4-2-2004

Learning From Spatially Disjoint Data

Divya Bhadoria
University of South Florida

Learning From Spatially Disjoint Data

by

Divya Bhadoria

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science & Engineering
College of Engineering
University of South Florida

Major Professor: Lawrence O. Hall, Ph.D.
Kevin W. Bowyer, Ph.D.
Dmitry Goldgof, Ph.D.

Date of Approval:
April 2, 2004

Keywords: data mining, decision tree, nearest neighbor, distributed learning, classification

DEDICATION

To Mom and Dad.

ACKNOWLEDGEMENTS

I would like to thank my major professor Dr. Lawrence O. Hall for his constant guidance and support throughout this work. I am also thankful to Dr. Kevin W. Bowyer for his valuable suggestions and helping me understand the work better. Thanks are also due to Dr. Dmitry Goldgof for giving an interesting course in Image Processing and agreeing to be on my supervisory committee. Sincere thanks to W. Philip Kegelmeyer for his thoughtful suggestions.

I would like to extend a big thanks to my parents for always encouraging me to aim higher and strive for the best that I can. My brother Amit has been a great source of inspiration and support for me all the way. Thank you, dear brother. I would also like to thank my sis-in-law Richa for all her love. Heartfelt thanks to my fiancé Sachin, for being the eternal source of love, support and encouragement. You really make my world go round!

All my friends, back home in India and here in the USA, have been very supportive of me. Without your encouragement, I would not have come so far in life. You all are wonderful people and I am lucky to have you all as friends. This section will not be complete without thanking some special people - thank you Rahul, Farid and my cousins Manish and Deepti.

This work was partially supported by the United States Department of Energy through Sandia National Laboratories ASCI Views Data Discovery program, contract number DE-AC04-76DO00789 and the National Science Foundation under grant EIA-0130768. Portions of the research in this paper use the *FERET* database of facial images collected under the *FERET* program.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LEARNING FROM SPATIALLY DISJOINT DATA

Divya Bhadoria

ABSTRACT

Committees of classifiers, also called mixtures or ensembles of classifiers, have become popular because they have the potential to improve on the performance of a single classifier constructed from the same set of training data. Bagging and boosting are some of the better known methods of constructing a committee of classifiers. Committees of classifiers are also important because they have the potential to provide a computationally scalable approach to handling massive datasets. When the emphasis is on computationally scalable approaches to handling massive datasets, the individual classifiers are often constructed from a small faction of the total data. In this context, the ability to improve on the accuracy of a hypothetical single classifier created from all of the training data may be sacrificed.

The design of a committee of classifiers typically assumes that all of the training data is equally available to be assigned to subsets as desired, and that each subset is used to train a classifier in the committee. However, there are some important application contexts in which this assumption is not valid. In many real life situations, massive data sets are created on a distributed computer, recording the simulation of important physical processes.

Currently, experts visually browse such datasets to search for interesting events in the simulation. This sort of manual search for interesting events in massive datasets is time consuming. Therefore, one would like to construct a classifier that could automatically label the “interesting” events. The problem is that the dataset is distributed across a large number of processors in chunks that are spatially homogenous with respect to the underlying physical context in the simulation. Here, a potential solution to this problem using ensembles is explored.

CHAPTER 1

INTRODUCTION

Learning predictive models from terabytes of data may involve utilizing distributed learning across a set of processors that may be locally or globally distributed [34]. The large computation times and the amount of data entail the use of significant computational power and often equally long times for tuning the prediction model. This thesis presents methods of distributed learning that are as accurate as learning from all the data and are designed to deal with small classes of interest [33]. Not only will this be faster than and almost as accurate as learning from all the data, but it also has an additional advantage of dealing effectively with classes of interest which occur in very small numbers, which happens very often in the real world.

In particular, the work presented here can be utilized to find areas of interest in large-scale simulations. In many real life situations, massive data sets are created on a distributed computer, recording the simulation of important physical processes [1, 2, 3]. Currently, experts visually browse such datasets to search for interesting events in the simulation. This sort of manual search for interesting events in massive datasets is, of course, time-consuming. Therefore, one would like to construct a classifier that could automatically label the “interesting” events. The problem is that the dataset is distributed

across a large number of processors in chunks that are spatially homogenous with respect to the underlying physical context in the simulation.

For example, a simulation might record the deformation of a fifty-five gallon metal drum over the time sequence of some event. In this case, each processor may hold the representation of part of the drum that was in a given volume of 3D space at the outset of the simulation. Thus, the data at a given processor will generally not be representative of the data for the whole simulation. Interesting events in the simulation are generally rare. For many processors, there might be no interesting event that occurs in their part of the overall representation. The problem is to find a good way of constructing a committee of classifiers that uses the available partition of data which is natural to the underlying application, even when this partition of data is decidedly "non-random" and may present pathological distributions of training data at the individual processors. The simulations can take weeks to debug and months can be spent finding areas of interest within them. From a small amount of training data, we intend to build an Avatar, which can advise the user where to look for interesting or anomalous results. This can be used during debugging and processing.

Generally, each of potentially many processors has enough data that it is not feasible to move it to a central location for processing. The challenge is that there is a very large amount of data and very little will get labeled by the user as of interest. This means most statistical learning algorithms learn to label everything as uninteresting and thereby be correct over, say, 90% of the time. However, we are most interested in the small number of interesting examples or areas. We attempt to present a novel approach to

addressing the problem of correctly identifying a small amount of labeled data that is interesting.

1.1 Overview of the thesis

This thesis presents work on learning predictive models from spatially disjoint data. Chapter 2 gives a detailed account of the data used for the experiments along with steps involved in creating the training and test data. Chapter 3 provides the theory behind different types of classifiers with special emphasis on the classifiers used in this work. It continues with an explanation of the architecture and operation of ensembles of classifiers. Also explained in this chapter are some of the different philosophies for creating ensembles. Chapter 4 presents the experiments and results and Chapter 5 concentrates on conclusions and directions for future work.

CHAPTER 2

DATA DESCRIPTION

To study the problem in a context amenable to experimentation, we formulated a simple example based on finding regions in images of the face. The face images have been taken from the Facial Recognition Technology (FERET) database [4]. We want to label interesting regions in some training images, and then construct a classifier to automatically label those regions in similar images. This is meant to be analogous to the situation in which salient features of a simulation data set are labeled by hand, and then a classifier is constructed to automatically label data sets from similar simulation experiments. The typical approach to creating a single classifier for this problem is as follows. First, we label the different classes/regions in a set of training images. Then we use all of the available training data to create a single classifier. The learned classifier is applied to a set of manually labeled test images to estimate performance.

2.1 FERET database

The FERET program ran from 1993 through 1997 [5]. Sponsored by the Department of Defense's Counterdrug Technology Development Program [6] through the Defense Advanced Research Products Agency (DARPA), its primary mission was to develop automatic face recognition capabilities that could be employed to assist security, intelligence and law enforcement personnel in the performance of their duties.

The FERET image corpus was assembled to support government monitored testing and evaluation of face recognition algorithms using standardized tests and procedures. The final corpus consists of 14051 eight-bit grayscale images of human heads with views ranging from frontal to left and right profiles as in [5]. The images are separated into two sets: gallery images and probes images. Gallery images are images with known labels, while probe images are matched to gallery images for identification. The database is broken into five categories:

- FA: This category consists of frontal views of the faces with neutral expressions.
- FB: One or more frontal images were taken of an individual. These images were taken with some other expression, generally smile. If more than one image is taken, they are taken one after the other. One of the images is placed into the gallery file while the other is used as a probe.
- Duplicate I: The only restriction in this category is that the gallery and probe images are different. The images could have been taken on the same day or a year apart.
- FC: Images in the probe set are taken with a different camera and under different lighting than the images in the gallery set. The gallery contains the same images as the FB & Duplicate I galleries.
- Duplicate II: Images in the probe set were taken at least 1 year after the images in the gallery.

2.2 Creating test and train sets

To create the train and test sets, five faces were chosen at random from the FERET database. Then, for each face, two images were selected which differed in terms of illumination, facial expression, hair-style etc. In this way a total of ten images were selected such that each face was represented in exactly two images. One image of each face constituted the training set and the other formed the test set. In this way, we formed train and test sets of five images each such that all faces are presented in both sets. Table 1 illustrates the train and test sets with original images. All of these images were taken either from categories FA or FB of the FERET database.

2.2.1 Pre-processing

Before features could be extracted from the images, some preprocessing was required to make sure that we normalize the noise and illumination across all the images. For this purpose, the intensity histograms of all the images were normalized. This ensured that the brightest pixel in all images had an intensity level of 255 and that the darkest pixel had an intensity level of 0.

A common practice of face recognition algorithms is to apply an elliptical mask on face images to remove “disturbances” in face identification arising from variations in hair styles. Since our goal here is somewhat similar to face recognition, we also follow this practice. This is also justified by the fact that we want our classifiers to be stable under noise. In other words, we want them to base their decision on more characteristic features rather than time varying features such as hair.

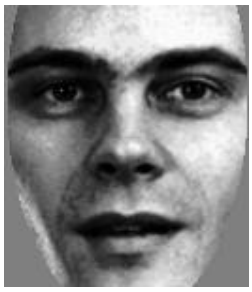
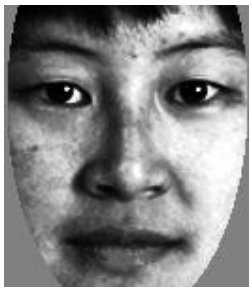
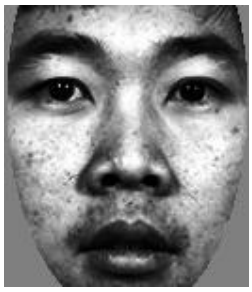
Table 1: Original train and test sets.

Sr. No.	Training Set	Test Set
1		
2		
3		
4		
5		

Another preprocessing step that the images were subjected to was the alignment of eyes at fixed pixel coordinates [49]. This ensured that the corresponding spatially homogeneous regions of the training images, which will be created during the course of the work, had somewhat similar types of data. Table 2 shows the images after pre-processing. In summation, the following pre-processing techniques were applied to all the images:

- Histogram equalization: to ensure that the maximum intensity across all images is the same.
- Eyes were automatically aligned to fixed pixel: to ensure that corresponding spatially disjoint regions have the same kind of data
- Elliptical mask: remove everything except the face. This is necessary because we want to ensure that our model is built on the actual characteristics of the image.

Table 2. Images after preprocessing

Sr. No.	Training Set	Testing Set
1		
2		
3		
4		
5		

2.2.2 Selecting classes

The next important question that had to be addressed after the initial processing of the images was how many classes should be present and how the class boundaries should be set. The decision on the number of classes was totally up to us since we are only simulating a real world situation in which data is highly skewed and distributed across different processors. Thus, as long as the basis of our problem holds, we could use any number of classes. For our study we chose to assign some level of salience to eyes, eyebrows and mouth. In this way, we get four "regions" in the faces viz. eyes, eyebrows, mouth and everything else.

Once these regions were picked, the next step was to define the boundaries for the classes they represent. There were two ways to do this. The first, and the simpler one, was to select an approximate region on images and label it. Thus, for example, a rectangular area around the mouth could be selected and labeled with the saliency associated with mouth. The other, and more realistic, approach would be to select the exact area and label it. That is, to label the mouth pixels select the exact area covering the mouth and label it. We defined class boundaries by coloring the face regions according to the level of interest. To create rectangular classes, rectangular regions could be chosen using the rectangular box tool. Exact class boundaries were drawn free hand. Any image manipulation software could be used to do this. We used a freely distributed image manipulation software called GIMP [46]. The two ways to define class boundaries are shown in Table 3. The boundaries for eyes, mouth and eyebrows are marked in red.

These are classes with which we associate some level of interest. Everything which is not marked red represents the “not interesting” class.

At the outset of experimentation stage, one obvious question that surfaced was whether we should (a) treat each of these regions as a different class, giving four classes in all, or (b) should we put more than one type of region into the same class. As explained in Chapter 4, we later combined eyes and mouth into one class, thereby reducing the number of classes by one and increasing the complexity of the problem. Table 4 illustrates the 4-class and 3-class scenarios. Green color marks the somewhat interesting regions. Interesting regions are marked with red color. Also, when 4-classes are considered, another interesting class is marked with blue. Note that although both classes are called interesting the two are considered different.

Experiments showed that using exact boundaries gave a higher accuracy than rectangular boundaries, so we decided to keep the former. Also, the accuracy with three classes was almost as good as that with four classes. However, real world situations are usually not so simple as to have clearly distinct types of examples in each class, so we decided to work with three classes (in which eyes and mouth were combined to increase complexity).

Table 3. Different ways of selecting class boundaries

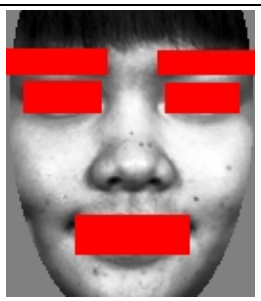
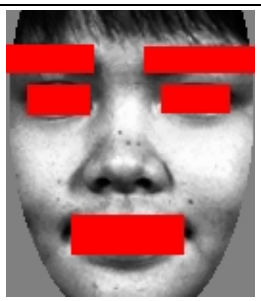
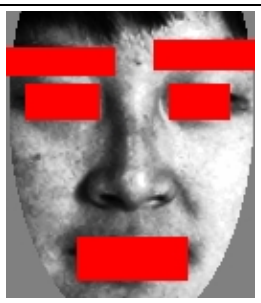
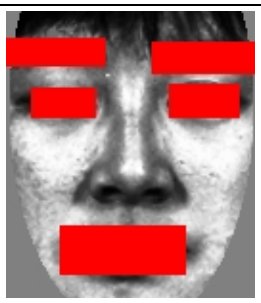
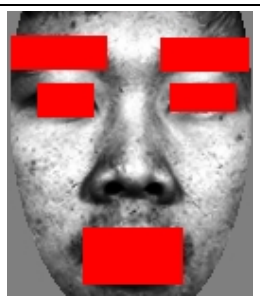
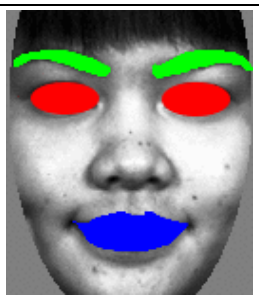
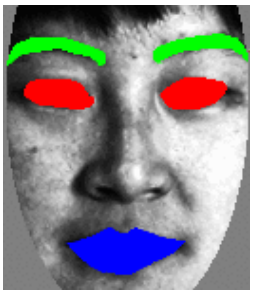
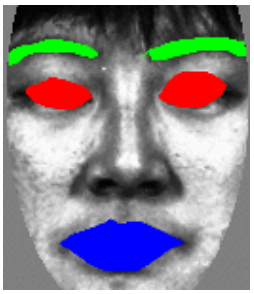
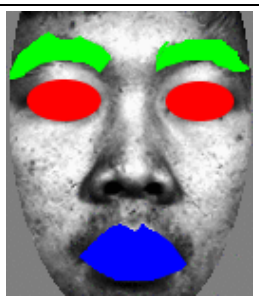
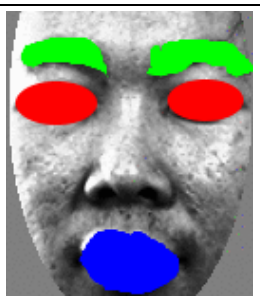
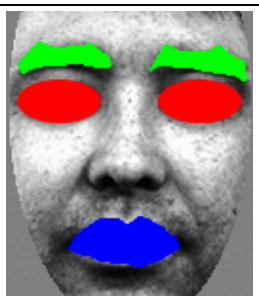
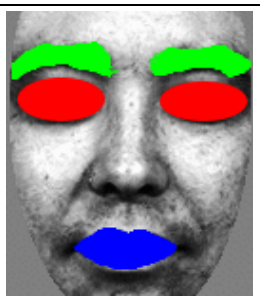
Sr. No.	Rectangular Boundaries		Exact Boundaries	
	Test	Train	Test	Train
1				
2				
3				
4				
5				

Table 4. Selecting number of classes

Sr. No.	4-class problem		3-class problem	
	Train	Test	Train	Test
1				
2				
3				
4				
5				

2.2.3 Feature extraction

Feature extraction is the process of extracting useful descriptive information from images in the form of a set of features which allow for differentiation from other images. Performing accurate measurements in images to extract the maximum descriptive information from the available data is fundamental to building good classifiers. In general, these measurements may be simple, such as the number, size, or color of objects, or more complicated, such as the shape [35], connectivity [37], or appearance (texture) [36] of objects (there are over 100 different measurements describing image texture alone). Additional measurements might first involve object extraction and then describing the spatial arrangement or distribution of objects in a scene, or the statistical distribution of properties across many objects. Sometimes it is well established beforehand which features of the image need to be measured, at other times suitable measurements are "discovered" from a large number of pre-computed possibilities. In other cases, a specific operator can be constructed to measure some particular property of the data.

In image processing, there are many classes of features and each has various techniques by which it is measured. In addition, combinations of the simpler measurements may form higher order features. For example:

- Size (area, volume, perimeter, surface) - obtained by counting pixels
- Shape - obtained by characterizing the border - Fourier descriptors, invariant moments, shape measures, skeletons, edge abruptness
- Color - description in color-space, integrated optical density, absolute and relative colors
- Appearance/texture - color variation in pixel neighborhoods - co-occurrence matrices, run lengths, fractal measures, statistical geometric features
- Parameters from fitted statistical models - used for texture, e.g. Markov Random fields - or to describe placement of objects in a scene (e.g. Poisson models)
- Distributional parameters - moments: mean, variance, skewness, kurtosis, median, inter-quartile range - used to describe statistical distributions of the more fundamental features, for example within a scene.

A feature may be a point property or a local property. A point feature is a property defined solely by the pixel of interest. A local feature, on the other hand, is defined by all the pixels over a certain neighborhood of the pixel of interest [37, 38]. To get good local features, it is necessary to choose just the right neighborhood. If the neighborhood is too small, it may not cover the defining property of the region. On the other hand, if the neighborhood is too large it may include irrelevant information.

For this study, we create the following combinations of six local and point features for our feature vectors. None of them require object detection. They are:

- Intensity value of the pixel of interest
- Maximum intensity over a fixed neighborhood around the pixel of interest
- Minimum intensity over a fixed neighborhood around the pixel of interest
- Intensity range computed as the difference between maximum and minimum intensities over the fixed neighborhood around the pixel of interest
- Arithmetic mean of intensity values over the neighborhood
- Standard deviation of intensities over the neighborhood.

For boundary pixels, local features were calculated by replicating pixels from one part of the window over to the missing part. Thus, for example, for all pixels in the left-most column of an image, we replicate immediate pixels from that column's right hand side on to its left in order to complete the window, or neighborhood, for feature extraction. To determine the best neighborhood size, local features were calculated using window sizes of 3x3, 5x5, 7x7, 11x11 pixels and then experiments were carried out to determine the best neighborhood amongst these. These experiments are described in detail in Section 4.1. The accuracy from training a single decision tree over all the training data and testing on each test set data was best for the 5x5 neighborhood. This means that 3x3 was too small to cover all the information and 7x7 and 11x11 neighborhoods were a little too big to contain just the defining properties and no misleading information.

2.3 Data example

Data sets were created in the USFC4.5 format [7]. This format makes use of a *names file* which contains all of the information about the data such as the classes, attributes and

attribute types. Attributes are listed in exactly the same order in which they occur in the data file. The attributes are as described in Section 2.2.3. The *names file* for our data is as shown in Figure 1. The comment lines start with a "|".

```
|classes:
|       0 = not interesting
|       1 = interesting
|       2 = somewhat interesting
0, 1, 2.

|attributes
Grayscale Value : continuous.
Max Value : continuous.
Min Value : continuous.
Range : continuous.
Mean : continuous.
Standard Deviation : continuous.
```

Figure 1. The *names* file for our data

A small window (14 examples) of data created from one of our face images is given in Figure 2. Note that the first six values are the six features in the order specified in the names file and the seventh value is the class label.

```
43,46,24,22,39.0,0.8,0
41,56,30,26,40.0,3.2,2
40,104,30,74,55.0,9.8,2
54,130,30,100,68.0,12.4,2
93,138,32,106,83.0,11.0,2
105,142,37,105,97.0,9.0,0
113,157,81,76,119.0,7.6,0
114,157,102,55,123.0,0.8,0
29,38,9,29,23.0,0.4,0
24,44,9,35,24.0,0.2,0
20,47,9,38,25.0,0.2,1
20,47,10,37,25.0,1.8,1
22,47,5,42,26.0,4.2,1
23,47,2,45,25.0,4.6,1
```

Figure 2. Example of data created.

2.4 Properties of data

We have created data with three classes: (1) “interesting,” representing the eyes and mouth, (2) “somewhat interesting,” representing the eyebrows, and (3) “not interesting,” representing the rest of the face. In the images that we use, approximately 84% of the pixels belong to the not interesting (NI) class, 11% of the pixels belong to the interesting (I) class, and 5% belong to the somewhat interesting class (SI). Class imbalance that is this extreme or greater is an inherent element of the application of interest. Almost by definition, only a small fraction of the data generated in a real world simulation experiment would fall in the I or SI class.

Data is represented as a feature vector for each pixel. The features included in this vector, in the order they appear in the vector from left to right, are - intensity value of the pixel of interest, maximum intensity over a fixed neighborhood around the pixel of interest, minimum intensity over the neighborhood, intensity range over the neighborhood, arithmetic mean of intensity values over the neighborhood and standard deviation of intensities over the neighborhood. The last value in the feature vector is a number representing the class assigned to that pixel.

The images obtained after preprocessing were 150 pixels (rows) by 130 pixels (columns) in size. Therefore, each image had a total of 19,500 pixels. Each pixel forms one train/test example. Thus, each of the ten images produced data consisting of 19,500 examples. Each example was described by six features. A class label was also assigned to each training example.

CHAPTER 3

CLASSIFIERS AND THEIR ENSEMBLES

Here, a classifier is a prediction model built from a Machine Learning algorithm which provides a classification given a feature vector. Some of the most popular classifiers used in Machine Learning and Data Mining are decision trees [7], clustering (e.g. K-means), neural networks [39, 40, 41], Bayesian classifiers [42, 43], Support Vector Machines [44], reinforcement learning [45] and case based classifiers [13, 14] (e.g. K-nearest neighbors). The problem of inducing general functions from specific training examples is central to learning. The concept of learning as applied to Machine Learning and Data Mining is described by Tom Mitchell, pp. 2, [8] as:

> "*Definition*: A computer program is said to *learn* from experience *E* with respect to some class of tasks *T* and performance measure *P*, if its performance at tasks in *T*, as measured by *P*, improves with experience *E*."

A classifier learns a task by forming a hypothesis that fits the target concept over the training data. It is assumed that the best hypothesis regarding unseen instances is the hypothesis that best fits the observed training data. This is called the inductive learning hypothesis by Tom Mitchell, pp. 23, [8].

> "*The inductive learning hypothesis.* Any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples."

This work uses some of these classifiers for learning the training data and making predictions on unseen test data. The following sections shed some light on the classifiers used.

3.1 Decision trees

Inductive learning identifies relationships between the attribute values of training examples and the class of the example, thus establishing a learned function or hypothesis. Decision tree learning [7, 9, 10] is one of the most widely used and practical methods for inductive inference. Decision trees are trained on examples comprised of a finite number of predictive attributes together with class labels and a learned model is established based on tests applied to these attributes. Decision tree learning approximates discrete-valued target functions, in which the learned function is represented by a decision tree. Learned trees can also be represented as sets of if-then rules to improve their readability. Decision trees are applicable to a broad range of tasks from learning to diagnose medical cases to learning to assess the credit risk of loan applicants.

3.1.1 The algorithm

A decision tree is a directed tree structure comprised of nodes. Decision trees classify instances by sorting them as one proceeds down the tree from the root to some leaf, which provides the classification of the instance. Each node in the tree specifies a test of some attribute of the instance, and each branch descending from that node corresponds to

one of the possible values (or a set of values) of this attribute. Each attribute test is strictly propositional, any boolean function can be written as a decision tree, though this is extensible to functions with a larger range of outputs. A decision tree is essentially a disjunction of conjunctions of constraints on the attribute values of instances. Each path from the tree root to a leaf corresponds to a conjunction of attribute tests, and the tree itself is a disjunction of these conjunctions. An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute in the given example. This process is then repeated for the subtree rooted at the new node. In this way a path is finally established to a leaf node, providing the classification of the instance. In the Boolean case, if the target attribute is true for the instance, it is called a *positive example*; otherwise it is called a *negative example*.

```
DTree( examples, attributes)

If all examples in one category,
then return a leaf node with this category as label
Else if attributes=Ø,
        then return a leaf node with the most common category in examples as label
Else find the "best" attribute test for node, call it A
        Assign A as decision attribute for node
        If A is a continuous attribute
                Define bins on the values of A
                Create discrete values of A using these bins
        For each value, v_i of A
                Let examples_i be the subset of examples with value v_i for A
                If examples_i is empty
                        Then create a leaf node with label most common in examples
Else call DTree( examples_i, attributes-A)
```

Figure 3. A simple decision tree algorithm.

There are different measures that can be applied to determine the best decision attribute for each node [7, 10, 11]. Continuous-valued attributes can be incorporated into the learned tree by dynamically defining new discrete valued attributes that partition the continuous attribute value into a discrete set of intervals.

Decision trees have been found to be a very useful tool in the field of Data Mining for classification. The simple structure and robustness to noisy data makes them a preferred classifier for most problems. Hunt et al [12] were the pioneers in the field of decision trees.

3.1.2 An example

Distance	Fuel	Re-fuel?
200	Low	Yes
87	Empty	Yes
65	Full	No
51	Low	No
112	Empty	Yes
200	Full	No

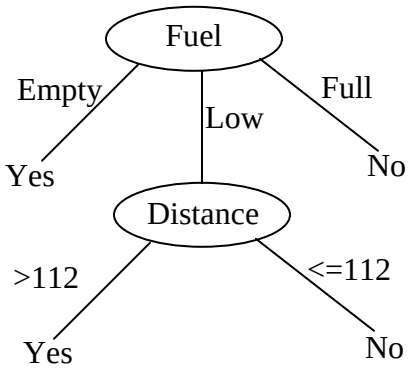


Figure 4. Constructing a decision tree

Figure 4. presents an example in which a decision tree is built from the training examples to decide on whether or not to refuel the car tank. The attribute “Distance” is continuous. The algorithm sets a threshold for this attribute at 112 and thus dynamically defines new discrete values for it. The new discrete values are <=112 (examples 2, 3, 4 and 5) and >112 (examples 1 and 6).

3.2 Instance-based learning

In contrast to learning methods that construct a general, explicit description of the target function when training examples are provided, instance based learning methods simply store the training examples [13, 14, 15]. Generalizing beyond these examples is postponed until a new instance must be classified. Each time a new query instance is encountered, its relationship to the previously stored examples is examined in order to assign a target function value for the new instance. Instance-based learning includes nearest neighbor and locally weighted regression which are conceptually straight forward approaches to estimating real-valued or discrete-valued target functions. These methods assume instances can be represented as points in a Euclidean space. These approaches also include case-based reasoning methods that are more complex, symbolic representations for instances. Instance based methods are sometimes referred to as “lazy” learning methods because they delay processing until a new instance must be classified.

A key advantage of this kind of delayed, or lazy, learning is that instead of estimating the target function once for the entire instance space, these methods can estimate it locally and differently for each new instance to be classified. In fact, many instance-based techniques construct only a local approximation to the target function that

applies in the neighborhood of the new query instance, and never construct an approximation designed to perform well over the entire instance space [8]. This has significant advantages when the target function is very complex, but can still be described by a collection of less complex local approximations. Instance-based methods can also use more complex, symbolic representations for instances. In case-based learning, instances are represented in this fashion and the process for identifying “neighboring” instances is elaborated accordingly. Case-based reasoning has been applied to tasks such as storing and reusing past experience at a help desk, reasoning about legal cases by referring to previous cases, and solving complex scheduling problems by using relevant portions of previously stored problems.

One disadvantage of instance-based approaches is that the cost of classifying new instances can be high [8]. This is because nearly all computation takes place at classification time rather than building a model or hypothesis when the training examples are first encountered and using that model every time a classification has to be made. This makes it important to have techniques for effectively indexing training examples to reduce the computation required at query time.

Another disadvantage to many instance-based approaches in general, and nearest-neighbor approaches in particular, is that they typically consider all attributes of the instances in determining the similar training examples. If only a few of the many available attributes determine the target function, then the instances that are actually similar may well be computed as very dissimilar.

3.3 K-nearest neighbors

The K-nearest neighbor algorithm [16, 17] is the most basic instance-based learning method. This algorithm assumes all instances correspond to points in the *n*-dimensional space R^n. The nearest neighbors of an instance are defined in terms of the standard Euclidean distance. So if instance *x* is described by the feature vector,

$$\langle a_1(x), a_2(x), \ldots, a_n(x) \rangle$$

where $a_r(x)$ denotes the value of the r^{th} attribute of instance *x*, then the Euclidean distance between two instances x_i and x_j is defined to be $d(x_i, x_j)$, where,

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

In nearest neighbor learning the target function may be either discrete-valued or real-valued. The following section sheds some light on the working of a nearest neighbor algorithm for discrete-valued target functions.

3.3.1 Discrete-valued target functions

3.3.1.1 The algorithm

Let the target function be of the form $f : R^n \rightarrow V$, where *V* is the finite set $\{v_1, \ldots, v_s\}$. The *k*-nearest neighbor algorithm for approximating this target function is given in Figure 5.

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, add the example to the list *training_examples*

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1, \ldots, x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$$

where $\delta(a,b) = 1$ if $a = b$, otherwise it is equal to zero.

Figure 5. K-nearest neighbor algorithm for discrete-valued target functions [8]

The value $\hat{f}(x_q)$ returned by this algorithm as its estimate of $f(x_q)$ is just the most common value of f among the k training examples nearest to x_q.

3.3.1.2 An example

Figure 6 illustrates an example in which $V = \{a, b, c\}$. The value of k = 5, that is five nearest neighbors are considered in classifying the query instance x_q.

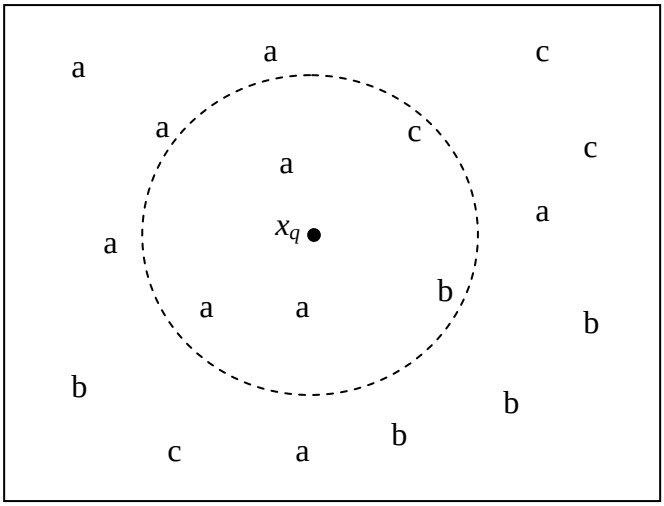


Figure 6. An example of *K*-nearest neighbor on discrete-valued target function

Out of the five nearest neighbors, three belong to class *a* and one each to classes *b* and *c*. Therefore, $\hat{f}(x_q) = a$.

3.4 Ensemble of classifiers

Committees of classifiers, also called mixtures or ensembles, have become popular because they have the potential to improve on the accuracy of a single classifier constructed from the same set of training data. Bagging [18, 19] and boosting [23, 24] are some of the better known methods of constructing a committee of classifiers [21]. Committees of classifiers are also important because they have the potential to provide a computationally scalable approach to handling massive datasets [20]. When the emphasis is on computationally scalable approaches for handling massive datasets, the individual classifiers are often constructed from a small fraction of the total data. In this context, the ability to improve on the accuracy of a hypothetical single classifier created from all of the training data may be sacrificed.

In an ensemble based approach each member is complete in itself, independently capable of a prediction on the entire problem space. The building of the classifiers is manipulated such that different classifiers generate potentially different predictions for the same example. The ensemble prediction is obtained by combining these individual predictions into one prediction. This is usually done by voting the individual predictions, either with or without some weights attached to each of them. Figure 7 shows the architecture and prediction production method for an ensemble.

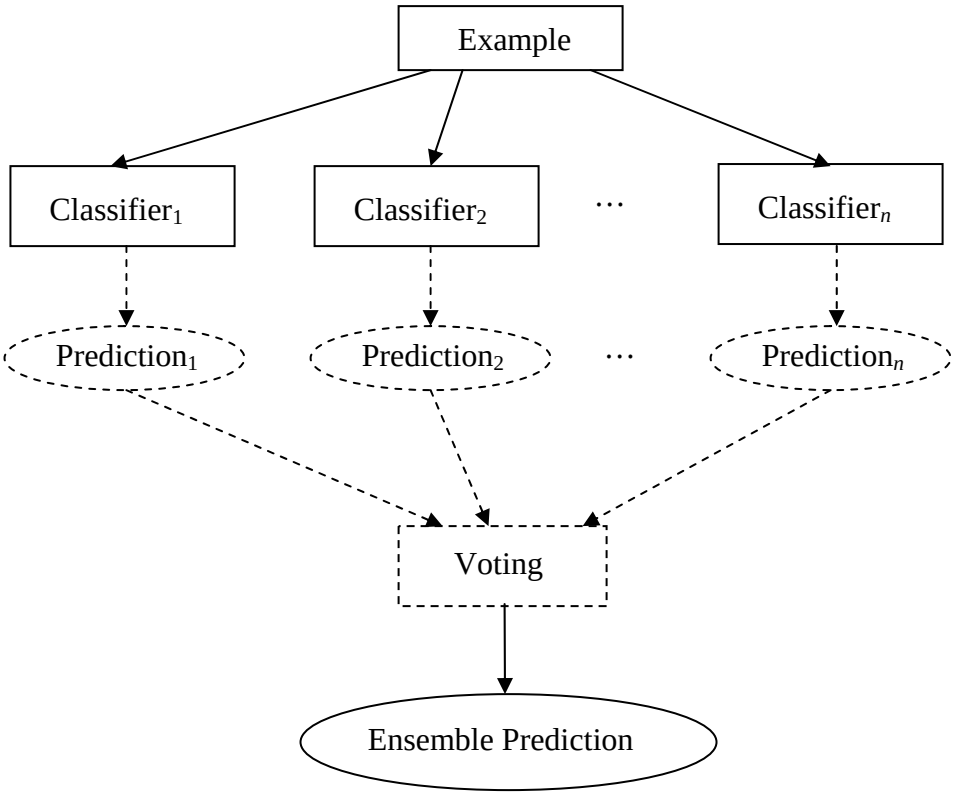


Figure 7. Ensemble architecture and prediction process.

Different decision boundaries amongst classifiers are produced by having some non-deterministic choices in the learning stage. Classifiers learned on different data will often learn (slightly to very) different concepts. Thus, by training each classifier on a randomly altered variation of the training set, different decision boundaries can be obtained. So an ensemble can be viewed as a more complex "classifier" with much more complex decision boundaries and thus capable of making more accurate predictions than a single classifier of the same type.

3.4.1 General methods for creating ensembles

Different philosophies of creating ensembles are discussed in the following [22]. One way of creating different classifiers is to modify the training data for the learning

algorithm. This can cause an unstable classifier to produce very different results. An unstable classifier is one which small changes in training data can result in very big changes in the decision boundaries [21]. Decision trees are an example of an unstable classifier. A stable classifier, like nearest neighbors, is not much affected by a small change in the training data i.e. these classifiers are more robust to noise in the training data. The other way of creating different classifiers is to modify the learning algorithm itself. This approach is more popular with decision trees. Ensemble creation approaches can be either sequential or distributed.

3.4.1.1 Sequential learning

In a sequential learning approach, a new classifier is built only after the existing classifier has been trained and evaluated. This approach is typically slower than its distributed counterpart. The advantage here is that the error of the existing classifier can be evaluated before building the next classifier. This information can then be used in intelligently creating training sets for new classifiers so as to train them on misclassified regions of the problem space. Thus together, the classifiers will be able to give better predictions on the entire problem set. Examples of this approach are boosting [23, 24] and Ivoting [32]

3.4.1.2 Distributed learning

In a distributed learning approach, classifiers are created in parallel on a large number of processors. This makes distributed learning faster than the sequential learning approach. The distributed approach also makes it possible to produce effective learning algorithms on large scale computing machines such as the ASCI Red [25], Blue [26, 27] and White [28] supercomputers created for simulating nuclear explosions [1, 2, 3] by the United

States' Department of Energy's Advanced Strategic Computing Initiative (ASCI). Examples of this approach are bagging [18], random subspaces [29, 30] and disjoint partitioning [31].

The work presented in this thesis centers around this kind of learning approach. We are interested in learning predictive models from terabytes of data utilizing distributed learning across a set of processors that may be locally or globally distributed.

CHAPTER 4

EXPERIMENTS AND RESULTS

The work presented in this thesis is on a new topic with no obvious previous study to refer to. Whenever a decision had to be made, be it about selecting the right neighborhood for feature extraction or defining class boundaries, experiments were carried out with all the choices to determine the best one. This chapter describes the experiments along with results and conclusions, where applicable, that were obtained at each step.

4.1 Feature extraction

The very first decision point we faced was to determine the best neighborhood size to calculate the local features for our data. Recall that a local feature is defined by all the pixels over a certain neighborhood of the pixel of interest. To get good local features, it is necessary to choose just the right neighborhood. If the neighborhood is too small, it may not cover the defining property of the region. On the other hand, if the neighborhood is too large it may include irrelevant information.

4.1.1 Experiments

To determine the best neighborhood size, local features were calculated using window sizes of 3x3, 5x5, 7x7 and 11x11 pixels. Then experiments were carried out to determine the best neighborhood amongst these. A single decision tree was trained on all the five

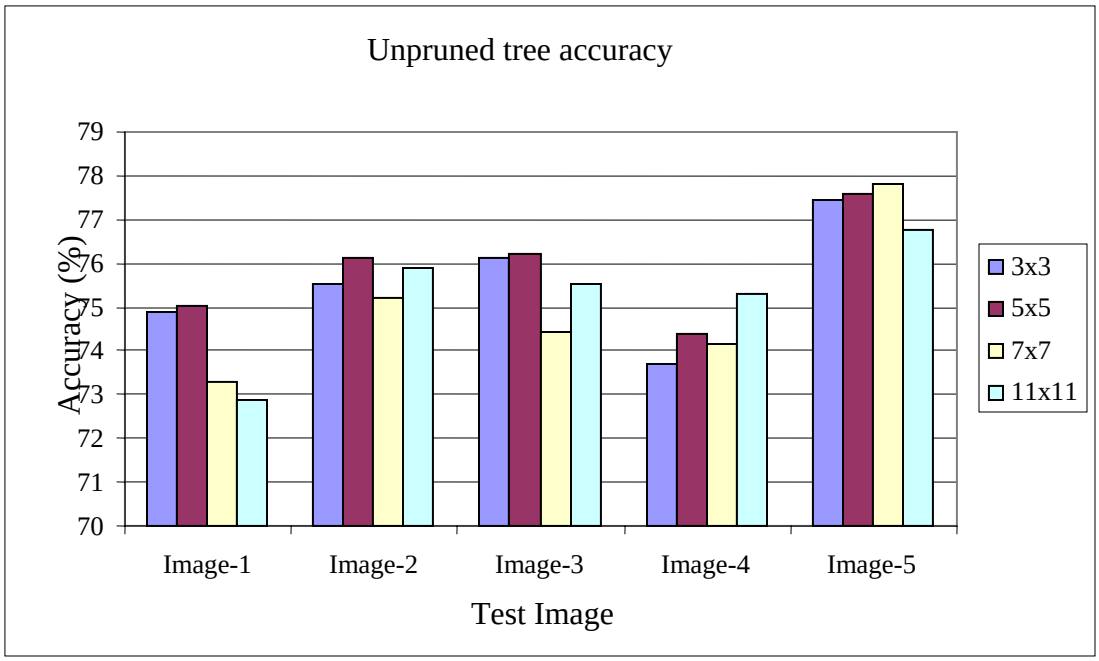
training images and tested on each of the five test images. Since this was only a preliminary stage to find good features, the number of classes and how their boundaries were defined was not considered important as long as all data was created using the same number of classes and the same shape of class boundaries. The data used in this experiment had 3 rectangular classes as described in Chapter 2. To recall, an approximate region on the images was selected and labeled. Thus, for example, a rectangular area around the mouth was be selected and labeled with the saliency associated with mouth.

4.1.2 Results

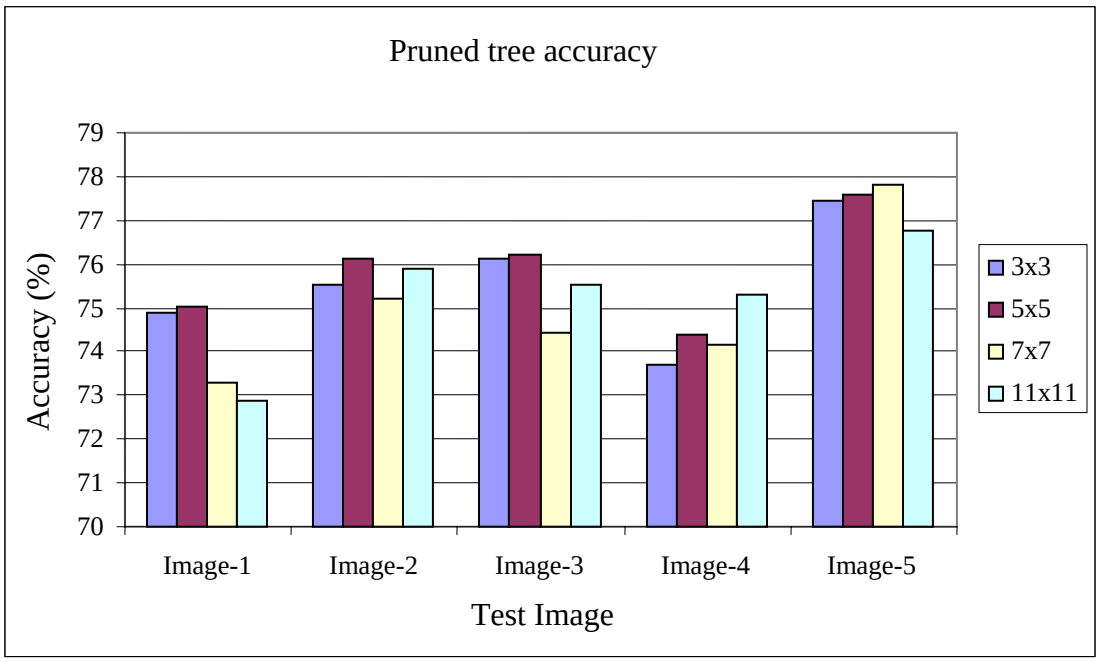
Figure 8 presents the results of this experiment. The X-axis represents the test image and the Y-axis gives the overall percentage accuracy at the pixel level on these images. For each test image, four bars have been plotted, one for each of the four neighborhood sizes used in calculating the features. Figure 8(a) gives the results obtained from the unpruned tree and Figure 8(b) gives the results for a default pruned tree. In both cases the accuracy is maximum the most number of times for a 5x5 neighborhood. It is best three times (image-1, image-2 and image-3) and the other two times it is second best (image-3, image-4).

Tables 5 and 6 show how each pixel in the five test images was classified by the decision trees. A red pixel indicates that the pixel was classified as interesting (I), a green color indicates that the pixel was classified as somewhat interesting (SI) and a blue pixel is one which was classified as not interesting (NI). Note that both eyes and mouth together make the interesting class and eyebrows make the somewhat interesting class. So ideally, pixels covering the eyes and mouth should have been colored red and those

covering the eyebrow region should have been green. All other pixels should have been blue. As can be seen in the tables, the nostrils tend to be I which is not correct as the true saliency of nostrils is NI. Also, the eyebrows are not detected well and the green color is distributed around eyes, eyebrows, mouth and nostrils. Tables 7 and 8 show whether or not the prediction was correct. Note that in Tables 5 and 6, the actual eyebrow pixels are either red or green. All other pixels in the rectangular region we labeled as SI are predicted as NI (blue color). That is, all these pixels are classified differently than what we would expect with rectangular classes. This explains the black boxes around eyebrows in Tables 7 and 8. In general, this analogy holds for all black “boxes” in Tables 7 and 8.



(a)



(b)

Figure 8. Selecting neighborhood size. Different neighborhood sizes were used to create features and a single tree was built on all training data and tested on each test image. (a) performance of unpruned tree, (b) performance of pruned tree.

Table 5. Unpruned tree prediction for different neighborhood sizes. Red =interesting, green = somewhat interesting and blue = interesting pixel.

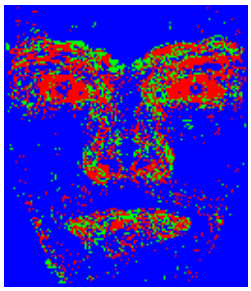
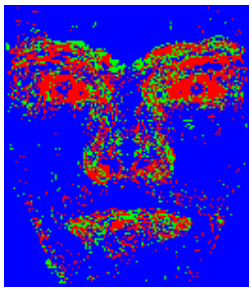
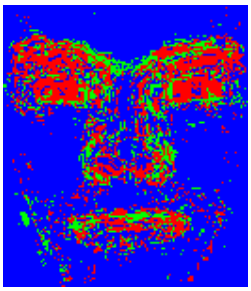
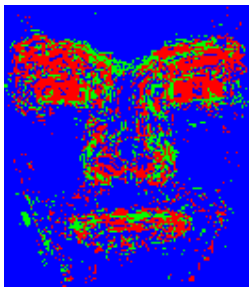
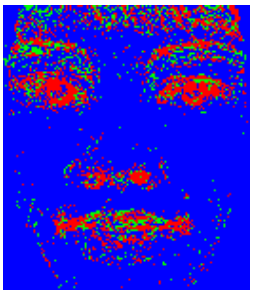
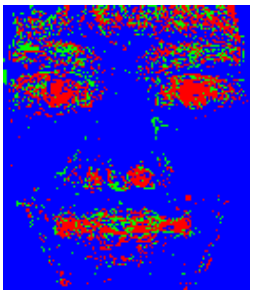
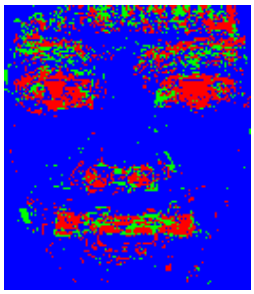
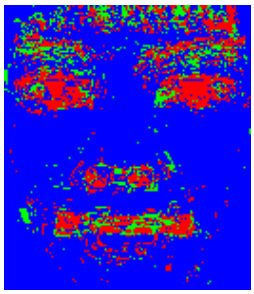
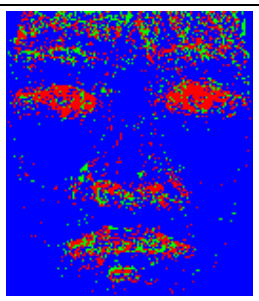
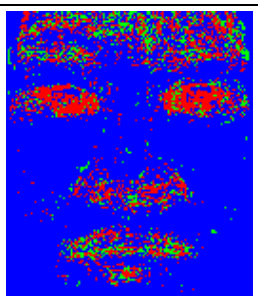
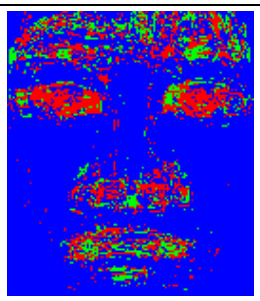
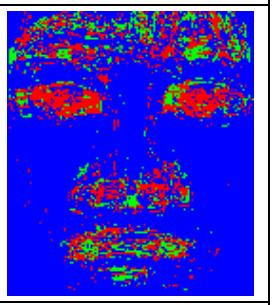
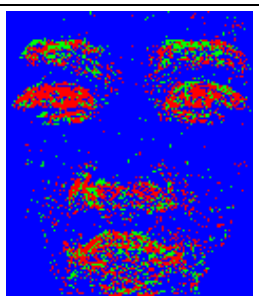
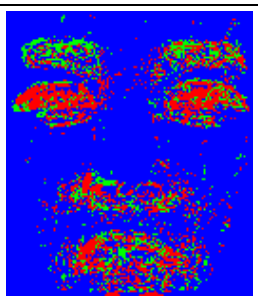
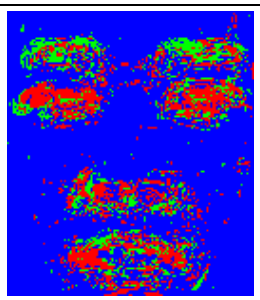
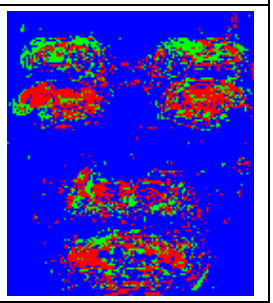
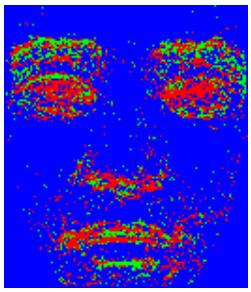
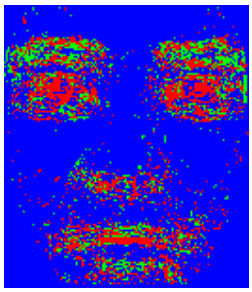
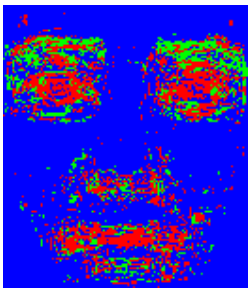
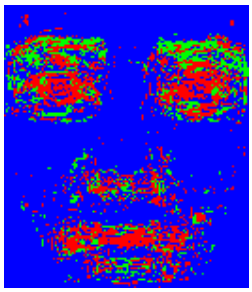
No.	3x3 window	5x5 window	7x7 window	11x11 window
1				
2				
3				
4				
5				

Table 6. Pruned tree prediction for different neighborhood sizes. Red denotes a pixel classified as interesting, green denotes a pixel classified as somewhat interesting and blue represents a not interesting pixel

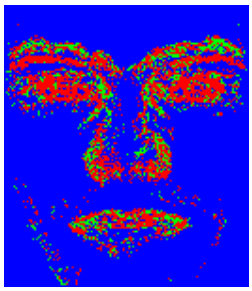
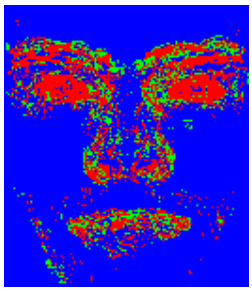
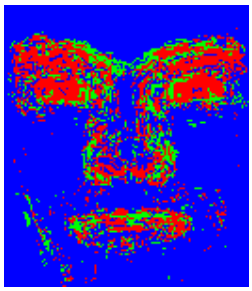
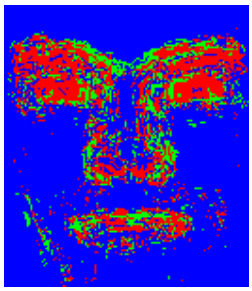
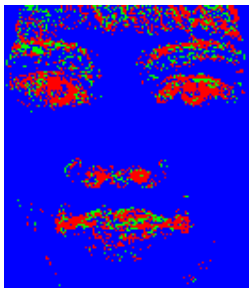
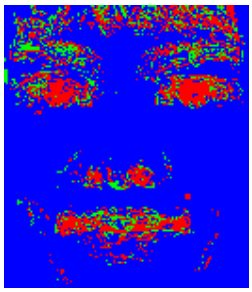
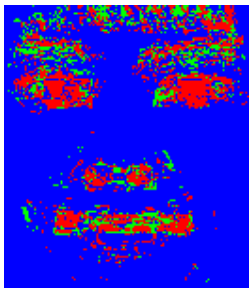
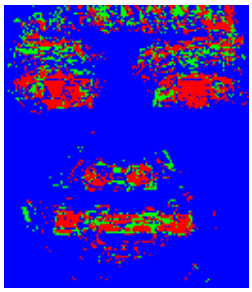
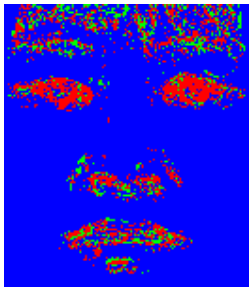
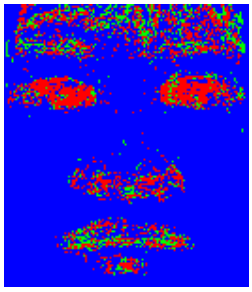
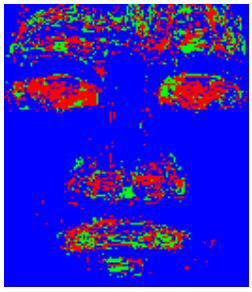
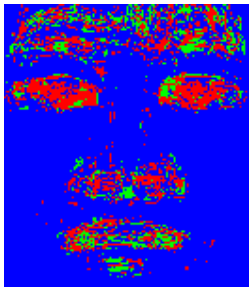
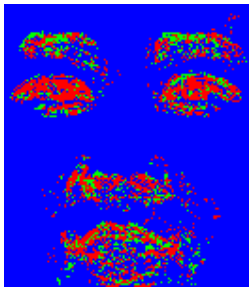
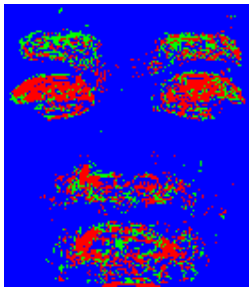
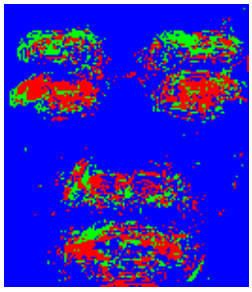
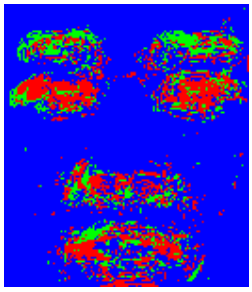
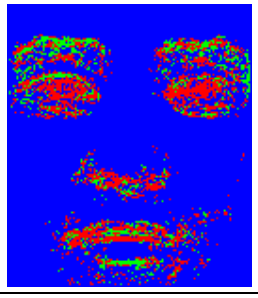
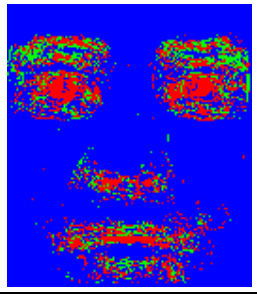
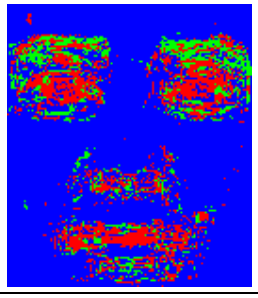
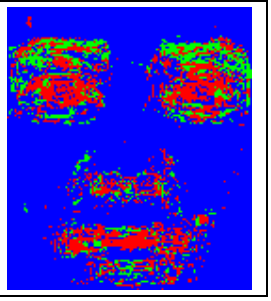
No.	3x3 window	5x5 window	7x7 window	11x11 window
1				
2				
3				
4				
5				

Table 7. Correctness of classification of an unpruned tree for features obtained with different neighborhood sizes. White = correctly classified pixel, black = incorrectly classified pixel.

No.	3x3 window	5x5 window	7x7 window	11x11 window
1				
2				
3				
4				
5				

Table 8. Correctness of classification of a pruned tree for features obtained with different neighborhood sizes. White = correctly classified pixel, black = incorrectly classified pixel.

No.	3x3 window	5x5 window	7x7 window	11x11 window
1				
2				
3				
4				
5				

4.1.3 Conclusions

The accuracy from training a single decision tree over all the train data and testing on each test data was best for the 5x5 neighborhood. This means that 3x3 was too small to cover all the information and 7x7 and 11x11 neighborhoods were a little too big to contain just the defining properties and no misleading information. So, local features for this work will be calculated using a 5x5 neighborhood size.

4.2 Classes boundaries

For this work eye, eyebrows and mouth were assigned to interesting and everything else was assigned to the not interesting class. Feature vectors associated with the interesting regions were labeled with the level of interest associated with them. It was necessary to define exactly which pixels around, say, the eyes will be considered as the eye class. Two obvious ways of doing this are (a) using an approximation – that is, select an approximate area like a rectangle around the eyes, or (b) using an exact description – that is, select only the pixels actually a part of the eye. While the first approach is simpler and saves considerable labor when very large datasets come into the picture, the second approach appears more reasonable. Experiments were carried out using the two ways of labeling pixels to find out whether or not using approximate class boundaries is acceptable.

4.2.1 Experiments

Train and test sets were created using each of the two ways of describing classes. To create the data, first, the face regions were colored according to level of interest. Then the original gray level image was taken and the feature vectors for each pixel in row first order were calculated. At the same time, the "color coded image" was being read for the

corresponding pixel. The color of the pixel in this image gave the class label. In other words, two images were being read at the same time for the same pixel. The first was a gray level image which was used to calculate the features and another was a "color" image, which was used to determine the class of this pixel. Any image manipulation software could be used for coloring the face regions according to its saliency. For experiments with rectangular classes, rectangular regions could be chosen using the rectangular box tool. For experiments with exact classes, regions could be chosen freehand. In this work GIMP [46] was used. The GIMP is the GNU Image Manipulation Program. It is a freely distributed piece of software suitable for such tasks as photo retouching, image composition and image authoring.

A single decision tree was built on all training data obtained using rectangular class shapes and then was tested on each of the test images in which the class boundaries were defined in the same way, that is using rectangular regions. The same training and testing procedure was done with data created using exact class boundaries. The two trees were pruned with the default certainty factor and the pruned trees were also tested on the respective test sets. The tree predictions for both pruned and unpruned trees were plotted as an image to see exactly how the tree is classifying the image pixels. Images were also plotted to see where the incorrect predictions were being made by the two trees.

4.2.2 Results

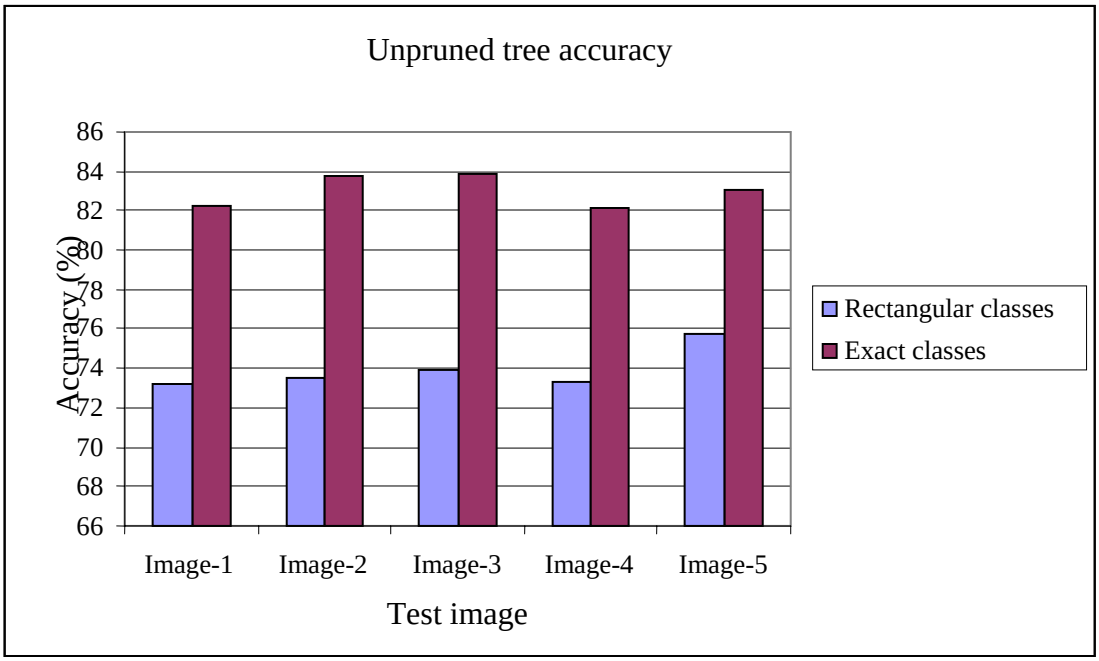
Figure 9 shows how the two decision trees performed on the test data. The decision tree built on data created using exact class boundaries is clearly much more accurate than the one built on data created using approximate (rectangular) classes. Table 9 shows how

each test pixel was classified by the two decision trees. In this table results from approximate class boundaries are given in the first two columns and those from exact class boundaries are given in the last two columns. A red pixel indicates that the pixel was classified as interesting; a green color indicates that the pixel was classified as somewhat interesting and a blue pixel is one which is classified as not interesting. Note that both eyes and mouth together make the interesting class and eyebrows make the not interesting class. So ideally, pixels covering the eyes and mouth should have been colored red and those covering the eyebrow region should have been green. All other pixels should have been blue. For both pruned and unpruned trees, there are many less red pixels and more blue pixels around the nostrils in the last two columns which give the results with exact classes. This is good because the nostrils are NI and so they should have been blue. The same holds for the chin, jaw line and forehead. They should all be colored blue and the exact class training set allows this better than rectangular classes. In general, there are less red pixels scattered around the face in the last two columns. This is desirable. Finally, Table 10 tells us which pixels were correctly classified and which were not in the two cases. A correctly classified pixel is white and an incorrectly classified pixel is black. Once again, there are less black pixels scattered around the faces in the last two columns which give the results with exact classes. It is also interesting that the hair on forehead in faces 2 and 3 is better classified with exact classes; rectangular classes tend to confuse these hair with other classes.

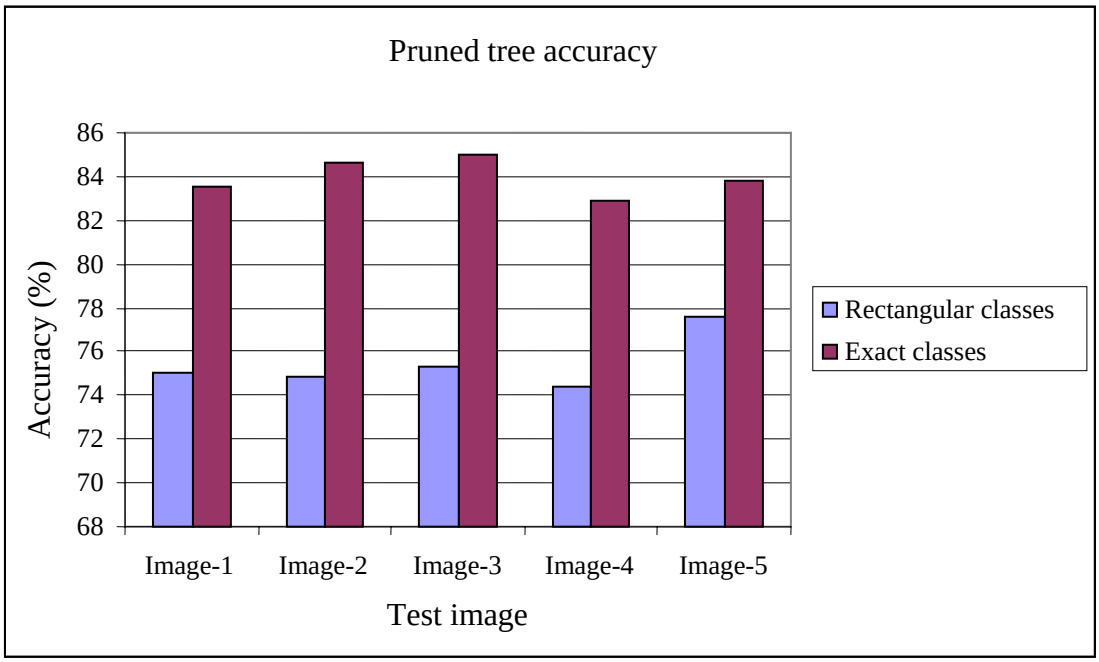
4.2.3 Conclusions

It can be seen in Figure 9 that using exact classes we can obtain unpruned trees with up to 83.84% accuracy on the test set and pruned trees with about 85.01% accuracy. On the other hand, the maximum accuracy obtained from the unpruned tree in the other case is about 75.71% and that from the pruned tree is 77.59%. Cleary the two trees are significantly different and so it can be concluded that information is better represented using exact class boundaries.

Comparing the results presented in Table 9 for the two cases, once can easily observe that there are a lesser number of red and green dots in the blue area in the last two columns which means that with exact classes it is easier to distinguish the not interesting class from the other two classes. There are also fewer green pixels around the eyes and mouth, which should have been all red, in the last two columns meaning that the classifier is better able to distinguish between the somewhat interesting and interesting classes. Also, Table 10 has less black pixels in the last two columns. This shows that the overall prediction is more accurate using exact class boundaries. Also, the interesting and somewhat interesting regions (eyes, mouth and eyebrows) have more white pixels in the last two columns; thus the trees are more accurate on the salient regions of data. This gives us yet another reason to keep exact class boundaries for all further experiments.



(a)



(b)

Figure 9. Rectangular vs. exact class boundary (a) unpruned tree accuracy (b) pruned tree accuracy

Table 9. Tree predictions for rectangular and exact class boundaries

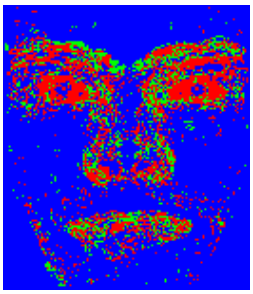
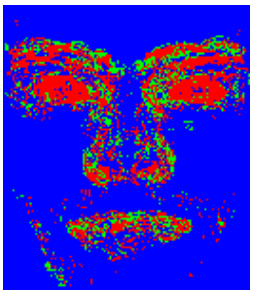
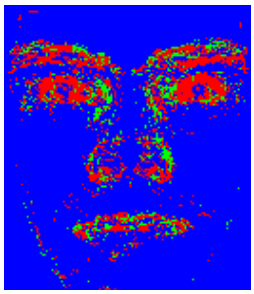
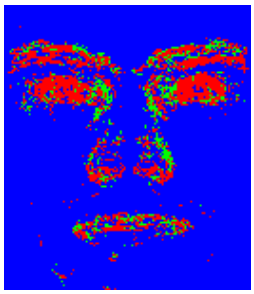
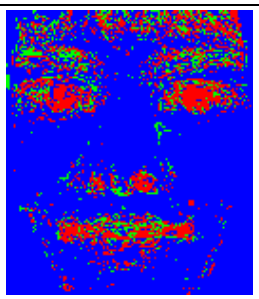
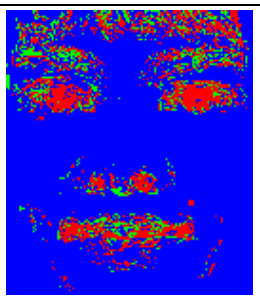
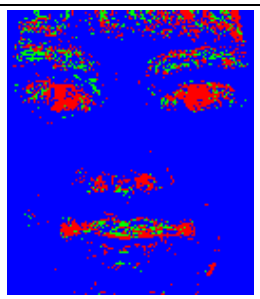
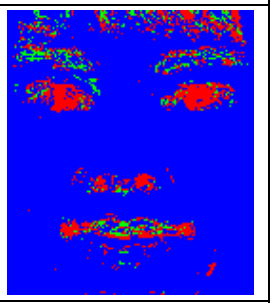
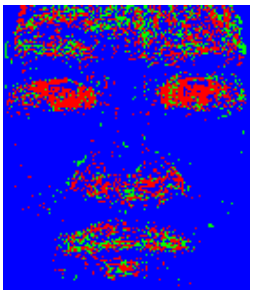
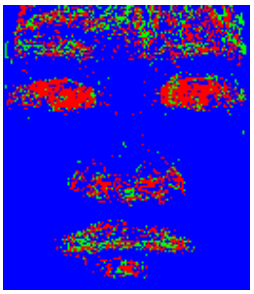
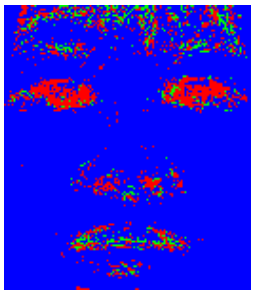
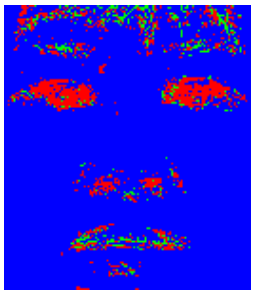
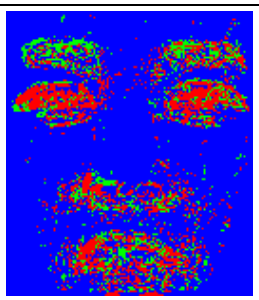
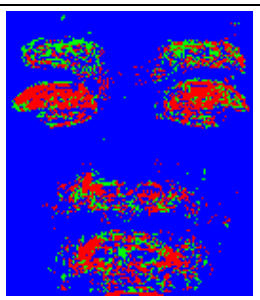
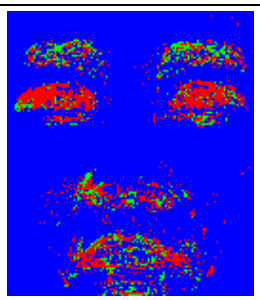
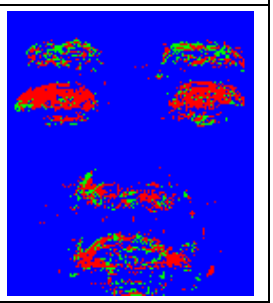
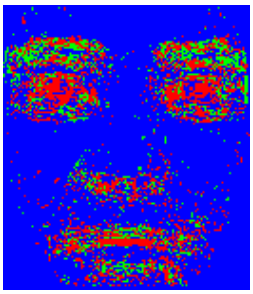
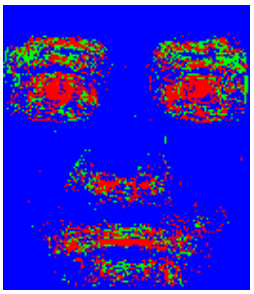
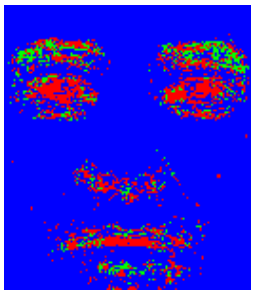
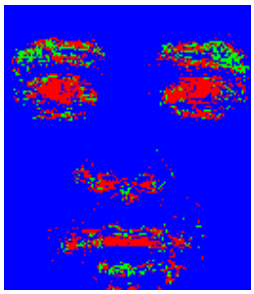
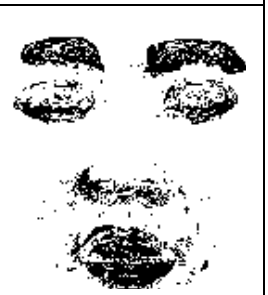
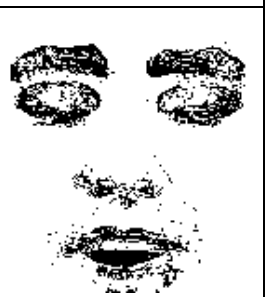
No.	Rectangular classes		Exact classes	
	Unpruned	Pruned	Unpruned	Pruned
1				
2				
3				
4				
5				

Table 10. Correctness of tree classification for rectangular and exact class boundaries. Black = incorrectly classified pixel, white = correctly classified pixel.

No.	Rectangular classes		Exact classes	
	Unpruned	Pruned	Unpruned	Pruned
1				
2				
3				
4				
5				

4.3 Number of classes

We segmented our face data set into four regions – eyes, mouth, eyebrows and everything else. One obvious question that comes to mind is how to assign classes to these regions. Should every region be assigned to a different class, resulting in a total of four classes, or should two, or more, of them be assigned to the same class, resulting in less than four classes and more a difficult classification problem.

Real world situations are usually not so simple as to have clearly distinct types of examples in each class. Very often two or more events or properties are equally important in a simulation. There is also usually more than one important event in a simulation, although their relative significance may not be the same. Thus it makes this work more relevant to real world situations by keeping more than one type of example (facial features) in a class and having different salience levels of classes. Thus, we can put, say, both eyes and mouth into one class – interesting – and eyebrows may be called somewhat interesting. Note that although eyes, eyebrows and mouth are all "regions of interest", we assign lesser importance to eyebrows.

4.3.1 Experiments

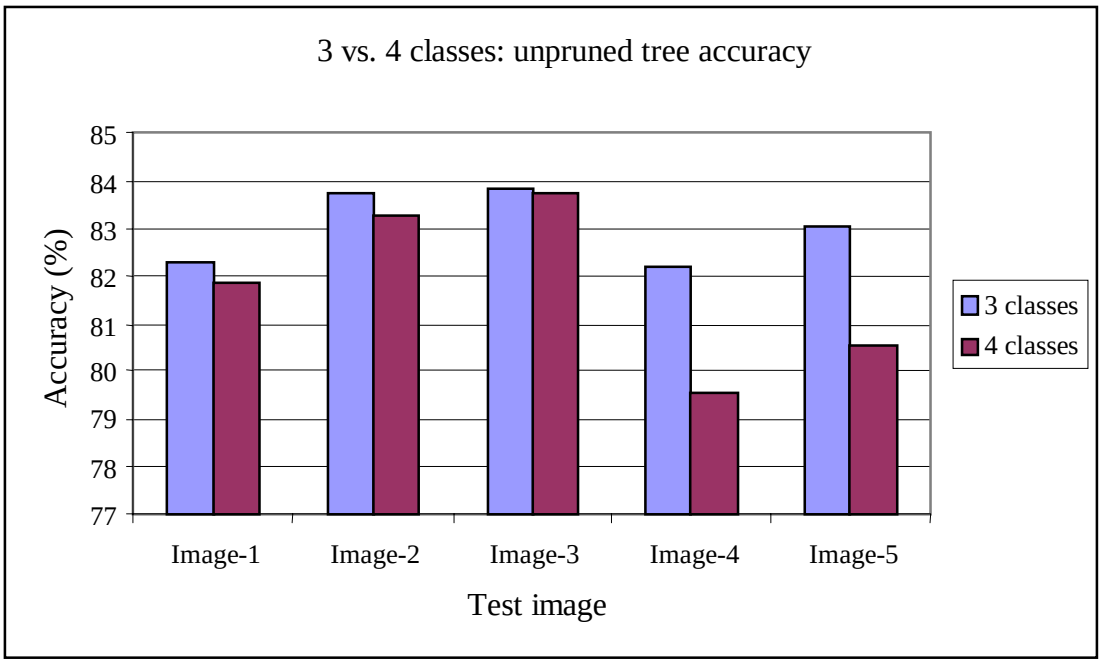
Experiments were carried out to see how models built on a three-class problem compare to those built on a four-class problem. Train and test sets were created for both a three-class problem and a four-class problem. For both types of data, a decision tree was built on the train set. Pruned trees were obtained using a default pruning factor. The pruned and unpruned decision trees were then tested on their respective test images.

4.3.2 Results

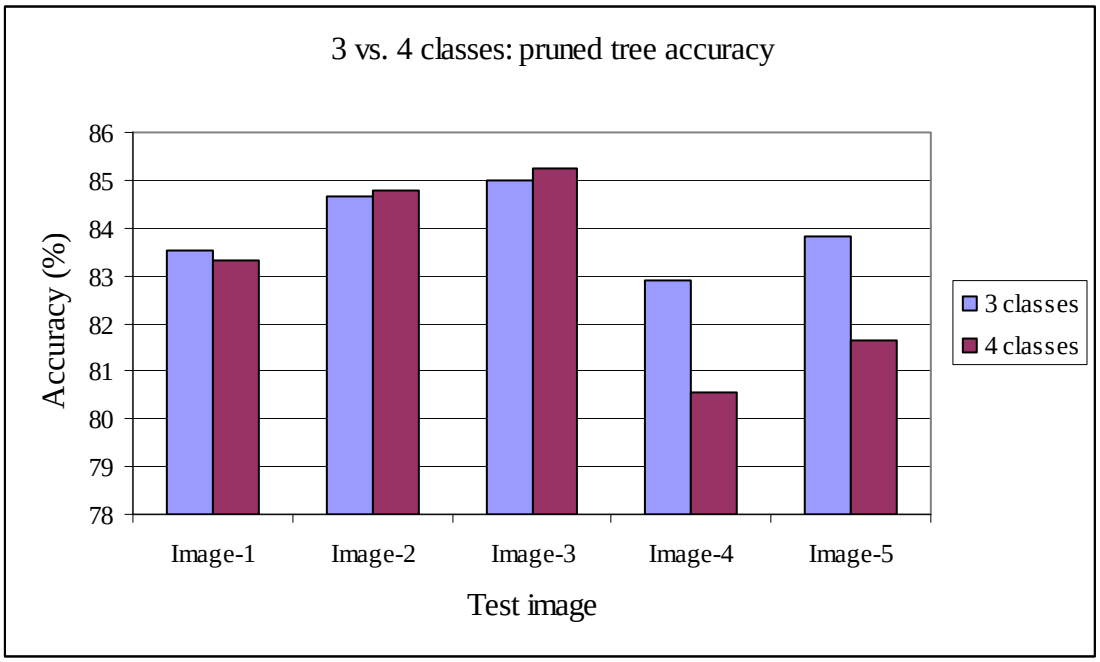
Figure 10 and Table 11 give the results of the experiments mentioned in the preceding section. The overall accuracy of unpruned and pruned trees for the two cases are compared in Figures 10(a) and 10(b) respectively. The unpruned tree with 3-classes is always more accurate than the unpruned tree with 4-classes. The pruned tree with 3-classes is better three times. It is out-performed by the pruned tree with 4-classes on test images 2 and 3. Table 11 gives the correctness of classification of each test pixel by the unpruned and pruned trees in both cases. The first two columns give the results for the unpruned and pruned trees with 3 classes and the last two columns give the results for the unpruned and pruned trees with 4 classes. Clearly there are less black pixels in the first two columns. In all the test images, including test images 2 and 3 which are more accurately predicted by 4-class pruned tree, there are significantly less black pixels over the regions of interest, viz. eyes, mouth and eyebrows. The high overall accuracy of the pruned tree with 4-classes over test images 2 and 3 is not because of better classification of areas of interest, but because the 4-class pruned tree did a better job of correctly classifying the hair on the forehead, which is not interesting for our problem. Probably, adding location as a feature would help. The actual (x, y) co-ordinates may not be very helpful since the same pixel position may refer to different saliency regions in different faces. Rather, using a general location feature, such as top-left quadrant, will be more useful because corresponding quadrants of different face images are very likely to contain same type of data.

4.3.3 Conclusions

Clearly the choice here is between defining our problem in a way which is closer to real world cases and also helps in building more accurate classifiers and another way which is farther away from real world problems and also results in classifiers which, although sometimes more accurate overall, are less accurate on regions we perceive as interesting. The former becomes the obvious choice for all work. Hence for all work from this point on, we keep three classes – interesting (eyes and mouth), somewhat interesting (eyebrows) and not interesting (everything else).



(a)



(b)

Figure 10. 3-classes vs. 4-classes (a) unpruned decision tree accuracy (b) pruned decision tree accuracy

Table 11. Correctness of decision tree classification. A correctly classified pixel is white and incorrectly classified pixel is black.

No.	3 classes		4 classes	
	Unpruned	Pruned	Unpruned	Pruned
1				
2				
3				
4				
5				

4.4 Baseline experiments

We use the C4.5 decision tree software [7] to create a 3-class classifier that will classify a feature vector as corresponding to the most interesting, somewhat interesting, or not interesting class. C4.5 has a "certainty factor" parameter that can be used to control the degree of pruning in the decision tree. The correct pixel classification rate on the test data varies only slightly with the degree of pruning, going from approximately 84% at a certainty factor of 100 (no pruning) to 85.7% at a certainty factor of 1 (heavy pruning) as illustrated in Figure 11. So the baseline performance reference point for a single well-pruned decision tree is 85.7%.

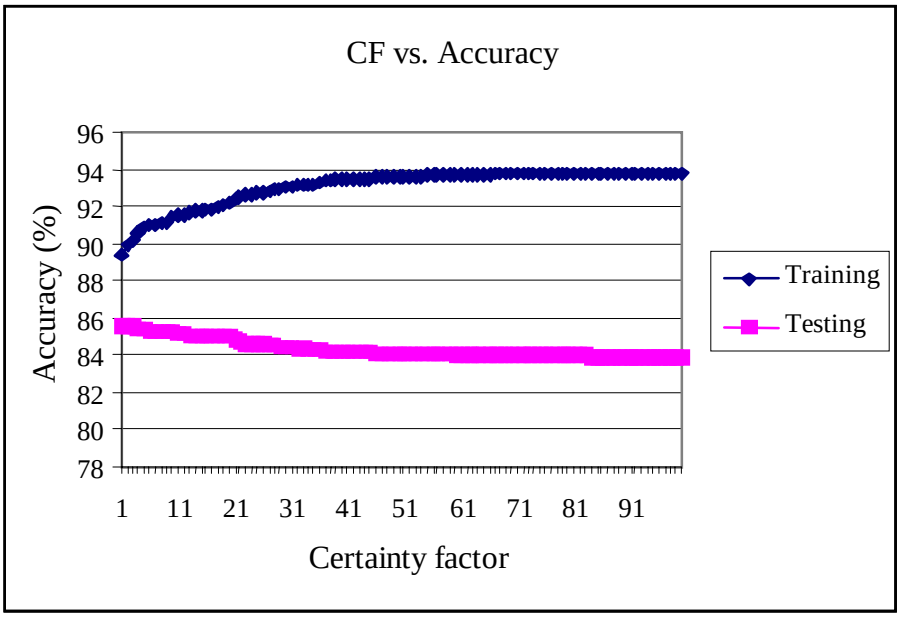


Figure 11. Change in accuracy with pruning

This appears to be a good classification rate, but if we recall that about 84% of the training (as well as test) data is made up of the NI class, even if the classifier only learns to classify everything as NI, it will still be about 84% accurate. Moreover, because the

training data had a severe minority of I and SI examples, it is very likely that the decision tree will not be able to learn the two classes well and will end up predicting most of the test pixels as NI.

4.5 Committee approaches

Now consider what happens in the type of application context that we are concerned with. The training data is divided into partitions that are spatially homogeneous with respect to the data. Each face image might be divided into eight equal-size rectangular regions, by cutting the image into two columns and four rows. Thus we get 5 x 8 = 40 spatial regions. Note that some partitions may have a pathological distribution of training data. In our example training data, 4 of these spatial regions have purely NI data, 27 have some I data, 12 have some SI data, and only 3 partitions have all three classes of data. Each spatial region might reside on a different processor. Randomly sampling across these regions could be prohibitively expensive. The problem is to design a committee approach that achieves the best performance possible, given that individual classifiers are built on the “arbitrary” subsets that result from the spatial partition of the dataset. Simply voting all classifiers, one from each partition, is a poor combination method in this context.

4.5.1 Decision trees

4.5.1.1 Bagging

A committee of 40 DTs was created by bagging on data from all five training images, with each bag equal to 1/8 of one image size (around 2437 examples), giving a total of 40 bags - which is the same number of bags as there were classifiers in the earlier

experiments. In an ideal case, each example would be seen once and only one. Recall that the highest performance by a DT alone was 85.7% (Section 4.4, Figure 11). This number came when a DT was trained on all five of the training images and it was pruned with certainty factor between 1 and 0.2. Testing was done over all the test data taken together. The motivation of the bagging experiments was that perhaps we could get a performance higher than 85.7%. The opposite was also possible given that it was very likely that not all examples may be picked in bagging, so the decision trees may not get 'enough' training data. Whatever the outcome, this would set yet another standard for the experiments that we are doing. An ensemble of 40 decision trees was created using bags of 2.5% size of training data. The trees were pruned for a range of CFs (1…100) and tested. This was done to make sure that we get the best possible accuracy. The results are presented in Figure 12.

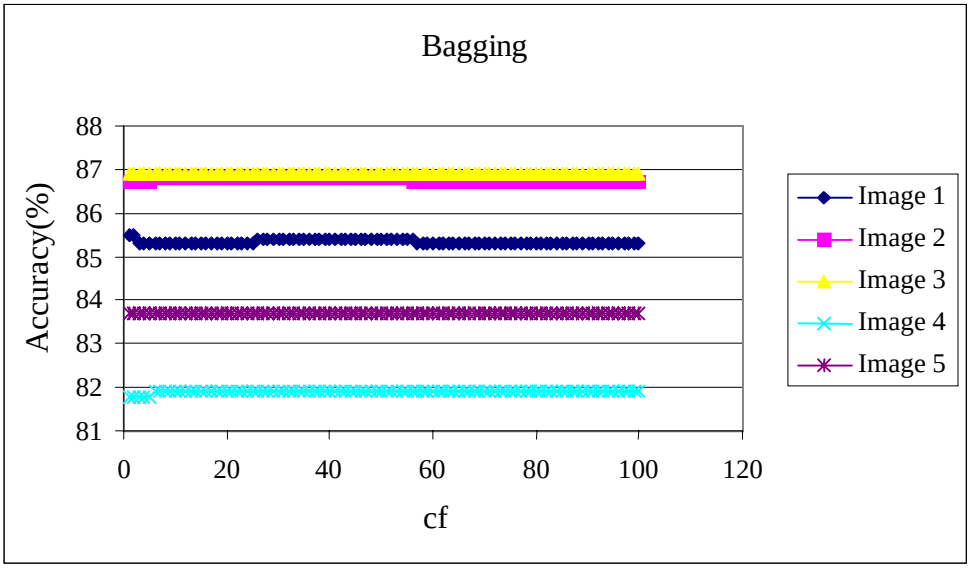


Figure 12. Committee of 40 bagged decision trees

The highest accuracy was 86.9% (image 3). This is higher than the highest accuracy we had obtained by training a single decision tree on all training data and testing it on each test image. The most accurate then had been 85.01% (Figure 9). The bagged trees are also more accurate than our baseline performance reference point. To make this last comparison fair, we tested the bagged ensemble over all the test data taken together. The result vis-à-vis a single decision tree is presented in Figure 13 for all certainty factors.

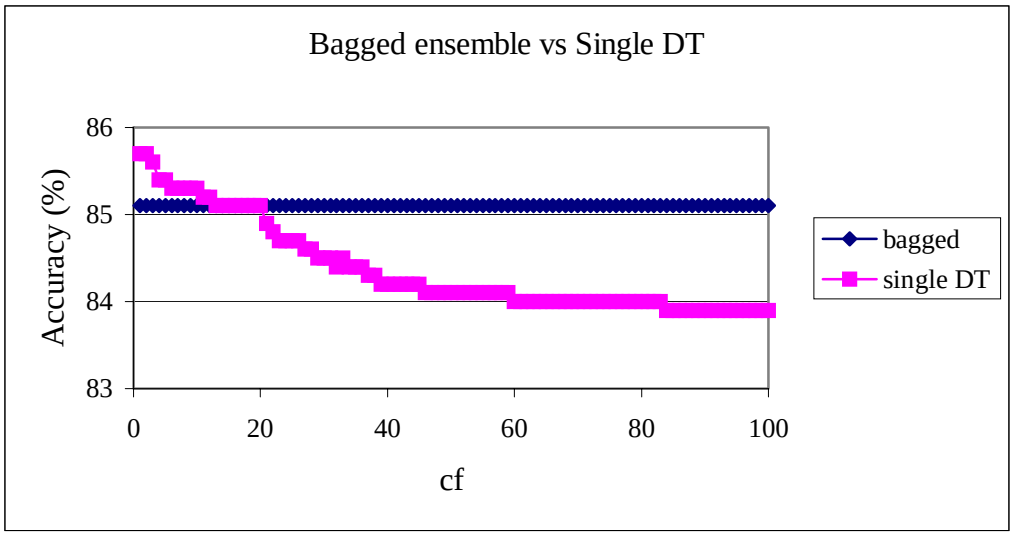


Figure 13. Comparing a bagged ensemble with a single decision tree

Although it may look like the error is exactly same, the fact is that there is variation in error, but the change is not big enough to produce a tenth of increase/decrease in the accuracy percentage. The graph in Figure 14 shows how the number of incorrect classifications changes for the bagged trees shown in Figure 13. However, one peculiar behavior observed in bagging experiments is that the trees did

poorly in classifying the “somewhat interesting” class. This is because this class is severely under represented in the data.

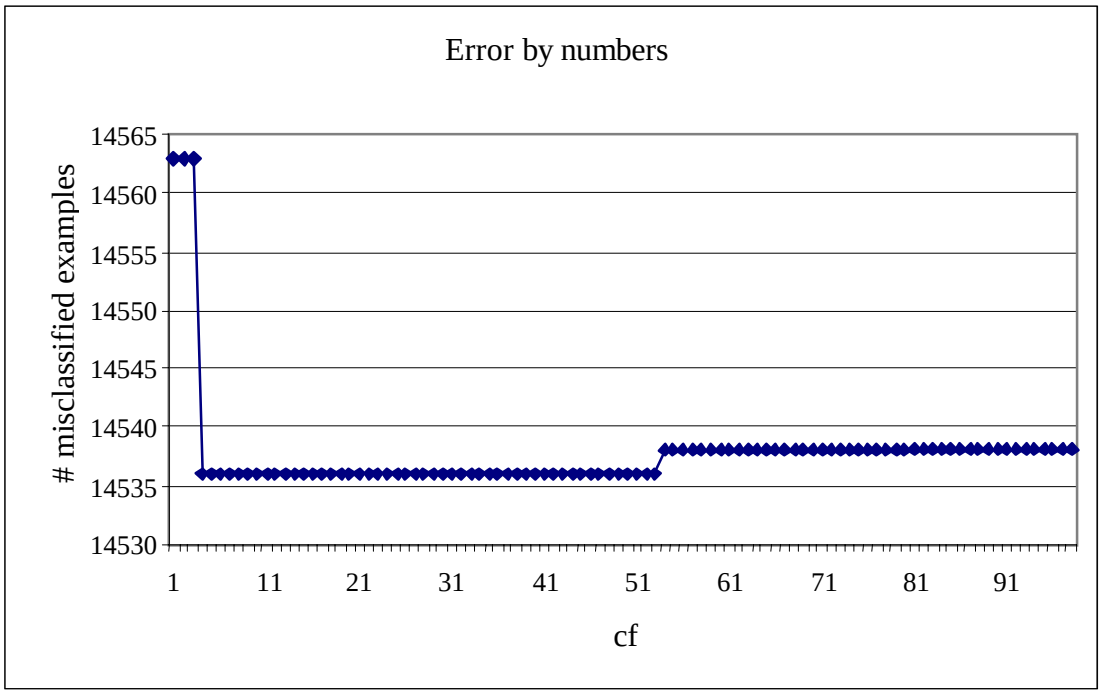


Figure 14. Errors by number of misclassified examples in Figure 13

4.5.1.2 Voting decision trees

40 DTs were created, one on each of the 40 spatial regions. For each test pixel, all the 40 trees were voted and the class assigned was the majority class. The overall percentage accuracy on the 5 test images ranged between 80 and 85 percent. Although this may appear to be good performance, the truth is that it isn’t. This is so because all (100%) of the “interesting” and “somewhat interesting” pixels were misclassified. In fact, the number of errors in each image was exactly equal to the number of pixels in the image belonging to these classes. One explanation for this is that while the class “not interesting” is present in all regions, the other two classes are present in very few regions.

Class I is present in only 27 spatial regions and SI in only 12. Of the 40 total spatial regions, 4 have only one class (NI), 33 have exactly two classes - 24 regions have NI and I; 9 have NI and SI. Only 3 spatial regions have all three classes. This means that the majority vote for SI pixels will always be something else, while that for I pixels will also be wrong with a very high probability. This shows that simply voting 40 decision trees is of no use.

4.5.2 K-nearest centroid classifier

We made use of a variant of the *k*-nearest neighbor classifier (KNN) which we call a *k*-nearest centroid classifier (KNC). First centroids are calculated for each class present in every region. All these centroids together form the "train set". The classifier then finds the *k*-nearest centroids for the pixel to be classified and these centroids are then allowed to vote for the pixel class. In the simplest case, if $k = 1$, then the test pixel will be assigned the class represented by the centroid to which it is closest.

4.5.2.1 Experiments

We have 40 spatial regions from the 5 training images. For each of these spatial regions, a centroid was calculated for every class present in it. Since the NI class is present in all 40 spatial regions, we get 40 centroids belonging to the NI class. Class I is present in 27 spatial regions, resulting in 27 centroids for I class and finally, class SI is present in only 12 spatial regions so we get 12 SI centroids. Thus we get a total of 79 centroids. Now for each test image, pixel classifications are made using KNC for $k = 1, 3 \ldots n$ where n is the number of centroids representing the smallest minority class. Allowing any more centroids to vote is very likely to affect the accuracy for the minority class since the

number of votes in their favor will always be insufficient. The smallest minority class here is SI, with 12 centroids. As a general rule, even numbers are avoided in nearest neighbor classification. Thus, we tried KNC for k = 1, 3, 5, 7, 9 and 11. Ties were broken in favor of the more interesting class, that is the order of preference was I>SI>NI.

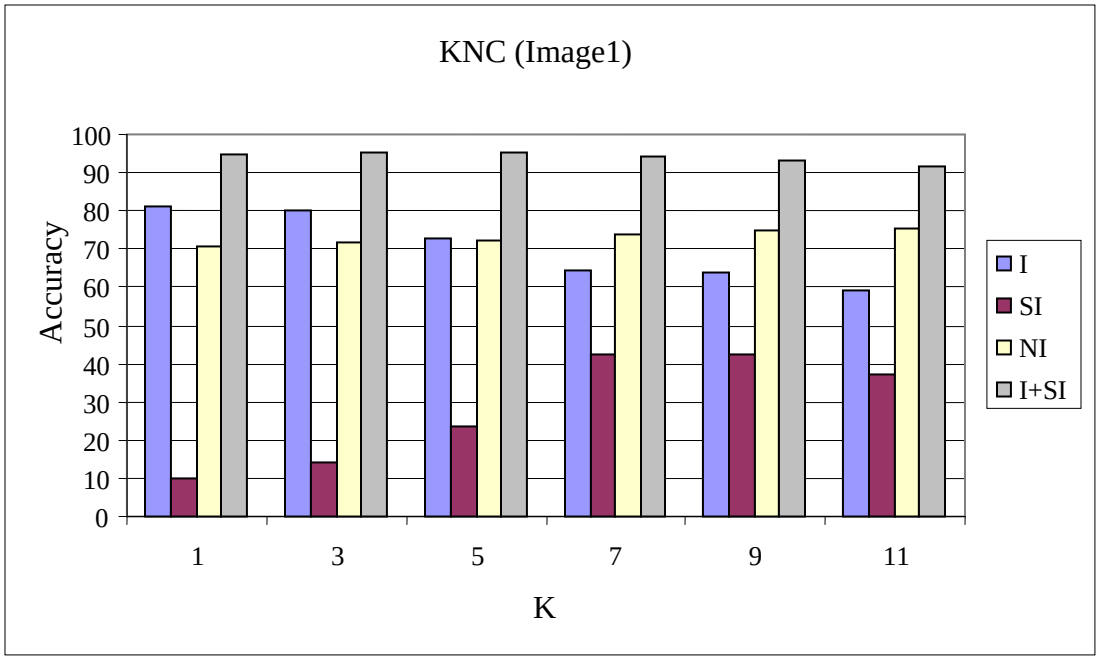
4.5.2.2 Results

The classification accuracies for the KNC classifier for each of the five test images are presented in Figure 15. The results are presented for the individual classes since a high overall accuracy is a very misleading term in this context. Our aim here to learn the regions of interest in the data even if in doing so we may have to lose some accuracy on the not interesting data. The idea is that if we do not completely miss any regions of interest (I and SI) then having false negatives is not an insurmountable problem. Further, any false positives in the vicinity of the interesting regions will be acceptable since they will tend to lead to the right place. Besides, confusing I and SI with each other is far more acceptable than confusing either of them with NI.

The charts in Figure 15 give the class-wise accuracy for the three classes as well as the combined accuracy of the I and SI classes. This combined accuracy tells us how well our classifier was able to differentiate any interesting event from an event of no interest to us.

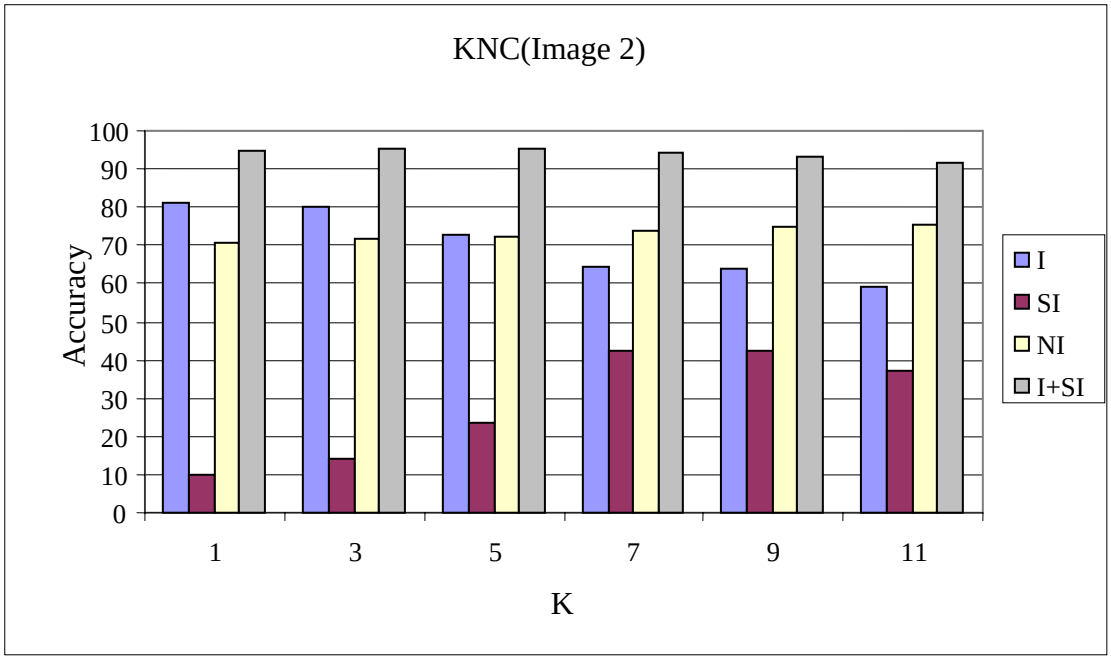
The accuracy histograms of Figure 15 show that the KNC classifier is very well able to achieve this goal. The combined percent accuracy of I and SI is almost always in high 90s. The accuracy for the NI class is also high. This means that the classifier did not greatly confuse the NI class with either of I and SI classes. This also tells us that while

the classifier did not greatly confuse NI with any other class, it did, however, get heavily confused between the I and SI examples. The low SI class accuracy and high I can be attributed to the tie-breaking rule we have used. Almost every time the classifier got a non NI test example for classification, there was a tie and since the tie breaker rule has a preference for the I class, these pixels get classified as I. This resulted in a high number of false positives, bringing down the SI class accuracy. To check this analysis, a quick experiment was run by changing the tie breaker rule to give higher preference to the SI class. That is, if we have the order of preference as SI>I>NI, we get similar results except this time the SI accuracy was high and I accuracy was low. Thus, our premise was validated.

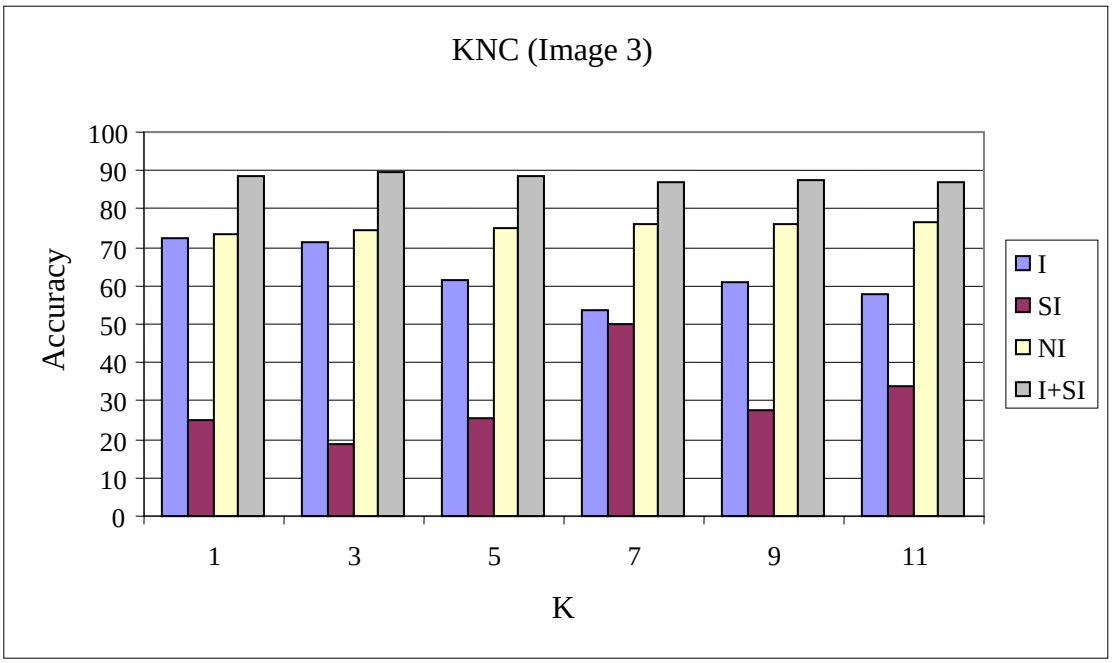


(a)

Figure 15. K-nearest centroids on the five test images.

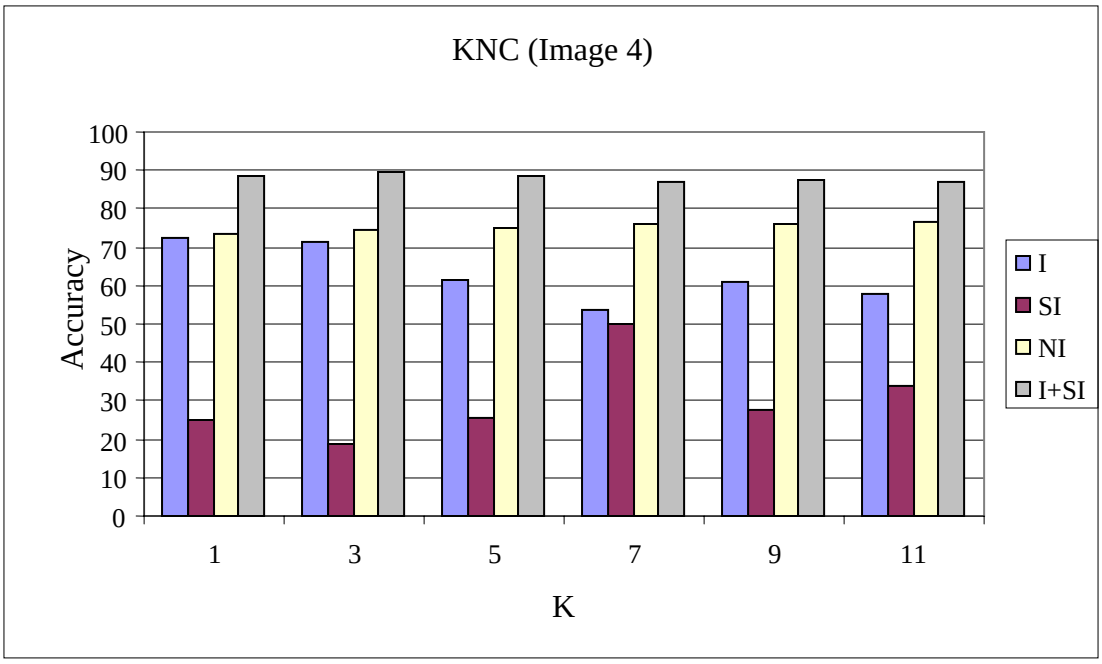


(b)

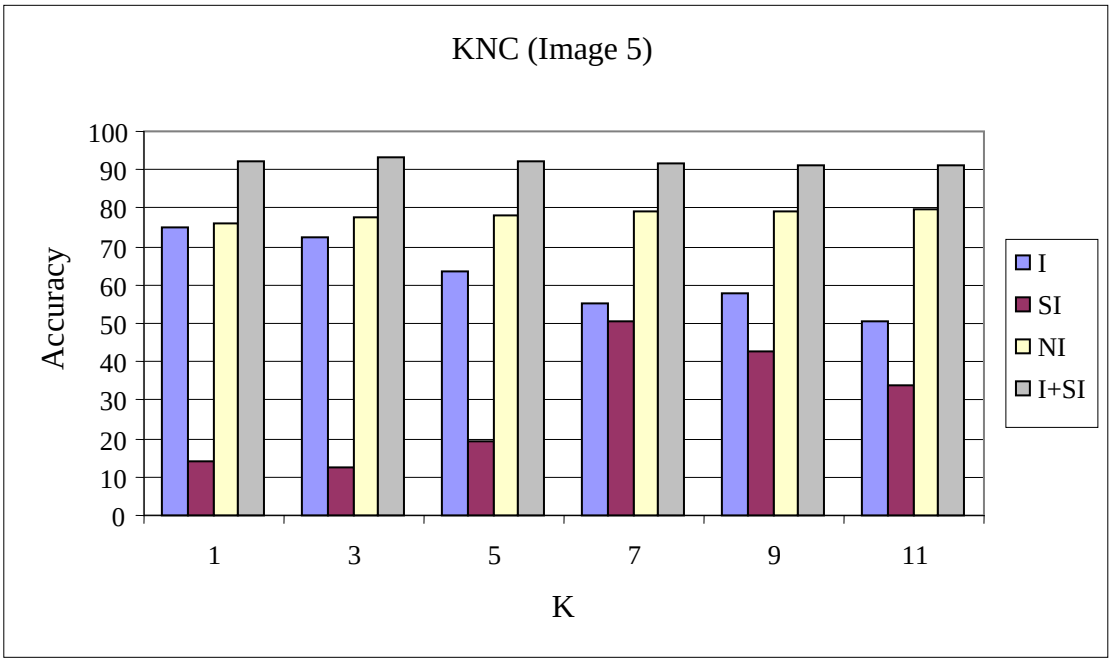


(c)

Figure 15. (continued)



(d)



(e)

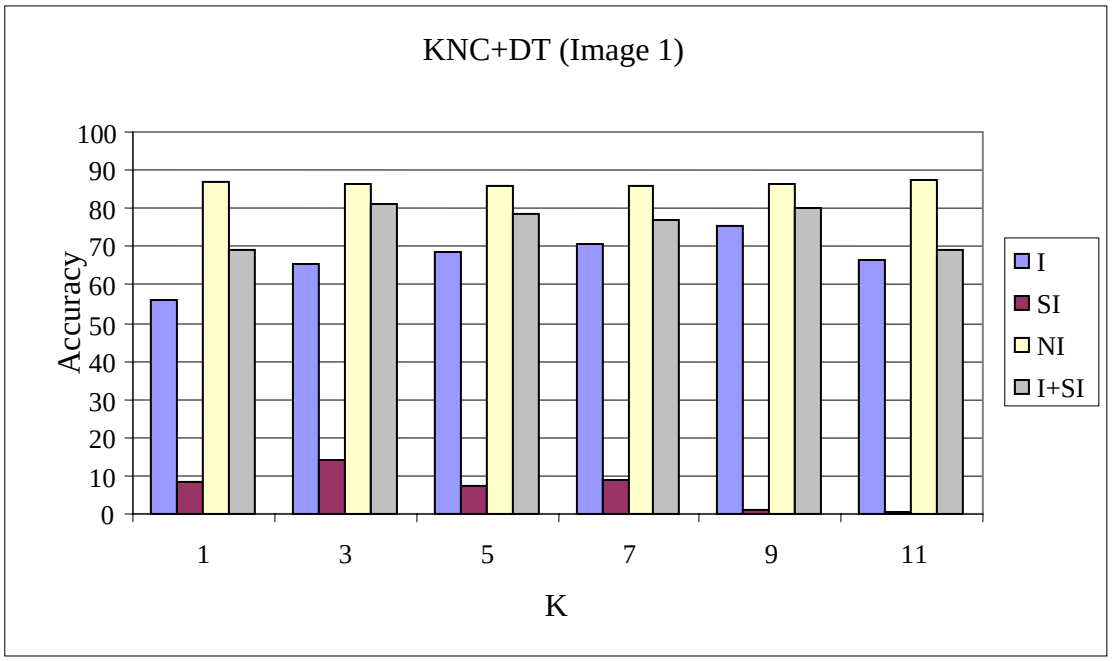
Figure 15. (continued)

4.5.3 Combining decision trees and KNC

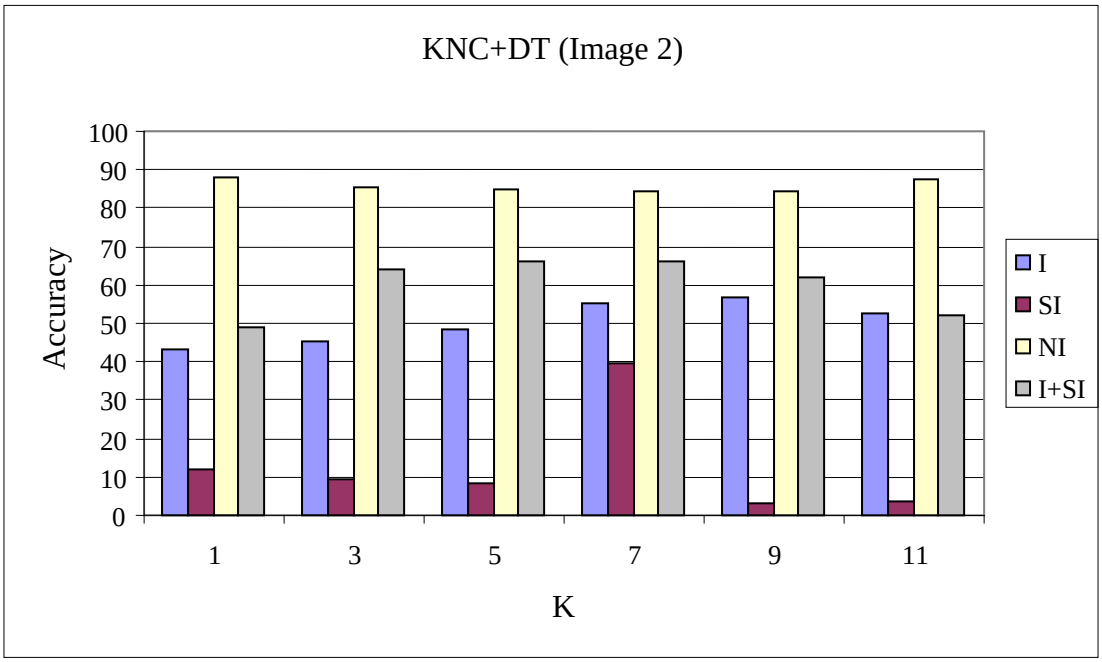
Up to this point, it was established that a committee of decision trees alone cannot do well on the areas of interest because there are not enough examples representing these classes. This lack of training examples prevents the decision tree from learning these classes well. KNC, on the other hand, can very efficiently separate the NI class from the other two classes. An interesting experiment at this point would be to combine the two approaches.

4.5.3.1 Experiments

KNC was used to identify the *n* regions 'nearest' to the test pixel. Then, decision trees associated with these regions were called to classify the pixel. These classifications were voted to get the predicted class of the pixel. Once again, n = 1, 3, 5, 7, 9 and 11 and ties were broken in favor of the more interesting class, that is the order of preference was I>SI>NI. The results are presented in Figure 16.

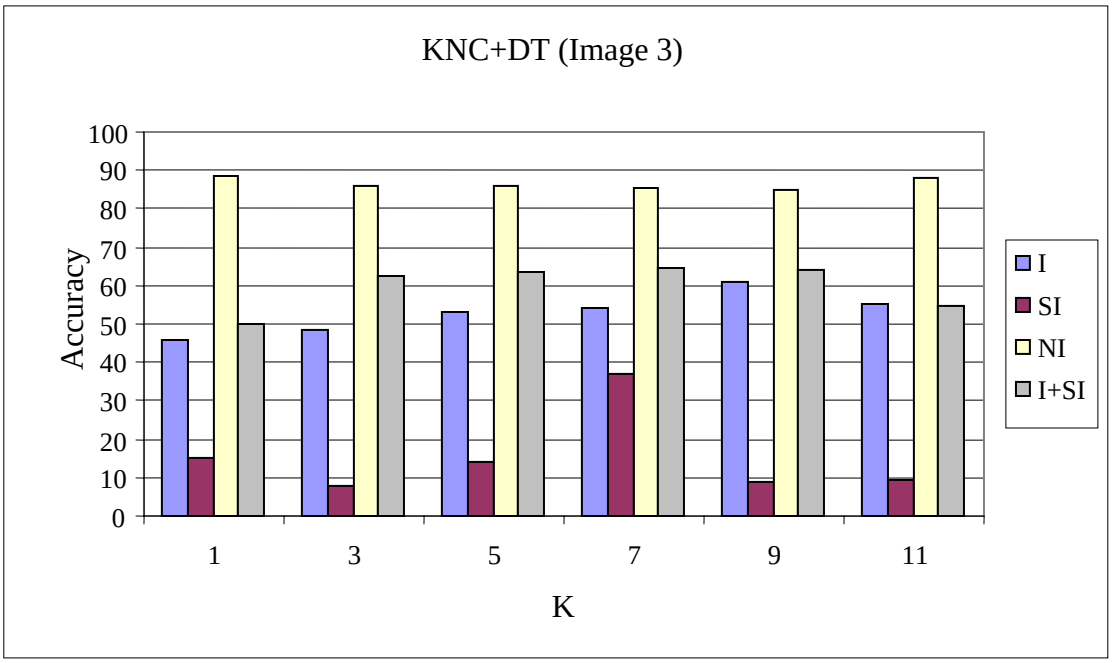


(a)

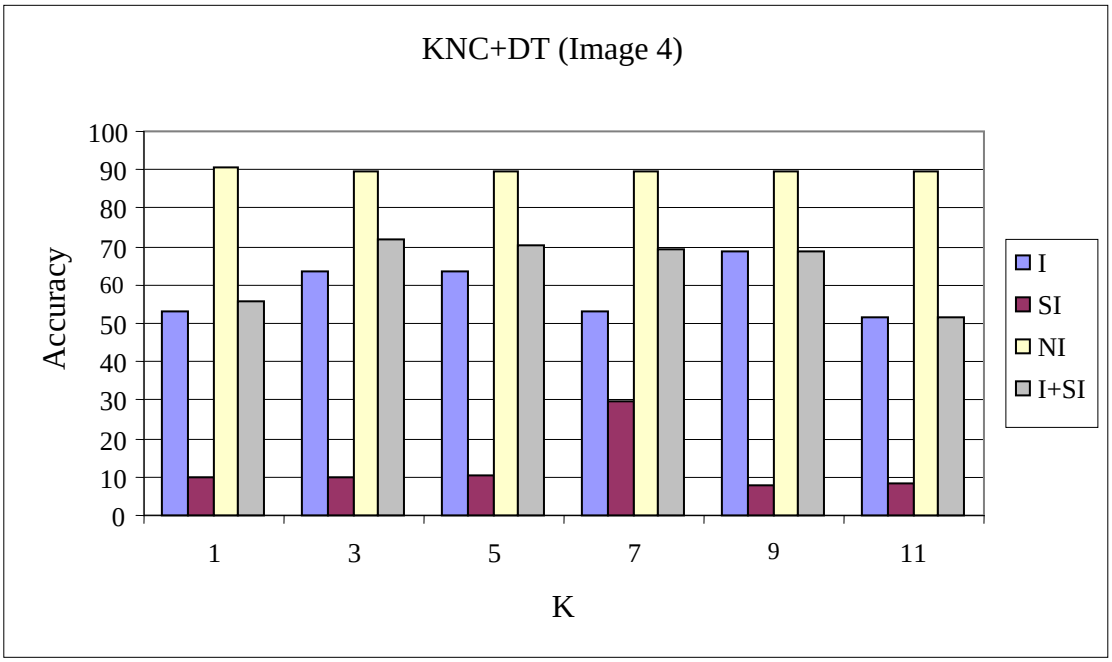


(b)

Figure 16. K-nearest centroids with decision trees on the five test images.

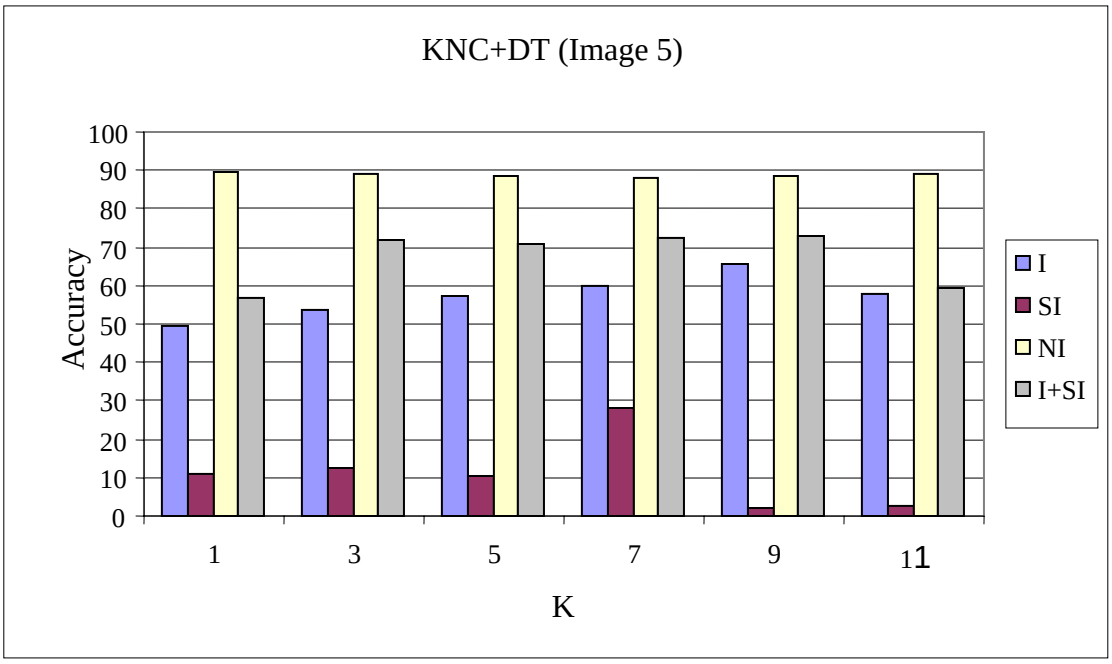


(c)



(d)

Figure 16. (continued)



(e)

Figure 16. (continued)

4.5.3.2 Results

Once again, the relative performance on the I and SI classes was found to change with the order in which ties were broken. Note that with KNC alone, the I and SI combined accuracy was very high (Figure 15). As soon as decision trees are thrown into the picture (Figure 16), the combined accuracy comes down and the NI class accuracy shoots up. This is once again understood by the fact that the training data is not balanced enough to help decision trees learn the minority classes and as a result, the tree predicts most of the test pixels as NI. Moreover, when we are using a combination of two classifiers like KNC and decision trees as shown in Figure 16, an I or SI test pixel has to be found correctly on both of them to be classified correctly. That is, first the KNC should match the pixel to a correct region and then the decision tree associated with that region should have a good

chance of predicting the correct class. By the term ‘correct region’ we mean a region that has at least some examples of that class. This is crucial since in our data only 12 regions have SI class and 27 have I class examples. In contrast all 40 regions have the NI examples, including a few which have only NI examples. A decision tree trained on only one type of example will always predict the same class. Such a tree voting for an I or SI class is not desirable. Thus, the trees taking part in the vote should be ‘good trees’.

4.6 Adding centroids for better class representation

Creating one centroid per class may not reasonably represent the classes in a region. This becomes more believable in the light of the fact that a lot of completely different types of pixels fall under the same class. For example, the I class consists of pixels from the mouth and eyebrows. In the feature space, mouth pixels may be located in the form of a cluster while the black and white parts of the eye may form two other clusters. A single centroid representing the I class will only give a point in the feature space which will be somewhere between all these clusters. Instead, it appears more reasonable to create multiple centroids for the class; with each constituent cluster represented in at least one of them.

Experiments were carried out with different numbers of centroid combinations. No really good centroid combination was found. The class accuracies changed with the number of centroids. The class with the most centroids always had the highest accuracy. This was because more and more test examples were matched to this ‘majority’ class. Thus, an increase in accuracy for one class always comes at the cost of some other class.

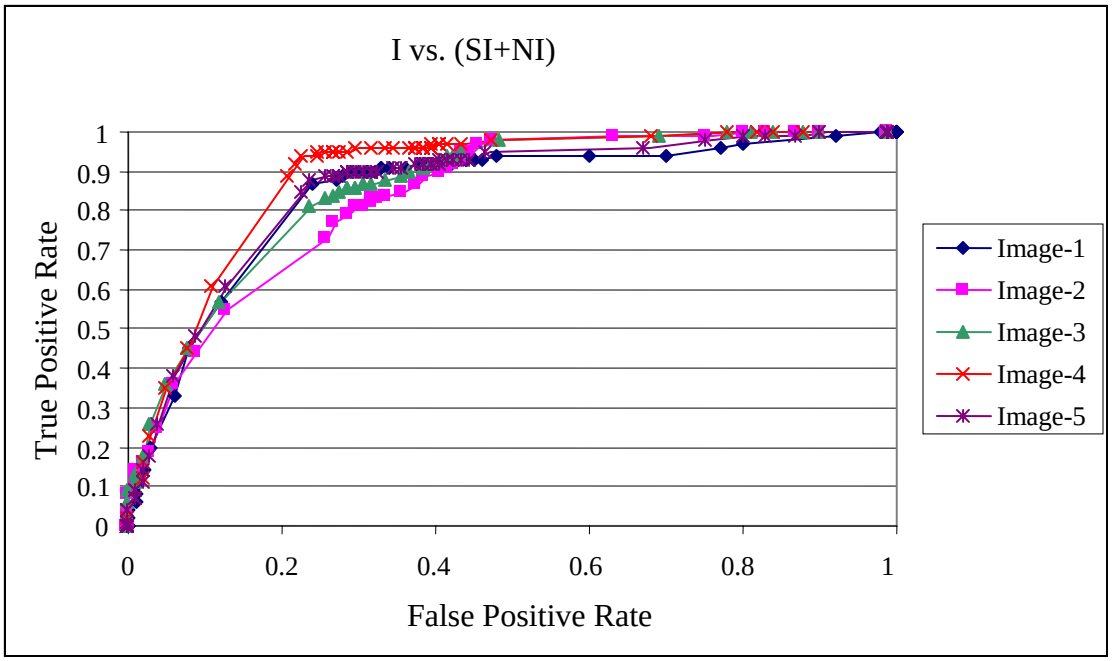
4.7 ROC curves

Receiver-Operating Characteristic (ROC) analysis [47, 48] is an evaluation technique used in signal detection theory, which in recent years has been increasingly used for types of diagnostic, machine-learning, and information-retrieval systems. It is simplest to apply in 2-class problems. ROC graphs plot false-positive (FP) rates on the x-axis and true-positive (TP) rates on the y-axis. The ROC curve identifies how many false positives one must tolerate to be guaranteed a certain percentage of true positives. An ideal ROC curve is the step-function.

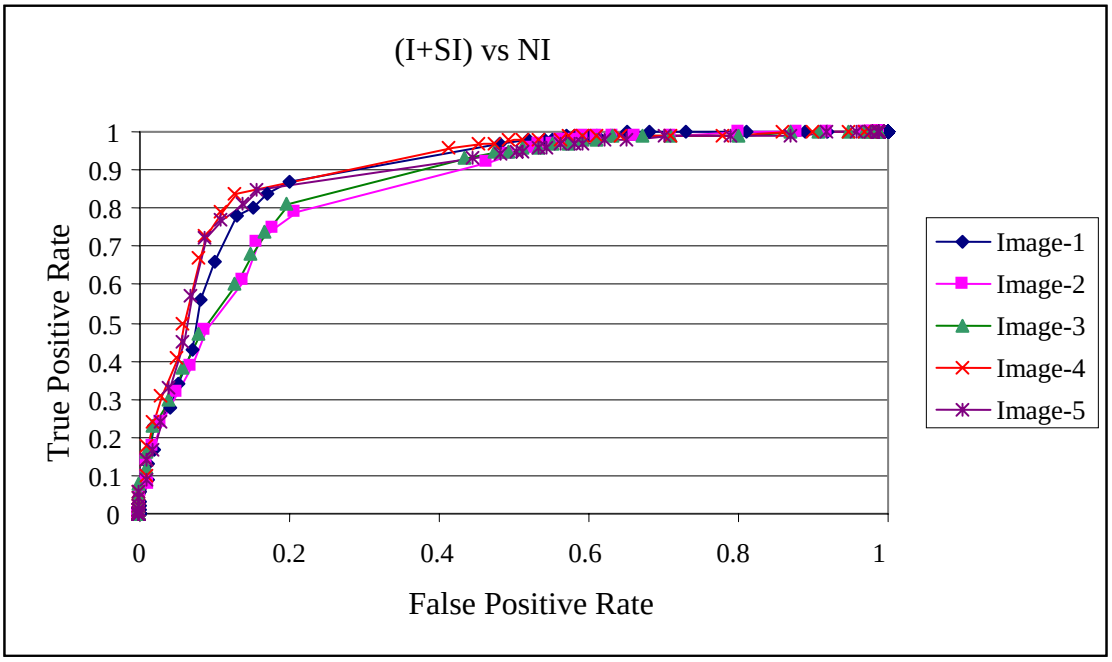
At some point, it becomes impossible to make fundamental improvements and one is left, in effect, with just moving to different points on an ROC curve. We are interested in two effects. One is fundamental improvement. The other is the ability to move along the ROC curve. We want any fundamental improvement we can get. Then we also want a smooth ROC curve from which to select points. Finding a fundamental improvement would mean a technique that leads to a better ROC curve or a greater Area Under the Curve (AUC).

Figure 17 gives the ROC curves with the K-nearest centroids approach. In Figure 17(a) classes SI and NI have been combined together as the not interesting class. In Figure 17(b) classes I and SI have been combined together as the interesting class. A total of 79 centroids were used (one centroid per class per region). The points on the curve were obtained as follows:

- K-nearest (k = 1, 2,...79) centroids were used to vote for the test class. Ties were broken in favor of the majority class. The TP and FP rates were highest for k = 1. As K was increased, the points moved towards zero. This was because the number of interesting centroids was too small (27 out of 79 for Figure 17(a) and 39 out of 79 for Figure 17(b)) so as the number of voting centroids was increased, more and more votes start coming for the not interesting class. After a certain value of k, the vote will always be in favor of the not interesting class.
- For every test pixel, its distance from all centroids was calculated. Then K nearest centroids were looked at, for $1 \leq k \leq 79$. If even one of them voted in favor of the interesting class, the test pixel was classified as interesting. So as the value of k increased, more and more TP and FP were created as the likelihood of encountering an interesting centroid increased. This method is exactly similar to the one above for k = 1, which forms the bridge point between the two methods.



(a)



(b)

Figure 17. ROC curves: K-nearest centroids

Similar analysis was done for KNC+DT type experiments in which first *k*-nearest centroids were found and then decision trees associated with the parent regions of these centroids were called to vote for the pixel class. The ROC curves are presented in Figure 18. Just as in the pure KNC ROC curves of Figure 17, this resulted in a decrease in TP and FP rates with an increase in the number of centroids. The *all or none* rule was also tried in which even if one of the *k* trees votes for the I class, the pixel was classified as I. This resulted in a smoothly increasing TP and FP rate with an increase in the number of trees.

The erratic left hand side portion of the curves in Figure 18 comes from the voting of the decision trees. This is because most of the training data used in building the trees was NI, so the trees tend to classify most pixels as NI. Also, out of the 40 trees (corresponding to the 40 spatial regions) almost half of them are guaranteed to never vote in favor of I class simply because they have been trained on only NI data. In addition, if the same tree was closest more than once on the same pixel, it will get to vote every time it is called. In this way, every time a tree voting for NI is called (and these trees are in the majority) the final vote is more likely to be in favor of NI. This is why there is no regularity in the curve. In addition to the *n* trees identified for $k = n$, $k = n+1$ may make a call to a new tree, which can vote either way, so there is no guarantee that the curve will always go in one direction with increasing *k*. However, the curves begin to smooth out as soon as the *all or none* rule comes in. Once again voting gives highest TP and FP with *k* = 1, i.e. when the tree associated with the single nearest centroid classifies the pixels. This point is the same as the lowest TP and FP point obtained from *all or none*. The

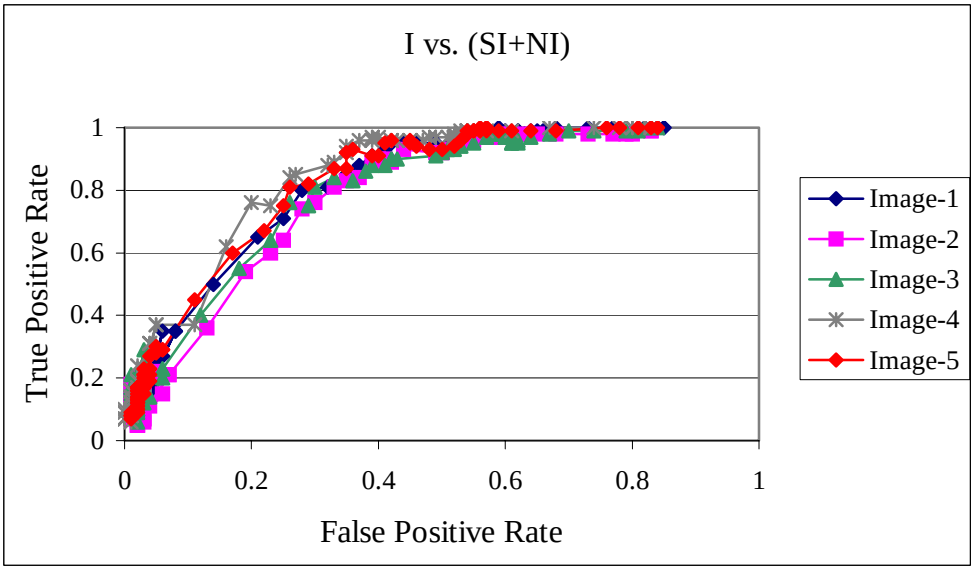
smooth increase in the former is understood by the fact that as we look at more and more trees, the chances of at least one of them voting for the I class increase.

The ROC curves for KNC with decision trees are more erratic due to the inefficient training of the trees. KNC alone gives smooth results, much the way we have been expecting. Also, the use of decision trees in Figure 18 prevents the curves from having a false positive rate of 1. Thus, from these ROC curves, it could be concluded that for data as imbalanced as ours, using KNC alone is a better approach than combining it with decision trees. This is also in agreement with the experimental results.

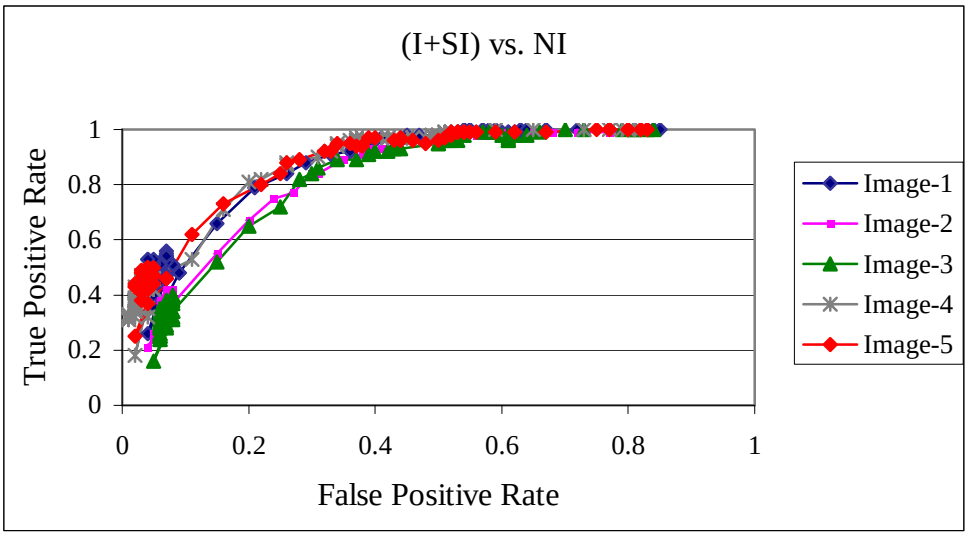
4.8 Comparison of all approaches

A comparison of all approaches used in this work was also carried out. ROC curves were plotted for average true positive rate and false positive rate over all the test data. This was done to make a fair comparison across all approaches and also with the baseline performance reference point. Recall that the baseline was established by taking the highest overall accuracy of a classifier (a decision tree) over all the test images taken together.

The ROC curves for this comparison are given in Figure 19. Curves are plotted for KNC, KNC+DT and bagging. The baseline performance reference point is also marked. The methodology for plotting the first two curves is the same as before. For bagging, different thresholds (ranging from 1 to the number of trees in the ensemble, viz. 40) were set on the number of votes required for a minority class. The TP and FP rates were highest when threshold was one. As this threshold was increased, the points moved towards zero. This was expected since not many trees will vote for the minority class.



(a)

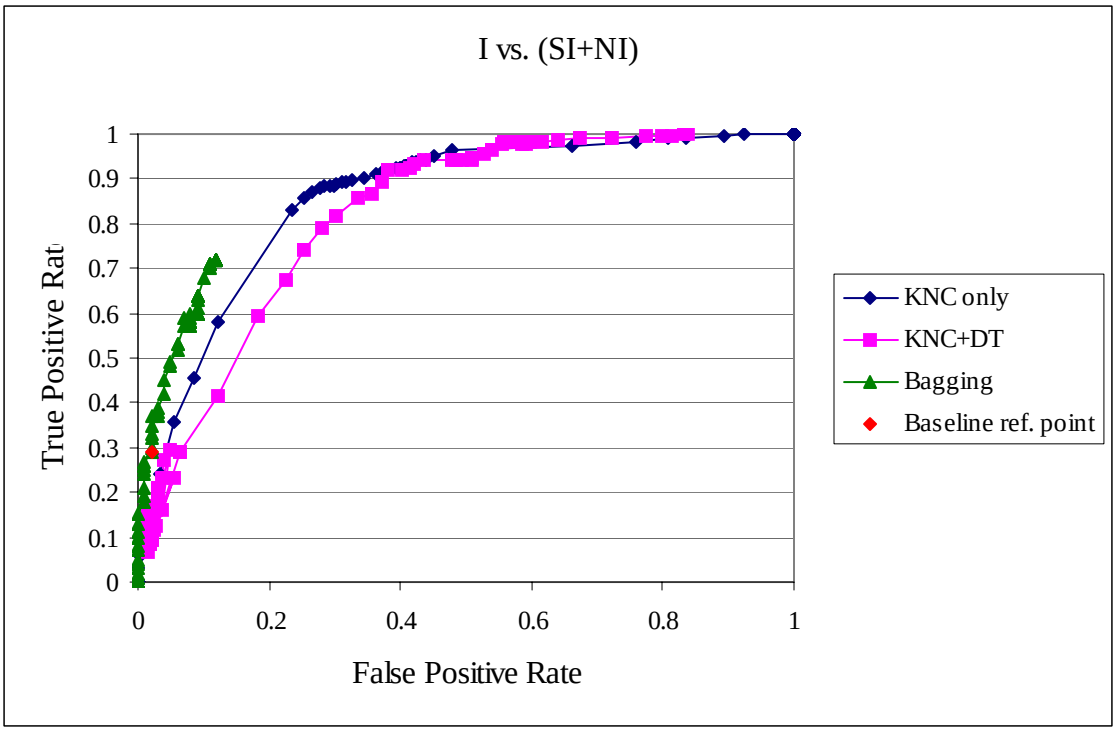


(b)

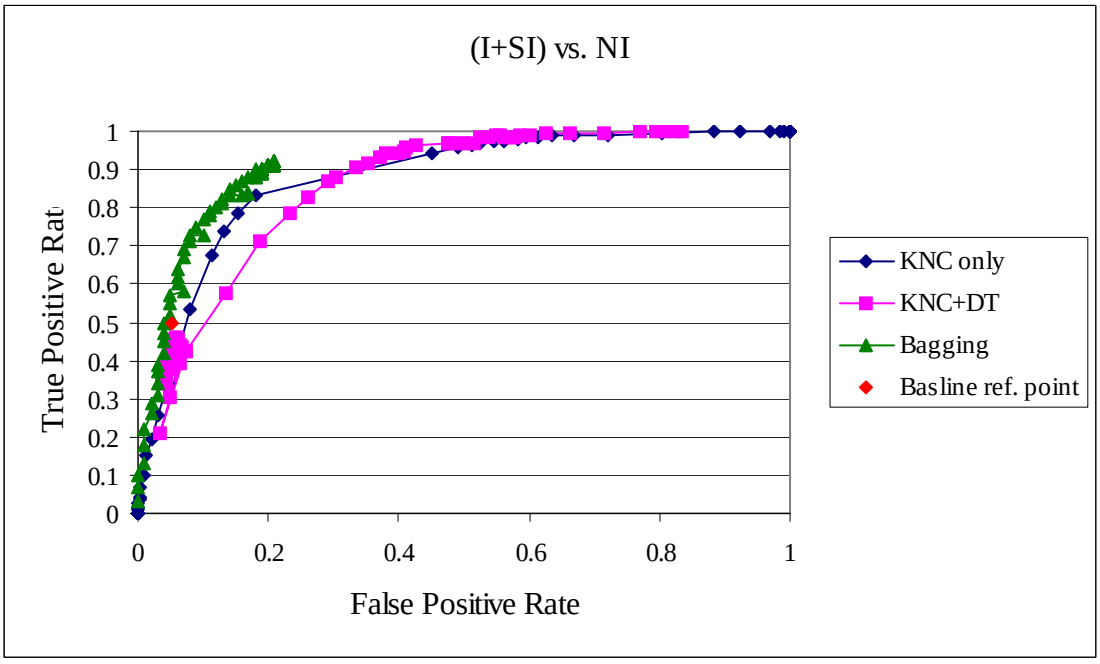
Figure 18. ROC curves: K-nearest centroids with decision trees

Note that the TP rate is much higher in Figure 19(b) in which the I and the SI classes were together treated as the interesting class. The curves for KNC and KNC+DT approaches are similar to the ROC curves of Figure 18. On the other hand, bagging fails to give a TP and/or FP rate of 1. However, it is interesting to note that at the baseline reference point, TP = 0.3 in Figure 19(a) and TP = 0.5 in Figure 19(b), the bagged ensemble has the smallest number of false positives. In other words, the bagged ensemble was better than any other approach at the baseline reference point.

In the ROC curves of Figure 19, the curve for KNC is always better than that for KNC+DT up to the bridge point which was described in Section 4.7. This supports our conjecture that using decision trees on this data is not a wise idea. Earlier we saw this when we combined KNC with DTs. The accuracy obtained from this classifier combination on the classes of interest was lower than the accuracy of KNC alone.



(a)



(b)

Figure 19. ROC curves for comparison of all approaches.

CHAPTER 5

SUMMARY AND FUTURE WORK

5.1 Summary

This thesis presented ways of creating ensembles of classifiers for the purpose of learning from disjoint data distributed across a large number of processors. An example problem was created to emulate a real world situation. A single decision tree established the baseline for performance and then committees of classifiers like decision trees and a variant of the *k*-nearest neighbors, which we call *k*-nearest centroids, were formed to try to give an improvement over this baseline. *K*-nearest centroids was developed because our data was distributed across different CPUs and any movement of such a large amount of data will be very expensive. So centroids were found which we hoped will capture the information of the data.

None of the classifiers built was able to learn all the properties of our data. Decision trees suffered because the training data did not have sufficient representation from the two classes of interest. The nearest centroid method did a good job of distinguishing the classes of any level of interest from the class of no interest. This was acceptable because the idea is that if we do not completely miss any regions of interest then having false negatives is not a problem. Further, any false positives in the vicinity of the interesting regions will be acceptable since they will tend to lead to the right place.

Besides, confusing regions of different levels of interest with each other is far more acceptable than confusing any of them with a region of no interest.

The idea of increasing the number of centroids to try to get the best centroids was also explored. This approach was motivated by the understanding that a lot of completely different types of pixels fall under the same class and so a single centroid may not adequately represent the properties of the class. However, as experiments were carried out with new centroids, it was found that an increase in accuracy for one class always comes at the cost of some other class and there may not be a way of improving on all the classes at the same time. This inspired us to do ROC analysis. The motivation here was that finding an improvement in accuracy would lead to better ROC curves.

The basic reason why the committees of classifiers failed to produce very accurate prediction models is the high imbalance of class distribution in the data. About 84% data points are not interesting, 11% are interesting and only 5% are somewhat interesting. It could be concluded that attempts at creating a classifier, or even committees of classifiers, from this data as such are not likely to result in any improvement.

5.2 Comparison with baseline results

Committees of classifiers in general perform better than a single decision tree which was used to establish a baseline for performance. A committee of 40 DTs that was created by bagging on data from all five training images, gave the highest accuracy of 86.9% (on image 3, see Figure 12). This is higher than the highest accuracy we had obtained by training a single decision tree on all training data and testing it on each test image. The most accurate then had been 85.01%. The bagged trees are also more accurate than the

most accurate pruned tree trained on all training data and tested on all test data together. The best performance was 85.7%.

KNC classifiers perform better than the baseline on the classes of interest. The overall accuracy with these classifiers is slightly lower than the baseline, but the accuracy of the classes of interest, both combined and separately, is much higher.

5.3 Future work

It would be interesting to see how this data can be modified or re-sampled to get a more learnable training set. Suggestions include minority over-sampling [33] and majority under-sampling. It may also help to use more training data, especially one with more examples of interest. The effects of adding location as a feature may also be explored.

More advanced classifiers like multi-layered neural networks and support vector machines may also be included in future work. An in-depth analysis of ROC curves of different classifiers may put forth a way of creating a distinctively accurate committee of one or more of these classifiers. The search for centroids that best represent the classes presented in this thesis is not exhaustive. This may also be pursued in future.

Boosting may also help increasing the accuracy of classifying the regions of interest. This is because boosting maintains a set of weights over the training set and after each classifier is learned, it increases the weight associated with the misclassified examples and decrease the weight of examples that are correctly classified. In our problem, most of the I and SI examples are misclassified due to the imbalanced nature of data. Incorporating boosting will add weight to the examples of interest; thus, it might be expected to produce better prediction models.

REFERENCES

[1] "ASCI at Sandia" http://www.sandia.gov/ASCI/, 2003.

[2] "ASCI at LLNL", http://www.llnl.gov/ASCI/, 2003.

[3] "ASCI at LANL", http://www.lanl.gov/ASCI/, 2003.

[4] P. J. Phillips, H. Wechsler, J. Huang, and P. Rauss, "The *FERET* database and evaluation procedure for face recognition algorithms," *Image and Vision Computing Journal*, Vol. 16, No. 5, pp 295-306, 1998.

[5] "The Facial Recognition Technology (FERET) Database", http://www.itl.nist.gov/iad/humanid/feret/feret_master.html

[6] "FERET", http://www.dodcounterdrug.com/facialrecognition/Feret/feret.htm

[7] J.R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA: Morgan Kaufmann, 1993.

[8] T.M. Mitchell, *Machine Learning*. New York, NY: McGraw Hill, 1997.

[9] J.R. Quinlan, "Simplifying decision trees", *International Journal of Man Machine Studies*, vol. 27, pp. 227-248, 1987.

[10] L. Breiman, J.H. Friedman, R.A. Olshen and P.J. Stone, *Classification and Regression Trees*. Belmont, CA: Wadsworth International Group,1984.

[11] J. Mingers, "An empirical comparison of selection methods for decision tree induction", *Machine Learning*, 3(4), pp. 319-342, 1989.

[12] E.B. Hunt, J. Martin and P.J. Stone, *Experiments in Induction*, New York, NY: Academic Press, 1966.

[13] A. Aamodt and E. Plazas, "Case-based reasoning: Foundational issues, methodological variations, and system approaches", *AI Communications*, 7(1), pp. 39-52, 1994.

[14] D. Aha, D. Kibler and M. Albert, “Instance based learning algorithms”, *Machine Learning*, 6, pp. 37-66, 1991.

[15] K.D. Ashley, *Modeling Legal Argument: Reasoning With Cases and Hypotheticals*, Cambridge, MA: MIT Press, 1990.

[16] T. Cover and P. Hart, “Nearest neighbor pattern classification”, *IEEE Transactions on Information Theory*, 13, pp. 21-27, 1967.

[17] J. L. Kolodner, *Case-based reasoning*, San Francisco, CA: Morgan Kaufmann, 1993.

[18] L. Breiman, “Bagging predictors”, *Machine Learning*, 24, pp. 123-140, 1996.

[19] N. Chawla, T. Moore, L. Hall, K. Bowyer, W. Kegelmeyer, and C. Springer, “Distributed learning with bagging-like performance”, *Pattern Recognition Letters*, 24, pp. 455–471, 2003.

[20] K. Bowyer, N. Chawla, J. T.E. Moore, L. Hall, and W. Kegelmeyer, “A parallel decision tree builder for mining very large visualization datasets”, *IEEE Systems, Man, and Cybernetics Conference*, pp. 1888–1893, 2000.

[21] T. Dietterich, “Ensemble methods in machine learning”, presented at 1st International Workshop on Multiple Classifier Systems, Cagliari, Italy, 2000.

[22] A.J.C. Sharkey, “Combining artificial neural nets: Ensemble and modular multi-net systems”, in *Multi-Net Systems*, pp. 1-30: Springer-Verlag, 1998.

[23] Y. Freund, “Boosting a weak learning algorithm by majority”, presented at Workshop on Computational Theory, 1990.

[24] Y. Freund and R.E. Schapire, “Experiments with a new boosting algorithm”, presented at 13th International Conference on Machine Learning, 1996.

[25] “ASCI Red”, http://www.sandia.gov/ASCI/Red/, 2003.

[26] “ASCI Blue Pacific”, http://www.llnl.gov/asci/platforms/bluepac/, 2003.

[27] “ASCI Blue Mountain”, http://www.lanl.gov/projects/asci/bluemtn/, 2003.

[28] “ASCI White”, http://www.llnl.gov/asci/platforms/white/, 2003.

[29] T.K. Ho, “Random decision forests”, presented at 3rd International Conference on Document Analysis and Recognition, 1995.

[30] T.K. Ho, “The random space method for constructing decision forests”, *IEEE Transactions on PAMI*, vol. 20, pp. 832-844, 1998.

[31] N.V. Chawla, S. Eschrich and L.O. Hall, “Creating ensembles of classifiers”, Department of Computer Science and Engineering, University of South Florida, ISL-01-01, 2001.

[32] L. Breiman, “Pasting small votes for classification in large databases and on-line”, *Machine Learning*, vol. 36, pp. 85-103, 1999.

[33] N.V. Chawla, K.W. Bowyer, L.O. Hall, P.W. Kegelmeyer, “SMOTE: Synthetic Minority Over-sampling technique”, *International Conference on Knowledge Based Computer System*s, 2000.

[34] L.O. Hall, N. Chawla, K.W. Bowyer, W. P. Kegelmeyer, “Learning rules from distributed data", *Large-Scale Parallel KDD Systems, International Conference of Knowledge Discovery and Data Mining*: Springer Verlag, 1999.

[35] S. Brand, J. Laaksonen, E. Oja**, “**Statistical Shape Features in Content-Based Image Retrieval”, *International Conference on Pattern Recognition*, 2000.

[36] M. Tuceryan and A.K. Jain, *Texture Analysis*. The Handbook of Pattern Recognition and Computer Vision (2nd Edition), pp. 207-248: World Scientific Publishing Co., 1998.

[37] M. Sonka, V. Hlavac, R. Boyle, *Image Processing, Analysis and Machine Vision*: Kluwer Academic Publishers, 1998.

[38] R.C. Gonzalez, R.E. Woods, *Digital Image Processing*, 2nd Edition: Addison-Wesley, 2001.

[39] C.M. Bishop, *Neural Networks for Pattern Recognition*. Oxford, England: Oxford University Press, 1996.

[40] L. Fu, *Neural Networks in Computer Intelligence*. New York, USA: McGraw Hill, 1994.

[41] D.E. Rumelhart and J.L. McClelland, *Parallel Distributed Processing: Exploration in the Microstructure of Cognition,* vol. 1 & 2. Cambridge, MA: MIT Press, 1986.

[42] G. Cooper and E. Herskovits, “A Bayesian method for induction of probabilistic networks from data”, *Machine Learning*, vol. 9, pp. 309-347, 1992.

[43] F.V. Jensen, *An Introduction to Bayesian Networks*. New York, NY: Springer Verlag, 1996.

[44] C.J.C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition", *Data Mining and Knowledge Discovery*, vol. 2(2), pp. 121-167, 1998.

[45] T.G. Dietterich and N.S. Flann, "Explanation based learning and reinforcement learning: A unified view", *Proceedings of the 12th International Conference on Machine Learning*, pp. 176-184, San Francisco: Morgan Kaufmann, 1995.

[46] "The GIMP", http://www.gimp.org/

[47] J.A. Hanley, B.J. McNeil, "The meaning and use of the area under the Receiver Operating Characteristic (ROC) curve", *Radiology*, 143, pp. 29-36, 1982.

[48] C.E. Metz, "Basic principles of ROC analysis", *Semin Nuclear Med*, vol 8(4), pp. 283-298, 1978.

[49] "The CSU Face Identification Evaluation System", http://www.cs.colostate.edu/evalfacerec/