Object Recognition with Partially Labeled Examples

by

Andrew S. Crane

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Computer Science and Engineering

and Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

September 1, 2002

Author_____
Department of Electrical Engineering and Computer Science
September 1, 2002

Certified by_____
Tomaso Poggio
Thesis Supervisor

Accepted by_____
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Object Recognition with Partially Labeled Examples

by
Andrew S. Crane

Submitted to the
Department of Electrical Engineering and Computer Science

September 1, 2002

In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

Abstract

Machine learning algorithms tend to improve in performance with larger training sets, but obtaining a large amount of training data comes at a high cost. Several methods of semi-supervised learning have been introduced recently to take advantage of a larger training set without the burden of labeling many samples. We apply these semi-supervised learning methods to a data set of cars and background images, attempting to separate the two classes. Some of the algorithms obtain very high classification accuracy and can be used towards a car-detection system.

Thesis Supervisor: Tomaso Poggio
Title: Professor, Brain Sciences Department and Artificial Intelligence Lab

Acknowledgements

First, I must thank Martin Szummer for all of his help in the work on my thesis. Even with his own deadlines approaching, he gave me countless hours of support and advice on the work, and supplied me with much of the code I used for the tests. I also appreciate the help from the rest of the members of Tommy Poggio's lab. The weekly meetings and social gatherings with them gave much needed breaks during the hard hours of work.

I've been taught by some great people throughout school, but two deserve special recognition. First is Michael Horner, because without his motivation, I never would have pursued my interests and landed where I did. Second is Patrick Winston, who initially sparked my interest in AI, then taught me about V-S-N and everything else important about writing a thesis. He also gave much needed advice throughout the process.

I owe my sanity over the last 8 months to my friends and fraternity brothers. They helped me enjoy the time in between running tests with Simpsons and Family Guy viewings and trips to Anna's. Especially McFly and KJ, who always were there to listen to me vent or to help me out in whatever way they could.

Last but certainly not least, I owe all my thanks to Mom, Dad, Jeremy, and the rest of my family. Without their constant motivation and support, I could not have made it through MIT, and certainly would not have been here in the first place without the ideals they taught me.

Table of Contents

List of Figures

Chapter 1

Introduction

We run into machine learning tasks every day through all aspects of life. Machine learning techniques are present when we use a credit card, play games, use voice activated commands, and walk around in public.

Recognition tasks are designed to identify labels for previously unseen samples. Some sample recognition tasks are:

- Who is the individual in this photograph?
- What is this news article about?
- What word did that person just say?

Slightly less complicated tasks involve determining whether a sample belongs in a given class. Some examples of this are:

- Is this a picture of Andy?
- Is this article about sports?
- Did he say *yes*?

There are two stages of machine learning. The first is training, where samples of known identity are fed to the classifier. This requires a human to select the training samples, look through them, and determine the appropriate label. The second stage is testing, where the new samples are labeled according to their similarities to the training points.

A classifier should be able to achieve better performance if it has more training samples. With lots of data, it can more accurately determine what features are most important in determining the identity.

In supervised learning tasks, all of the training data is labeled ahead of time, and the classifier uses this information to determine what distinguishes one class from another. Unsupervised learning is slightly different, as it does not use labels for the

training data. Instead it determines which samples are likely to have the same label based on their similarities to each other.

In this thesis we explore a technique, semi-supervised learning, which has labels for some training points but not others. A classifier uses all of the data to learn the distribution of points. With some of these labeled, it is able to infer the labels of the rest by assuming that nearby points are likely to be in the same class.

Semi-supervised, or partially labeled, methods can improve the classification by giving a better idea of the distribution of data samples, as Figure 1-1 illustrates. A classifier trained with only a few samples can do reasonably well, but is highly dependent on which samples are chosen for training. With a large set of unlabeled data, the classifier can better learn how to separate the classes. For the simple example shown, the separating boundary is quite different depending on whether the unlabeled samples are used. Using the wrong boundary can cause several of the points to be mislabeled.

In this thesis we discuss several of these semi-supervised methods and their applications towards real classification tasks.

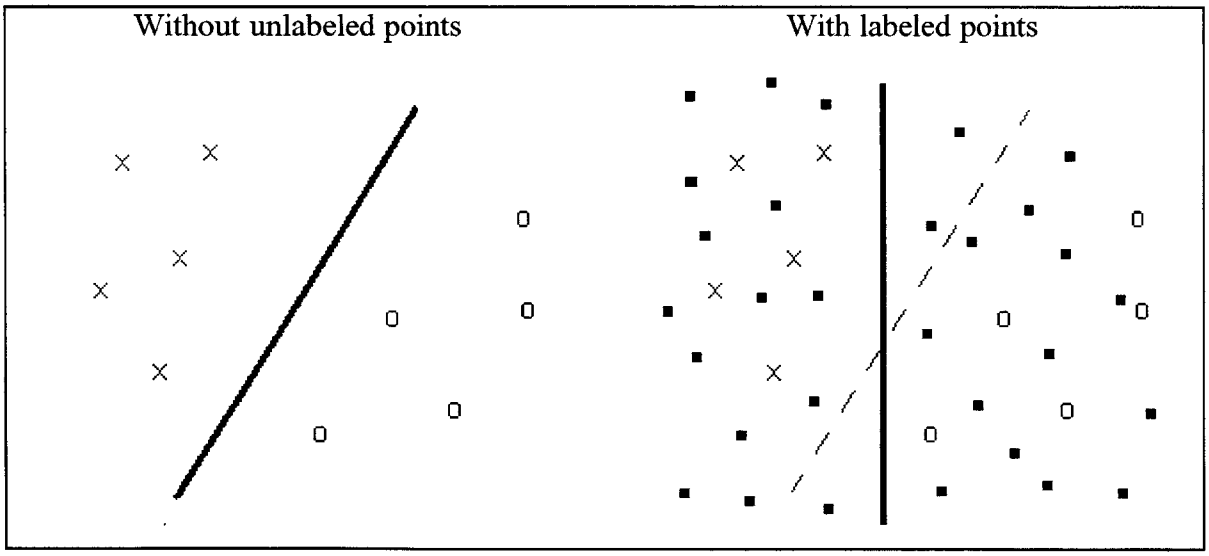


Figure 1-1: The left figure is an example classifier constructed based on the few labeled points present. The right figure uses the same labeled points plus additional unlabeled points, and finds a different separating boundary.

1.1 Motivation

In recent years, computational power has increased steadily to the point where many problems have become feasible as machine learning tasks. Running on a high-

speed computer, the best face recognition tasks took about a third of a second for a single image several years ago [Turk and Pentland, 1991]. Newer applications have been designed that can process thousands of images per second running on a handheld computer [Weinstein *et al.*, 2002]. With machine learning becoming more applicable to real-world problems, there has been a big push for improvements in the algorithms to attack the problems. Methods such as Support Vector Machines [Vapnik, 1998] have been introduced over the last several years, and there is continuing work for improvement.

The increased capability is due to the great increase in capability of computers recently. Processor speeds in computers double approximately every eighteen months according to Moore's Law, and the cost of disk space is dropping steadily. Over several years, this allows the same program to be run in a fraction of the time. Furthermore, larger training databases are feasible because of higher disk capacity. With faster processing and larger training sets, machine learning tasks can run in less time and with a higher degree of accuracy.

One aspect of machine learning that will never benefit from faster processors or more disk space is the human interaction required. For any method to work well, it needs to be given a sufficiently sized training set. Each training sample needs to be manually processed by a human, and the amount of time depends on the task. For document classification, it takes time for someone to read through the text and determine the appropriate class. Some object recognition problems require someone to label various components of the object or carefully sketch the outline. The labeling process can't be automated, because if a system existed that could label the training data well enough, the machine learning task would already be solved.

With the vast amount of information on the World Wide Web, finding unlabeled samples may be very simple. Using a large amount of unlabeled data to assist in the machine learning process without requiring more labeled data would be extremely helpful. These semi-supervised methods will help remove the burden of a human to do the labeling while still allowing for an accurate classifier.

1.2 Previous work

1.2.1 Object detection

[Papageorgiou, 2000] has developed a system to detect cars and bodies primarily from a collection of wavelet features. Using Haar wavelets [Mallat, 1989], he converts each image into a feature vector. These feature vectors are then plugged into a classifier to determine the identity of the object. He further extends the system to work on video sequences with the ability to track objects as they move over time in the video. The work in this thesis uses features similar to those constructed in [Papageorgiou, 2000].

[Viola and Jones, 2001] have implemented a system capable of face recognition in real-time. They are able to attain such fast performance with several ideas that avoid extraneous computations, and some of the ideas are applicable to other image recognition domains. The main contribution is a method of selecting an appropriate set of features for a classifier and quickly discarding sections of an image that don't match some of the features. By avoiding redundant processing, the classifier works very quickly with a high degree of accuracy.

Several systems have been based on models of the objects being recognized. [Schneiderman and Kanade, 2000] use a statistical approach towards face and car recognition. They construct 3D models for each of the objects in different poses. For new images, they use a product of histograms to determine the likelihood that an image was generated from one of the object models. [Selinger and Nelson, 2000] construct 3D models of the objects from poses evenly spread across a viewing sphere.

Rather than detecting full objects, it is often useful to detect components. [Mohan et al., 2001] detect bodies by components. They first use individually trained classifiers to detect portions of the object, such as the legs, arms, and head in bodies. The location and certainty score for all of the components are fed into another classifier which determines whether the appropriate object is present. This approach allows a person to be detected even when parts are occluded or absent in the image (such as a one-armed man) or the person blends into the background. A system by [Heisele et al., 2001] takes a similar approach towards detecting faces, using components such as the eyes, nose, and mouth.

1.2.2 Partially labeled data

Classifiers have previously been designed to be trained with entirely labeled data. Some newer approaches have considered including some unlabeled data in the training data.

[Joachims 1999B] has constructed Transductive Support Vector Machines, a way of extending SVMs to take unlabeled data into account during training. He has applied this method towards text classification, and it can be used for other domains as well.

Kernel expansion, by [Szummer and Jaakkola, 2000], applies a density-dependent representation that focuses on high-density regions of data. The decision boundary is made more flexible in these regions, and the influence of labeled points located in high-density regions is increased.

Another method by [Szummer and Jaakkola, 2001] relies on the Markov random walk representation. It measures similarity across samples by following the data manifold, which may be a low-dimensional subspace of the embedding space. They have used this method with good performance on problems that are difficult to separate using other models such as SVMs.

Several variations of spectral clustering methods exist for unsupervised learning [Ng et al., 1999]. These methods form a new set of features based on the similarity of samples. These methods can be modified to work as semi-supervised algorithms [Chapelle et al., 2002].

1.3 Contributions

Many researchers have applied supervised learning techniques towards real classification tasks, but with semi-supervised techniques, much of the existing work has been theoretical. We apply several of these methods towards an actual vision task and find which algorithms are most suitable. We find that some algorithms do not work as well because our data set does not have characteristics that are assumed to be true in theoretical work.

Our findings provide the initial steps towards creating a useful car-detecting system. We determine which learning algorithms are most useful and recommend some modifications to our tests that will lead to a system that can be implemented.

1.4 Outline

In Chapter 2, we describe the data set being used in our tests and introduce the classification algorithms. Results from the tests and analysis of the algorithms are given in Chapter 3. Some additional tests, using slightly different data and methods, are described in Chapter 4. Chapter 5 summarizes the work and gives some ideas on how these methods can be used in a real-life system.

Chapter 2

Data and Methods

2.1 Data format

The primary objective of this thesis is to determine how well various learning methods work with partially-labeled samples on a real set of data. The domain that is used in this project is the task of recognizing frontal images of cars, although numerous other domains could be used instead.

For the task of detecting images of cars, it will be important to have a well-chosen set of features that can detect similarities and differences among images. Some prior knowledge should be used in selecting the features, but not so much that the problem can't be generalized. Humans can recognize cars by the shape and components such as the windows, license plate, headlights, and wheels. Further knowledge can be extracted from the background. Pictures of cars tend to have a road below and objects such as buildings, trees, or the sky above and to the sides.

This tells us that a window slightly larger than the car should be viewed in order to take advantage of the background scenery. Furthermore, based on the shape of cars, a square viewing window should be sufficient rather than a wider or taller rectangle, although this would change if we were attempting to detect side-views of cars. We use a 128x128 pixel image window, and attempt to detect cars that are 64 pixels across the front bumper, centered horizontally and vertically. This is a reasonable size to extract enough detail to discriminate all of the relevant features of the car, while not too large for computational considerations. Sample images are given in Figure 2-1.

Images of non-cars are negative samples and can be anything besides cars in the correct configuration. They are generated randomly from the images containing cars, so they tend to contain roads, buildings, trees, and portions of cars. Some also contain cars that are not correctly centered or scaled. This can give a classifier the ability to distinguish cars from other objects in the images. An alternate approach would be to

construct negative samples from images of completely different objects, such as pedestrians or faces. This may have a disadvantage, because when the classifier is in its testing phase, it would be distinguishing cars from background components which it may never have seen in training.

The images are all in grey-scale, where each pixel's intensity is measured as an integer between 0 and 255. A different method, used in a similar system by [Papageorgiou, 2000], uses three color channels and records each pixel as the highest of the three values. We use grey-scale to accommodate a larger variety of images as inputs. Humans are perfectly capable of determining whether an image is a car or not from a grey-scale image, so this task is certainly feasible.

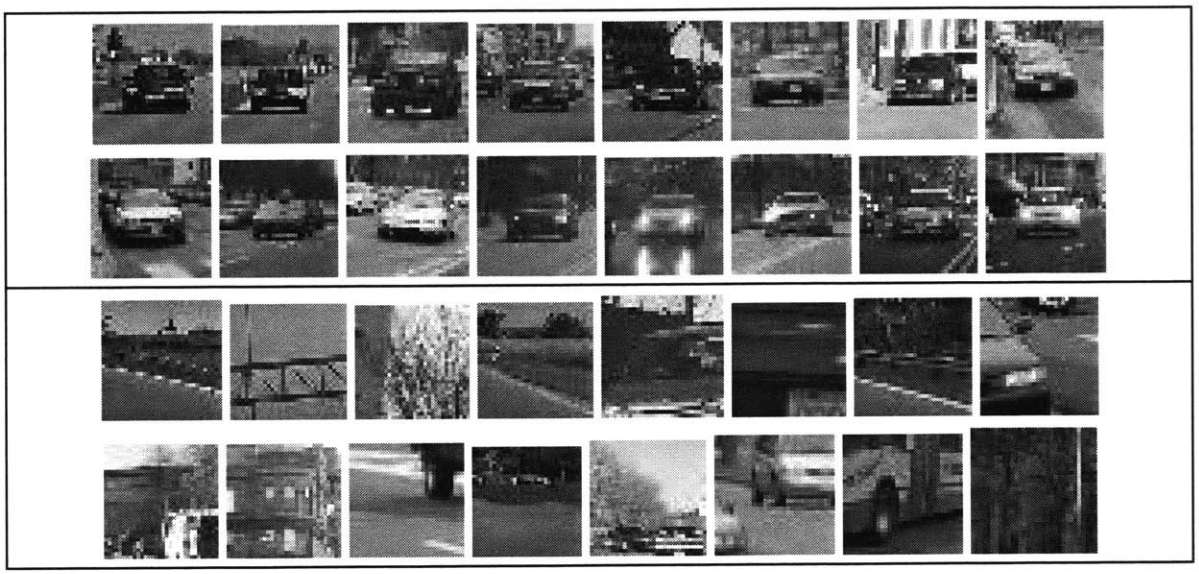


Figure 2-1: Sample images from both the car class (top rows) and non-car class (bottom rows). The non-car images frequently contain parts of cars, but are not placed in the car class because they are not appropriately centered and scaled.

2.2 Features

It is important that the system works on a wide variety of cars, regardless of model or color. The size and shape varies from one model to another, although all cars have roughly the same ratios across dimensions. The color is far more variable, however. Even in grey-scale, there is a large difference in the pixel intensity for dark and light cars. This variation may make it difficult to train a system based on pixel intensities.

It is important to note that the color is not what helps us recognize cars, but rather the change in colors at the edges. An image of the hood of the car shows very little

change in color. The point where the side of the car meets the road, however, gives far more information. This leads to the idea of using an edge detector to extract new features from the image that can be useful in classification. Figure 2-2 shows a car image with distorted colors. By observing the shape, it is still possible for a human to determine that this is a car.

Haar wavelets are useful in constructing an edge detecting representation [Mallat, 1989]. To convert an image into features, a sub-window is moved horizontally and vertically across the 128x128 pixel image. For each sub-window, three values are extracted, corresponding to different angles of lines to detect:

- Horizontal edges: The average pixel intensity of the top half minus the average pixel intensity of the bottom half.
- Vertical edges: The average pixel intensity of the left half minus the average pixel intensity of the right half.
- Diagonal lines: The average pixel intensity of the upper-left corner and bottom-right corner minus the average pixel intensity of the upper-right corner and bottom-left corner.

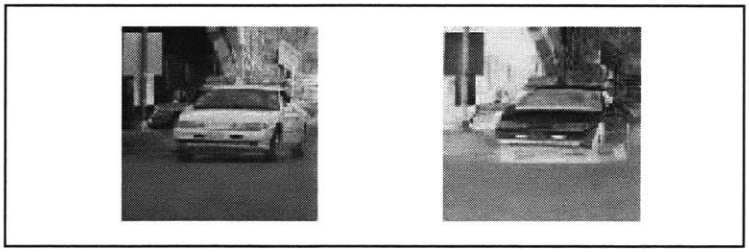


Figure 2-2: A car image from the database and the same image with its colors distorted. In the distorted image, it is still possible to tell that it is a car.

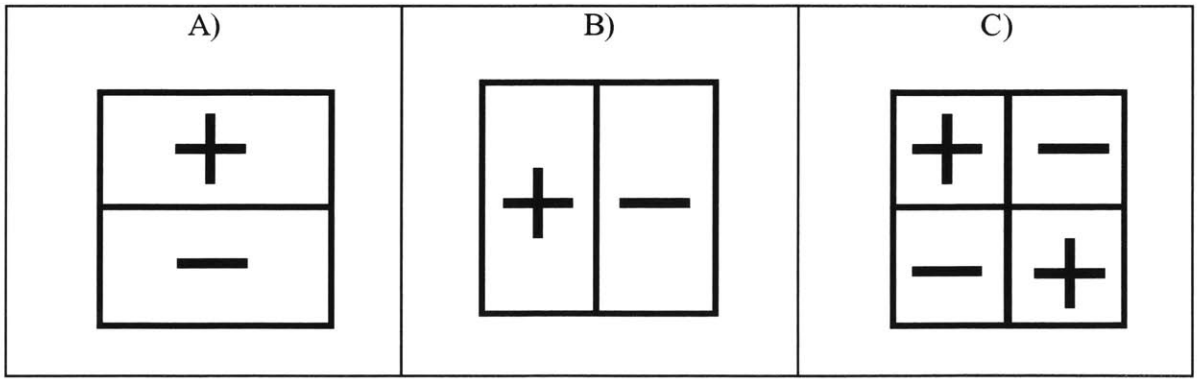


Figure 2-3: Haar wavelet templates for (A) Horizontal edges, (B) Vertical edges, and (C) Diagonal lines.

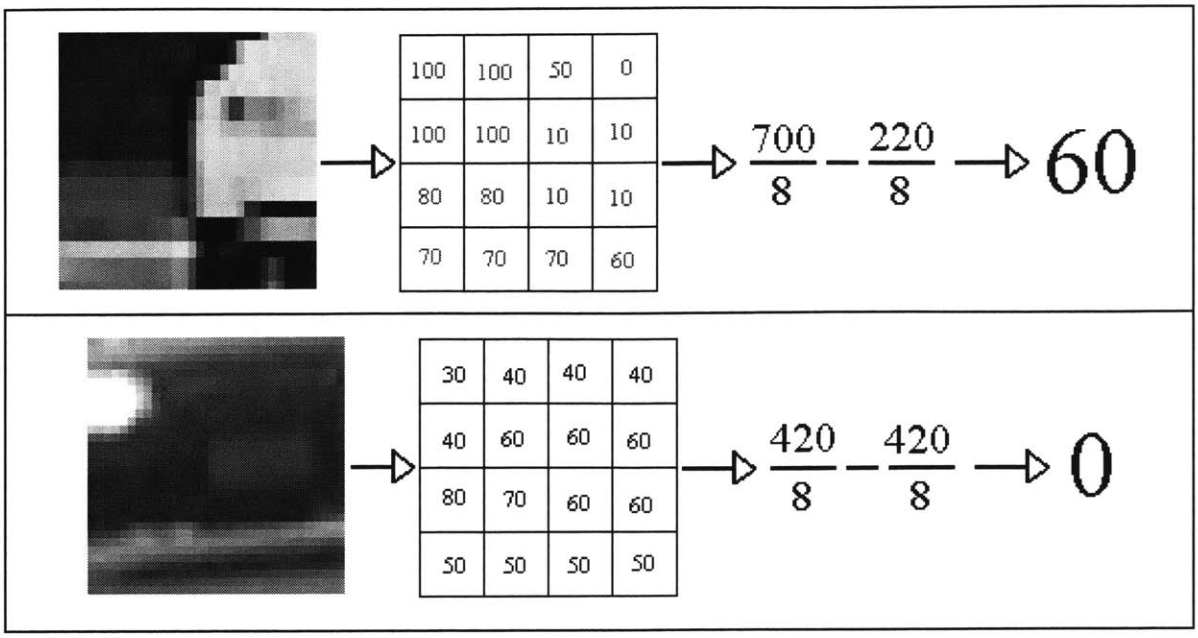


Figure 2-4: Vertical edge detection on parts of two images. The numbers listed are not the actual values, but darker pixels correspond to higher numbers.

Figure 2-4 shows the process for vertical edge detection on two different images. The image is converted into a numeric representation of the pixel intensities. From here, the average intensity of the left half and the right half are computed. Finally, we compute the wavelet value for the sub-window as the difference in average intensities of the two halves. For the window with a vertical edge, the value is much higher than the window without an edge. All 3030 features are computed in this way using the different combinations of wavelet size, direction, and sub-windows within the image.

The feature vector itself can be viewed as an image, showing the edges at each resolution for all three orientations. Lower values, displayed as darker shades, correspond to regions with little change. Higher values, displayed in lighter shades, are where edges or lines are present. Figure 2-5 shows a sample car image along with its representation. Edges of the car can clearly be seen, as well as some of the lines on the road and in the background.

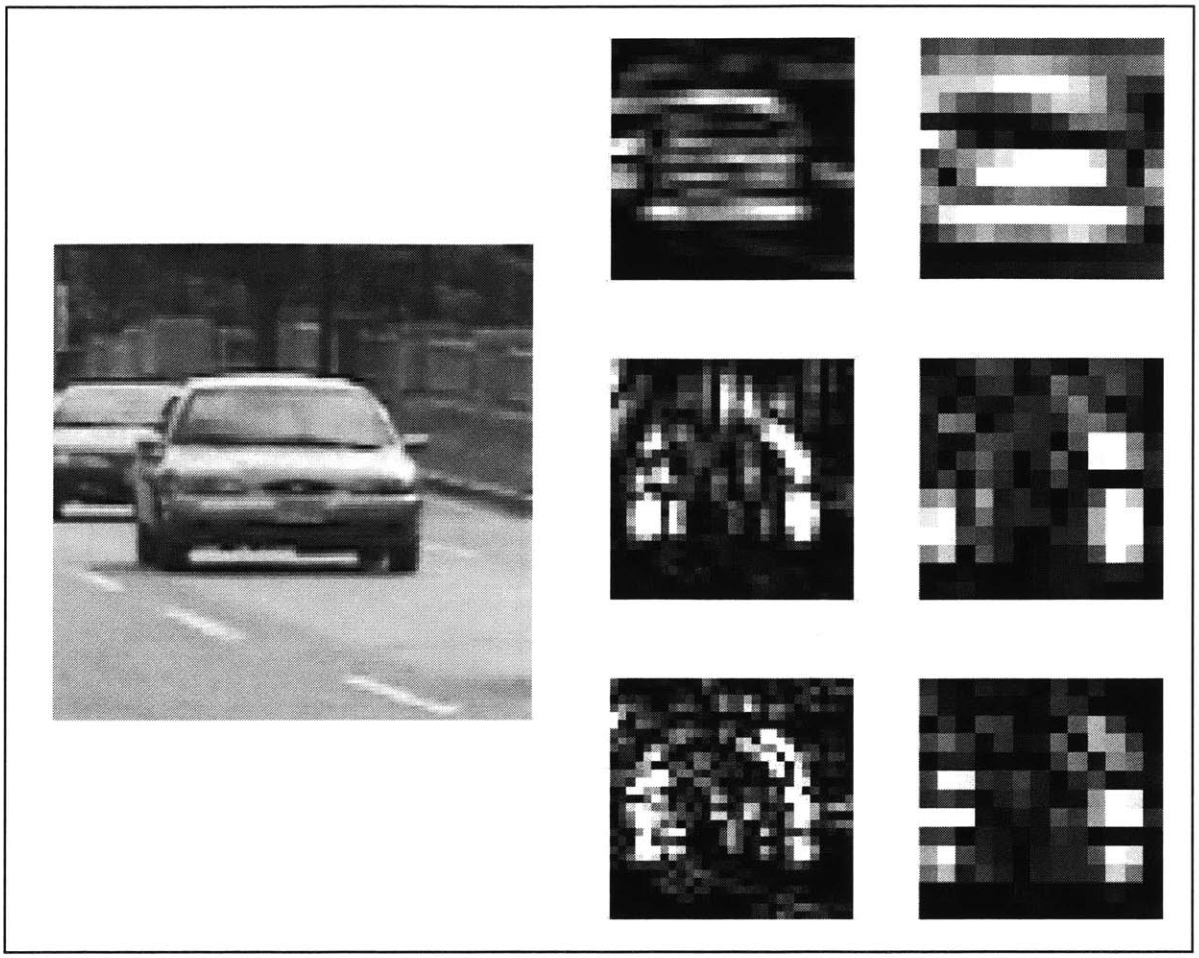


Figure 2-5: A sample car image with its features. The upper row of features detects horizontal edges at 16x16 and 32x32 resolution, the middle row detects vertical edges, and the bottom row detects diagonal edges.

2.3 Normalization

The values for each feature can range from -255 to 255. The extreme cases are when one half of the image window is entirely black and the other is entirely white. Symmetric cases essentially give us the same information content, so we use absolute values of the features. This is not always the case, as with some other vision tasks, dark-to-light changes must be distinguished from light-to-dark. For the algorithms that are used, it is better to work with smaller values (i.e., where all features have a value below 1) for the features. There are several ways of normalizing the data to accomplish this.

The simplest method of normalization is to divide all features by 256, which will force all values to be between 0 and 1. As long as parameters are scaled accordingly, this

does not affect the results with any of the algorithms. It is used for reasons of numerical stability, as it is much easier to work with smaller numbers.

Other forms of normalization vary each feature by a different amount. These will affect the results of the algorithms because the distance measures between any two samples will have changed entirely. One such normalization divides each feature by the average value for that type of feature within that image. For example, if the average value for all horizontal 16 x 16 wavelets in an image is $\overline{h}_{16}$, then each value h_{16i} is mapped to $h_{16i}/\overline{h}_{16}$. This normalization can take lighting conditions into account, because the average values will be lower for images with less contrast. Dividing all of the values by a smaller average will cause a further spread in the data.

Another type of normalization takes knowledge of the domain into account. In our example, features towards the center of the image, where the car is located, are far more important than features toward the edges. If two images are nearly identical except for a small difference, that difference is more important if it is in the car rather than the background. [Bateson, 1972] calls information "differences that make a difference." Here the differences that help us distinguish one class of images from another are the most important. Because the center of the image holds more information in this regard, a normalization that weighs this part more heavily could improve performance for some of the algorithms. We avoid using such a normalization in an attempt to keep this as a generic machine learning problem. In other domains, such information may be unavailable.

2.4 Distance metric

A necessary component of any learning algorithm is a way to evaluate how similar two samples are. The distance metric should be minimized for two samples that are identical, and increase as more differences appear.

For the car images, the representation allows for a simple way of computing the distance between two samples, the Euclidean distance between their feature vectors. This is measured as

$$D(x_1, x_2) = \sum_i (x_1(i) - x_2(i))^2 \qquad \textbf{Equation 2-1}$$

where $x_k(j)$ refers to the value of feature j of image k.

If $x_1(i) \sim x_2(i)$ for most of the features, then the distance between the two points will be small. This corresponds to when the two images have similar Haar wavelets. It is possible, though unlikely, that two different images will have identical wavelets, and hence a distance of 0.

Some useful properties of the distance metric can be observed from Equation 2-1. First, any point has a distance of 0 from itself (Let $x_1 = x_2$, then $D(x_1,x_2) = 0$). Second, the distance metric is symmetric (i.e., $D(x_1,x_2) = D(x_2,x_1)$).

A desirable property is for the intra-class distances to be small compared to the cross-class distances. If the features are well chosen, this condition should be met for many pairs of samples, but not all. For some samples that are borderline cases (such as an image that contains a car slightly off-center) it may be the case that a number of samples from the opposite class are closer than some in the same class.

Figure 2-6 gives histograms for three samples. The histogram in (A) is for an ordinary car, and as expected, the majority of other cars are closer than the non-cars. A different car that is often misclassified is shown in histogram (B). The other cars are fairly close, but there are a number of images from the other class that are closer, which explains why the sample is so difficult to identify. The histogram for a non-car image is shown in (C), which shows that non-cars tend to be closer.

Figure 2-7 shows some sample images from the database and their distances to each other. This demonstrates that it is not always the case that intra-class distances are smaller than cross-class ones. Two of the cars are closer to a non-car than they are to the third car.

We ran some further tests on the data to see how reliable the distance metric is overall. For a randomly chosen car sample within this data set, if one other car and non-car are randomly chosen, the car is closer 68% of the time. For a random non-car, it is closer to the other non-car 77% of the time. This means that more likely than not, any random sample will be closer to a single sample from the same class than from the opposite class, although there is still a high degree of error. By looking at 10 images of each class, the closest point is within the same class 87% of the time for cars and 92% for non-cars.

The distance metric is used to find how likely it is that two samples came from the same distribution. Each algorithm uses it in a slightly different fashion, which will be described in the discussion of the individual algorithms.

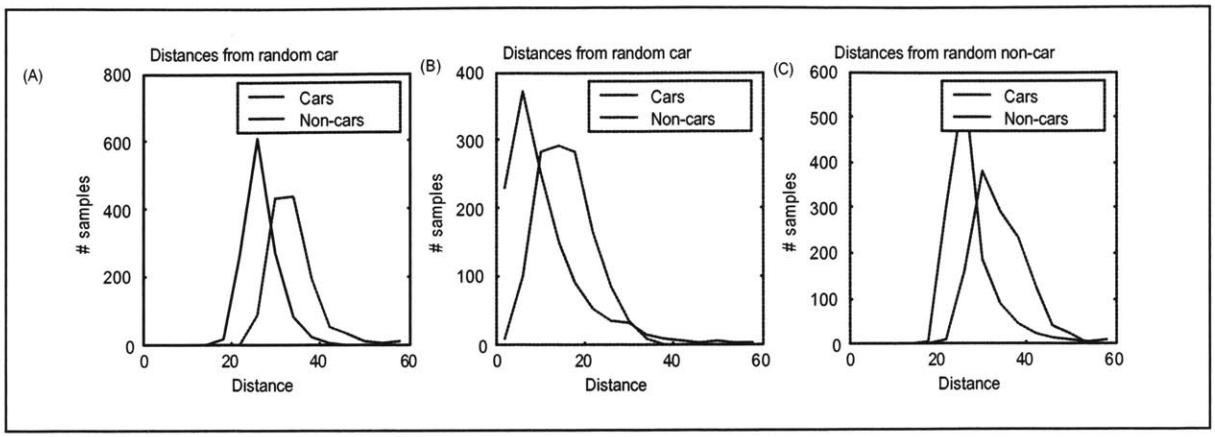
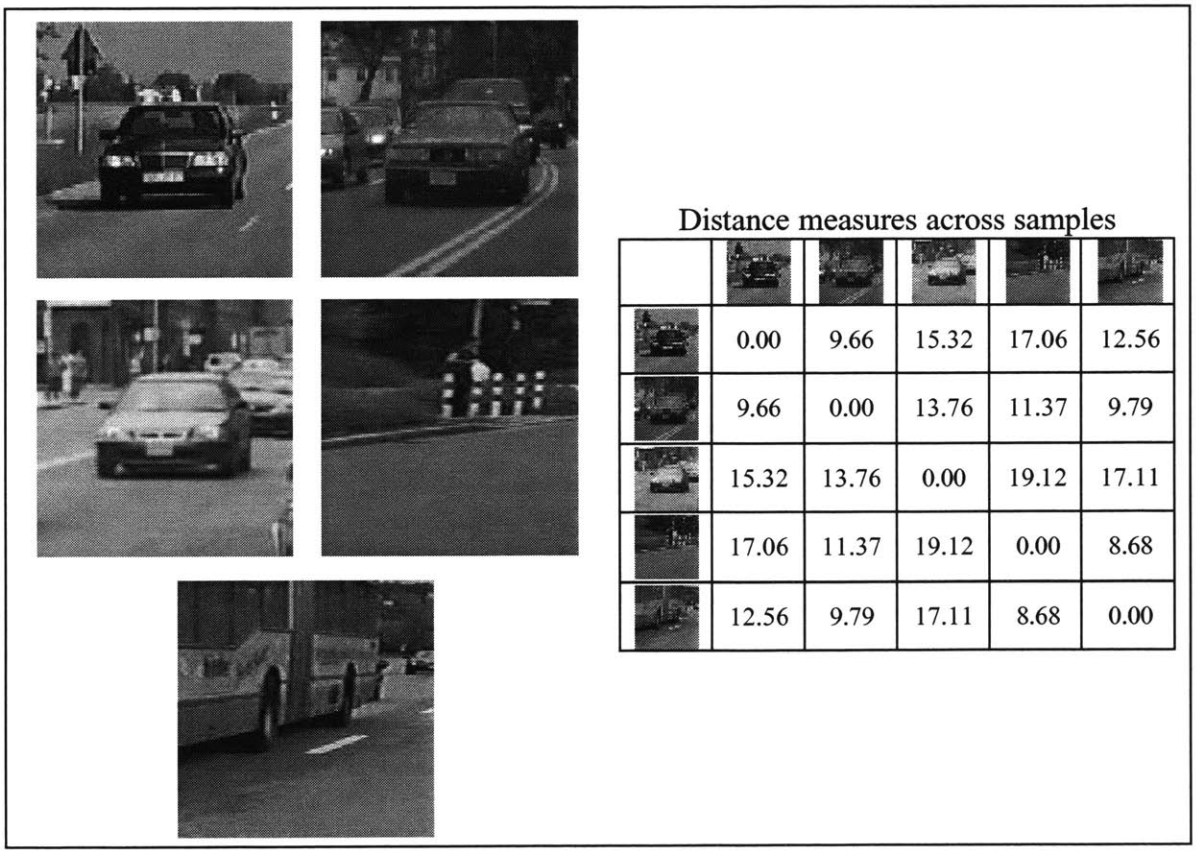


Figure 2-6: Histograms of inter-class and intra-class distances for images of (A) a normal car, (B) a difficult to label car, and (C) a non-car.



Distance measures across samples

	0.00	9.66	15.32	17.06	12.56
	9.66	0.00	13.76	11.37	9.79
	15.32	13.76	0.00	19.12	17.11
	17.06	11.37	19.12	0.00	8.68
	12.56	9.79	17.11	8.68	0.00

Figure 2-7: Some images from the database and their distances. The inter-class distances are generally larger than intra-class distances, but not in all cases.

2.5 Similarities and differences across samples

As demonstrated, two images from the same class aren't always close together according to the distance metric. Even two different images of a stationary object taken at different times can have a large separation between them. Some factors that influence this are:

- Lighting conditions: Depending on the illumination to various parts of the image, the features can come out very differently. With less sunlight, there will be less contrast between the objects in the image and the sky. Also, the shadows will be affected by the lighting. Shadows cast from the object onto the ground will be picked up by the wavelet features, so if shadows move or are absent entirely, the features will change.
- Season and weather: Weather factors such as rain, snow, and clouds that are visible in the image will directly affect the features.
- Pose and distance from object: Taking a picture from further away and zooming in can cause some loss of quality and cause parts of the zoomed-in images to appear pixelated.
- Backgrounds: Even if the object is stationary, objects in the background often will have changed position.
- Specularity: Shiny objects will cause reflections, which may vary in a similar way as the backgrounds.
- Camera model: Different cameras will have slightly different functionality, leading to different images. For example, the flash may have different intensity on different cameras, and the results pictures will appear different.
- Occlusion: Portions of an object are often blocked by other items in the image.

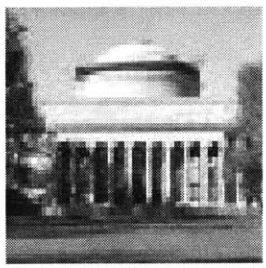
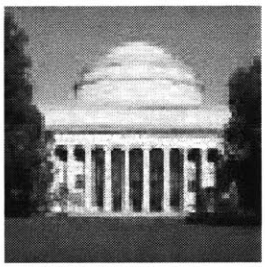
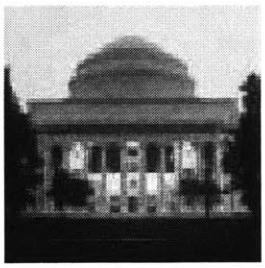
Within the car class, there are several sources of variation.

- Car model and year: Even though the basic shape of most cars are the same, different models will have significant variation amongst them. The size ratios of different models are different, some cars are curvier than others, and headlights and other components are in different places.

- Color: The images are in grayscale, but dark cars will be significantly different from light ones.
- Road: Images from city roads, country roads, and highways will have different types of scenery.

Non-car images will also vary significantly. Because these images are generated from random segments of car pictures, they contain portions of cars, buildings, roads, street signs, pedestrians, trees, and sky. The difference between two such samples will depend on the types of objects visible in the images. Additional variation will come from the same sources described above.

Figure 2-8 gives an actual example of how significant these sources of variation are on a stationary object. The images shown are all the exact same building taken from the same point of view, but are under very different conditions. The distances across the images are quite large, and as Figure 2-9 shows, some of the pictures of the building have smaller distances to some car images than they do to each other.


Distance measures across samples

				
	0.00	17.2	26.4	24.1
	17.2	0.00	23.5	20.6
	26.4	23.5	0.00	26.5
	24.1	20.6	26.5	0.00

Figure 2-8: Four pictures of an identical object taken at different times of year and under different lighting conditions. Distances from each image to the others are displayed alongside.

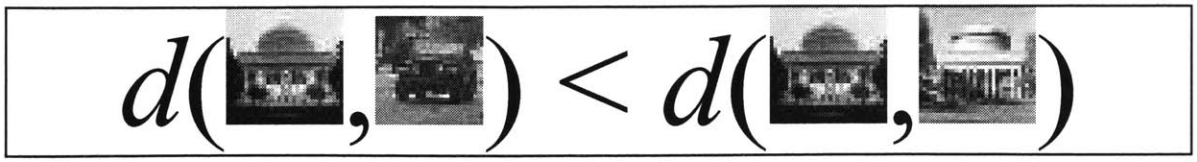


Figure 2-9: The distance from one of the dome images is closer to the picture of the car shown than it is to any of the other pictures of the dome.

2.6 Cost of labeling

The task of labeling can be very tedious, particularly for some domains. For an image classification problem, it is easy for a human to identify the label, but may be difficult to get the samples in the appropriate format. In the domain used here, preparing a labeled sample requires looking through an image, rescaling to normalize the size of the car, and selecting the appropriately sized bounding box. A much more tedious task is finding which pixels in the image correspond to the desired object, called *segmentation*. This requires carefully tracing the outline of the object and identifying which regions are part of the car and which are not.

In a text classification domain, samples are very easy to collect but more difficult to identify. Text can be automatically retrieved from online sources but it takes a human time to read through and determine the appropriate label.

For any domain, a large training set can more accurately represent the true distribution of samples. This allows for a more accurate classifier that can identify previously unseen examples. Obtaining such a large labeled training set is often difficult. Obtaining a larger training set without the high labeling costs would be desirable, and partially labeled methods make this possible.

2.7 Clustering

In Section 2.4 we showed that samples from the same class are not always close together as measured by the distance metric, but with a larger collection of samples, it becomes more likely that a sample in the same class will be nearby. It is important that this condition is met for partially labeled methods to work properly.

The distribution of feasible images will lead to clustered regions in the transformed high-dimensional feature space. For example, images of light cars driving during the day on country roads may occupy a region of the feature space. A different

cluster will be formed from images containing portions of the road. Within any cluster, all of the images will be similar in the high dimensional space, but there is not as much similarity from one cluster to another. Most importantly, samples within the same cluster should all have the same label.

If the features were randomly chosen, then all of the regions of the space would have the same density. However, there is dependence among the features in an image, allowing the clusters to form. The 16x16 and 32x32 features for nearby regions are often very similar. Also, because the representation uses overlapping image windows, adjacent features will be similar to one another, as they are based on similar calculations.

Partially labeled methods exploit the concept of clustering. The feature space is filled with clusters like these. In general, points within the same cluster will have the same label, and hopefully the same is true for nearby clusters. The distribution of the data is analyzed to find the clusters, and using the assumptions, the unlabeled points can be identified. Unfortunately, real life data sets don't always split into easily distinguishable clusters, which is why it is very difficult to get a classifier with perfect accuracy.

2.8 Measuring performance

In determining how accurate a classifier is, it is necessary to know how the system will be used. If an application that sorts email into various categories makes a mistake, the repercussions are perhaps that someone will need to manually re-sort some of the messages. At worst, this will cause some lost productivity from wasted time. On the opposite end of the spectrum, a mistake in a medical diagnosis system could lead to death.

Frequently, there is a significant difference in social costs between mistakenly labeling a negative sample as positive, or vice versa. The car detecting task demonstrates this well, depending on how it is used. One potential use of the system is to indicate to a truck driver whether a car is in his blind spot, which can help determine whether it is safe to change lanes. If the system beeps when the driver turns on the turn signal and a car is in the blind spot, then a false positive (i.e. saying there is a car nearby when there isn't) would prevent the driver from changing lanes when he should be able to. This causes no

harm, but a false negative, where a car goes undetected, could cause a collision, which is clearly far worse. Another way the system can be used it to inform truck drivers any time a car is in their blind spot. Frequently reporting false positives will become such a nuisance that most drivers will probably want to get rid of the detector.

Further complications arise when seeking to find a numerical measure of the performance. Measuring the percentage of samples classified correctly can be misleading when the classes are unbalanced. Consider a classifier used by a weather station in a desert that predicts every day whether it will rain. If it rains only once every thousand days, then a classifier can have 99.9% accuracy by saying it will never rain. However, it will never correctly guess when it will rain, which is far more relevant. In cases like this, it is useful to observe a Receiver Operating Characteristic (ROC) curve. This measures the portion of positive examples that the classifier correctly identifies along with the number of negative samples that are labeled as positive. Some algorithms allow some flexibility in how much to lean towards one class or the other, yielding a curve that compares false positives to false negatives.

For a classifier that has false positive and false negative rates that are approximately the same, it may be sufficient to simply report the total error rate. When the errors are much more one sided, it will be interesting to look at why this is so.

2.9 Partially labeled learning methods

Training a machine learning algorithm with partially labeled methods and fully labeled methods have similarities, but differ in the way the data is analyzed. The common thread between the two is that some samples are given with a known label, and there are additional samples where the label is desired.

With fully labeled data, an algorithm is trained on the labeled data only. Then the unknown points are tested one by one to predict the labels. These points are not incorporated in any way into the training set, so essentially the test points are classified only using information about the training set, which can be small.

Partially labeled methods, however, take all points into consideration when labeling the test points. They have all of the data available to begin with, and hence can easily take advantage of the clustering structure.

Each method has advantages over the other. Supervised learning methods are practical for fast real-world applications. For example, a system designed to recognize new images in real time does not have all of the data up front. The classifier can only base its information on images it has already seen. Contrast this with a system that must go through a collection of images and determine the labels. This has all the information initially and can use this fact to improve performance.

Often, the labeled samples make up only a small fraction of the total data set. Because partially labeled methods take all points into account as opposed to just the training set, they can be much slower than a fully labeled method in labeling a single point. This will typically be prohibitive for real-time use.

Inductive learning methods allow classification on test points that have not previously been seen. Fully supervised methods fall in this category. Alternatively, semi-supervised methods are transductive, which allows training and testing with unseen points. Some methods are strictly transductive, which prevents testing on points that were not available when training. Some methods are in both categories, making them more applicable towards real-world applications.

A method that is both inductive and transductive will be very useful for our car detector. It can be loaded with a number of labeled examples of cars and non-cars. Then, an on-board camera can collect an arbitrary amount of unlabeled samples. Offline, the system trains using this data, and finally can test on unseen points during future uses. The training time will be slow, which is acceptable because training occurs offline, while the car is not driving. What is more important is the real-time performance of labeling new samples.

2.9.1 K-nearest neighbor

A simple approach to identifying an unlabeled set of samples is the k-nearest neighbor algorithm. In this algorithm, each point finds the k closest labeled points according to the distance metric (where k is a pre-selected parameter).

In the inductive variation, the label for a new point is what the majority of the neighbors are labeled, considered a voting scheme. Alternatively, closer points can be weighed more heavily. For example, each point can be given weight inversely

proportional to the distance, or according to a Gaussian distribution. Our implementation weighs the points equally.

A similar approach is followed in the transductive version of k-nearest neighbor. All unlabeled points are given a proposed label based on the neighbors along with a measure of how certain that label is. The certainty measure is based on how many neighbors have each label and their distances. A point with all neighbors nearby and in agreement on the labels is assigned a high certainty measure. At each iteration, the unlabeled point with the highest certainty is labeled and added to the training set for later iterations.

One flaw with the nearest neighbor approach is that if the classes are unbalanced, it may tend to favor the larger class. One point may be mislabeled at an early iteration, which will cause its neighbors to be mislabeled, and this process will cascade. This can cause large skews in the test data and lead to poor results. Two ways to fix this are ensuring that the initial data is roughly balanced, and alternating assigning labels to positive and negative points at each iteration as long as that is possible.

An advantage of nearest neighbor is that any number of classes can be accommodated, but more data may be necessary for the classifier to work well. In order to be able to compare nearest neighbor with the other algorithms, we leave it with just two classes for the image detection, cars and non-cars.

2.9.2 Transductive Support Vector Machine

Support Vector Machines offer a way to train a classifier that maximizes the separation between the classes. This works in high dimensional space without requiring a large training set [Vapnik, 1998]. Some common applications of SVMs are text classification [Joachims, 1998] and image detection [Papageorgiou, 2000].

SVMs attempt to form the widest possible linear separator between the classes. This is done by maximizing

$$\sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) \qquad \textbf{Equation 2-2}$$

subject to $\sum_i \alpha_i y_i = 0$ and $0 = \alpha_i = C$. This is essentially finding the widest separating boundary between the classes, with some penalty for a point being on the wrong side of the margin.

The α values are weights for each sample, C is a global parameter, and x and y are the samples and their labels. Many classification problems will not have any way to separate the data linearly, so the dot product $(x_i \cdot x_j)$ is replaced with a kernel function, $K(x_i, x_j)$, which maps the points into a higher dimension space where the data can be separated. The kernel functions will be discussed in further detail in the next chapter.

After the training phase is complete and the α values have been determined, test points are labeled according to the function

$$f(x) = sign\left(\sum_i \alpha_i y_i K(x, x_i) + b\right) \qquad \textbf{Equation 2-3}$$

over all of the training points i. Here, α_i is a non-negative weight for each point that is determined during training, y_i is the known label, $K(x,x_i)$ is the kernel mapping function between the test point and training points, and b is a bias term that shifts the hyperplane away from the origin.

Ordinary SVMs do not use the test examples in any way to improve classification. The classification of a new point is based only on the l labeled points. An alternative method is Transductive SVMs (TSVMs), which use all of the data samples available to construct the initial classifier [Joachims, 1999B]. The summation in Equation 2-3 is over all points, labeled and unlabeled. Because the y's are unknown for the unlabeled points, the algorithm uses information about the distribution of the points to infer the labels. It seeks to find a way of assigning labels to the unknown points such that Equation 2-2 is maximized.

The algorithm works by initially assigning labels to the unlabeled points based on a run of an ordinary SVM. Following the initial training, pairs of points have their labels swapped to increase the objective function and the algorithm is retrained. The algorithm is terminated when no pair exists that will increase the margin.

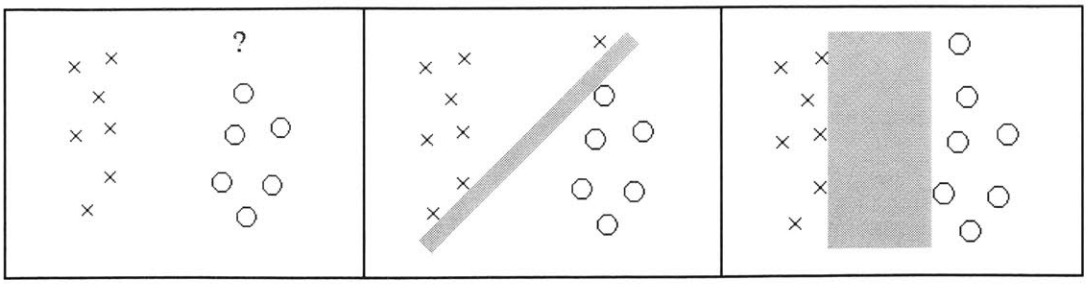


Figure 2-10: In the left figure, all points have a known label except one. An SVM that treats it as an X (right) has a wider margin than one that treats it as an O (middle), so the point gets labeled as such by a TSVM.

2.9.3 Kernel Expansion

Kernel expansion is a method that tries to find the probability of each label for any point in the feature space. It learns the distribution from all of the training points, which does not need to be fully labeled. Taking all neighbors into consideration, the label for a new point is estimated by the density of points of each class.

Specifically, the label is given to a test point by finding the label that maximizes the formula

$$P_{post}(y \mid x_j) = \sum_i P(y \mid x_i) P(x_i \mid x_j). \qquad \textbf{Equation 2-4}$$

In Equation 2-4, y is the label of the test point x, and has a value of +1 or -1. The summation is over all training points x_i. The probability that one point comes from the same Gaussian distribution as another, $P(x_i \mid x_j)$, is

$$P(x_i \mid x_j) = \frac{1}{z_j} e^{\frac{-d(x_i, x_j)^2}{2\sigma^2}}. \qquad \textbf{Equation 2-5}$$

z_j is a normalization factor such that the sum of all distances for a point is 1. Equation 2-5 has two different probability measures for a label y given a sample x. $P_{post}(y|x_j)$ is the probability that the output of the classifier is label is the probability that the output of the classifier is label y for sample j. The $P(y|x_i)$ values are parameters for each point i as computed by one of several methods [Jaakkola et al., 1999] [Szummer, 2001]:

- Maximum Entropy Discrimination (MED): This is a discriminative method that estimates values of $P(y = 1 \mid x) - P(y = -1 \mid x)$.
- EM estimation: We try to maximize the conditional log-likelihood, $\sum_{i=1}^{NL} \log P(y_i^* \mid x_i) = \sum_{i=1}^{NL} \log \sum_{j=1}^{NL+NU} P(y_i^* \mid x_j) P(x_j \mid x_i)$, where NL is the number of

labeled samples and NU is the number of unlabeled. We do so with EM estimation, which alternates steps between re-computing values of $P(y_i^* \mid x_j)P(x_j \mid x_i)$ and updating $P(y|x_i)$.

- Linear programming average margin: The margin on point k and class d, γ_{kd}, is defined as $P_{post}(y = y_k^* \mid k) - P_{post}(y = d \mid k)$, where y_k^* is the appropriate label for sample k. This method tries to maximize the average margin over all points, which will force as many points as possible to be correctly labeled. This offers a closed-form linear program, which makes running time very fast.
- Linear programming minimum margin: This method is similar to average margin, but seeks to maximize the smallest margin. There is very little noise-tolerance because the point with the smallest margin could have an incorrect label or appear to fall in the opposite class, which can cause very poor results.
- Consistent: This parameter estimation method seeks to keep the parameter estimates consistent with the corresponding posterior probabilities.

As seen in Equation 2-5, kernel expansion has a σ parameter which controls how much points influence one another based on their distance. Most algorithms use a single value of σ for all points. Kernel expansion has a feature called adaptive σ which allows each point to have a unique σ value. The values of σ get smaller as the region around a point gets more densely packed. This allows points in the sparse regions of the feature space to receive a larger influence from points that are more distant.

One method, KNN, places a kernel around training points, with the kernel width determined by a parameter *kfrac*. If there are N training points, the distance to the (*N*kfrac*)th point is multiplied by another parameter, *sigmamult*, to establish the kernel width. Another adaptive method is BMP, which works in a similar way, but places the kernels around the points in space for testing.

2.9.4 Markov Random Walk

The fundamental idea of the Markov random walk algorithm is that points are locally connected to nearby points, and can reach other points globally by a series of steps through neighbors. After several steps, one point can be reached by another through any number of paths, with higher likelihood assigned to a path with the points

closer together. One can determine the probability that a path ending at a point originated on a path from any other point. Considering this over all of the initially labeled points gives a way of assigning a label.

Transition probabilities are assigned based on the distance metric. Each point i is connected to a small number of neighbors j, with a weight of

$$W_{ij} = e^{\frac{-d(x_i,x_j)}{\sigma}}, \qquad \textbf{Equation 2-6}$$

and the single step probability of a transition from point i to j is

$$P(x_i, x_j) = \frac{W_{ij}}{\sum_k W_{ik}}. \qquad \textbf{Equation 2-7}$$

For pairs of points that are not connected by a single step, $W_{ij} = P(x_i, x_j) = 0$, and for a point to itself, $W_{ij} = 1$, which can be obtained from Equation 2-6 by noting that $d(x_i, x_i) = 0$.

The multiple step transition probabilities are computed from the single step probabilities. The probability of reaching point j on a t-step path originating from point i, $P_{t|0}(x_j \mid x_i)$, is found as entry (i,j) of A^t, where A is the transition matrix constructed from the values of P. From this, one can directly compute the probability that the path originated at i given that it ended at j, $P_{0|t}(x_i \mid x_j)$.

The equation used to determine the probability of the label is similar to that of kernel expansion,

$$P_{post}(y \mid x_j) = \sum_i P(y \mid x_i) P_{0|t}(x_i \mid x_j). \qquad \textbf{Equation 2-8}$$

The values for the $P(y \mid x_i)$ parameters are assigned by the same methods as kernel expansion, while the representation, $P_{0|t}(x_i \mid x_k)$, is computed as described above.

An advantage of obtaining global distances by stepping through local neighbors is that it gives the ability to classify data sets that are not easily separable by other methods. This allows a different way of measuring the distance between two points than a Euclidean measure.

One way that Markov random walk differs from other algorithms is that it can only be used to evaluate points that were available at training time. Methods such as

SVMs allow for inductive testing of new points, and kernel expansion can be estimated at any point in space, but the nature of the Markov random walk does not allow this.

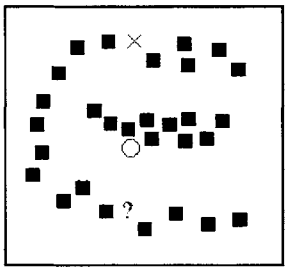


Figure 2-11: The unknown point (?) is much closer to the O than it is to the X by a Euclidean measure. However, based on the location of the other points, it seems more likely that it is in the X class. Using Markov random walk to step along paths to neighbors would be able to determine this.

2.9.5 Spectral Clustering

There are a number of methods that can be used to cluster the sample points and find labels accordingly. The algorithm that is used here is a variation of spectral clustering [Ng et al., 1999]. In spectral clustering, an affinity matrix is created based on the distances between pairs of points. To separate the data into clusters, the eigenvectors corresponding to the k largest eigenvalues of the Laplacian of the affinity matrix are combined to form a new feature vector for every sample. These transformed points are then separated into the appropriate clusters by any appropriate algorithm. Figure 2-12 gives a simple example of how clustering works.

The clustering does not take the labels into account, but simply attempts to separate the clusters into linearly separable regions. The algorithm can be seen as essentially mapping each data point to a new k-dimensional point. Once this is done, any of the above methods can be used to attempt to separate the clusters. We use a linear SVM in the tests.

Spectral clustering, like Markov random walk, can only be tested on points available during training. Each sample is mapped to a new point by finding the eigenvectors of a matrix. There is not a linear mapping between the original feature space and the transformed one. Because computing the eigenvectors is a large portion of the computation time of the algorithm, it is not easy to incrementally add points.

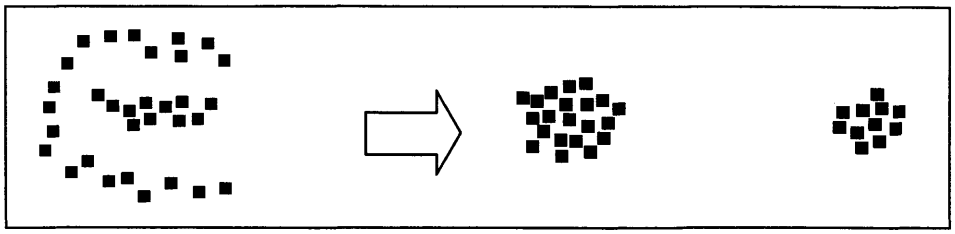


Figure 2-12: Spectral clustering takes a set of points and maps them into a transformed feature space that is easier to separate into classes. The original feature space typically has higher dimensionality than the transformed space.

Chapter 3

Experimental Results

As labeled samples are added to a training set, the performance of any algorithm is expected to improve. If the partially labeled methods are working as they should, adding unlabeled samples will also help out the performance. A large focus here is to determine how the size of the data set affects each of the algorithms being examined.

To see how the amount of labeled data relates to performance, we take an unlabeled set of fixed size and vary the number of labeled samples across a reasonable interval. Once the classifier has been trained with the labeled and unlabeled data, it is run on a test set to determine its accuracy. For consistency, the same test set of 400 samples is used for all of the algorithms, with an equal mix of positive and negative samples. The total error is measured rather than individual statistics for false positives and false negatives. The test points can, and often do, overlap with the unlabeled points used in training, but are never the same as a labeled training point.

It is very important to see how much help the algorithms receive from unlabeled points. Most of the time, adding labeled samples will help the performance far more than adding unlabeled samples. The point of the partially labeled methods, however, is to remove some of the burden of labeling so many points. For example, it will be interesting to observe whether an algorithm with 32 labeled points and a large amount of unlabeled data can reach the performance levels of 64 labeled points.

The importance of unlabeled data is found in a similar way to the labeled samples. The labeled set is fixed at a specified size and unlabeled samples are added. For algorithms with an inductive counterpart, the accuracy is measured when zero unlabeled samples are supplied, to treat as a baseline for comparison. The charts plot the error as a function of the number of unlabeled points, with different curves corresponding to different sizes of labeled sets.

Some algorithms have limitations that restrict the sizes used, and these will be mentioned in the discussion. Markov Diffusion and Spectral Clustering can only be

tested on points that are available at train time, so the accuracy of these algorithms is determined based on the exact set it was trained with. This will be important to keep in mind when comparing the different algorithms.

The performance depends highly on which particular samples are used in training, not just the size of the data sets. For example, if there is not very much diversity in a large set of labeled samples, the performance could be far worse than a smaller set. The points selected are all randomized, but to dampen the effect of test variability the statistics reported are averages over about 20 trials. When necessary, some tests will be looked at more closely to understand the results.

3.1 Result format

The data results are plotted for each algorithm, with the appropriately chosen parameters. Each set of results shows four graphs, with more detailed graphs focusing in on certain regions whenever necessary. The first such graph can be seen as Figure 3-5.

The graph in the upper-left corner is an average over 20 trials to show the effects of adding labeled data. The test classification error is plotted against the number of labeled samples used, sampled at all powers of 2 from 2 to 256. There is an equal balance of positive and negative data in these labeled samples. In the lower-left corner, the error is again plotted over the number of labeled training points, but this is for a single trial with the sample size increasing at each step linearly, not geometrically. The scale runs from 4 to 100, which does not fully cover the graph above. The rationale for this is that above 100 samples is a large amount of labeled data and the purpose of this work is to determine how well algorithms work without requiring so many points to be labeled. We still use more than 100 labeled points in some tests to observe the patterns from adding labeled data.

In both graphs that vary the number of labeled samples, there are four separate curves, corresponding to 0, 256, 512, and 1024 unlabeled points. Over a trial, the unlabeled set remains constant while the labeled points vary. These points are chosen as the "region of interest." It is interesting to see the performance with 0 unlabeled points, which are fully inductive methods. For example, a TSVM with no unlabeled samples becomes an ordinary SVM, and Nearest Neighbor does not use any bootstrapping. Other

interesting values to look at are not too small, because the algorithms really won't be expected to benefit much unless enough unlabeled data is used. The size is limited above by time restrictions and the amount of data available. Sample sizes from 256 to 1024 best match these criteria.

The upper-right graph plots the error over 20 trials against the number of unlabeled samples. The number of unlabeled samples goes from 2 to 1024, doubling at each step, as well as 0 to measure the baseline performance. The lower-right graph plots error while varying unlabeled samples for a single trial, from 0 to 1200 samples, increasing by 30 or 40 at each step. There are unlabeled data learning curves corresponding to 16, 32, 64, and 128 labeled samples, which are the region of interest here. These values are chosen because they are not so small that they lead to poor performance, but not so large that the labeling would become a burden in an actual system.

There is overlap in the two charts (see Figure 3-1), although points may not correspond exactly because they are taken as averages from separate trials. Because the values are averages over 20 trials, they tend to be in agreement, but occasionally sampling error causes a difference in the values. What is more important is getting an idea of the general shape of the curves, so the sampling error does not hurt the analysis significantly.

Because the lower graphs are based on a single trial, they are not always generalizable, but will give some insight into the algorithms. For example, it will give an idea whether the error drops sharply when a certain sample is added or whether there is a more steady but gradual improvement.

In the ensuing discussion, the number of labeled samples will often be denoted by NL, and the number of unlabeled samples by NU.

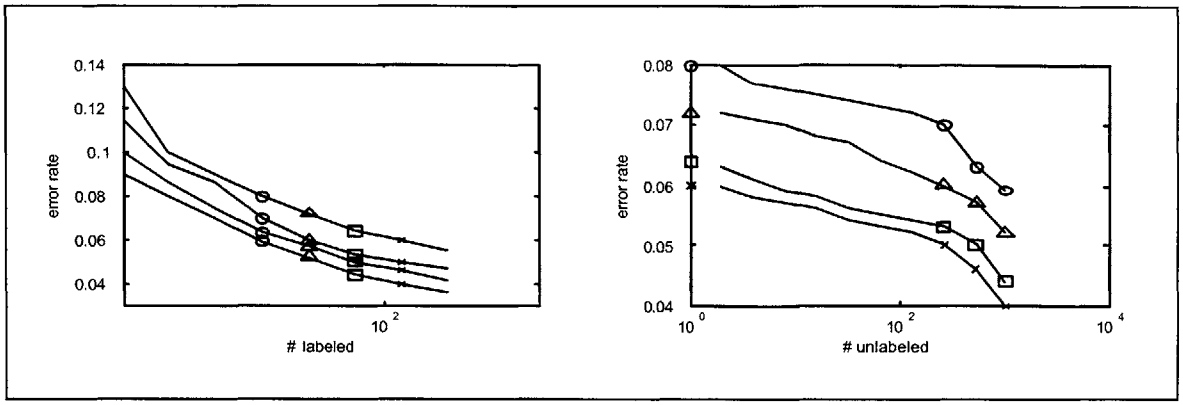


Figure 3-1: Sample charts plotting error rates over the number of labeled and unlabeled points. Corresponding points between the two charts are represented by the same symbols (e.g. all points marked with an X represent 128 labeled points and various numbers of unlabeled points).

3.2 Parameter selection

We attempt to find the parameter settings that give the lowest error for each test by observing performance over several trials with randomly selected samples. To choose which parameter setting is optimal, we pick a target size for the data set. Unless otherwise specified, this is always 64 labeled points and 1024 unlabeled points, which is a reasonable size for a real life data set.

To make a perfect comparison, the optimal parameters should be determined and set for all sizes of training sets. However, once the parameters are set, they are used for all tests on a particular algorithm, regardless of the size of the training set. A real-life application that is striving for the best possible accuracy may take much more care in setting the parameters for training sets of all sizes.

We run several trials on data sets of the selected size and measure the performance with multiple parameter settings. To prevent any sampling error from the particular points used in training and testing, the performance is always measured on identical training sets for each parameter setting. Otherwise sampling error could have a significant effect on a small data set.

After running the tests, we plot the error for each parameter setting over each trial, and look at how often each parameter had the lowest error, second lowest, and so on. Sometimes, one choice of the parameter is clearly best, as it almost always attains the lowest error (as in Figure 3-2A). More frequently, there will be several settings of the parameter that alternate for the best classification. In cases like this, we select the

parameter choice that most frequently comes near the best. Some settings of the parameter may do better in some cases, but very poorly in others (See Figure 3-2B). In cases like this, we use a parameter that comes close to the best performance over all trials.

The test set that is used to determine the parameter settings is separate from the set that is used in the ordinary tests. This prevents the algorithms from overfitting to a particular part of the data set.

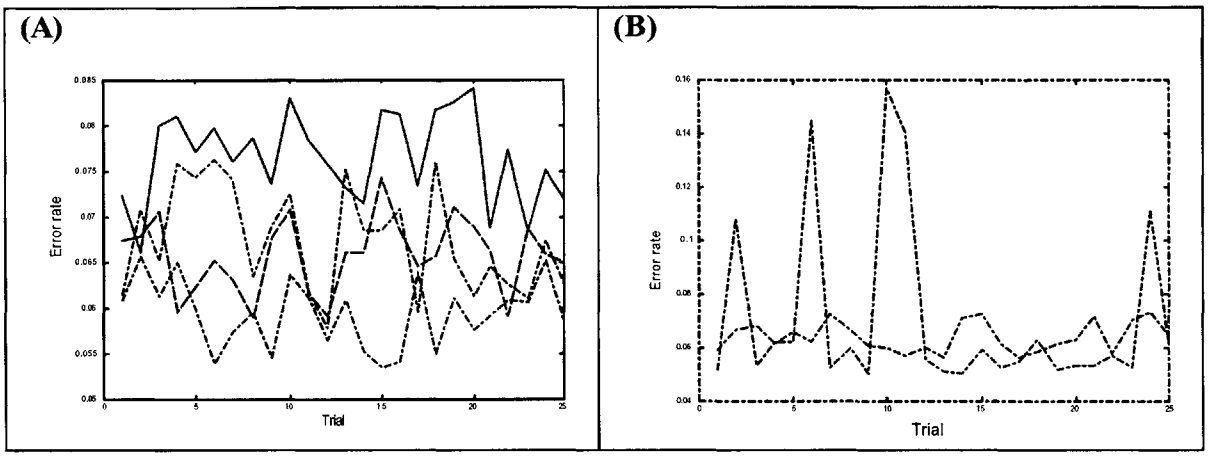


Figure 3-2: (A) Sample results shown for 25 trials using four different parameter settings. The parameter setting indicated with the solid line generally has the best performance, so that parameter setting will be used for further tests. (B) Sample results using two different parameter settings. The results marked by the solid line do not generally do as well as the dashed line, but in some trials the dashed line has very poor results. Because the solid line rarely has a trial that does this poorly, its parameters will be used for further tests.

3.3 K-nearest neighbor

3.3.1 *k* parameter

An important parameter in nearest neighbor is k, the number of neighbors to consider for each point. Different values of k will lead to different levels of performance. In order to avoid ties in voting where exactly half of the neighbors belong to each class, an odd number is generally used so that one class will always have a majority when dealing with only two classes.

When k is not large enough, the classifier will be highly susceptible to noise. For example, when $k = 1$, each point is labeled in accordance with its single closest neighbor. Samples in regions near points of the opposite class are likely to be misclassified. These borderline points are more difficult for any classifier to label, so a few errors of this type

are acceptable. A more serious problem is when points are mislabeled. For an unlabeled point that lies in a cluster of one class, if the nearest point is the opposite class and $k = 1$, it will be labeled the same as the single point, not the rest of the cluster, as in Figure 3-3A.

Larger values of k will be more tolerant of noise so that a small number of mislabeled points won't throw everything off. The computational requirements increase only slightly with a larger k. However, if k is too large, then each point's label will be based on a large number of points. For data sets without enough samples in each class, something may get mislabeled simply because of the lack of data, as Figure 3-3B demonstrates. We set up our tests such that this effect is minimized.

Experimentally, we find that k=3 or k=5 work well on this data set, so we run further tests with these two values.

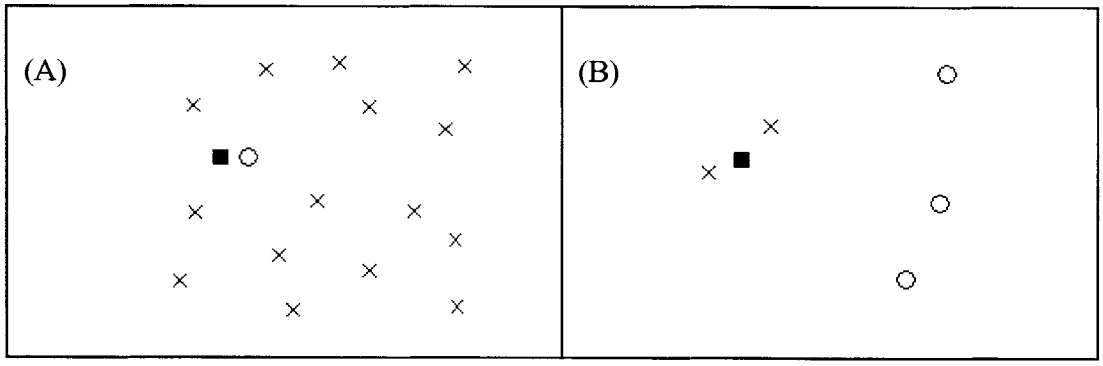


Figure 3-3: Scenarios where nearest neighbor does poorly. Labeled points are X and O, and unlabeled points are represented as ¦ . (A) Using k=1, the unlabeled point will be assigned to the O class, even though it lies in a large X cluster. (B) If k=5, the unlabeled point will be labeled as O because there are not enough X points in the figure, even though the two X points are much closer than an O.

3.3.2 σ parameter

With an appropriate value of k chosen for this task, the next parameter to set is σ. This value is used for weighing a point's k neighbors, according to a multivariate normal distribution. This takes the distances and labels of each of the k neighbors into account, and is maximized when all k points contain the same label and have a distance of 0.

At each step of the algorithm, all points with the highest portion of neighbors with the same label are chosen, and the multivariate normal measure is used as the tiebreaker to select which point will be added to the labeled set.

If σ is too small, then the confidence measure for a point will be small unless there is a labeled neighbor very close by. The confidences will all be close to 0, and the nearest point may outweigh all others. This removes most of the benefit of using a value of k larger than 1. (See Figure 3-4)

If σ is too large, all points will appear to be the same, making distant neighbors weigh almost as much as closer ones. Clearly this is not ideal, because closer points are more likely to be from the same distribution. This fact would essentially be ignored when using a large σ.

After trying out a reasonable range of values for σ, a good value was determined to be 1 for $k = 3$ and $k = 5$. Further experiments using the Nearest Neighbor algorithm will use this value.

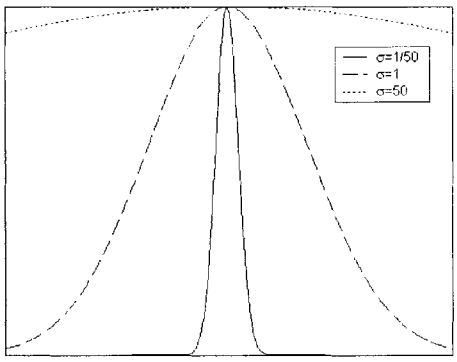


Figure 3-4: How the influence between neighbors varies by distance for different values of σ.

3.3.3 Results (k = 3)

While it is desirable to watch the performance of all algorithms when as few as 2 points are initially labeled, the structure of nearest neighbor prevents that. For a 3-nearest neighbor situation, having only 2 points initially labeled makes it impossible for a new point to consider 3 neighbors. Even with 4 labeled points, 2 of each class, there is no way for a point's 3 closest neighbors to be in agreement. The algorithm could still be used under these conditions, but it would likely do very poorly. Thus, the smallest training set used for nearest neighbor when varying the labeled size is 8. The upper bound is not affected, so we run tests with up to NL=256.

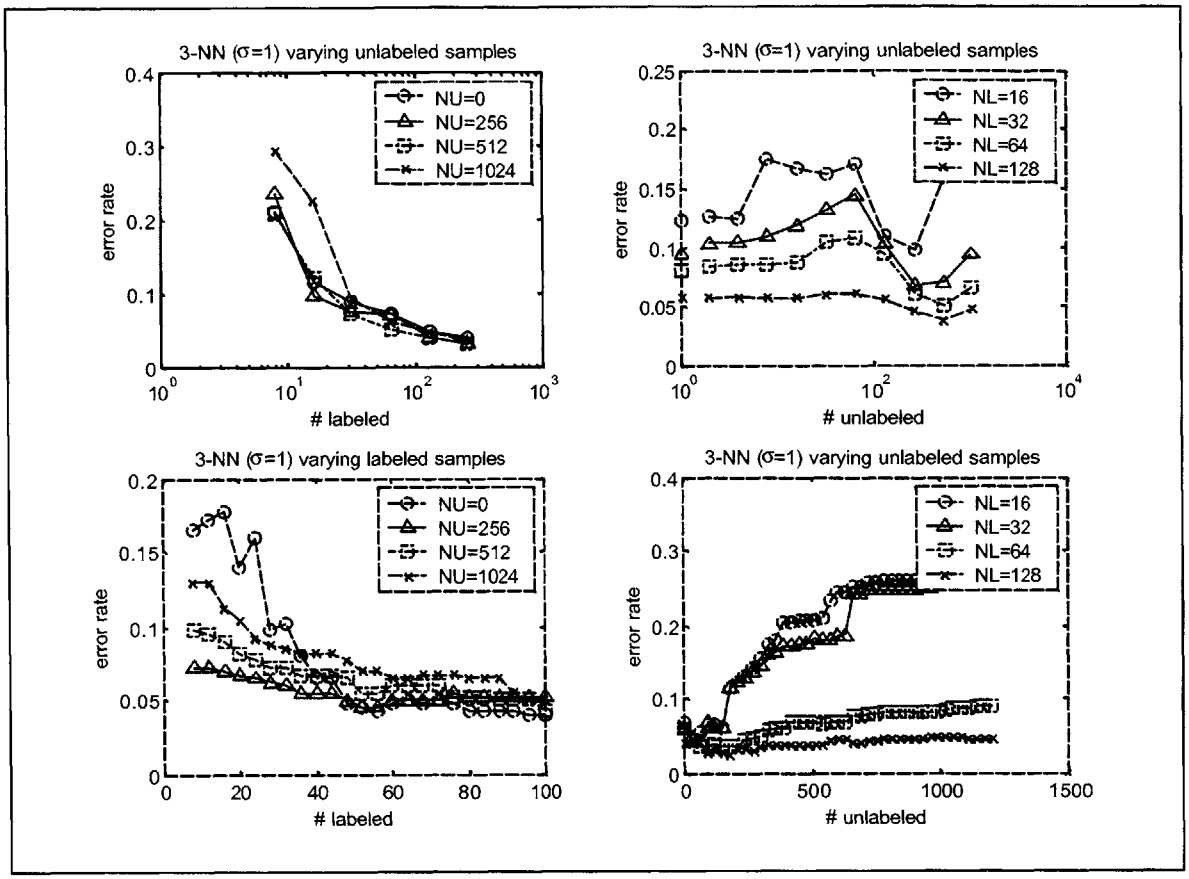


Figure 3-5: Error rates for tests using 3-nearest neighbor. $k = 1$.

Figure 3-5 displays the results for tests using $k = 3$, $\sigma = 1$. As with all tests, the results plotted are the average error ratio on the 400 test points over 20 trials for each of the parameter settings.

The performance gets better as labeled points are added to the training set, as expected. As labeled points fill up the feature space, it increases the chances that a new sample will be near a point of the appropriate class. This holds true for any number of unlabeled samples.

Unfortunately, this algorithm does not fully benefit from unlabeled data. A small amount of unlabeled data tends to perform worse than using no unlabeled data at all. The algorithm works by selecting the unlabeled point each iteration that it is most certain of the label. With very few unlabeled points, it is likely that it will be wrong at guessing a label, and subsequent iterations will be affected.

With around 128 to 512 unlabeled samples, the performance improves, because it becomes more likely that the point it is most certain of will be correct. However, with

these larger data sets, any noise will have a large effect, because the unlabeled data will overwhelm the labeled data. We start to notice this with 1024 unlabeled points, as the error rates begin to increase again.

3.3.4 Results (k = 5)

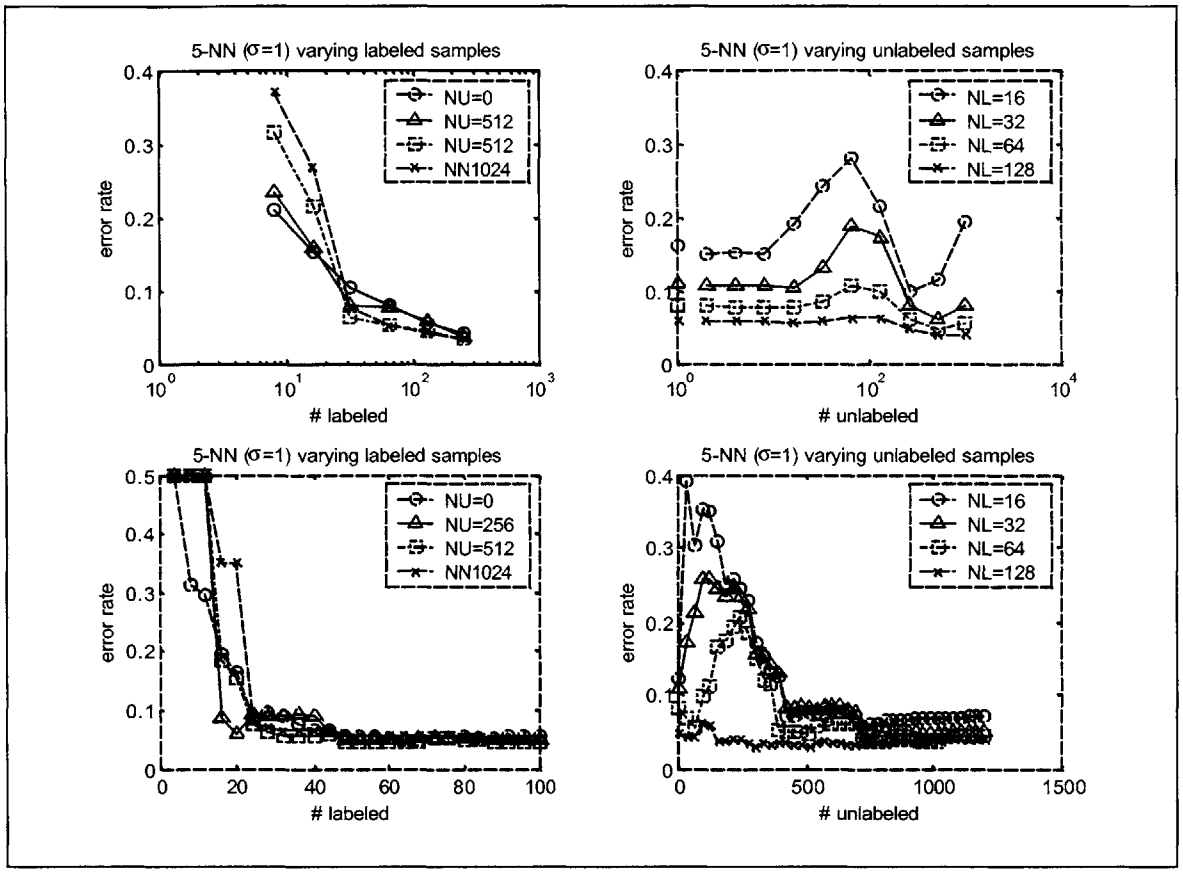


Figure 3-6: Error rates for tests using 5-nearest neighbor. $k = 1$.

The results for the 5-nearest neighbor tests are shown in Figure 3-6. The trends are the same for $k = 3$, so we will not discuss them in depth. Figure 3-7 displays some curves from the two sets side-by-side. The results appear a little better when $k = 3$ most of the time, although $k = 5$ works better when NL = 64 for most values of NU.

3.3.5 Nearest neighbor summary

With too few labeled samples, nearest neighbor does very poorly, often getting nearly 50% error. It benefits significantly from more labeled samples. Unlabeled samples affect the performance, but are not always beneficial. This is primarily because

any errors in labeling a point early on in the bootstrapping method will propagate, and it is difficult for the algorithm to recover from one of these errors. Small amounts of unlabeled data often lead to poor results because the point with the highest certainty score is less likely to be correct than when more samples are present.

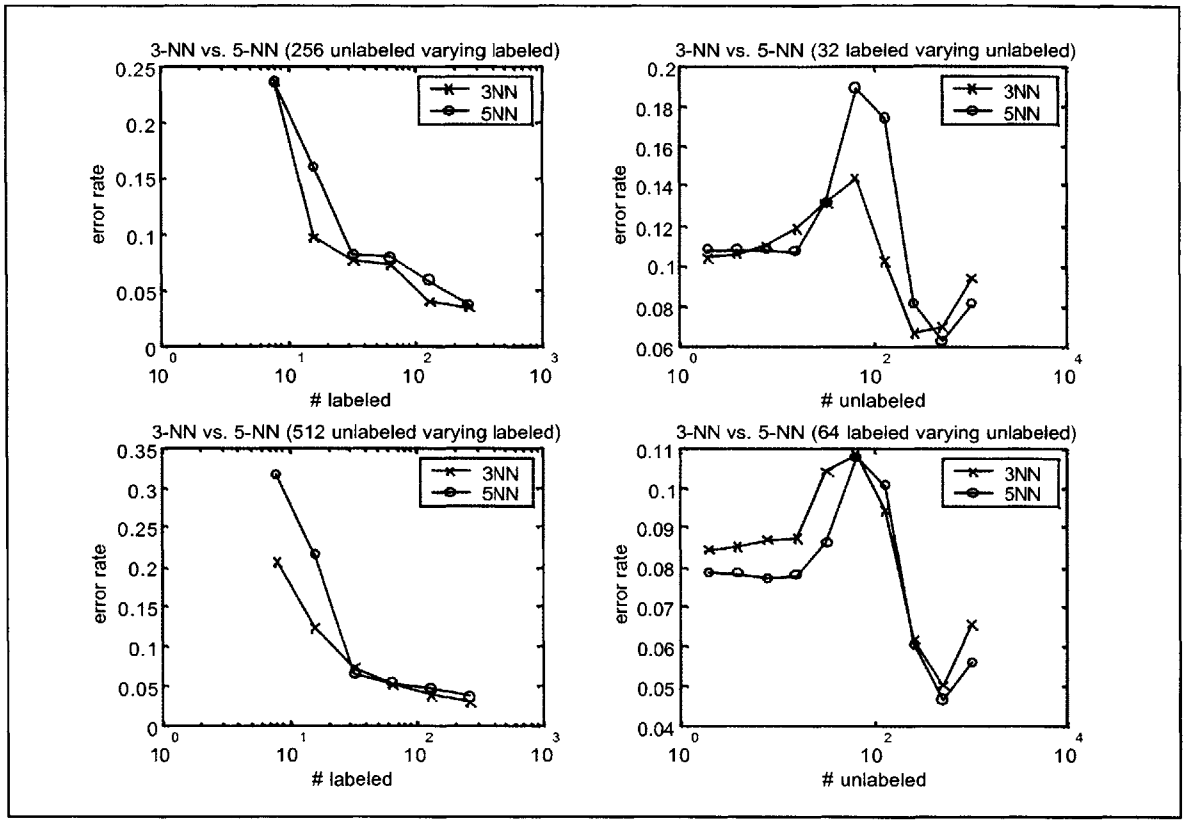


Figure 3-7: Comparison of 3-nearest neighbor and 5-nearest neighbor.

3.4 Transductive SVM

We use the SVMlight software package [Joachims 1999A] to run tests with a TSVM classifier.

Recall that after training in a TSVM, a new point is labeled with equation:

$$f(x) = sign\left(\sum_i \alpha_i y_i K(x, x_i) + b\right). \qquad \textbf{Equation 3-1}$$

The various parameters will all be described shortly.

The performance of any SVM will depend on the kernel, $K(x_1, x_2)$, which establishes how the distances between two points are weighted. Typical choices for the kernel are

- Linear: $x_1 \cdot x_2$
- Polynomial: $(x_1 \cdot x_2 + 1)^d$
- Gaussian: $e^{\frac{-(x_1 - x_2)^2}{2\sigma^2}}$

In the polynomial kernel, choosing a larger value of d gives more ability to make a separation in the data, although this can overfit to the training data (see Figure 3-8). Using a smaller value of σ in the Gaussian kernel also improves training accuracy but may overfit, giving poor test results.

We run tests on the data using linear and Gaussian kernels. The discussion below focuses on each of these individually.

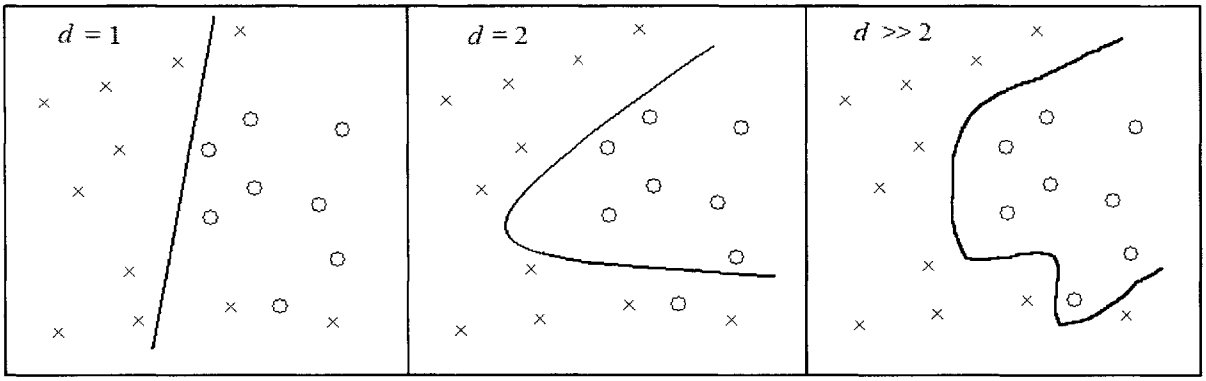


Figure 3-8: Various values of *d* for a polynomial kernel SVM. As *d* is increased, it becomes more possible to separate all of the training points perfectly, but may worsen the classification accuracy on non-training points.

3.4.1 Linear kernel

The most basic model to use involves a linear kernel, where $K(x_i, x_j) = x_i \cdot x_j$. Once this is trained, to classify a test sample, you compare the point to the training points. The label inferred for the test point is

$$f(x) = \text{sign}(\sum_i \alpha_i y_i x \cdot x_i + b). \qquad \textbf{Equation 3-2}$$

Furthermore, when the absolute value of this summation is large, the point is far from the separating hyperplane, and is more likely to be accurate.

3.4.1.1 Parameters

A parameter that must be set for any kernel is C, the tradeoff between small training error and a large margin. The setting of C limits the possible range of values for

the α_i parameters. Setting C very large will then increase the set of values that are tried and can cause longer training times. On the other hand, if C is too small, the α values will be small and it will be difficult for the algorithm to correctly label points.

Theoretically, as more training samples are added, a smaller value of C will work better, so the value of C should be changed as the training sizes change. A useful formula is $C = C_{rate}/N$, where C_{rate} is a fixed parameter and N is the total number of training samples for the current trial.

Through cross-validation, we find a good value for C_{rate} to be 500. This works well for training sets of certain sizes, but not all. There might be a more appropriate way to set C, but we use this method.

3.4.1.2 Relevant features and samples

A number of interesting questions arise in the use of a linear TSVM. How many of the initial training points, both labeled and unlabeled, are significant? Furthermore, which features of the image are most relevant? These questions can be answered by observations about Equation 3-2.

A training sample is significant when its corresponding value of α is greater than 0. In a fully labeled set you can train without all samples with $\alpha = 0$ and still classify everything the same in testing. Table 3-1 gives the number of Support Vectors (samples with non-zero α values) for different sizes of labeled and unlabeled training sets. Table 3-2 shows the same results, but gives the portion of total points that are Support Vectors rather than the absolute number. These values are determined from just one trial, but there is not a large amount of variation across trials.

Table 3-1: Number of support vectors in a linear kernel TSVM. C_{rate} = 500.

# unlabeled points \ # labeled points	2	4	8	16	32	64	128	256
0	2	4	8	12	19	30	48	73
2	4	6	9	12	19	31	49	73
4	6	8	10	14	20	32	49	73
8	8	11	12	14	19	32	51	75
16	15	15	16	19	22	37	52	74
32	21	22	19	25	28	43	53	77
64	36	33	28	32	33	48	62	79
128	46	43	42	43	44	58	69	85
256	70	63	63	64	66	77	87	94
512	103	101	87	93	99	99	113	109
1024	158	144	132	131	138	145	141	151

Table 3-2: Portion of total points that are support vectors in a TSVM.

# unlabeled points \ # labeled points	2	4	8	16	32	64	128	256
0	1.00	1.00	1.00	0.75	0.59	0.47	0.38	0.29
2	1.00	1.00	0.90	0.67	0.56	0.47	0.38	0.28
4	1.00	1.00	0.83	0.70	0.56	0.47	0.37	0.28
8	0.80	0.92	0.75	0.58	0.48	0.44	0.38	0.28
16	0.83	0.75	0.67	0.59	0.46	0.46	0.36	0.27
32	0.62	0.61	0.48	0.52	0.44	0.45	0.33	0.27
64	0.55	0.49	0.39	0.40	0.34	0.38	0.32	0.25
128	0.35	0.33	0.31	0.30	0.28	0.30	0.27	0.22
256	0.27	0.24	0.24	0.24	0.23	0.24	0.23	0.18
512	0.20	0.20	0.17	0.18	0.18	0.17	0.18	0.14
1024	0.15	0.14	0.13	0.13	0.13	0.13	0.12	0.12

While the absolute number of support vectors tends to increase as more samples are added, the portion of total points falls. When the data set is smaller, adding a new point is likely to be significant. However, with a larger data set, any new sample is more likely to be similar to an existing point and add very little help to the classifier. In general, a high proportion of samples being Support Vectors with large training sets is evidence that the classifier may be overfit.

The values in Table 3-1 are based on a single trial for each data set. Because of this, we can't infer whether the number of support vectors depends on the total number of points, or the number of the points that are labeled and unlabeled. To find this out, we choose a random set of data and vary the portion of points that are labeled, measuring

how many points are support vectors. If only the number of points matters, there shouldn't be much variation in how many points are support vectors. If the labeled and unlabeled points matter then the number of points that are support vectors will vary. We take 1000 points with 50 initially labeled and at each step label 50 more until everything is labeled. Figure 3-9 presents the results. As more of the points are labeled, more become support vectors, implying that the ratio of labeled and unlabeled data affects how many points are support vectors.

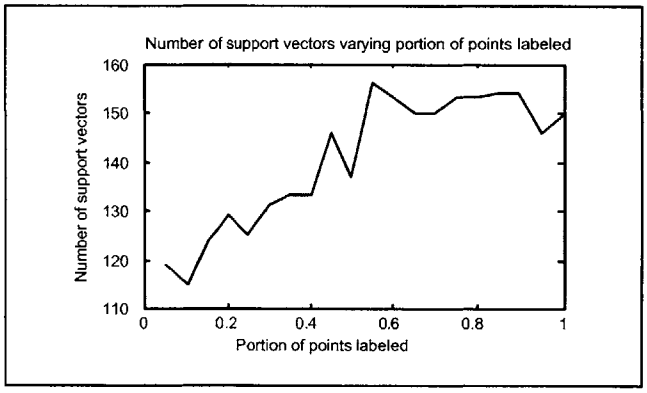


Figure 3-9: Number of support vectors for a fixed data set when varying which points are labeled and which are unlabeled

One may also wish to determine which features are important to the labeling of a new point. To do so, you can rewrite the summation in Equation 3-2 as

$$\sum_i (\alpha_i y_i x_i) \cdot x + b = \sum w \cdot x + b = w \cdot \boldsymbol{b}. \qquad \textbf{Equation 3-3}$$

w is a vector with the same dimensionality as a sample. Essentially, a template image is created which each test image is matched against. For a feature with a positive value for w, presence of that feature in a test image pushes it more towards the positive class. On the contrary, a negative value of a feature in w means that presence of the feature pushes it towards the negative class. It is possible to observe w as an image, and expect to see positive values where an image of a car is likely to contain an edge (e.g. the outline of the car or the horizon), and negative values where it is unlikely to see any edges (e.g. the hood of the car, windshield, road). Figure 3-10 shows this for one run of the classifier. As expected, regions corresponding to the outline of the car can be seen to be positive.

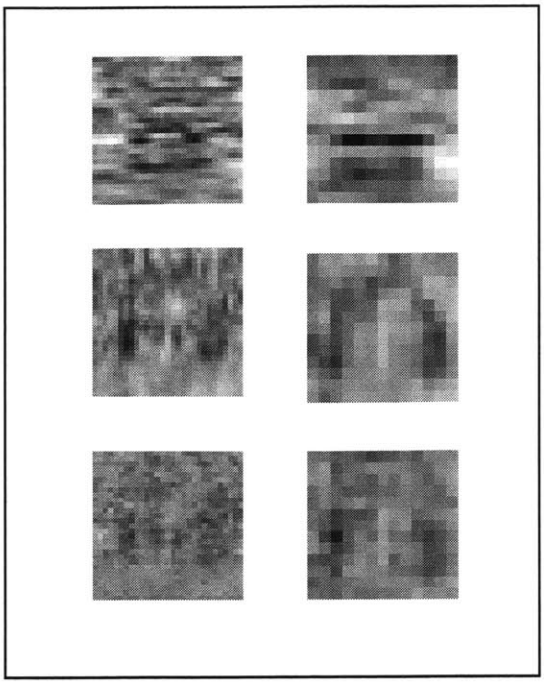


Figure 3-10: Values of w found from a linear kernel run of TSVM, plotted as an image based on wavelet features. Black is the most positive value, white is the most negative, and grey is approximately zero.

3.4.1.3 Results

The results of the TSVM tests are plotted in Figure 3-11[1]. Adding more labeled samples helps almost all of the time. Looking at the average results, the error drops when moving across the axis varying labeled points, and curves corresponding to more labeled samples are further down in the plots varying unlabeled. However, looking at an individual test run, adding labeled samples seems to have an oscillatory behavior, where adding samples typically lowers the error but occasionally causes the error to rise. The separating margin may get shifted slightly from one of the samples being added, which is responsible for these changes. There is a fairly big rise in the error for all of the curves around the 36^{th} labeled point, which implies that one of the points added changes the margin in a way that hurts performance. As samples are added beyond this, the error drops back down.

[1] The scale differs from that in Figure 3-5, so be careful in comparing these charts directly. Figure 3-48 and Figure 3-49 compare all algorithms on the same scale, which may be more useful in comparisons across algorithms.

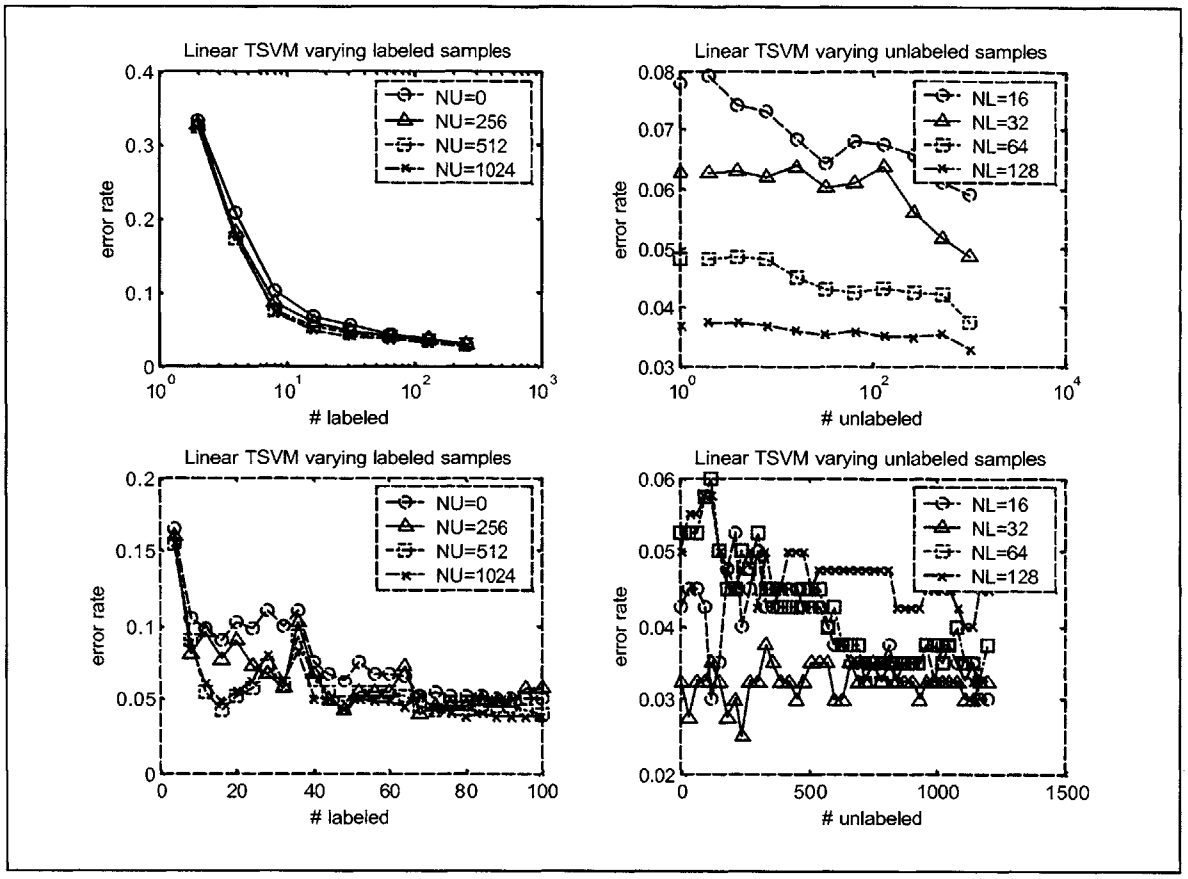


Figure 3-11: Error rates for tests using linear kernel TSVM. $C_{rate} = 500$.

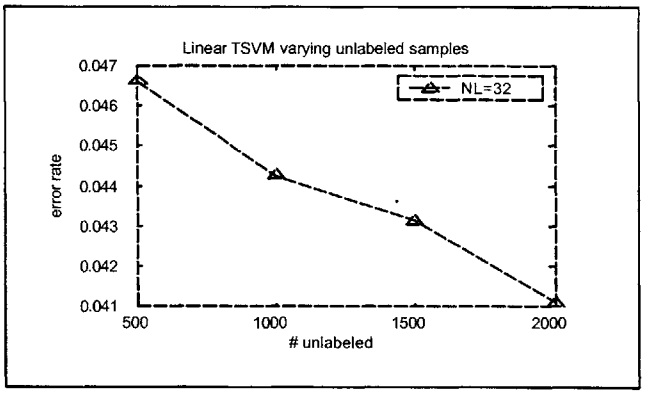


Figure 3-12: Error rates for tests using linear kernel TSVM. $C_{rate} = 500$.

The individual curves for 512 and 1024 unlabeled drop very sharply in the first 20 or so labeled samples and then steady off. With 0 and 256 unlabeled, the drop is more gradual. It is promising to observe that with enough unlabeled data, the classifier fares well with only a few labeled samples.

There is a clear decline in the error from adding unlabeled data, as seen in Table 3-3. It shows that running a TSVM with about 1024 unlabeled samples is roughly equivalent to doubling the labeled set and not including any unlabeled data.

The average error curves continue to decline without saturating. To see if this trend continues beyond 1024 samples, we run some tests with 32 labeled points and up to 2000 unlabeled points, with results shown in Figure 3-12. The downward trend suggests that a linear kernel TSVM can continue to benefit from unlabeled data. The only reason we do not run further tests is that our database is of limited size, but more data could always be collected. When training with 32 labeled points and 2000 unlabeled points, the average error rates are nearly as low as an SVM with 128 labeled points on this data.

Table 3-3: Average error for a TSVM with 0 and 1024 unlabeled points for different sizes of labeled sets.

	0 unlabeled	1024 unlabeled
16 labeled	.077	.060
32 labeled	.063	.050
64 labeled	.049	.038
128 labeled	.037	.033

In all of the curves varying NU, there are some points where adding more data causes the average error to increase slightly. Figure 3-13 shows a boxplot[2] for NL = 32 to determine whether this is possibly caused by some individual results affecting the averages. It shows us that the best results are about the same regardless of the size of the unlabeled set, but the median and third quartile points decrease slightly as NU is increased. Interestingly, there are still outliers at large values of NU, and if not for these points, the averages could be even lower. Unfortunately this means that occasionally a trial will have poor classification, even with a reasonably sized data set.

Even though adding unlabeled data tends to help performance, if only a small number of unlabeled samples are added, it often does worse than if no unlabeled data is used. This is particularly true with smaller NL. The unlabeled points are supposed to

[2] The box-plots graph the 1st quartile and 3rd quartile points as the bounding points of the box. The line in the center of the box is the median. Points within 1.5 * the inter-quartile range (the difference between the 1st and 3rd quartile) are within the bands extending out of the box, and anything outside of this range is an outlier, represented by a + above or below.

help get an idea of how the data is distributed, but if only a few points are supplied, it does little good.

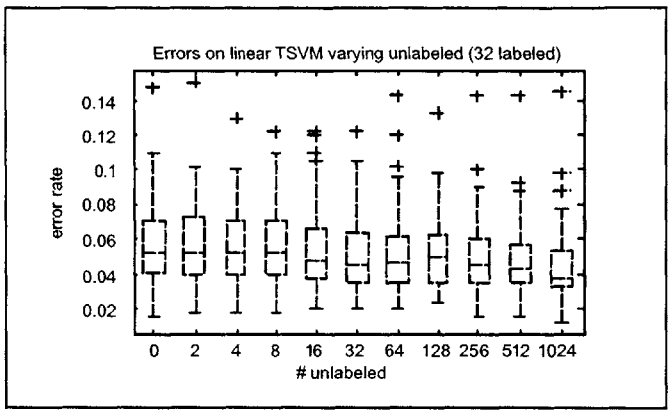


Figure 3-13: Box-plot showing error rates using linear kernel TSVM varying NU, fixed at 512 unlabeled.

The errors when only 2 or 4 samples are labeled tend to be very high. Occasionally an individual trial will label almost everything as cars or everything as non-cars, and thus have nearly 50% error. This shows that more labeled training data is necessary for a TSVM, as such large error makes the classifier useless. Ideally, a partially labeled data method would require just one sample to be labeled from each class and a large amount of unlabeled data to determine the distribution, but this does not work on this data with TSVMs.

The design of the TSVM algorithm, with a modification we have made to it, can help explain why this will not work. The first step of the TSVM algorithm is to run an inductive SVM, trained with the labeled points, and evaluated on the unlabeled points with continuous numbers as outputs. The algorithm chooses a priori how many points will be in each class. If n samples are to receive positive labels, then it places the n points with the largest outputs into this class. After this is done, one positive sample and one negative sample are chosen and their labels are switched to increase the objective function, until no more improvements can be made. Once the initial class proportions are set, they will never change. If the original proportions are not the same as the true class ratio in the unlabeled data, then some of the points will be mislabeled.

The TSVM code chooses the class proportions to be the same as the ratio of the labeled samples. When running the algorithm in this way, we have found that it is limiting in certain situations. In particular, when the class balance of the unlabeled data

is very different from that of the labeled data, as is the case in many real-world situations, the code does not work well. There is an option in SVMlight to manually set the ratio, although, again, this is not a realistic situation. We changed the code to choose the ratio based on the initial classification run. Rather than choosing the top n samples and putting those in the positive class, all of the points that have a value above 0 go in the positive class, as a normal SVM would classify them. After this, the labels are switched in the same manner as described above.

This modification is more limiting in cases where there are very few labeled points. With only two labeled points, a classifier assigns the label as whichever point it is closer to in the feature space. This is very dependent on which points are chosen, as Figure 3-14 demonstrates. Occasionally, the randomly selected points will be such that almost everything lies on one side of the margin. Even with the label switches, there is no way to recover from this and lead to accurate classification. Choosing the ratio a priori might do better in these situations, but not necessarily for larger training sets.

Another factor that limits performance on a TSVM is that there is not a global search on the potential labels, which will often lead to a local maximization of the objective function, rather than a global maximization. Performing a global search would be exponential with respect to the number of unlabeled points, which is infeasible for a large unlabeled set.

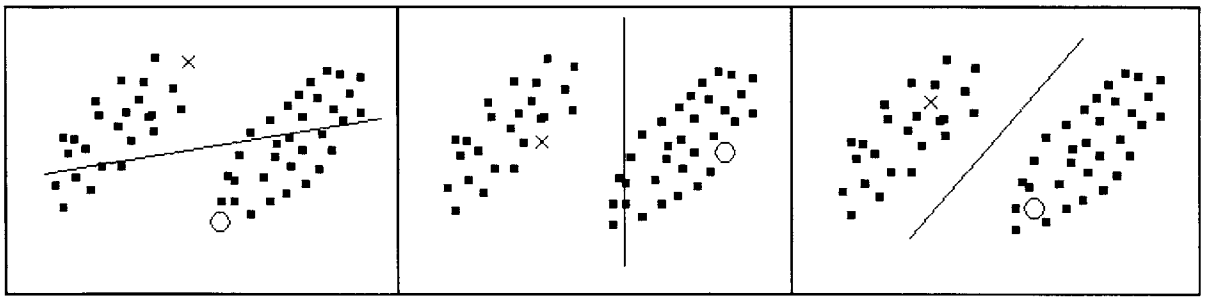


Figure 3-14: A TSVM trained on a data set with only one point from each class labeled is highly dependent on which point is chosen.

3.4.2 Gaussian kernel

3.4.2.1 Parameters

The σ parameter for the Gaussian kernel TSVM is used to determine how much a test point will be influenced by a training point depending on its distance. The value of σ

establishes how quickly the Gaussian distribution falls out. Large values correspond to a slower decay, which means that points further away will have a bigger influence. Through cross-validation, we find that 3 is a good value for σ, giving smaller errors than other values for trials with NL=64 and NU=1024.

3.4.2.2 Relevant features and samples

As with the linear kernel, it is interesting to see how many points become support vectors for different values of NL and NU. Table 3-4 gives this information for one trial and Table 3-5 gives the portion of total points that become support vectors. Far more points are support vectors in comparison to the linear kernel, which is because the kernel function allows the separation line to curve more to fit the data.

3.4.2.3 Results

The results for the Gaussian kernel TSVM are shown in Figure 3-15. All of the curves tend to improve as more labeled points are added. Some of the curves actually have a slight increase in error going from around 16 to 32 labeled, but this is a very minor change and is due to sampling error. In the individual trial plots, the error tends to drop when adding samples, but the curves for both 512 and 1024 points get worse with more data, showing better results from 4 labeled points than 100. These results are not typical, or otherwise the average curves would follow this trend as well. This does, however, demonstrate what can happen on individual trials.

The performance of this algorithm is very good, and has among the lowest error levels for smaller data sets. However, it does not perform well when many unlabeled points are provided. With 1024 unlabeled points, error is much worse than if no unlabeled data is used, regardless of how many points are labeled. It is unusual to see an algorithm's performance degrade so much from additional unlabeled data, so we analyze the results further to understand why.

Table 3-4: Number of support vectors in a Gaussian kernel TSVM. C_{rate}= 500, σ = 3.

	# labeled points								
# unlabeled points		2	4	8	16	32	64	128	256
	0	2	4	8	15	28	41	67	109
	2	4	6	10	17	29	42	68	110
	4	6	8	12	19	31	43	69	111
	8	10	10	13	22	34	46	69	112
	16	18	15	19	27	38	51	72	114
	32	27	23	30	34	44	58	81	120
	64	38	40	45	47	58	67	87	124
	128	56	77	72	72	81	90	116	149
	256	98	131	118	96	114	122	141	177
	512	159	272	204	172	179	190	200	233
	1024	290	490	433	295	309	325	350	368

Table 3-5: Portion of total points that are support vectors in a Gaussian kernel TSVM. C_{rate}= 500, σ = 3.

	# labeled points								
# unlabeled points		2	4	8	16	32	64	128	256
	0	1.00	1.00	1.00	0.94	0.88	0.64	0.52	0.43
	2	1.00	1.00	1.00	0.94	0.85	0.64	0.52	0.43
	4	1.00	1.00	1.00	0.95	0.86	0.63	0.52	0.43
	8	1.00	0.83	0.81	0.92	0.85	0.64	0.51	0.42
	16	1.00	0.75	0.79	0.84	0.79	0.64	0.50	0.42
	32	0.79	0.64	0.75	0.71	0.69	0.60	0.51	0.42
	64	0.58	0.59	0.63	0.59	0.60	0.52	0.45	0.39
	128	0.43	0.58	0.53	0.50	0.51	0.47	0.45	0.39
	256	0.38	0.50	0.45	0.35	0.40	0.38	0.37	0.35
	512	0.31	0.53	0.39	0.33	0.33	0.33	0.31	0.30
	1024	0.28	0.48	0.42	0.28	0.29	0.30	0.30	0.29

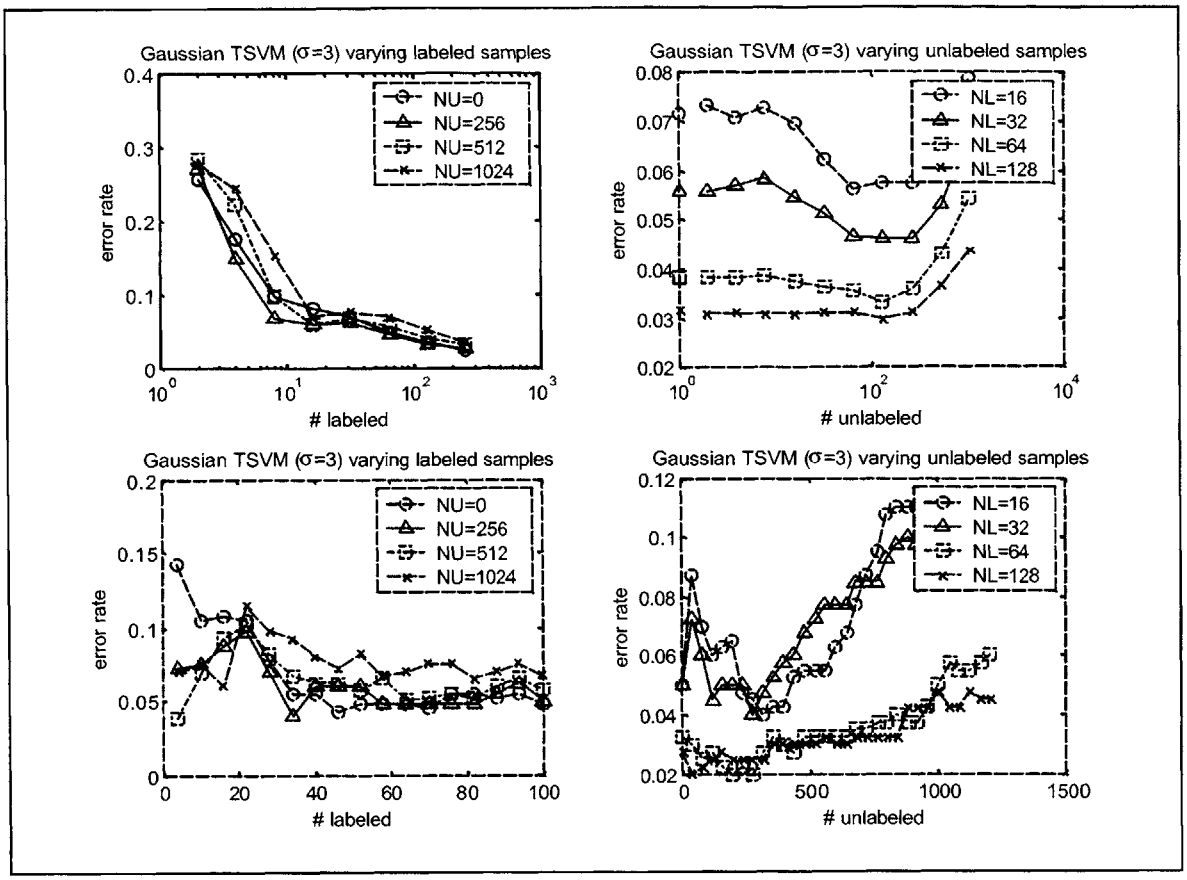


Figure 3-15: Error rates for tests using Gaussian kernel TSVM. $\sigma = 3$, $C_{rate} = 500$.

We choose a set of 1000 points and vary how many are labeled at each step, measuring the objective function at each level. The TSVM is trying to find labels for the unlabeled points in order to maximize this objective function. The results are given in Figure 3-16. The accuracy and objective function are different measurements, so they can't be compared to each other, but what is important is the general trend for the objective function as the portion of labeled points increases. For the most part, this curve is increasing, which implies that the TSVM is getting stuck in a local maximum for the objective function. Each iteration, two labels are switched to raise the objective function, but this may lead to the labeling that gives the global maximum. These local maxima solutions give reasonably good results, but are not as good as when all points are labeled. A modification of the algorithm that switches more than one pair of labels at a time could improve the performance, but would also increase the complexity.

Unfortunately, this is not the full explanation because of what appears within the first 100 labeled points. The objective function starts out around 88, then increases to 90

with 50 points labeled, which is slightly above the value of the objective function when all points are labeled. When there is a label assignment that gives a higher objective measure than with fully labeled data, the objective function may not be an appropriate measure. Some points in the data set appear different from the majority of points within their class and may lie on the wrong side of the margin. We confirm the existence of these outliers by training a classifier with a large fully labeled set, and noting that there is sometimes up to 1% error on the training points. For outliers such as these, the TSVM can obtain a higher objective function by assigning the label that the samples appear to be, even though this is the wrong classification. With most classifiers, it is good to use samples in training that are close to the margin, but here using these as unlabeled points has an undesirable effect.

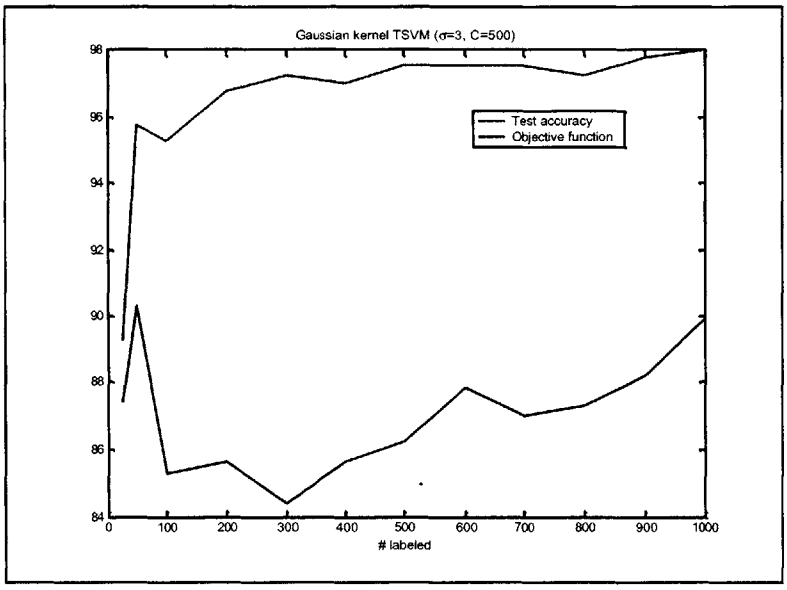


Figure 3-16: Plot of the test performance and objective function while varying the number of points that are labeled. The test accuracy displayed is the percentage of the 400 test points correctly labeled.

3.4.3 TSVM Summary

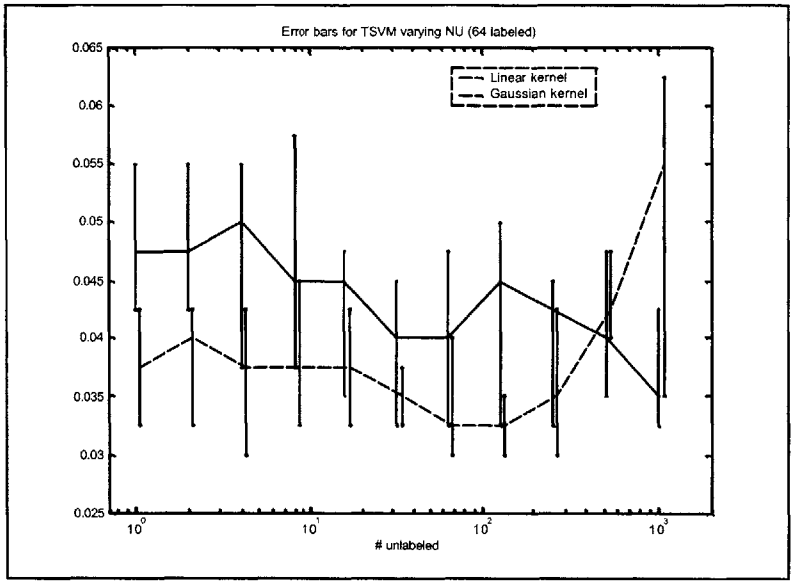


Figure 3-17: Percent error for linear and Gaussian kernel TSVMs with 64 labeled points, varying the number of unlabeled points. The linear kernel starts out with higher error rates but drops steadily, while the Gaussian kernel starts low, drops slightly, then rises at 256 unlabeled points and above. The figures show the mean values with error bars extending to the 1st and 3rd quartile points. Outliers are not shown.

Figure 3-17 directly compares the error rates for the different kernels with 64 labeled points and varying the amount of unlabeled data. For smaller values of NU, the Gaussian kernel is better, while for larger NU, the linear kernel has much better accuracy. Interestingly, the average performance using the linear kernel with 1024 unlabeled points is as good as the Gaussian kernel with no unlabeled data. However, we found that the linear kernel continues to improve with more unlabeled data, so with enough data it will outperform the Gaussian kernel.

For any amount of unlabeled data, a TSVM classifier generally benefits from adding labeled samples. Performance tends to be very poor when fewer than 8 labeled samples are used, but improves beyond this without appearing to saturate.

A linear TSVM does very well as a large amount of unlabeled data is added. Its performance is continually improving, and with enough labeled data, the classification error is the same as if a much larger training set were used. While a Gaussian kernel TSVM benefits from some unlabeled data, it does not do well on this data set with too many unlabeled samples because of the problems with the objective function described earlier. Even with this effect, the error rates on the Gaussian TSVM tend to be very low.

Figure 3-18 shows the eight test points from each class that are most frequently mislabeled by a TSVM. Several of the non-cars that were often labeled as cars have edges where a car typically would, which could easily trick a TSVM if enough of these edges are present.

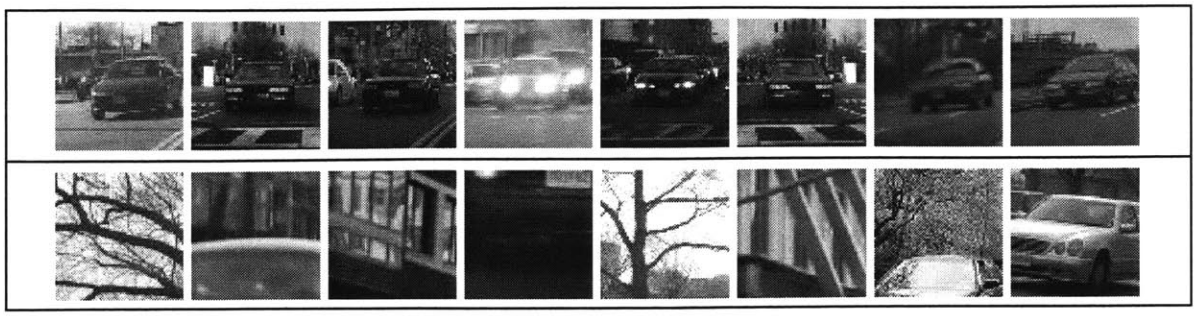


Figure 3-18: The eight images in each class that are most difficult for a TSVM to classify.

3.5 Kernel Expansion

As mentioned in the previous chapter, there are a number of ways to estimate the probability of the label for each point with kernel expansion. We run tests and describe the results separately for each estimation method. There are also different ways of setting the σ parameter, which shows to have a very important effect.

3.5.1 Average margin results

Using the average margin for estimating the $P(y|i)$ parameters offers a closed form formula, which gives it a fast training time. Its only parameter is σ, which can be set in different ways. The first way is to specify one value of σ for all points in space. The other two methods, BMP and KNN, are adaptive and choose a unique value for each point based on the density of points within the region and a multiplier. We give results for each of these ways of choosing σ for this parameter selection method and all subsequent ones.

Constant σ

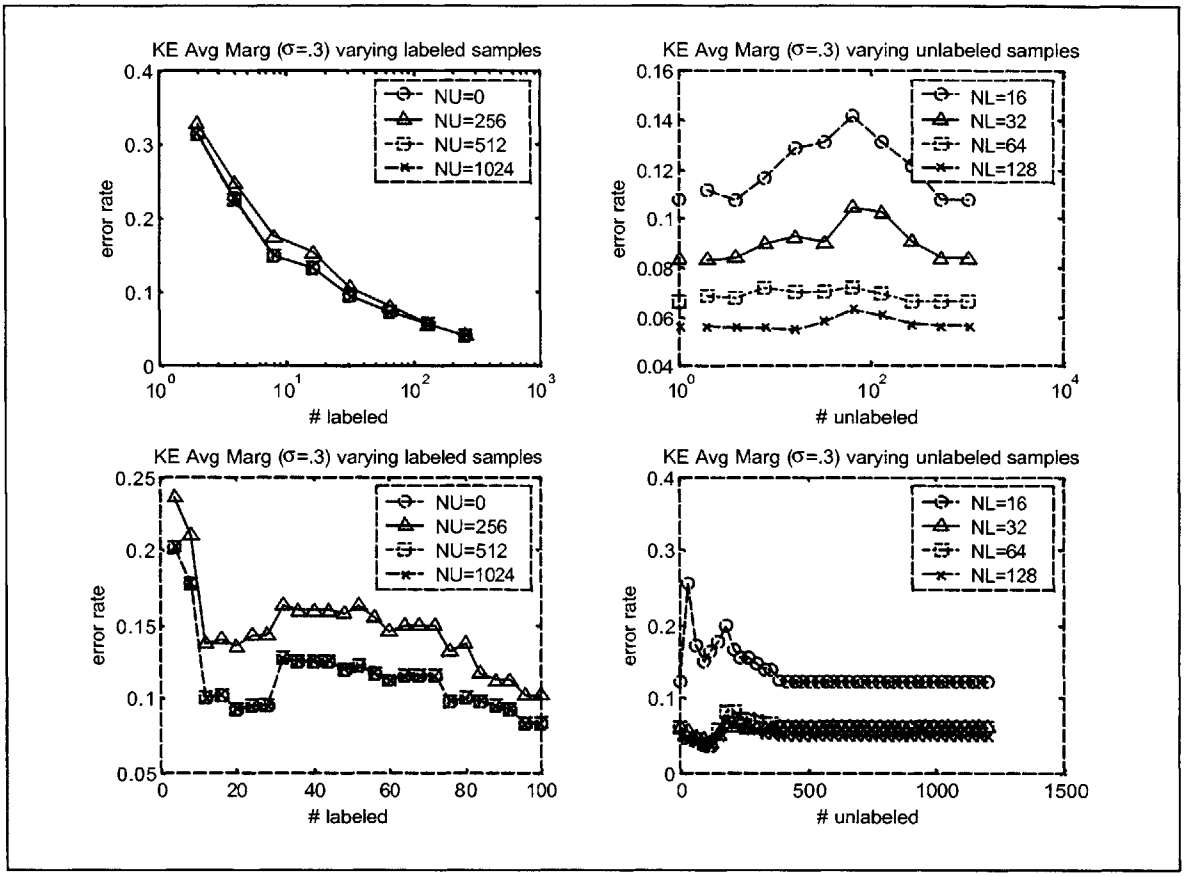


Figure 3-19: Error rates for tests using kernel expansion average margin. $\sigma = 0.3$.

When using a constant value of σ for all samples, we find a good value to be 0.3. This gives the best results over several trials of 64 labeled samples and 1024 unlabeled samples. Results are plotted in Figure 3-19.

Adding more labeled samples steadily improves the classification accuracy. All curves varying labeled samples slope downward, and haven't even fully leveled off after 256 samples are added. This implies that adding more labeled data could lead to further improvement.

Unfortunately, very poor results are found from adding unlabeled data. First, it can be seen that running kernel expansion with no unlabeled points does as well as running with 512 or 1024 unlabeled points, and does better than 256. In fact, having fewer than about 512 unlabeled samples seems to do worse than none, and beyond this, the performance steadies off with very little change. When fewer samples are labeled, this is particularly true.

This is what is observed with some other methods, where having only a small amount of unlabeled data does not help the algorithm very much. The poor performance here suggests that using an adaptive σ may be more appropriate.

Adaptive σ (BMP)

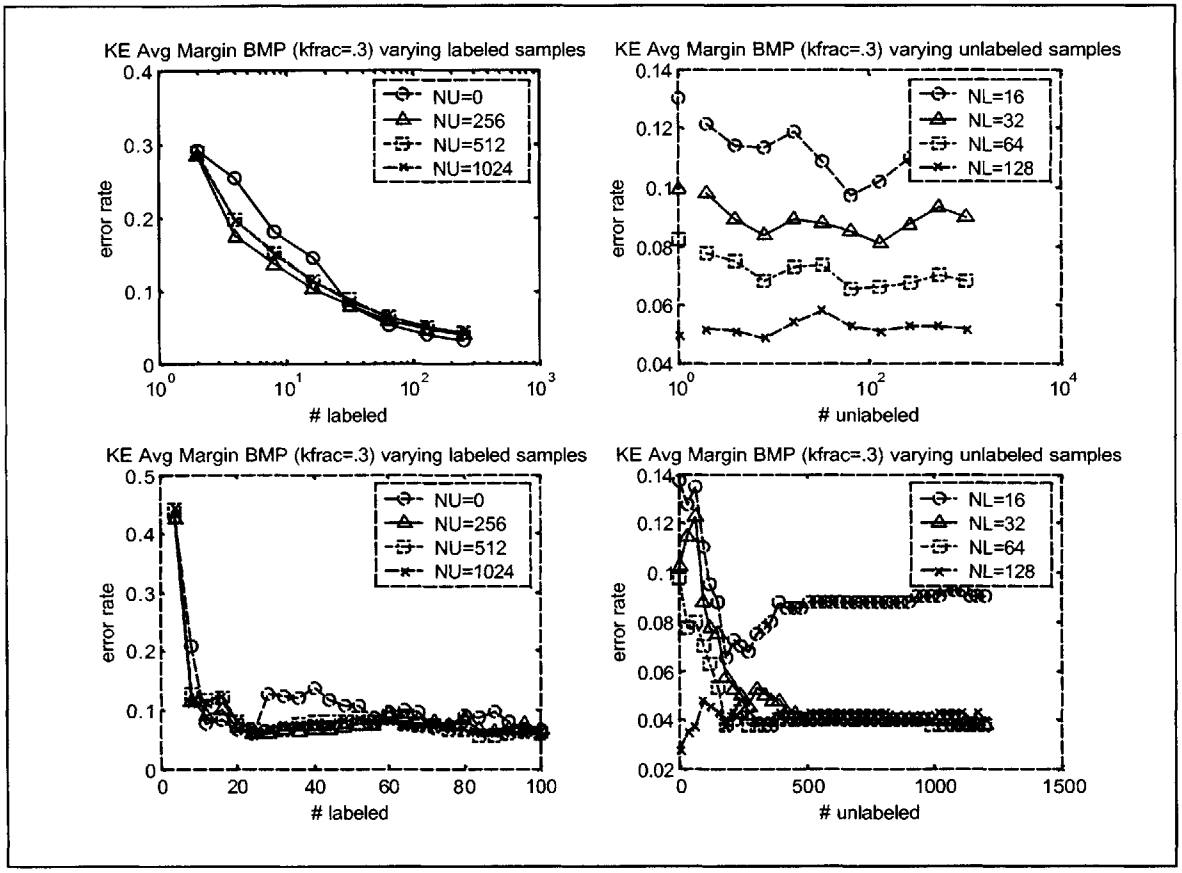


Figure 3-20: Error rates for tests using kernel expansion average margin (BMP adaptive σ). *kfrac* = 0.3. *sigmamult* = 0.2.

The BMP adaptive σ method requires two parameters. The first, *kfrac*, specifies which neighbor's distance to use. The second parameter, *sigmamult*, is a multiplying factor. The value of σ for any point when there are N total points is *sigmamult* times the distance to the (*kfrac* * N)th nearest neighbor. We find the optimal settings to be 0.3 for *kfrac*, and 0.2 for *sigmamult*. For these parameter settings, the results are shown in Figure 3-20. Adding more labeled samples gives a fairly steady improvement. The curves for an individual trial varying labeled show that there is a quick period of improvement from the first few labeled points, and then afterwards the error rate remains

fairly steady. Much like with TSVMs, performance tends to be poor with only 2 labeled samples, often with nearly everything being labeled as one class, yielding 50% error.

As with the constant setting of σ for this method, there does not appear to be much benefit from adding many unlabeled samples. With 128 labeled points, the algorithm tends to perform its best without any unlabeled data. Without as many labeled points, there is some benefit to having unlabeled data, but not as expected. The performance tends to peak between 8 and 128 unlabeled points, and levels off or gets worse beyond this.

Adaptive σ (KNN)

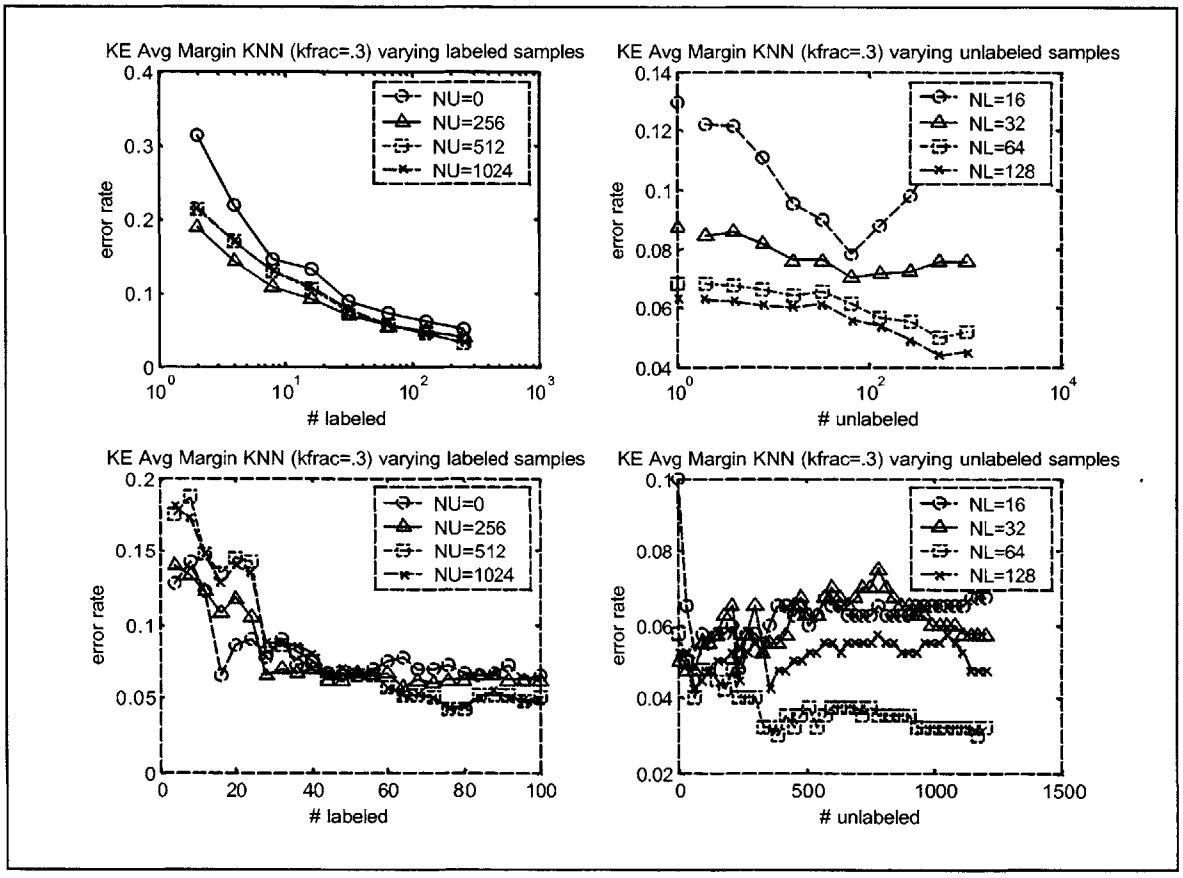


Figure 3-21: Error rates for tests using kernel expansion average margin (KNN adaptive σ). *kfrac* =0.3. *sigmamult* = 0.2.

We find the optimal parameter settings as *kfrac* = 0.3, *sigmamult* =0.2, with results shown in Figure 3-21. These results appear slightly better than the other sigma estimation methods.

Error drops for all curves when adding more labeled samples. The initial error even starts out fairly low compared to some other methods. With at least 256 unlabeled samples, the average initial error is around 20% with only 2 labeled samples. This is certainly not a good classifier in general, but there were some trial runs which achieved as little as 8% error with only 2 labeled samples, which is impressive for a small labeled set. Looking at individual trials, the error drops fairly quickly until about 20 to 40 labeled samples are added, then steadies off with some oscillation as more samples are added.

Having unlabeled data always seems better than having none, but what is interesting is that with fewer labeled samples, too much unlabeled data can hurt. This can be seen by observing that the error rates in the NU = 256 curve lie below the NU = 512 and NU = 1024 curves when NL is 32 or smaller. The curves corresponding to 16 and 32 labeled points further show that average error drops up until 64 unlabeled samples are added, but beyond this the error increases with more unlabeled data. With 64 or 128 labeled points, the opposite effect is observed. There is very little improvement at all up to around 128 unlabeled points, and beyond this the average error drops a few percent while increasing to 1024 samples. This effect is found through most of the samples and is not caused by a small number of outliers.

The parameters may not be optimal for training sets of all sizes, which may cause some of the increases when adding data. In particular, when the unlabeled data overwhelms the labeled data, a much smaller of σ may be more appropriate. Because we optimize the parameters only once for each algorithm, not for each size data set, this can not be avoided.

3.5.2 Minimum margin results

Constant σ

A good value of σ for minimum margin is 4, which seems fairly large compared to values of σ computed for other methods. Minimum margin is not very tolerant to noise, and the larger value of σ here helps for smoothing. Figure 3-22 plots the results.

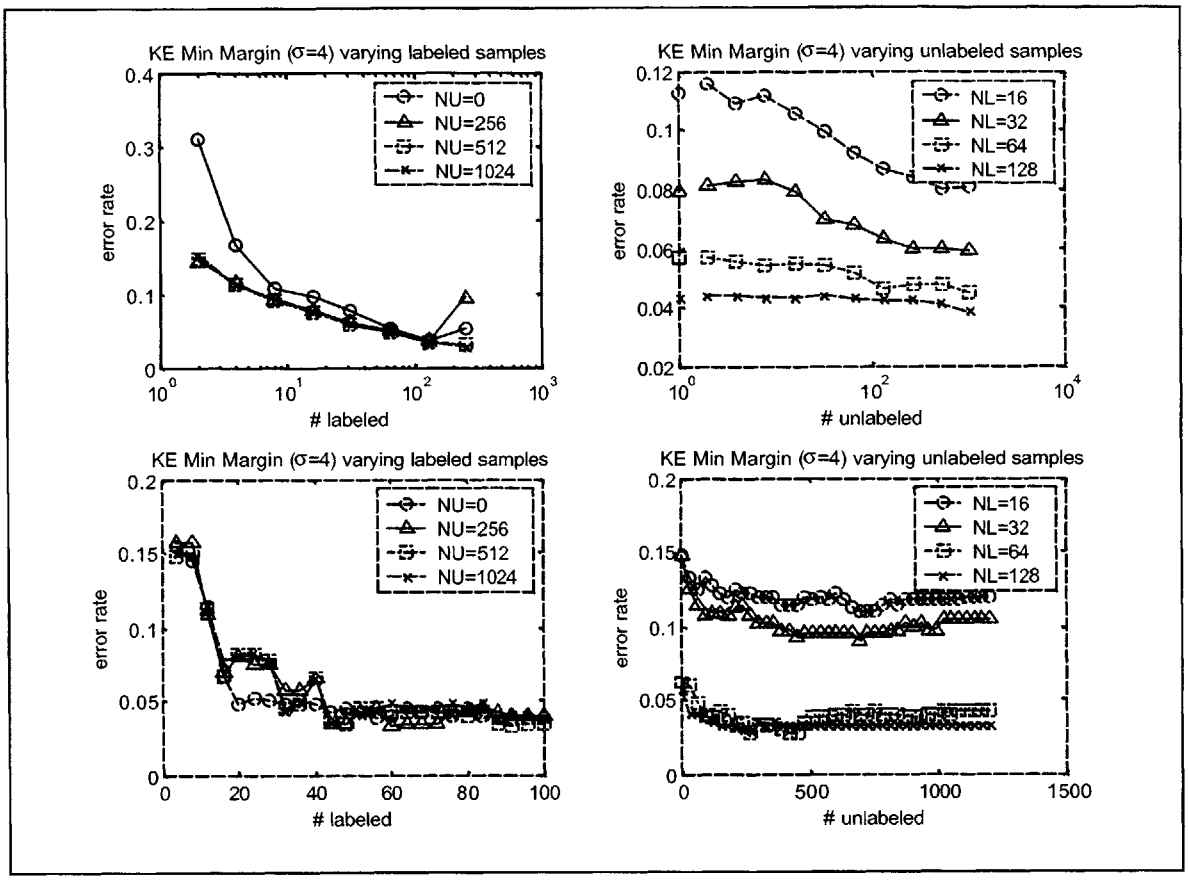


Figure 3-22: Error rates for tests using kernel expansion minimum margin. σ = 4.

For the most part, adding labeled samples helps classification, except in some cases with large amounts of unlabeled data. For example, with 0 or 256 unlabeled points, going from 128 to 256 samples causes the average error to rise. In Figure 3-23 we look at further tests with up to 500 labeled samples, fixed at 1024 unlabeled samples. In the figure at the right, which represents averages over all trials, there are several points where the error goes up as NL is increased, which is not what we expect. This is because there are a few outliers which are affecting the averages. We remove the outliers (cases where all points are classified as positive) and show the averages in the right part of Figure 3-23. This tells us that the error levels can be fairly low, although there is some chance that any trial will give around 50% error.

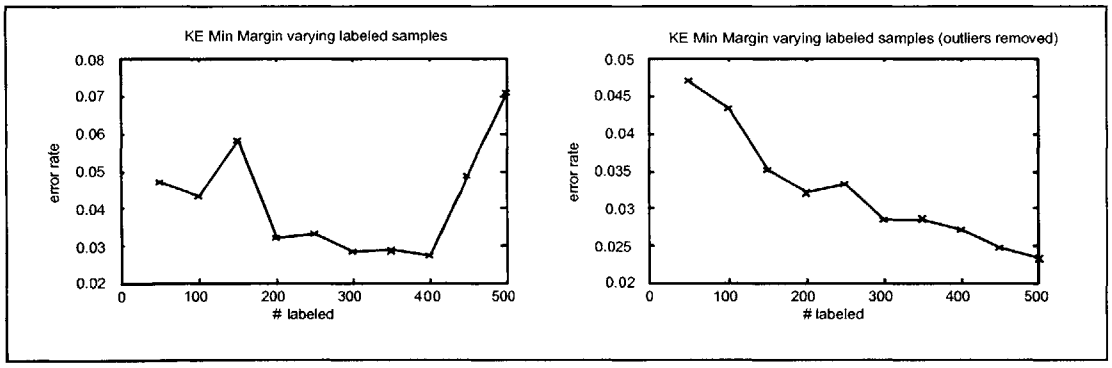


Figure 3-23: Error rates for tests using kernel expansion minimum margin, varying labeled, fixed at 1024 unlabeled. $\sigma = 4$. In the left figure, all trials are shown, while in the right figure, four outlier points are removed.

Individual trial curves varying labeled show that as the initial samples are added, the error can drop fairly quickly. Beyond this, there is not much change from incrementally adding additional samples.

Adding unlabeled points is fairly helpful for this algorithm. The NL = 16, 32, and 64 curves see a drop in error from the labeled data. With a few hundred labeled points, their performance almost matches that from doubling the amount of labeled data. With 128 labeled points, the performance is very good but doesn't change much as unlabeled points are added.

Looking at curves for individual trials shows some change as the first few hundred unlabeled points are added, then periods where the error remains fairly constant. The curves with fewer labeled points improve more rapidly, as is typically seen.

Adaptive σ (BMP)

The best parameter settings found for this method are *kfrac* = 0.05 and *sigmamult* = 0.15. Figure 3-24 displays the results from these trials. Note that the optimal value found for *kfrac* is smaller than used in other methods, although the constant sigma value used with minimum margin is larger than usual. These settings yield the best performance in the test trials, but they perhaps explain the erratic behavior seen in the charts.

Adding labeled points often leads to higher average error rates. For curves with any unlabeled data, the average error goes up when going beyond 64 labeled points. It is not a good sign for any algorithm to experience a rise in error from adding labeled data.

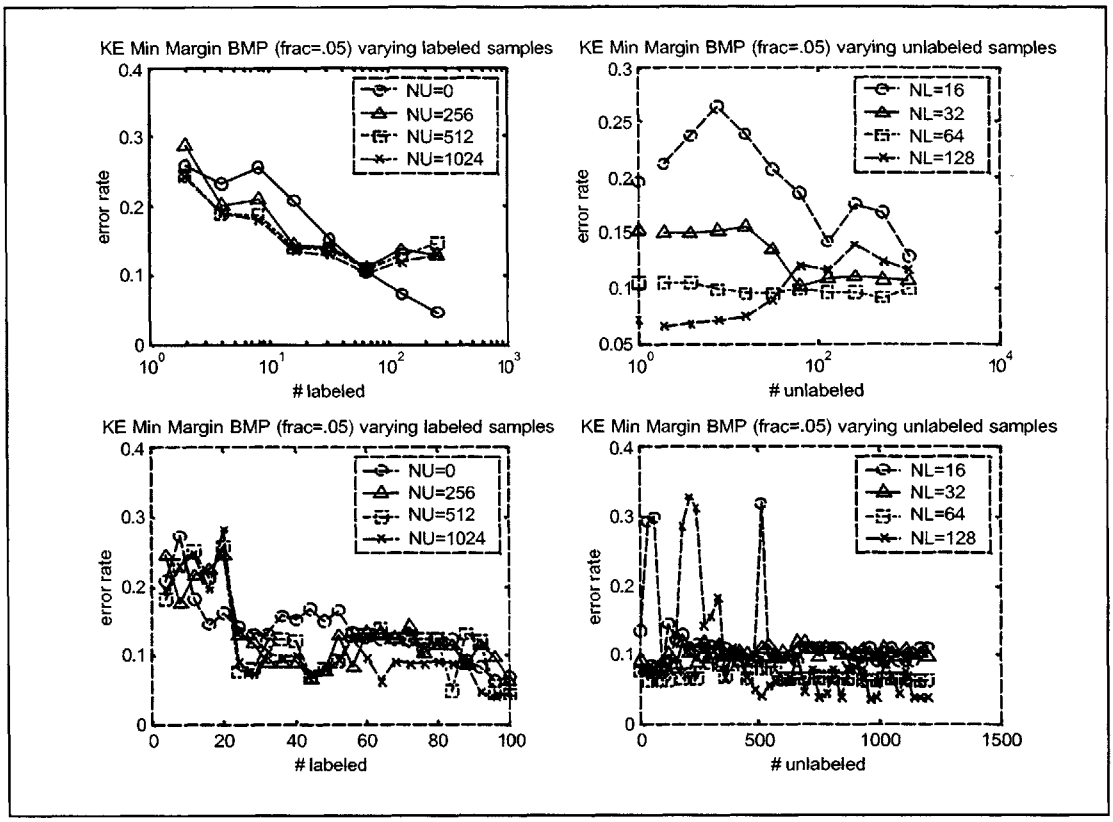


Figure 3-24: Error rates for tests using kernel expansion minimum margin (BMP adaptive σ). *kfrac* = 0.05. *sigmamult* = 0.15.

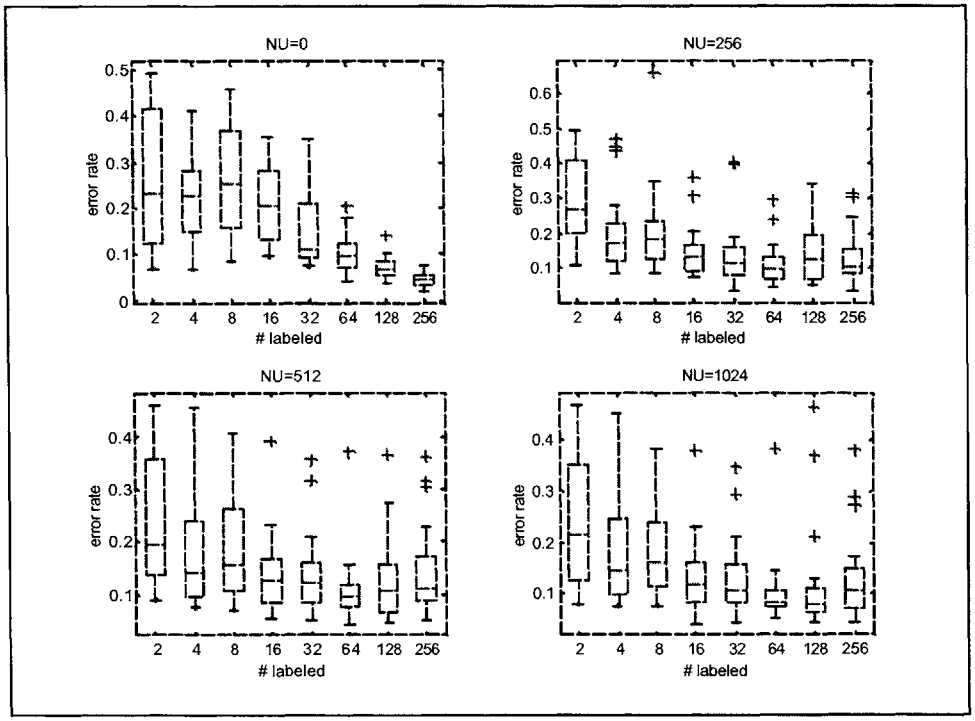


Figure 3-25: Box-plot showing error rates using kernel expansion minimum margin (BMP adaptive σ) varying NL, fixed at 0, 256, 512, and 1024 unlabeled.

However, looking at the box-plots in Figure 3-25 shows that the higher averages are caused by some outliers. The outliers become more common as there is more unlabeled data, which means that the algorithm is less reliable in these situations.

Adding unlabeled data helps slightly when there are only a few labeled points. The curves for 16 and 32 labeled see an overall drop in error, although portions of the graph do cause an increase. The NL = 64 curve has very little dependence on the number of unlabeled points and has a fairly steady average error of about 10%. The individual curve also shows very little change. With 128 points, the average error gets uniformly worse when increasing the unlabeled size.

The results demonstrated here show that this particular parameter selection method is not suitable for our data set. Even though the error rates can occasionally be very low, the higher rates occur too frequently.

Adaptive σ (KNN)

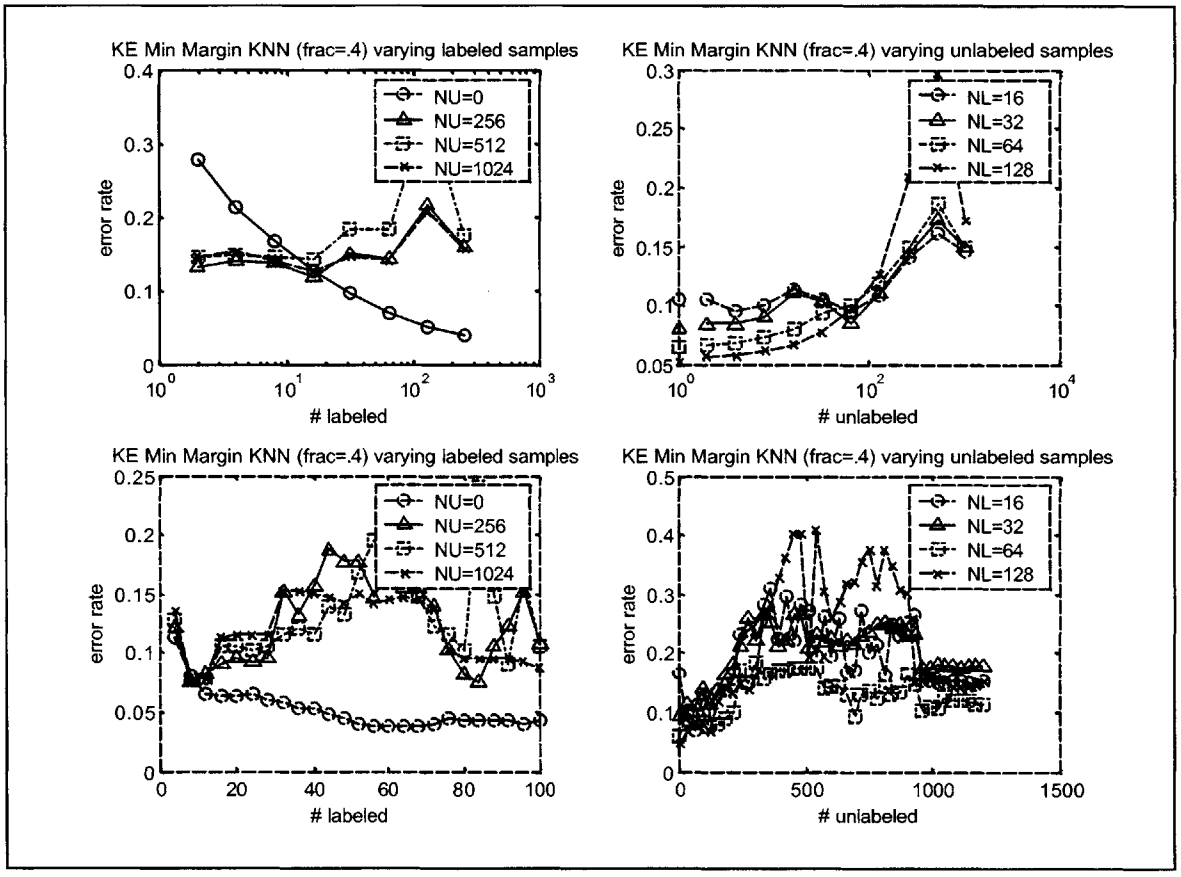


Figure 3-26: Error rates for tests using kernel expansion minimum margin (KNN adaptive σ). *kfrac* = 0.4. *sigmamult* = 0.15.

Figure 3-26 shows the results using the KNN adaptive σ with minimum margin, with *kfrac* = 0.4 and *sigmamult* = 0.15. The results for this method are among the worst we have seen. Adding more labeled samples only helps when there is a very small amount of unlabeled data. With more than 64 unlabeled points, adding labeled data appears to have the opposite effect it should. The labeled data learning curves rise in error as data is added, and unlabeled data learning curves have lower error in curves with smaller values of NL.

Adding unlabeled points also causes performance to degrade. The average errors increase until 512 points are added, then begin to decrease. Tests that go beyond this number of unlabeled points (Figure 3-27) show that after this sharp decrease, the error levels off without much change.

3.5.3 Maximum entropy discrimination (MED) results

Constant σ

A good setting of parameters for maximum entropy discrimination is $\sigma = 0.3$, *margin* = 0.1, $C_{rate} = 1000$, with results using these settings shown in Figure 3-28. This algorithm shows a quick and steady drop in the error level as labeled points are added to the classifier. Even going from 128 to 256 points causes the average error to drop from 5.6% to 4.3% for each of the curves, meaning it hasn't leveled off from labeled data by that point. Despite very low error rates, this method does not seem to improve with unlabeled data in any way. The flatness of the individual trial curves verifies that this is not caused by some samples affecting the average. Outside of the region of interest, with NL = 2 or NL = 4, having many unlabeled points is actually slightly worse.

It is very odd to see results that are almost entirely independent of the number of unlabeled points, so we run some tests with different parameter settings. We only focus on the averages over 20 trials when varying the number of unlabeled points, and show the results in Figure 3-29. These curves do vary as NU changes, but actually get worse with more samples.

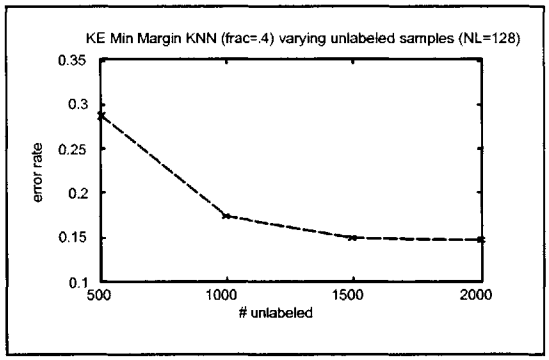


Figure 3-27: Error rates for tests using kernel expansion minimum margin (KNN adaptive σ) varying NU, with NL fixed at 128. *kfrac* = 0.4. *sigmamult* = 0.15.

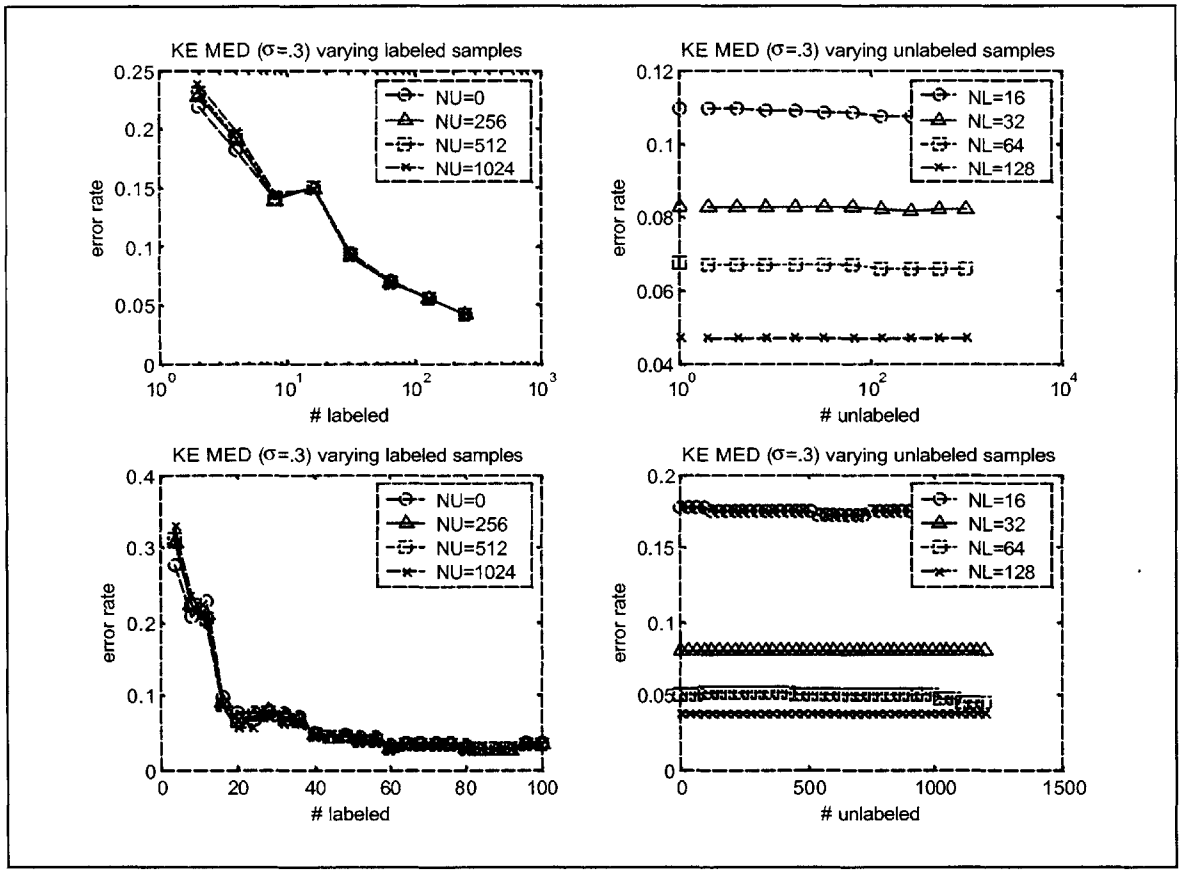


Figure 3-28: Error rates for tests using kernel expansion MED. σ = 0.3, *margin*=0.1, C_{rate} = 1000.

Adaptive σ (BMP)

The results of MED parameter estimation using BMP are shown in Figure 3-30, with parameters set as *kfrac* = 0.3, *sigmamult* = 0.2, C_{rate} = 1000, and *margin* = 0.1. These charts show that an adaptive sigma helps out quite a bit. Without any unlabeled samples, the error is slightly worse than with a constant σ, but there is a big improvement

as unlabeled points are added. Using about 256 unlabeled points with a fixed size labeled set is roughly equivalent in some cases to doubling NL. At 512 unlabeled, there is even more improvement, but beyond that the performance levels off.

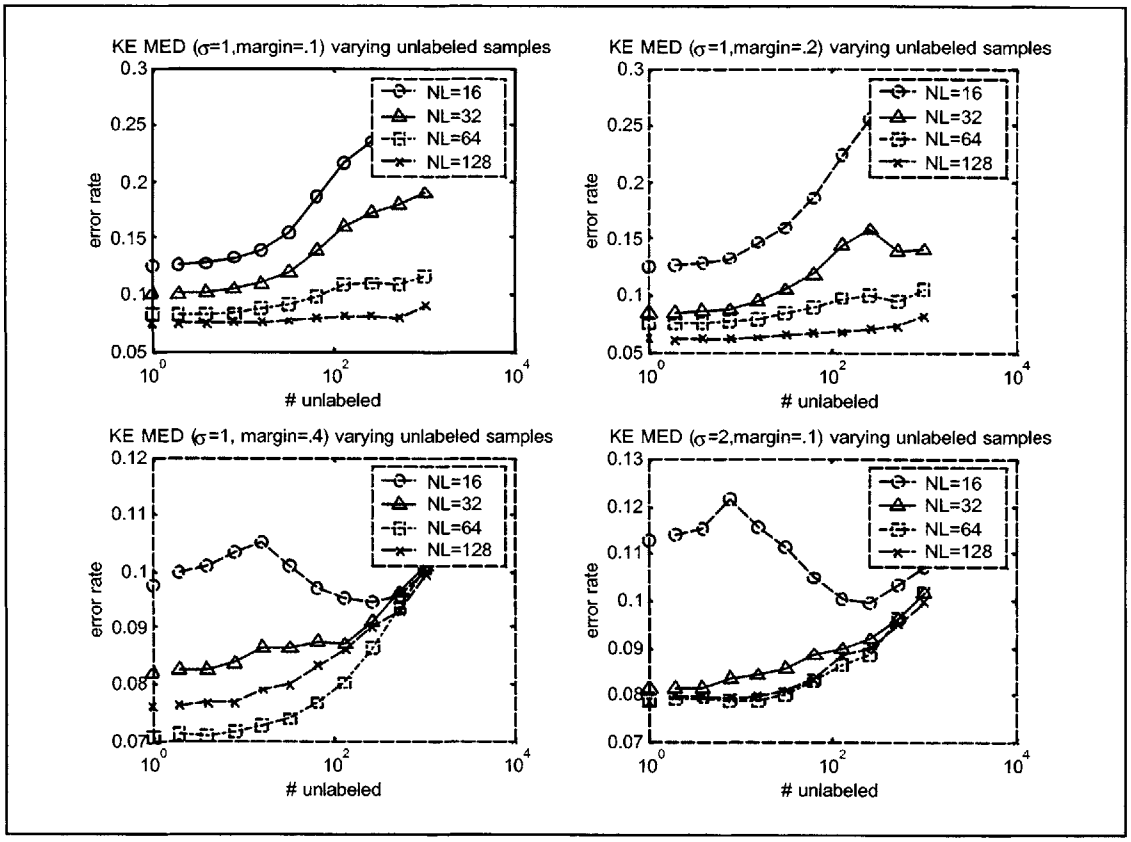


Figure 3-29: Error rates for tests using kernel expansion MED with non-optimal parameter settings. The curves tend to get worse with more unlabeled data and better with more labeled data.

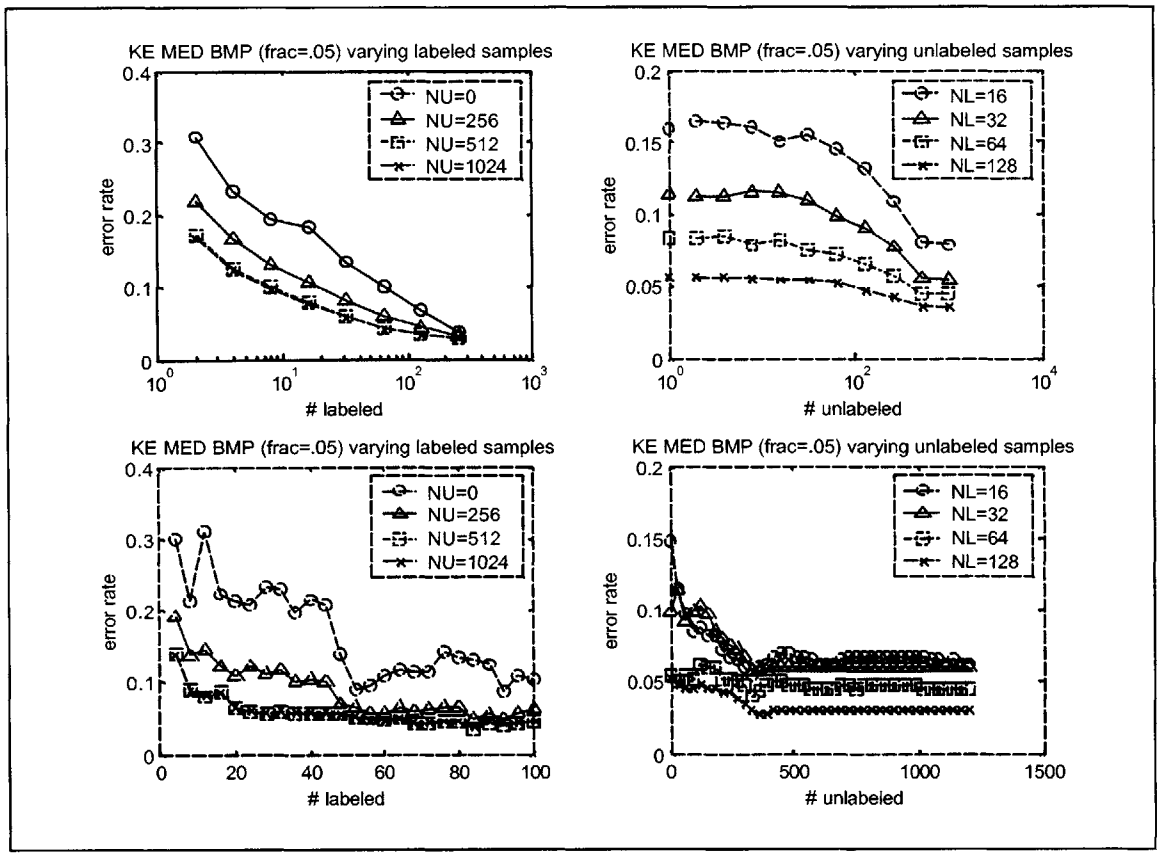


Figure 3-30: Error rates for tests using kernel expansion MED (BMP adaptive σ). *kfrac* = 0.05, *sigmamult* = 0.2, *margin*=0.1, C_{rate} = 1000.

Adaptive σ (KNN)

In Figure 3-31 we show the results using the KNN method, with *kfrac* = 0.05, *sigmamult* = 0.2, *margin*=0.1, and C_{rate} = 1000 as the parameter settings. These charts show the same trends as using BMP, but with lower errors.

The individual curves reveal some important information. The first few labeled points are very important, as they cause a sharp drop in the error. Around the 36th sample, there is a big jump in the error for the NU=0 curve. This is likely due to a single bad sample which causes parts of the test set to be mislabeled. At this point, there is still a small rise in the error for the NU=256 curve, but it is much less extreme, and the NU=512 and NU=1024 curves barely budge. Here, a large unlabeled set has helped dampen the effects of a poorly labeled point. This is a very nice property, as one cannot always expect the labeled data to be perfect.

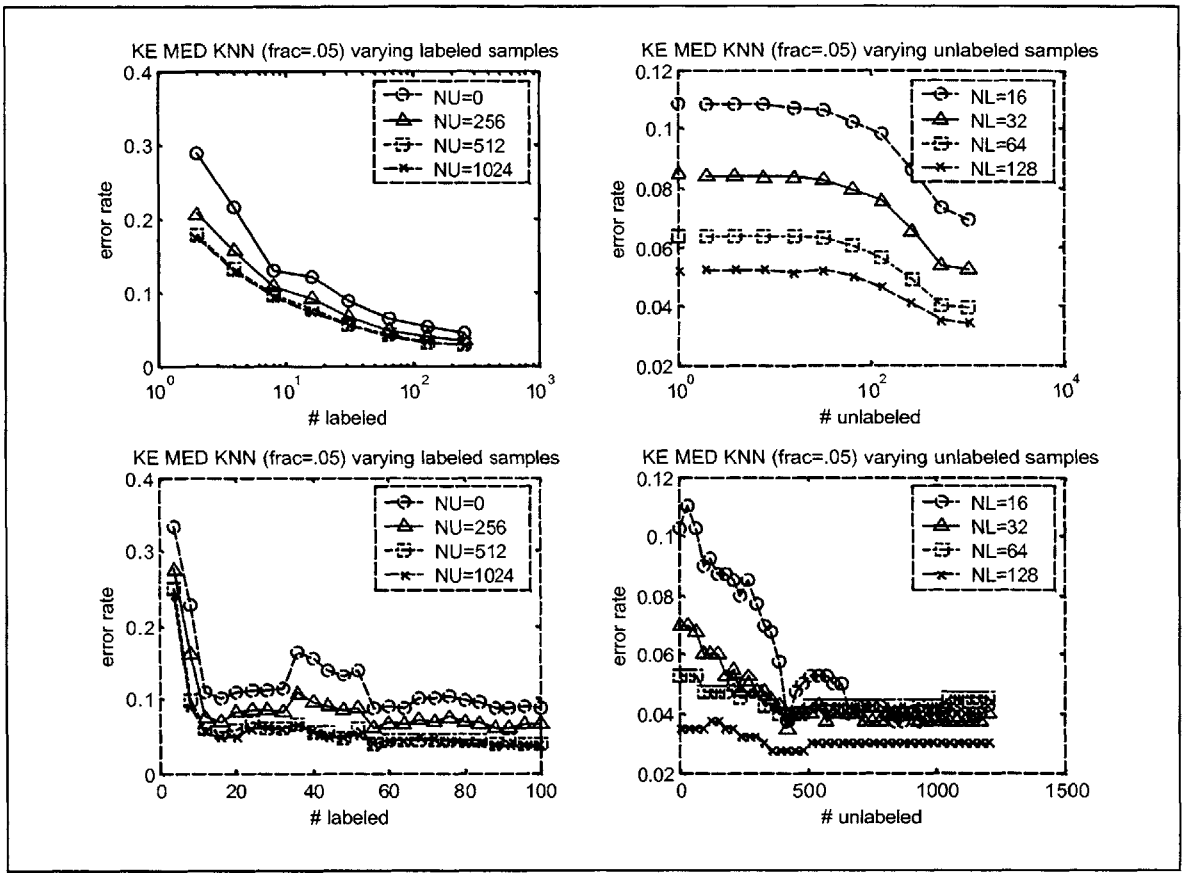


Figure 3-31: Error rates for tests using kernel expansion MED (KNN adaptive σ). *kfrac* = 0.05, *sigmamult* = 0.2, *margin*=0.1, C_{rate} = 1000.

The results for all three methods of computing sigma are compared together in Figure 3-32, with the optimal parameters used for each method. When varying the labeled data, fixed at NU = 1024, all three methods have a downward trend, but KNN is generally has between 1% and 2% lower error than BMP, which is below the constant σ by the same amount. When varying the amount of unlabeled data and training with 32 labeled points, KNN always has the best performance. For small amounts of unlabeled data, BMP does the worst of all three methods, but improves and surpasses the constant σ with about 128 or more unlabeled points.

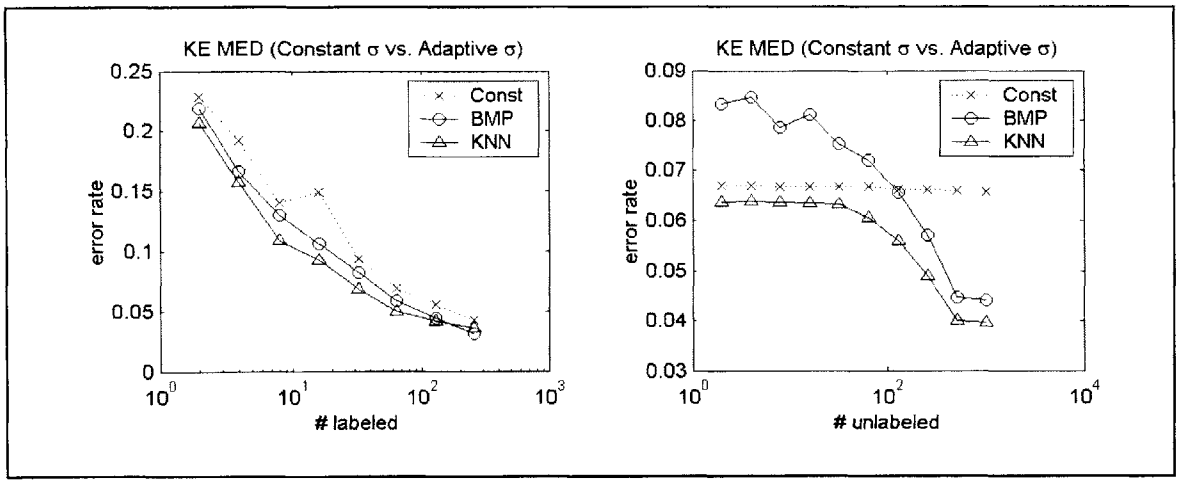


Figure 3-32: Comparison of constant and adaptive σ for kernel expansion MED. In the left chart, NU=1024, and in the right chart, NL=32. Optimal parameters found in the previous section are set for each method.

3.5.4 EM estimation results

Constant σ

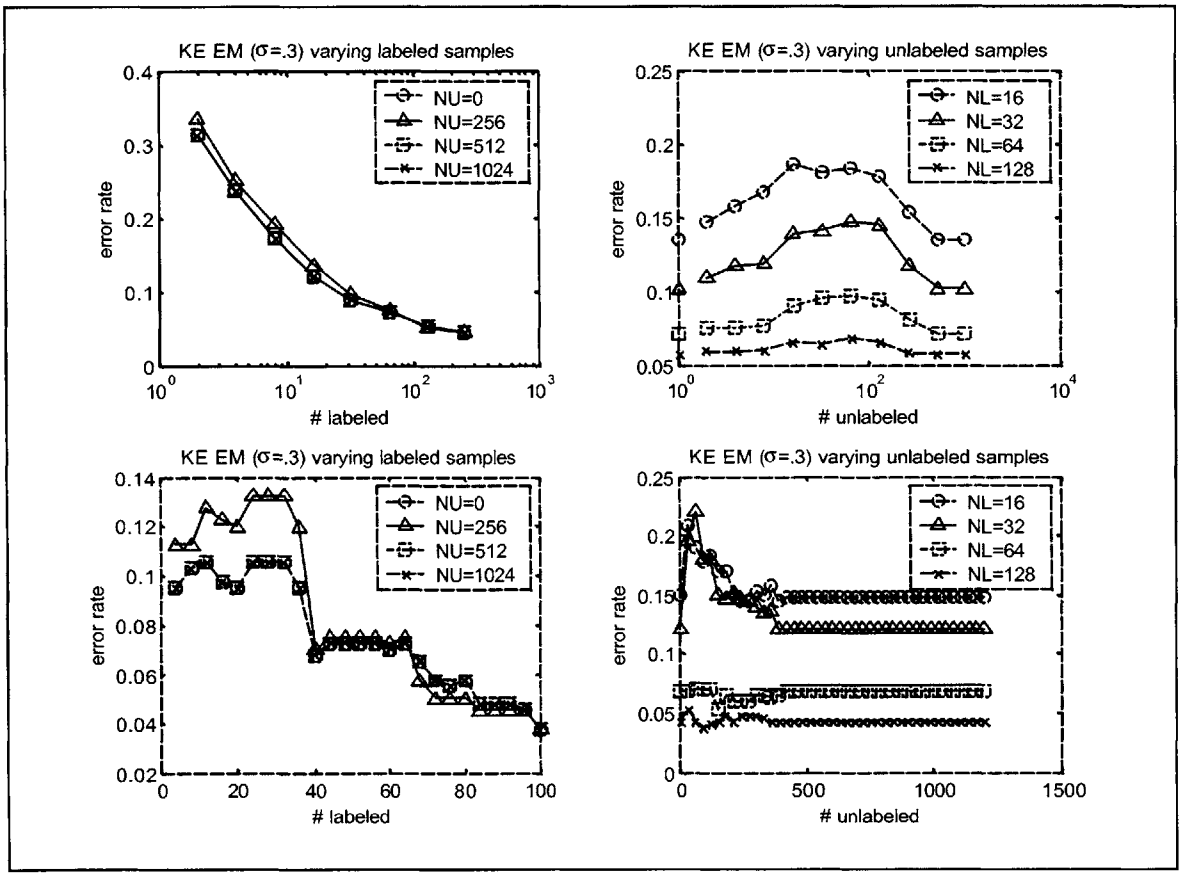


Figure 3-33: Error rates for tests using kernel expansion EM. σ = 0.3.

The final parameter estimation method we explore for kernel expansion is EM. For a constant value of σ, we find 0.3 as a good value. The results from the trials are shown in Figure 3-33.

Adding labeled points is very helpful for this algorithm. The average error drops very steadily and is just beginning to level off around the 256^{th} labeled point. Looking at individual curves shows that very rarely the error increases when adding additional labeled points, but for the most part remains steady or drops. Beyond a certain point, the error tends to level off for long periods and drop occasionally.

Unfortunately, adding unlabeled points does almost no good. In the region of interest, the 0 unlabeled curve performs nearly identically to 512 or 1024, and better than 256 unlabeled. The learning curves for unlabeled data support this. As the first 64 to 128 samples are added, the error rises slightly, particularly for smaller NL. Beyond this, the error drops, but does not get below its initial level. Because of this, this algorithm is not very useful on our data set.

Adaptive σ (BMP)

We try adaptive σ methods with EM parameter estimation, hoping that they will help improve the results. For tests using the BMP method, with *kfrac* = 0.05 and *sigmamult* = 0.2, we show the results in Figure 3-34.

The performance of this algorithm improves slightly as NL increases. The individual curves show that there is a sharp drop in the error, usually in the first 10 to 20 samples, and then a fairly steady error rate beyond that.

Adding unlabeled points has a very important effect when there is not much labeled data. With 16 labeled points, the average error drops significantly, even with just a few extra samples. With 32 labeled points, there is some improvement, but less extreme. With 64 or more labeled, the unlabeled points lead to hardly any improvement. Regardless of the number of labeled points, having large NU seems cause degrade the performance, as the average errors increase slightly beyond the 256^{th} point. The individual curves demonstrate this pattern very clearly. Around NU=300, the curves all start going towards higher error levels, and eventually level off.

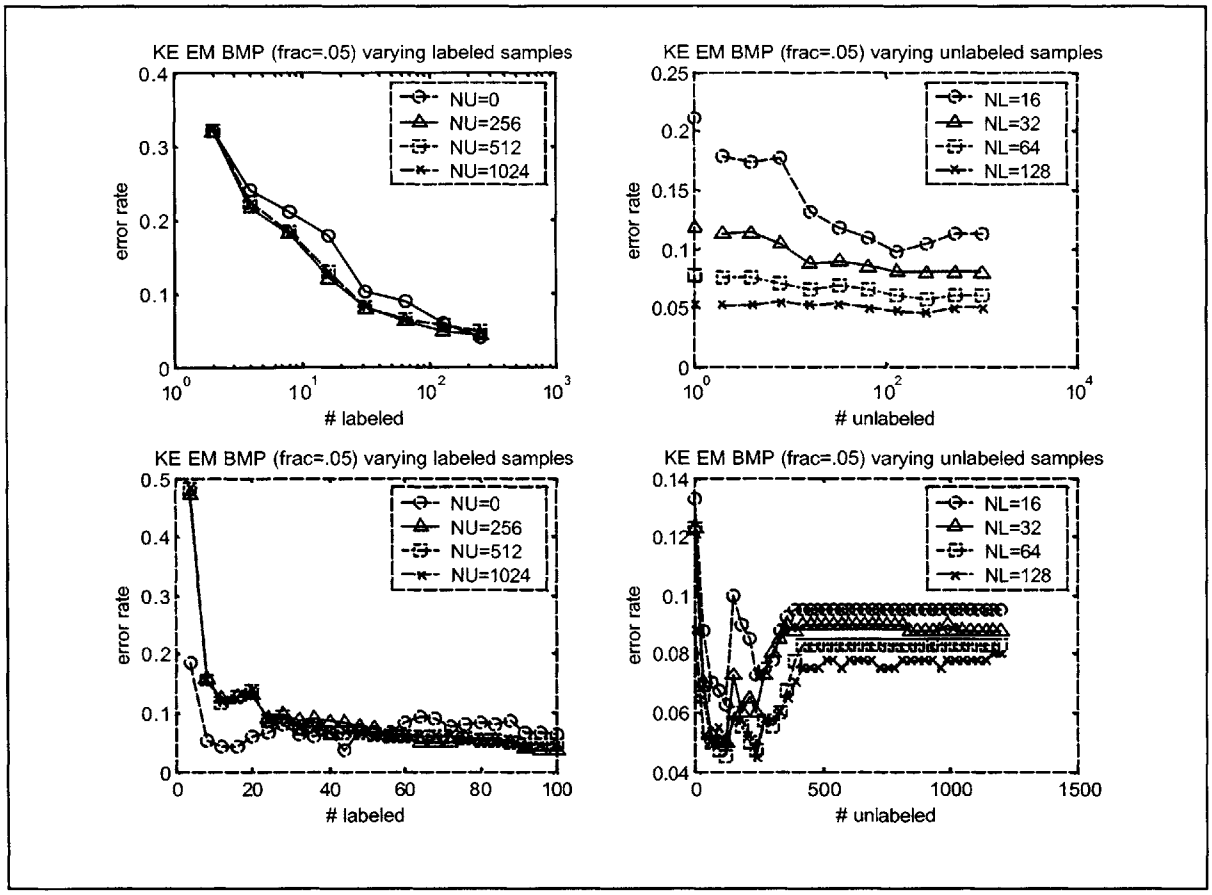


Figure 3-34: Error rates for tests using kernel expansion EM (BMP adaptive σ). *kfrac* = 0.05, *sigmamult* = 0.2.

Adaptive σ (KNN)

We show the results for kernel expansion with EM, using KNN adaptive σ method in Figure 3-35, with *kfrac* = 0.2 and *sigmamult* = 0.2. This estimation method benefits from more labeled data but does not seem to gain much from unlabeled points, except in a few cases. For curves with small NL, a small amount of unlabeled data can improve the classifier. Beyond this, the error starts to go up, well beyond the initial rates at NU=0. If more points are labeled, the classifier is able to handle more unlabeled points before reaching its peak performance. This can be seen more clearly with the individual trial curves, where the performance peaks around the 90th sample for the NL=16 curve, and closer to the 360th sample for the curves with higher NL.

This pattern is not good for the intended purpose of these algorithms, because the desired use is with a few labeled samples and a large amount of unlabeled points. Even though other methods have a U-shaped curve like this when varying NU, this one is much

worse. This is because of how quickly the peak performance is reached, and the fact that beyond this the performance is much worse than if no unlabeled data is used.

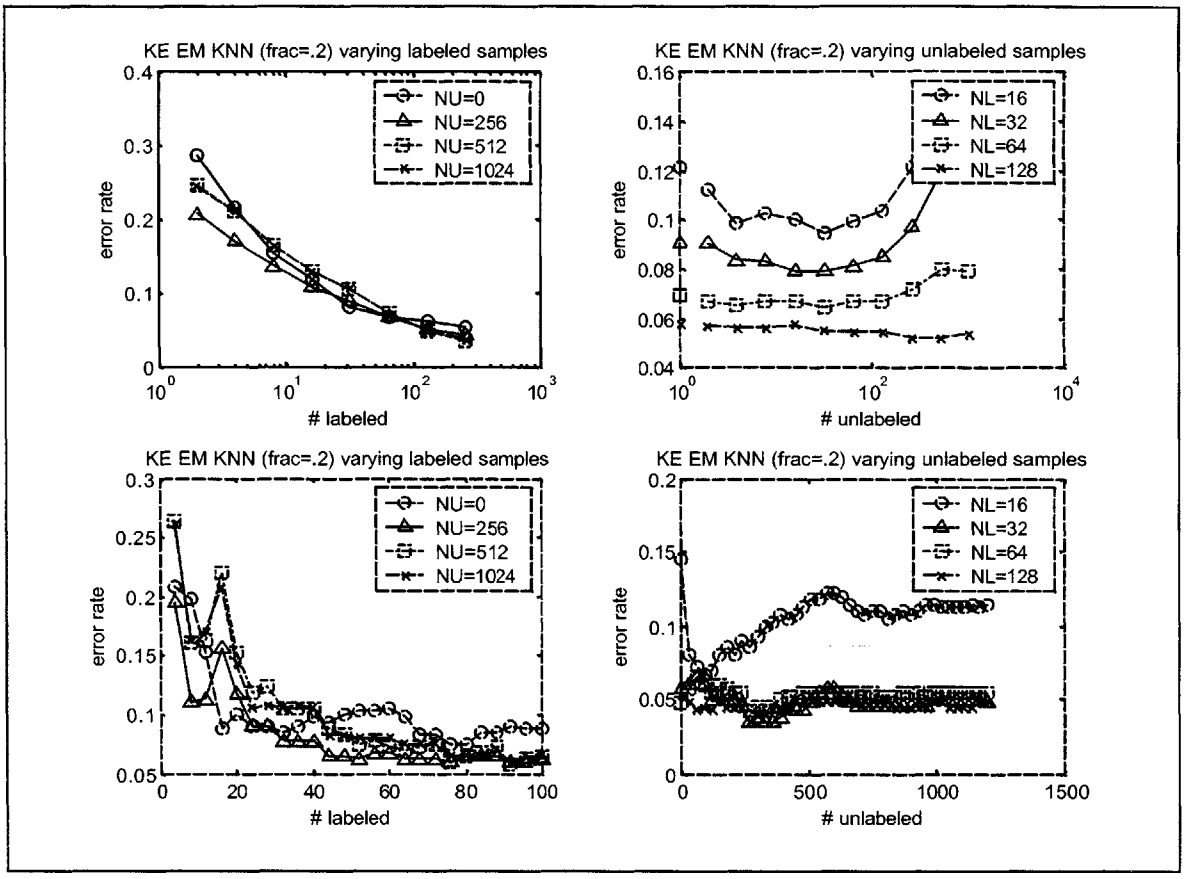


Figure 3-35: Error rates for tests using kernel expansion EM (KNN adaptive σ). *kfrac* = 0.2, *sigmamult* = 0.2.

3.5.5 Kernel expansion summary

Of the several methods we explore with kernel expansion, we find that some work reasonably well, while others work very poorly. Of the three methods of computing σ, none is superior to others in all cases, but instead it depends on which parameter estimation method it is used alongside.

For average margin parameter estimation, only KNN works well, and only when it has enough labeled data. Minimum margin yields reasonable results with a constant σ or BMP, with the exception of occasional outliers. Without these outliers the average error rates are good, but the fact that an occasional trial can have such high error means that the method should be used with caution. Using KNN or BMP with MED parameter

estimation gives low errors that improve further with the addition of labeled and unlabeled data. Finally, EM with BMP σ estimation works well, although the performance doesn't seem to improve once a certain amount of data is supplied. The remainder of the methods give high error rates or get worse with larger data sets, so they should not be used in practice for a data set similar to ours.

3.6 Markov Random Walk

The concept of connected components is important when dealing with Markov random walk. This value tells us how many disjoint units the data is split into. If there is only one connected component, then any point can reach any other point along some path, by single step transitions to one of the point's k neighbors. If there is more than one component, then some points can not reach others along any path, regardless of how many time steps there are. This will typically occur when there are several data clusters that are isolated from each other, as in Figure 3-36. The number of components depends on k, and as is non-increasing as k gets larger.

For Markov random walk to work as it should, the data will be best if there is only one connected component for our choice of k. Otherwise the labels on unknown points will only be influenced by part of the data set, not all of it. We find that for our data set, as long as k is 4 or larger, there is one connected component, and thus we will only run tests with this algorithm when k is above 4.

As with kernel expansion, we run Markov random walk tests with each of the parameter estimation methods and analyze the results individually.

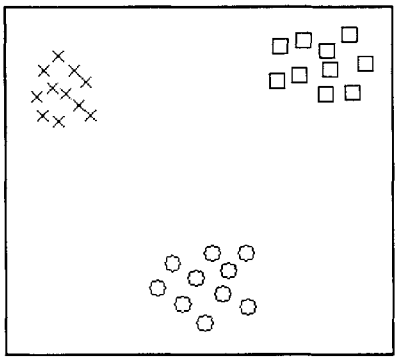


Figure 3-36: When there are isolated clusters of points, there will typically be more than one connected component. This chart has three connected components, each represented by a different symbol, for values of k less than about 10.

3.6.1 Average margin results

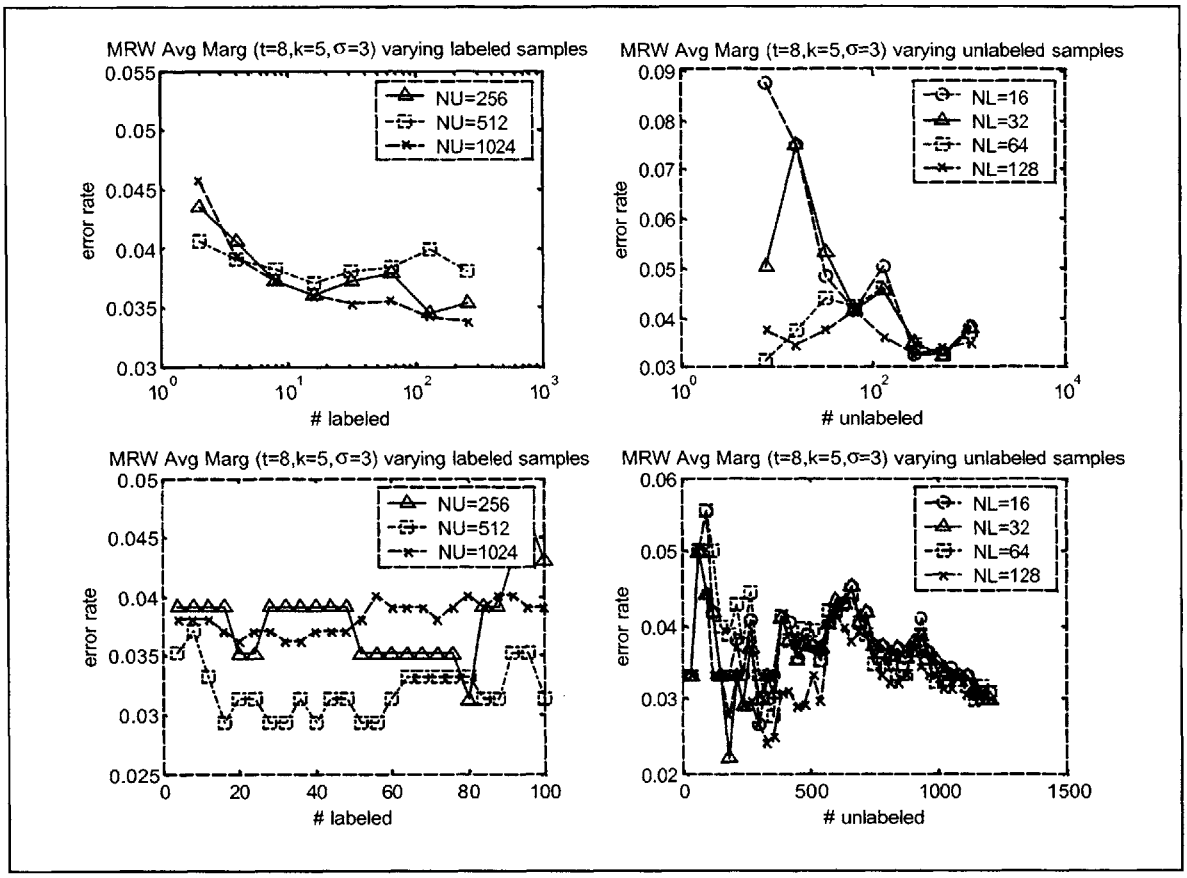


Figure 3-37: Error rates for tests using Markov random walk average margin. $t = 8$, $k = 5$, $\sigma = 3$.

The results for average margin estimation of the parameters are shown in Figure 3-37, with t=8, k=6, and σ=3. Looking at the charts shows how erratic this method is, both from adding labeled and unlabeled samples. The average errors oscillate, so it will be useful to look at errors on individual trials. Figure 3-38 looks at the box plots for some of the curves.

Looking at results for when NU=512, it seems that adding labeled samples tends to help in the worst case. The highest error for any trial is much lower for 8 or more labeled samples than it is for 2 or 4. However, beyond this, there is not a significant amount of change from adding more labeled data. The error tends to fall between about 3% and 4.5% without any appearance of converging. This implies that the variation in error is mostly due to sampling error. These averages are based on 20 trials, and with a much larger sample the results will smooth out more.

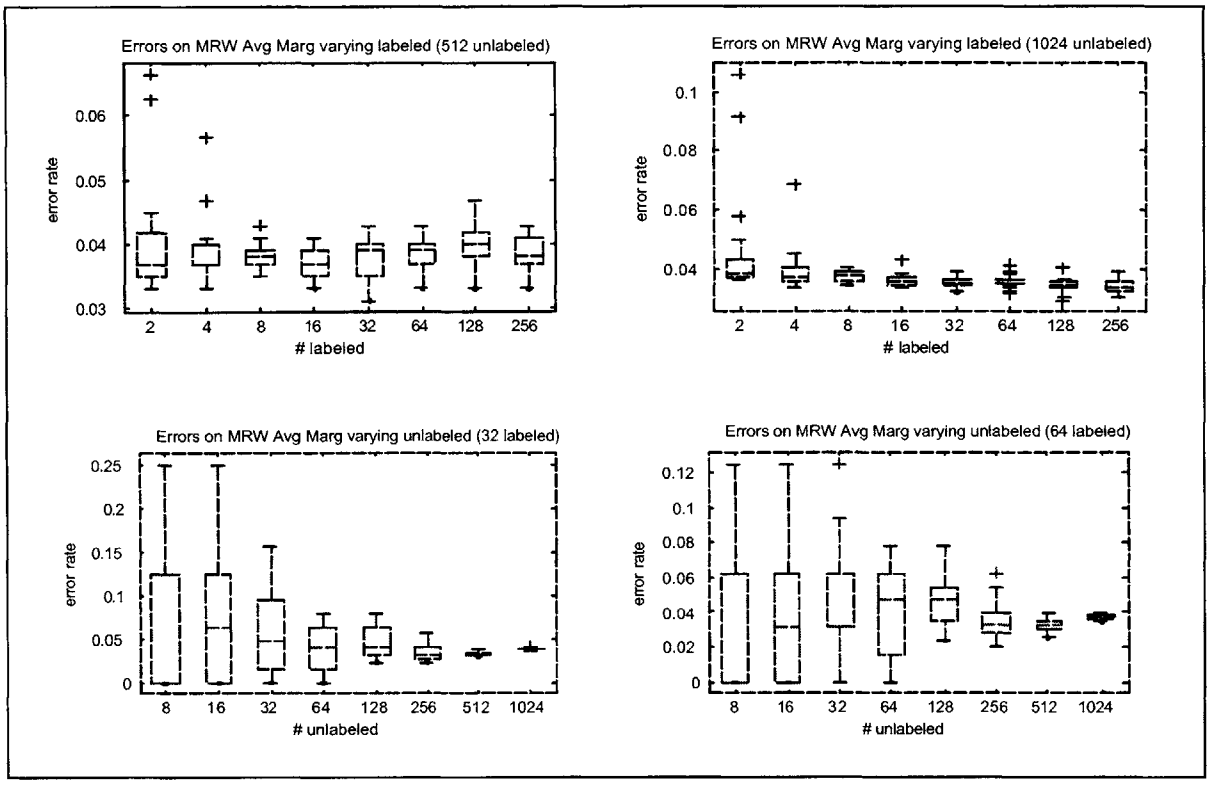


Figure 3-38: Box-plot showing error rates using Markov random walk average margin varying NL, fixed at 512 and 1024 unlabeled (top) and varying NU, fixed at 32 or 64 labeled (Bottom).

The average results varying unlabeled data seems to oscillate as well. From the box-plots here, more unlabeled samples reduce the largest error found in any trial, but also increases the lowest error rates. In some cases with fewer unlabeled samples, the error rate is 0, but this never happens in trials with more than 64 unlabeled points. This is just due to random chance, and does not imply that the performance on smaller unlabeled sets is better. As mentioned earlier, the Markov random walk algorithm does not allow testing on non-training points. For smaller amounts of unlabeled data, the test set is smaller. Consider a set of tests run on a random set of data where there is a 3% chance of error for any trial, regardless of the amount of labeled points. The same basic pattern emerges here as in the bottom of Figure 3-38. As the test set increases, the extreme values converge on the average error rate, which is 3% in this case. This tells us that we should not be concerned that some trials occasionally have lower error rates than trials with larger training sets. What is more important is the average error, which will converge given enough trials.

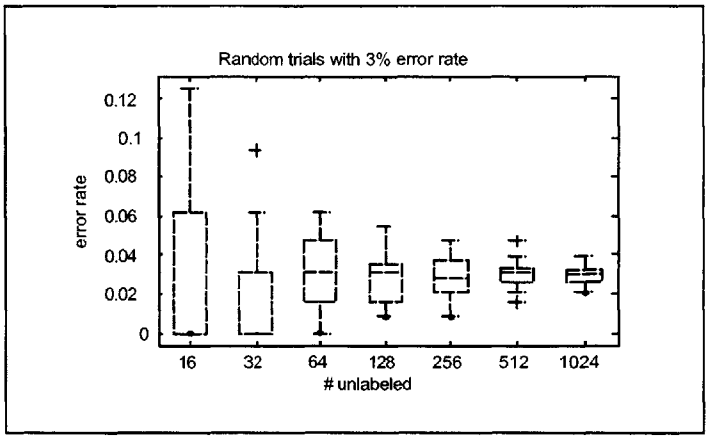


Figure 3-39: Box-plot for randomly generated trials, where the test set is equal to the number of unlabeled points. Any individual trial has a 3% chance of being an error.

3.6.2 Minimum margin results

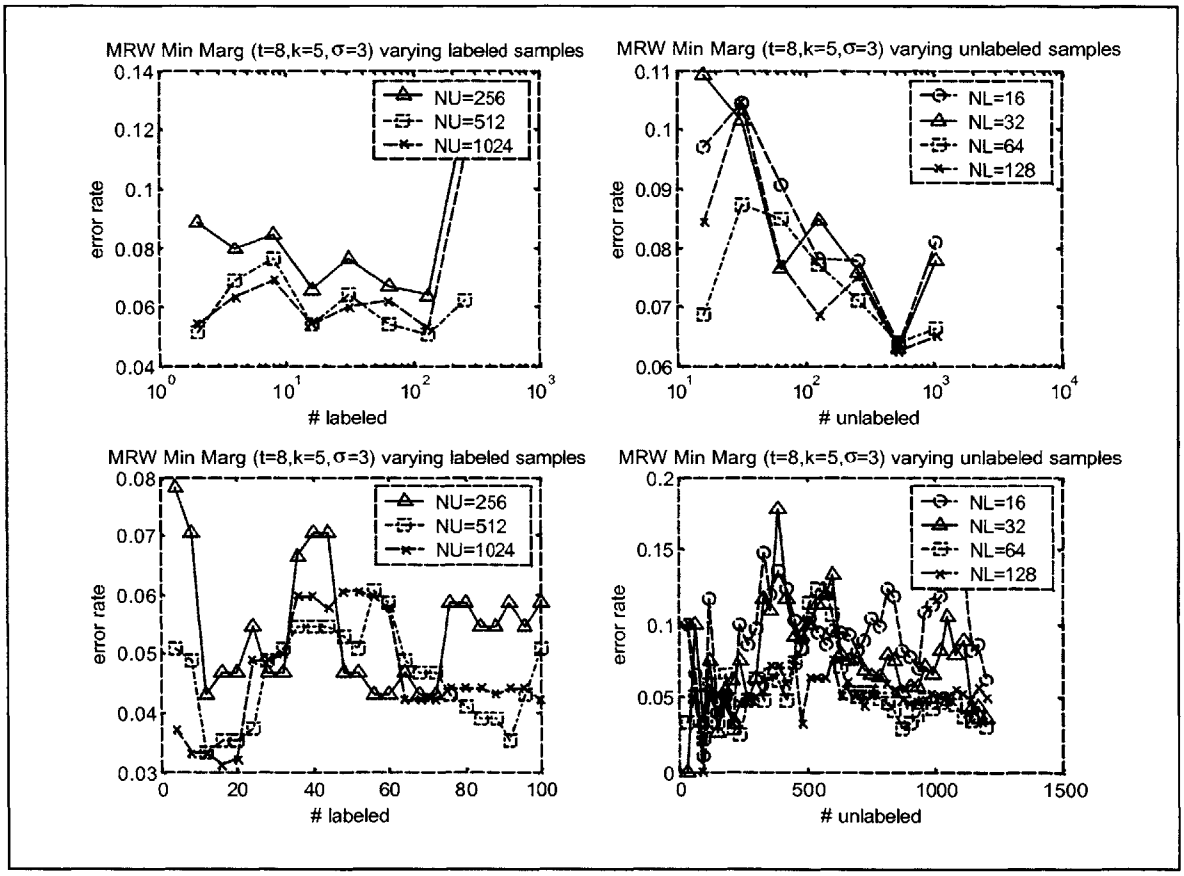


Figure 3-40: Error rates for tests using Markov random walk minimum margin. $t = 8$, $k = 5$, $\sigma = 3$.

We plot the results in Figure 3-40 using minimum margin with Markov random walk, with parameter settings $t = 8$, $k = 5$, and $\sigma = 3$. The individual trials show a lot of oscillation, as well as the average curves. The average curves demonstrate that the error

does not always drop as more labeled points are added, and in particular, 256 labeled points has very poor results. This is due to noise, which minimum margin does not handle well, but it makes the algorithm very unreliable.

Even though labeled samples do not always help, this algorithm benefits from unlabeled data. Each of the curves varying unlabeled samples has a general downward trend, although only 16 unlabeled points tends to do well.

Overall, this algorithm does not seem very useful because of its unpredictability. Adding unlabeled points helps, as is typically true with Markov random walk, but some trials have error rates that make this algorithm unacceptable for actual use.

3.6.3 Consistent results

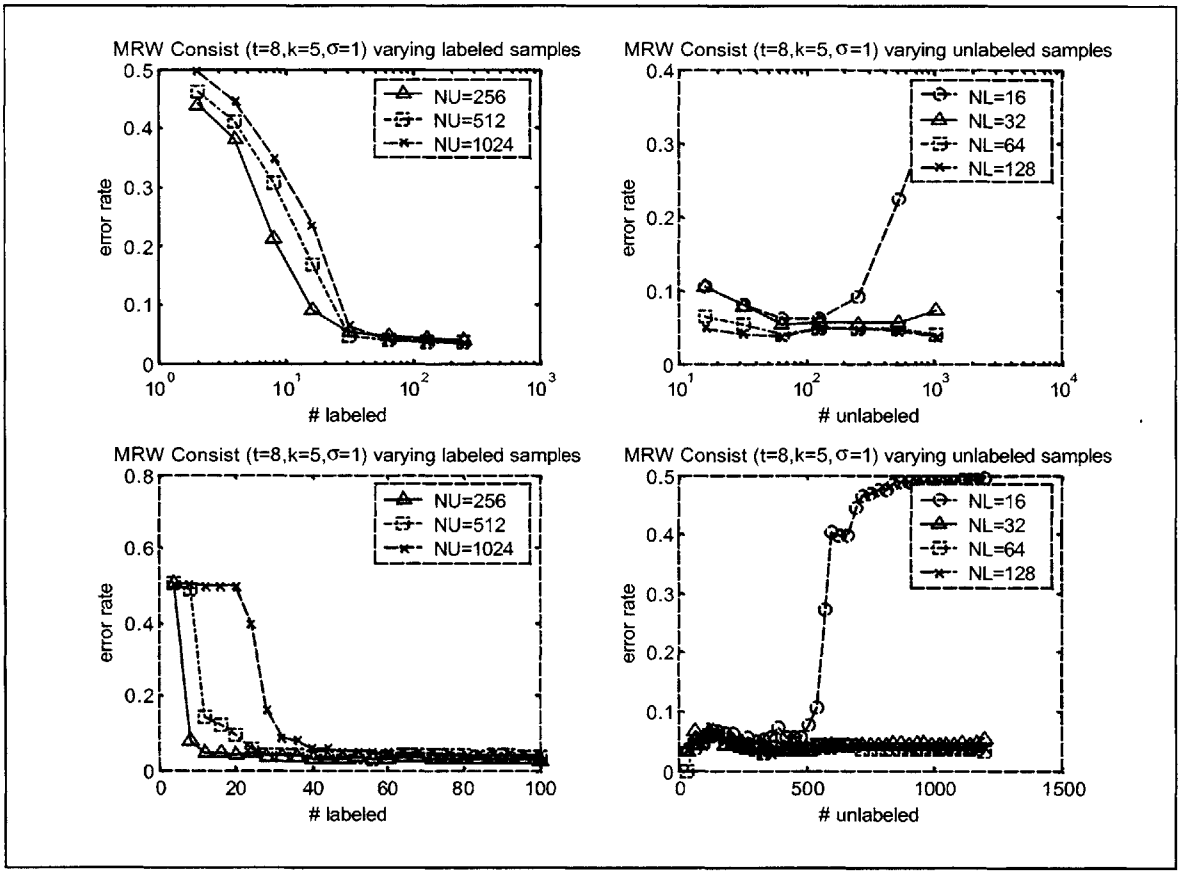


Figure 3-41: Error rates for tests using Markov random walk consistent. $t = 8$, $k = 5$, $\sigma = 1$.

Using the consistent estimation method with Markov Random Walk, the optimal value of σ is found to be 1, with $t = 8$ and $k = 5$. Results are shown in Figure 3-41.

With a very small NL, this method has very poor performance. With just 2 or 4 labeled samples, the error rate is almost invariably 50%, with all samples being labeled in the positive class. This is more likely to be true with larger NU. As more points are labeled, the average error drops, as the classifier less frequently labels nearly everything as positive. In these trials, the error is generally below 10%, and it is very rare that an intermediate error rate is seen.

The individual trial curves show this in more detail. Initially, there is around a 50% error regardless of the amount of unlabeled data. As a few labeled samples are added, the error does not change. At some threshold the error quickly drops, and although the threshold varies from trial to trial, it takes more labeled samples to reach the threshold when NU is larger. In other words, with a lot of unlabeled data, more labeled samples are needed before the classifier can get a reasonable performance. This is because with more data overall, it takes more labeled points to be able to reach all points in just t=8 steps by stepping to neighbors. Another important thing to note is that beyond the threshold, it only takes a few more labeled points before approaching its peak performance. From the trials run, it is typically between 10 and 20 additional samples required. As samples are continually added, there is rarely a change in the error rate.

This method doesn't benefit very much from unlabeled data. The error rates tend to be minimized when NU = 64. The NL = 16 curve has a very sharp rise in the error at 1024 labeled points, which seems to be the opposite of most of the algorithms we have seen so far. However, it is consistent with the other graphs for this method, which simply will not work if there is too much unlabeled data. The NL=16 curve is the only one that spikes upward like this in the region shown, but all of the others will if the NU axis is continued out further.

One possible way to improve the performance is to raise t as NU gets larger in comparison to NL. With more time steps, the labeled points will have more opportunities to reach the rest of the data set. However, for the trials that we calibrated the parameters on, $t = 8$ gave the best results, and for a fair comparison with the rest of the algorithms, we do not vary the parameters as the size of the training set changes.

3.6.4 Maximum entropy discrimination (MED) results

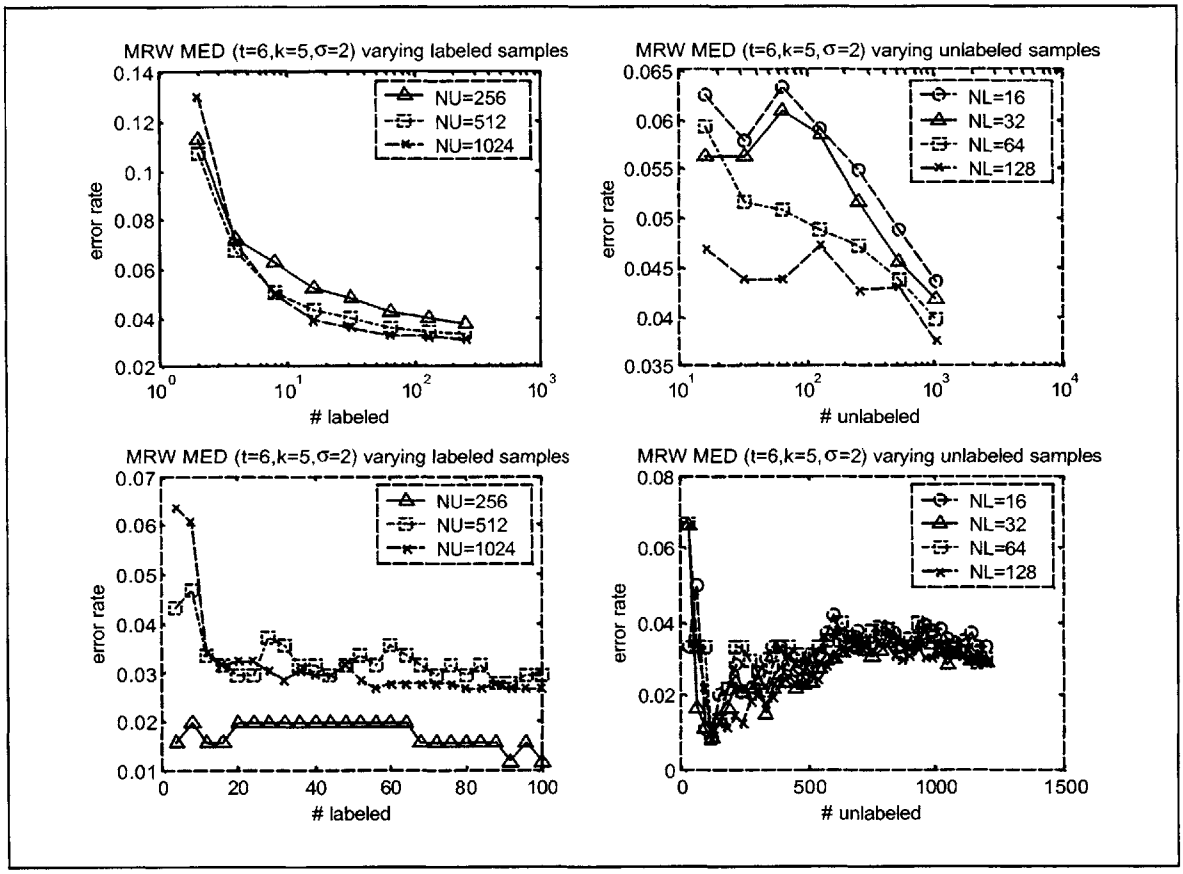


Figure 3-42: Error rates for tests using Markov random walk MED. $t = 6$, $k = 5$, $\sigma = 2$, $C_{rate} = 1000$, $margin = 0.2$.

Using maximum entropy discrimination to determine the label probabilities with Markov random walk has very good classification rates. The results of the trials are shown in Figure 3-42, with parameter settings of t=6, $k = 5$, $C_{rate} = 1000$, $margin = 0.2$, and $\sigma = 2$.

The average error rates drop quite a bit as the first few labeled samples are added, and the improvement slows down but has not completely leveled off when NL=256. At this point, the average errors are around 3% to 4%, depending on how many unlabeled samples are used. This general trend is observed in the NU=512 and NU=1024 individual trial curves, but NU=256 is different. It starts out around 1.6% error with only 4 labeled points, rises to 2%, and eventually drops to 1.2% error with just less than 100 labeled points. This is absolutely phenomenal performance, incorrectly labeling 3 to 5 of

the 256 test points! The 256 test points are also present in the 512 and 1024 trials, and those do not do as well, especially with fewer labeled points. This implies that the 256 points used are placed in such a way that they are easy to classify, whereas the additional points in the other curves make the task slightly more difficult. This seems odd, but with a Markov random walk, adding or removing even a single point will change the neighborhood structure and can change the performance.

Adding more unlabeled samples almost always seems to help as well. Except for when there are 2 or 4 labeled samples, the average error for the NU=1024 curve is better than NU=512, which is better than NU=256. In the curves varying unlabeled samples, there is also a general downward trend, although there are some points where the error rate increases from adding data. While many algorithms tend to improve more as the first initially and then level off, this benefits a lot from a large amount of unlabeled data. The curves are steeper for large values of NU than for small values. The axis of the graph is logarithmic, so the incremental benefit from a single labeled sample is still dropping, but not nearly as much as with other methods.

Because the error continues to decline even around 1024 samples, we run tests with more unlabeled data, and display the results in Figure 3-43. There is a small jump in error from 500 to 1000, which contradicts what we find in Figure 3-42, but the errors are subject to some sampling error. What we find useful is that the error is still declining at 2000 unlabeled points, even if it is a fairly small change. This means that MED parameter estimation in use with Markov random walk can potentially benefit from a very large set of unlabeled data, although parameters such as t may have to be changed.

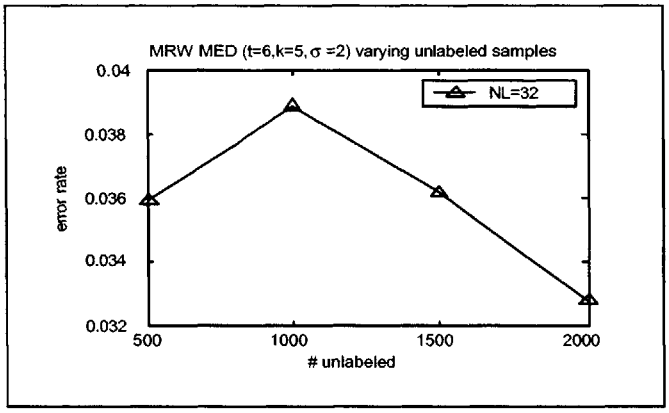


Figure 3-43: Error rates for tests using Markov random walk MED, varying NU, with NL fixed at 32. $t = 6, k = 5, \sigma = 2, C_{rate} = 1000, margin = 0.2$.

3.6.5 EM estimation results

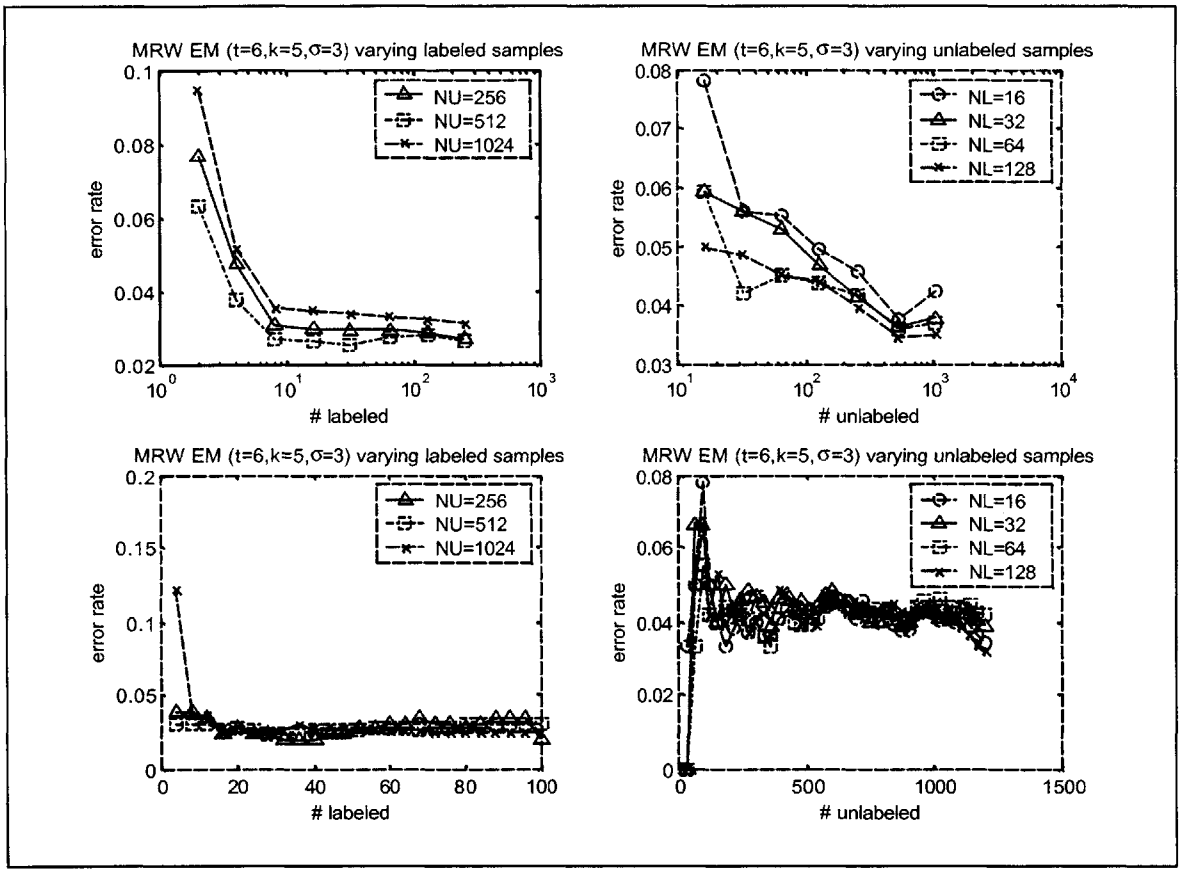


Figure 3-44: Error rates for tests using Markov random walk EM. $t = 6, k = 5, \sigma = 3$.

We show the results using EM estimation in Figure 3-44, with $t = 6$, $k = 5$, and $\sigma = 3$. We can see that this method has excellent performance and benefits from both labeled and unlabeled data.

When adding labeled samples, the error drops fairly steadily from 2 to 8 samples. Beyond this, the error doesn't change significantly as more labeled data is added, but it does occasionally rise, which is not as expected.

Performance with only one labeled sample from each class is fairly good, with average error rates between 6% and 10% depending on how large NU is. With 8 labeled samples total, the performance nears its peak, with average errors between 2.5% and 4%. Some individual trials achieve close to 2% error. These error rates are excellent compared to other algorithms, and are consistent across trials.

Figure 3-45 shows box-plots for one curve varying labeled and one varying unlabeled. The plots point out an interesting trend when varying NL. For a fixed number of unlabeled points, the median and minimum error rates are almost identical for any amount of labeled points. However, the maximum error tends to drop from adding more samples. This shows that this algorithm can have very good performance with even two labeled points, but not consistently. With more labeled data, the high performance levels become more consistent.

From Figure 3-44, we know that the average increases when going from 512 to 1024 unlabeled points. From the box-plot, it appears that with 1024 unlabeled points, the error rates are converging on a very narrow range, while for 512 there is slightly more variation.

The individual curves varying NU tend to oscillate together. When one curve rises or falls, the others follow in the same direction. Figure 3-46 focuses on a small region of the lower-right curve from Figure 3-44 to demonstrate this. This reinforces the fact that once enough labeled samples are present, adding more does not improve the performance significantly.

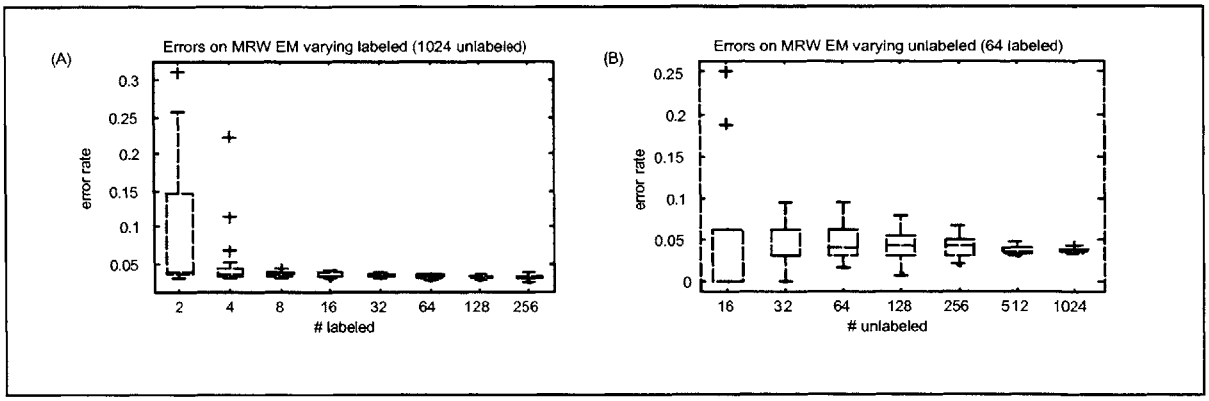


Figure 3-45: (A) Box-plot showing error rates using Markov random walk EM varying NL, fixed at 1024 unlabeled. (B) Box-plot showing error rates using Markov random walk EM varying NU, fixed at 64 labeled.

3.6.6 Markov random walk summary

The results on Markov random walk are somewhat more sporadic than those we have already seen, due to the fact that the test set changes on each trial. However, we are still able to get a sense of which algorithms are most useful.

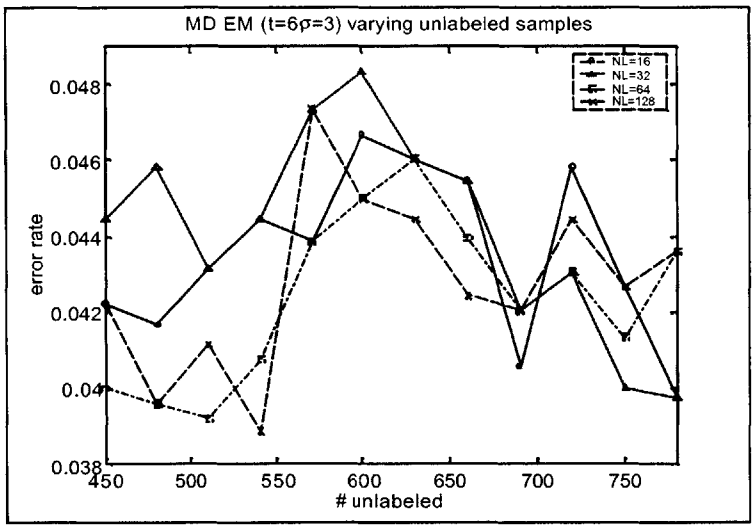


Figure 3-46: Error rates for an individual trial using Markov random walk EM. $t = 6$, $k = 5$, $\sigma = 3$.

Maximum entropy discrimination provides excellent results, with steady improvement from adding labeled or unlabeled data. Unfortunately, this parameter estimation method is not as strong as others when the labeled set is very small. EM estimation gives some lower error rates than MED, but steadies off eventually, so it will not improve with increasingly large unlabeled sets, whereas MED will. Average margin does okay, but varies significantly from trial to trial. Consistent estimation does fairly well under most conditions, but does not work at all when only a small portion of the points are labeled. This is still a potentially useful method for our purposes, but the labeling requirements may be difficult to meet. Due to its unpredictable nature, minimum margin is not useful. Even though some trials have low error rates, other trials do very poorly, so it is never certain to do well on any data set.

3.7 Spectral Clustering

In spectral clustering, we map each sample into a new feature vector based on its distance to other points. The dimensions in the new feature space only tell which clusters a point belongs to. There is no correspondence to the original feature space.

3.7.1 Parameters

Spectral clustering has several important parameters that control its performance. The first parameter, *nclust*, establishes how many clusters the data gets separated into. Using too many clusters will create clusters with very few points. If there are as many

clusters as there are data points, each point will be put into its own cluster and virtually no information is gained, even though the dimensionality has decreased. If the number of clusters in the algorithm is less than the number of clusters actually in the data set, then not enough information is being used. Ideally only 2 clusters would be needed, one for each class, but the images and their corresponding feature vectors are very complex and it is unlikely that the data would cluster this perfectly. It is important to find a number of clusters in between the extremes that will work well.

The second parameter is σ. As usual, σ establishes how points influence each other as their distance varies. This is used for the affinity matrix, which eventually helps determine where the clusters are formed.

3.7.2 Clusters

To see whether the clustering method is likely to work at all on our data set, we run some preliminary tests. We take 1000 points, map to the new feature vectors using the parameter settings, and then use k-means to assign each point to a cluster. When using spectral clustering as a classifier, we use a linear SVM rather than *k*-means to separate the clusters, but this will give an idea of what points are nearby using the converted features.

If clustering works perfectly, then each cluster will contain all samples from the same class. Table 3-6 shows how many positive and negative points are contained within each cluster, and unfortunately it does not work as well as expected. Clusters 3, 5, 6, 7, and 9 contain points from only one class, but they are very small, which reduces the likelihood that one of those points will be labeled. What is most troublesome is Cluster 2, which contains a large number of points, but there is about a 3-to-1 split of positive and negative samples. If all points within the cluster are labeled identically, then over 100 of these samples could be mislabeled, which gives over 10% error on the set.

Table 3-6: For a sample clustering task of 1000 points, how many went into each class for each cluster. *nclust* = 10, σ = 1.

Cluster	# Pos	# Neg
1	8	203
2	344	111
3	6	0
4	88	0
5	1	0
6	1	0
7	6	0
8	32	2
9	1	0
10	0	197

3.7.3 Results

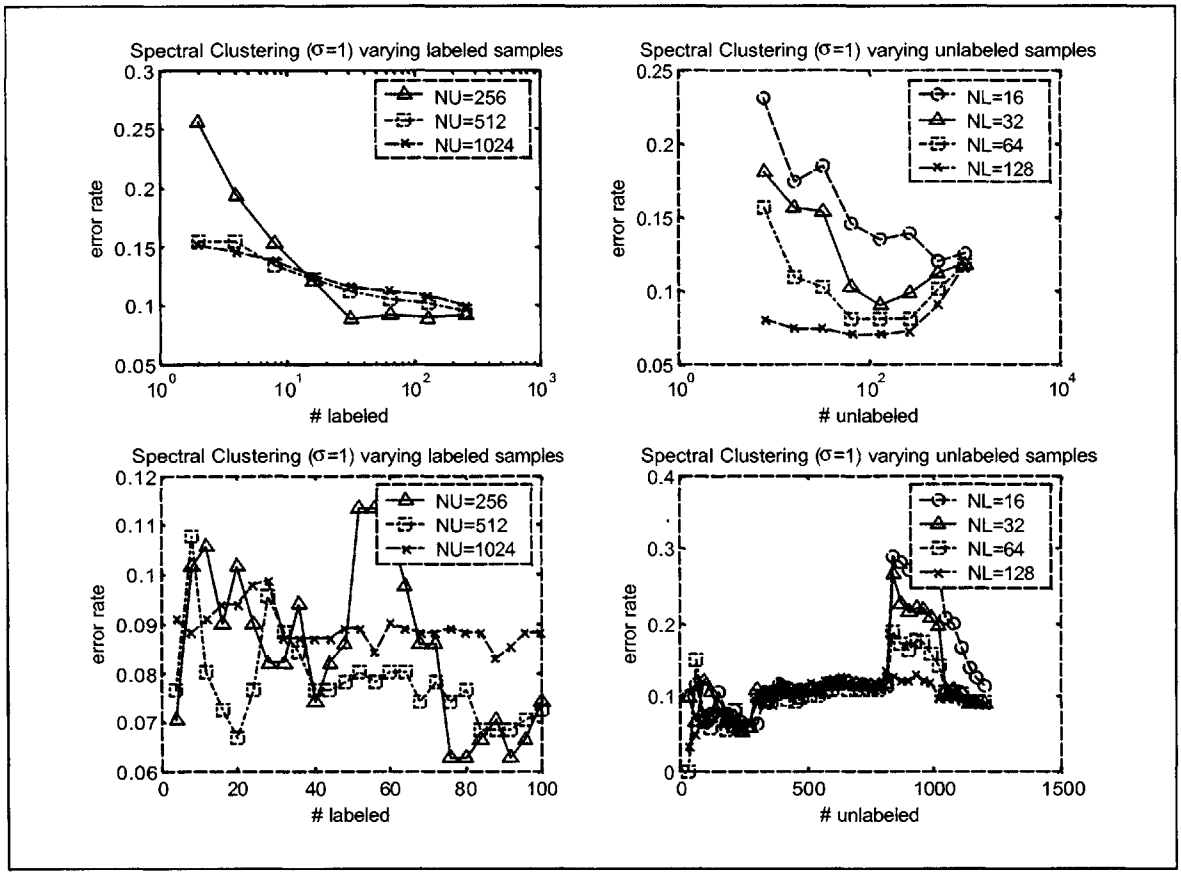


Figure 3-47: Error rates for tests using spectral clustering. *nclust* = 10, σ = 1.

Note that all test points need to be available at training time for this clustering method. Therefore, it cannot be used for testing on new points, so performance can only be measured on the training points. This also means that the algorithm can't be run with

0 unlabeled points, as there are no new points to test the performance. A minimum of 8 unlabeled points are used in the tests, because measuring on a smaller set would be too prone to sampling error and would be a useless measure. The results are plotted in Figure 3-47. It is optimized for performance on 64 labeled points and 1024 unlabeled points and the parameter setting are *nclust*=10 and σ =1.

With only 256 unlabeled points, the performance is initially very poor, but improves steadily when adding more labeled points. Once enough samples have been added, the error levels off. With 512 or 1024 unlabeled points, the performance is better than 256 with very few samples, but they don't improve as much when more points are added. With about 32 or more labeled points, the performance is better on the 256 point set than it is on the others. Looking at the curves varying unlabeled points, the curves with more labeled points are always better, but there is much less of a difference for larger unlabeled sets.

Varying the number of unlabeled points improves performance somewhat, but it seems to break down for very large amounts of unlabeled data. All of the curves see large drops in average error until around 128 unlabeled points. Beyond this, the 32, 64, and 128 labeled point curves get slightly worse from more data, while the 16 labeled point curve improves a little. Despite the rise in error above 128 points, all of the curves except the 128 labeled are still better at 1024 than they are at 8 unlabeled points. This seems natural, as the larger total samples give a good idea of the overall distribution of points, and clustering should be more possible.

The individual trial curves give some interesting insight into this algorithm. When adding unlabeled points, performance tends to steady off for long periods. At certain points, there are large increases or decreases in the error rate for all curves. As can be seen with the jump around 800 samples, more labeled samples dampens the amount of change. Because all of the curves change at the same point, it implies that certain samples being added to the training set can make a huge difference. The error in one case goes from 10% to 30%, which is quite shocking.

For most algorithms, the classification on the labeled training samples is nearly 100%, as the classifiers strive to reach the best classification on these points. Spectral clustering does not always do this well, often reaching error rates above 10% on the

labeled points. In fact, the labeled and unlabeled training samples have similar performance measures, which is quite surprising.

While this algorithm does improve from both labeled and unlabeled data to an extent, the best error rates are still far worse than other algorithms.

3.7.4 Spectral clustering summary

The performance of the spectral clustering algorithms gets much better as NL is increased, particularly when NU is relatively small. This is because smaller data sets will be less likely to give a true indication of the structure of the clusters. Adding unlabeled points also helps up to a point, but beyond this the error rises.

Despite improvements by raising the size of the data set, the errors for spectral clustering are far too high in comparison with other algorithms on our data set. The classifier used after the clustering step is simply an SVM, and comparing to our earlier results, an ordinary SVM easily out-performs clustering. Most likely, clustering is just not a good option on this data set because of the structure of the data. We show that using k-means clustering on labeled points typically leads to large clusters that contain significant amounts of samples from both classes. If performance is this poor on fully labeled data, there is no chance for it to do well on partially labeled data.

3.8 Comparison of algorithms

We have looked at each classifier's performance individually, but to get an understanding of which are most useful on our data set, we will compare the algorithms with each other. In comparing the algorithms, we consider both the time and accuracy, as both are important factors in any practical application.

Table 3-7 compares a few useful measures for each of the algorithms described in this chapter, while Figure 3-48 and Figure 3-49 display the curves for the different learning methods together. In the chart, we list the average performance rates with 2 labeled and 1024 unlabeled to see how well an algorithm does with only one sample labeled from each class. We also list the time and error rates for 32 labeled and 1024 unlabeled, which is a reasonable sized data set for an application of our sort.

Table 3-7: Comparison of each of the classifiers discussed in this section. Columns state whether the classifier can be used to separate data with more than two classes, whether it can classify a sample that was not used in training, average error rate when training with 2 labeled and 1024 unlabeled points, average error rate when training with 32 labeled and 1024 unlabeled points, and the average time to train and test on 32 labeled and 1024 unlabeled points.

Algorithm	Can test non-training points	Error rate[3] for NL=2, NU=1024	Error rate for NL=32, NU=1024	Time for NL=32, NU=1024 (seconds)
3-NN	Y	N/A[4]	8.8 [7.5 10]	160
5-NN	Y	N/A	8 [6.8 8.8]	177
Lin. TSVM	Y	37 [19.5 42.5]	4.2 [3.5 5.5]	367[5]
Gaussian TSVM	Y	30.5 [7.8 47.8]	7.2 [5.0 9.0]	270
KE Avg (const. σ)	Y	32 [19.2 42.8]	8.5 [6.2 10.5]	2.0
KE Avg (BMP)	Y	24 [20.2 38]	8.8 [6.8 10]	3.3
KE Avg (KNN)	Y	20.2 [17.5 25.5]	7.5 [6 8.8]	2.7
KE Min (const. σ)	Y	14 [10.8 14.8]	5.2 [4.5 7]	3.6
KE Min (BMP)	Y	22.5 [12.5 34.5]	10 [8.2 14]	8.3
KE Min (KNN)	Y	14.5 [13.8 16]	14 [12.2 16.8]	8.6
KE EM (const. σ)	Y	31.8 [21.8 40]	9.5 [7.8 11]	3.2
KE EM (BMP)	Y	32.8 [29.2 39.2]	7.2 [6.5 8.8]	11.3
KE EM (KNN)	Y	24.2 [20.5 28]	10 [7.2 15.5]	7.6
KE MED (const. σ)	Y	22.2 [13.8 30]	8 [6.5 9.5]	3.3
KE MED (BMP)	Y	15.5 [13.8 20.5]	5.5 [4.8 6]	5.3
KE MED (KNN)	Y	17.2 [13.2 20.2]	5.8 [4.5 6.2]	5.4
MRW Avg	N	3.8 [3.7 4.1]	3.6 [3.5 3.8]	15.7
MRW Min	N	3.7 [3.6 5.5]	5.6 [4.3 9.7]	16.3
MRW Consist	N	50.7 [49.7 50.7]	5.7 [4.7 7.4]	27.3
MRW EM	N	4 [3.6 9.2]	3.5 [3.3 3.9]	24.9
MRW MED	N	9.3 [5.5 20.9]	3.6 [3.4 4.1]	19.1
Spect. Clust.	N	15.7 [12.1 16.9]	11.7 [11.3 12]	7.6

[3] To give a sense of the range, the median error level is reported, with 1st and 3rd quartiles in brackets.
[4] Because k-NN needs several neighbors to be labeled in order to work properly, we do not run tests with fewer than 8 labeled samples.
[5] The TSVM code is written in C and called from Matlab, while all other algorithms are written and called from Matlab. This accounts for some of the difference in times.

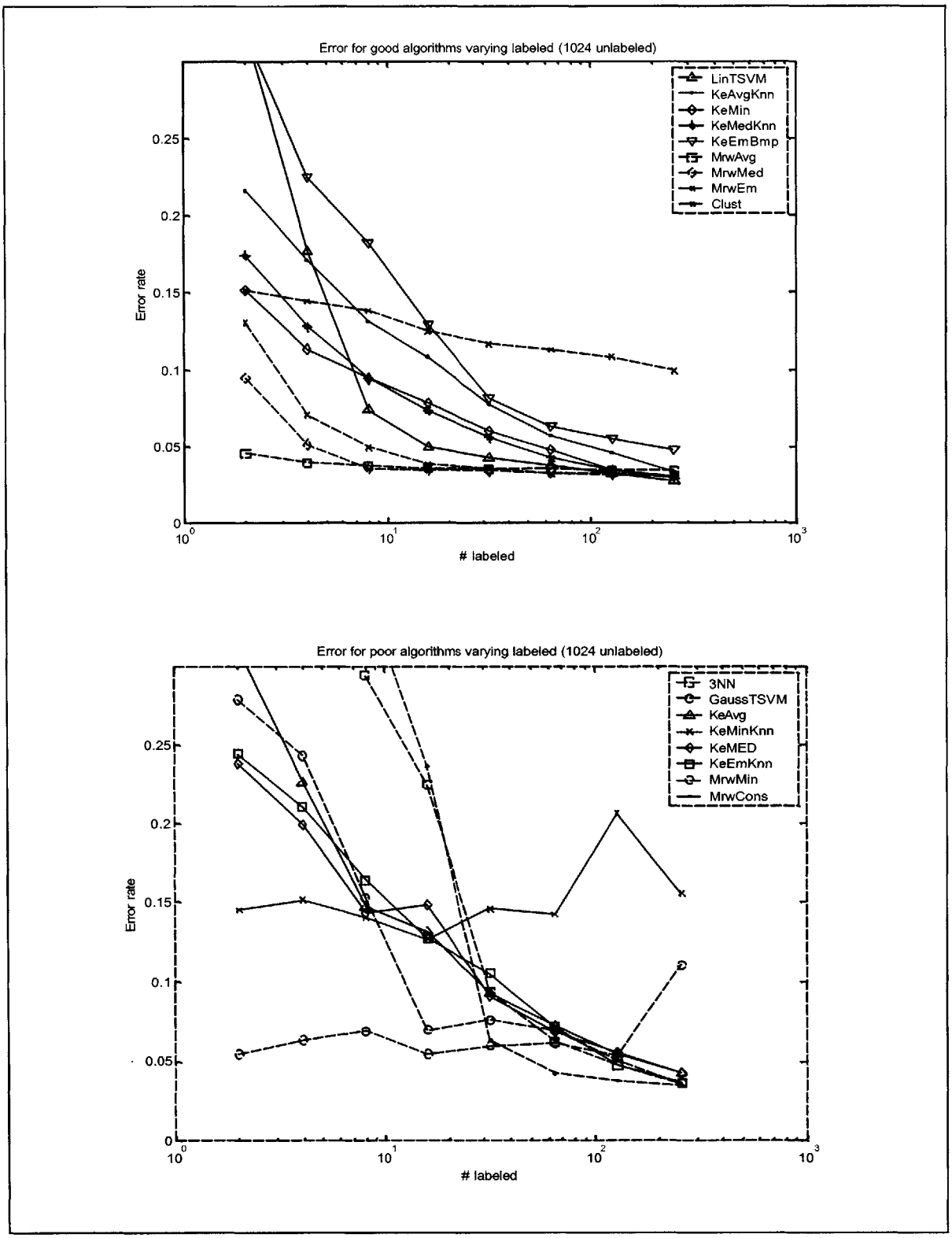


Figure 3-48: Charts displaying error rates for all algorithms varying NL while NU is fixed at 1024. The top figure shows algorithms that tend to improve from adding unlabeled data, and the bottom figure shows those that do not.

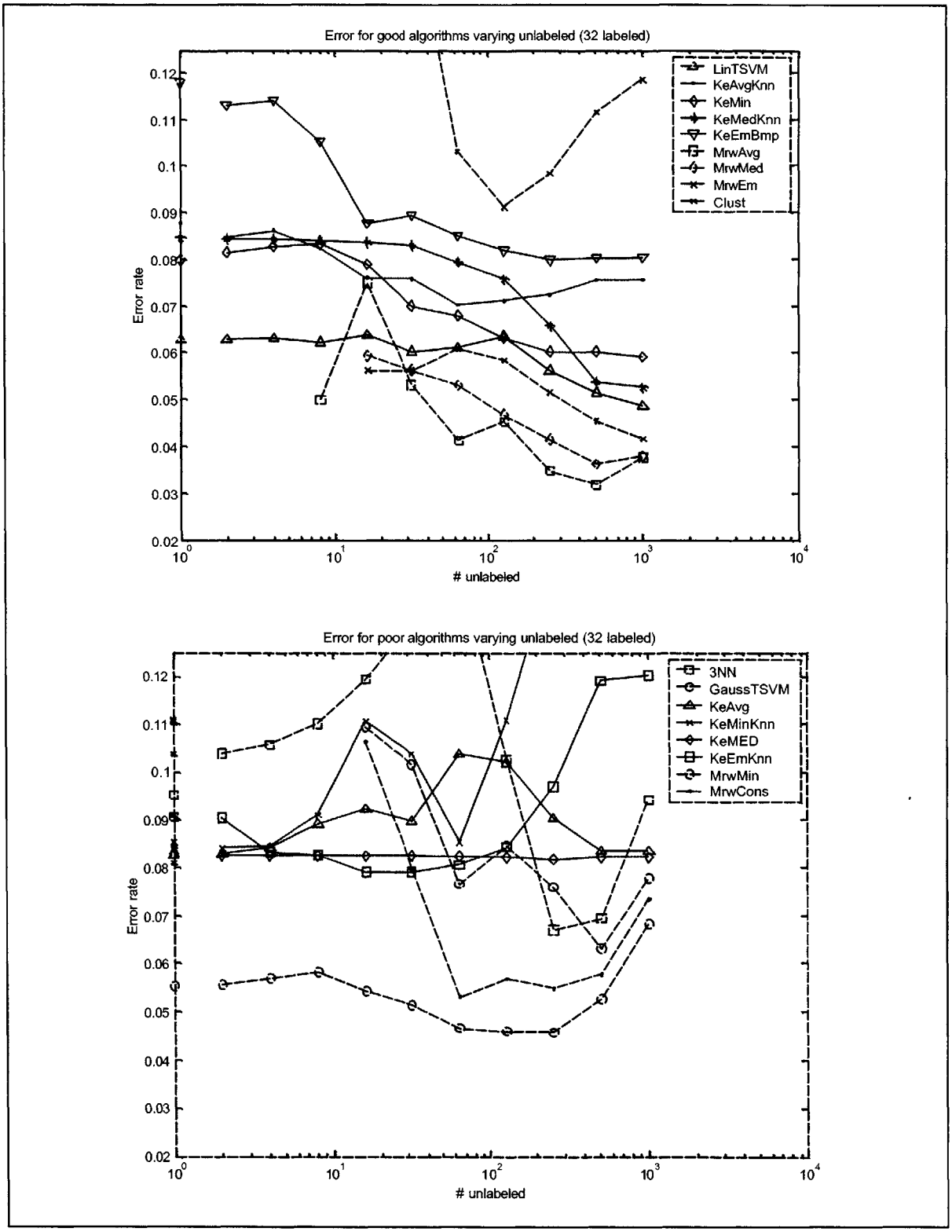


Figure 3-49: Charts displaying error rates for all algorithms varying NU while NL is fixed at 32. The top figure shows algorithms that tend to improve from adding unlabeled data, and the bottom figure shows those that do not.

Something important to consider is that we had a database of limited size. Many car images in this set were difficult to classify simply because nothing else in the database looked similar enough. In a real system, this problem can be avoided by collecting more data. Because all of the classifiers are at this same disadvantage, it should not affect the comparison across various algorithms.

What does affect the comparison, however, is that some algorithms are tested only on points that were used for training, and some were tested on separate points. Results for Markov random walk and spectral clustering are based on classification accuracy on the unlabeled training points, while all of the others are tested on a fixed test set of 400 samples.

3.8.1 Training and testing time

The training of our classifiers can take place while the car is not in use, which will not require it to be optimized for speed. Our tests use at most around 1000 samples, while the amount of data in an actual system may vary by orders of magnitude. The algorithms take more time with substantially more data, so even though we are not trying to optimize for speed, we want the classifiers to be relatively fast.

The times listed in Table 3-7 are the total time to train the classifier and then test on the test set. Pre-processing that is not included in this time is converting images from pixels into feature vectors and calculating distances between samples by a Euclidean measure. [Viola and Jones, 2001] describe ways to optimize the time to convert an image into the feature vector, and it is fast enough to occur as the samples are collected. Computing the distances between samples, can be rather slow. On a 2 GHz machine, computing all distances for 2500 points takes 67 seconds and scales with the square of the size of the data set. Some of these computations can occur as the samples are generated, but once the database gets too large, this will take too long to occur in real-time. The TSVM classifiers call the SVMlight program, which is written in C, while the remainder of the code uses Matlab. Part of the time for TSVM is writing a file of the data samples that is read in as input to SVMlight, which is extraneous work. Additional time is required to re-compute the distances between samples. This is partially accountable for why TSVM appears to take longer than the other algorithms.

The kernel expansion classifiers tend to have the fastest training time, with the constant σ method significantly faster than adaptive σ. Average margin is the fastest because it uses a closed form solution. Spectral clustering is also fairly fast, with most of the time required for computing the eigenvalues of the affinity matrix. Markov random walk is slightly slower because of the more complicated matrix multiplications that take place. Nearest neighbor is very slow compared to the other classifiers, with larger values of k requiring slightly more time. Most of the computations with this classifier involve calculating the multivariate norm for several points to break ties. This algorithm can potentially be sped up by making modifications, such as adding more than one point to the labeled set during every step.

What is important in addition to training time is how long it takes to classify a single new point, which is not given explicitly in the chart. Each algorithm requires different computations to classify a new sample.

- **Nearest neighbor**: The distance is calculated from the new sample to each of the training samples, and the label is assigned as the majority of the k nearest points. This is reasonably fast and scales linearly with the number of training samples.
- **TSVM**: The label for the new points is sign($\sum_i \alpha_i y_i K(x, x_i) + b$), summed over only the training samples that are Support Vectors. Because not all points are support vectors, this scales roughly linearly with respect to the number of training points.
- **Kernel expansion**: The label y for a new point is assigned as the value that maximizes $P_{post}(y \mid x_j) = \sum_i P(y \mid x_i) P(x_i \mid x_j)$. The parameters are all computed in training, so classifying a new point is fairly fast, and scales linearly with the number of training points.
- **Markov random walk**: In order to label a point, it has to be used in training, so a new point could potentially require re-training the entire algorithm, which can be a fairly expensive operation.
- **Spectral clustering**: This classifier also requires that it was used in training in order to classify. A method of assigning labels to newer points by extrapolating

previously seen samples is given in [Chapelle et al., 2002], although in our experiments we do not do this.

3.8.2 Test accuracy

It is important that the classifiers we use have low error rates. None of our methods consistently show below about 4% error, while anything useful probably needs to have error rates well below 1%. However, we do not make extensive attempts to select the features or modify the distance metric in a way that will push down the error rates. We are more interested in the relative performance of the algorithms and their trends as data is added.

One interesting statistic to look at is how well an algorithm does with only one labeled point from each class. If the data is appropriately separated into classes and clustered, then this can potentially work with a large corpus of unlabeled points. Most of the Markov random walk methods do very well, with the exception of the consistent parameter estimation, which almost always got 50% error. Spectral clustering also does fairly well, which is rather surprising, because it has problems when clusters exist with no labeled points at all. Kernel expansion and TSVM are not very reliable under these conditions; while some trials give good classification rates, they both occasionally have nearly 50% error, which is as bad as randomly guessing a label.

A more likely situation in the real-world is that we have closer to 32 or 64 samples that are labeled, and a lot of unlabeled data. The algorithms which have the best classification rates in this situation are Markov random walk with average margin, EM, or MED parameter estimation, and a linear kernel TSVM. With 32 labeled and about 128 or fewer unlabeled points, a Gaussian kernel TSVM has extraordinary performance, but it is unable to maintain these rates with additional unlabeled points. Under these conditions, spectral clustering and nearest neighbor do poorly compared to the other methods.

3.8.3 Same σ value

As mentioned earlier, we always optimize the parameters individually for each method. For example, the term $e^{\frac{-d(x_i,x_j)^2}{2\sigma^2}}$ appears in the equations for nearest neighbor,

Gaussian kernel TSVM, and kernel expansion, yet the values of σ found vary by up to a factor of 10 in some cases. In an application where selecting the optimal parameters is difficult, it may only be possible to choose one value of σ to use. We run experiments and track performance where several methods have σ set to the same value and run on identical data sets. In these tests, NL = 64, and the number of unlabeled points varies from 0 to 1024. In Figure 3-50 we show the results from these experiments, as well as the results for the same data sizes when the optimal values for σ are used.

Note that in some cases, the results for an algorithm improve from using a value of σ other than what was found earlier to be optimal. This does not necessarily mean that the values we previously found were not optimal, because the data sets used to set parameters were of a different size.

When σ is small (0.5), all of the algorithms shown have similar performance. As σ is increased to 1.5, kernel expansion with average margin estimation gets significantly worse, while the other algorithms are still performing within about 1% of each other. Increasing σ even more is when there starts to be more of a discrepancy in the performance of the different algorithms. The Gaussian kernel TSVM improves while the others get worse.

The performance each of the four types of classifiers shown is best for values of σ near its optimal value. It is also interesting that the trends are similar for different values of σ. For example, with 3-nearest neighbor, adding unlabeled data always causes a slow increase followed by a sharp decrease, then a rise again at the end of the chart. The error levels are shifted slightly and the amounts of change vary, but the pattern is clearly visible for all values of σ. This helps support some of the conclusions we have made throughout the chapter regarding the outcome of different amounts of data, as the results are repeatable for different parameter settings.

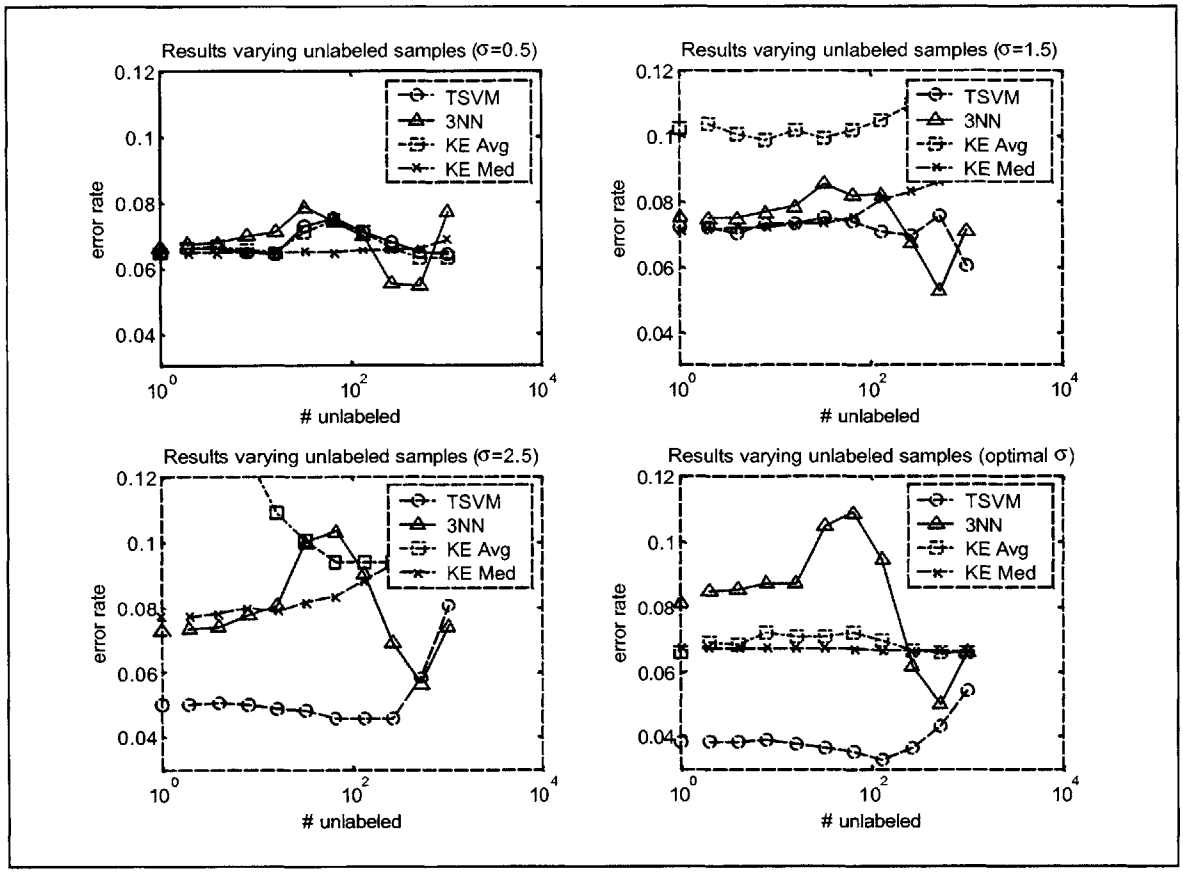


Figure 3-50: Results of experiments using σ = 0.5 (upper left), σ = 1.5 (upper right), σ = 2.5 (lower left), and optimal values of σ (lower right) for 64 labeled samples and 0 to 1024 unlabeled samples.

Chapter 4

Additional Tasks

The tests we run in Chapter 3 explore the various algorithms, but as we present them they can't be used towards a real application. There are several properties of a real data set which are not met in the previous experiments. In this chapter we discuss some of these issues and how we can modify what we have done so that it will be more useful in a true application.

4.1 Unbalanced classes

Typically, real-world problems do not have equally balanced classes. This can occur when part of the feature space is less dense or when samples from one class occupy only a small piece of the feature space. For example, in text classification domains, certain types of documents are more readily available. Image recognition domains will also tend to focus more on certain classes. In the case of cars and non-cars, there will be a much higher proportion of non-cars, simply because randomly selected images are very unlikely to contain a car. One can usually set the ratio of the labeled data in any way desired, as those samples are manually chosen. If the unlabeled points are automatically retrieved, they will be closer to the actual ratio of the data.

The tests run in Chapter 3 assume that classes are evenly balanced. Here we run tests that are closer to the true situation. A randomly selected image that is 640x480 pixels can be broken down into thousands of sub-windows, shifted and scaled to different sizes. Of these thousands of image windows, only a handful will contain a car scaled and centered appropriately to be placed in the car class. Assuming unlabeled data will be constructed from random images in this way, around 99.9% of the data will be non-cars. We don't use this true balance, but we set about 90% of the unlabeled data to be non-cars. This will help show some trends of the algorithms in the presence of unbalanced data. Some algorithms may be more resistant to unbalanced classes than others, as we will see.

We run one set of tests fixing the unlabeled set at 1024 samples, 90% of which are in the negative class, while varying NL from 2 to 256. Another set of tests fixes the amount of labeled data at 64 samples, equally balanced, and varies NU from 16 to 1024 samples, with 90% of the data in the negative class. In all tests, the labeled data has an equal number of positive and negative training samples.

Instead of measuring the total error ratio, as in the previous tests, here we measure recall. Recall is defined as the number of positive samples in the test data that are found by the classifier. This is equal to $\frac{TP}{TP+FN}$ where TP is true positives (the number of positive samples that the classifier identifies as positive) and FN is false negatives (the number of positive samples that the classifier mistakenly calls negative.) This measure ignores false positives, which are negative samples that are identified as positive. While it is important to keep all errors low, in this domain we are more concerned with the classifier detecting as many images of cars as possible. We do not make an attempt to improve classification on the algorithms when there is unbalanced data. These experiments are merely to see how the performance differs when the classes are not equally represented.

Figure 4-2 displays the results of several algorithms, with recall measured as a function of the number of labeled or unlabeled samples. When comparing across charts, note that the scale on the recall measure varies for each algorithm to best view the performance over the range.

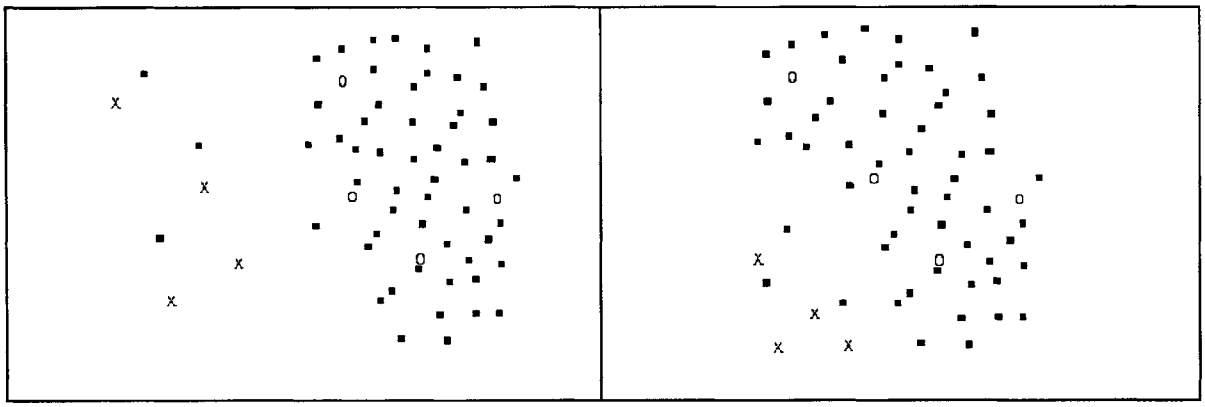


Figure 4-1: Some example situations of unbalanced data. X is a labeled positive sample, O is a labeled negative sample, and dots are unlabeled points. The labeled points have a 1:1 ratio, but there is a much larger collection of unlabeled points in the negative class in both figures. In the figure at left, the X points are found in a less dense region of the feature space, while in the figure at right, the X points occupy only a small portion of the feature space.

4.1.1 Results

K-nearest neighbor

With a lot of unlabeled data, nearest neighbor does not do as well with unbalanced classes. Because there are so many more negative points, the data will typically be skewed towards the negative class. Therefore, more points tend to be labeled as negative samples. This means that the precision is fairly high, i.e. if a point is identified as positive, it is fairly likely that it actually is. However, the recall, which is more of a concern for us, is very low.

Transductive SVM

For TSVMs with enough labeled data, having unbalanced data isn't too detrimental. With 64 samples and any amount of unlabeled data, the recall is usually within 2% of the rate when the classes are equal. With 1024 unlabeled and fewer than 8 labeled points, a linear kernel does very well, but a Gaussian kernel has a very low recall rate.

Earlier in the paper, we discussed the C parameter for a TSVM, which sets the penalty for points lying near the margin or on the wrong side. Rather than using one value, separate parameters can be set for the positive and negative class, C_+ and C_- [Morik et al., 1999]. For example, we can set C_+ to be much larger than C_- to try to limit the false negatives.

Kernel expansion

Using most estimation techniques for kernel expansion, the performance with unbalanced classes follows the same trends as balanced classes. The recall rate is generally about 5% lower with unbalanced data for average margin and MED. For minimum margin, the recall is up to 2% higher than when the data is balanced. EM also does slightly better with unbalanced data when there are about 256 or more unlabeled points.

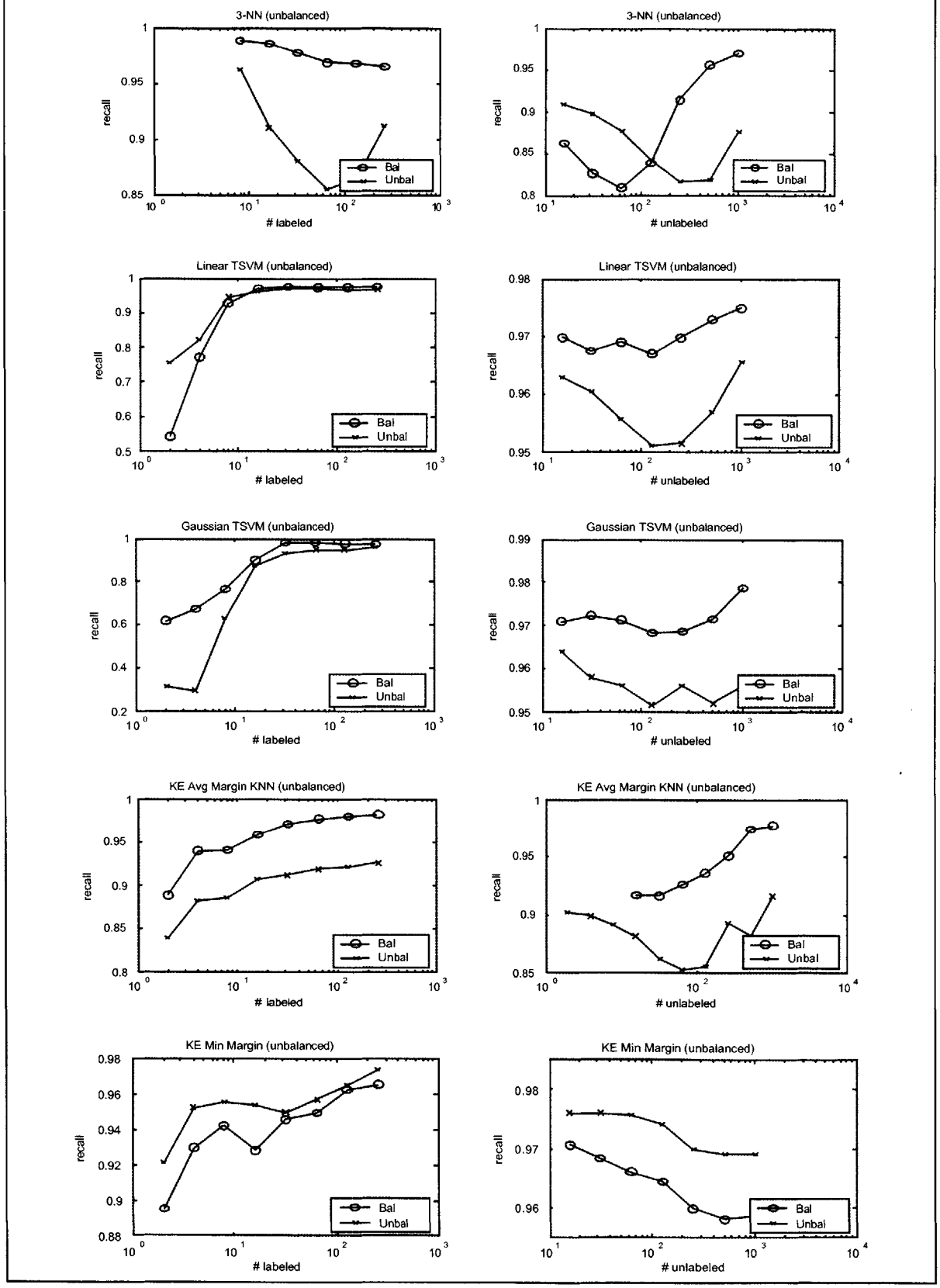
Markov random walk

For Markov random walk and spectral clustering, the results are based on a slightly different measure. For the other algorithms, a fixed test set of 200 points from each class is used. With these two algorithms, the performance can only be tested on the training points. For small values of NU, there will be only a few positive samples in the test set. Recall measures the percentage of positive samples that are detected, and because there are so few positives, these results will not be as reliable as some of the others.

Using MED parameter estimation, Markov random walk does fairly well with unbalanced data. The recall rate with two labeled samples is just below 95%, but this rate does not increase significantly from adding more data. Varying NU, the performance is erratic, but stays within a few percent of the balanced curve. EM parameter estimation does not work as well under unbalanced classes, as the recall rates tend to be 4% to 8% lower than balanced training data.

Spectral clustering

When very few samples are labeled, spectral clustering does not do very well in the presence of unbalanced data. However, as more samples are labeled, the recall rate increases steadily. When there is a larger proportion of negative samples, more clusters will form in those areas of the feature space. If more points have labels, it is more likely that the clusters will all be represented with at least one labeled sample.


3-NN (unbalanced)
recall
labeled
Bal
Unbal
3-NN (unbalanced)
recall
unlabeled
Bal
Unbal
Linear TSVM (unbalanced)
recall
labeled
Bal
Unbal
Linear TSVM (unbalanced)
recall
unlabeled
Bal
Unbal
Gaussian TSVM (unbalanced)
recall
labeled
Bal
Unbal
Gaussian TSVM (unbalanced)
recall
unlabeled
Bal
Unbal
KE Avg Margin KNN (unbalanced)
recall
labeled
Bal
Unbal
KE Avg Margin KNN (unbalanced)
recall
unlabeled
Bal
Unbal
KE Min Margin (unbalanced)
recall
labeled
Bal
Unbal
KE Min Margin (unbalanced)
recall
unlabeled
Bal
Unbal

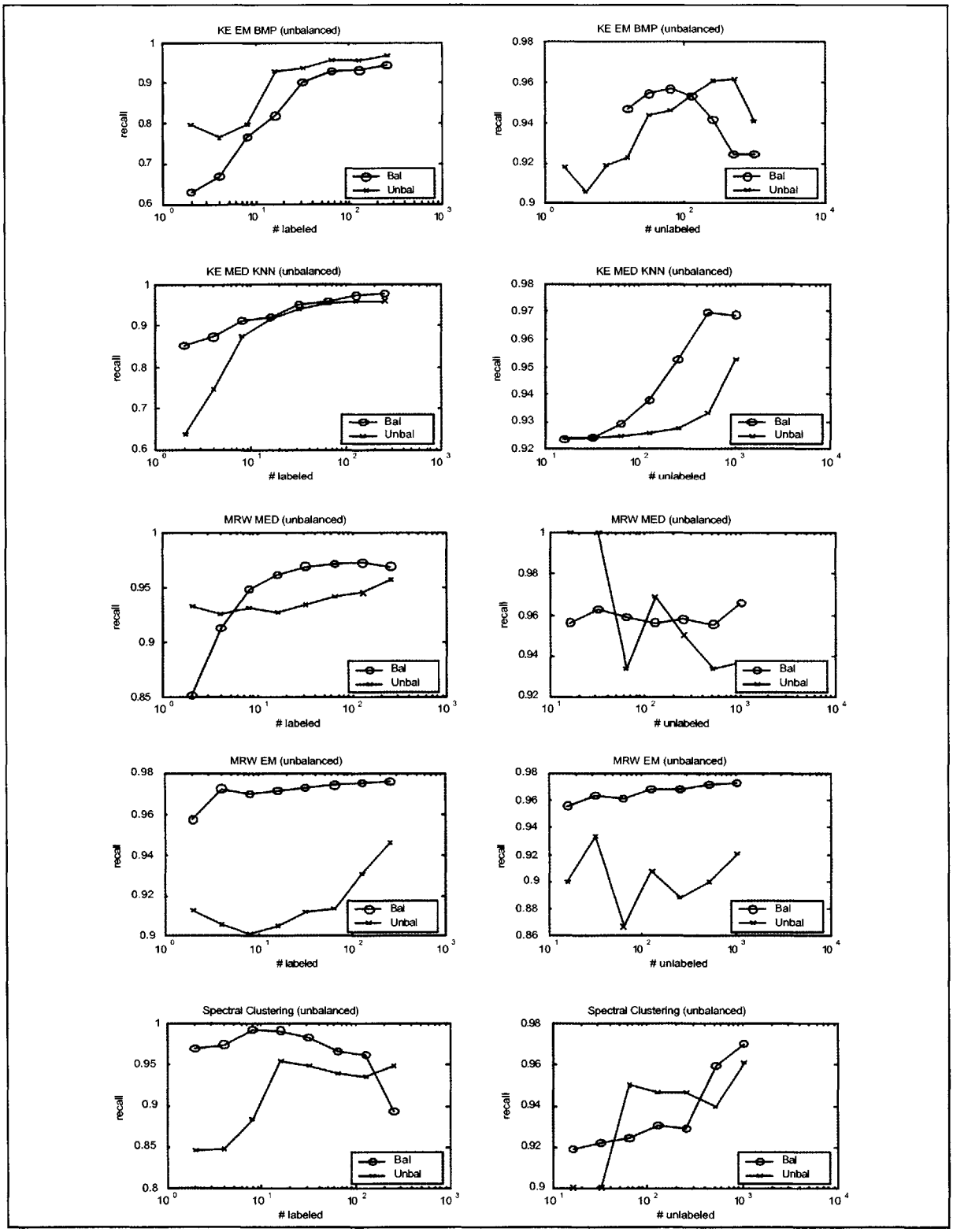


Figure 4-2: Results on tests with unbalanced data, compared to tests with equally balanced data for the same algorithms under identical conditions. Experiments in the left column vary labeled data with 1024 unlabeled samples, and experiments in the right column vary unlabeled data with 64 labeled samples.

4.2 Pre-selecting labeled points

So far, all labeled and unlabeled samples have always been randomly selected for each trial. In reality, it is more likely that a human will select the labeled samples. Some knowledge of the classification task will go into selecting the appropriate samples to label. In the domain used here, it will be wise to use a very diverse set of images which covers as wide of a spectrum of the images that are likely to be seen. A well-chosen set would include images of cars on different types of road, in different poses, and under different lighting conditions, and the non-car images would cover a wide variety of background objects.

There are a number of different ways to select the samples to label for the algorithm. These may work better than if a randomly chosen labeled set of the same size is selected. Using the following methods, a labeled set is incrementally built, a few samples at a time.

- Active learning: We run an SVM on the labeled points and test on all others. The points from each class which lie closest to the margin are added to the labeled set. [Schohn and Cohn, 2000]
- Maximum average distance: Points which have a large average distance measure to the labeled points are added to ensure variety in the labeled set.
- Maximum classification error: Using any classifier, a point may be difficult to identify because there is no labeled point similar to it. Constructing the labeled set as points which are frequently mis-classified by any algorithm can help to label other points.

We run some classifiers on the original data set when selecting the labeled points in these ways. All tests are run with 512 randomly chosen unlabeled points and between 4 and 64 labeled points. Every trial adds four labeled points in addition to those used in the previous trial. The class balance stays around 50% throughout the tests, but sometimes there are more samples from one class.

Because the labeled set does not vary from one iteration to another, as it does in earlier tests, there should not be as much sampling error. The average error over 10 iterations is reported, and compared to the tests of randomly selecting labeled points. The

results are all run on the same test set, so it is a fair comparison across the different methods.

The methods of selecting points to label are described in more detail in the following sections. The selection methods all leave out the final test set to avoid overfitting to these points.

4.2.1 Active learning

When running an SVM, or any comparable algorithm that gives continuous outputs, certain samples will be difficult to classify. These samples will have outputs that are small in magnitude, lying close to the margin. By taking these samples and adding them to the labeled set, the classifier will get better at distinguishing points near the margin.

Initially, four randomly selected samples are selected to be the initial labeled set. An SVM is trained on these points and tested on all others. Two test points from each class that lie closest to the margin are given labels and added to the classifier. The process is repeated until 64 points have been labeled.

4.2.2 Maximum average distance

If two points in the labeled training set are very similar to each other, then the classifier is not likely to do much better than if only one of the points is used. To avoid this, new samples can be added that differ the most from those that are already labeled. This should ensure that the labeled samples will be diverse and span the entire sample space. The positive and negative points are selected separately, considering only the distances to labeled points within their own class.

The same four points from section 4.3.1 are used as the starting labeled set. For all remaining points, the average distance to the labeled points within the same class are computed. The point with the largest distance is put into the labeled set and the process is repeated, until 32 points have been labeled from each class. The points are kept in the order they were added, e.g., when running a classifier with 16 labeled points, the samples used are the first 8 to be added from each class.

4.2.3 Largest classification error

Some points will be difficult to classify because they are not like other points in the training set. By finding which points are hard to classify and assigning them labels, it may become easier to label other points. This is done by randomly selecting 16 points to label, then training a classifier and testing on all the remaining data. Fifty trials are run on an SVM, and 100 trials with kernel expansion. The number of times each sample is assigned the wrong label is recorded. Samples are ranked by the number of times they are mislabeled, and added to the labeled set in this order. No class balance is taken into account here, but the classes are fairly evenly represented.

4.2.4 Results

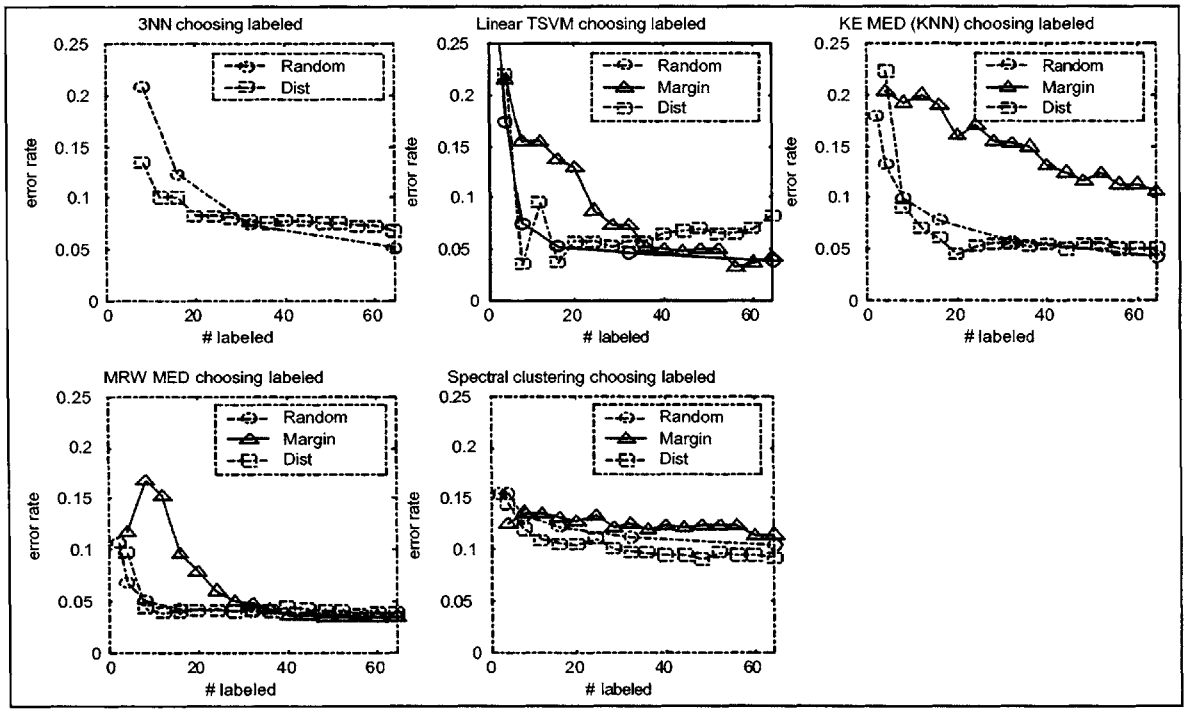


Figure 4-3: Errors plotted for five different classifiers, choosing the samples to label by random selection, active learning margin, and maximum average distance.

Tests with manually selected labeled points are run using the following classifiers:

- 3-nearest neighbor
- Linear kernel TSVM
- Kernel expansion with MED parameter estimation and KNN adaptive σ

- Markov random walk with MED parameter estimation
- Spectral clustering

Each algorithm is run with the parameters determined in Chapter 3. Figure 4-3 displays all of the results, with curves plotting performance when using randomly chosen points (Random), points closest to the margin with active learning (Margin), and maximum average distance (Dist). The results are not displayed for trials where points that are most difficult to classify are assigned labels, as this method did not work at all.

Nearest neighbor

For the nearest neighbor classifier, the maximum average distance measure is the only point selection method that works. This gives better results than random until 32 samples are labeled, and then receives very little improvement beyond this. This method is essentially spreading out the points in space as much as possible. As the first few points are added to the training set, more of the feature space is covered by a labeled point, which is important for nearest neighbor. Once enough points have been added, most unlabeled points will be near a labeled point, and the average distance to a labeled point will no longer be a useful measure.

Both the active learning and hardest to classify metrics led to about 50% error, with almost all samples being placed in the positive class. Neither of these methods does anything to ensure that points are spread out across the space, which is essential for nearest neighbor to work.

TSVM

Selecting the points to label helps a TSVM, but not as much as expected. The average distance method works well when there are very few labeled points, but starts to get worse beyond this. Once enough points are added through this method, any more points are likely to lie far away from the center of the labeled cluster. These points are spread out from the rest of the data and will either not affect the margin, or will push it in the wrong direction.

The active learning method, which uses an SVM to determine the points lying near the margin, does not work well. Once 36 or so points have been labeled, it surpasses

the performance of the maximum distance method, but it performs no better than random selection beyond this.

Using labeled points that are difficult to classify seems like it could help, but here it works very poorly. With 4 labeled points, the error is slightly above 50%, and gets worse as more data is added. This means that a classifier would do better by assigning the opposite label of what this method says. The original idea behind this way of selecting points was that some points are hard to label because there isn't any training data like it. The points used are selected by running several iterations of SVM and kernel expansion, and using the points that are most frequently mislabeled. For an SVM, something is mislabeled if it lies on the wrong side of the margin determined from training. Many of the points will be difficult to label because they are very similar to points from the opposite class. Constructing a classifier with these as the labeled points flips the margin, assigning the opposite label more frequently (see Figure 4-4).

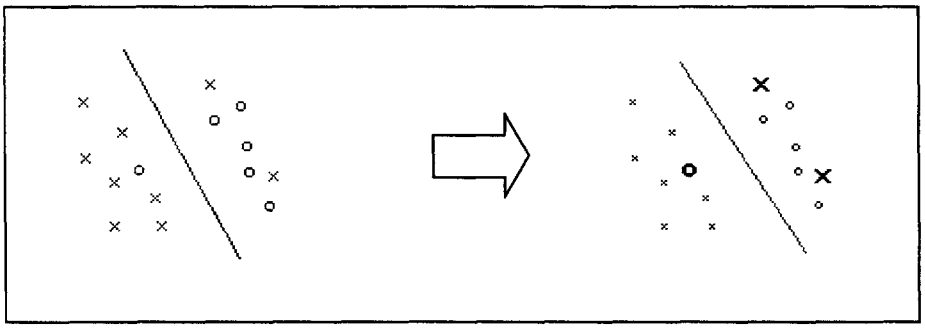


Figure 4-4: A TSVM is trained and run on test points, and some points lie on the wrong side of the margin. A new TSVM that is trained with these points that are on the wrong side of the margin may assign the opposite labels for test points.

Kernel expansion

Results for kernel expansion are similar to those from the TSVM when selecting labeled points. The maximum average distance method gives very similar results, with performance improving quickly, then peaking off around 20 labeled samples. Adding labeled data beyond this causes the error to rise, but eventually it levels off. This is slightly better than the data set on the TSVM, where the error continues to rise from adding more points.

Using the active learning method to label samples does not give better performance than random, but it does continue to improve as data is added. The labeled

samples to use are found from an SVM, so it does not focus as heavily on points that are difficult to classify by kernel expansion.

When using the most difficult points to label, the average error rate starts at around 70% with 4 labeled points. Beyond this, it goes down slightly, and eventually levels off slightly below 50% error, as most of the points are being classified in the positive class.

Markov random walk

For Markov random walk, choosing points by the maximum average distance metric performs almost identically to random selection. With a small amount of data, the margin method does not work as well, but with about 32 or more samples, it is on par with random and maximum average distance. The maximum error method starts at around 50% error with 4 labeled samples, rises slightly as more data is added, then eventually levels off at around 50% error.

Markov random walk is already capable of branching out to samples in the feature space over several time steps. It is not as important for the labeled samples to be diverse as long as there is a large amount of unlabeled data, which is why we see very little improvement here.

Spectral clustering

We have found that clustering often does not work well if the labeled samples in the data do not fully cover the clusters. When points are randomly selected, this can certainly be a problem, which leads to the relatively high error rates. Because of this, the clustering method can potentially benefit from manually selecting the points to label.

The best method is using maximum average distance, for the same reason it works with nearest neighbor. This is the best method to ensure that points will be well spread around the feature space, which is likely to span as many of the clusters as possible. This method consistently gets error rates several percent lower than randomly selecting points.

The margin method does slightly worse than random selection, but still improves performance as more data as added. Using difficult to classify samples is the worst, consistently getting between 70% and 80% classification error.

4.2.5 Conclusions

Surprisingly, selecting points to label does not help classification much, or makes it worse in many cases. The method that seems to give the most benefit is choosing labeled points that are the maximum average distance to the others. However, even this method often does worse than randomly selecting labeled data.

Using active learning does not generally do as well as expected. Typically this technique is used to label samples as the algorithm is progressing, choosing samples that lie near the margin of the data set at hand. However, here we have pre-selected points to label from one fixed set of data. The margin on that set of samples may be different from the margin on another set.

Labeling points that are difficult to classify does not work at all. At best, it places most samples into one class, yielding around 50% error, but in some cases it does even worse. The original intuition behind this algorithm is clearly wrong, as points are often difficult to classify because they are more similar to the opposite class. As a result, labeling these samples can be detrimental, as it often trains the classifier to assign the opposite labels than what it should.

Spectral clustering and nearest neighbor benefit more than other algorithms from selecting the points to label. Both depend on the labeled points being spread out across the feature space, which the maximum average distance measure is able to accomplish.

4.3 Different classification tasks

The results from chapter 3 are important for gaining insight on the algorithms. Nevertheless, it is important to see whether the results can be generalized, or if they only hold true for the particular data set we used. To help answer this, we run similar tests on a new set of data. For the positive data set, there are 1000 images of cars similar to those used in the earlier tests. However, the negative samples are 997 trucks rather than parts of background images. All of these images are resized to 128x128 and converted into a feature vector just as before. Figure 4-5 displays some selected images from each class. Note that images in the truck class display only the bottom portion, with the bottom of the truck roughly aligned with the bottom of cars.

There will be many similarities between the different classes, as both cars and trucks will have edges in common, as well as some similar backgrounds. This will force the classifier to focus on different features than before to distinguish among the classes. With the original data set, there will not be as much similarity between the two classes, which will change the focus of the classifiers.

For some points, histograms showing inter-class and intra-class distances are given in Figure 4-6. The car in the bottom left frame is more difficult to classify, as trucks images are just as similar as many of the cars. Most of the histograms for trucks appear similar to those in Figure 4-6. The nearest trucks tend to be closer than the majority of the cars, but many other trucks have a large distance measure.

Results for tests varying labeled and unlabeled are displayed in Figure 4-7. Figure (A) gives the error measures on the Cars/Trucks database with NU = 1024, varying the number of labeled samples. Separate curves are given for TSVM, kernel expansion MED (KNN adaptive σ), Markov random walk MED, and spectral clustering. Each of these algorithms has its parameters tuned to the new data set. Figure (B) gives the results for these same algorithms on the original data set for comparison. Results on tests with NL = 64, varying unlabeled are shown in Figure (C) for the Cars/Trucks data, and Figure (D) for the Cars/Non-cars data. The new parameter settings are given in the following sections for each algorithm.

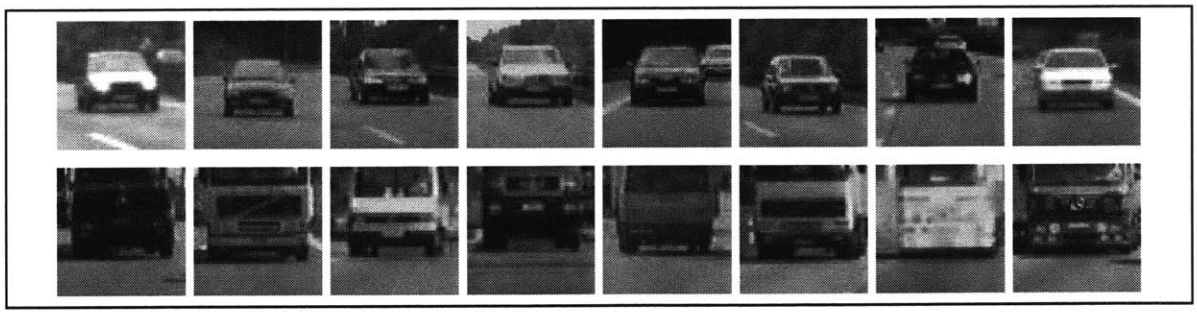


Figure 4-5: Sample images of cars (top row) and trucks (bottom row) from the database.

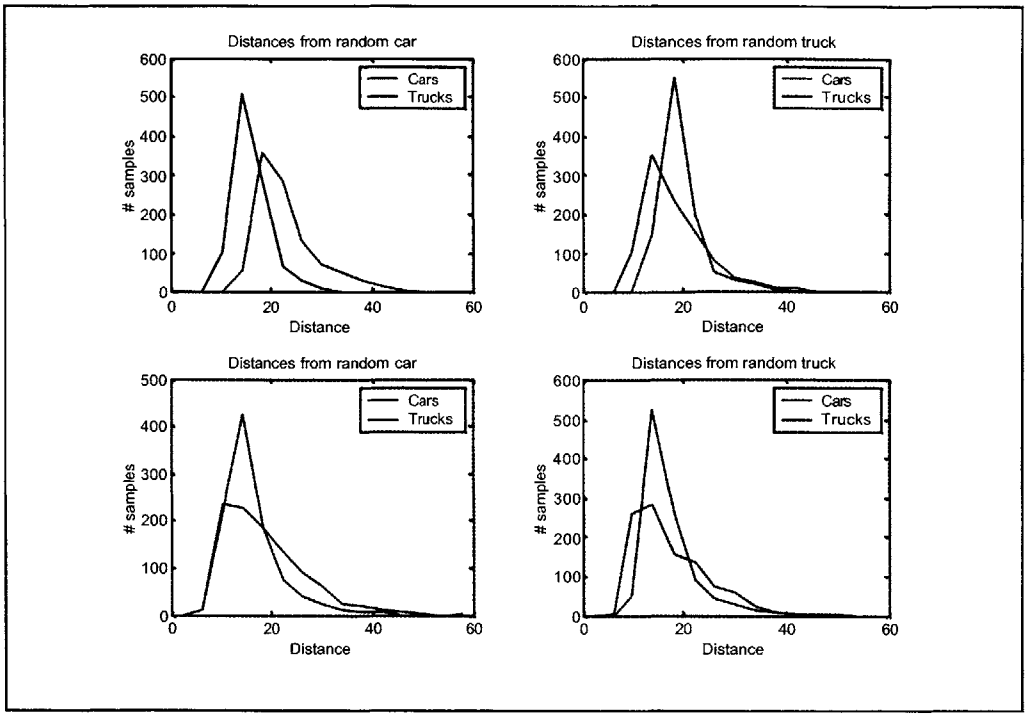


Figure 4-6: Histograms of distances to other samples for two cars and two trucks.

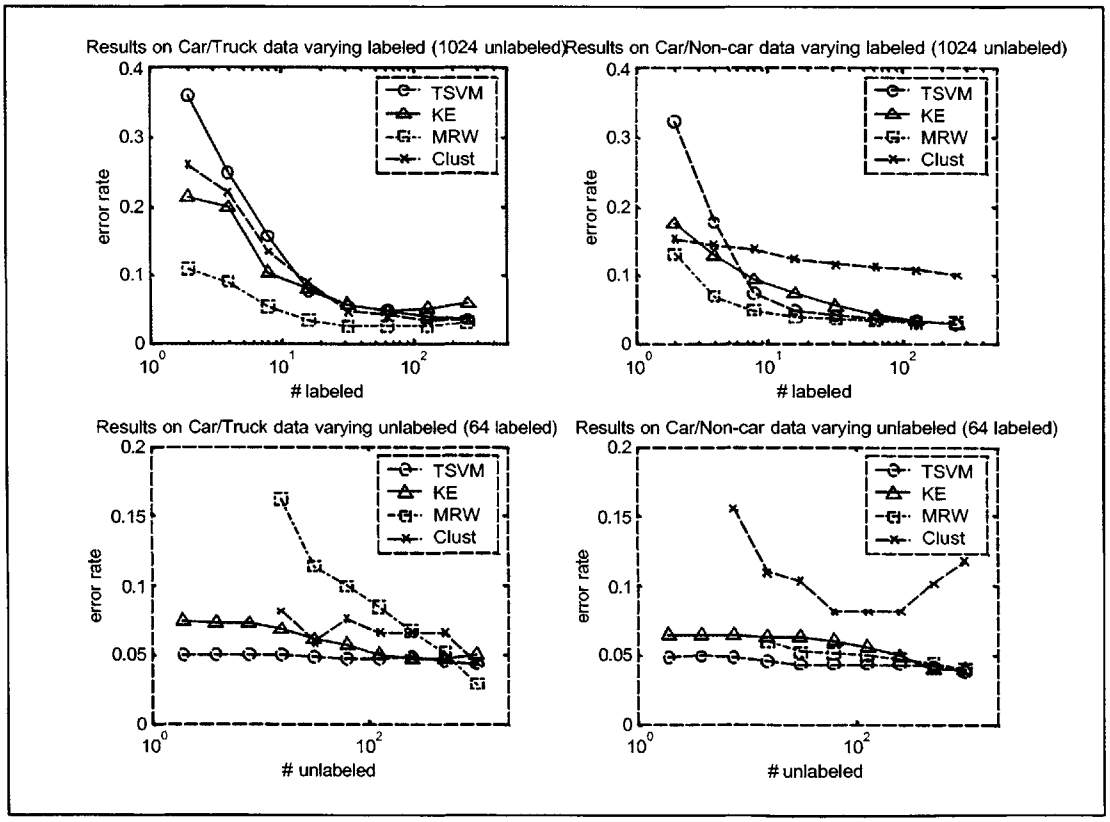


Figure 4-7: Results from tests on Car/Truck database, compared with results for the same tests on Cars/Non-cars database.

4.3.1 K-nearest neighbor

As seen in the histograms from Figure 4-6, the truck images are very spread out in the feature space. This leads to a fairly high probability of error with nearest neighbor, as a truck may mistakenly be labeled as a car. Using bootstrapping leads to more errors, as any mistakes made during early iterations will be compounded. This causes a high number of false positives and very few false negatives. If the main objective is identifying as many car images as possible, this is okay, but for anything that is labeled a car, there is around 50% chance that it is in fact a car.

Nearest neighbor only works on this data set when the bootstrapping method is not used, i.e. when NU = 0. The best value of k is 1, and error rates tend to be around 6% to 10% with 64 labeled samples. Because this does not fit into the semi-supervised learning scenario, we do not explore this any further.

4.3.2 Linear TSVM

For a linear kernel TSVM on the new data set, we use C_{rate} = 500, which is the same setting as the original data. The results here follow the same patterns as observed earlier, with the error decreasing as more unlabeled or labeled samples are added. The error rates tend to be somewhat higher in the new data set, though. The error rate with NU = 1024 and NL = 64 is about 4.5%, whereas the original data averages 3.7% error for the same size data sets.

This data is more difficult to classify because of the many similarities between the two classes. Figure 4-8 displays some of the test points that were most commonly mislabeled in these trials. Three of the images from each class are vans, which appear similar to both cars and trucks. The final car image that is difficult to classify is an ordinary car with a truck behind it. The classifier can easily be tricked by the edges of the truck, which it most likely uses to help distinguish the classes. In the truck set, the final image displayed is in fact a car, which means it was mistakenly placed in the wrong class.

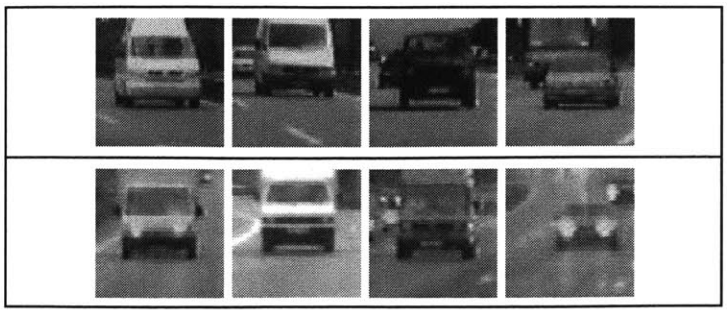


Figure 4-8: Commonly mislabeled samples from the TSVM trials on the Cars/Trucks database. The upper row is false negatives (cars labeled as trucks) and the bottom row is false positives (trucks labeled as cars.)

4.3.3 Kernel expansion MED (KNN)

The parameter settings used on this data set are *kfrac* = 0.1, *sigmamult* = 0.4, *Crate* = 1000, and *margin* = 0.1. Kernel expansion does not work as well on this data set when very few labeled samples are provided. The average error rate with only 2 labeled points and 1024 unlabeled is 21.4%, as opposed to 17.4% with the other data set. For intermediate amounts of labeled data, it approaches the performance of the original data (4.8% error with 64 labeled and 1024 unlabeled, while the other data set has 4.2% error). However, if too many points are labeled, the error begins to rise, which is undesirable for any classifier.

A situation where the performance is slightly better on the Cars/Trucks database than the Cars/Non-cars database is when NL=64 and NU is between 32 and 256. The difference is never more than .5%, which is not very large.

Many of the samples that are most difficult to identify overlap with those found with the TSVM. The same reasons apply here, as these images visually appear to be in between the two classes.

4.3.4 Markov random walk MED

For Markov random walk on the Cars/Trucks database, we set parameters to k = 6, t = 8, C_{rate} = 1000, *margin* = 0.1, and σ = 3. We find interesting results with this classifier that demonstrates an important characteristic of Markov random walks. This data set is primarily made up of sequences of images in both classes, while the other data set contained mostly isolated images. This means that most images have one or more

neighbors that are fairly close, as they come from the same sequence. Markov random walk exploits similarities across low-dimensional curves in the feature space, which is just what these sequences are doing. Points will tend to connect to members of their own sequence within a few steps, and connect to neighboring sequences through several steps.

This holds particularly true on this data set, where the results are very good with a large amount of unlabeled data, but poor otherwise. With 2 labeled and 1024 unlabeled points, the average error is about 10.8%, which is lower than the 13% level on the Cars/Non-cars database. For points further along the curve, the results are generally the same or slightly better on the Cars/Trucks database.

Looking at the curve where the unlabeled points are varied shows much higher error rates. For the curve fixed at 64 labeled points, Markov random walk has higher error rates than any of the other methods until it has 512 or 1024 points. When there are this many data points, most of the sequences will be adequately represented, and the lower dimension paths can be traversed.

4.3.5 Spectral Clustering

In spectral clustering, we use 10 clusters and set σ to 2. This classifier has the largest improvement over the original data set, except when there are very few labeled points. This most likely is true because of the presence of sequences, as with Markov random walk. The sequences correspond well with the concept of clusters; most images from a sequence will tend to be in the same cluster, and a cluster may contain several sequences. As long as most of the clusters contain a labeled point, the classifier can do fairly well.

Mislabeled points tend to be grouped along with other members of their sequence. Similar images are mapped to nearly identical feature vectors and generally receive the same label from the classifier.

4.3.6 Conclusions

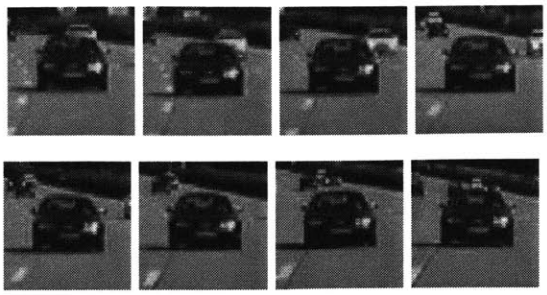
Different image databases force the classifiers to discover different ways of separating the classes. Certain data sets may have properties that make them more suitable for various algorithms.

The inter-class distances in the Cars/Trucks database are not very large because the images themselves are fairly similar between classes. This makes k-nearest neighbor virtually useless on the data because the closest points are not always from the appropriate class. It makes TSVM and kernel expansion less effective because points frequently fall closer to the separating boundary, leading to high likelihood of being mislabeled.

The presence of sequences in the Cars/Trucks data is a crucial difference from the Cars/Non-cars. Markov random walk and spectral clustering benefit from the similarities among images in the same sequence, and thus have improved performance. TSVM and kernel expansion benefit somewhat from the sequences, because it will be easier to classify samples when training with anything that is similar. They do not gain enough from this, however, to offset the drop in performance from the borderline samples. The following section describes how we use the presence of sequences to try to improve a classifier.

4.4 Sequences

Images will often be collected as individual frames of video sequences. From frame to frame, there are some changes in the images, but many features will be similar. For example, consider the sequence of images shown in Figure 4-9. Going through the sequence, the position of the car on the road moves slightly, the cars in the background move, and the lighting conditions change, but for the most part, the images look alike. As a result, their feature vectors are very close to each other. The chart from Figure 4-9 shows the average distance of a selected image to images within the same sequence, other cars not in the sequence, and trucks. It reiterates the fact that images within the sequence are much closer. It is also much closer on average to cars than to trucks, which seems logical.



Average distance from one car image to:		
a car within the sequence	a car outside the sequence	A truck
9.2	15.6	23.1

Figure 4-9: Several consecutive frames of a video sequence. The position and size of the car are kept fixed but the road and background change slightly from frame to frame. For one of these images, we give the average distance to another frame within the sequence, cars outside the sequence, and trucks.

Because samples from a sequence tend to have small distance measures, a classifier will likely do well on a sample if it is trained on other images from that sequence. Earlier tests never used images from the same sequence in the labeled and test data to prevent the classifiers from focusing too much on images from certain sequences. However, in many cases it will be good to have test images from the same sequences as labeled points. There may be dozens of frames available for a given sequence, and training the classifier with a car from one frame can help label all other frames.

We run some tests to see how much of a difference there is on performance when an image from the same sequence was used in training. In the car class of the cars and trucks database, we identify 27 sequences of various sizes, with about 250 images distributed across these sequences. We choose a subset of these sequences and label one image from each of them. The sequences not represented with a labeled point can be among the unlabeled data. The performance is measured separately for images from sequences that did or did not have a labeled point, tabulated over five trials.

Table 4-1 shows the error rates on these trials. Other than Trial 2, there is always a lower error rate on the points that had a member of their sequence labeled, and overall these are mislabeled less than half as frequently.

An example of the benefits from the sequences is shown in Figure 4-10. Image A is mislabeled in trials where there is not an image from its sequence among the labeled data. In a trial that is trained with image B, the other sample is correctly labeled.

For the semi-supervised data scenario, this improvement in recognition using sequences is very helpful. The system can collect a large group of video sequences and a

human can then label a fraction of the frames from each sequence. One of the classifiers can then be used to label the remainder of the data with reasonable accuracy.

Training with image sequences this way is a way to obtain labels for a large training set while labeling just a portion of the images, which is helpful for testing on unseen samples. By gaining many new labeled images, it increases the chances that a new sample is very similar to a training point. Previously, we have found that the accuracy on some tests is limited by the fact that certain points are not like anything that has been seen before.

Table 4-1: Error rates on images from sequence when there is not a labeled point from that sequence in the training data, and when there is. Errors are shown over 5 trials and the total error rate.

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Total
Unlabeled	8/154	1/167	6/139	7/139	9/160	**31/759 = .0408**
Labeled	0/83	4/71	1/99	2/99	1/77	**8/429 = .0186**

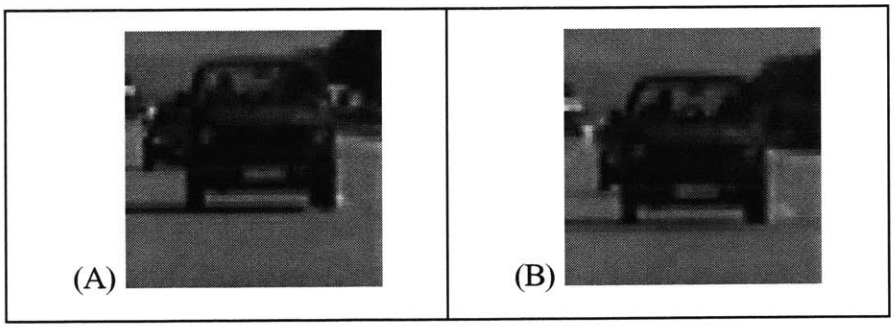


Figure 4-10: (A) An image that is mislabeled when no other samples from the same sequence are in the labeled data. (B) When this image is labeled, image A is classified correctly.

Chapter 5

Summary

5.1 Future work

Our experiments help gain insight into which algorithms are more useful than others on our car images. By themselves, none of the experiments we run accurately model a real situation where this system would be used. Furthermore, the error rates are too high to use for the intended purpose. In this section we describe some ways to address these issues.

5.1.1 True data sets

Our original set of tests runs various semi-supervised techniques on a data set that contains real images of cars and non-cars, but this data set is missing many of the important characteristics found in a real-life data set:

- The unlabeled samples would most likely be heavily unbalanced
- The data would probably contain many image sequences, not just isolated images
- The labeled points would be selected manually to attempt to maximize performance, not randomly chosen

While we address these issues individually, we do not address them all at once. To truly test how well these algorithms will work in a navigation system of the type we have described, the data set should be constructed from several sequences, with the unlabeled points generated by scaling and sliding a window across each image.

5.1.2 Unbalanced data

As we have mentioned, the negative samples will heavily outweigh the positive samples when the data is collected randomly. Even though the classifiers still work under these conditions, it will be helpful to remove many of the unlabeled samples for computational considerations. The classifiers will generally not lose accuracy from

removing a large part of the negative data if it is highly redundant, but it will cut the training time down significantly. To do this, it will be important to have a way of determining which negative samples to remove from the training set.

It will be useful to have a quick reject feature that can quickly determine when an isolated image is definitely not a car without running the actual classifier. Of course, it is very important that this never mistakenly eliminates a car image. This can be accomplished by using properties that all car images have in common. For example, all car images have horizontal and vertical edges in certain positions. Images that do not pass a minimum threshold for these edges can be eliminated from the training set. A system that could analyze points of each class and determine some of these characteristics would work well with our task.

5.1.3 Automatically setting parameters

For each classifier, we manually set the parameters by running the classifier on a portion of the data set and observing which values do well. This requires the labels of the points to be known, because otherwise we can't measure the performance. We are trying to explore algorithms that work without labeling everything, so clearly this is not a realistic situation.

The algorithms do not have the same optimal parameters when the data set changes, as we experienced between the Cars/Non-cars and Cars/Trucks databases. Finding a way to automatically determine some of the parameters from the samples will be very helpful.

5.1.4 Improved distance metric

The distance metric we use in our tests is adequate, but in some cases may not accurately represent the relative distances between samples. The classifiers depend on this distance metric being correct. If the distances are wrong, even the best classification algorithms will not work.

From Figure 3-10, which displays values of w for the linear kernel TSVM, we found that some features are far more useful than others in distinguishing the classes. A distance measure that focuses more heavily on these features could improve performance.

It may also help to change the feature vector representation and keep a Euclidean distance measure. In section 3.4.1.2, we show how to find relevant features from a linear TSVM. Adding weight to the features that are most instrumental in separating the classes could prove useful.

5.2 Conclusions

We have found some promising algorithms that can accurately distinguish cars from trucks or other scenery. A linear kernel TSVM and Markov random walk with MED parameter estimation seem very promising because of their continuing improvement from both labeled and unlabeled data. More importantly, they both are robust and work on additional data sets, and also maintain high accuracy when the training data is unbalanced. Algorithms such as Gaussian kernel TSVM have lower error rates in some cases, but frequently get worse as more unlabeled data is added.

Our results demonstrate that a system such as the one we have described throughout the thesis is certainly feasible. While it requires modifications and optimizations from some of the experiments we have run, we show which algorithms are useful under various conditions.

Bibliography

[Bateson, 1972] G. Bateson. *Steps to an Ecology of Mind.* 1972.

[Chapelle et al., 2002] O. Chapelle, J. Weston, and B. Schölkopf. Cluster Kernels for Semi-Supervised Learning. 2002.

[Heisele et al., 2001] B. Heisele, T. Serre, M. Pontil and T. Poggio. Component-based Face Detection. In *Proceedings of 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2001)*, IEEE Computer Society Press, Kauai, Hawaii, Vol. 1, 657-662, December 2001.

[Jaakkola et al., 1999] T. Jaakkola, M. Meila, and T. Jebara. Maximum entropy discrimination. In *Advances in Neural Information Processing System 12*, 1999.

[Joachims, 1998] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *European Conference on Machine Learning.*

[Joachims, 1999A] T. Joachims. Making large-Scale SVM Learning Practical. Advances in Kernel Methods - Support Vector Learning, B. Schölkopf and C. Burges and A. Smola (ed.), MIT-Press, 1999.

[Joachims, 1999B] T. Joachims. Transductive inference for text classification using support vector machines. In *International Conference on Machine Learning.* Morgan Kaufmann Publishers, 1999.

[Mallat, 1989] S. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674-693, 1989.

[Mohan et al., 2001] A. Mohan, C. Papageorgiou and T. Poggio. Example-based Object Detection in Images by Components, *IEEE (PAMI)*, Vol. 23, No. 4, 349-361, April 2001.

[Morik et al., 1999] K. Morik, P. Brockhausen, and T. Joachims, *Combining statistical learning with a knowledge-based approach - A case study in intensive care monitoring.* 16th International Conference on Machine Learning, 1999.

[Ng et al., 1999] A. Ng, M. Jordan, and Y. Weiss. On Spectral Clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14*, 2001.

[Papageorgiou, 2000] C. Papageorgiou, A Trainable System for Object Detection in Images and Video Sequences. MIT AI Technical Report No. 1685, May 2000.

[Schneiderman and Kanade, 2000] H. Schneiderman and T. Kanade. A Statistical Method for 3D Object Detection Applied to Faces and Cars. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2000.

[Schohn and Cohn, 2000] G. Schohn and D. Cohn. Less is More: Active Learning with Support Vector Machines. In *International Conference on Machine Learning*. Morgan Kaufmann Publishers, 2000.

[Selinger and Nelson, 2000] A. Selinger and R. Nelson. Minimally Supervised Acquisition of 3D Recognition Models from Images. TR 724, Computer Science Dept., U. Rochester, 2000.

[Szummer and Jaakkola, 2000] M. Szummer and T. Jaakkola. Kernel expansions with unlableled examples. In *Advances in Neural Information Processing System 13*, 2000.

[Szummer and Jaakkola, 2001] M. Szummer and T. Jaakkola. Partially labeled classification with Markov random walks. In *Advances in Neural Information Processing System 14*, 2001.

[Turk and Pentland, 1991] M. Turk and A. Pentland. Eigenfaces for recognition. In *Journal of Cognitive Neuroscience*, March 1991.

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

[Viola and Jones, 2001] P. Viola and M. Jones. Robust real-time object detection. Technical report, COMPAQ Cambridge Research Laboratory, Cambridge, MA., 2001.

[Weinstein et al., 2002] E. Weinstein, P. Ho, B. Heisele, T. Poggio, K. Steele, A. Agarwal. Handheld Face Identification Technology in an Ubiquitous Computing Environment. 2002.