color, stalk-shape, stalk-root, stalk-surface-above-ring, stalk-surface-below-ring, stalk-color-above-ring, stalk-color-below-ring, veil-type, veil-color, ring-number, ring-type, spore-print-color, population, habitat. Like the breast cancer example, we do not have any information regarding feature partitioning. Perhaps we can treat features related to gill (gill-attachment, gill-spacing, gill-size, gill-color) as a separate feature-vector and features related to stalk (stalk-shape, stalk-root, stalk-surface-above-ring, stalk-surface-below-ring, stalk-color-above-ring, stalk-color-below-ring) as a separate feature-vector, in the hope that together the features are more correlated with the class label or with each other (since they represent the information about the same source).

Appendix B

Discriminant Analysis

In this appendix, linear and quadratic discriminant analysis are briefly explained. Let us consider a simple binary classification problem (classes are $c1$ and $c2$). From decision theory for classification (Berger, 1985; Bather, 2000), we know that we have to estimate the class posterior probabilities, that is $P(c = c1|X)$ and $P(c = c2|X)$. According to Bayes' theorem, posterior probability can be defined in terms of the likelihood function ($p_c(\vec{x})$) and the prior probability (π_c). Let us assume that the prior probabilities of each class are the same, that is:

$$\pi_c = \frac{1}{C}, \quad \forall\, c \tag{B.1}$$

It will be useful to write the class posterior probabilities in terms of the likelihood function and prior probability. For example, we can write the posterior probability of class c_1 as:

$$P(c = c1|X) = \frac{p_{c1}(\vec{x})\ \pi_{c1}}{\sum_{c=1}^{C} p_{c1}(\vec{x})\pi_c} \tag{B.2}$$

Following Bayes' theorem, we have written the posterior probability of class c_1 in equation B.2 in terms of the likelihood function and the prior probability. The term in the denominator is the normalization factor.

Let us model the likelihood function of each class as a multi-variate Gaussian distribution. Therefore, we can write:

$$p_{c1}(\vec{x}) = \frac{1}{(2\pi)^{P/2}|\Sigma|^{1/2}} e^{-\left(\frac{1}{2}(\vec{x}-\mu_{c1})^T \Sigma_{c1}^{-1}(\vec{x}-\mu_{c1})\right)} \tag{B.3}$$

In Linear Discriminant Analysis (LDA), we assume that all classes have the same covariance matrix, therefore, $\Sigma_c = \Sigma\ \forall c$. For our binary classification problem, we can write: $\Sigma_{c1} = \Sigma_{c2} = \Sigma$. For comparing the two classes, we can look at the log ratio of the posterior probabilities of the two classes. Based on equation B.2 we can write the log ratio as:

$$\log\left(\frac{P(c = c1|\vec{x})}{P(c = c2|\vec{x})}\right) = \log\left(\frac{p_{c1}(\vec{x})}{p_{c2}(\vec{x})}\right) + \log\left(\frac{\pi_{c1}}{\pi_{c2}}\right) \tag{B.4}$$

Let us compute the log ratio of the likelihood functions in equation B.4. Based on equation B.3 we can write:

$$\begin{aligned}
\log\left(\frac{p_{c1}(\vec{x})}{p_{c2}(\vec{x})}\right) &= \log\left(\frac{(2\pi)^{P/2}|\Sigma_{c2}|^{1/2}}{(2\pi)^{P/2}|\Sigma_{c1}|^{1/2}}\frac{e^{-\frac{1}{2}(\vec{x}-\mu_{c1})^T\Sigma_{c1}^{-1}(\vec{x}-\mu_{c1})}}{e^{-\frac{1}{2}(\vec{x}-\mu_{c2})^T\Sigma_{c2}^{-1}(\vec{x}-\mu_{c2})}}\right) \qquad \text{(B.5)}\\
&= -\frac{1}{2}\Sigma^{-1}\left((\vec{x}-\mu_{c1})^T(\vec{x}-\mu_{c1})-(\vec{x}-\mu_{c2})^T(\vec{x}-\mu_{c2})\right)\\
&\quad (\text{as } \Sigma_{c1}=\Sigma_{c2}=\Sigma)\\
&= -\frac{1}{2}\Sigma^{-1}\left(-2\vec{x}^T\mu_{c1}+2\vec{x}^T\mu_{c2}+\mu_{c1}^T\mu_{c1}-\mu_{c2}^T\mu_{c2}\right)\\
&= -\frac{1}{2}\Sigma^{-1}\left(2\vec{x}^T(\mu_{c1}-\mu_{c2})+\mu_{c1}^T\mu_{c1}-\mu_{c1}^T\mu_{c2}+\mu_{c2}^T\mu_{c1}-\mu_{c2}^T\mu_{c2}\right)\\
&= -\frac{1}{2}\Sigma^{-1}\left(2\vec{x}^T(\mu_{c1}-\mu_{c2})+(\mu_{c1}^T+\mu_{c2}^T)(\mu_{c1}+\mu_{c2})\right)\\
&= \Sigma^{-1}\vec{x}^T(\mu_{c1}-\mu_{c2})-\frac{1}{2}\Sigma^{-1}(\mu_{c1}^T+\mu_{c2}^T)(\mu_{c1}+\mu_{c2})
\end{aligned}$$

It can be seen that the assumption of an equal covariance matrix for each class has caused the normalization factors and quadratic part in the exponent to cancel. Based on above derivations, we can write equation B.4 as:

$$\log\left(\frac{P(c=c1|\vec{x})}{P(c=c2|\vec{x})}\right) = \Sigma^{-1}\vec{x}^T(\mu_{c1}-\mu_{c2})-\frac{1}{2}\Sigma^{-1}(\mu_{c1}^T+\mu_{c2}^T)(\mu_{c1}+\mu_{c2})+\log\left(\frac{\pi_{c1}}{\pi_{c2}}\right) \qquad \text{(B.6)}$$

Equation B.6 implies that all the points in the input space satisfying the condition $P(c = c1|\vec{x}) = P(c = c2|\vec{x})$ represent the decision boundary. The linear discriminant function for class $c1$ can be written as:

$$\delta_{c1}(\vec{x}) = \Sigma^{-1}\vec{x}^T\mu_{c1}-\frac{1}{2}\mu_{c1}^T\Sigma^{-1}\mu_{c1}+\log\pi_{c1} \qquad \text{(B.7)}$$

and the equivalent decision rule $G(\vec{x})$ is:

$$\underset{c}{\operatorname{argmax}}\,\delta_c(\vec{x}) \qquad \text{(B.8)}$$

For quadratic discriminant analysis (QDA), we cannot assume the same covariance matrix for all classes. Hence, we cannot simplify the log ratio of the likelihood term in equation B.4 as we did in equation B.5. That is, we can not cancel out the quadratic part in the exponent. Therefore, when assuming that $\Sigma_{c1} \neq \Sigma_{c2}$, we can write equation B.5 as:

$$\begin{aligned}
\log\left(\frac{p_{c1}(\vec{x})}{p_{c2}(\vec{x})}\right) &= \log\left(\frac{(2\pi)^{P/2}|\Sigma_{c2}|^{1/2}}{(2\pi)^{P/2}|\Sigma_{c1}|^{1/2}}\frac{e^{-\frac{1}{2}(\vec{x}-\mu_{c1})^T\Sigma_{c1}^{-1}(\vec{x}-\mu_{c1})}}{e^{-\frac{1}{2}(\vec{x}-\mu_{c2})^T\Sigma_{c2}^{-1}(\vec{x}-\mu_{c2})}}\right)\\
&= \log\left(\frac{|\Sigma_{c2}|^{1/2}}{|\Sigma_{c1}|^{1/2}}\frac{e^{-\frac{1}{2}(\vec{x}-\mu_{c1})^T\Sigma_{c1}^{-1}(\vec{x}-\mu_{c1})}}{e^{-\frac{1}{2}(\vec{x}-\mu_{c2})^T\Sigma_{c2}^{-1}(\vec{x}-\mu_{c2})}}\right)\\
&= -\frac{1}{2}\log\left(\frac{\Sigma_{c1}}{\Sigma_{c2}}\right)-\frac{1}{2}(\vec{x}-\mu_{c1})^T\Sigma_{c1}^{-1}(\vec{x}-\mu_{c1})-\frac{1}{2}(\vec{x}-\mu_{c2})^T\Sigma_{c2}^{-1}(\vec{x}-\mu_{c2})
\end{aligned}$$

Equation B.4 will become:

$$\log\left(\frac{P(c=c1|\vec{x})}{P(c=c2|\vec{x})}\right) = -\frac{1}{2}\log\left(\frac{\Sigma_{c1}}{\Sigma_{c2}}\right) - \frac{1}{2}(\vec{x}-\mu_{c1})^T\Sigma_{c1}^{-1}(\vec{x}-\mu_{c1}) - \frac{1}{2}(\vec{x}-\mu_{c2})^T\Sigma_{c2}^{-1}(\vec{x}-\mu_{c2}) + \log\left(\frac{\pi_{c1}}{\pi_{c2}}\right) \tag{B.9}$$

The quadratic discriminant function based on equation B.9 can be written as:

$$\delta_{c1}(\vec{x}) = -\frac{1}{2}\log|\Sigma_{c1}| - \frac{1}{2}(\vec{x}-\mu_{c1})^T\Sigma^{-1}(\vec{x}-\mu_{c1}) + \log\pi_{c1} \tag{B.10}$$

The decision boundary between classes $c1$ and $c2$ is described as a quadratic function such that: $\delta_{c1}(\vec{x}) = \delta_{c2}(\vec{x})$. Like LDA, an equivalent decision rule for QDA is given in equation B.8.

Appendix C

Comparative Study of Dense versus Sparse Features

There has been a significant debate over the use of features extracted at some small set of interest points (sparse features) compared with features extracted at points on a regular grid (dense features). The first approach is to identify some interest points on the object and compute scale and rotation invariant features at these points to represent the object. These are typically known as 'sparse features' in contrast to 'dense features', which are computed at grid points (figure C.1). The use of interest points has the advantage that features computed are scale or rotation invariant, but of course a disadvantage because of the sparsity and the consequent inability to capture all the information in the image. In this work, a comparative analysis of dense and sparse features for object categorization problem is presented.

To compare the performance of dense and sparse features, the *Caltech-101 Object Database* (2006) is used. Multi-class classification is done by training binary classifiers using the one-versus-all rule. 15 training images per category were used for training. The rest of the images were used for testing. For efficiency, the number of testing images was limited to 50. It is noteworthy that some categories were quite small and we ended up with a single test image per category. The performance results are given as the average of per-class recognition performance. An alternate performance measure could be to record

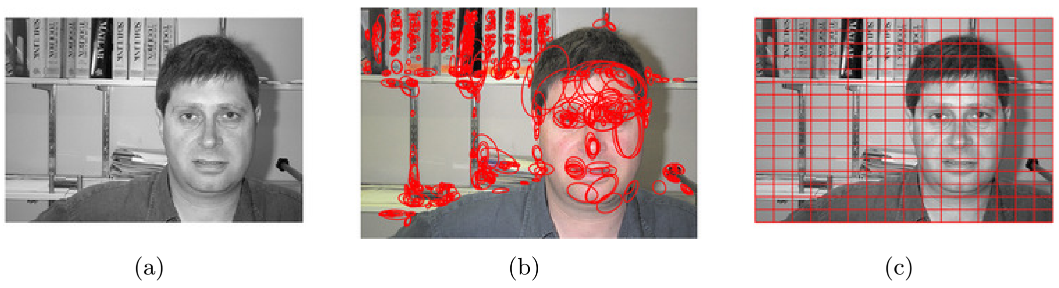


Figure C.1: Images showing the interest points and the grid points over which the features are to be calculated. Figure C.1(a): original image, Figure C.1(b): shows the interest points along with area at which the features are to be computed, Figure C.1(c): shows the grid points at which features are to be computed.

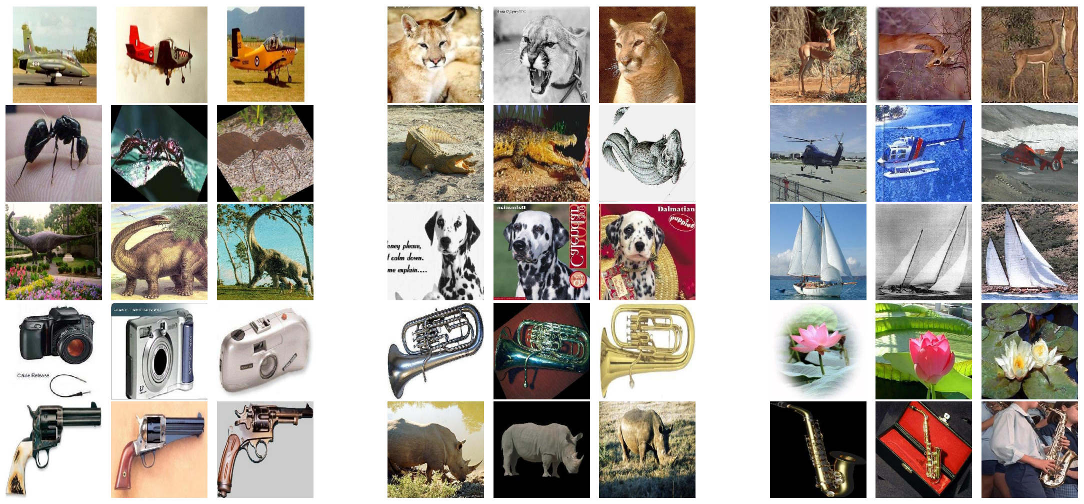


Figure C.2: Some example images from Caltech101 object database (3 images per category). Caltech101 database consists of 101 object categories. Each category contains 31 to 800 images. The database is ideal for testing categorization tasks as the objects are not cluttered and mostly present at the center of the image.

the percentage of all correctly classified test images. Since, the size of each category varies a lot, such a measure may have been biased and, therefore, is not used.

The pyramid-based histogram computation approach described in section 10.4.1 was used to compute the dense features. SIFT features (Lowe, 1999) were extracted at points on a regular grid of size 20×20. At each grid point, the features were extracted at scales of 4, 8, 16, 24 and 32 pixels. Once the features were extracted, they were clustered into a codebook of size 200 using the k-means clustering algorithm. Once the codebook was computed, the features were vector-quantized. A pyramid-based approach was used to compute the feature-vector representing the image. Sparse features were calculated using the same approach as for dense features, except the SIFT features were extracted on some interest points detected using Harris-Affine interest point detector. For both dense and sparse features, a maximum pyramid level of three was used. Hence both dense and sparse feature-sets were constituted of four feature-vectors of length 200, 800, 3200 and 12800 each.

The resulting dense and sparse feature-sets are classified using HML2-CF scheme as described in chapter 10 using the CRAB version of Boosting classifier (section 10.1.2). The following variants of HML2-CF scheme are compared:

- **HML2-CF, equal weighting:** The classifiers trained for each feature-vector are combined to predict the class label. All classifiers are given equal weight. The classification results are given in table C.1.

- **HML2-CF, priority weighting:** The classifiers trained for each feature-vector are combined to predict the class label. A priority weighting scheme is used. The classifiers trained at higher pyramid levels are given less weights as compared to classifiers trained at lower pyramid levels (equation 10.18). The classification results are given in table C.1.

Pyramid Level	0	1	2	3
HML2-CF (equal weighting)				
Dense Features	33.19 ± 0.25	46.25 ± 0.15	53.42 ± 0.10	53.13 ± 0.10
Sparse Features	15.28 ± 1.37	20.89 ± 0.39	24.19 ± 0.20	24.41 ± 0.34
HML2-CF (priority weighting)				
Dense Features	33.19 ± 0.25	46.30 ± 0.19	53.99 ± 0.15	54.20 ± 0.25
Sparse Features	15.28 ± 1.37	20.30 ± 0.50	24.40 ± 0.08	24.50 ± 0.17

Table C.1: Mean correctness rate of dense and sparse features with HML2-CF (equal and priority weighting) on Caltech101 object database. Mean correctness rate and standard deviation of the results are reported over ten experiments with random training and testing data.

It can be seen from table C.1 that the dense features resulted in much better performance as compared to the sparse features. With both dense and sparse features, the best performance resulted at pyramid level 3 with HML2-CF scheme and priority weighting. But the performance is 54.20% in the case of dense feature as compared to meager 24.50% with sparse features. This suggest that even though dense features are computationally expensive, they can result in a huge performance gain.

Appendix D

Support Vector Machines

An SVM is a classifier that computes an optimal hyper-plane that separates the data into two classes such that the distance between the hyper-plane and the training data (also known as the margin) is maximized. SVM are based on Support Vector Classifier (SVC), which are actually optimized linear classifiers obtained in a way such that their margin are maximized. Therefore, before delving into the details of SVM and the optimization strategy behind them, simple linear classifiers will be briefly discussed and we will describe how the SVC formulation originates when an optimal linear classifier is desired. In figure D.1, a simple classification problem with two linearly separable classes is shown. The decision boundaries of some of the classifiers (linear) that can classify the data correctly are shown as blue lines. It can be seen from the figure that there are many linear classifiers possible as solutions to this classification problem. Which classification solution is the best? Clearly there is a need to determine the best classifier for this problem. In machine learning, a classifier that has the capability of generalizing well to unseen data is more desirable. The SVC formulation seeks such a solution. The discussion in the remainder of this appendix (closely) follows the development in Hastie et al. (2001).

A linear classifier can be parameterized by two variables: β_0 and $\vec{\beta}$. For example, in figure D.1 each classifier is written as $\beta_0+\beta_1x_1+\beta_2x_2$, and the sign is used to determine the class of the query point. The classifier will return zero for any data point on the decision surface. Such linear combinations of input features $(x_1,\ldots,x_n)$ were called perceptrons in the early 70's and were the basis of every neural network techniques (Minsky and Papert, 1969). Given a training data set $\{(\vec{x}_1,y_1),\ldots,(\vec{x}_n,y_n)\}$, a point with label $y_i=+1$ will be misclassified if the perceptron outputs is < 0, that is $\beta_0+\vec{\beta}^T\vec{x}<0$ and similarly for point with label $y_i=-1$, it is deemed misclassified if $\beta_0+\vec{\beta}^T\vec{x}>0$. Rosenblatt (1958) addressed the problem of finding an optimal hyper-plane by minimizing the following objective function using stochastic gradient descent:

$$-\sum_{i=1}^{N1} y_i(\beta_0+\vec{\beta}^T\vec{x}) \tag{D.1}$$

where $N1$ is the number of misclassified points. There are at least three problems with Rosenblatt's (1958) formulation of optimal linear classifiers. First, when the data is linearly

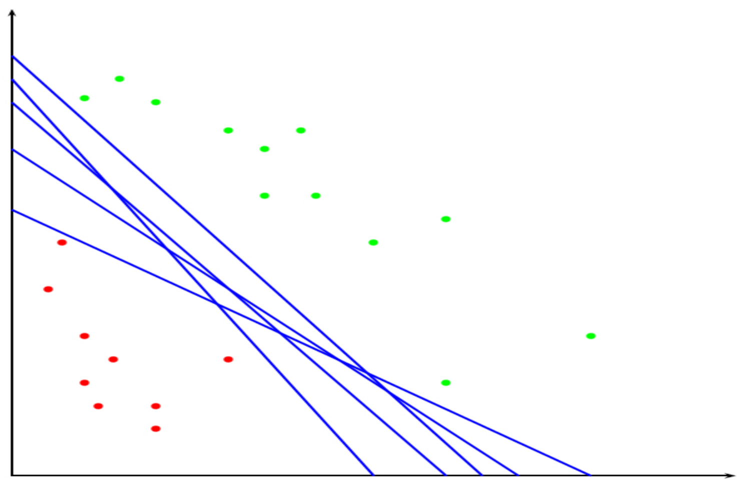


Figure D.1: A toy classification problem representing the case where data can be well separated by many linear classifiers. The class A is shown in red and class B is shown in green. The blue lines represents the decision boundaries of some of the classifiers that can separate the data well.

separable, there may be many solutions. As each solution will depend on the initialization of the algorithm parameters, a unique solution can not be guaranteed. Secondly, if the problem is not linearly separable, the algorithm will not converge. Finally, the number of steps required to achieve convergence can be so large that we may not be able to determine if the problem is non-separable or the algorithm is simply too slow to converge.

In figure D.2, some linear algebra that serves as the basis for linear classification is reviewed. The red line represents a linear classifier specified by $y = \beta_0 + \vec{\beta}^T \vec{x}$. For any two data points $\vec{x}_a$ and $\vec{x}_b$ on the decision surface, the equation of the decision surface is $y = 0$, therefore, $\vec{\beta}^T(\vec{x}_a - \vec{x}_b) = 0$. This suggests that the vector $\vec{\beta}$ is orthogonal to the decision boundary y. Similarly for any data point $\vec{x}_a$ on the decision surface, we can write $\vec{\beta}^T(\vec{x}_a) = -\beta_0$. Using the distance formula, the distance $\mathcal{C}$ of any point $\vec{x}$ from the decision surface can be calculated as:

$$\mathcal{C} = \frac{1}{\|\vec{\beta}\|}(\vec{\beta}^T \vec{x} + \beta_0) = \frac{y(\vec{x})}{\|\vec{\beta}\|} \tag{D.2}$$

The margin of a classifier is defined as double the perpendicular distance of any closest data point to the boundary surface. Since, $\mathcal{C}$ in equation D.2 is the perpendicular distance of $\vec{x}$ to the decision surface, the margin of the classifier is actually $2\mathcal{C}$. Intuitively it can be seen that maximizing this margin may result in better classification.

To develop a classifier that results in maximizing the margin, we need to rephrase equation D.1 in the form of the following optimization problem:

$$\max_{\beta_0, \vec{\beta}} \; \mathcal{C}$$
$$\text{subject to} \quad \tfrac{1}{\|\vec{\beta}\|} y_i(\vec{\beta}^T \vec{x}_i + \beta_0) \geq \mathcal{C}, \quad i = 1, 2, \ldots, N \tag{D.3}$$

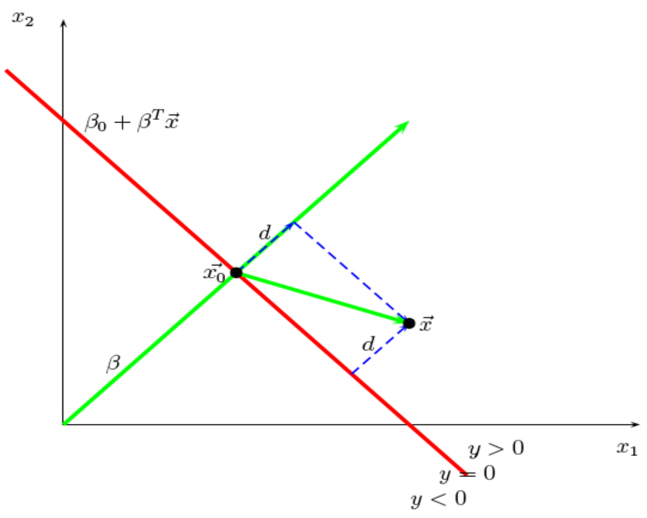


Figure D.2: Review of some basic linear algebra concepts required for deriving maximal margin classifier.

The conditions in equation D.3 ensure that each point is at least distance $\mathcal{C}$ from the hyperplane that is defined by β_0 and $\vec{\beta}$. Instead of maximizing $\mathcal{C}$ we can also minimize $1/\mathcal{C}$. Equation D.3 also has another important interpretation. It suggests that even if we re-scale β_0 and $\vec{\beta}$, the distance of the hyperplane to each data point is going to remain same. Hence, we can set $1/\mathcal{C} = \|\vec{\beta}\|$ and reformulate constraints as $\frac{1}{\|\vec{\beta}\|}y_i(\vec{\beta}^T\vec{x}_i+\beta_0) \geq 1, \;\; i = 1, 2, \ldots, N$ in equation D.3. Therefore, equation D.3 can be written as:

$$\begin{aligned} &\min_{\beta,\vec{\beta}} \;\; \frac{1}{2}\|\vec{\beta}\|^2 \\ \text{subject to} \;\; &\frac{1}{\|\vec{\beta}\|}y_i(\vec{\beta}^T\vec{x}_i + \beta_0) \geq 1, \;\; i = 1, 2, \ldots, N \end{aligned} \tag{D.4}$$

This results in a quadratic programming problem, since the objective function is quadratic with linear inequalities constraints and can be solved efficiently.

So far we have assumed that the classes are linearly separable. For those cases where the two classes overlap, we can still maximize $\mathcal{C}$ in equation D.3, but we need to build a mechanism to allow some data points to be on the wrong side of margin. This can be done by introducing *slack variables* $\xi_1, \xi_2, ...\xi_n$ (one slack variable for each point in the data set), each of which is actually a penalty applied to those data points that are present on the wrong side of the margin. That is, $\xi_i = 0$ for those data points that are on the right side of the margin (correctly classified). Whereas, $\xi_i = |y_i - f(\vec{x}_i)|$ for the data points on the wrong side of the margin (misclassified points). Figure D.3 illustrates a simple margin-based classification scenario with slack variables constrained to satisfy $\xi_i \geq 0$. The data points correctly classified in figure D.3 have $\xi = 0$. Those data points that are on the decision surface; $\xi_i = 1$ as $f(\vec{x}_i) = 0$. Points lying inside the margin but on the correct side of decision boundary satisfy the constraint $0 < \xi_i \leq 1$, whereas misclassified points satisfy constraint $\xi_i > 1$.

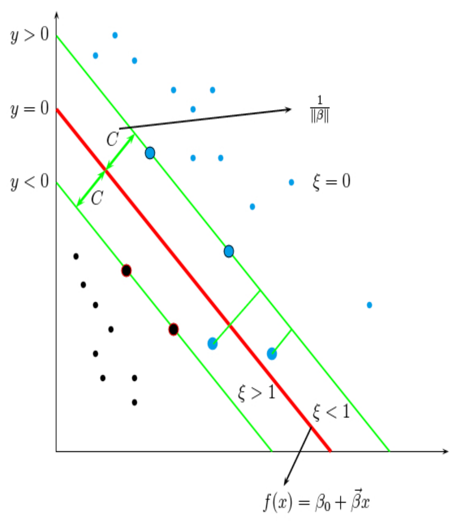


Figure D.3: Illustration of slack variables in margin-based classification. The decision boundary is shown in red, while the green lines bound the maximal margin of width $2\mathcal{C} = 2/\|\vec{\beta}\|$.

It can be seen that $\sum_i \xi_i$ is an upper bound on the number of misclassified training points. Hence by bounding $\sum_i \xi_i$ we bound the total number of training points that can be on the wrong side of decision boundary. This introduction of slack variables is known as 'soft margin', as it allows some data points to be on the wrong side of the decision boundary, in contrast to the hard margin where no data points are allowed to be misclassified. With the incorporation of slack variables, the constraints in equation D.4 can be modified as:

$$\min_{\beta,\vec{\beta}} \frac{1}{2}\|\vec{\beta}\|^2$$
$$\text{subject to} \quad y_i(\vec{\beta}^T\vec{x}_i + \beta_0) \geq (1-\xi_i),\ \forall i,\ \xi_i \geq 0,\ \textstyle\sum_i \xi_i \leq \mathcal{C} \tag{D.5}$$

The objective function and constraints in equation D.5 is the standard SVC formulation for non-separable data. For computational efficiency we can write equation D.5 as:

$$\min_{\beta_0,\vec{\beta}} \frac{1}{2}\|\vec{\beta}\|^2 + \mathcal{C}\sum_i^N \xi_i$$
$$\text{subject to} \quad y_i(\vec{\beta}^T\vec{x}_i + \beta_0) \geq (1-\xi_i),\ \forall i,\ \xi_i \geq 0, \tag{D.6}$$

The Lagrange (primal) of equation D.6 is:

$$\mathcal{L}_p(\beta_0, \vec{\beta}, \alpha) = \frac{1}{2}\|\vec{\beta}\|^2 + \mathcal{C}\sum_i^N \xi_i - \sum_i^N \alpha_i[y_i(\vec{x}_i^T\vec{\beta} + \beta_0)] - \sum_i^N a_i\xi_i \tag{D.7}$$

Taking partial derivative of $\mathcal{L}_p$ in equation D.7 with respect to $\beta_0, \vec{\beta}$ and ξ_i and setting the respective derivatives equal to zero, we get the following results:

$$\beta = \sum_i^N \alpha_i y_i \vec{x}_i, \;\; 0 = \sum_i^N \alpha_i y_i, \;\; \text{and } \alpha_i = \mathcal{C} - a_i \; \forall i \tag{D.8}$$

By substituting equation D.8 in equation D.7, we get the following Lagrangian (Wolfe) dual objective function:

$$\mathcal{L}_p(\beta_0, \vec{\beta}, \alpha) = \sum_i^N \alpha_i - \frac{1}{2} \sum_i^N \sum_j^N \alpha_i \alpha_j y_i y_j \vec{x}_i^T \vec{x}_j \tag{D.9}$$

This again represents a quadratic programming problem. An optimal value of α can be found from equation D.9 in the lights of Karush-Kuhn-Tucker (KKT) constraints which are:

$$\alpha_i \geq 0, \;\; a_i \geq 0, \;\; \xi_i \geq 0 \tag{D.10}$$

$$a_i \xi_i = 0 \tag{D.11}$$

$$\alpha_i [y_i(\vec{\beta}\vec{x}_i + \beta_0) - (1 - \xi_i)] = 0 \tag{D.12}$$

$$y_i(\vec{\beta} + \beta_0) - (1 - \xi_i) \geq 0 \tag{D.13}$$

for $i = 1, 2, \ldots, N$. From equation D.9, we can find the solution, that is the parameters $\vec{\beta}^*$ of an optimal hyperplane. We can write $\vec{\beta}^*$ as:

$$\vec{\beta}^* = \sum_i^N \alpha_i^* y_i \vec{x}_i \tag{D.14}$$

It should be noted that α will be non-zero for those data points ($\vec{x}_i$) which meet the constraints in equation D.12. These data points ($\vec{x}_i$) are known as the *support vectors* as $\vec{\beta}^*$ is only represented in terms of these data points. As can be seen that obtaining the optimal decision boundary by maximizing the dual in equation D.9 is a much simpler optimization problem (convex programming) than maximizing the primal in equation D.7. Therefore, solving for $\vec{\beta}^*$ and β_0^* in equation D.9 gives the following decision function:

$$y(\vec{x}) = \sum_i^N \vec{\beta}^{*T} \vec{x} + \beta_0^* \tag{D.15}$$

The classifier in equation D.15 (depicting both the separable and non-separable scenarios) is a linear classifier in the input space. Such a formulation is known as SVC. It is common to project the input data to a high dimensional space and find linear boundaries (SVC for example) in that space. This results in a non-linear decision surface in the input space. Non-linear decision boundaries in the input space often lead to better class separation. Expanding feature space using basis functions such as splines and polynomials is common in machine learning. SVM is the incorporation of this idea in SVC. An input space is embedded into an expanded feature space and SVC is used to find linear boundaries in that space. It should be noted that with sufficient basis functions data can be completely separated, which will often induce over-fitting. Therefore, SVM has to deal with the issue of over-fitting. We can modify the Lagrangian dual form in equation D.9 as:

$$\mathcal{L}_p(\beta_0, \vec{\beta}, \alpha) = \sum_{i}^{N} \alpha_i - \frac{1}{2} \sum_{i}^{N} \sum_{j}^{N} \alpha_i \alpha_j y_i y_j \langle \Phi(\vec{x}_i), \Phi(\vec{x}_j) \rangle \tag{D.16}$$

where $\Phi(\vec{x})$ is the transformation of data. As can be seen, equation D.16 calculates the dot product in the transformed space. The beauty of the SVM formulation is that we do not need to take care of actual transformation and calculation of the dot product. All we require is the knowledge of the kernel function ($k(.,.)$) which implicitly computes the inner product in the transformed space, for example:

$$k(\vec{x}_i, \vec{x}_j) = \langle \Phi(\vec{x}_i), \Phi(\vec{x}_j) \rangle \tag{D.17}$$

It should be noted that k has to be a positive-semi-definite function. Following are typical kernels used with SVM:

$$\begin{aligned} \text{d-degree Polynomial :} \quad & k(\vec{x}_i, \vec{x}_j) = (1 + \langle \vec{x}_i, \vec{x}_j \rangle)^d \\ \text{Radial Basis Function (RBF) :} \quad & k(\vec{x}_i, \vec{x}_j) = \exp(-\|\vec{x}_i - \vec{x}_j\|^2)/\sigma^2) \end{aligned} \tag{D.18}$$

In the light of the above discussions on kernels, we can write equation D.15 as:

$$y(\vec{x}) = \sum_{i}^{N} \alpha_i^* y_i k(\vec{x}_i, \vec{x}) + \beta_0^* \tag{D.19}$$

In summary, it can be seen that an SVM classifier has two tuning parameters: the misclassification penalty $\mathcal{C}$ and the kernel function $k(.,.)$. A large value of $\mathcal{C}$ will discourage positive ξ_i and will lead to a 'wiggly' decision surface in the original space. On the other hand, a smaller value of $\mathcal{C}$ will encourage a small value of $\|\vec{\beta}\|$; this will make $y(\vec{x})$ more smooth.

Appendix E

Gaussian Processes

The GP is a regression/classification technique that has been very popular in machine learning. Though interest in its use was revived by the work of Rasmussen and Williams (2006), it should be noted that GP estimation under the name of the 'kriging' estimate has been used in spatial statistics for a long time (Cressie, 1993). A general broad overview of GP framework is provided in this appendix. Maximum likelihood estimation, linear models of regression, and Bayesian methods will also be explained to build a foundation for GP estimation. The discussion in the remainder of this appendix follows the development in Bishop (2006).

In a typical machine learning scenario, we are given some training data $\mathbf{X} = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N\}$ and we are interested in predicting the underlying process (or estimating the function) that generated these observations. For convenience, we assume that the observations are drawn independently from some probability distribution. Since a probability distribution can be specified completely by some parameters, the problem of estimating an unknown function transforms into the problem of estimating the parameters for that probability distribution.

Let us assume that the observations are drawn independently from a Gaussian distribution. The univariate Gaussian distribution with mean μ and variance σ^2 and the multivariate (P-dimensional) Gaussian distribution with mean $\vec{\mu}$ and covariance Σ can be written as:

$$\begin{aligned} \mathcal{N}(x|\mu,\sigma^2) &= \frac{1}{(2\pi)^{1/2}(\sigma^2)^{1/2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) && \text{(E.1)} \\ \mathcal{N}(\vec{x}|\vec{\mu},\Sigma) &= \frac{1}{(2\pi)^{P/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\vec{x}-\vec{\mu})^T\Sigma^{-1}(\vec{x}-\vec{\mu})\right) && \text{(E.2)} \end{aligned}$$

respectively. Since the data is independent and is identically drawn from the distribution $\mathcal{N}(\vec{x}|\vec{\mu},\Sigma)$, we can write the likelihood function as:

$$p(\mathbf{X}|\vec{\mu},\Sigma) = \prod_{n=1}^{N} \mathcal{N}(\vec{x}_n|\vec{\mu},\Sigma) \tag{E.3}$$

We can estimate the parameters for the Gaussian distribution by maximizing the likelihood function in equation E.3. Such an approach is known as the maximum likelihood method. Maximum likelihood views the parameters as quantities whose value is fixed but unknown. The best estimate of their values is the one that maximizes the probability of obtaining samples that are actually observed. Taking the log of equation E.3, we have:

$$\ln p(\mathbf{X}|\vec{\mu},\Sigma) = -\frac{N}{2}\ln(2\pi) - \frac{N}{2}\ln|\Sigma| - \frac{1}{2}\sum_{n=1}^{N}(\vec{x}_n-\vec{\mu})^T\Sigma^{-1}(\vec{x}_n-\vec{\mu}) \tag{E.4}$$

Let us denote ln $p(\mathbf{X}|\vec{\mu},\Sigma)$ with J which is actually an objective function that needs to be maximized to obtain the parameters of the distribution. It can be seen that maximizing J in equation E.4 with respect to $\vec{\mu}$ leads to:

$$\vec{\mu}^* = \frac{1}{N}\sum_{n=1}^{N}\vec{x}_n \tag{E.5}$$

which is actually the sample mean. Similarly maximizing J with respect to Σ, we can get the maximum likelihood solution for the covariance, that is:

$$\Sigma^* = \frac{1}{N}\sum_{n=1}^{N}(\vec{x}_n-\vec{\mu}^*)^T(\vec{x}_n-\vec{\mu}^*) \tag{E.6}$$

Let us denote the probability distribution parameters as θ. So far, we have assumed a Gaussian distribution, hence $\theta = \{\vec{\mu},\Sigma\}$. But, in general, the sample data can assumed to be generated by any underlying probability distribution. Therefore, the objective function J can be written as $p(\mathbf{X}|\theta)$. In short, the maximum likelihood estimate of θ is the value θ^* that maximizes the objective function $p(\mathbf{X}|\theta)$. Sometimes, we have information about the distribution parameters (θ). An alternative is the maximum posterior estimation (MAP), which corresponds to finding the value of θ that maximizes $p(\mathbf{X}|\theta)p(\theta)$. Of course, this technique has its roots in Bayesian estimation

Before introducing Bayesian methods, we should consider a simple linear model for regression:

$$y(\vec{x},\mathbf{w}) = w_0 + w_1x_1 + w_2x_2 + \cdots + w_Px_P \tag{E.7}$$

$$= w_0 + \sum_{j=1}^{M-1} w_j\phi_j(\vec{x}) \tag{E.8}$$

$$= \mathbf{w}^T\phi(\vec{x}) \tag{E.9}$$

where $\mathbf{w} = (w_0, w_1, ...w_{M-1})^T$ and $\phi = (1, \phi_1, ...\phi_{M-1})^T$. Note M represents the total number of parameters in the model and is equal to $P+1$ where P is the number of features. In general the samples or observations are noisy, therefore, we need to cater for this noise in the regression model in equation E.7. We can modify $y(\vec{x},\mathbf{w})$ in equation E.7 as:

$$t = y(\vec{x},\mathbf{w}) + \epsilon \tag{E.10}$$

where ϵ is small Gaussian noise around $\vec{x}$ and t represent the actual value of the function at point $\vec{x}$. Therefore, to make predictions about the value of t, we are interested in predicting the values of the Gaussian distribution with a mean of $y(\vec{x}, \mathbf{w})$ and the variance of ν, we can write:

$$p(t|\vec{x}, \mathbf{w}, \nu) = \mathcal{N}(t|y(\vec{x}, \mathbf{w}), \nu) \tag{E.11}$$

Again, as the observations are independent and identically drawn, if $\mathbf{t} = \{t_1, \ldots, t_n\}$, we can write:

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \nu) = \prod_{n=1}^{N} \mathcal{N}(t_n|y(\vec{x}_n, \mathbf{w}), \nu) \tag{E.12}$$

Let us take the log of both sides of equation E.12. Using equation E.7, we can write

$$\begin{aligned} \ln p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \nu) &= \sum_{n=1}^{N} \mathcal{N}(t_n|\mathbf{w}^T \phi(\mathbf{x_n}), \nu) & \text{(E.13)} \\ &= \frac{N}{2}\ln(\nu) - \frac{N}{2}\ln(2\pi) - \frac{\nu}{2}\sum_{n=1}^{N} (t_n - \mathbf{w}^T \phi(\vec{x}_n))^2 & \text{(E.14)} \end{aligned}$$

Now, maximizing the objective function ($\ln p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \nu)$) in equation E.14 with respect to $\mathbf{w}$ and ν, we can write:

$$\begin{aligned} \mathbf{w}^* &= (\Phi^T \Phi)^T \Phi^T \mathbf{t} & \text{(E.15)} \\ \nu^* &= \frac{1}{N}\sum_{n=1}^{N} (t_n - \mathbf{w}^* \phi(\vec{x}_n))^2 & \text{(E.16)} \end{aligned}$$

where

$$\Phi = \begin{pmatrix} \phi_0(x_1) & \phi_1(x_1) & \cdots & \phi_{M-1}(x_1) \\ \phi_0(x_2) & \phi_1(x_2) & \cdots & \phi_{M-1}(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_N) & \phi_1(x_N) & \cdots & \phi_{M-1}(x_N) \end{pmatrix} \tag{E.17}$$

is a $M \times N$ dimensional matrix, also known as the design matrix. Now, we can write equation E.11 in terms of $\mathbf{w}^*$ and ν^* from equation E.15 and E.16 as:

$$p(t|\vec{x}, \mathbf{w}^*, \nu^*) = \mathcal{N}(t|y(\vec{x}, \mathbf{w}^*), \nu^*) \tag{E.18}$$

The quantity $\mathbf{w}^* = (\Phi^T \Phi)^T \Phi^T \mathbf{t}$ is also known as the 'Moore-Penrose pseudo-inverse' of the matrix Φ.

Bayesian techniques view the parameter $\mathbf{w}$ as a random variable having some known prior distribution. Let us suppose that the prior distribution over parameter $\mathbf{w}$ is Gaussian, with mean 0 and covariance of Σ_p, so we can write:

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|0, \Sigma_p) = \frac{1}{(2\pi)^{P/2}|\Sigma_p|^{1/2}} \exp\left(\frac{1}{2}\mathbf{w}\Sigma_p\mathbf{w}^T\right) \tag{E.19}$$

If we take the log on both sides of equation E.19, we can write:

$$\ln p(\mathbf{w}) = \ln\left(\frac{1}{(2\pi)^{P/2}|\Sigma_p|^{1/2}}\right) + \frac{1}{2}\mathbf{w}\Sigma_p\mathbf{w}^T \tag{E.20}$$

We are interested in following equations:

$$\begin{aligned} p(\mathbf{w}|\mathbf{t}) &= p(\mathbf{t}|\mathbf{w})p(\mathbf{w}) && \text{(E.21)} \\ \ln p(\mathbf{w}|\mathbf{t}) &= \ln p(\mathbf{t}|\mathbf{w}) + \ln p(\mathbf{w}) && \text{(E.22)} \end{aligned}$$

Plugging ln $p(\mathbf{t}|\mathbf{w})$ from equation E.14 and ln $p(\mathbf{w})$ from equation E.20 (by using only terms that involve $\mathbf{w}$), we can write ln $p(\mathbf{w}|\mathbf{t})$ as:

$$\begin{aligned} \ln p(\mathbf{w}|\mathbf{t}) &= -\frac{\nu}{2}\sum_{n=1}^{N}(t_n - \mathbf{w}^T\phi(x_n))^2 - \frac{1}{2}\mathbf{w}\Sigma_p\mathbf{w}^T && \text{(E.23)} \\ &= -\frac{\nu}{2}(\mathbf{t} - \Phi\mathbf{w})^T(\mathbf{t} - \Phi\mathbf{w}) - \frac{1}{2}\mathbf{w}\Sigma_p\mathbf{w}^T && \text{(E.24)} \end{aligned}$$

Now maximizing the objective function ln $p(\mathbf{w}|\mathbf{t})$ in equation E.23 with respect to $\mathbf{w}$, we can write:

$$\begin{aligned} \mathbf{w}^* &= \frac{\nu\Phi^T\mathbf{t}}{\Sigma_p^{-1} + \nu\Phi^T\Phi} && \text{(E.25)} \\ &= \nu\mathbf{S}_N\Phi^T\mathbf{t}, \text{where } \ \mathbf{S}_N = (\Sigma_p^{-1} + \nu\Phi^T\Phi)^{-1} && \text{(E.26)} \end{aligned}$$

Plugging the value of $\mathbf{w}^*$ from equation E.25 in equation E.9 reveals some interesting results.

$$\begin{aligned} y(\mathbf{x}, \mathbf{w}^*) &= \mathbf{w}^{*T}\phi(\mathbf{x}) && \text{(E.27)} \\ &= \nu\phi(x)^T\mathbf{S}_N\Phi\mathbf{t} && \text{(E.28)} \\ &= \sum_{n=1}^{N}\nu\phi(x)^T\mathbf{S}_N\phi(x_n)t_n && \text{(E.29)} \\ &= \sum_{n=1}^{N}k(x, x_n)t_n \quad \text{where } \ k(x, x^{'}) = \nu\phi(x)^T\mathbf{S}_N\phi(x^{'}) && \text{(E.30)} \end{aligned}$$

The kernel $k(.,.)$ in equation E.30 is known as the ‘smoother matrix’ or the ‘equivalent kernel’. The regression functions which make predictions by taking the linear combinations of the data target values are known as the linear smoothers. This formulation of linear regression in terms of a kernel function suggests an alternative strategy. Instead of defining a set of basis functions, which implicitly determines an equivalent kernel, we can instead directly define a kernel density function and use this to make predictions for a new input data point $\vec{x}$, given the training data. This leads us to a practical framework for regression and classification called GP.

Let us consider the linear model of regression in equation E.9. It can be seen that the probability distribution over $\mathbf{w}$ as defined in equation E.19 induces a probability distribution over functions $y(\mathbf{x})$. Let us simplify equation E.9 and write it as:

$$\mathbf{y} = \mathbf{w}^T\mathbf{\Phi} \tag{E.31}$$

where $\mathbf{y} = y(\vec{x}_1), \ldots, y(\vec{x}_N)$ and $\mathbf{\Phi}$ is the design matrix. Since $\mathbf{y}$ is a linear combination of Gaussian distributed variables given by the elements of $\mathbf{w}$, it itself is Gaussian. We can find its mean and covariance as:

$$\begin{aligned} \mathrm{E}[\mathbf{y}] &= \mathbf{\Phi}\mathrm{E}[\mathbf{w}] \ = 0 && \text{(E.32)} \\ \mathrm{cov}[\mathbf{y}] &= \mathrm{E}[\mathbf{y}\mathbf{y}^T] \\ &= \mathbf{\Phi}\mathrm{E}[\mathbf{w}\Sigma_p\mathbf{w}^T]\mathbf{\Phi}^T && \text{(E.33)} \\ &= \mathbf{K} && \text{(E.34)} \end{aligned}$$

where $\mathbf{K}$ is the Gram matrix with elements:

$$K_{nm} = k(\vec{x}_n, \vec{x}_m) \tag{E.35}$$

where $k(.,.)$ is the kernel function. This is a particular example of GP. A GP is defined as a probability distribution over functions $y(.)$ such that the set of values $y(\vec{x})$ evaluated at some points jointly have a Gaussian distribution.

Let us consider how we can make prediction using the GP formulation. In summary, from the definition of GP, we can consider the distribution $p(\mathbf{y})$ as having zero mean and covariance defined by the Gram matrix $\mathbf{K}$ such as:

$$p(\mathbf{y}) = \mathcal{N}(\mathbf{y}|0, \mathbf{K}) \tag{E.36}$$

Let us suppose that we are to predict the target label y_{N+1} for testing data $\vec{x}_{N+1}$. If we denote $(y_1, \ldots, y_N, y_{N+1})$ as $\mathbf{y}_{N+1}$, we can write:

$$p(\mathbf{y}_{N+1}) = \mathcal{N}(\mathbf{y}_{N+1}|0, \mathbf{K}_{N+1}) \tag{E.37}$$

where $\mathbf{K}_{N+1}$ is an $(N+1) \times (N+1)$ covariance matrix calculated using equation E.35. It should be noted that matrix $\mathbf{K}_{N+1}$ can be partitioned as:

$$\mathbf{K}_{N+1} = \begin{pmatrix} \mathbf{K}_N & \mathbf{k} \\ \mathbf{k}^T & c \end{pmatrix} \tag{E.38}$$

where $\mathbf{K}_N$ is an $N \times N$ covariance matrix, the vector $\mathbf{k}$ has elements $k(\vec{x}_n, \vec{x}_{N+1})$ for $n = 1, \ldots, N$ and the scalar c is $k(\vec{x}_{N+1}, \vec{x}_{N+1})$. We can write the mean and the covariance of the predictive distribution in equation E.37 as:

$$\begin{aligned} m(\vec{x}_{N+1}) &= \mathbf{k}^T\mathbf{K}_N^{-1}\mathbf{y} \\ cov(\vec{x}_{N+1}) &= c - \mathbf{k}^T\mathbf{K}_N^{-1}\mathbf{k} && \text{(E.39)} \end{aligned}$$

The results in equation E.39 defines the predictive distribution for GP regression with an arbitrary kernel function $k(\vec{x}_n, \vec{x}_m)$. Since $\mathbf{k}$ is a function of $\vec{x}_{N+1}$, the predictive distribution is a Gaussian whose mean and variance both depend on $\vec{x}_{N+1}$.

Appendix F

UCIML Database details

The University of California, Irvine's (UCI) repository of machine learning databases (often known as UCIML) consists of over 180 databases (Frank and Asuncion, 2010). The databases in the repository have been used as a benchmark for performance evaluation of various machine learning methods. In this appendix, the details of the UCIML databases that are used in this dissertation are given. A brief summary of these databases is also given in table F.1.

Database	Data	Features	Classes	Database	Data	Features	Classes
ionosphere	352	34	2	monks2	432	6	2
sonar	208	60	2	monks3	432	6	2
statlog numeric	1000	24	2	monks1	432	6	2
statlog heart	270	13	2	tictactoe	958	9	2
hepatitis	80	19	2	vowel	528	10	11
balance-scale	625	4	3	credit screening	653	15	2
dermatology	358	34	6	Iris	150	4	3
liver-disorder	345	6	2	hayesroth	132	5	3
house vote	232	16	2	parkinson	197	23	2
diabetes	768	8	2	echocardiogram	61	12	2
satimage	6435	36	6	pageblock	5473	10	5
spambase	4601	57	2	segment	2310	19	7

Table F.1: Details (number of data, features and classes) of the UCIML databases used for the comparison of different algorithms during the course of this dissertation.

The following description of the UCIML databases comes from Frank and Asuncion (2010).

ionosphere:

The *ionosphere* database consists of radar data collected by a system installed in Goose Bay, Canada. Received signals were processed using an autocorrelation function whose arguments are the time of a pulse and the pulse number. There were 17 pulse numbers for the installed system. Instances in this database are described by two attributes per pulse number, corresponding to the complex values returned by the function resulting from the complex electromagnetic signal. 'Good' radar returns are those showing evidence of some type of structure in the ionosphere (free electrons in the ionosphere of the Earth). 'Bad' returns are those that do not; their signals pass through the ionosphere. The classification task is to classify good and bad radar returns.

sonar:

The *sonar* database (also known as 'Connectionist Bench') consists patterns obtained by bouncing sonar signals off rocks and a metal cylinder at various angles and under various conditions. The label associated with each pattern is 'R' if the object is a rock and 'M' for mine (metal cylinder). The goal is to classify rock and mine patterns.

statlog numeric:

The *statlog numeric* database, known as Statlog (German credit data) consists of information about customers of a bank. Different attributes are given along with their target labels characterizing customer as either suitable for the issuing of a credit card or otherwise.

statlog heart:

The *statlog heart* database consists of attributes for predicting the heart diseases. Based on attributes such as chest pain type, blood pressure, blood sugar, induced angina, etc. the goal is to classify conditions representing a potential heart disease with those that do not.

hepatitis:

Like *statlog heart*, *hepatitis* database consists of attributes representing some medical conditions. The goal is to classify these attributes representing potential Hepatitis with those that do not.

balance-scale:

The *balance-scale* database was generated to model psychological experimental results. Each example is classified as having the balance scale tip to the right, tip to the left, or be balanced (three classes). The attributes are the left weight, the left distance, the right weight, and the right distance. The correct way to find the class is the greater of left-distance $\times$ left-weight and right-distance $\times$ right-weight. If they are equal, it is balanced.

dermatology:

The *dermatology* database consists of clinical and histopathological attributes to predict erythemato-squamous diseases in dermatology.

liver-disorder:

The *liver-disorder* database consists of attributes related to the blood tests which are thought to be sensitive to liver disorders that may arise from excessive alcohol consumption.

house vote:

The *house vote* database also known as 'Congressional Voting Records' database includes votes for each of the U.S. house of representatives congressmen on the 16 key votes identified by the Congressional Quarterly Almanac (CQA). The CQA lists nine different types of votes: voted for, paired for, announced for, voted against, paired against, announced against, voted present, voted present to avoid conflict of interest, and did not vote or otherwise make a position known. Based on the voting pattern, the goal is to classify congressman as either republican or democrat.

diabetes:

The *diabetes* database consists of attributes representing diabetes related conditions. The goal is to determine if a certain pattern of conditions results in diabetes or not.

satimage:

The *satimage* database (also known as 'Statlog') consists of the multi-spectral values of pixels in 3×3 neighborhoods in a satellite image, and the classification associated with the central pixel in each neighborhood. The aim is to predict this classification, given the multi-spectral values. In the sample database, the class of a pixel is coded as a number.

spambase:

The *spambase* consists of various attributes representing an email. The goal here is to classify such email as either spam or not-spam.

monks1, monks2, monks3:

The three MONK problems were the basis of the first comparison of learning algorithms. The problems have the same domain, that is they have the same attribute. The database *monks1*, *monks2* and *monks3* are obtained by altering the class labels (target).

tictactoe:

The *tictactoe* database (known as 'Tic-Tac-Toe End game') encodes the complete set of possible board configurations at the end of tic-tac-toe games, where 'x' is assumed to have played first. The target concept is 'win for x'.

vowel:

The vowel database also known as Connectionist Bench (Vowel Recognition), consists of data related to vowels spoken by different speakers (each utterance of a speech characterized by ten floating point values). The problem is to classify vowels based on the information.

credit screening:

The *credit screening* database (Credit Approval database) is concerned with credit card applications. All attribute names and values have been changed to meaningless symbols to protect confidentiality of the data.

Iris:

The *Iris* database is one of the best known database found in pattern recognition research. The features include the sepal and petal length of the iris plants. The database contains three classes where each class refers to a type of the iris plant.

hayesroth:

The *hayesroth* database (also known as Hayes-Roth database) consists of features related to the study of human subjects. For example, age, hobby, educational level, etc. The subjects are categorized into three classes based on these attributes.

parkinson:

The *parkinson* database is composed of a range of biomedical voice measurements with Parkinson's disease (PD). Each column in the table is a particular voice measure, and each row corresponds one of 195 voice recording from these individuals. The goal is to discriminate healthy people from those with PD.

echocardiogram:

The *echocardiogram* database consists of data about patients who have suffered heart attacks at some point in the past. Some are still alive and some are not. The survival and still-alive variables, when taken together, indicate whether a patient survived for at least one year following the heart attack. The goal is to predict from the other variables whether or not the patient will survive at least one year.

pageblock:

The *pageblock* database consists of attributes like height, length, eccentricity about the blocks of the page layout of a document that has been detected by a segmentation process. The goal is to classify these blocks into five categories.

segment:

The *segment* database (also known as Image Segmentation) consists of outdoor images data described with high level numeric attributes. The instances from the images are extracted which are 3×3 region and represented in terms of features like intensity, mean, hue, mean hedge, region centroid, etc. The goal is to classify instances into seven categories.

Glossary

***k*-nearest neighbor** methods work by predicting the label of a query point by taking into consideration the labels of the training points in its neighborhood. Typically each training point cast a weight for the prediction proportional to its distance from the query point. xi, 3–5, 7, 14, 15, 18, 19, 23, 28, 29, 32–34, 36, 41, 43, 44, 47–50, 58, 63–65, 68, 70, 72, 74, 83, 84, 89, 92, 97, 98, 101, 107, 111, 113, 120, 135, 136, 170, 171

ARD is a well-known technique for scale estimation (or the relative importance of each feature) in the GP framework. 33, 110

BoostML1 is the proposed local adaptive metric learning algorithm. At each iteration of the algorithm, the best feature is selected and neighborhood is adapted based on the selected feature. 5, 79–82, 84, 169

BoostML2 is the proposed variant of BoostML1. At each iteration of the algorithm, all features are used to adapt neighborhood based on their individual relevances. 5, 79, 82, 84

Complete metric learning is the case of learning a full distance matrix A (diagonal and off-diagonal terms) in the distance measurement framework. xi, 3, 5, 32, 36, 37, 39, 40, 43, 44, 64, 67, 68, 77, 87, 89, 97, 99–101, 103, 110, 113, 119, 121, 136, 170

CRAB is a modification of AdaBoost algorithm such that the weak classifier gives the prediction for the entire range of the input space. xxvi, 4, 124–126, 130, 150

Distance measurement framework provides a mechanism to measure distance and learn a data-dependent distance metric. 3, 19, 32

Feature-set is a collection of different feature-vectors, denoted as $\mathcal{FS}$. 7, 8, 30, 32, 34, 36–39, 119–121, 142, 143, 150, 170

Feature-vector is a set of values concatenated together in a vector, denoted as $\vec{x}$. Each element of the feature-vector is denoted as a ‘feature’. 7–11, 13, 17, 20, 30, 36–42, 56, 57, 72, 92, 103, 112, 119–123, 126, 128–133, 142, 143, 150, 170

GASVM1 is the proposed algorithm that trains an SVM classifier by learning a kernel using the MEGM algorithm. A metric is learned for all classes. 5, 89, 91, 170

GASVM2 is a variant of GASVM1 such that a separate metric is learned for each category. 5, 89, 91

GML is the proposed generic metric learning algorithm for naive and semi-naive metric learning. Any feature relevance measure can be used. 5, 67, 73

GP Gaussian Processes. xi, 3, 5, 14, 15, 26–28, 30–34, 44, 109–112, 120–122, 135, 136, 159, 162–164, 169, 170

GPML1 is the proposed algorithm that train a GP classifier by learning the kernel parameters using the MEGM algorithm. A separate metric (kernel) is learned for all categories. 5, 110, 112, 170

GPML2 is a variant of GPML1 such that a single metric (kernel) is learned for each category. 5, 110, 112

High-level metric learning deals with learning a data-dependent distance metric when data is represented as feature-set. It is concerned with handling feature-vectors in the feature-set effectively for increasing classification performance. xi, 3, 5, 32, 36, 40–42, 119, 121, 122, 128, 129, 136, 170

HML High-level Metric Learning. 119, 129

HML1 is a high-level metric learning scheme that works by concatenating different feature-vectors in the feature-set into a single feature-vector. 120, 121, 130, 131, 133

HML2-CF is a high-level metric learning scheme that treats different feature-vectors in the feature-set separately by training a different classifier for each individual feature-vector. 121–123, 130, 131, 133

HML2-DF is a high-level metric learning scheme that treats different feature-vectors in the feature-set separately by learning a distance metric for each individual feature-vector. 41, 120–123, 130, 133

LASVM is the proposed locally adaptive SVM algorithm which is a modification of LSVM. A LSVM classifier is trained in adapted neighborhoods. 3, 5, 97, 99–107

LDA Linear Discriminant Analysis. 12, 45, 145, 147

LSVM is a technique for training an SVM classifier in the neighborhood of a query point. A separate SVM classifier is trained for each query point. 97–107, 170

MEGM is the proposed global metric learning algorithm based on the minimization of the gradient of MSE in k-nearest neighbor settings. 3, 5, 39, 43, 45, 47–51, 55–60, 64–68, 70, 71, 87, 89–92, 99–104, 110–113, 135, 136, 169, 170

MEGM-SNML is a variant of MEGM complete metric learning for semi-naive metric learning. 5, 67, 70–73

MSE Mean-Square-Error. 3, 17, 23, 43, 45–47, 49, 65, 71, 72, 170

Naive metric learning is the case of learning the diagonal elements of the distance matrix A in the distance measurement framework while assuming that the features are independent from each other. xi, 3, 5, 32, 35–37, 67, 68, 70–74, 77, 87, 89, 100, 109, 110, 119, 121, 136, 170, 171

NCA is an effective global metric learning algorithm based on the maximization of the margin of a classifier in k-nearest neighbor settings. 43–45, 50, 51, 58, 59, 63–66, 100, 113, 135

PCA Principal Component Analysis. 35, 39, 45, 49, 57, 92, 103

Semi-naive metric learning is the same as naive metric learning but it does not assume that features are independent from each other. xi, 3, 5, 32, 35–37, 43, 44, 67–74, 77, 87, 89, 100, 109, 110, 119, 121, 136, 170

SVC Support Vector Classifier. 153, 156–158

SVM Support Vector Machines. xi, 3–5, 7, 14, 15, 23–27, 29–34, 57–59, 63, 64, 79, 87–92, 97–101, 104–107, 109, 111, 113, 120–122, 129–131, 133, 135, 136, 153, 157, 158, 169, 170

Vita

Publications arising from this thesis include:

Zaidi, N. and Squire, D. (2010), Local Adaptive SVM for Object Recognition, pp: 196-201. In *Proceedings of the International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, Sydney, Australia. (Oral presentation: Acceptance rate for oral presentations: 17%)

Zaidi, N. and Squire, D. and Suter, D.(2010), A Gradient-based Metric Learning Algorithm for k-NN Classifiers, pp: 194-203. In *Proceedings of the 23rd Australasian Joint Conference on Artificial Intelligence*, Adelaide, Australia. (Acceptance rate: 47%)

Zaidi, N. and Squire, D. (2010), SVMs and Data Dependent Distance Metric. In *Proceedings of the 25th International Conference of Image and Vision Computing New Zealand*, Queenstown, New Zealand. (Acceptance rate: 36%)

Zaidi, N. and Squire, D. and Suter, D. (2010), BoostML: An Adaptive Metric Learning for Nearest Neighbor Classification, pp: 142-149. In *Proceedings of the 14th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, Hyderabad, India. (Acceptance rate: 23.3%)

Zaidi, N. and Squire, D. and Suter, D. (2010), A Simple Gradient-based Metric Learning Algorithm for Object Recognition. *Technical Report (2010/256), Clayton School of IT, Monash University*, VIC, Australia.

Zaidi, N. and Suter, D. (2008), Confidence Rated Ada-Boost for Generic Object Detection, pp: 1-4. In *Proceedings of the 19th International Conference on Pattern Recognition (ICPR)*, Tampa FL, USA. (Acceptance rate: 45%)

Zaidi, N. and Suter, D. (2008), Object Detection Using a Cascade of Classifiers, pp: 600-605. In *Proceedings of the International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, Canberra ACT, Australia.

Other publications by the author:

Dowe, D. and Zaidi, N. (2010), Database Normalization as a by-product of Minimum Message Length inference, pp: 82-91. In *Proceedings of the 23rd Australasian Joint Conference on Artificial Intelligence* Adelaide, Australia. (Acceptance rate: 47%)

Zaidi, N. (2005), A Statistical Framework for Scale, Rotation and Translation Invariant Shape Recognition. *unpublished, excerpt from undergrad final year project.*

Permanent Address:	Clayton School of Information Technology
	Monash University
	Australia

This thesis was typeset with $\LaTeX\,2_\varepsilon$[1] by the author.

[1] $\LaTeX\,2_\varepsilon$ is an extension of $\LaTeX$. $\LaTeX$ is a collection of macros for $\TeX$. $\TeX$ is a trademark of the American Mathematical Society. The macros used in formatting this thesis were written by Glenn Maughan and modified by Dean Thompson and David Squire of Monash University.

References

Amari, S. and Wu, S. (July 1999). Improving support vector machine classifiers by modifying kernel functions, *Neural Networks* **12**(6): 783–789.

Bachman, G., Narici, L. and Beckenstein, E. (2000). *Fourier and Wavelet Analysis*, Springer-Verlag.

Bachrach, R., Navot, A. and Tishby, N. (2004). Margin based feature selection - theory and algorithms, *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, ACM, New York, NY, USA, pp. 43–50.

Bar-Hillel, A., Hertz, T., Shental, N. and Weinshall, D. (2003). Learning distance functions using equivalence relation, *Proceedings of the Twentieth International Conference on Machine Learning*, pp. 11–18.

Bather, J. (2000). *Decision Theory: An Introduction to Dynamic Programming and Sequential Decisions*, Wiley.

Belongie, S., Malik, J. and Puzicha, J. (2005). Shape matching and object recognition using shape contexts, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **24**(4): 509–522.

Bengio, Y. (2000). Gradient-based optimization of hyper-parameters, *Neural Computation* **12**(8): 1889–1900.

Berg, A., Berg, T. and Malik, J. (2005). Shape matching and recognition using low distortion correspondence, *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, IEEE Computer Society, Washington, DC, USA, pp. 26–33.

Berg, A. and Malik, J. (2001). Geometric blur for template matching, *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 607–614.

Berger, J. (1985). *Statistical Decision Theory and Bayesian Analysis*, Springer.

Bie, T., Tranchevent, L., Liesbeth, M. and Moreau, Y. (2007). Kernel-based data fusion for gene prioritization, *Bioinformatics* **23**(13): 125–132.

Bishop, C. (2006). *Pattern Recognition and Machine Learning*, Springer.

Blei, D., Ng, A. and Jordan, M. (2003). Latent dirichlet allocation, *Journal of Machine Learning Research* **3**: 993–1022.

Bosch, A., Zisserman, A. and Munoz, X. (2007). Representing shape with a spatial pyramid kernel, *Proceedings of the 6th ACM international conference on Image and video retrieval*, CIVR '07, ACM, New York, NY, USA, pp. 401–408.

Bourbaki, N. (1986). *Topological Vector Spaces, Elements of Mathematics*, Springer-Verlag.

Caltech-101 Object Database (2006).
URL: *http://www.vision.caltech.edu/ImageDatasets/Caltech101/*

Chang, C. (2010). Generalized iterative relief for supervised distance metric learning, *Pattern Recognition* **43**(8): 2971–2981.

Chapelle, O., Haffner, P. and Vapnik, V. (1999). Support vector machines for histogram-based image classification, *IEEE Transactions on Neural Networks* **10**(5): 1055–1064.

Chapelle, O. and Keerthi, S. (2008). Multi-class feature selection with support vector machines, *Joint Statistical Meetings (JSM).*

Chapelle, O., Vladimir, V., Bousquet, O. and Mukherjee, S. (2002). Choosing multiple parameters for support vector machines, *Machine Learning* **46**: 131–159.

Cheng, H., Tan, P. and Jin, R. (2010). Efficient algorithm for localized support vector machine, **22**(4): 537.

Coello, C. A., Lamont, G. B. and VanVeldhuizen, D. A. (2007). *Evolutionary Algorithms for Solving Multi-Objective Problems*, Springer.

Comaniciu, D. and Meer, P. (2002). Mean shift: A robust approach towards feature space analysis, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **24**(5): 603–619.

Cover, T. (1968). Rates of convergence for nearest neighbor procedures, *Proceedings of the International Conference on Systems Sciences.*

Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification, *IEEE Transactions on Information Theory* **13**(1): 21–27.

Cover, T. and Thomas, J. (2006). *Elements of Information Theory*, John Wiley and Sons.

Cox, T. and Cox, M. (2001). *Multidimensional Scaling*, Chapman and Hall/CRC.

Cressie, N. A. C. (1993). *Statistics for Spatial Data (Revised Edition)*, Wiley-Interscience, New York.

Cristianini, N. and Shawe-Taylor, J. (2002). On kernel-target alignment, *Advances in Neural Information and Processing Systems*, The MIT Press, pp. 367–373.

Dance, C., Willamowski, J., Fan, L., Bray, C. and Csurka, G. (2004). Visual categorization with bags of keypoints, *Workshop on Statistical Learning in Computer Vision, European Conference on Computer Vision*, pp. 1–22.

Davis, J. and Dhillon, I. (2008). Structured metric learning for high dimensional problems, *Proceeding of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, ACM, New York, NY, USA, pp. 195–203.

Davis, J., Kulis, B., Jain, P., Sra, S. and Dhillon, I. (2007). Information theoretic metric learning, *Proceedings of the 24th International Conference on Machine learning*, ICML '07, ACM, New York, NY, USA, pp. 209–216.

Deb, K. (2002). *Multi-Objective Optimization using Evolutionary Algorithms*, Wiley.

Deb, K., A, P., Agarwal, S. and Meyarivan, T. (2002). A fast and elitist multi-objective genetic algorithm: Nsga-ii, *IEEE Transaction on Evolutionary Computation* **6**: 181–197.

Domenciconi, C., Peng, J. and Gunopulos, D. (2005). Large margin nearest neighbor classifiers, *IEEE transactions on Neural Networks* **16**(4): 899–909.

Domeniconi, C., Peng, J. and Gunopulos, D. (2000). An adaptive metric machine for pattern classification, *Advances in Neural Information and Processing Systems*, MIT Press, pp. 458–464.

Duda, R., Hart, P. and Stork, D. (2006). *Pattern Classification*, John Wiley and Sons.

Evans, A., Thacker, N. and Mayhew, J. (1993). The use of geometric histograms for model based object recognition, *Proceedings of the British Machine Vision Conference*.

Fawcett, T. (2006). An introduction to roc analysis, *Pattern Recognition Letters* **27**: 861–874.

Fei-Fei, L., Fergus, R. and Torralba, A. (2007). Course on recognizing and learning object categories, cvpr2007.
URL: *http://web.mit.edu/torralba/www/*

Fei-Fei, L. and Perona, P. (2005). A bayesian hierarchical model for learning natural scene categories, *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005. CVPR 2005*, pp. 524–531.

Fei-Fei, L. and Perona, P. (2007). Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories, *Computer Vision and Image Understanding, Special issue on Generative Model Based Vision* **106**(1): 59–70.

Feldman, J. and Yakimovsky, Y. (1974). Decision theory and artificial intelligence: A semantics based region analyzer, *Artificial Intelligence* **5**(4): 349–371.

Fischer, R. (1936). The use of multiple measurements in taxonomic problems, *Annals of Eugenics* **7**: 179–188.

Fix, E. and Hodges, J. (1951). Discriminatory analysis - nonparametric discrimination: consistency properties, *Technical report*, Randolph Field Texas, US Air force School of Aviation Medicine.

Frank, A. and Asuncion, A. (2010). UCI machine learning repository.
URL: *http://archive.ics.uci.edu/ml*

Friedman, J. (1994). Flexible metric nearest neighbor classification, *Technical report*, Department of Statistics, Stanford University.

Friedman, J., Hastie, T. and Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting, *Annals of Statistics* **28**(2): 337–407.

Frome, A., Singer, Y. and Malik, J. (2006). Image retrieval and classification using local functions, *Proceedings of Neural Information and Processing Systems*.

Globerson, A. and Roweis, S. (2005). Metric learning by collapsing classes, *Advances in Neural Information and Processing Systems*.

Goldberger, J., Roweis, S., Hinton, G. and Salakhutdinov, R. (2005). Neighborhood component analysis, *Advances in Neural Information and Processing Systems*.

Grauman, K. and Darrell, T. (2005). The pyramid match kernel: Discriminative classification with sets of image features, *Tenth IEEE International Conference on Computer Vision*, Vol. 2 of *ICCV '05*, pp. 1458–1465.

Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection, *Journal of Machine Learning Research* **3**: 1157–1182.

Guyon, I., Gunn, S., Nikravesh, M. and Zadeh, L. (2004). *Feature Extraction, Foundation and Applications*, Springer.

Guyon, I., Weston, J., Barnhill, S. and Vapnik, V. (2002). Gene selection for cancer classification using support vector machines, *Machine Learning* **46**(1-3): 389–422.

Hastie, T. and Tibshirani, R. (1996). Discriminative adaptive nearest neighbor classification, *IEEE transactions on Pattern Analysis and Machine Intelligence* **18**(6): 607–616.

Hastie, T., Tibshirani, R. and Friedman, J. (2001). *The Elements of Statistical Learning*, Springer Series in Statistics.

Hoffmann, T. (2001). Unsupervised learning by probabilistic latent semantic analysis, *Machine Learning* **42**(1-2): 177–196.

Huang, J., Jingjing, L. and Charles, X. (2003). Comparing naive bayes, decision trees, and svm with auc and accuracy, *Third IEEE International Conference on Data Mining*, ICDM '03, pp. 553–.

Hull, J. (1994). A database for handwritten text recognition research, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **16**(5): 550–554.

Iba, W., Wogulis, J. and Langley, P. (1988). Trading off simplicity and coverage in incremental concept learning, *International Conference on Machine Learning*, pp. 73–79.

Janusz, K. (ed.) (2005). *Support Vector Machines: Theory and Applications*, Springer Berlin/ Heidelberg, chapter Adaptive Discriminant and Quasiconformal Kernel Nearest Neighbor Classification.

Jolliffe, I. (2002). *Principal Component Analysis*, Springer Series in Statistics.

Jordan, M. and Jacobs, R. (1992). Hierarchies of adaptive experts, *Advances in Neural Information Processing Systems*.

Jordan, M. and Jacobs, R. (1994). Hierarchical mixture of experts and em algorithm, *Neural Computation* **6**(2): 181–214.

Keerthi, S. (2001). Efficient tuning of svm hyperparameters using radius/margin bound and iterative algorithms, *Technical Report CD-01-02*, Department of Mechanical Engineering, National University of Singapore.

Kira, K. and Rendell, L. (1992). A practical approach to feature selection, *Proceedings of the ninth international workshop on Machine learning*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 249–256.

Kittler, J. (1998). Combining classifiers: A theoretical framework, *Pattern Analysis and Applications* **1**(1): 18–27.

Kohavi, R. and John, G. (1997). Wrappers for feature subset selection, *Artificial Intelligence* **97**: 273–324.

Kohavi, R. and Sommerfield, D. (1995). Feature subset selection using the wrapper model: Overfitting and dynamic search space topology, *Proceedings of the International Conference on Knowledge Discovery and Data Mining*.

Kononenko, I. (1994). Estimating attributes: Analysis and extensions of relief, *European Conference on Machine Learning* **784**: 171–182.

Kononenko, I. (1995). On biases in estimating the multivalued attributes, *International Joint Conference on Artificial Intelligence*.

Kumar, A. and Sminchisescu, C. (2007). Support kernel machines for object recognition, *IEEE 11th International Conference on Computer Vision*.

Lanckriet, G., Cristianini, N., Bartlett, P., Ghaoui, L. and Jordan, M. (2002). Learning the kernel matrix with semidefinite programming, *Journal of Machine Learning Research* **5**: 27–72.

Lazebnik, S., Schmid, C. and Ponce, J. (2006). Beyond bags of features, spatial pyramid matching for recognizing natural scene categories, *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 2169–2178.

LeCun, Y., Denker, J., Solla, S., Howard, R. E. and Jackel, L. D. (1990). Optimal brain damage, *Advances in Neural Information Processing Systems*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 598–605.

Leibe, B., Leonardis, A. and Schiele, B. (2007a). Combined object categorization and segmentation with an implicit shape model, *European Conference on Computer Vision Workshop on Statistical Learning in Computer Vision*, pp. 17–32.

Leibe, B., Leonardis, A. and Schiele, B. (2007b). Robust object detection with interleaved categorization and segmentation, *International Journal of Computer Vision* **77**(1-3): 259–289.

Leibe, B. and Schiele, B. (2003). Interleaved object categorization and segmentation, *Proceedings of the British Machine Vision Conference*, pp. 759–768.

Leyton, M. (1988). A process grammar for shape, *Artificial Intelligence* **34**(2): 213–247.

Lin, Y., Liu, T. and Fuh, C. (2007). Local ensemble kernel learning for object category recognition, *IEEE Conference on Computer Vision and Pattern Recognition.*

Lowe, D. (1995). Similarity metric learning for a variable-kernel classifier, *Neural Computation* **7**(1): 72–85.

Lowe, D. (1999). Object recognition from local-scale invariant features, *Seventh International Conference on Computer Vision*, Vol. 2 of *ICCV '99*, p. 1150.

Lowe, D. G. (1987). Three-dimensional object recognition from single two-dimensional images, *Artificial Intelligence* **31**(3): 355–395.

MacKay, D. (2003). *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press.

MacKay, D. J. C. (1994). Bayesian nonlinear modelling for the prediction competition, *ASHRAE Transactions* **100**: 1053–1062.

Mahalanobis, P. (1930). On tests and measures of group divergences, *Journal of the Asiatic Society of Bengal* pp. 541–588.

Mahalanobis, P. (1936). On the generalised distance in statistics, *Proceedings of the National Institute of Sciences of India*, pp. 49–55.

Mangasarian, O. L., Street, W. N. and Wolberg, W. (1995). Breast cancer diagnosis and prognosis via linear programming, *Operations Research* **43(4)**: 570–577.

Marr, D. (1982). *Vision, A computational investigation into the Human Representation and Processing of Visual Information*, W.H. Freeman and Company.

Mikolajczyk, K., Tuytelaars, T., Schmid, C., Zisserman, A., Matas, J., Schaffalitzky, F., Kadir, T. and Van Gool, L. (2005). A comparison of affine region detectors, *International Journal of Computer Vision* **65**: 43–72.

Minsky, M. and Papert, S. (1969). *Perceptrons*, Cambridge, MA: MIT Press.

Neal, R. (1996). *Bayesian Learning for Neural Networks*, Springer-Verlag New York.

Nene, S., Nayar, S. and Murase, H. (1996). Columbia object image library (coil-100), *Technical report*, CUCS-006-96, February 1996.

Nguyen, N. and Guo, Y. (2008). Metric learning, a support vector approach, *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pp. 125–136.

Nilsback, M.-E. and Zisserman, A. (2006). A visual vocabulary for flower classification, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Vol. 2, pp. 1447–1454.

Nilsback, M.-E. and Zisserman, A. (2007a). Delving into the whorl of flower segmentation, *Proceedings of the British Machine Vision Conference*, Vol. 1, pp. 570–579.

Nilsback, M.-E. and Zisserman, A. (2007b). Flower datasets.
URL: *http://www.robots.ox.ac.uk/%7Evgg/data/flowers/*

Oliva, A. and Torralba, A. (2001). Modeling the shape of the scene a holistic representation of the spatial envelope, *International Journal of Computer Vision* **42**(3): 145–175.

Opelt, A. and Pinz, A. (2006). Fusing shape and appearance information for object category detection, *Proceedings of the British Machine Vision Conference*.

Opelt, A., Pinz, A. and Zisserman, A. (2006). A boundary-fragment-model for object detection, *European Conference on Computer Vision*, Vol. 3952/2006, pp. 575–588.

Pearson, K. (1901). On lines and planes of closest fit to systems of points in space, *Philosophical Magazine* **2**(6): 559–572.

Pekalskai, E., Paclik, P. and Duin, R. (2002). A generalized kernel approach to dissimilarity based classification, *Journal of Machine Learning Research*, Vol. 2, pp. 175–211.

Peng, H., Long, F. and Ding, C. (2005). Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy, **27**(8): 1226–1238.

Peng, J., Heisterkamp, D. and Dai, H. (2001). Lda/svm driven nearest neighbor classification, *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Vol. 1, p. 58.

Press, W., Teukolsky, S., Vetterling, W. and Flannery, B. (1988). *Numerical recipes in C. The art of scientific computing*, Cambridge University Press, Cambridge, UK.

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*, The MIT Press.

Robnik-Sikonja, M. and Kononenko, I. (2003). Theoretical and empirical analysis of relieff and rrelieff, *Machine Learning* **53**(1-2): 2369.

Rodgers, J. and Nicewander, W. (1988). Thirteen ways to look at the correlation coefficient, *The American Statistician* .

Rosch, E. (1973). Natural categories, *Cognitive Psychology* **4**(3): 328–350.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain, *Psychological Review* **64**: 386–408.

Roweis, S. and Saul, L. (2000). Nonlinear dimensionality reduction by locally linear embedding, *SCIENCE* **290**(5500): 2323–2326.

Saul, S. and Roweis, S. (2000). An introduction to locally linear embedding, *Technical report*, AT&T Labs Research.

Schapire, R. (2003). The boosting approach to machine learning: An overview, *D. D. Denison, M. H. Hansen, C. Holmes, B. Mallick, B. Yu, editors, Nonlinear Estimation and Classification. Springer* .

Schapire, R. and Singer, Y. (1998). Improved boosting algorithms using confidence rated predictions, *Machine Learning* **37**(3): 297–336.

Schmidt, A. M. and O'Hagan, A. (2003). Bayesian inference for non-stationary spatial covariance structure via spatial deformations, *Journal of the Royal Statistical Society* **65**(3): 743–758.
URL: *http://www.jstor.org/stable/3647549*

Scholkopf, B. and Smola, A. (2004). *Learning with Kernels, Support Vector Machines, Regularization, Optimization and Beyond*, The MIT Press.

Shafer, S., Kanade, T. and Ikeuchi, K. (1993). Image understanding research at cmu, *Proceedings of the 1993 DARPA Image Understanding Workshop.*

Shakhnarovich, G., Darrell, T. and Indyk, P. (2006). *Nearest Neighbor Methods in Learning and Vision, Theory and Practice*, The MIT Press.

Shamsheyeva, A. and Sowmya, A. (2004). The anisotropic gaussian kernel for svm classification of hrct images of the lung, *Proceedings of the 2004 Intelligent Sensors, Sensor Networks and Information Processing Conference*, pp. 439–444.

Shawe-Taylor, J. and Cristianini, N. (2004). *Kernel Methods for Pattern Analysis*, Cambridge University Press.

Shotton, J., Blake, A. and Cipolla, R. (2005). Contour based learning for object detection, *Tenth IEEE International Conference on Computer Vision*, Vol. 1, pp. 503–510.

Snapp, R. and Venkatesh, S. (1998). Asymptotic expansions of the k-nearest neighbor risk, *The Annals of Statistics* .

Snelson, E., Rasmussen, C. E. and Ghahrammani, Z. (2003). Warped gaussian processes, *Advances in Neural Information and Processing Systems.*

Sriperumbudar, B., Lang, O. and Lanckriet, G. (2008). Metric embedding for kernel classification rules, *Proceedings of the 25th international conference on Machine learning*, ICML '08, pp. 1008–1015.

Steuer, R. (1986). *Multiple Criteria Optimization: Theory, Computations, and Application*, New York, John Wiley & Sons, Inc.

ten Brinke, W. (2010). *On similarity and the semantic gap in content-based image retrieval: A metaphysical approach*, PhD thesis, Clayton School of Information Technology, Monash University, Clayton.

The AT&T Face Database (2002).
URL: *http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html*

The PASCAL Object Recognition Database Collection (2005).
URL: *http://pascallin.ecs.soton.ac.uk*

The Yale Face B Database (2001).
URL: *http://cvc.yale.edu/projects/yalefacesB/yalefacesB.html*

The Yale Face Database (1997).
URL: *http://cvc.yale.edu/projects/yalefaces/yalefaces.html*

Torralba, A., Murphy, K. and Freeman, W. (2007). Sharing visual features for multiclass and multiview object detection, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **29**(5): 854–869.

Turk, M. and Pentland, A. (1991). Eigenfaces for recognition, *Journal of Cognitive Neuroscience* **3**(1): 71–86.

Varma, M. and Ray, D. (2007). Learning the discriminative power-invariance trade-off, *IEEE 11th International Conference on Computer Vision.*

Varma, M. and Zisserman, A. (2005). A statistical approach to texture classification from single images, *International Journal of Computer Vision* **62**(1-2): 61–81.

Viola, P. and Jones, M. (2001). Rapid object detection using a boosted cascade of simple features, *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 511–518.

Weinberger, K., Blitzer, J. and Saul, L. (2005). Distance metric learning for large margin nearest neighbor classification, *Proceedings of Neural Information and Processing Systems.*

Weinberger, K., Blitzer, J. and Saul, L. (2009). Distance metric learning for large margin nearest neighbor classification, *Journal of Machine Learning Research* **10**: 207–244.

Willamowski, J., Arregui, D., Csurka, G., Dance, C. and Fan, L. (2004). Categorizing nine visual classes using local appearance descriptors, *Workshop on Learning for Adaptable Visual Systems, International Conference on Pattern Recognition.*

Williams, C. K. I. and Rasmussen, C. E. (1996). Gaussian processes for regression, *Advances in Neural Information and Processing Systems*, pp. 514–520.

Wu, Y., Chang, E., Chang, K. and Smith, J. (2004). Optimal multimodal fusion for multimedia data analysis, *Proceedings of the 12th annual ACM international conference on Multimedia*, MULTIMEDIA '04, ACM, New York, NY, USA, pp. 572–579.

Xiao, R., Li, W., Tian, Y. and Tang, X. (2006). Joint boosting feature selection for robust face recognition, *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Vol. 2, pp. 1415–1422.

Xing, E., Ng, A., Jordan, M. and Russell, S. (2002). Distance metric learning with application to clustering with side-information, *Advances in Neural Information and Processing Systems*, pp. 505–512.

Xu, J. and Liu, L. (2009). Gradient based optimization of kernel polarization for rbf kernels, *International Conference on Computational Intelligence and Neural Computing*, CINC '09, pp. 85–87.

Yakimovsky, Y. (1964). *Scene Analysis Using a Semantic Base for Region Growing*, PhD thesis, Stanford University, Stanford, CA, USA.

Yakimovsky, Y. (1975). Boundary and object detection in real world images, *International Joint Conference on Artificial Intelligence*, pp. 695–704.

Yakimovsky, Y. and Feldman, J. (1974). On the recognition of complex structures: Computer software using ai applied to pattern recognition, *Proceedings of the International Conference on Pattern Recognition*, pp. 345–353.

Yijun, S. (2007). Iterative relief for feature weighting: Algorithms, theories and applications, *IEEE Transactions on Pattern Analysis and Machine Learning* **29**(6): 1035–1051.

Yijun, S. and Dapeng, W. (2008). A relief based feature extraction algorithm, *SIAM International Conference on Data Mining*, pp. 188–195.

Yijun, S. and Jian, L. (2006). Iterative relief for feature weighting, *Proceedings of the 23rd international conference on Machine learning*, ICML '06, ACM, New York, NY, USA, pp. 913–920.

Zaidi, N. and Squire, D. M. (2010a). Local adaptive svm for object recognition, *2010 International Conference on Digital Image Computing: Techniques and Applications*, pp. 196–201.

Zaidi, N. and Squire, D. M. (2010b). Svms and data dependent distance metric, *Proceedings of the 25th International Conference of Image and Vision Computing New Zealand.*

Zaidi, N., Squire, D. M. and Suter, D. (2010a). A gradient-based metric learning algorithm for k-nn classifiers, *Proceedings of the Australasian Joint Conference on Artificial Intelligence*, pp. 194–203.

Zaidi, N., Squire, D. M. and Suter, D. (2010b). BoostML: An adaptive metric learning for nearest neighbor classification, *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 142–149.

Zaidi, N., Squire, D. M. and Suter, D. (2010c). A simple gradient-based metric learning algorithm for object recognition, *Technical Report 2010/256*, Clayton School of IT, Monash University, VIC.

Zaidi, N. and Suter, D. (2008a). Confidence rated ada-boost for generic object detection, *Proceedings of the International Conference on Pattern Recognition*, pp. 1–4.

Zaidi, N. and Suter, D. (2008b). Object detection using a cascade of classifiers, *2008 International Conference on Digital Image Computing: Techniques and Applications*, pp. 600–605.

Zhan, D., Li, M., Li, Y. and Zhou, Z. (2009). Learning instance specific distance using metric propagation, *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, ACM, New York, NY, USA, pp. 1225–1232.

Zhang, H., Berg, A., Maire, M. and Malik, J. (2006). Svm-knn: Discriminative nearest neighbor classification for visual category recognition, *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 2126–2136.

Zhang, J., Marszalek, M., Lazebnik, S. and Schmid, C. (2006). Local features and kernels for classification of texture and object categories: A comprehensive study, *Conference on Computer Vision and Pattern Recognition Workshop*, CVPRW '06, p. 13.

Zhou, H. and Suter, D. (2008). Improved building detection by Gaussian processes classification via feature space rescale and spectral kernel selection, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–6.

Zhou, Z. and Dai, H. (2006). Query-sensitive similarity measure for content-based image retrieval, *Sixth International Conference on Data Mining*, pp. 1211–1215.