A Learning Approach to Detecting Lung Nodules in CT Images

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Michael G. Aschenbeck

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Fadil Santosa

December, 2009

Abstract:

Lung cancer is one of the most common types of cancer and has the highest mortality rate. Unfortunately, it is a long and difficult process for the physician to detect the presence of this disease. He/she must search through three-dimensional medical images and look for possibly cancerous, small structures that are roughly spherical. These structures are called pulmonary nodules.

Due to the difficult and time consuming detection task faced by the physician, computer-aided detection (CAD) has been the focus of many research efforts. Most of these works involve segmenting the image into structures, extracting features from the structures, and classifying the resulting feature vectors. Unfortunately, the first of these tasks, segmentation, is a difficult problem and many times the origin for missed detections.

This work attempts to eliminate the structure segmentation step. Instead, features are extracted from fixed size subwindows and sent to a classifier. Bypassing the segmentation step allows for every location to be classified.

Feature extraction is accomplished by learning a complete basis for the subwindow on the training set and using the inner product of the subwindow with each basis element. This approach is preferred over choosing features based on human interpretation, as the latter approach will most likely result in valuable information being overlooked. The bases used are derived from the singular value decomposition (SVD), a modification of the SVD, tensor decompositions, vectors reminiscent of the Haar wavelets, and the Fourier basis.

The features are sent to a number of different classifiers for comparison. The classifiers include parametric methods such as likelihood classifiers and probabilistic clustering, as well as non-parametric classifiers such as kernel support vector machines (SVM), classification trees, and AdaBoost.

While different feature and classifiers bring about a wide range of results, the non-parametric classifiers unsurprisingly produce much better detection and false positive rates. The best combination on the test set yields 100% detection of the nodule subwindows, while only classifying 1% of the non-nodule windows as nodules. This is in comparison to previous CAD approaches discussed in this thesis which achieve no better than 85% detection rates.

Acknowledgements

I would like to acknowledge all of the positive influences I have received over the course of my studies. This dissertation would not have been possible without the support and guidance I have received.

First and foremost, I am deeply indebted to my advisor, Dr. Fadil Santosa. With his guidance, I discovered my passion for computer vision. He taught me about mathematics, research, and industry. He inspired me with his knowledge, enthusiasm, and drive, while maintaining a great sense of humor. I owe much of my success to him.

I am very grateful for the financial and academic support that the School of Mathematics at the University of Minnesota gave me. In particular, my committee members, Fadil Santosa, Gilad Lerman, Willard Miller, and Yuhong Yang, provided me with helpful feedback during my research.

A semester of my financial support was provided by National Science Foundation grant DMS0937703. I would like to thank the NSF for their role in my research.

I would also like to thank Mathematics Department at my undergraduate institution, the University of St. Thomas. The St. Thomas faculty is largely responsible for sparking my interest in mathematics and motivating me to take it to the next level.

Finally, I owe everything to my family and friends. They have supported me one hundred percent throughout my work. When I needed encouragement, they were there. When I needed some time away from my studies, they were there. When I experienced a setback they would make sure I took them with a grain of salt, but when I achieved a milestone they would make sure I stopped to enjoy it. I would not be where I am today without them.

Dedication

This dissertation is dedicated to my parents, Glenn and Susan Aschenbeck. Without your unconditional love and support, I would not have accomplished anything in life, much less this work.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem Statement

For women aged between 40 to 79 and men between 60 to 79, cancer is the leading cause of death [3]. Among all age groups for both males and females, cancer is one of the 5 leading causes of death, and lung cancer is one of the most common types. The American Cancer Society reports that there were an estimated 215,020 new cases of lung cancer in the United States in 2008, accounting for 15% of the estimated total 1,437,180 new cases of cancer. Lung cancer's mortality rate is the highest of all the cancers; the estimated 161,840 deaths due to lung cancer account for 29% of the estimated total 565,650 deaths due to cancer.

The difficulty of early detection for this disease is a main reason why lung cancer has the highest mortality rate. Like most cancers, survival rate depends on how early cancer is detected. For cases in stage I, ones in which the tumor has not spread to any lymph nodes, the five year survival rate is about 70%. This is in contrast to a five year survival rate of 1% for stage IV lung cancer, cases in which the tumor has spread to the other lung or other parts of the body [4]. Unfortunately, in its early stages, there are usually no symptoms. This makes early detection very difficult. Indeed, only 15% of lung cancers are found when they are still localized [5].

While a biopsy is the only way to make a definitive diagnosis, there are other tests that physicians commonly perform to help determine if lung cancer is present. The more

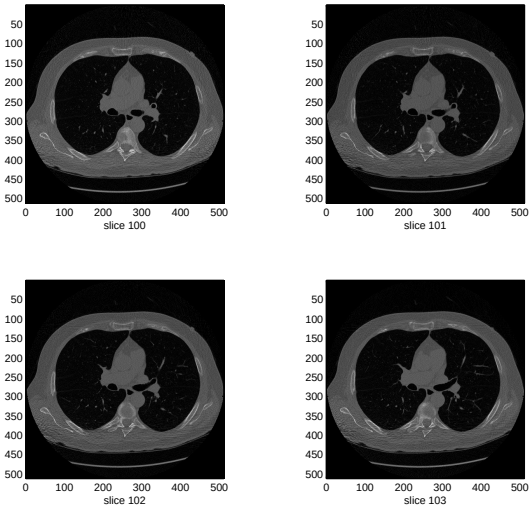

Figure 1.1: Example of 4 slices of a Computed Tomography (CT) scan.

common of these tests are the positron emission tomography (PET) scan, magnetic resonance imaging (MRI) scan, and computed tomography (CT) scan (figure 1.1) with the latter two being the most detailed. Each of these 3D images are expensive to produce and time consuming to analyze.

In a typical PET, MRI, or CT screen for lung cancer, the physician looks for pulmonary nodules. These nodules are small round growths, less than three centimeters in diameter, in the lungs. Nodules are not necessarily cancerous, but about 40% turn out to be.

To make things more complicated, there are three classes of nodules: juxta-pleural, juxta-vascular, and isolated. Juxta-pleural nodules are nodules which are attached to the pleura, or wall, of the lung. Juxta-vascular nodules are nodules that are attached to vessels in the lung. Finally, isolated nodules are neither attached to pleura or vessels. Isolated nodules tend to be much more spherical, while juxta-pleural and juxta-vascular can be stretched and may appear to be joined with the respective pleura or vessel.

Searching for a nodule is a long and tedious task for a physician. Figure 1.1 shows 4 slices of a CT scan of a lung. While there appears to be many round structures that

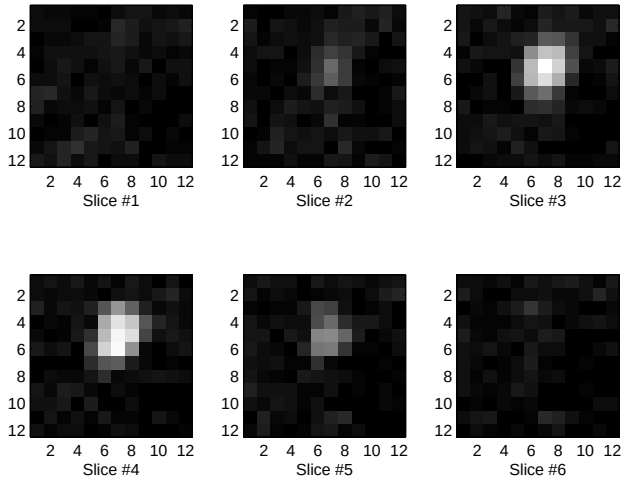

Figure 1.2: Example of a nodule in a CT scan. Notice the structure begins to appear in slice 2, gets larger, then diminishes after slice 5.

resemble nodules, there are no nodules in this image. Rather, the round structures are cross-sections of blood vessels running through the lungs. Examples of some CT scan subwindows that contain a blood vessel can be seen in figures 1.3 and 1.4.

Since the blood vessels look so similar to nodules when looking at a single slice, all slices must be considered. This means, a physician must start at a small region at the top of a lung, and traverse down the slices, looking for round structures that begin, get bigger, diminish, and finally disappear, such as the nodule shown in figure 1.2. The physician then continues to move to a separate region at the top of the lung and repeats the process until all portions of the lung have been considered. The complete diagnosis process takes the physician a large amount of time. With so much data to sort through, the process is also subject to missed nodule detections.

For the reasons outlined above, a computer-aided system designed to identify lung nodules is a very attractive idea. Eliminating human interpretation is not the primary goal of such a system. Instead, the uses of such a system would include double checking the physician's diagnosis (for missed nodules), checking for lung nodules on patients

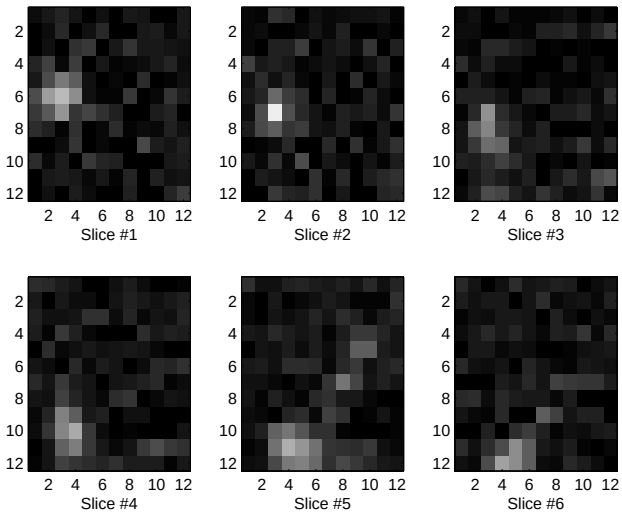

Figure 1.3: First example of a vessel in a CT scan. Notice there is no apparent slice where the structure begins or ends. Further, the diameter of the structure on each slice seems about the same. This is evidence that the structure is a vessel running perpendicular to the slices.

who had medical images taken for other reasons, aiding the physician in his/her search, etc.

A computer-aided lung nodule detection system has several requirements. First, the false negative rate must be extremely low. False negatives are nodules that are not detected by the system. One missed nodule can be a matter of life and death, so a system missing nodules is useless. Second, the false positive rate, the rate at which the system reports a non-nodule structure as a nodule, must be relatively low. Directing the attention of a physician to a large number of non-nodule structures will waste his/her time, and will thus eliminate the value of a computer-aided system. Finally, a system must run in a reasonable amount of time.

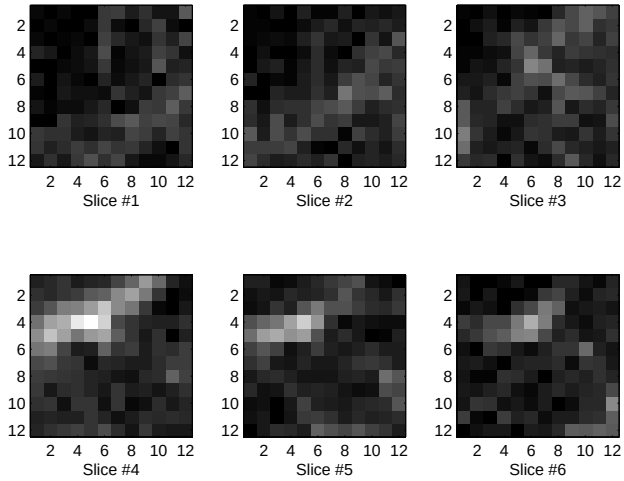

Figure 1.4: Second example of a vessel in a CT scan. While it seems the structure begins on slice 4 and ends on slice 6, the non-circular shape with similar width gives evidence that this is a vessel that is running diagonally through the slices.

1.2 Current Approaches to Computer-Aided Lung Nodule Detection

The vast majority of computer-aided lung nodule detection systems begin by first segmenting the lung, and then segmenting anatomical structures and nodules in the lung. Once these structures inside the lung are recorded, they are all treated as nodules. The next step is false positive reduction, i.e. eliminating most of the anatomical structures which are not nodules. A false positive reduction scheme usually consists of extracting features from the structure and feeding them to a classifier.

Giger et al. [6] used basic fixed thresholding [7] to identify the lung, as well as other structures inside the lung. The process is repeated several times with a different threshold to make sure every structure is segmented properly. Structures from one such threshold can be seen in figure 1.5. From each structure, nine features are computed.

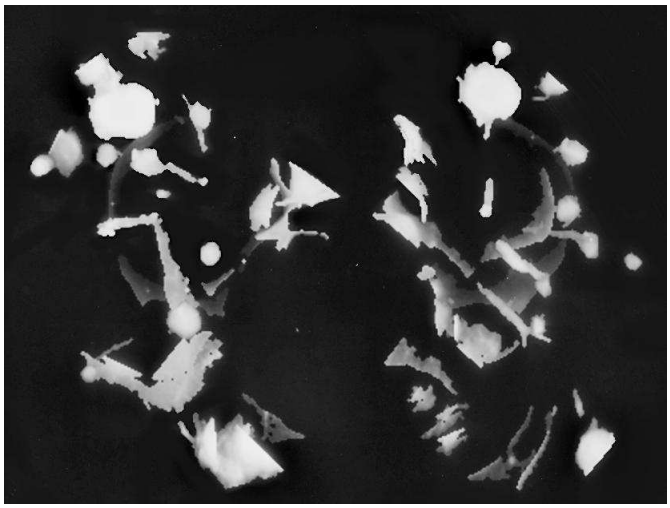

Figure 1.5: Segmentation of the lung by a fixed threshold [6]. The resulting structures to be classified appear in white.

The features consisted of geometric and gray level values. The geometric values computed are volume, sphericity, radius of the equivalent sphere, maximum compactness, maximum circularity, and maximum eccentricity, and the gray level values computed are mean intensity, standard deviation of the intensities, and the threshold at which the structure first decreases below the upper volume bound. Each of these nine-dimensional features is analyzed by a linear discriminant analysis (LDA) classifier [8]. Giger reported a detection rate of 82% with an approximate 9% false positive rate. Ko et al. [9], uses the same repeated fixed thresholding idea with similar results.

Gurcan et al. [10] use similar ideas. In this system, the lungs are first segmented using a fixed gray level threshold. Then, rather than doing multiple fixed thresholds, a weighted k-means clustering is performed on the voxel intensities. This segmentation technique is a form of the iterative approach to finding a threshold developed by Ridler and Calvard [11]. With the resulting structures, they perform a two-dimensional false positive reduction step. This rule-based system attempts to throw out vascular structures which have a long, skinny shape or are "V" shaped. Finally, an LDA classification is performed, similar to the one used by Giger et al. Their results were recorded

per slice. They reported an 84% detection rate, with a false positive rate of 1.74 false positives per slice.

The above systems were developed on thick-section CT scans. Due to the large slice thickness, most of the processing steps above were performed on the two-dimensional section. Thus the above methods are considered two-dimensional. As the CT scan technology has improved, thin-section CT images are now available. While this makes diagnosis of smaller nodules feasible, which is a more difficult task [12], it also makes possible the use of three-dimensional methods as there is more continuity from slice to slice. Outlined below are some of the three-dimensional methods created for thin-section CT scans.

Brown et al. [13] begin their approach by using a three-dimensional segmentation of the lung. To do so, they implement a combination of attenuation thresholding, region growing [7], and mathematical morphology [14]. These techniques are used in accordance with an anatomical model. First the chest wall is segmented, then, in this order, the mediastinum, lungs, lung opacities (nodules and vascular structures), and nodules. From here, three-dimensional shape information is recorded for each candidate. The shape information is processed in a scoring function. If this score is greater than a predefined amount, the structure is presented to the user as a nodule. Brown's paper reported a 78% detection rate (22 correct out of 22 for nodules with diameter 3mm and larger, 40 correct out of 57 for nodules with diameter less than 3mm). The false positive rate was recorded per subject, 15 on average.

While using a fixed threshold to identify the lung region, Zhao [12] segments structures inside the lung using the local density maximum (LDM) algorithm. The LDM algorithm combines structures identified using different thresholds. From the resulting structures, features are extracted. Zhao's work uses the following ratios as features: the ratio of volume of the object to volume of the smallest bounding box, the ratio of the projection length of the object along the z axis to the maximal projection length of the object along the x or y axis, whichever is larger, and finally the ratio of the maximal projection length of the object along x or y axis, whichever is larger, to the maximal

projection length of the object along the x or y axis, whichever is smaller. These features are designed to quantify how close the structure is to a perfect sphere, the shape in which a nodule usually appears. A rule based on these three features classifies the structure as a nodule to be reported, or a structure to be thrown out. The reported results were similar to the above methods, an 84% detection rate. (However, the data were computer generated.)

Many other works have been written on lung nodule detection, a few representations are [15], [16], [17], and [18] . As with the methods above, each of these starts out with a segmentation algorithm to locate structures, extracts features, and sends the features to a classifier. While this process has proven to be valuable, there are two aspects that may cause inaccurate results. First, the problem of segmenting out regions of interest is a difficult one. Some regions of interest may be grouped together as one structure, while some regions of interest may be completely missed during segmentation. That is, a segmentation step brings up one possible cause for error. Second, the classification step may hinder positive results as well. Classification relies on having a proper set of features, as well as knowledge of the distribution of these feature vectors. This is a big issue, and another cause for error in a computer aided diagnostics (CAD) system.

The work by Lee et al. [1] attempts to avoid the segmentation task. They reason that nodules resemble the density function of a Gaussian distributed random variable. A genetic algorithm thus searches through the image, matching their Gaussian template to similar structures in the image. The templates used by Lee are illustrated in figure 1.6. As with other methods above, the number of false positives is very large. They reported 258 false positives per case with their genetic algorithm. To overcome this large number of false positives, they extract features from the structures reported from the genetic algorithm, and feed them to a classifier. This step is quite similar to some of the above methods' feature extraction and classification processes. Using this reduction step, the number of false positives are reduced from 258 per case to 30 per case. They reported a detection rate of 72%.

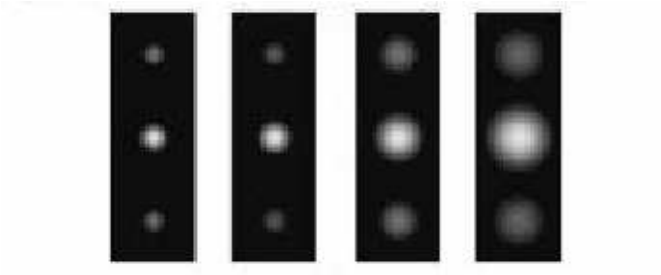

Figure 1.6: The different templates used by Lee et al. [1]

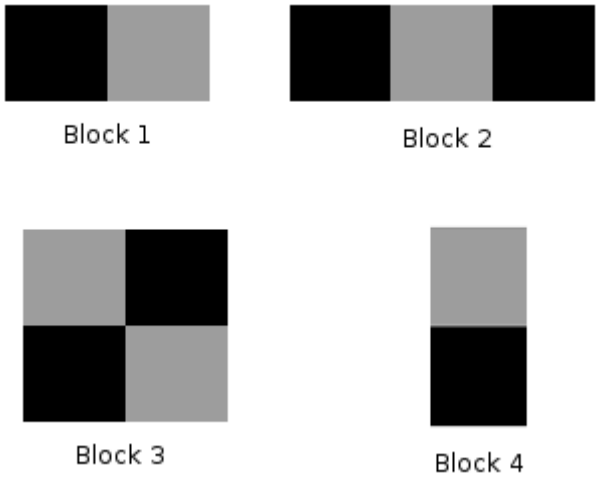

Figure 1.7: Features used in the Viola and Jones algorithm.

1.3 This Research

1.3.1 Motivation

This research is partly motivated by the work of Viola and Jones [19]. Viola and Jones created a face detection system that detects and locates faces in various backgrounds of two-dimensional images. Their method for rapid object detection uses a boosted cascade of simple features. The algorithm is described below.

The algorithm relies on features reminiscent of Haar basis functions. There are three types of features: two-rectangle, three-rectangle, and four-rectangle features. These are

illustrated in figure 1.7. The feature value is computed as the difference between the sum of the pixels in the black region and the sum of pixels in the white region. These features can be scaled to hold a different number of pixels, as long as the scaling is uniform.

Using a training set of images, a set of these features will be chosen using a variant of AdaBoost [20], a boosting algorithm by Freund and Schapire. Details of AdaBoost will be given in a later section. The training set of size N is a set of pairs $(\vec{x}_i, y_i)$, $\forall i \in \{1, \ldots, N\}$, where the $\vec{x}_i$ is an image reshaped as a vector and y_i is 1 if the image contains an instance of the object of interest and 0 otherwise. A weak learning algorithm selects a feature from the feature pool and trains a classifier based on the feature values of the training set. This is accomplished by iterating through the features. For each feature, the optimal values of a polarity, p_j, and a threshold, θ_j, are selected so that the following weak classifier,

$$h_j(\vec{x}_i) = \begin{cases} 1 & \text{if} \quad p_j f_j(x_i) < p_j \theta_j \\ 0 & \text{else} \end{cases}$$

performs the best on the training set. It is important to note that this set of features is complete. This means that the image can be completely represented by translations and scalings of the feature set in figure 1.7. Once the AdaBoost classifier is trained, it is possible to search through new images for the object of interest. To do this, a subwindow, the same size as the training set images, sweeps through the image. At each location, the subwindow is classified by the AdaBoost classifier.

In the Viola and Jones framework, there is no image segmentation invoked on the new image, only classification of subwindows. There is no human choice of features, only a learning algorithm choosing from a complete set of features. The result is an extremely robust system for the detection of objects in two-dimensional images.

1.3.2 Goal of this Research

The goal of this research is thus to eliminate the segmentation step, as well as the traditional method for feature extraction, that has been primarily used in previous

CAD algorithms. By eliminating the segmentation step, the possibility of skipping over structures that were not properly segmented is avoided. The problems associated with the particular segmentation algorithms (such as choice of threshold for thresholding, confidence level for region-growing, etc) will not be issues holding the system back.

In the CAD approaches outlined above, the feature extraction step consists of extracting information from the segmented structures. The information is not complete, it is simply the compilations of a few key quantities. These quantities are certainly useful, but do not completely represent the image. For example, Giger did not consider the intensities of adjacent pixels of each structure; perhaps these quantities are extremely important in classifying structures as nodules or as other lung opacities. Also, Lee chose only spherical features for detecting lung nodules. However, he overlooked the possibility of other features that help identify juxta-pleural and juxta-vascular nodules. One can continue to add features that help with the classification, but there will always be doubt that every key quantity is represented in the feature extraction. By eliminating the need to segment structures, the extraction of features from such structures is not necessary, and the chance of human error in choosing features from structures is eliminated.

With this in mind, this thesis will do the following.

1. Outline the suggested approach.
2. Explain the different type of features that were tested.
3. Explain the different classification methods that were tested.
4. Report the results from experiments on the various combinations of the features and classification methods.
5. Outline future directions for this research.

1.4 Outline of Approach: Training and Classifying

In the work of Viola and Jones, only faces that fit inside a 24 pixel by 24 pixel square box were considered. Although, this is not to say that they ignored faces bigger and smaller than this size. Indeed, detecting different sized faces was made possible by taking the final classifier and scaling it. For example, when looking for faces that fit inside a 12 pixel by 12 pixel square box, the Haar-like features were scaled to half the size and the location was the same relative to the size.

Lung nodules can be roughly anywhere from 3 mm to 30 mm in diameter. The images used in this paper use voxels with approximate size .75 mm by .75 mm by 1.25 mm, where 1.25 mm is the slice thickness. (There are minor differences in voxel size depending on the machine that captured the CT image; the experiments below assume they are the same.). Thus, the difference between 3 mm and 30 mm is more than 21 slices. This makes scaling the features to detect different sized objects a necessity.

Training only needs to be done on one size. Therefore, in this work, a $12 \times 12 \times 6$ voxel box will be used. This yields a size of about 9 mm $\times$ 9 mm $\times$ 7.5 mm, which is close to the dimensions of a cube. This is convenient since most nodules are roughly spherical; they will fit nicely in a cube shaped box.

Once a classifier is trained, classification proceeds as follows. A fixed sized subwindow starts in the top-upper-left corner and sweeps through the image to the bottom-lower-right corner. At each location, the classifier classifies the window as either having a nodule candidate, or not. Then, the size of the subwindow is changed, and the process repeats.

As mentioned above, once the feature pool is chosen, the features used by the classifier will be chosen by a machine learning algorithm. This will yield a set of features that best captures the difference between nodule and non-nodule windows.

1.4.1 Training the Features and Classifier

Training requires a good training set. In the end, the classifier will classify $\mathcal{X}$, the set of all $12 \times 12 \times 6$ subwindows, so the training set must consist of a set of pairs $(\vec{x}_i, y_i)$,

$\forall i \in \{1, \ldots, N\}$, where each $\vec{x}_i$ is an $12 \times 12 \times 6$ image and y_i is 1 if the image contains a nodule and 0 otherwise. Since features will be automatically learned from this set, it is very important that this set represents $\mathcal{X}$ well. The subwindows are $12 \cdot 12 \cdot 6 = 864$-dimensional. This means the training set must be quite large in order to learn how to classify 864-dimensional images. Further, it is important that the training data are randomly chosen from all parts of the lung. Doing so will make sure that the training data are an accurate approximation of the statistical distribution of all subwindows which are $12 \times 12 \times 6$.

Once the training set, $(\vec{x}_i, y_i)$, $\forall i \in \{1, \ldots, N\}$, is compiled, feature vectors can be computed. Choices of the types of features used will be discussed in a later section, but are learned from this data set. Regardless of the feature, there will be a function, Φ, that transforms each $\vec{x}_i$ into the M-dimensional feature vector $\vec{z}_i$.

The choice of classifier will also be discussed in a later section. Regardless of the choice, the classifier will be trained on, not $(\vec{x}_i, y_i)$, $\forall i \in \{1, \ldots, N\}$, but $(\vec{z}_i, y_i)$, $\forall i \in \{1, \ldots, N\}$. The result is a classifier $h : \mathcal{X}' \to \{0, 1\}$, where $\mathcal{X}' = \Phi(\mathcal{X})$.

1.4.2 Classifying Subwindows

After training is complete, it is possible to classify new images. Sweeping through the three-dimensional image, as discussed above, yields a set of $12 \times 12 \times 6$ subwindows available to be classified. The process is as follows:

For each subwindow $\vec{x}$:

1. Compute $\vec{z} = \Phi(\vec{x})$.
2. Compute $y = h(\vec{z})$.
3. Label window as a nodule if $y = 1$ or a non-nodule if $y = 0$.

Figures 1.8 and 1.9 illustrate the difference in the conceptual designs of this approach and the typical CAD approach for detecting lung nodules.

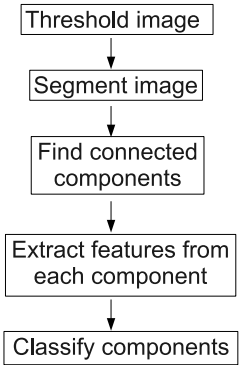

Figure 1.8: The flow chart for detecting lung nodules in a typical CAD system.

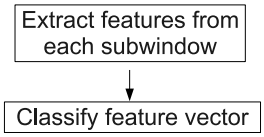

Figure 1.9: The flow chart for the lung nodule detection scheme suggested in this paper.

1.5 A Note on Features and Classification

1.5.1 Learning Features

In this work, features are learned rather than individually chosen based on human interpretation as mentioned above. In order to do this, one must start with a set of features that is complete. The information included in a complete set of features is the same as the information included in the original data itself. Using such a feature set guarantees no information will be mistakenly omitted.

Several complete feature sets will be analyzed in this paper. These include bases from matrix decomposition methods, a Fourier basis, and Haar-like features. The features will be discussed in detail in chapter 2.

When choosing what type of features to use, it is natural to look at the distribution of the data in feature space. There will be a distribution of feature vectors corresponding to subwindows with nodules present, and a separate distribution of feature vectors corresponding to subwindows without a nodule. While performance depends on the type of classifier used, distributions that appear far apart are generally easier to classify. Features that tend to separate these two distributions are usually a better choice than

features that do not.

Figure 1.10 shows one choice of a two-dimensional feature vector extracted from the data. A distinct separation of the two distributions is obvious, and a linear classifier will have an easy time separating them. These features are thus a good choice for discriminating between the two classes of subwindows – nodules and non-nodules. This is in contrast to the features used in figure 1.11. These two features yield no separation of the two classes. A linear classifier would certainly not be able to separate the two distributions, and there is little hope that a nonlinear classifier would do much better. Of course, the dimension of the feature vectors will usually be greater than four, making it difficult to analyze how the two classes are distributed.

1.5.2 Classifying

Once the features have been learned from the data, a classifier can be chosen. There are two types of classifiers, parametric and non-parametric. In this work, both are considered.

Parametric classifiers make the assumption that the data come from statistical distributions. These classifiers have the ability to perform much better than non-parametric classifiers if the assumptions on the distributions are correct. If these assumptions are incorrect, these methods can perform very poorly. Examples of parametric classifiers include likelihood methods and Probabilistic D-Clustering, which will be explained later.

Non-parametric classifiers make no assumptions on how the data are distributed. These methods are usually much more robust than parametric methods, performing better on data whose distribution is unknown. Examples of non-parametric classifiers include support vector machines, AdaBoost, and classification trees.

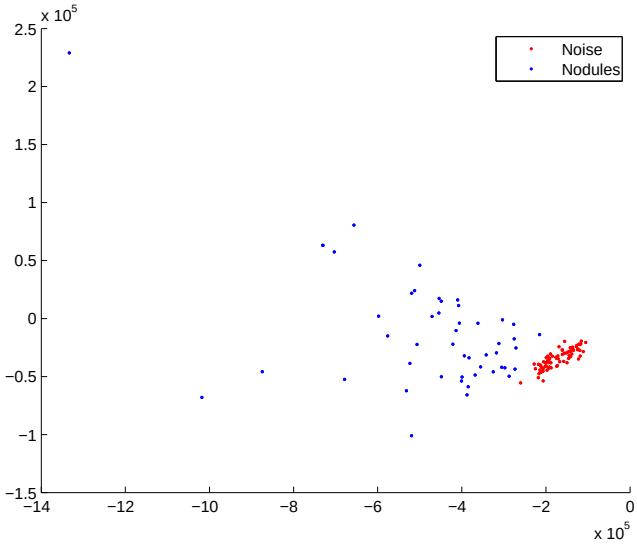

Figure 1.10: Example of a two-dimensional feature vector extracted from the data that brings about a good separation. Classifiers should have an easy time separating the two distributions.

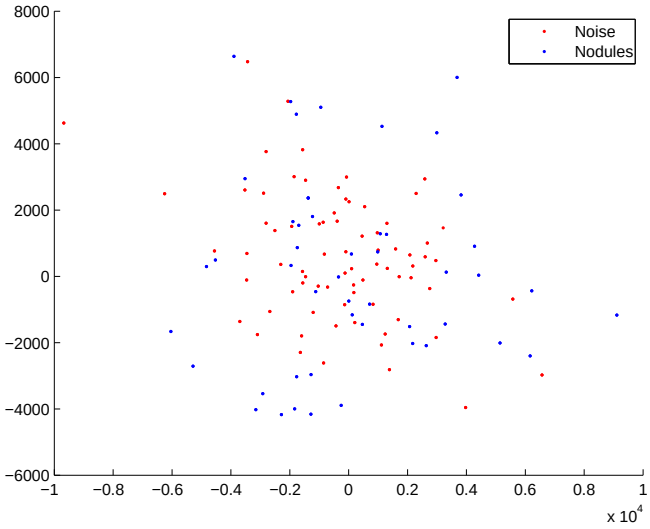

Figure 1.11: Example of a two-dimensional feature vector extracted from the data that brings about a poor separation. Separating the two distributions will be extremely difficult and probably impossible.

Chapter 2

Features

This work experiments with several types of features. In each case, features are chosen from a large pool of features in such a way that the two classes, nodules and non-nodules, will be as close to separable in the feature space as possible. The features explained below are a basis of vectors from a singular value decomposition (SVD), vectors from a modification of the SVD, vectors from a tensor decomposition, Haar-like features, and lower Fourier modes.

In every case, the features are complete. In other words, there is just as much information given in the set of all feature values as there is in the subwindow itself. (In order to make implementation practical, it may be necessary to use only a subset of these features to train the classifier.)

2.1 Left Singular Vectors

One way to represent N observations of P-dimensional training data, $\{\vec{x}_i\}_{i=1}^N$, is to create a matrix, X, whose columns are the vectors $\vec{x}_i$. That is,

$$X = \begin{bmatrix} | & | & \cdots & | \\ x_1 & x_2 & \cdots & x_N \\ | & | & \cdots & | \end{bmatrix}$$

This motivates the use of matrix decomposition methods to analyze the data.

2.1.1 Background on the Singular Value Decomposition

The singular value decomposition (SVD) is a very useful decomposition for matrices. It decomposes any real $P \times N$ matrix X into the product $U\Sigma V^{\mathrm{T}}$. Each matrix has special properties [21]:

- U is $P \times P$ matrix whose columns form an orthonormal basis for $\mathbb{R}^P$
- V is $N \times N$ matrix whose columns form an orthonormal basis for $\mathbb{R}^N$
- Σ is an $P \times N$ diagonal matrix with nonnegative real numbers on the diagonal.

It is useful to order the diagonal elements of Σ, called the singular values, in nonincreasing fashion. These are labeled σ_i, so that $\sigma_1 \geq \sigma_2 \geq \sigma_r$, where r is the rank of X. There are exactly r nonzero singular values in Σ. The columns of U, labeled $\vec{u}_i$, are called the left singular vectors, and the columns of V, labeled $\vec{v}_i$, are called the right singular vectors.

The SVD exists for every matrix regardless of dimensions and rank. In addition to existence, the SVD is unique in that the singular values, σ_i, are uniquely determined. Further, for each distinct σ_i, the left and right singular vectors $\vec{u}_i$ and $\vec{v}_i$ are uniquely determined.

While there are many interpretations of the SVD, the following motivates its use in extracting features from the training data. This interpretation involves viewing X as the following sum of r rank-one matrices.

$$X = \sum_{i=1}^{r} \sigma_i \vec{u}_i \vec{v}_i^{\mathrm{T}} \tag{2.1}$$

To see why this is useful, look at each term in the above sum:

$$\sigma_i \vec{u}_i \vec{v}_i^{\mathrm{T}} = \sigma_i \begin{bmatrix} | & | & & | \\ \vec{u}_i v_{i1} & \vec{u}_i v_{i2} & \cdots & \vec{u}_i v_{iN} \\ | & | & & | \end{bmatrix}$$

When all of these terms are added together, the matrix looks like the following.

$$X = U\Sigma V^{\mathrm{T}} = \begin{bmatrix} | & | & & | \\ \sum_{j=1}^{r} \sigma_j v_{j1} \vec{u}_j & \sum_{j=1}^{r} \sigma_j v_{j2} \vec{u}_j & \cdots & \sum_{j=1}^{r} \sigma_j v_{j2} \vec{u}_j \\ | & | & & | \end{bmatrix}$$

That is, every column in X is a weighted sum of the vectors $\vec{u}_i$. This is not surprising considering the fact that $\{u_i\}_{i=1}^{P}$ is a basis for $\mathbb{R}^P$. However, the following theorem, proven in [21], yields valuable information about the usefulness of each rank-one term in the sum (2.1).

Theorem 1. *For any integer r' with $0 < r' < r$, define*

$$X_{r'} = \sum_{i=1}^{r'} \sigma_i \vec{u}_i \vec{v}_i^{\,T}$$

Then,

$$||X - X_{r'}||_2 = \inf_{\substack{A \in \mathbb{R}^{P \times N} \\ \mathrm{rank}(A) \leq r'}} ||X - A||_2$$

In other words, $X_{r'}$ is the best rank-r' approximation to X in the sense of the matrix 2-norm.

This means, the best, rank-one approximation of X, in the 2-norm sense, is

$$\sigma_1 \vec{u}_1 \vec{v}_1^{\mathrm{T}} = \begin{bmatrix} | & | & & | \\ \sigma_1 \vec{u}_1 v_{11} & \sigma_1 \vec{u}_1 v_{12} & \cdots & \sigma_1 \vec{u}_1 v_{1N} \\ | & | & & | \end{bmatrix}$$

The best rank-two approximation of X, in the 2-norm sense, is

$$\sigma_1 \vec{u}_1 \vec{v}_1^{\mathrm{T}} + \sigma_2 \vec{u}_2 \vec{v}_2^{\mathrm{T}} = \begin{bmatrix} | & | & & | \\ \sigma_1 \vec{u}_1 v_{11} + \sigma_2 \vec{u}_2 v_{21} & \sigma_1 \vec{u}_1 v_{12} + \sigma_2 \vec{u}_2 v_{22} & \cdots & \sigma_1 \vec{u}_1 v_{1N} + \sigma_2 \vec{u}_2 v_{2N} \\ | & | & & | \end{bmatrix}$$

and so on.

Using this idea, the basis $\{\vec{u}_i\}_{i=1}^{P}$ can be used to extract a set of features from the columns of X. Since **Theorem 1** guarantees that the best rank-r' approximation to X uses linear combinations of $\vec{u}_1$, $\vec{u}_2$, ..., $\vec{u}_{r'}$, the coefficients of these first r' basis elements will be used as the best r' features.

To find out how these coefficients are determined, suppose X^* is the best rank-r' approximation to X, with $0 < r' < r$. Then, from the SVD, there exists $\{\sigma_i\}_{i=1}^{r'}$, $\{\vec{u}\}_{i=1}^{r'}$, and $\{\vec{v}\}_{i=1}^{r'}$ so that

$$X^* = \sum_{i=1}^{r'} \sigma_i \vec{u}_i \vec{v}_i^{\mathrm{T}}$$

This can be written in matrix form as

$$X^* = U^* \Sigma^* V^{*\mathrm{T}}$$

where

$$U^* = \begin{bmatrix} | & | & & | \\ \vec{u}_1 & \vec{u}_2 & \cdots & \vec{u}_{r'} \\ | & | & & | \end{bmatrix}$$

$$V^* = \begin{bmatrix} | & | & & | \\ \vec{v}_1 & \vec{v}_2 & \cdots & \vec{v}_{r'} \\ | & | & & | \end{bmatrix}$$

$$\Sigma^* = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \sigma_{r'} \end{bmatrix}$$

So

$$X^* = \begin{bmatrix} | & | & & | \\ \sum_{j=1}^{r'} \sigma_j v_{j1} \vec{u}_j & \sum_{j=1}^{r'} \sigma_j v_{j2} \vec{u}_j & \cdots & \sum_{j=1}^{r'} \sigma_j v_{j2} \vec{u}_j \\ | & | & & | \end{bmatrix} \tag{2.2}$$

Both sides of equation (2.2) can be multiplied on the left by $U^{*\mathrm{T}}$ to yield

$$U^{*\mathrm{T}} X^* = \Sigma^* V^{*\mathrm{T}}$$

So, we have

$$\Sigma^* {V^*}^{\mathrm{T}} = {U^*}^{\mathrm{T}} X^* = \begin{bmatrix} \vec{u}_1^T \vec{x}_1 & \vec{u}_1^T \vec{x}_2 & \cdots & \vec{u}_1^T \vec{x}_N \\ \vec{u}_2^T \vec{x}_1 & \vec{u}_2^T \vec{x}_2 & \cdots & \vec{u}_2^T \vec{x}_N \\ \vdots & \vdots & \ddots & \vdots \\ \vec{u}_{r'}^T \vec{x}_1 & \vec{u}_{r'}^T \vec{x}_2 & \cdots & \vec{u}_{r'}^T \vec{x}_N \end{bmatrix} \tag{2.3}$$

and also

$$\Sigma^* {V^*}^{\mathrm{T}} = \begin{bmatrix} \sigma_1 v_{11} & \sigma_1 v_{12} & \cdots & \sigma_1 v_{1N} \\ \sigma_2 v_{21} & \sigma_2 v_{22} & \cdots & \sigma_2 v_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{r'} v_{r'1} & \sigma_{r'} v_{r'2} & \cdots & \sigma_{r'} v_{r'N} \end{bmatrix} \tag{2.4}$$

From mth column of the matrix in equation (2.2), the coefficient of $\vec{u}_i$ is $\sigma_i v_{im}$. Equating equations (2.3) and (2.4), yields the fact that the coefficient of $\vec{u}_i$ in the mth column of (2.2) is $\vec{u}_i^T \vec{x}_m$. Again, since $\{\vec{u}\}_{i=1}^{r'}$ spans the column space of X^*, it is not surprising that the coefficient of $\vec{u}_i$ in the mth column of X^* is the inner product of $\vec{u}_i$ with the mth column of X, $\vec{x}_i$.

2.1.2 Application to Lung Nodule Detection

Now consider the task at hand, separating subwindows which contain nodules from the ones that do not contain nodules. The set of all subwindows containing nodules have a white spherical blob in the center. The set of all subwindows which do not contain nodules should have an average subwindow which looks like low intensity noise. Indeed, the two mean subwindows from the data set used in this paper reflect this reasoning and are shown in figures 2.1 and 2.2.

With this in mind, one method to learn features for classification using the SVD is to learn the basis for U on the positive subwindows. The first few left singular vectors should describe the nodules well, and thus should create a clear separation between subwindows containing nodules and subwindows not containing nodules. The first left singular vectors are shown in figures 2.3 and 2.4. The results of this approach will be given in chapter 4.

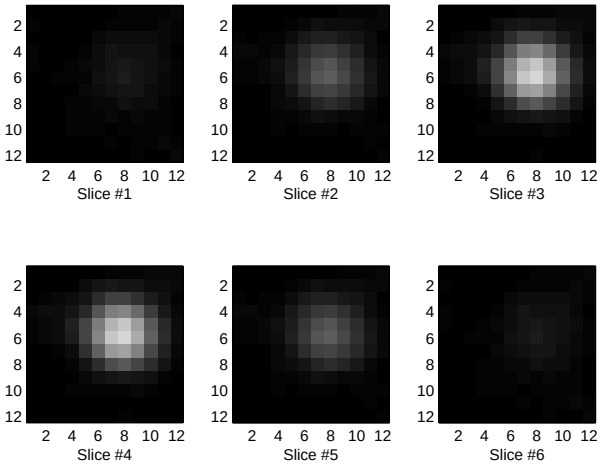

Figure 2.1: The average subwindow from the set of subwindows in the training set with nodules present.

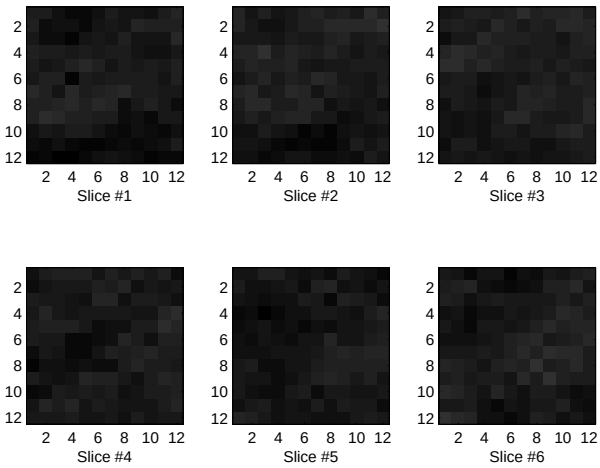

Figure 2.2: The average subwindow from the set of subwindows in the training set with no nodules present. This set does include subwindows with vessels present, but their random locations prevent them from being visible in the mean subwindow.

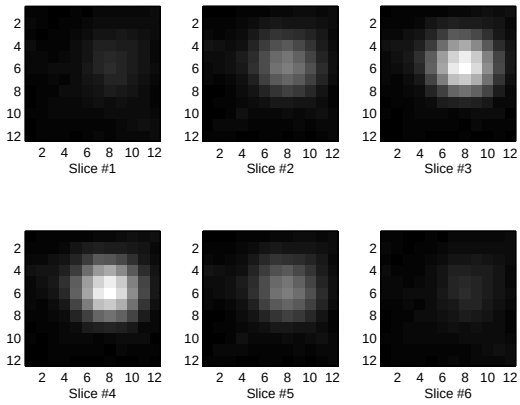

Figure 2.3: The first basis vector found by the SVD of the matrix with columns made up of only subwindows containing nodules. Since the vector has norm 1, it has been shifted and rescaled to resemble the intensities of a normal subwindow (1 to 256). This should be compared to the mean subwindow from figure 2.1.

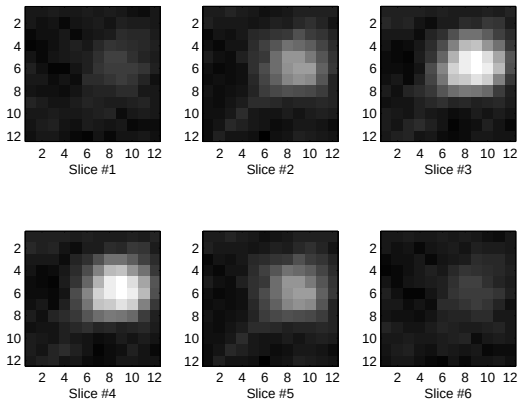

Figure 2.4: The second basis vector found by the SVD of the matrix with columns made up of only subwindows containing nodules. Again, it has been shifted and rescaled.

2.1.3 Relation to Principal Component Analysis

This method of learning features is very similar to a popular approach called principal component analysis (PCA). PCA determines the independent linear combinations

of vector elements that create the largest empirical variance of the resulting data. In this way, possibly correlated, high-dimensional variables are transformed into uncorrelated low-dimensional variables. In the same way that the first basis element of the SVD explains the data the best, the first linear combination, called the first principal component, captures as much variability as possible. The second captures as much of the remaining variability as possible, and so on.

Suppose the mean has been subtracted from the columns of the data matrix, X, so that the set of all columns has zero mean. Let $\vec{w}$ contain the desired linear combination of the variables in the columns of X. Since the empirical variance of a zero-mean sample of M observations in the columns of matrix Y is given by $\frac{1}{M-1}YY^T$, it is clear that $\vec{w}$ solves the following problem.

$$\max_{||\vec{w}||_2=1} \vec{w}^T X X^T \vec{w} \tag{2.5}$$

Again, let the SVD of X be $U\Sigma V^T$. This yields

$$\begin{aligned}
(2.5) &= \max_{||\vec{w}||_2=1} \vec{w}^T U\Sigma V^T V\Sigma^T U^T \vec{w} \\
&= \max_{||\vec{w}||_2=1} \vec{w}^T U\Sigma^2 U^T \vec{w} \\
&= \max_{||\vec{w}||_2=1} ||\Sigma U^T \vec{w}||_2^2 \\
&= \max_{||\vec{w}||_2=1} \left|\left| \begin{bmatrix} — & \sigma_1\vec{u}_1^T & — \\ — & \sigma_2\vec{u}_2^T & — \\ & \vdots & \\ — & \sigma_M\vec{u}_M^T & — \end{bmatrix} \vec{w} \right|\right|_2^2
\end{aligned}$$

Since $\{\vec{u}_i\}_{i=1}^M$ is an orthonormal set, $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_r > 0$, $\vec{w} = \vec{u}_1$ maximizes this expression. Similarly, the vector that captures the remaining variability is $\vec{w} = \vec{u}_2$, and so on.

While this is almost identical to the SVD approach described above, there is a slight difference. The SVD approach above did not subtract the mean of the samples $\{\vec{x}_i\}_{i=1}^N$. Consider the first component found by the SVD of the matrix with columns made up of only subwindows containing nodules shown in figure 2.3. Since the mean explains

every subwindow quite well, it is no surprise that the most influential basis element is virtually identical to the mean, which was shown in figure 2.1.

2.1.4 A Note on the Use of PCA in this Paper

PCA is a very convenient way to reduce dimension of data. In most of the feature extraction techniques described in this paper, feature vectors have dimension larger than three. This makes visualizing the data difficult, as it is only possible to see three dimensions at a time. Thus PCA can be used to reduce many dimensions to two or three for visualization purposes, and is commonly used in this paper to see if there is any separation in feature space. While separated clusters of data in reduced space indicate separation in high-dimensional space, non-separated clusters in reduced space *does not* necessarily indicate that the two sets of data are not separable in high-dimensional space. This should be considered when looking at plots of data in reduced space.

2.2 Modification of the SVD and Fisher's Discriminant

The SVD method to extract features explained above works well in describing the subwindows that contain nodules. The group of subwindows that do not contain nodules should have similar feature values. These feature values should be different from those of the group of subwindows that contain nodules. However, by not training the features on the non-nodule subwindows in addition to the subwindows with nodules present, all the information contained in the data set is not utilized. Using the entire training set may bring about better features, and thus better performance.

Consider the goal of PCA. PCA aims to find a linear combination of the variables to maximize the variance in the resulting set of linear combinations. The reason that both classes of subwindows, nodules and non-nodules, cannot be used is that this method views that data as only one class. The goal of PCA is accomplished regardless of how the two classes are separated in the resulting principal component space.

Instead of maximizing the variance of a *single* class, the goal should be to bring about maximum separation between the *two* classes. Call $\vec{w}$ the desired principal component,

X_1 the matrix with the N_1 nodule subwindows as its columns, X_2 the matrix with the N_2 non-nodule subwindows as its columns, and $\vec{x}_{\mu_i}$ the mean of the columns of X_i. One way to accomplish this goal is to maximize the separation of the two means. Mathematically, this is

$$\max_{||\vec{w}||=1} \left|\left|\vec{w}^T\vec{x}_{\mu_1} - \vec{w}^T\vec{x}_{\mu_2})\right|\right|_2^2$$

which is identical to

$$\max_{||\vec{w}||=1} \left|\left|\vec{w}^T(\vec{x}_{\mu_1} - \vec{x}_{\mu_2})\right|\right|_2^2$$

However, separating the means does not help much if the variances of the respective classes are large. Thus the cost function should be modified to make sure the variances of the two classes are small. Let the variance of the set of nodule subwindows be called $S_{X_1}^2$ and the variance of the set of non-nodule subwindows be called $S_{X_2}^2$. i.e.

$$S_{X_i}^2 = \frac{1}{N_i - 1}\sum_{j=1}^{N_i}(\vec{x}_j^{(i)} - \vec{x}_{\mu_i})(\vec{x}_j^{(i)} - \vec{x}_{\mu_i})^T$$

where $\vec{x}_j^{(i)}$ is the jth column of X_i. Then the new optimization problem should be the following.

$$\max_{||\vec{w}||=1} \left|\left|\vec{w}^T(\vec{x}_{\mu_1} - \vec{x}_{\mu_2})\right|\right|_2^2 - \frac{1}{N_1 - 1}\sum_{j=1}^{N_1}(\vec{w}^T\vec{x}_j^{(1)} - \vec{w}^T\vec{x}_{\mu_1})^2 - \frac{1}{N_2 - 1}\sum_{j=1}^{N_2}(\vec{w}^T\vec{x}_j^{(2)} - \vec{w}^T\vec{x}_{\mu_2})^2$$

This can be rewritten as

$$\max_{||\vec{w}||=1} \left|\left|\vec{w}^T(\vec{x}_{\mu_1} - \vec{x}_{\mu_2})\right|\right|_2^2 - \frac{1}{N_1 - 1}\sum_{j=1}^{N_1}\vec{w}^T(\vec{x}_j^{(1)} - \vec{x}_{\mu_1})(\vec{x}_j^{(1)} - \vec{x}_{\mu_1})^T\vec{w}$$

$$- \frac{1}{N_2 - 1}\sum_{j=1}^{N_2}\vec{w}^T(\vec{x}_j^{(2)} - \vec{x}_{\mu_2})(\vec{x}_j^{(2)} - \vec{x}_{\mu_2})^T\vec{w}$$

which can be factored into the following.

$$\max_{||\vec{w}||=1} \left|\left|\vec{w}^T(\vec{x}_{\mu_1} - \vec{x}_{\mu_2})\right|\right|_2^2 - \vec{w}^T \left(\frac{1}{N_1 - 1} \sum_{j=1}^{N_1} (\vec{x}_j^{(1)} - \vec{x}_{\mu_1})(\vec{x}_j^{(1)} - \vec{x}_{\mu_1})^T \right) \vec{w}$$

$$-\vec{w}^T \left(\frac{1}{N_1 - 1} \sum_{j=1}^{N_1} (\vec{x}_j^{(2)} - \vec{x}_{\mu_1})(\vec{x}_j^{(2)} - \vec{x}_{\mu_1})^T \right) \vec{w} \tag{2.6}$$

After identifying the covariance matrices, $S_{X_1}^2$ and $S_{X_2}^2$, (2.6) becomes

$$\max_{||\vec{w}||=1} \left|\left|\vec{w}^T(\vec{x}_{\mu_1} - \vec{x}_{\mu_2})\right|\right|_2^2 - \vec{w}^T S_{X_1}^2 \vec{w} - \vec{w}^T S_{X_2}^2 \vec{w}$$

which can be written as

$$\max_{||\vec{w}||=1} \vec{w}^T \left((\vec{x}_{\mu_1} - \vec{x}_{\mu_2})(\vec{x}_{\mu_1} - \vec{x}_{\mu_2})^T \right) \vec{w} - \vec{w}^T S_{X_1}^2 \vec{w} - \vec{w}^T S_{X_2}^2 \vec{w}$$

Factoring out $\vec{w}$ twice yields

$$\max_{||\vec{w}||=1} \vec{w}^T \left((\vec{x}_{\mu_1} - \vec{x}_{\mu_2})(\vec{x}_{\mu_1} - \vec{x}_{\mu_2})^T - S_{X_1}^2 - S_{X_2}^2 \right) \vec{w}$$

This is equivalent to

$$\max_{||\vec{w}||=1} \vec{w}^T H \vec{w} \tag{2.7}$$

where

$$H = (\vec{x}_{\mu_1} - \vec{x}_{\mu_2})(\vec{x}_{\mu_1} - \vec{x}_{\mu_2})^T - S_{X_1}^2 - S_{X_2}^2$$

The matrix H is the sum of symmetric matrices, so is also symmetric. Thus $H^{\frac{1}{2}}$ is well-defined and this yields the following equivalent problem

$$(2.7) \iff \max_{||\vec{w}||=1} \left|\left| H^{\frac{1}{2}} \vec{w} \right|\right|_2^2$$

This is clearly optimized by taking $\vec{w}$ to be the eigenvector corresponding to the largest eigenvalue of $H^{1/2} = U \Lambda U^T$, which is the same as the eigenvector corresponding to the

largest eigenvalue of $H = U\Lambda^2 U^T$. Further, the next best uncorrelated $\vec{w}$ is given by the eigenvector corresponding to the second largest eigenvalue, and so on. This means, to find the linear combination that best separates the classes and minimizes their respective variances, all one has to do is compute the SVD of H and take the left singular vectors, with the corresponding singular values giving the importance of the respective vector.

This can even be improved by weighing the importance of each term in H. Define

$$H(\lambda, \xi) = (\vec{x}_{\mu_1} - \vec{x}_{\mu_2})(\vec{x}_{\mu_1} - \vec{x}_{\mu_2})^T - \lambda S_{X_1}^2 - \xi S_{X_2}^2 \tag{2.8}$$

Then, replace H in equation (2.7) with $H(\lambda, \xi)$. The parameters λ and ξ control the importance of the three terms in H. There is no need for a third parameter since it only would rescale the entire expression. The best choice of the parameters can be learned using a cross-validation scheme. Cross-validation will be discussed in chapter 4.

In figures 2.5 and 2.6, the top two vectors have been reconstructed as a subwindow. Figures 2.7 and 2.8 show the greater separation obtained by the modified SVD method in the space of the first two respective features. Results of classification in the two feature spaces will be compared in chapter 4.

2.2.1 The Related Fisher's Linear Discriminant

The method explained above is very similar to a traditional approach to maximize distance between classes and minimize variance within classes called Fisher's Linear Discriminant. Define Fisher's Linear Discriminant as

$$S_{\text{Fisher}} = \frac{\vec{w}^T \Sigma_B \vec{w}}{\vec{w}^T \Sigma_W \vec{w}}$$

where

$$\Sigma_B = (\vec{x}_{\mu_1} - \vec{x}_{\mu_2})(\vec{x}_{\mu_1} - \vec{x}_{\mu_2})^T$$

and

$$\Sigma_W = S_{X_1}^2 + S_{X_2}^2$$

are the between class covariance and within class covariance, respectively. Then Fisher's method chooses $\vec{w}$ with $||\vec{w}|| = 1$ as to maximize Fisher's Linear Discriminant. Maximizing the discriminant effectively maximizes the between class variance (numerator)

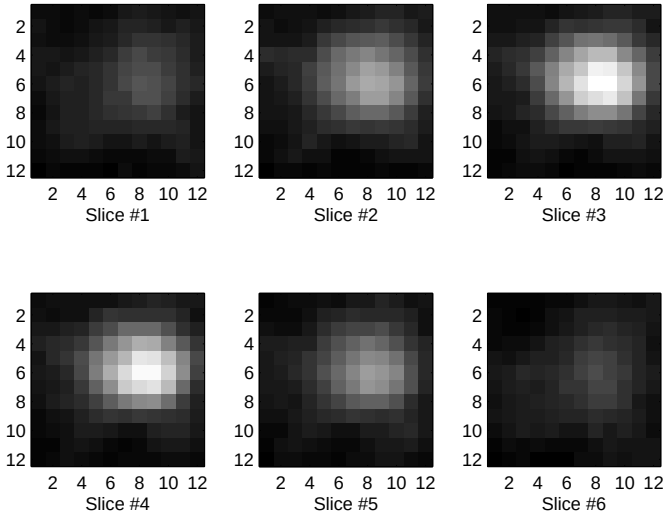

Figure 2.5: .

The first vector chosen by the modified SVD method discussed above.

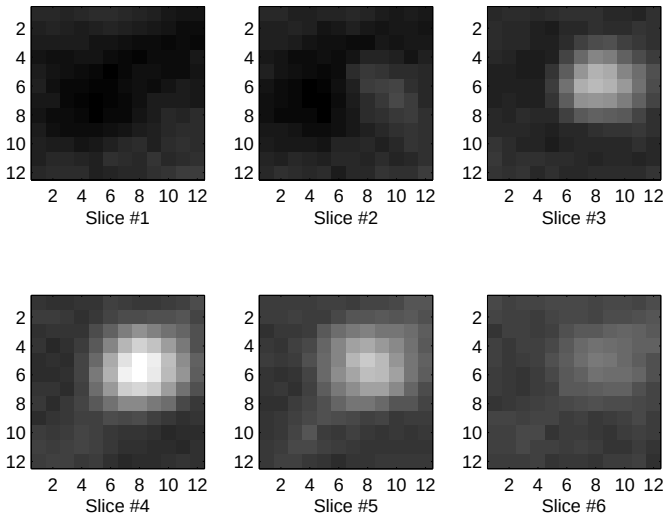

Figure 2.6: The second vector chosen by the modified SVD method discussed above.

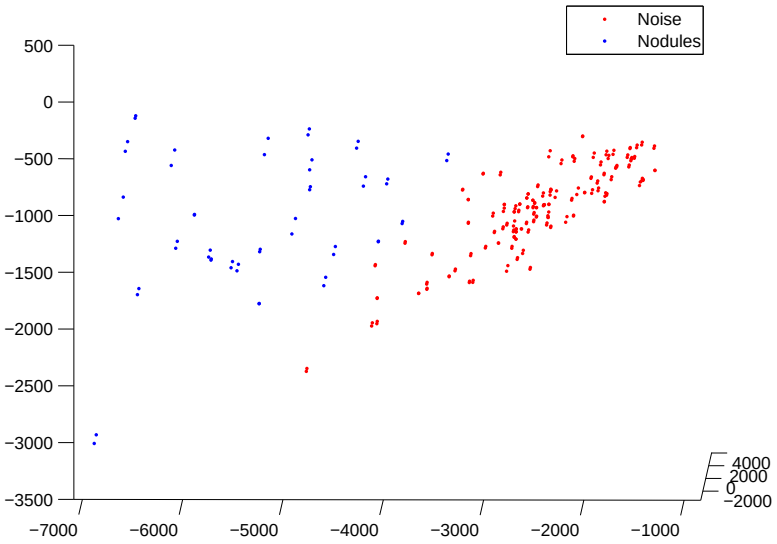

Figure 2.7: The first two features from the SVD of the nodule subwindows matrix.

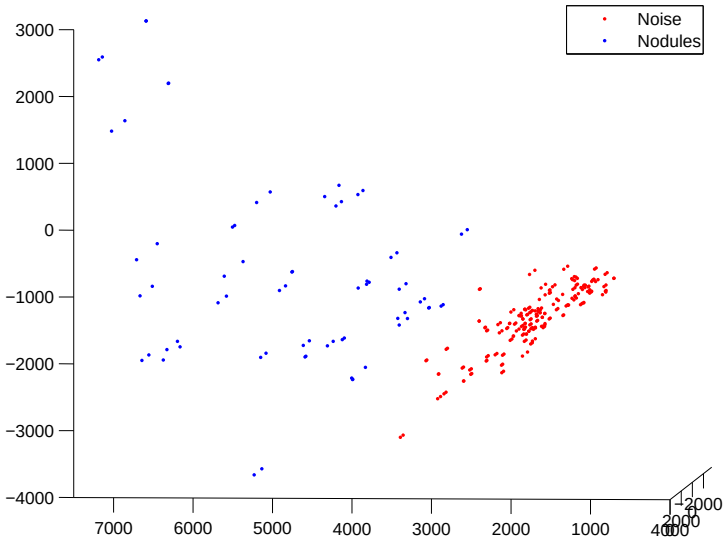

Figure 2.8: The first two features from the modified SVD of the nodule subwindows matrix.

while minimizing the within class variances (denominator).

Since $||\vec{w}|| = 1$ is a compact set, S_{Fisher} achieves its supremum over this set. Call the maximum M. Then the desired $\vec{w}$ satisfies

$$M = \frac{\vec{w}^T \Sigma_B \vec{w}}{\vec{w}^T \Sigma_W \vec{w}}$$

Multiplying by the denominator on both sides yields

$$\vec{w}^T \Sigma_B \vec{w} = M \vec{w}^T \Sigma_W \vec{w}$$

which, after a factorization, becomes

$$\vec{w}^T (\Sigma_B - M\Sigma_W) \vec{w} = 0$$

Since M is the largest value such that this equality holds, the matrix $(\Sigma_B - M\Sigma_W)$ is necessarily non-positive definite. This means that $\vec{w}$ satisfies

$$\underset{||\vec{w}||=1}{\text{argmax}} \; \vec{w}^T (\Sigma_B - M\Sigma_W) \vec{w}$$

which uses the same objective function given by equation (2.8) with fixed weights on the covariance matrices. That is, cross-validation on the parameters in equation (2.8) should yield a performance that is at least as good as the traditional Fisher's method.

2.3 Tensor Decompositions

With the SVD performing quite well, perhaps a generalization from two-dimensional matrix decompositions to three-dimensional tensor decompositions would work even better. A tensor is a multidimensional array. It is worth looking into tensor methods since a stack of $12 \times 12 \times 6$ subwindows can be considered a four-dimensional array.

First, some notation and preliminary tools should be introduced. A tensor will be denoted by a cursive bold capital letter. Suppose $\boldsymbol{\mathcal{X}}$ is a three-way tensor. Then $x_{i,j,k}$ denotes element (i, j, k) of the tensor, where i denotes the ith element in the first mode, j denotes the jth element in the second mode, etc. A colon will be used to denote all

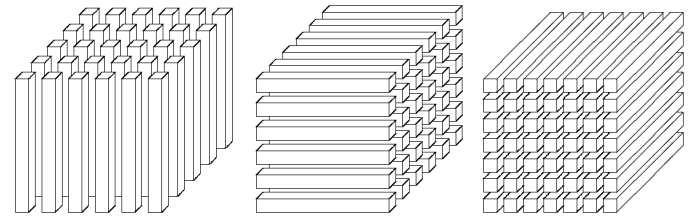

Figure 2.9: All the fibers for a three-way tensor.

the entries in the respective mode, e.g. if the second mode has J elements in it,

$$x_{i,:,k} = \begin{bmatrix} x_{i,1,k} \\ x_{i,2,k} \\ \vdots \\ x_{i,J,k} \end{bmatrix}$$

This $x_{i,:,k}$ is called a fiber. Fibers are a higher order analogue of a matrix row or column. Figure 2.9 shows all the fibers for a three-way tensor.

The inner product of two similar sized, P-way tensors, $\langle \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}} \rangle$, is defined as

$$\langle \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}} \rangle = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_P=1}^{I_P} x_{i_1,i_2,\cdots,i_P} y_{i_1,i_2,\cdots,i_P}$$

A P-way tensor, $\boldsymbol{\mathcal{X}}$, is rank one if it can be written as an outer product of P-vectors. i.e,

$$\boldsymbol{\mathcal{X}} = \vec{a}_1 \circ \vec{a}_2 \circ \cdots \circ \vec{a}_P \tag{2.9}$$

Here, "$\circ$" represents the vector outer product. For example, in the outer product (2.9), element $(i_1, i_2, ..., i_P)$ is as follows.

$$x_{i_1,i_2,...,i_P} = a_{1,i_1} a_{2,i_2} \cdots a_{P,i_P}$$

where $a_{i,j}$ represents the jth entry of vector $\vec{a}_i$

The p-mode product of a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_P}$ with a matrix $U \in \mathbb{R}^{J \times I_p}$ is denoted $\boldsymbol{\mathcal{X}} \times_p U$. It has dimensions $I_1 \times I_2 \times \cdots \times I_{p-1} \times J \times I_{p+1} \times \cdots \times I_P$. The formula for each element is as follows.

$$(\boldsymbol{\mathcal{X}} \times_p U)_{i_1,i_2,\ldots,i_{p-1},j,i_p+1,\ldots,i_P} = \sum_{i_p=1}^{I_p} x_{i_1,i_2,\ldots,i_P} u_{j,i_p}$$

The CP decomposition was studied by several authors. The name "CP" was coined by Kolda and Bader [22], who combined the names CANDECOMP and PARAFAC, two independent developments of the CP decomposition. This decomposition factorizes a tensor, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_P}$, into a sum of rank-one tensors.

$$\boldsymbol{\mathcal{X}} \approx \sum_{j=1}^{R} \vec{a}_j^{(1)} \circ \vec{a}_j^{(2)} \circ \cdots \circ \vec{a}_j^{(P)}$$

where $\vec{a}_j^{(i)} \in \mathbb{R}^{I_i}\ \forall i$. Each of these rank-one tensors that make up the summands are $I_1 \times I_2 \times \cdots \times I_P$ P-way tensors. Figure 2.10 illustrates a three-way rank-one tensor.

The rank of a tensor, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_P}$, is defined as the smallest number of rank-one tensors whose sum is $\boldsymbol{\mathcal{X}}$. The rank of $\boldsymbol{\mathcal{X}}$ is denoted rank($\boldsymbol{\mathcal{X}}$). More precisely,

$$\text{rank}(\boldsymbol{\mathcal{X}}) = \min\{R \mid \boldsymbol{\mathcal{X}} = \sum_{j=1}^{R} \vec{a}_j^{(1)} \circ \vec{a}_j^{(2)} \circ \cdots \circ \vec{a}_j^{(P)}, \text{ where } \vec{a}_j^{(i)} \in \mathbb{R}^{I_i}\ \forall i\}$$

An exact CP decomposition, $\boldsymbol{\mathcal{X}} = \sum_{j=1}^{R} \vec{a}_j^{(1)} \circ \vec{a}_j^{(2)} \circ \cdots \circ \vec{a}_j^{(P)}$, with $R = \text{rank}(\boldsymbol{\mathcal{X}})$ is called the rank decomposition.

The Tucker decomposition is another method of decomposing tensors [23]. Like the CP decomposition, this method decomposes a tensor into a core tensor multiplied

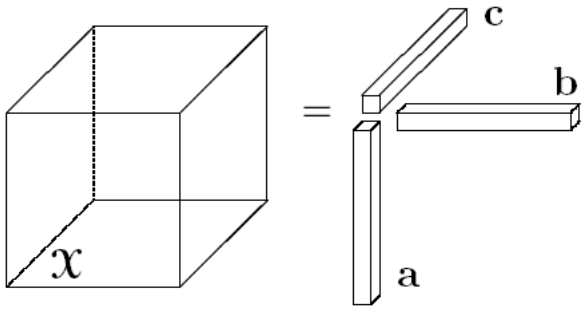

Figure 2.10: A three-way rank-one tensor. It is the outer product of three vectors. i.e. $\vec{a} \circ \vec{b} \circ \vec{c}$

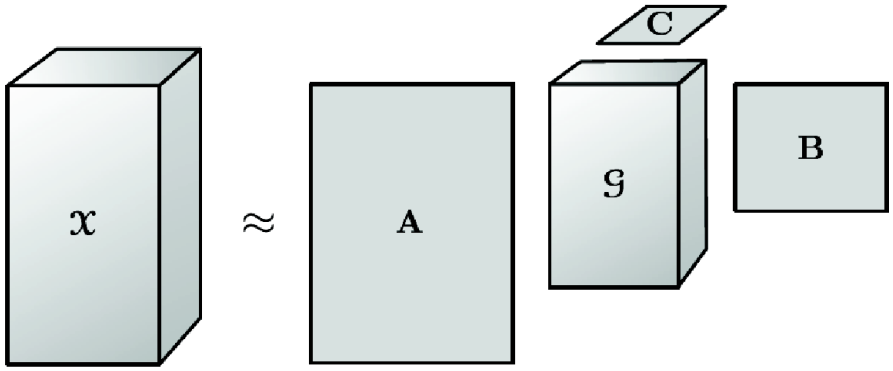

Figure 2.11: Tuckers decomposition on a three-way tensor. Placement of the factor matrices represents the mode in which the respective matrix is multiplied. [22]

by a two-dimensional matrix along each mode. For example, in the three-way case, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$,

$$\boldsymbol{\mathcal{X}} \approx \boldsymbol{\mathcal{G}} \times_1 A \times_2 B \times_3 C$$

where $A \in \mathbb{R}^{I \times Q}$, $B \in \mathbb{R}^{J \times R}$, and $C \in \mathbb{R}^{K \times S}$. $\boldsymbol{\mathcal{G}}$ is called the core tensor and the three matrices A, B, and C are the factor matrices. With the Tucker decomposition, the factor matrices are usually orthogonal. (This is not true with the CP decomposition.) The Tucker decomposition is illustrated in figure 2.11 for the three-way case. The P-way products can be expanded as the sum of outer products.

$$\boldsymbol{\mathcal{X}} = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_P=1}^{I_P} \boldsymbol{\mathcal{G}}_{i_1,i_2,\ldots,i_P} \vec{a}_{i_1}^{(1)} \circ \vec{a}_{i_2}^{(2)} \circ \cdots \circ \vec{a}_{i_P 1}^{(P)} \tag{2.10}$$

The Tucker decomposition is a higher-order principle component analysis. The columns of the factor matrices can be thought of as principle components in each mode. The core tensor shows the level of interaction between the different components.

As remarked earlier, a set of N, $12 \times 12 \times 6$ subwindows can be written as a four-way tensor having dimensions $12 \times 12 \times 6 \times N$. It can also be written using formula (2.10) as a sum of outer products. The following illustrates this for a four-way tensor with core

matrix $\boldsymbol{\mathcal{G}}$ and orthogonal factor matrices A, B, C, and D.

$$\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{G}} \times_1 A \times_2 B \times_3 C \times_4 D \tag{2.11}$$

$$\boldsymbol{\mathcal{X}} = \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{k=1}^{K} \sum_{l=1}^{L} \boldsymbol{\mathcal{G}}_{i,j,k,l} \vec{a}_i \circ \vec{b}_j \circ \vec{c}_k \circ \vec{d}_l \tag{2.12}$$

where $\vec{a}_i$ is the ith column of matrix A, and similarly for B, C, and D. Grouping the first few factors in the summand yields

$$\boldsymbol{\mathcal{X}} = \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{k=1}^{K} \sum_{l=1}^{L} \boldsymbol{\mathcal{G}}_{i,j,k,l} (\vec{a}_i \circ \vec{b}_j \circ \vec{c}_k) \circ \vec{d}_l \tag{2.13}$$

This should be compared with equation (2.2), where the SVD of a matrix is rewritten with each column being represented by a linear combination of the basis elements, $\{\vec{u}_i\}_{i=1}^{R}$. Equations (2.12) and (2.13) are similarly viewed as a linear combination of the following three-way tensor basis elements in each mode.

$$\{\vec{a}_i \circ \vec{b}_j \circ \vec{c}_k \mid \forall i = 1, 2, ..., I,\ j = 1, 2, ..., J,\ k = 1, 2, ..., K\}$$

The coefficient of basis element (i, j, k) for the qth element of $\boldsymbol{\mathcal{X}}$ is

$$\sum_{l=1}^{L} \boldsymbol{\mathcal{G}}_{i,j,k,l} d_{q,l}$$

which is the entry located at (i, j, k, q) in the tensor $\boldsymbol{\mathcal{G}} \times_4 D$.

The matrix $\boldsymbol{\mathcal{G}} \times_4 D$ can be found quite easily, given the following property of the P-way product for properly sized matrices A_1 and A_2.

$$\boldsymbol{\mathcal{X}} \times_i A_1 \times_i A_2 = \boldsymbol{\mathcal{X}} \times_i (A_2 A_1)$$

Since A is orthogonal, equation (2.11) yields

$$\boldsymbol{\mathcal{X}} \times_1 A^T = \boldsymbol{\mathcal{G}} \times_1 A \times_2 B \times_3 C \times_4 D \times_1 A^T$$

$$= \boldsymbol{\mathcal{G}} \times_1 (A^T A) \times_2 B \times_3 C \times_4 D$$

$$= \boldsymbol{\mathcal{G}} \times_1 I \times_2 B \times_3 C \times_4 D$$

$$= \boldsymbol{\mathcal{G}} \times_2 B \times_3 C \times_4 D$$

So $\boldsymbol{\mathcal{X}} \times_1 A^T = \boldsymbol{\mathcal{G}} \times_2 B \times_3 C \times_4 D$. Similarly,

$$\boldsymbol{\mathcal{X}} \times_1 A^T \times_2 B^T \times_3 C^T = \boldsymbol{\mathcal{G}} \times_4 D$$

Thus the coefficient of basis element $\vec{a}_i \circ \vec{b}_j \circ \vec{c}_k$ for the qth element of $\boldsymbol{\mathcal{X}}$ is the entry located at (i, j, k, q) in the tensor

$$\boldsymbol{\mathcal{X}} \times_1 A^T \times_2 B^T \times_3 C^T$$

Now, the same idea to extract features with the SVD can be applied with the Tucker decomposition. First a classifier can be trained on some or all of the basis elements $\{\vec{a}_i \circ \vec{b}_j \circ \vec{c}_k \mid \forall i = 1, 2, ..., I,\ j = 1, 2, ..., J,\ k = 1, 2, ..., K\}$, using the feature values from the tensor $\boldsymbol{\mathcal{X}} \times_1 A^T \times_2 B^T \times_3 C^T$. (Each fixed index in mode-four corresponds to one training case.) Then, classifying a three-way tensor, $\boldsymbol{\mathcal{Y}}$, is accomplished by computing

$$\boldsymbol{\mathcal{Y}} \times_1 A^T \times_2 B^T \times_3 C^T$$

and feeding this tensor to a classifier.

With the SVD, the singular values represent the importance of the corresponding basis element in representing the training data. With the Tucker decomposition, however, there is no such property. In fact, the core tensor is usually not even super diagonal. (A super diagonal tensor is one such that every tensor element, $x_{i_1,i_2,...,i_P}$, is zero except possibly when $i_1 = i + 2 = \cdots = i_P$.) Because of this, there is no easy way to order the basis elements by importance. The basis elements, $\vec{a}_1 \circ \vec{b}_1 \circ \vec{c}_1$, $\vec{a}_2 \circ \vec{b}_1 \circ \vec{c}_1$, $\vec{a}_1 \circ \vec{b}_2 \circ \vec{c}_1$, and $\vec{a}_1 \circ \vec{b}_1 \circ \vec{c}_2$ can be seen in figures 2.12, 2.13, 2.14, and 2.15.

2.4 Haar-like Features

These features are a generalization and modification of the Haar basis functions. They are again inspired by the work of Viola and Jones. After a brief introduction to the Haar wavelets, Viola and Jones' two-dimensional features are described. The features can then be generalized to three dimensions.

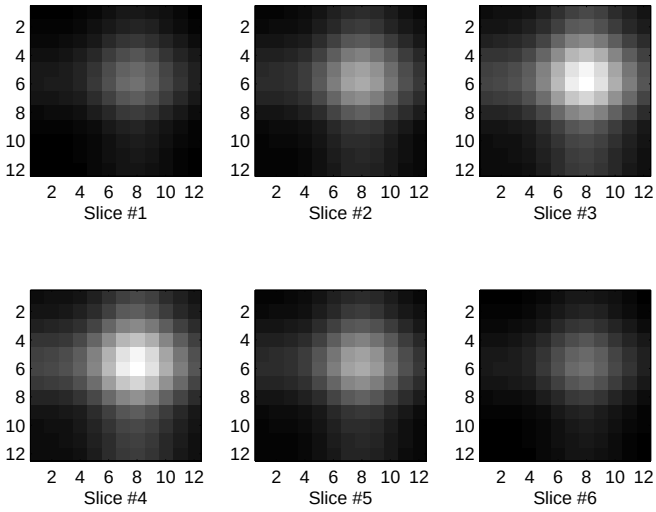

Figure 2.12: The basis tensor $\vec{a}_1 \circ \vec{b}_1 \circ \vec{c}_1$.

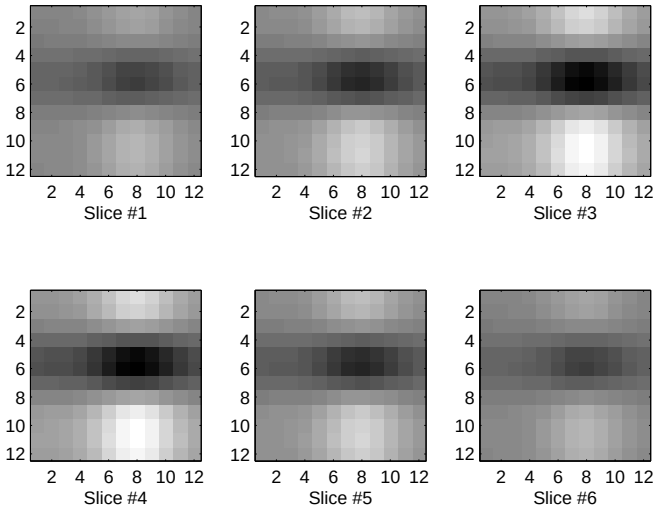

Figure 2.13: The basis tensor $\vec{a}_2 \circ \vec{b}_1 \circ \vec{c}_1$.

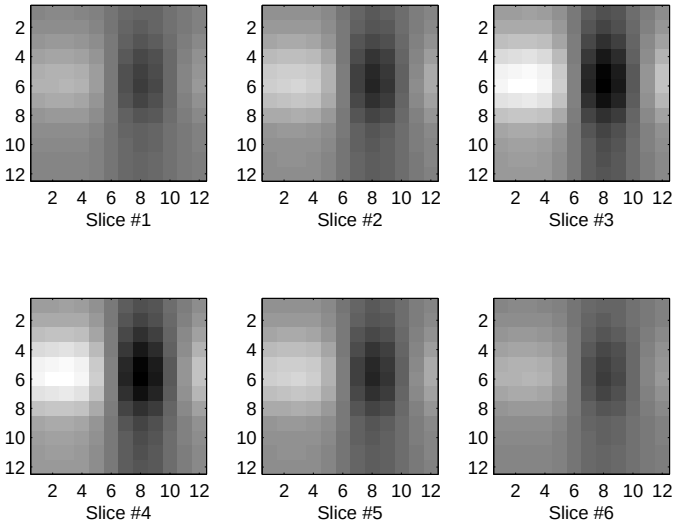

Figure 2.14: The basis tensor $\vec{a}_1 \circ \vec{b}_2 \circ \vec{c}_1$.

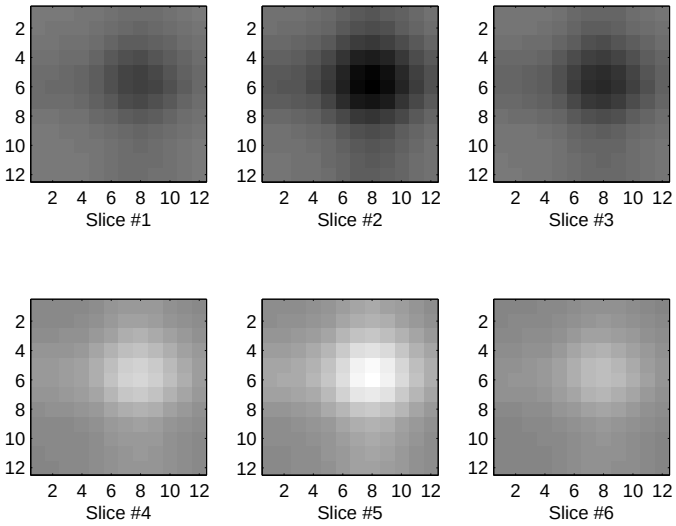

Figure 2.15: The basis tensor $\vec{a}_1 \circ \vec{b}_1 \circ \vec{c}_2$.

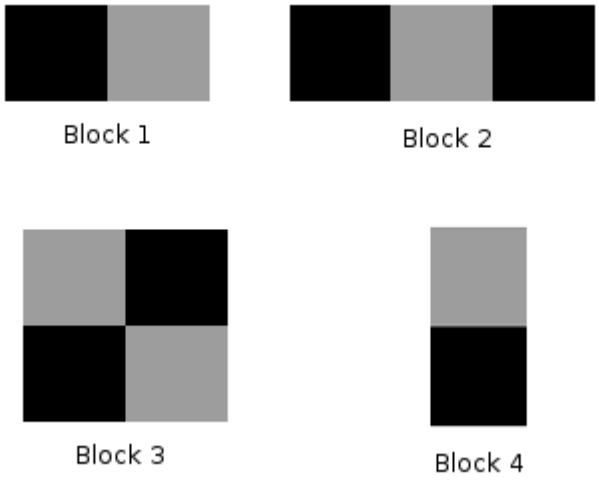

Figure 2.16: Features used in the algorithm.

The Haar system is one of the simplest examples of a wavelet [24]. Let

$$\psi(t) = \begin{cases} 1 & \text{if} \quad 0 \leq t < 1/2 \\ -1 & \text{if} \quad 1/2 \leq t < 1 \\ 0 & \text{else} \end{cases}$$

Without going into too much detail, the set of functions

$$\Psi = \{2^{n/2}\psi(2^n t - k) \ \ \forall \text{ non-negative integer } n \text{ and } 0 \leq k < 2^n\} \cup 1$$

is a complete orthonormal system for the set of functions defined on the unit interval. That is, scalings and translations of $\psi(t)$, along with the constant 1, yield a basis for functions on a unit interval. Since this basis is orthonormal, the inner product of the function with each basis element yields the coefficient of that basis element in the expansion. A graph of $\psi(t)$ is shown in figure 2.17.

Inspired by $\psi(t)$, the Haar mother wavelet, Viola and Jones used three types of two-dimensional features: two-rectangle, three-rectangle, and four-rectangle features. These are illustrated in figure 2.16. The feature value is computed as the difference between the sum of the intensities under the black pixels and the sum of the intensities under the white pixels. These features can be scaled to hold a different number of pixels, as long

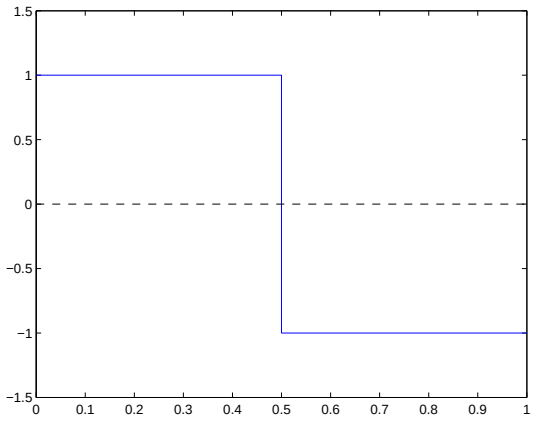

Figure 2.17: The Haar mother wavelet.

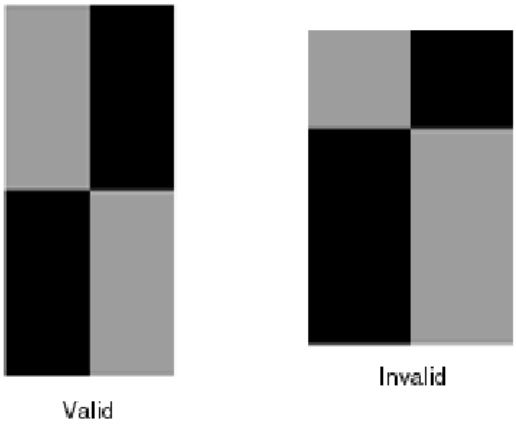

Figure 2.18: The left shows a valid feature while the right shows an invalid feature due to the bottom left region being a different size than the top left.

as the scaling is uniform. Figure 2.18 illustrates a non-uniform scaling resulting in an invalid feature, which is not used. Features with different feature type, size, orientation, or location will be considered distinct features.

Viola and Jones described a very efficient method of computing feature values using

an intermediate representation called the integral image. The integral image is the matrix of values which, at position (r, s), are the sum of the pixels above and to the left of (r, s), inclusively. Using this representation, a two, three, and four-rectangle feature value can be computed in six, eight, and nine array references, respectively.

The following process uses the integral image to calculate the sum of the pixel intensities in a rectangle. Call the integral image ii, so that $ii(r, s)$ is the sum of the pixel intensities above and to the left of pixel (r, s). Consider the shaded rectangle in figure 2.19 as an example. Then, $ii(x_1, y_1)$ is the sum of the pixel intensities in the shaded rectangle and in regions 1, 2 and 3. However, the latter three regions are undesired. To remove their influence, $ii(x_2, y_2)$, which sums the intensities in regions 1 and 2, is subtracted and $ii(x_3, y_3)$, which sums the intensities in regions 1 and 3, is also subtracted. Now, region 1 has been included once, but subtracted twice. Since region 1 should have no influence, $ii(x_4, y_4)$ must be added once. Finally, the sum of the pixel intensities in the gray rectangle is given by

$$ii(x_1, y_1) - ii(x_2, y_2) - ii(x_3, y_3) + ii(x_4, y_4)$$

The integral image can be computed with just one pass over the image using the following pair of recurrences:

$$\begin{cases} s(x, y) = s(x, y - 1) + i(x, y) \\ ii(x, y) = ii(x - 1, y) + s(x, y) \end{cases}$$

where s is the cumulative row sum, ii is the integral image, i is the original image, and any of these functions is zero when evaluated at a negative index.

All of these ideas can be generalized to three dimensions. First, feature types need to be created to fit in $12 \times 12 \times 6$ voxel subwindows. There are many choices for these features. For example, figures 2.20 and 2.21 shows two feature types such that, when all the orientations are placed in every location of the window, provide a complete basis without scaling. Each particular orientation of this $2 \times 2 \times 2$ voxel feature can be placed in $11 \cdot 11 \cdot 5 = 605$ locations. As there are 8 orientations of the feature in figure 2.20 and 12 orientations of the feature in figure 2.21, there are $20 \cdot 605 = 12,100$ distinct

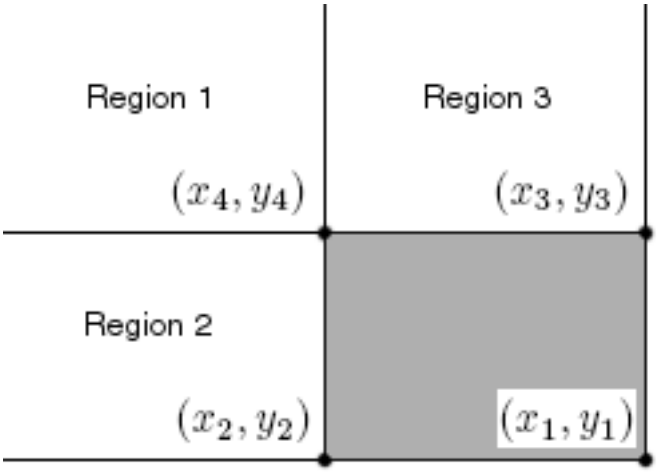

Figure 2.19: The shaded rectangle is computed from the integral image with only 4 array references.

feature values just using these two types. There are many more feature types of this size, each which can be placed in 605 locations and oriented up to 20 different ways. When features of different sizes are considered, such as the two-dimensional case of Viola and Jones, the number of distinct feature values gets extremely large.

Most likely, with such a large number of distinct feature values, it is computationally infeasible to compute every single feature to send to a classifier. However, that does not mean a certain number of these features must be thrown out before training. It also does not mean that all feature values can be used. It all depends on the classifier chosen.

Some classifiers will choose the best combination of every feature, inherently using ever feature at least once. However, some classifiers, such as AdaBoost, choose features until a desired accuracy is achieved. Further, other classifiers can be trained to use a sparse set of the features by penalizing the number of features used; these are called shrinkage methods.

The integral image can also be generalized to three dimensions. Let iii, denote the three-dimensional integral image. Then, $iii(r, s, t)$ is the sum of all intensities to the

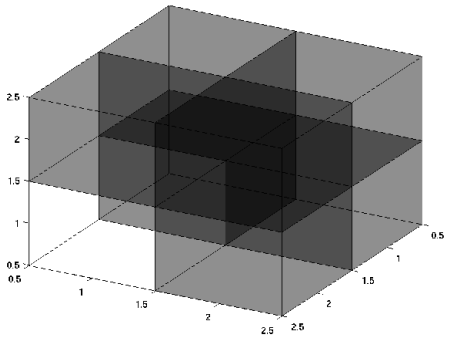

Figure 2.20: Example of a three-dimensional Haar-like feature, which can be placed in multiple locations in the subwindow. All intensities of voxels in the close lower left white box are summed and subtracted from the sum of intensities in the gray areas.

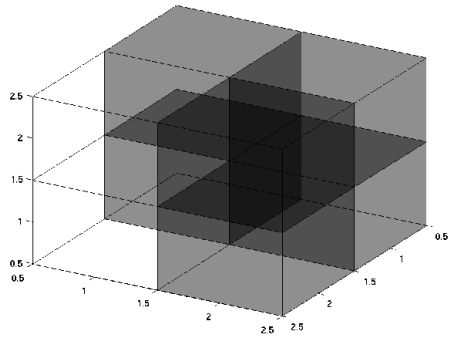

Figure 2.21: Another example of a three-dimensional Haar-like feature. All intensities of voxels in the close left white boxes are summed and subtracted from the sum of intensities in the gray areas.

left of r, above s, and all slices above t. In other words,

$$iii(r,s,t) = \sum_{r'=1}^{r}\sum_{s'=1}^{s}\sum_{t'=1}^{t} i(r',s',t')$$

The three-dimensional integral image also be computed with one pass through the image using the following set of recurrences.

$$\begin{cases} s(x,y,z) = s(x,y-1,z) + i(x,y,z) \\ ii(x,y,z) = ii(x-1,y,z) + s(x,y,z) \\ iii(x,y,z) = iii(x,y,z-1) + ii(x,y,z) \end{cases}$$

This can save a large number of computations. Consider the feature in figure 2.20. Assuming this is a $6\times6\times4$ voxel feature, $6\cdot6\cdot4 = 144$ array references must be computed without the integral image. With the availability of the integral image, this computation can be computed in 15 array references (which is the sum of the 8 references for the

large cube plus the 8 references for the small black cube minus the corner common to both, which only needs to be referenced twice).

In this work, only features of size $4 \times 4 \times 2$ voxels were computed at different locations and orientations in a $12 \times 12 \times 6$ subwindow. The choice for these features was twofold. First, implementation, while fairly straight forward, is very involved. Feature type and orientation must be hard coded into the system, although location can be done automatically. However, using 10 feature types, each having up to 20 different orientations, yields up to 200 combinations of type and orientation. Using different sizes in a $12 \times 12 \times 6$ subwindow could multiply this by a factor of 50, yielding up to $10,000$ features to be hard coded in. (This does not include distinct locations within the subwindow.) Second, the nodules that were used in this study were all approximately $4 \times 4 \times 2$ voxels large. Larger features would likely fail to capture the details of the discrepancy between nodule and non-nodule subwindows. Assuming the classifier does not use every feature, ignoring larger and smaller features is a poor decision. However, this work is meant to look into the performance of this type of feature, and this subset should yield some understanding. The performance will be reviewed in chapter 4.

2.5 Lower Fourier Modes

The last feature extraction technique explored in this paper is lower Fourier modes. Consider the image in figures 2.22 and 2.23. Figure 2.22 shows all of the nodule subwindows, reshaped as row vectors. In these rows, there is a noticeable low frequency component in each row. This is a result of a nodule in the center of the subwindow. Figure 2.23 displays the same idea with the non-nodule subwindows. These rows do not display any common low frequency components.

In three dimensions the low frequency components are also visible. In figure 2.24, looking in each of the three dimensions of the example subwindow shows that these low frequency components are present in nodules, while they would not be in a non-nodule window. This motivates the use of a Fourier decomposition of the subwindows, using the Fourier coefficients as feature vectors.

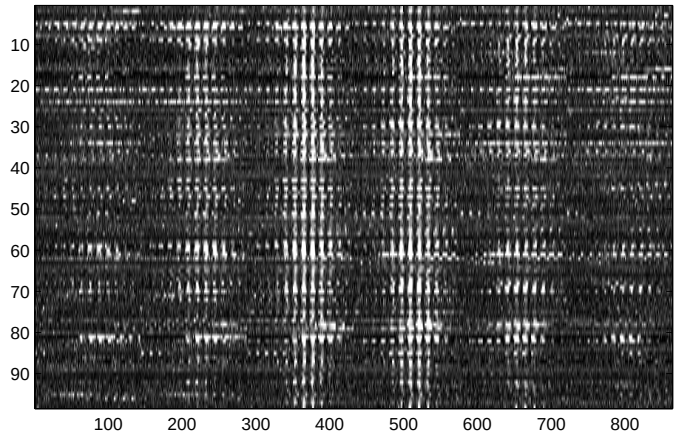

Figure 2.22: Each nodule subwindow is reshaped as a row vector using first the slice, then the column, then the row. Each row of the above figure is a separate instance of a nodule subwindow.

Suppose the subwindow, f^*, is a continuous function of $(x, y, z) \in [-\pi, \pi] \times [-\pi, \pi] \times [-\pi, \pi]$. Then the formula for the decomposition of f^* is

$$f^*(x, y, z) = \sum_{\ell_1=0}^{\infty} \sum_{\ell_2=0}^{\infty} \sum_{\ell_3=0}^{\infty} a^*_{\ell_1,\ell_2,\ell_3} b^X_{\ell_1}(x) b^Y_{\ell_2}(y) b^Z_{\ell_3}(z)$$

where

1. $b^W_t(w) = \begin{cases} \cos(\frac{t}{2}w) & \text{if} \quad t \text{ is even} \\ \sin(\frac{t+1}{2}w) & \text{if} \quad t \text{ is odd} \end{cases}$ for $t = 0, 1, 2, ...$

2. $a^*_{\ell_1,\ell_2,\ell_3} = \frac{1}{(2\pi)^3} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} f^*(x, y, z) b^X_{\ell_1}(x) b^Y_{\ell_2}(y) b^Z_{\ell_3}(z)\, dx\, dy\, dz$

However, the real subwindow, f, is an approximation of a continuous function of $(x, y, z) \in [.5, 12.5] \times [.5, 12.5] \times [.5, 6.5]$. Thus the above is modified using the midpoint rule to yield the following formula for f.

$$f(x, y, z) = \sum_{\ell_1=0}^{\infty} \sum_{\ell_2=0}^{\infty} \sum_{\ell_3=0}^{\infty} a_{\ell_1,\ell_2,\ell_3} c^I_{\ell_1}(x) c^J_{\ell_2}(y) c^K_{\ell_3}(z)$$

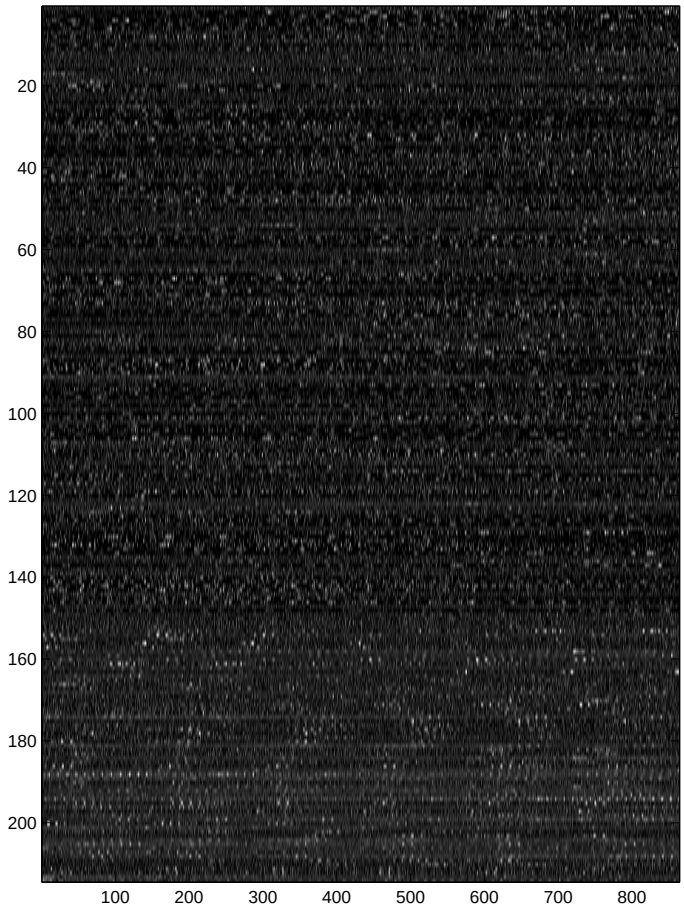

Figure 2.23: The same type of plot as in figure 2.22, but this time with the non-nodule subwindows. The first 152 rows correspond to subwindows without nodules or vessels, while the remaining rows correspond to subwindows with only vascular structures present.

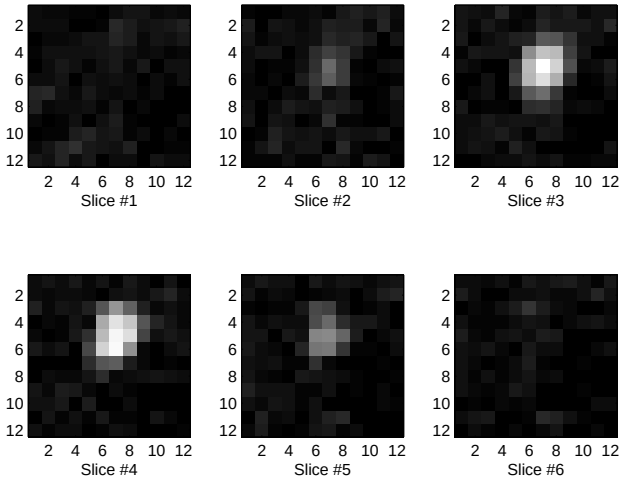

Figure 2.24: Example of a nodule in a CT scan.

where

$$c_t^W(w) = \begin{cases} \cos(\frac{t}{2}\frac{(w-m_W)\pi}{s_W}) & \text{if} \quad t \text{ is even} \\ \sin(\frac{t+1}{2}\frac{(w-m_W)\pi}{s_W}) & \text{if} \quad t \text{ is odd} \end{cases}$$

for $t = 0, 1, 2, ...,$ m_W and s_W being the midpoint and length of the interval for which variable w is defined, respectively. The coefficients can be determinted by the following approximation.

$$a_{\ell_1,\ell_2,\ell_3} \approx \frac{1}{12 \cdot 12 \cdot 6} \sum_{i=1}^{12} \sum_{j=1}^{12} \sum_{k=1}^{6} f(i,j,k) c_{\ell_1}^I(i) c_{\ell_2}^J(j) c_{\ell_3}^K(k)$$

Since some of the low-frequency components are large for nodule subwindows and small for non-nodule windows, it is not necessary to use the high frequency components in the feature extraction. Indeed, only the first few values of ℓ_s are needed to capture the low frequency element of nodules for each $s = 1, 2, 3$. Figure 2.25 shows some of these coefficients from nodule and non-nodule subwindows in reduced dimension.

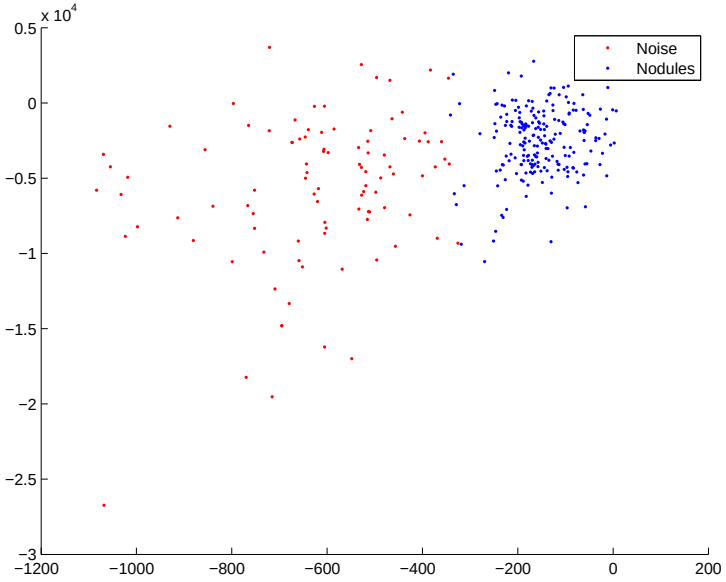

Figure 2.25: Lower Fourier modes show good separation of nodule and non-nodule subwindows after reducing dimension.

The plot in figure 2.25 was created with the following triplets of (ℓ_1, ℓ_2, ℓ_3).

- $(0, 0, 0)$
- $(0, 0, r)$, $(0, r, 0)$, and $(r, 0, 0)$ for $r = 1, 2, 3, 4$
- $(0, r, s)$, $(r, 0, s)$, and $(r, s, 0)$ for $r = 1, 2$ and $s = 1, 2$
- (q, r, s), for $q = 1, 2$, $r = 1, 2$, and $s = 1, 2$

Including all second order Fourier components, i.e. (q, r, s), for $q, r, s = 0, 1, 2, 3, 4$ would yield an improvement of the detail captured from the subwindow. However, this set has $5^3 = 125$ elements, as opposed to the 33 elements used above. For simplicity, the second order components will not be considered. Regardless, these features yield very good results, which can be seen in chapter 4.

Chapter 3

Classification Methods

3.1 Preliminaries

3.1.1 Cross-Validation

There is a common issue involved with all the classification methods about to be discussed, and that is validation. With such high-dimensional data, it is important to utilize all the data available. That is, it is a poor choice to simply divide the data into a training set and a testing set. Thus, in this paper, a cross-validation scheme will be used in order to utilize as much of the data as possible. The leave-one-out cross-validation scheme is illustrated in **Cross-Validation 1** for the set $\{(\vec{x}_i, y_i)\}_{i=1}^N$, where $\vec{x}_i$ is a feature vector and y_i is the corresponding label, 1 for nodule and 0 for non-nodule.

Algorithm: Cross-Validation 1 (3.1)

1. Input N labeled examples $\{(\vec{x}_i, y_i)\}_{i=1}^N$
2. For $j = 1$ to N do
   - **a.** Train a classifier h on $\{(\vec{x}_i, y_i)\}_{i=1}^N \setminus \{(\vec{x}_j, y_j)\}$
   - **b.** Calculate $h(\vec{x}_j)$
3. Output the confusion matrix

	Classified as 0	Classified as 1
Class 0	·	·
Class 1	·	·

Table 3.1: The form of the confusion matrix

(The confusion matrix will be discussed shortly.) Leave-one-out cross-validation allows every sample to be classified without a biased classifier. This means that the classifiers were trained on a set that does not include the sample to be tested.

Besides being a useful way to validate how a feature set and classifier perform, there is another use for cross-validation. Sometimes there are parameters involved in the model. For example, AdaBoost uses the parameter T to limit the number of features used, the SVD can be limited on the number of basis vectors to be used, there are two parameters needed to weigh the covariance matrices in the modification of the SVD to extract features, and etc. Cross-validation works very well for deciding these parameters.

The algorithm for learning parameters is essentially the same. However, sometimes performance cannot just be measured by a confusion matrix. Many times, a cost function needs to be chosen. A cost function is a function that states how costly an error is. The larger the cost, the worse the performance is. Here are a few examples of cost functions:

- Number of misclassifications on the data set (for discrete labels)

- Weighted sum of misclassifications, e.g.
  $2 \cdot (\#$ of false negatives$) + 1 \cdot (\#$ of false positives) (for discrete labels)

- Sum of squared errors (for continuous labels)

Denote the cost function by $C(\cdot)$. Then the following modification of **Cross-Validation 1** uses a cost function to learn a parameter.

Algorithm: Cross-Validation 2 (3.2)

1. Input N labeled examples $\{(\vec{x}_i, y_i)\}_{i=1}^{N}$ and possible parameter values $\{\lambda_j\}_{j=1}^{J}$
2. For $j = 1$ to J do
   - **a.** Train a classifier h on $\{(\vec{x}_i, y_i)\}_{i=1}^{N} \setminus \{(\vec{x}_j, y_j)\}$, using parameter λ_j
   - **b.** Calculate $h(\vec{x}_j)$
   - **c.** Calculate the cost $C(\lambda_j)$
3. Choose $\lambda = \underset{\lambda_j}{\operatorname{argmin}}\, C(\lambda_j)$

In the case of learning more than one parameter, $\lambda^{(1)}$, $\lambda^{(2)}$,..., and $\lambda^{(M)}$, choosing the best value from the mesh

$$\{\lambda_{j_1}^{(1)}\}_{j_1=1}^{J_1} \times \{\lambda_{j_2}^{(21)}\}_{j_2=1}^{J_2} \times \cdots \times \{\lambda_{j_M}^{(M)}\}_{j_M=1}^{J_M}$$

works well. **Cross-Validation 2** would thus be modified to choose the argument minimizer of $C(\lambda_{j_1}^{(1)}, \lambda_{j_2}^{(2)}, ..., \lambda_{j_M}^{(M)})$.

If cross-validation is to be used to learn a parameter, the parameter will depend on all the samples. Since the classifier depends on the parameter(s), this means that the results reported in the confusion matrix are slightly biased. This can be resolved by modifying **Cross-Validation 2**. In step 2a, a nested cross-validation can be used to learn this parameter. That way, the parameter will be chosen according to every sample but $\vec{x}_j$, and the classifier used in step 2b will not depend on this sample.

While leave-one-out cross-validation is a useful way to estimate how a classifier performs, it may be much too computationally intensive to use on a large data set. For example, on a data set with 600 samples, the classifier will have to be trained 600 times on 599 samples each. While the number of training iterations goes up, the complexity of each classifier also increases, and the run-time quickly becomes impractical. This is the case for the data set used in this paper.

Instead of leaving one *sample* out and training on the rest of the samples, a quick method to obtain unbiased results is to leave one *subset* out and train on the rest of the samples. This is the approach that is used in chapter 4 of this thesis. The training set is divided up into M subsets. Then for each $i = 1, 2, ..., M$, a classifier is trained on every subset but the ith subset, and then tested on the ith subset. For example, on a data set with 600 (randomly ordered) samples and $M = 6$, six classifiers will be trained on 500 samples each. This method makes validation much less time consuming, while not sacrificing too much accuracy. For the results in chapter 4, M is chosen to be five.

3.1.2 The Confusion Matrix

Since making the mistake of classifying a nodule subwindow as a non-nodule subwindow is much more costly than making the mistake of classifying a non-nodule subwindow as a nodule subwindow, the total number of errors is not a suitable quantity to analyze the performance of a system. In the cross-validation system above, results are reported in the confusion matrix:

	Classified as 0	Classified as 1
Class 0	a_{00}	a_{01}
Class 1	a_{10}	a_{11}

where the element a_{ij} is the number of total samples from class i that are classified to class j. Note that row i adds up to the total number of samples from class i, and all the entries in the confusion matrix add up to the total number of samples in the data set.

This matrix explains the results in full. The off-diagonal elements can be added to yield the number of errors. The diagonal elements can be added to get the number of correct classifications. The detection rate can be found by calculating $\frac{a_{00}}{a_{00}+a_{01}}$, the false positive rate can be found by calculating $\frac{a_{10}}{a_{10}+a_{11}}$, and etc.

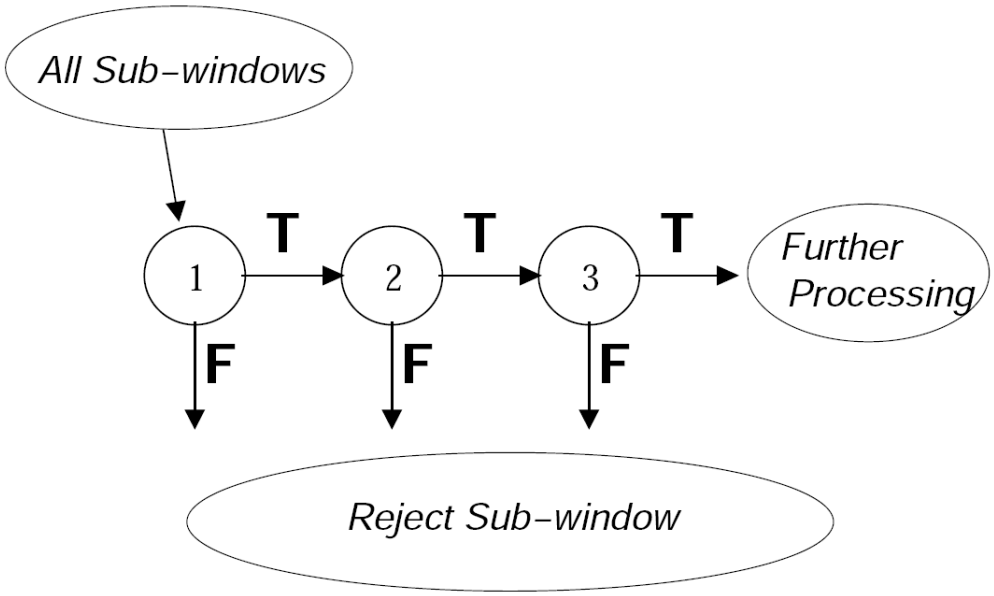

Figure 3.1: The attentional cascade to speed up classification.

3.1.3 Attentional Cascade

In Viola and Jones' face detection work [19], they describe a method to speed up classifications. This method, the attentional cascade of classifiers, can be applied to many of the feature types and classifiers described in this paper.

The attentional cascade is an idea based on the following insight. Smaller, more efficient classifiers can be constructed that reject many of the negative subwindows while detecting nearly all of the positive subwindows. These classifiers will have a large false positive rate, however, the detected positive subwindows and false detections are passed on sequentially to larger classifiers that yield a better false positive rate. While these larger classifiers will be less efficient, they will be only be needed on a much smaller number of subwindows. A diagram representing the attentional cascade is shown in figure 3.1.

While most classifiers can be used for the larger classifiers which are evaluated later on in the process, only certain classifiers can be used for the smaller classifiers evaluated first. Viola and Jones used AdaBoost for these classifiers. Trees also work well. These

classifiers will be discussed later, with an explanation of how they fit into the attentional cascade.

This attentional cascade is especially useful when using features such as the SVD, Fourier modes, and tensor decompositions. To extract these features, every pixel in the subwindow must be used. If a more efficient classifier can first eliminate 60% of the negative subwindows, as Viola and Jones were able to do in the two-dimensional face detection problem, it will not be necessary to extract these costly features on over half of the subwindows. This would be a very relevant savings in run time.

3.2 Parametric Methods

Parametric methods have the ability to perform extremely well given knowledge of the distribution from which the cases were sampled. In order to use parametric methods, the distribution of the extracted features from chapter 2 must be known. With such noisy data, it is a stretch to assume that any of these distributions are known. However, in some cases, such assumptions may bring about a useful classifier.

3.2.1 Likelihood Methods

Optimal Classifiers

Suppose the distribution of extracted features from nodule subwindows and non-nodule subwindows are known. Then the following classifier, known as the Optimal Bayes Classifier, performs the best in the sense of minimizing the probability of an error [8]. Define the following values, with subscript 1 denoting nodule subwindows and 0 denoting non-nodule subwindows.

$P(\omega_i)$: the prior probability that an arbitrary subwindow belongs to class i
$P(\omega_i|\vec{x})$: the posterior probability that a specific feature vector, $\vec{x}$, comes from class i.
$P(\vec{x})$: the probability distribution of all feature vectors.
$P(\vec{x}|\omega_i)$: the probability distribution of all feature vectors belonging to class i.

Since these functions are all assumed to be known, the best classifier to minimize the probability of an error in deciding which class observation $\vec{x}$ belongs to is

$$\underset{i}{\text{argmax}} \;\; P(\omega_i|\vec{x}) \tag{3.3}$$

It is computed using Bayes Rule:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Using Bayes rule, the following holds.

$$\underset{i}{\text{argmax}} \;\; P(\omega_i|\vec{x}) = \underset{i}{\text{argmax}} \;\; \frac{P(\vec{x}|\omega_i)P(\omega_i)}{P(\vec{x})}$$

Since $P(\vec{x})$ is the same regardless of class, this can be written as

$$\arg\max_i P(\vec{x}|\omega_i)P(\omega_i) \tag{3.4}$$

The classifier (3.4) performs the best possible in terms of minimizing the probability of an error. However, for this problem, this is not a good solution. For one, it assumes all the probabilities involved are known when in reality they are not. There is also a larger issue. Consider the prior probabilities $P(\omega_i)$. This is the probability that, given a feature vector from an arbitrary subwindow, the subwindow is from class i. For nodule subwindows, where $i = 0$, this probability will be extremely low, since nodules make up such a near zero percentage of the whole image. This fact causes $P(\vec{x}|\omega_0)P(\omega_0)$ to be extremely small, even if $P(\vec{x}|\omega_0)$ is close to one. This causes the classifier to choose the non-nodule class virtually all the time.

The problem just discussed is not a flaw in the classifier, it is a flaw in the choice of loss function. A loss function is a cost function that states how costly each decision is, e.g. deciding on class 0 when class 1 is correct, deciding class 1 when class 1 is correct, etc. The classifier above minimizes the probability of an error, and thus weights all types of errors the same. If the same loss is incurred for making a mistake on a nodule subwindow as it is for making a mistake on a non-nodule subwindow, a classifier that classifies a subwindow as a non-nodule will be near perfect since there is an extremely

small number of nodule subwindows. It will make a mistake on every nodule subwindow, but that happens so few times that the cumulative loss will remain small.

The solution to this problem is to adjust the loss function to yield a more suitable classifier. Suppose $\vec{x}$ is sent to a classifier. The classifier can make two decisions, call these δ_1 for deciding $\vec{x}$ corresponds to a nodule and δ_0 for deciding $\vec{x}$ corresponds to a non-nodule. Define the loss function as

$$L(\delta_i|\vec{x} \in \omega_j) = \begin{cases} \alpha & \text{if} \quad \vec{x} \text{ is mistakenly classified as a non-nodule subwindow} \\ (1-\alpha) & \text{if} \quad \vec{x} \text{ is mistakenly classified as a nodule subwindow} \\ 0 & \text{else} \end{cases}$$

Here there is no cost for classifying a sample correctly.

The classifier should be chosen to minimize the expected loss, which is called risk. The risk for making decision δ_i about observation $\vec{x}$, is thus defined as

$$R(\delta_i|\vec{x}) = L(\delta_i|\omega_{1-i})P(\omega_{1-i}|\vec{x})$$

Thus the optimal classifier, according to this loss function, is

$$\underset{\delta_i}{\text{argmin}} \;\; R(\delta_i|\vec{x})$$

In other words, decide ω_1 if

$$L(\delta_1|\omega_0)P(\omega_0|\vec{x}) \leq L(\delta_0|\omega_1)P(\omega_1|\vec{x}) \tag{3.5}$$

Now, if $L(\delta_1|\omega_0) = L(\delta_2|\omega_1)$, i.e. each error is weighted the same, this rule is exactly the same as the rule (3.3), the decision that minimizes the probability of an error. Since making the mistake of classifying a nodule subwindow as a non-nodule subwindow is much more costly than making the mistake of classifying a non-nodule subwindow as a nodule subwindow, the following needs to hold.

$$L(\delta_1|\omega_0) > L(\delta_0|\omega_1)$$

Applying Bayes rule to (3.5) and dividing by $L(\delta_0|\omega_1)P(\omega_1)P(\vec{x}|\omega_0)$ on both sides yields the following rule:

Decide ω_1 if

$$\frac{P(\omega_0)L(\delta_1|\omega_0)}{P(\omega_1)L(\delta_0|\omega_1)} \leq \frac{P(\vec{x}|\omega_1)}{P(\vec{x}|\omega_0)}$$

This rule is an ideal classifier for the classification task at hand for any given loss function.

While this works perfectly in theory, it is usually impossible to know what the true values of the $P(\omega_i)$'s are, what the loss function should be, and what the distribution $P(\vec{x}|\omega_i)$ is. However, since the classifier is of the form:

$$C(\vec{x}) = \begin{cases} 1 & \text{if} \quad \dfrac{P(\vec{x}|\omega_1)}{P(\vec{x}|\omega_0)} \geq c \\ 0 & \text{else} \end{cases} \tag{3.6}$$

it is not necessary to choose the prior probabilities and loss function explicitly. Rather, it is appropriate to find only a constant c so that a desired performance is achieved. This can be accomplished by a cross-validation scheme.

Dealing with an Unknown Distribution

The previous section outlines an approach when the distributions, $P(\vec{x}|\omega_i)$, are known. However, many of the types of features discussed above have unknown distributions. Even using a multivariate normal distribution is a horrible approximation for many of the sets of feature vectors.

The classifier in equation (3.6) can be viewed as a threshold on a ratio of likelihoods. Each conditional probability, $P(\vec{x}|\omega_i)$, is the likelihood that $\vec{x} \in \omega_i$. If one of $P(\vec{x}|\omega_i)$ is unknown but the classes are relatively separable in feature space, it is still possible to use the likelihood of the other to obtain a good classifier.

Equation (3.6) can be rewritten as

$$C(\vec{x}) = \begin{cases} 1 & \text{if} \quad P(\vec{x}|\omega_1) \geq cP(\vec{x}|\omega_0) \\ 0 & \text{else} \end{cases}$$

Now suppose the distribution of $\vec{x}$ under ω_0 is unknown, but the distribution of $\vec{x}$ under ω_1 is known, an assumption that comes in play with features from the SVD. Since

$P(\vec{x}|\omega_0)$ is unknown, replace it with a constant. Then (3.6) becomes

$$C(\vec{x}) = \begin{cases} 1 & \text{if} \quad P(\vec{x}|\omega_1) \geq c' \\ 0 & \text{else} \end{cases}$$

which can be interpreted as thresholding the likelihood of the known distribution. Once again the c' should be learned using cross-validation to yield the best performance.

3.2.2 Probabilistic D-Clustering

Clustering is the problem of finding K clusters and a rule assigning each sample in a data set to a cluster. The rule is called the clustering criterion.

A clustering criterion can be a metric. This type of clustering is called distance clustering or d-clustering. Each cluster has a center, and each data point is assigned to the center it is closest to with respect to that metric. Such clustering algorithms usually iterate between updating the center of each cluster and reassigning the samples to their cluster.

A clustering criterion can also be based on the probability that a sample is a member of each cluster. This type of clustering is called probabilistic clustering. For each sample, there is a probability that it belongs to each cluster. The sample is assigned to the cluster which is the most probable.

Ben-Israel et al. describe a method to use a relationship between probability and distances to cluster a data set [25].

Details of this Method

The key to this clustering is a relationship between the distance metric and the probabilities. The authors experiment with more than one relationship, although the best results came from the following.

$$p_k(\vec{x})d(\vec{x}, \vec{c}_k) = D(\vec{x}), \text{ a constant, depending on } \vec{x} \text{ (not on } k) \tag{3.7}$$

where $p_k(\vec{x})$ is the probability that $\vec{x}$ is a member of the kth cluster, $\mathcal{C}_k$, $d(\vec{x}_1, \vec{x}_2)$ is the distance between $\vec{x}_1$ and $\vec{x}_2$, and $\vec{c}_k$ is the center of cluster $\mathcal{C}_k$. The function $D(\vec{x})$ is

called the joint distance function (JDF). When dealing with a set of points, $X = \{\vec{x}_i\}_{i=1}^N$, the JDF is defined as

$$D(X) = \sum_{i=1}^{N} D(\vec{x}_i)$$

In addition, the authors report that good results were attained by the following relationship.

$$p_k(\vec{x})e^{d(\vec{x},\vec{c}_k)} = D(\vec{x}), \text{ a constant, depending on } \vec{x} \text{ (not on } k) \tag{3.8}$$

Here, details will be discussed for the relationship (3.7). Further, the case $K = 2$ will be used. This is because there are only two classes of subwindows in the problem at hand. Details for the relationship (3.8) and other K values are similar.

First, the equation

$$D(\vec{x}) = p_k(\vec{x})d(\vec{x}, \vec{c}_k)$$

implies

$$p_k(\vec{x}) = \frac{D(\vec{x})}{d(\vec{x}, \vec{c}_k)}$$

Since the probabilities add to one over all values of k, the following relationship holds.

$$1 = \sum_{k=1}^{K} p_k(\vec{x}) = D(\vec{x}) \sum_{k=1}^{K} \frac{1}{d(\vec{x}, \vec{c}_k)}$$

This can be rewritten as

$$1 = D(\vec{x}) \frac{\sum_{k=1}^{K} \prod_{j \neq k} d(\vec{x}, \vec{c}_j)}{\prod_{k=1}^{K} d(\vec{x}, \vec{c}_k)}$$

which means

$$D(\vec{x}) = \frac{\prod_{k=1}^{K} d(\vec{x}, \vec{c}_k)}{\sum_{k=1}^{K} \prod_{j \neq k} d(\vec{x}, \vec{c}_j)}$$

Thus the JDF gives a measure of the total distance from $\vec{x}$ to the cluster centers. This relationship also yields:

$$p_t(\vec{x})d(\vec{x}, \vec{c}_t) = D(\vec{x}) = \frac{\prod_{k=1}^{K} d(\vec{x}, \vec{c}_k)}{\sum_{k=1}^{K} \prod_{j \neq k} d(\vec{x}, \vec{c}_j)}$$

which implies

$$p_t(\vec{x}) = \frac{\prod_{k \neq t} d(\vec{x}, \vec{c}_k)}{\sum_{k=1}^{K} \prod_{j \neq k} d(\vec{x}, \vec{c}_j)} \tag{3.9}$$

That is, each probability, $p_k(\vec{x})$, can be written in terms of the distances to the cluster centers.

Putting this aside for the moment, consider the extremal problem for two clusters, which is the following:

$$\begin{aligned} \textit{Minimize: } & d(\vec{x},\vec{c}_1)p_1^2 + d(\vec{x},\vec{c}_2)p_2^2 \\ \textit{Subject to: } & p_1 + p_2 = 1 \\ & p_1, p_2 \geq 0 \end{aligned} \tag{3.10}$$

The Lagrangian of this problem is

$$L(p_1, p_2, \lambda) = d(\vec{x},\vec{c}_1)p_1^2 + d(\vec{x},\vec{c}_2)p_2^2 - \lambda(p_1 + p_2 - 1)$$

Then, setting the partial derivatives of $L(p_1, p_2, \lambda)$ with respect to p_1 and p_2 equal to zero yields

$$2p_i d(\vec{x},\vec{c}_i) - \lambda = 0$$

This can be written as

$$p_i d(\vec{x},\vec{c}_i) = \frac{\lambda}{2}$$

which implies that the following relationship holds.

$$p_1 d(\vec{x},\vec{c}_1) = p_2 d(\vec{x},\vec{c}_2) \tag{3.11}$$

This relationship is the same as principle (3.7). Again setting $D(\vec{x}) = p_i d(\vec{x},\vec{c}_i)$ and plugging this into the objective function in (3.16) yields the problem

$$\textit{Minimize: } D(\vec{x})p_1 + D(\vec{x})p_2 \tag{3.12}$$

$$\textit{Subject to: } p_1 + p_2 = 1 \tag{3.13}$$

$$p_i d(\vec{x},\vec{c}_i) = \lambda/2 \tag{3.14}$$

$$p_1, p_2 \geq 0 \tag{3.15}$$

which is clearly optimized by the p_1 and p_2 which satisfies equation (3.11), as well as the sum to one constraint. This has already been solved in equation (3.9). Thus the

optimal value of the extremal problem is the JDF and it is achieved by the probabilities in equation (3.9).

When dealing with set of points, $X = \{\vec{x}_i\}_{i=1}^N$, the extremal problem is

$$\begin{aligned} \textit{Minimize: } & \sum_{i=1}^{N} d(\vec{x}_i, \vec{c}_1) p_1(\vec{x}_i)^2 + d(\vec{x}_i, \vec{c}_2) p_2(\vec{x}_i)^2 \\ \textit{Subject to: } & p_1(\vec{x}_i) + p_2(\vec{x}_i) = 1 \\ & p_1, p_2 \geq 0 \end{aligned} \tag{3.16}$$

This problem is easily solved by noting that it separates into N problems like (3.16). The optimal value turns out to be the JDF,

$$D(X) = \sum_{i=1}^{N} \frac{\prod_{k=1}^{K} d(\vec{x}_i, \vec{c}_k)}{\sum_{k=1}^{K} \prod_{j \neq k} d(\vec{x}_i, \vec{c}_j)}$$

which is realized by the same probabilities, i.e. (3.9).

Since this is a clustering problem, the real goal is to find the best centers for K clusters with respect to the metric and probabilities. To do this, the extremal problem is solved viewing the centers as the variable. While the same relationship, described in equation (3.11), holds, dealing with the centers requires a fixed distance function. That is, the clustering algorithm relies on the metric used. Consider the elliptical distance function defined separately for each cluster

$$d_k(\vec{x}, \vec{y}) = \left((\vec{x} - \vec{y})^T Q_k (\vec{x} - \vec{y})\right)^{1/2}$$

where Q_k is positive-definite. The objective function is

$$f(\vec{c}_1, \vec{c}_2) = \sum_{i=1}^{N} \left((\vec{x}_i - \vec{c}_1)^T Q_k (\vec{x}_i - \vec{c}_1)\right)^{1/2} p_1(\vec{x}_i)^2 + \left((\vec{x}_i - \vec{c}_2)^T Q_k (\vec{x}_i - \vec{c}_2)\right)^{1/2} p_2(\vec{x}_i)^2 \tag{3.17}$$

Thus

$$\nabla_{\vec{c}_k} f(\vec{c}_1, \vec{c}_2) = -Q_k \sum_{i=1}^{N} \frac{\vec{x}_i - \vec{c}_k}{d_k(\vec{x}_i, \vec{c}_k)} p_k(\vec{x}_i)^2$$

Setting this gradient equal to zero yields the following equation

$$0 = -Q_k \sum_{i=1}^{N} \frac{\vec{x}_i - \vec{c}_k}{d_k(\vec{x}_i, \vec{c}_k)} p_k(\vec{x}_i)^2$$

since Q_k is positive definite, it is invertible and thus the following holds

$$0 = \sum_{i=1}^{N} \frac{\vec{x}_i - \vec{c}_k}{d_k(\vec{x}_i, \vec{c}_k)} p_k(\vec{x}_i)^2$$

Multiplying through and grouping the $\vec{x}_i$ terms yields.

$$\sum_{i=1}^{N} \left(\frac{p_k(\vec{x}_i)^2}{d_k(\vec{x}_i, \vec{c}_k)} \right) \vec{x}_i = \left(\sum_{i=1}^{N} \frac{p_k(\vec{x}_i)^2}{d_k(\vec{x}_i, \vec{c}_k)} \right) \vec{c}_k$$

Letting

$$u_{k,i} = \frac{p_k(\vec{x}_i)^2}{d_k(\vec{x}_i, \vec{c}_k)}$$

simplifies this expression to

$$\sum_{i=1}^{N} u_{k,i} \vec{x}_i = \left(\sum_{j=1}^{N} u_{k,j} \right) \vec{c}_k$$

which yields a formula for the centers $\vec{c}_k$:

$$\vec{c}_k = \sum_{i=1}^{N} \frac{u_{k,i} \vec{x}_i}{\sum_{i=1}^{N} u_{k,i}} \tag{3.18}$$

Finally, these ideas are implemented in the following algorithm.

Algorithm: Probabilistic D-Clustering [25]

1. Input data X, two guesses $\vec{c_1}$ and $\vec{c_2}$, and stopping criterion ϵ.
2. Compute distances $d_k(\vec{x}_i, \vec{c}_k)$
3. Update centers $\vec{c}_k^{+}$ using (3.18)
4. While $||\vec{c}_1^{+} - \vec{c}_1|| + ||\vec{c}_2^{+} - \vec{c}_2|| \geq \epsilon$ do
   - **a.** Replace old centers, $\vec{c}_k = \vec{c}_k^{+}$
   - **b.** Compute distances $d_k(\vec{x}_i, \vec{c}_k)$
   - **c.** Update centers $\vec{c}_k^{+}$ using (3.18)

5. Output the final centers $\vec{c}_k^+$

It may be necessary to update the metric at each iteration. For example, the Mahalanobis distance is defined as

$$d(\vec{x}, \vec{c}_k) = \left((\vec{x} - \vec{c}_k)^T \Sigma_k^{-1} (\vec{x} - \vec{c}_k)\right)^{1/2}$$

where Σ_k is the covariance matrix of the k-th cluster. If this distance is used, Σ_k should be updated at each iteration.

This algorithm will decrease the objective function (3.17) at each iteration, and the algorithm will converge [25].

Application of this Method

This clustering problem is an *unsupervised learning* problem, i.e the algorithm does not use labels to learn parameters in the model. That is, centers are learned from the raw data. With relation (3.11) holding, a choice of the distance function completely determines the probability function, as in (3.9). That is, if the goal is to separate two certain distributions with labels, the distance function must be chosen very carefully.

3.3 Non-parametric

3.3.1 Support Vector Machines

One of the most popular classification methods is Support Vector Machines (SVM). This non-parametric approach was developed by Vapnik [26]. It seeks the hyperplane that maximizes the distance between each class and the hyperplane.

Details of this Method

Suppose a data set X consists of samples X_{-1}, which makeup class negative one, and X_1, which makeup class one. Further, suppose these classes are separable, meaning there exists a hyperplane that completely separates the two classes. Define the margin

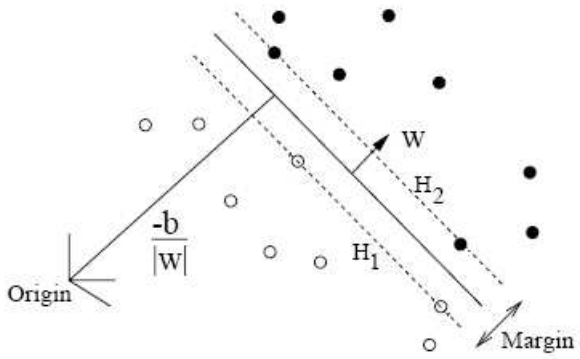

Figure 3.2: The margin is illustrated [2].

for a hyperplane as sum of the minimum distances from each set to the hyperplane. The margin is illustrated in figure 3.2.

Now, suppose $X = \{\vec{x}_i\}_{i=1}^{N}$, $y_i = -1$ for $\vec{x}_i \in X_{-1}$, and $y_i = 1$ for $\vec{x}_i \in X_1$. Consider all hyperplanes $\vec{w} \cdot \vec{x} + b = 0$ with normal $\vec{w}$. When looking for the hyperplane with the largest margin, it is sufficient to suppose all training data satisfies the following:

$$\vec{w} \cdot \vec{x}_i + b \geq 1 \text{ for each } x_i \in X_1$$

$$\vec{w} \cdot \vec{x}_i + b \leq -1 \text{ for each } x_i \in X_{-1}$$

which can be written as the single inequality

$$(\vec{w} \cdot \vec{x}_i + b)y_i \geq 1 \tag{3.19}$$

Without loss of generality, we can assume that there is at least one x_i in both X_{-1} and X_1 that yield equality in equation (3.19). (This is possible by simply rescaling $\vec{w}$ and b.) With this assumption, it is possible to calculate the margin.

The margin is the perpendicular distance between the two parallel lines $\vec{w} \cdot \vec{x} + (b - 1) = 0$ and $\vec{w} \cdot \vec{x} + (b + 1) = 0$. First, however, the perpendicular distance from a line

$$\vec{w} \cdot \vec{x} + c = 0 \tag{3.20}$$

to the origin is calculated.

The line perpendicular to $\vec{w} \cdot \vec{x} + c = 0$ that goes through the origin is given by

$$(w_2, -w_1) \cdot \vec{x} = 0$$

where $\vec{w} = (w_1, w_2)$. Writing $\vec{x} = (x_1, x_2)$ yields

$$x_2 = \frac{w_2 x_1}{w_1} \tag{3.21}$$

Now plug equation (3.21) into equation (3.20) to find the point of intersection.

$$w_1 x_1 + w_2 \frac{w_2 x_1}{w_1} + c = 0$$

which implies

$$x_1 = \frac{-c w_2}{||\vec{w}||^2}$$

Similarly, one can find that

$$x_2 = \frac{-c w_1}{||\vec{w}||^2}$$

That is, the perpendicular distance from (3.20) to the origin is given by

$$\begin{aligned}
\sqrt{x_1^2 + x_2^2} &= \sqrt{\left(\frac{-c w_1}{||\vec{w}||^2}\right)^2 + \left(\frac{-c w_2}{||\vec{w}||^2}\right)^2} \\
&= \frac{|c|}{||\vec{w}||^2} \sqrt{w_1^2 + w_2^2} \\
&= \frac{|c|}{||\vec{w}||}
\end{aligned}$$

So, the distance from the origin to $\vec{w} \cdot \vec{x} + (b - 1) = 0$ is

$$\frac{|b-1|}{||\vec{w}||}$$

and the distance from the origin to $\vec{w} \cdot \vec{x} + (b + 1) = 0$ is

$$\frac{|b+1|}{||\vec{w}||}$$

This makes the margin

$$\begin{aligned}
\left| \frac{|b+1|}{||\vec{w}||} - \frac{|b-1|}{||\vec{w}||} \right| &= \left| \frac{|b+1| - |b-1|}{||\vec{w}||} \right| \\
&= \frac{2}{||\vec{w}||}
\end{aligned}$$

Thus maximizing the margin is equivalent to minimizing $||\vec{w}||$ while observing the constraints mentioned above. Finally, this is posed as the following minimization problem

$$\begin{aligned} &\textit{Minimize: } \frac{1}{2}||\vec{w}||^2 \\ &\textit{Subject to: } (\vec{w} \cdot \vec{x}_i + b)y_i \geq 1 \end{aligned} \tag{3.22}$$

The exponent appears on $||\vec{w}||$ to make the minimization problem much easier to solve as it makes the functional differentiable. This problem is easy to solve, however it is not extremely useful as most classification tasks involve non-separable classes.

It is possible to make this into a classification scheme for non-separable data. This is done by introducing the non-negative slack variables $\{\xi_i\}_{i=1}^N$ so that

$$\vec{w} \cdot \vec{x}_i + b \geq 1 - \xi_i \text{ for each } x_i \in X_1$$

$$\vec{w} \cdot \vec{x}_i + b \leq -1 + \xi_i \text{ for each } x_i \in X_{-1}$$

which can be combined into the equation

$$(\vec{w} \cdot \vec{x}_i + b)y_i \geq 1 - \xi_i \tag{3.23}$$

These slack variables are zero for points on the correct side of the margin, and otherwise represent the perpendicular distance from the margin to the point. (The real distance is $\frac{\xi_i}{||\vec{w}||}$). An example of this is given in figure 3.3. In this figure, there is only one non-zero slack variable.

Since the hyperplane is still given by $(\vec{w} \cdot \vec{x}_i + b) = 0$, the classifier makes a mistake only when $\xi_i \geq 1$. This means that $\sum_{i=1}^N \xi_i$ is an upper bound on the number of training errors. Thus an objective function such as

$$\frac{1}{2}||\vec{w}||^2 + c\sum_{i=1}^{N} \xi_i$$

is a good way of maximizing the margin while keeping training errors low. The constant c is a tuning parameter that can be raised for a higher penalty on errors and lowered for more importance on a large margin.

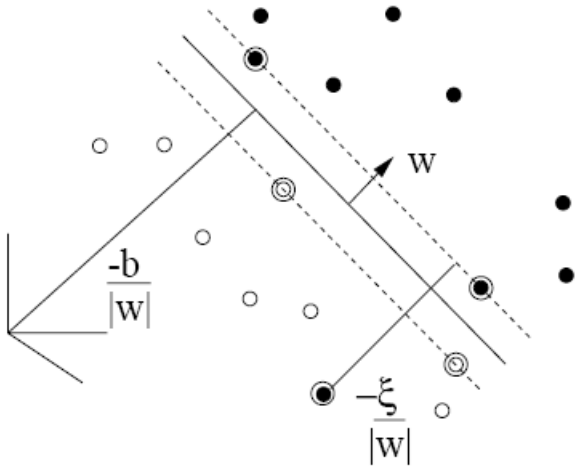

Figure 3.3: The slack variable, ξ, is illustrated [2].

The minimization problem now takes the form

$$\begin{aligned} &\textit{Minimize: } \frac{1}{2}||\vec{w}||^2 + c\sum_{i=1}^{N}\xi_i \\ &\textit{Subject to: } (\vec{w}\cdot\vec{x}_i + b)y_i \geq 1 - \xi_i \\ &\qquad\qquad \xi_i \geq 0 \qquad\qquad \forall i = 1, 2, ..., N \end{aligned} \tag{3.24}$$

To solve this problem, the Lagrangian is formulated. (The "P" subscript denotes the primal problem.)

$$L_P = \frac{1}{2}||\vec{w}||^2 + c\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i\left(y_i(\vec{x}_i\cdot\vec{w} + b) - 1 + \xi_i\right) - \sum_{i=1}^{N}\mu_i\xi_i$$

where the α_i and μ_i's are the positive Lagrange multipliers.

Now, instead of solving the primal problem of minimizing L_P with respect to $\vec{w}$, $\vec{\xi}$ and b while requiring $\nabla_{\alpha_i,\mu_i}L_P = 0$ for each $\alpha_i, \mu_i \geq 0$, it is sufficient to solve the dual problem of maximizing L_P subject to the constraints that $\nabla_{\vec{w},\vec{\xi},b}L_P = 0$ while $\alpha_i, \mu_i \geq 0$ for each $i = 1, 2, ..., N$.

The KKT conditions can now be applied.

Since $\nabla_{w_i} L_P = 0$,

$$\begin{aligned} 0 &= \nabla_{w_i} L_P \\ &= w_i - \sum_{j=1}^{N} \alpha_j y_j (\vec{x}_j)_i \end{aligned}$$

which yields

$$\vec{w} = \sum_{j=1}^{N} \alpha_j y_j \vec{x}_j \tag{3.25}$$

Since $\nabla_b L_P = 0$,

$$\begin{aligned} 0 &= \nabla_b L_P \\ &= \sum_{i=1}^{N} \alpha_i y_i \end{aligned}$$

which yields

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \tag{3.26}$$

Since $\nabla_{\xi_i} L_P = 0$,

$$\begin{aligned} 0 &= \nabla_{\xi_i} L_P \\ &= C - \alpha_i - \mu_i \end{aligned}$$

which yields

$$\alpha_i = C - \mu_i \tag{3.27}$$

Using equations (3.25) – (3.27), the complete KKT conditions are listed below.

$$w_i - \sum_{j=1}^{N} \alpha_j y_j (\vec{x}_j)_i = 0 \tag{3.28}$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \tag{3.29}$$

$$c - \alpha_i - \mu_i = 0 \tag{3.30}$$

$$(\vec{w} \cdot \vec{x}_i + b) y_i - 1 + \xi_i \geq 0 \tag{3.31}$$

$$\xi_i \geq 0 \tag{3.32}$$

$$\alpha_i \geq 0 \tag{3.33}$$

$$\mu_i \geq 0 \tag{3.34}$$

$$\alpha_i \left(y_i (\vec{w} \cdot \vec{x}_i + b) - 1 + \xi_i \right) = 0 \tag{3.35}$$

$$\mu_i \xi_i = 0 \tag{3.36}$$

Further, note that equation (3.30) as well as the requirement $\alpha_i, \mu_i \geq 0$ for every i yields

$$0 \leq \alpha_i \leq c \tag{3.37}$$

However, it is much easier to solve the dual problem. This problem, based on the primal problem (3.24), is

$$\begin{aligned} &\textit{Maximize:} \quad \inf_{\vec{w}, \vec{\xi}, b} L_P \\ &\textit{Subject to:} \quad \alpha_i, \mu_i \geq 0 \ \ \forall i = 1, 2, ..., N \end{aligned} \tag{3.38}$$

From equation (3.25) - (3.27), which must hold at the infemum, L_P can be rewritten as

$$\begin{aligned}
L_P &= \frac{1}{2}||\sum_{j=1}^{N}\alpha_j y_j \vec{x}_j||^2 + c\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i\left(y_i(\vec{x}_i\cdot\left(\sum_{j=1}^{N}\alpha_j y_j\vec{x}_j\right)+b)-1+\xi_i\right) - \sum_{i=1}^{N}\mu_i\xi_i \\
&= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \vec{x}_i\cdot\vec{x}_j + c\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j\vec{x}_i\cdot\vec{x}_j \\
&\qquad - b\sum_{i=1}^{N}\alpha_i y_i - \sum_{i=1}^{N}\alpha_i(-1+\xi_i) - \sum_{i=1}^{N}\mu_i\xi_i \\
&= -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j\vec{x}_i\cdot\vec{x}_j + c\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i(-1+\xi_i) - \sum_{i=1}^{N}\mu_i\xi_i \\
&= -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j\vec{x}_i\cdot\vec{x}_j + \sum_{i=1}^{N}\xi_i(c-\alpha_i-\mu_i) + \sum_{i=1}^{N}\alpha_i \\
L_D &\triangleq -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j\vec{x}_i\cdot\vec{x}_j + \sum_{i=1}^{N}\alpha_i
\end{aligned}$$

Further, note that equation (3.27) as well as the requirement $\alpha_i, \mu_i \geq 0$ for every i yields $0 \leq \alpha_i \leq c$. Thus the dual problem is

$$\begin{aligned}
&\textit{Maximize:} \quad -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j\vec{x}_i\cdot\vec{x}_j + \sum_{i=1}^{N}\alpha_i \\
&\textit{Subject to:} \quad 0 \leq \alpha_i \leq c \\
&\qquad\qquad \sum_{i=1}^{N}\alpha_i y_i = 0
\end{aligned} \tag{3.39}$$

which can easily be solved using numerical optimization.

Once this problem has been solved, the vector $\vec{w}$ is given by equation (3.28) and b is given by equation (3.35) by using an i such that $\alpha_i \neq 0$.

Kernel SVM

Up until now, a linear SVM classifier has been explained. However, what makes SVM so popular is the ability to make it nonlinear without too much trouble.

There are two ways to look at a nonlinear classifier. First, the separating surface is not a plane. Classification trees are one example of this, where rectangular boxes are used to divide the feature space. Another example is likelihood methods, where the decision boundary is usually a curved line.

The second way to look at a nonlinear classifier is by a transformation of the feature space. By transforming the feature space, a linear classifier could be used in the transformed feature space. This same linear classifier would be nonlinear in the original feature space. Mathematically, suppose the transformation is given by Φ. Then a linear SVM classifier is trained on the positive set, $\{\Phi(\vec{x}_i)|\ \ \forall i \text{ such that } y_i = 1\}$ and the negative set, $\{\Phi(\vec{x}_i)|\ \ \forall i \text{ such that } y_i = -1\}$.

The latter approach is extremely easy to implement without much change to the above framework. Consider problem (3.39). Solving this problem is all that is needed to solve the linear SVM problem. However, the only place the data, $\{\vec{x}_i\}_{i=}^N$, shows up is in the cost function as the dot product, $\vec{x}_i \cdot \vec{x}_j$, for every $i, j = 1, 2, ...N$. To train on the transformed feature space, simply change this dot product to an evaluation of the kernel, $K(\vec{x}_i, \vec{x}_j)$, where

$$K(\vec{x}_i, \vec{x}_j) = \Phi(\vec{x}_i) \cdot \Phi(\vec{x}_j) \tag{3.40}$$

This has been called the kernel trick. It can be used in most classification methods in which the data shows up strictly as inner products.

Why is this useful? After all, any classification scheme can be made nonlinear by using $\{\Phi(\vec{x}_i)\}_{i=1}^N$ instead of $\{\vec{x}_i\}_{i=1}^N$. The answer depends on the transformation Φ. The range of Φ can have any dimension, even infinity. Indeed, one of the most popular kernels is

$$K(\vec{x}_i, \vec{x}_j) = e^{-||\vec{x}_i - \vec{x}_j||^2/(2\sigma^2)}$$

This kernel is a valid kernel [2], i.e. there is a function Φ such that equation (3.40) holds [27]. However, Φ maps the feature space $\mathcal{X}$ to a space of infinite-dimensional, but finite length (in the L^2 norm sense), vectors. That is, it would be impossible to compute $\{\Phi(\vec{x}_i)\}_{i=1}^N$ explicitly.

The difficulty in computing $\{\Phi(\vec{x}_i)\}_{i=1}^N$ also yields some trouble in classifying new

data points. In the linear case, equations (3.28) and (3.35) can be used to find $\vec{w}$ and b. However, if $\{\Phi(\vec{x}_i)\}_{i=1}^N$ is being used instead of $\{\vec{x}_i\}_{i=1}^N$, this may not be possible.

This turns out to be no trouble at all, since the hyperplane is defined as

$$(\vec{w} \cdot \Phi(\vec{x}) + b) = 0$$

By equation (3.28), $\vec{w} = \sum_{i=1}^N \alpha_i y_i \Phi(\vec{x}_i)$ so this hyperplane becomes

$$\begin{aligned} 0 &= \vec{w} \cdot \Phi(\vec{x}) + b \\ &= \left(\sum_{i=1}^N \alpha_i y_i \Phi(\vec{x}_i) \right) \cdot \Phi(\vec{x}) + b \\ &= \sum_{i=1}^N \alpha_i y_i \left(\Phi(\vec{x}_i) \cdot \Phi(\vec{x}) \right) + b \\ &= \sum_{i=1}^N \alpha_i y_i K(\vec{x}_i, \vec{x}) + b \end{aligned}$$

This makes the classifier, call it h,

$$h(\vec{x}) = \begin{cases} 1 & \text{if} \quad \sum_{i=1}^N \alpha_i y_i K(\vec{x}_i, \vec{x}) + b \geq 0 \\ -1 & \text{else} \end{cases}$$

Application of this Method

The other methods discussed above are not linear classifiers. Support vector machines are applicable with all of the feature extraction techniques described above, but care must be taken to analyze the feature space and choose a kernel that transforms the data properly. This can be hard to do, as the feature space for these subwindows is difficult to visualize. Although it requires a bit of faith that a data set represents the distribution properly, the success of a kernel for an SVM classifier can be measured by the confusion matrix.

One thing to keep in mind is that the feature extraction methods used in this work are designed to separate the data. Thus in many cases like SVD features, Fisher discriminant features, Fourier modes, a linear SVM classifier will suffice. SVM may be less applicable with features such as the Haar-like features due to the fact that it does not choose the best features to use, and most likely will use every feature value instead.

3.3.2 Classification Trees

Tree-based methods partition a P-dimensional feature space into P-dimensional rectangles. For classification trees, each of these rectangles corresponds to a class.

Details of this Method

Suppose the data set consists of N feature vectors, $\vec{f}_1, \vec{f}_2, ..., \vec{f}_N$, taken from the feature space $\mathcal{F}$, with labels $\{y_i\}_{i=1}^N$. Training a tree involves making splits in the feature space, with each split corresponding to only one feature. To make a split, there are two arguments needed, the index, i, denoting which feature used, and the threshold, θ. Given a range in feature space, R, one split would have the form

$$\vec{f} \in \begin{cases} R_1 & \text{if} \quad f_{i_1} < \theta_1 \\ R_2 & \text{else} \end{cases}$$

As an example, region R_1 can be split again so that, for $\vec{f}$ in R_1,

$$\vec{f} \in \begin{cases} R_{1,1} & \text{if} \quad f_{i_2} < \theta_2 \\ R_{1,2} & \text{else} \end{cases}$$

These splits can be represented in a binary tree. A tree for the example above is shown in figure 3.4. Each spot where the tree splits is called an interior node and each location where the tree ends is called a terminal node.

Once the space has been partitioned into M regions, $R_1, R_2, ..., R_M$, it is easy to classify the regions. Let

$$q_{m,k} = \frac{1}{N_m} \sum_{\vec{f}_i \in R_m} I(y_i = k)$$

where $I(\cdot)$ is the indicator function, in other words, $q_{m,k}$ is the proportion of class k observations in region R_m. Then every $\vec{f} \in R_m$ is classified as

$$k(m) = \underset{k}{\text{argmax}} \ \ q_{m,k}$$

Thus the final classifier takes the form

$$h(\vec{f}) = \sum_{m=1}^{M} k(m) I(\vec{f} \in R_m)$$

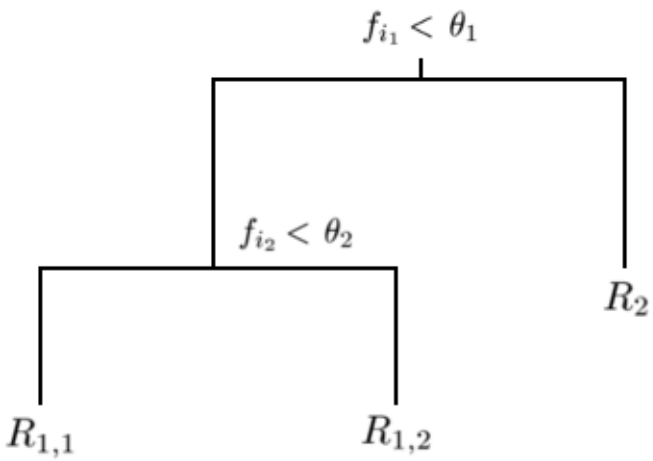

Figure 3.4: An example of a tree corresponding to the partitions mentioned earlier. When a feature vector $\vec{f}$ satisfies the criteria at a node, it moves to the left, otherwise it moves to the right.

Of course, making the splits is non-trivial. The algorithm needs to do this automatically. First, however, a criterion to be minimized must be chosen. This criterion is called the impurity measure and there are a few different choices that can be made. For a region R_m of a tree T, the following are different impurity measures, $Q_m(T)$, that can be used [28].

- Misclassification error: $\frac{1}{N_m} \sum_{\vec{f}_i \in R_m} I(y_i \neq k(m)) = 1 - q_{m,k(m)}$
- Gini index: $\sum_{k \neq k'} q_{m,k} q_{m,k'}$
- Cross-entropy: $\sum_{k=1}^{K} q_{m,k} \log q_{m,k}$

After the impurity measure is chosen, the algorithm looks for the optimal splitting variables and thresholds to minimize $Q(T) = \sum_{m=1}^{M} N_m Q_m(T)$. However, this is usually computationally infeasible. Instead, greedy algorithms are used.

Starting with all of the data, F, take a splitting index j and a threshold θ. Define

the sides of the hyperplane $f_j = \theta$ as follows.

$$R_1(j, \theta) = \{\vec{f} | f_j \leq \theta\}$$

$$R_2(j, \theta) = \{\vec{f} | f_j > \theta\}$$

Also let T be the tree obtained by making this split. Then choose j and θ as

$$\underset{j,\theta}{\operatorname{argmin}} \, Q(T)$$

The result of the previous problem is the best split according to the chosen impurity measure. Using this split, partition the data into the two resulting regions. The above process is then run recursively on each of the regions.

One problem remains. What should the size of the final tree be? A small tree might not be able to capture all the detail of the distribution. However, a large tree will clearly overfit the training data.

A widely used solution to this problem is to use a strategy called tree pruning. The first step in such a process is to grow a large tree, T_0. Then change one interior node to a terminal node resulting in a tree T_1, choosing the interior node so that the change brings about the smallest difference in the total impurity. This difference is

$$D_{j_1,j_2} = Q(T_0) - Q(T_1)$$

This process is repeated until the tree consists of one interior node and two terminal nodes, yielding a sequence $T_0, T_1, ..., T_S$.

To decide which tree is the best of $T_0, T_1, ..., T_S$, the following criterion is used [28].

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha |T|$$

where $|T|$ denotes the number of terminal nodes in the tree T, and $\alpha >= 0$ is a tuning parameter. Letting $\alpha = 0$ yields the usual impurity measure for the tree. Such a choice of α will always use the largest tree T_0, since adding more levels cannot increase the impurity. A strictly positive α will manage the tradeoff between tree size and goodness of fit. Large α will result in smaller trees and less overfitting but less goodness of fit, while small α values will yield the opposite. Choosing a suitable α can be accomplished by the cross-validation scheme in **Cross-Validation 2**.

Application of this Method

Like AdaBoost, discussed in the next section, the features that perform the best on the data set are the first ones chosen. This makes classification trees very useful with the overcomplete Haar-like feature set, as well as using the whole subwindows as the feature vectors. Regardless, they can be used with any feature type.

Use in the Attentional Cascade

Classification trees are also useful to create small and efficient classifiers for use in the attentional cascade. Each node represents one feature value. Using features such as the Haar-like features or simply the voxel intensity values will allow a simple classification tree to eliminate many of the negative windows without analyzing every voxel in the subwindow.

In the next section, a trick explained by Viola and Jones will be described which allows the AdaBoost classifier to be modified to meet a predefined detection rate. This same trick can be used on classification trees as well. Each interior node has a threshold associated to it. The threshold in the last interior node of each branch of a tree can be modified so that every subwindow containing a nodule is associated to a region that is labeled as a nodule. This will yield a detection rate of 100%, while substantially decreasing the number of negative subwindows.

3.3.3 AdaBoost

A strong learning algorithm is an algorithm that, given $\epsilon, \delta > 0$, outputs a hypothesis with an error at most ϵ, $(1 - \delta)100\%$ of the time. On the other hand, a weak learning algorithm satisfies the same conditions but for only each $\epsilon_t = \frac{1}{2} - \gamma_t$ where $\gamma_t > 0$ is either a constant or decreases at a rate inversely proportional to the number of parameters in the model. AdaBoost is an algorithm invented by Freud and Schapire [20] to "boost", or combine, a number of weak classifiers into one, strong classifier.

Details of this Method

Previous boosting algorithms required prior knowledge of the performance of the weak learner. However, AdaBoost adapts to the errors of the weak learner. The original algorithm is given by the steps below [20].

Algorithm:AdaBoost

1. Input N labeled examples $< (x_1, y_1), \ldots, (x_N, y_N) >$, distribution D over the N examples, weak learning algorithm **WeakLearn**, and integer T specifying number of iterations.

2. Initialize the weight vector $w_{t,1} = D(i)$ for $i = 1, \ldots, N$.

3. For $t = 1, 2, \ldots, T$, do

   a. Set
   $$\mathbf{p}_t = \frac{\mathbf{w}_t}{\sum_{i=1}^{N} w_{t,i}}$$

   b. Call **WeakLearn**, providing it with the distribution $\mathbf{p}_t$, get back a hypothesis $h_t : X \to \{0, 1\}$.

   c. Calculate the error of h_t, $\epsilon_t = \sum_{i=1}^{N} p_{t,i} |h_t(x_i) - y_i|$.

   d. Set $\beta_t = \dfrac{\epsilon_t}{1 - \epsilon_t}$.

   e. Set the new weights vector to be $w_{t+1,i} = w_{t,i} \beta_t^{1-|h_t(x_i)-y_i|}$.

4. Output the hypothesis
$$h(x) = \begin{cases} 1 & \text{if} \quad -\sum_{t=1}^{T} \log \beta_t h_t(x) \geq -\frac{1}{2} \sum_{t=1}^{T} \log \beta_t \\ 0 & \text{otherwise} \end{cases} \tag{3.41}$$

Let $S = \{i; h(x_i) \neq y_i\}$. Now a result that bounds the error

$$\epsilon = Pr_{i \in D}(h(x_i) \neq y_i) = \sum_{i \in S} w_{1,i}$$

which is the probability of an error, will be proven. First, however, the following two lemmas are introduced.

Lemma 1. *If* $0 \leq x$ *and* $0 \leq y \leq 1$*, then*

$$x^y \leq 1 - (1 - x)y \tag{3.42}$$

Proof. Let $f(t) = x^t$ for fixed $x \geq 0$. Then

$$f''(t) = (\log x)^2(x^t) \geq 0$$

This means that x^t is a convex function of t and thus the inequality,

$$x^{yt_1+(1-y)t_2} \leq yx_1^t + (1 - y)x_2^t$$

holds for $y \in [0, 1]$. Choosing $t_1 = 1$ and $t_2 = 0$ yields the result. □

Lemma 2. *After* T *rounds, the strong classifier,* h*, makes a mistake on instance* i *only if*

$$\prod_{t=1}^{T} \beta_t^{-|h_t(x_i)-y_i|} \geq (\prod_{t=1}^{T} \beta_t)^{-\frac{1}{2}} \tag{3.43}$$

Proof. From the final step in the algorithm above, the final hypothesis is

$$h(x) = \begin{cases} 1 & \text{if} \quad -\sum_{t=1}^{T} \log \beta_t h_t(x) \geq -\frac{1}{2} \sum_{t=1}^{T} \log \beta_t \\ 0 & \text{else} \end{cases}$$

This is the same as

$$h(x) = \begin{cases} 1 & \text{if} \quad \log \prod_{t=1}^{T} \beta_t^{-h_t(x)} \geq \log \prod_{t=1}^{T} \beta_t^{-\frac{1}{2}} \\ 0 & \text{else} \end{cases}$$

which can be rewritten as

$$h(x) = \begin{cases} 1 & \text{if} \quad \prod_{t=1}^{T} \beta_t^{-h_t(x)} \geq \prod_{t=1}^{T} \beta_t^{-\frac{1}{2}} \\ 0 & \text{else} \end{cases}$$

Showing that (3.43) holds when the final classifier makes a mistake will prove our claim.

Case 1: $y_i = 0$ **but** $h(x_i) = 1$**:**
Since $h(x_i) = 1$,

$$\prod_{t=1}^{T} \beta_t^{-h_t(x_i)} \geq \prod_{t=1}^{T} \beta_t^{-\frac{1}{2}}$$

However, $y_i = 0$ and $h_t(x_i) \geq 0$, so

$$\prod_{t=1}^{T} \beta_t^{-|h_t(x)-y_i|} \geq \prod_{t=1}^{T} \beta_t^{-\frac{1}{2}}$$

Case 2: $y_i = 1$ **but** $h(x_i) = 0$**:**

Now, $h(x_i) = 0$, so

$$\prod_{t=1}^{T} \beta_t^{-h_t(x_i)} < \prod_{t=1}^{T} \beta_t^{-\frac{1}{2}}$$

Since $y_i = 1$, this can be written as

$$\prod_{t=1}^{T} \beta_t^{-1+|h_t(x_i)-y_i|} < \prod_{t=1}^{T} \beta_t^{-\frac{1}{2}}$$

Multiplying by $\prod_{t=1}^{T} \beta_t$ on both sides yields

$$\prod_{t=1}^{T} \beta_t^{|h_t(x_i)-y_i|} < \prod_{t=1}^{T} \beta_t^{\frac{1}{2}}$$

or

$$\prod_{t=1}^{T} \beta_t^{-|h_t(x_i)-y_i|} > \prod_{t=1}^{T} \beta_t^{-\frac{1}{2}}$$

□

Theorem 2. $\epsilon \leq 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t(1-\epsilon_t)}$

Proof. First,

$$\begin{aligned}
\sum_i^N w_{T+1,i} &= \sum_i^N w_{T,i} \beta_t^{1-|h_t(x_i)-y_i|} \\
&\leq \sum_i^N w_{T,i}(1-(1-\beta_t)(1-|h_t(x_i)-y_i|)) \text{ by Lemma 3.42} \\
&= (\sum_i^N w_{T,i}) - (1-\beta_t)(\sum_i^N (w_{T,i} - w_{T,i}|h_t(x_i)-y_i|)) \\
&= (\sum_i^N w_{T,i}) - (1-\beta_t)(\sum_i^N w_{T,i})(\sum_i^N (\frac{w_{T,i}}{\sum_i^N w_{T,i}} - \frac{w_{T,i}}{\sum_i^N w_{T,i}}|h_t(x_i)-y_i|) \\
&= (\sum_i^N w_{T,i}) - (1-\beta_t)(\sum_i^N w_{T,i})(1-\epsilon_t)
\end{aligned}$$

Where the last equality holds because $\epsilon_t = \sum_{i=1}^{N} w_i^t \, |h_t(x_i) - y_i|$ with the weights already normalized. This yields

$$\sum_{i}^{N} w_{T+1,i} \leq (\sum_{i}^{N} w_{T,i})(1 - (1 - \beta_t)(1 - \epsilon_t))$$

Iterating this process,

$$\sum_{i}^{N} w_{T+1,i} \leq \prod_{t=1}^{T} (1 - (1 - \epsilon_t)(1 - \beta_t)) \tag{3.44}$$

On the other hand, for each instance, i,

$$\begin{aligned} w_{T+1,i} &= w_{T,i} \, \beta_T^{1-|h_T(x_i)-y_i|} \\ &= w_{T-1,i} \, \beta_{T-1}^{1-|h_{T-1}(x_i)-y_i|} \beta_T^{1-|h_T(x_i)-y_i|} \\ &= \vdots \\ &= w_{1,i} \, \prod_{t=1}^{T} \beta_t^{1-|h_t(x_i)-y_i|} \end{aligned}$$

Thus,

$$w_{T+1,i} = D(i) \prod_{t=1}^{T} \beta_t^{1-|h_t(x_i)-y_i|} \tag{3.45}$$

Recall that S is the set of indices where the strong classifier makes an error. Thus the following holds

$$\begin{aligned} \sum_{i=1}^{N} w_{T+1,i} &\geq \sum_{i \in S} w_{T+1,i} \\ &= \sum_{i \in S} D(i) \prod_{t=1}^{T} \beta_t^{1-|h_t(x_i)-y_i|} \text{ by (3.45)} \\ &\geq (\sum_{i \in S} D(i)) \, (\prod_{t=1}^{T} \beta_t)^{\frac{1}{2}} \text{ by (3.43)} \\ &= \epsilon \, (\prod_{t=1}^{T} \beta_t)^{\frac{1}{2}} \end{aligned}$$

Now bounding ϵ,

$$\begin{aligned}\epsilon &\leq \frac{\sum_{i=1}^{N} w_{T+1,i}}{\prod_{t=1}^{T}(\beta_t)^{\frac{1}{2}}} \\ &\leq \frac{\prod_{t=1}^{T}(1-(1-\epsilon_t)(1-\beta_t))}{\prod_{t=1}^{T}(\beta_t)^{\frac{1}{2}}}\end{aligned}$$

So,

$$\epsilon \leq \prod_{t=1}^{T} \frac{(1-(1-\epsilon_t)(1-\beta_t))}{(\beta_t)^{\frac{1}{2}}} \tag{3.46}$$

Further, $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$, so

$$\begin{aligned}\epsilon &\leq \prod_{t=1}^{T} \frac{(1-(1-\epsilon_t)(1-\frac{\epsilon_t}{1-\epsilon_t}))}{(\frac{\epsilon_t}{1-\epsilon_t})^{\frac{1}{2}}} \\ &= \prod_{t=1}^{T} \frac{1-1+\epsilon_t+\frac{\epsilon_t}{1-\epsilon_t}-\frac{\epsilon_t^2}{1-\epsilon_t}}{(\frac{\epsilon_t}{1-\epsilon_t})^{\frac{1}{2}}} \\ &= \prod_{t=1}^{T}(1-\epsilon_t)^{\frac{1}{2}}(\epsilon_t^{\frac{1}{2}}+\frac{\epsilon_t^{\frac{1}{2}}}{1-\epsilon_t}-\frac{\epsilon_t^{\frac{3}{2}}}{1-\epsilon_t}) \\ &= \prod_{t=1}^{T}\sqrt{\epsilon_t\,(1-\epsilon_t)}\frac{1-\epsilon_t+1-\epsilon_t}{1-\epsilon_t} \\ &= \prod_{t=1}^{T} 2\,\sqrt{\epsilon_t\,(1-\epsilon_t)} \\ &= 2^T \prod_{t=1}^{T}\sqrt{\epsilon_t\,(1-\epsilon_t)}\end{aligned}$$

□

In fact, from (3.46) one can see where the choice of β_t comes from. Looking for the sharpest bound possible, the right hand side of (3.46) is minimized with respect to β_t. This yields the aforementioned choice, $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.

In addition to giving a bound on the error, this theorem says more. It says that the final error depends on the error of each of the weak hypotheses. This is in contrast to previous bounds on errors of boosting algorithms, which only depended on the error of the weakest hypothesis. Considering the many distributions that a weak learner can be based upon, this is a very relevant advantage to the practical applications of boosting.

Application of this Method

AdaBoost can be used with any of the feature extraction methods used above. It is particularly useful with the Haar-like features, since it will choose the features that perform the best. It is also possible to fix either the number of features used, or the desired error rate. This, as well as the non-parametric nature of AdaBoost, make this a very suitable classifier for most data sets.

Even more useful is the ability to weigh examples in the data set. Since it is much more important to detect all the nodule examples, and less important to prevent false detections, the weighting should be larger for nodules and less for non-nodules. i.e.

$$D(i) > D(j)$$

for i corresponding to nodule subwindows and j corresponding to non-nodule subwindows.

The weak learner can be any classifier that performs with better than 50% accuracy. Due to this flexibility, decision stumps are a very popular approach and the one used in this thesis. While the performance of each decision stump is marginal, they can be trained to have an error rate that is under 50%. Further, they are extremely fast to evaluate, which makes these weak classifiers very attractive. Let

$$\vec{f} = (f_1, f_2, ..., f_K)^T$$

Then a decision stump g consists of an index k, a threshold θ, and a polarity p and takes the following form.

$$g(\vec{f}) = \begin{cases} 1 & \text{if} \quad p\, f_k \geq p\, \theta \\ 0 & \text{else} \end{cases}$$

where the result of 1 corresponds to deciding the subwindow is a non-nodule and 0 corresponds to deciding the subwindow is a nodule. A decision stump can be viewed as a classification tree with one parent and two child nodes.

Use in the Attentional Cascade

As mentioned earlier, Viola and Jones used a series of AdaBoost classifiers in their attentional cascade [19], and suggest a method to use a very small number of features while maintaining a detection rate of close to 100%. They reported that a two-dimensional Haar-like feature AdaBoost classifier was trained to use only two features and still detect 100% of the faces with a false positive rate of 40%.

The AdaBoost classifier does not do this on its own. Instead, after the two-feature classifier was trained, the threshold in equation (3.41) is lowered to yield a 100% detection rate. This threshold,

$$\theta = -\frac{1}{2}\sum_{t=1}^{T}\log\beta_t$$

can be modified to be the smallest feature value so that the detection rate is 100%.

Chapter 4

Analysis on Sample Data Set

4.1 Data

Acquiring expert annotated data is a very important step to developing a CAD system. It is imperative that the training set captures the complete distribution of both nodule and non-nodule subwindows. Since the dimension of the subwindow is $12 \cdot 12 \cdot 6 = 864$, the training set should have an extremely large number of samples. Given a thoracic CT scan, it is easy to generate an extremely large number of non-nodule subwindows. However, generating unique nodule subwindows is a much more difficult task since there are usually only at most a handful of nodules for each CT scan. This is a problem that cannot be overcome easily, as there are few public databases of thoracic CT scans available.

The database used in this work is the result of a collaboration between the International Early Lung Cancer Action Program (I-ELCAP) and the Vision and Image Analysis Group (VIA). Their Public Lung Image Database [29] consists of 50 low-dose documented whole-lung CT scans. These images have an approximate 1.25 mm slice thickness with resolution of about .75 mm $\times$ .75 mm on each slice. Nodules in the images have been detected and recorded by radiologists. The database can be accessed at `http://www.via.cornell.edu/databases/lungdb.html`.

For the ease of experimentation, only nodules fitting inside a $12 \times 12 \times 6$ subwindow will be considered. As mentioned earlier, it is necessary to resize the subwindow and

then modify the classifier accordingly, in order to detect smaller and larger nodules. However, using a fixed size will still allow for a good estimate of performance, since the rates should be similar across different sized subwindows. The goal of this section is to report on the accuracy of the methods described in the earlier section, regardless of runtime. Hence tools to speed up algorithms, such as the integral image and attentional cascade, will not be implemented in these experiments.

The sample data set just described is used in this section to compare the results from each pair of feature extraction techniques and classification methods. For initial comparisons, the confusion matrix will be given for each pair. As mentioned above, the confusion matrix yields the detection rate, false positive rate, and false negative rate.

When cross-validation needs to be used to learn parameters, the complete training set is randomly divided up into 5, approximately equally sized, sets. As explained above, a classifier will be trained on every combination of 4 subsets, leaving the 5th for validation in the form of a confusion matrix. The confusion matrices are then summed into one confusion matrix.

In each case where it is necessary, cross-validation is again used to estimate an optimal value of the parameter. The parameters are chosen as to minimize the loss function, which is the sum of number of false positives with a weighted sum of the number of false negatives. The training set consists of nodule and non-nodule subwindows with a ratio of approximately 3.5 non-nodule subwindows for every nodule subwindow. For the initial experimentation, a weight of 7 is used for every false negative. Since a weighting of 3.5 would penalize both types of errors the same, 7 penalizes false negatives twice as much as false positives. Since the original data set was split up into training and testing sets, the parameters must be learned on only the training set. Thus the training set is split up again, to yield a nested training and testing set for cross-validation to learn the parameters. A diagram illustrating the overall idea is shown in figure 4.1.

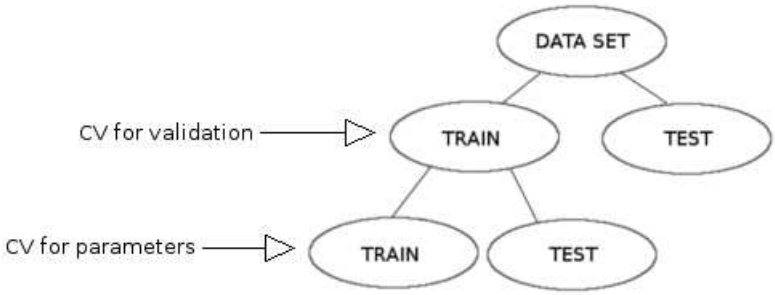

Figure 4.1: Illustration of how each iteration of cross-validation is partitioned.

4.2 The Classifiers and Implementations Used

There were approximately five classifiers described above: likelihood methods, Probabilistic D-Clustering, SVM, classification trees, and AdaBoost. Unfortunately, the statistical distributions of the two classes, nodules and non-nodules, in each feature space are unknown. This prevents the use of Probabilistic D-Clustering for this problem.

While likelihood methods also require knowledge of the distribution, a multivariate normal distribution assumption leads to some respectable results. However, it will be clear that likelihood methods are outperformed by other non-parametric classifiers. Likelihood methods and Probabilistic D-Clustering algorithms could be revisited at a later time, when more knowledge of the statistical distributions are known.

There are many implementations of SVM and classification trees available. Since MATLAB was used for all experiments, the MATLAB implementations of SVM and classification trees were used. Documentation of these algorithms can be found on the MathWorks website.

The implementation of AdaBoost used was the RealAdaBoost function created by Alexander Vezhnevets [30]. The weak learners are classification trees, but in this work they are chosen to be decision stumps which are explained above.

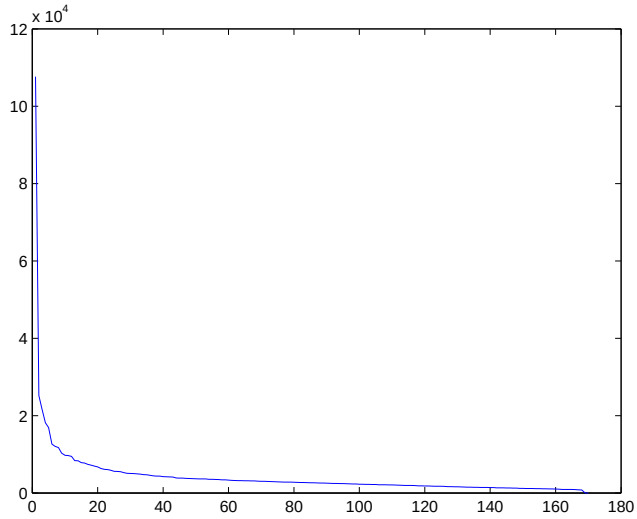

Figure 4.2: A plot of the singular values from the set of all nodule subwindows. The x-axis gives the index of the singular value.

4.3 SVD Feature Results

The singular value decomposition on the nodule subwindows gives a complete basis for all of the $12 \times 12 \times 6$ subwindows. The basis elements are ordered by the importance relative to the nodule subwindows. So before beginning, an important question to answer is how many singular vectors should be used. As explained above, the singular values give an idea of the importance of each singular vector. Thus, looking at these values should help give a rough idea how many singular vectors should be used. The plot of the singular values for the complete training set is shown in figures 4.2 and 4.3. From figure 4.3, it looks like only the first 6 singular values are somewhat useful in describing nodules, while the others explain noise.

While this observation gives some insight into the importance of each singular vector, it is not necessary to base the classifiers on it. Rather, the number of singular vectors used can be viewed as a tuning parameter to be learned by cross-validation. The number of singular values is a tuning parameter for every classifier. In addition, the classifier

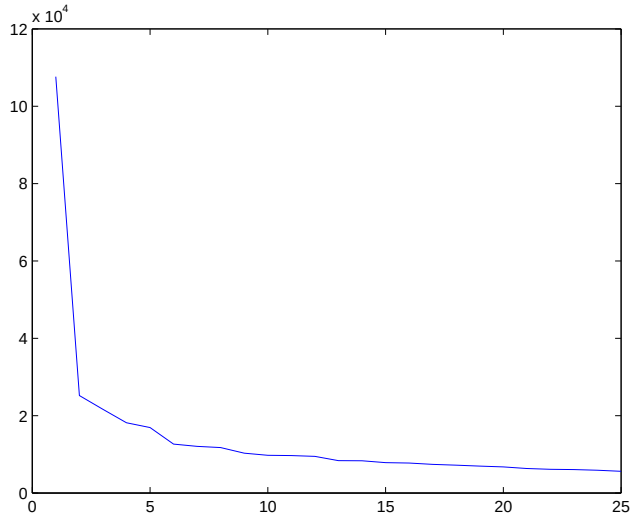

Figure 4.3: A plot of the first 25 singular values as shown in figure 4.2

based on likelihood requires a threshold to be learned.

4.3.1 Likelihood

As mentioned earlier, the inner product between the singular vectors and non-nodule subwindows are expected to be very small since these windows will not look much like nodule subwindows. Thus this classification method makes the assumption that the non-nodule subwindows are distributed according to a multivariate normal distribution. A classifier is thus found by learning a threshold on the likelihood that each sample fits the distribution.

Let the mean and covariance of this multivariate distribution be denoted by $\vec{\mu}$ and Σ, respectively. Then the likelihood of a P-dimensional sample, $\vec{x}$, is

$$P(\vec{x}) = \frac{1}{(2\pi)^{P/2}\det(\Sigma)^{1/2}} \exp\left(-\frac{1}{2}(\vec{x}-\vec{\mu})^T\Sigma^{-1}(\vec{x}-\vec{\mu})\right)$$

With threshold c on the likelihood, the classifier, which outputs 1 when deciding a

nodule and 0 otherwise, is thus

$$C(\vec{x}) = \begin{cases} 1 & \text{if} \quad P(\vec{x}) < c \\ 0 & \text{else} \end{cases}$$

But $P(\vec{x}) < c$ implies the following.

$$\begin{aligned} \frac{1}{(2\pi)^{P/2}\det(\Sigma)^{1/2}} \exp\left(-\frac{1}{2}(\vec{x}-\vec{\mu})^T\Sigma^{-1}(\vec{x}-\vec{\mu})\right) &< c \\ \exp\left(-\frac{1}{2}(\vec{x}-\vec{\mu})^T\Sigma^{-1}(\vec{x}-\vec{\mu})\right) &< c_1 \\ -\frac{1}{2}(\vec{x}-\vec{\mu})^T\Sigma^{-1}(\vec{x}-\vec{\mu}) &< c_2 \\ (\vec{x}-\vec{\mu})^T\Sigma^{-1}(\vec{x}-\vec{\mu}) &> c_3 \end{aligned}$$

Since c is still arbitrary, this simplifies the classifier to

$$C(\vec{x}) = \begin{cases} 1 & \text{if} \quad (\vec{x}-\vec{\mu})^T\Sigma^{-1}(\vec{x}-\vec{\mu}) > c \\ 0 & \text{else} \end{cases}$$

the parameter c is learned by cross-validation as mentioned at the beginning of this chapter.

The confusion matrix for this approach on the data set is as follows.

SVD/Likelihood		
	Classified as Nodule	Classified as Non-nodule
Nodules	162	8
Non-nodules	161	355

which corresponds to a 95.3% detection rate but a 31.2% false positive rate. The method selected around 5 singular vectors, which is consistent with the above discussion on singular vectors. The false positive rate is very large, which may suggest that the multivariate normality assumption incorrect.

For multivariate normal distributed data $\{\vec{x}_i\}$, the statistic

$$D_i = (\vec{x}_i - \vec{\mu})^T\Sigma^{-1}(\vec{x}_i - \vec{\mu})$$

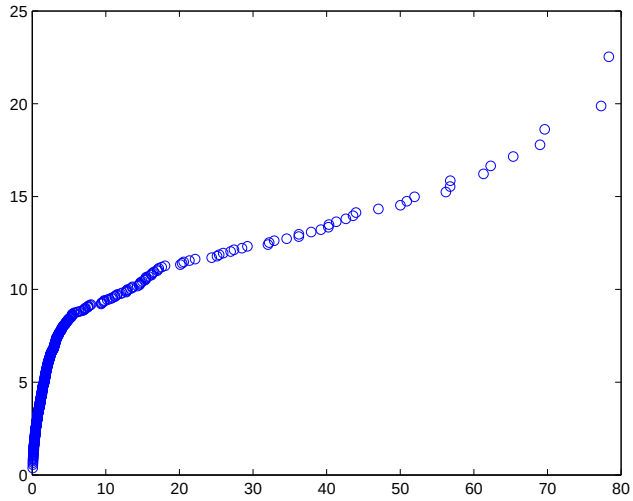

Figure 4.4: The Q-Q plot of D_i vs. the χ^2_P quantiles. A linear Q-Q plot implies the data are multivariate normal distributed and vice versa. However, this plot is definitely not linear.

should be χ^2_P distributed, where P is the degrees of freedom which is equal to the dimension of the data [31]. That is, the Q-Q plot of D_i against the corresponding χ^2_P quantiles should be roughly linear. Figure 4.4 shows this plot. It looks like the Q-Q plot is *not* linear and thus data are not normally distributed. This means the likelihood classifier above has problems, which explains the large false positive rate.

4.3.2 SVM

Linear SVM was applied to the data set with cross-validation to learn the number of singular vectors. The confusion matrix is below.

SVD/SVM (Linear)		
	Classified as Nodule	Classified as Non-nodule
Nodules	141	29
Non-nodules	19	497

That is, an 82.9% detection rate and only a 3.7% false positive rate.

Using kernel SVM is a good idea since a linear hyperplane is not necessarily a good assumption. A polynomial kernel would allow for a more flexible classifier. The order p polynomial kernel is defined as [2]

$$k(\vec{x}, \vec{y}) = (\vec{x} \cdot \vec{y} + 1)^p$$

The results from a polynomial kernel SVM are as follows, with $p = 3$.

SVD/SVM (Polynomial Kernel)		
	Classified as Nodule	Classified as Non-nodule
Nodules	165	5
Non-nodules	25	491

This is a 97.1% detection rate and a 4.8% false positive rate.

Finally, the Gaussian kernel is one of the most popular choices. It is a radial basis function (RBF) that is based on the distance to the centers, which, in the SVM case, are the support vectors. It is defined as follows.

$$k(\vec{x}, \vec{y}) = \exp\left(-\frac{||\vec{x} - \vec{y}||^2}{2\sigma^2}\right)$$

Using a value of $\sigma^2 = 1$ produces the following results on the data set.

SVD/SVM (RBF Kernel)		
	Classified as Nodule	Classified as Non-nodule
Nodules	162	8
Non-nodules	18	498

This corresponds to a 95.3% detection rate and only a 3.5% false positive rate.

4.3.3 Classification Tree

The classification trees were trained using the MATLAB "treefit" function. The results are as follows.

SVD/Classification Tree		
	Classified as Nodule	Classified as Non-nodule
Nodules	158	12
Non-nodules	29	487

This is a 92.9% detection rate and only a 5.6% false positive rate.

4.3.4 AdaBoost

AdaBoost was used on the data set with cross-validation again used to learn the number of singular vectors. The number of weak classifiers used was arbitrarily limited at 500. The confusion matrix is below.

SVD/AdaBoost		
	Classified as Nodule	Classified as Non-nodule
Nodules	163	7
Non-nodules	14	502

These results correspond to a 95.9% detection rate and only a 2.7% false positive rate.

4.3.5 Summary of SVD Methods

Below is a summary of the SVD methods, ordered by appearance.

SVD Methods			
Classifier	Detection Rate (%)	False Positive Rate (%)	Average Number of Vectors Used
Likelihood	95.3	31.2	5
SVM (Linear Kernel)	82.9	3.7	17
SVM (Polynomial Kernel)	97.1	4.8	8
SVM (R.B.F. Kernel)	95.3	3.5	5
Classification Tree	92.9	5.6	6
AdaBoost	95.9	2.7	14

4.4 Modified SVD Feature Results

As explained above, the modified SVD chooses feature vectors to separate the two classes by decomposing the matrix, H, into left singular vectors, singular values, and right singular vectors. In this experiment, H is defined as follows.

$$H = \lambda(\vec{x}_{\mu_1} - \vec{x}_{\mu_2})(\vec{x}_{\mu_1} - \vec{x}_{\mu_2})^T - \xi S_{X_1}^2 - (51)S_{X_2}^2 \tag{4.1}$$

where λ and ξ are tuning parameters chosen from the set $\{1, 26, 51, 76, 101\}$, which weight the importance of the various terms.

Like the SVD, the set of vectors given by the modified SVD on the data set gives a complete basis for all of the $12 \times 12 \times 6$ subwindows. This time, the basis elements are ordered by the importance relative to the separation of the nodule subwindows group and the non-nodule subwindows group. Again, the number of singular values used is a tuning parameter. The plot of the singular values for the complete training set for $\lambda, \xi = 51$ is shown in figures 4.5 and 4.6. This shows that, again, only about the first six singular vectors are very important.

4.4.1 Likelihood

Before training a classifier based on likelihood with normality assumptions, consider again the Q-Q plot in figure 4.7. This is the analog of figure 4.4, on the modified SVD

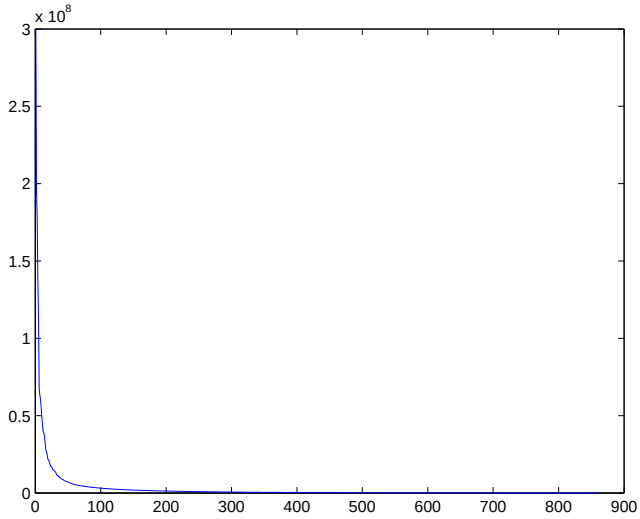

Figure 4.5: A plot of the singular values of the decomposition of H. The x-axis gives the index of the singular value.

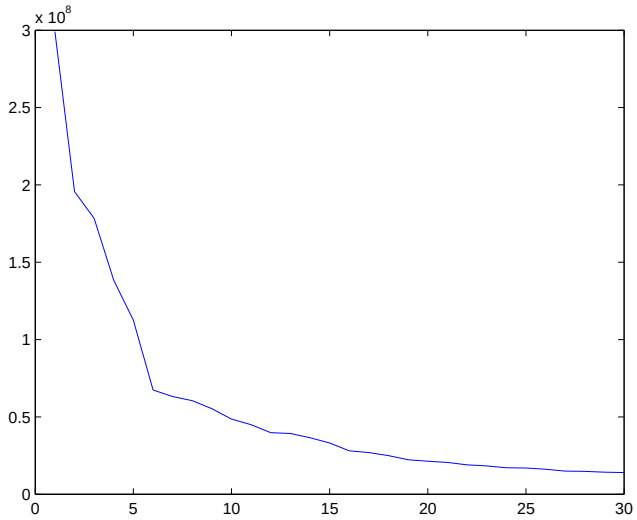

Figure 4.6: A plot of the first 30 singular values as shown in figure 4.5

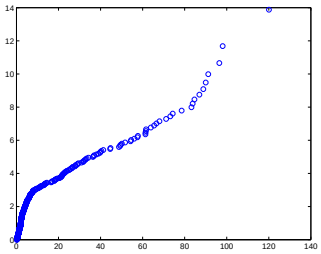

Figure 4.7: The Q-Q plot for the non-nodule class based on multivariate normality. The curve does not look linear, indicating normality was a bad assumption.

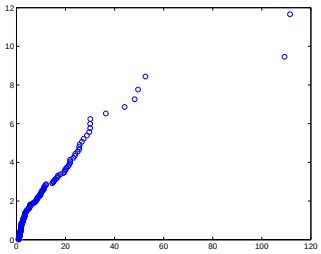

Figure 4.8: The Q-Q plot for the nodule class based on multivariate normality. Since the curve look roughly linear, a likelihood classifier may yield good results.

features. Again, this plot is for samples in the non-nodules class. Again, this does not look very linear, which probably means the classifier will not be optimal. Indeed, this intuition is reflected in the confusion matrix below.

Modified SVD/Likelihood on Non-nodules		
	Classified as Nodule	Classified as Non-nodule
Nodules	162	8
Non-nodules	89	427

This is a 95.3% detection rate and a 17.2% false positive rate, which is quite poor.

Now consider the Q-Q plot for the nodules class in figure 4.8. This plot looks roughly linear, which could mean that a classifier based on likelihood that a sample is derived from the nodule distribution will perform well. Unfortunately, an implementation of this likelihood classifier returned very poor results. This is most likely due to one of two things. First, the data are probably not normal for most of singular vectors chosen. Second, the covariance matrices require a large amount of data to train. When using the nodule class, there are many less subwindows that can be used for training which results in a singular covariance matrix. This yields big problems when trying to invert the covariance matrix for use in the likelihood function.

4.4.2 SVM

An SVM classifier will now be trained on the Modified SVD feature space. The following confusion matrix gives the results for a linear kernel.

Modified SVD/SVM (Linear)		
	Classified as Nodule	Classified as Non-nodule
Nodules	141	29
Non-nodules	22	494

That is a 94.0% detection rate and only a 4.3% false positive rate. While these results are satisfactory, a nonlinear kernel will allow for more flexibility. The following results correspond to nonlinear kernels.

Modified SVD/SVM (Polynomial Kernel)		
	Classified as Nodule	Classified as Non-nodule
Nodules	165	5
Non-nodules	40	476

Thus a polynomial kernel yields a 97.1% detection rate and a 7.8% false positive rate.

Modified SVD/SVM (RBF Kernel)		
	Classified as Nodule	Classified as Non-nodule
Nodules	165	5
Non-nodules	15	501

This corresponds to a 97.1% detection rate and only a 2.9% false positive rate.

4.4.3 Classification Tree

Now a classification tree is trained on this feature space.

Modified SVD/Classification Tree		
	Classified as Nodule	Classified as Non-nodule
Nodules	159	11
Non-nodules	14	502

This is a 93.5% detection rate and only a 2.7% false positive rate. That is, classification trees do a great job eliminating false positives.

4.4.4 AdaBoost

With classification trees working well, an AdaBoost classifier should also yield good results.

Modified SVD/AdaBoost		
	Classified as Nodule	Classified as Non-nodule
Nodules	163	7
Non-nodules	15	501

These results correspond to a 95.9% detection rate and only a 2.9% false positive rate.

4.4.5 Summary of Modified SVD Methods

Below is a summary of the modified SVD methods, ordered by appearance.

Modified SVD Methods			
Classifier	Detection Rate (%)	False Positive Rate (%)	Average Number of Vectors Used
Likelihood	95.3	17.2	2
SVM (Linear Kernel)	94.0	4.3	18*
SVM (Polynomial Kernel)	97.1	7.8	18*
SVM (R.B.F. Kernel)	97.1	2.9	4
Classification Tree	93.5	2.7	8
AdaBoost	95.9	2.9	13

(Superscript " * " denotes a classifier in which the number of vectors used was not a tuning parameter. In these cases, cross-validation became too time consuming as the empirical M.S.E. was reduced with each vector added.)

4.5 Tensor Decomposition Results

The tensor decomposition methods are quite similar to the SVD, but a little less straight forward. First, the Tucker decomposition is used. This was chosen due to the fact that it is regarded as a higher order SVD, a convenient analog to the two-dimensional SVD. Furthermore, the Tucker decomposition has proven to be very useful in image compression [32]. The N-way toolbox for MATLAB, created by Andersson and Bro [33] was used.

There is one difference when adapting the methods used in the previous sections to tensor decomposition features. When using tensor decomposition features, there is no good way to choose the basis functions. With the SVD, the singular values ordered importance and the number of basis elements used could be trained as a tuning parameter. As mentioned above, the core matrix of the Tucker decomposition does *not* give any insight into the importance of each basis element. This means there is no simple tuning parameter to be trained with cross-validation, and thus cross-validation is not needed. This is not necessarily a bad thing, however as described above, the previous

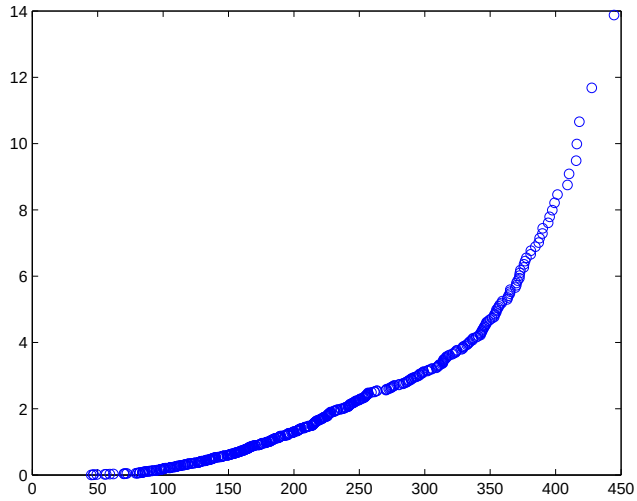

Figure 4.9: The Q-Q plot for the features extracted from the tensor decomposition of the non-nodules subwindows. The trend is nonlinear, hence the density most likely not normal.

tuning parameters were trained according to a certain loss function. This gives some flexibility to the classifier to cater to the loss function. With no parameters to train according to a loss function, the results will be optimized according to the internal loss function of the classifiers (hinge loss for SVM, exponential loss for AdaBoost, etc..). This same reasoning applies to Fourier modes, where there are also no tuning parameters.

4.5.1 Likelihood

Looking at the Q-Q plot in figure 4.9, it is clear that the class of non-nodule subwindows in the feature space is not normal. Thus the likelihood classifier will not be implemented in this section.

4.5.2 SVM

As mentioned earlier, SVM is a non-parametric approach and might be able to classify this feature space. The following results are for SVM classifiers.

Tensor Features/SVM (Linear)		
	Classified as Nodule	Classified as Non-nodule
Nodules	144	26
Non-nodules	59	457

Thus a linear kernel yields an 84.7% detection rate and an 11.4% false positive rate.

Tensor Features/SVM (Polynomial)		
	Classified as Nodule	Classified as Non-nodule
Nodules	81	89
Non-nodules	70	446

These results show that a polynomial kernel is a poor choice. The detection rate is 47.6% with a 13.6% false positive rate.

Tensor Features/SVM (RBF)		
	Classified as Nodule	Classified as Non-nodule
Nodules	4	166
Non-nodules	0	516

Clearly an RBF kernel is no better than a polynomial kernel for this feature space. For SVM classifiers operating in the tensor decomposition feature space, it is best to stick with a linear kernel. Since the feature space for tensor decompositions is generally higher than the SVD and modified SVD, other kernels may perform better with more training data.

4.5.3 Classification Tree

A classification tree was also trained on this feature space.

Tensor Features/Tree		
	Classified as Nodule	Classified as Non-nodule
Nodules	151	19
Non-nodules	32	484

This is an 88.9% detection rate and a 6.2% false positive rate.

4.5.4 AdaBoost

With classification trees yielding marginal results, the AdaBoost classifier may show an improvement. The results using this classifier are below.

Tensor Features/AdaBoost		
	Classified as Nodule	Classified as Non-nodule
Nodules	162	8
Non-nodules	9	507

This is a 95.3% detection rate and a very low 1.7% false positive rate.

4.5.5 Summary of Tensor Decomposition Methods

Below is a summary of the tensor decomposition methods, ordered by appearance.

Fourier Modes Methods		
Classifier	Detection Rate (%)	False Positive Rate (%)
SVM (Linear Kernel)	84.7	11.4
SVM (Polynomial Kernel)	47.6	13.6
Classification Tree	88.9	6.2
AdaBoost	95.3	1.7

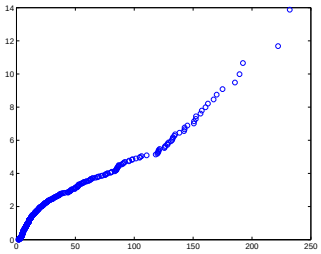

Figure 4.10: The Q-Q plot for the non-nodule class based on multivariate normality. The curve does not look linear, indicating normality is a bad assumption.

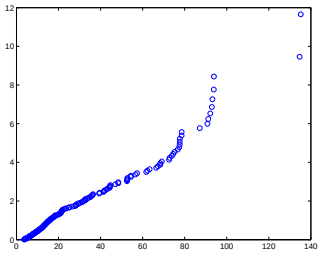

Figure 4.11: The Q-Q plot for the nodule class based on multivariate normality. Since the curve look roughly linear, a likelihood classifier may yield good results.

4.6 Fourier Modes

4.6.1 Likelihood

First, consider the Q-Q plot for the non-nodules class in figure 4.10 and for the nodules class in 4.11. The Q-Q plot for the nodule class looks roughly linear. This means that the normality assumption could be a good assumption, and a likelihood classifier may yield good results.

First, a look at the confusion matrix for the classifier based on the non-nodules class reaffirms the discussion above; the results are poor.

Fourier Modes/Likelihood on Non-nodules		
	Classified as Nodule	Classified as Non-nodule
Nodules	163	7
Non-nodules	363	153

This matrix shows a 95.8% detection rate and a 70.3% false positive rate.

Now the confusion matrix for the classifier based on the nodules class.

Fourier Modes/Likelihood on Nodules		
	Classified as Nodule	Classified as Non-nodule
Nodules	51	119
Non-nodules	24	492

This is a 30% detection rate and a 4.6% false positive rate. Clearly a likelihood classifier is not a good choice.

4.6.2 SVM

Now an SVM classifier is trained to classify this feature space.

Fourier Modes/SVM (Linear)		
	Classified as Nodule	Classified as Non-nodule
Nodules	147	23
Non-nodules	27	489

That is, an 86.5% detection rate and a 5.2% false positive rate for a linear kernel. The following results show the performance of a nonlinear SVM classifier.

Fourier Modes/SVM (Polynomial Kernel)		
	Classified as Nodule	Classified as Non-nodule
Nodules	169	1
Non-nodules	49	467

Thus a polynomial kernel yields a 99.4% detection rate and a 9.5% false positive rate.

Fourier Modes/SVM (RBF Kernel)		
	Classified as Nodule	Classified as Non-nodule
Nodules	42	128
Non-nodules	0	516

This corresponds to an unacceptable 32% detection rate with a 0% false positive rate.

That is, a RBF kernel does not work well on this feature space.

4.6.3 Classification Tree

A classification tree will now be used to classify the feature space. The results are below.

Fourier Modes/Classification Tree		
	Classified as Nodule	Classified as Non-nodule
Nodules	150	20
Non-nodules	20	496

This is a 88.2% detection rate and only a 3.9% false positive rate.

4.6.4 AdaBoost

Now, the results of an AdaBoost classifier are presented.

Fourier Modes/AdaBoost		
	Classified as Nodule	Classified as Non-nodule
Nodules	170	0
Non-nodules	5	511

Thus AdaBoost yields a perfect detection rate and only a 1% false positive rate.

4.6.5 Summary of Fourier Modes Methods

Below is a summary of the Fourier modes methods, ordered by appearance.

Fourier Modes Methods		
Classifier	Detection Rate (%)	False Positive Rate (%)
Likelihood (Non-nodule)	95.8	70.3
Likelihood (Nodule)	30.0	4.6
SVM (Linear Kernel)	86.5	5.2
SVM (Polynomial Kernel)	99.4	9.5
SVM (R.B.F. Kernel)	32.0	0.0
Classification Tree	88.2	3.9
AdaBoost	100.0	1.0

4.7 Haar-Like Feature Results

As described above, only features of size $4 \times 4 \times 2$ are used. Despite this limitation, about 9,720 features are present in each subwindow due to the $9 \cdot 9 \cdot 5 = 405$ distinct locations in each subwindow. Since the dimension of the feature space is so large, only two classification schemes will be used for the Haar-like features, AdaBoost and classification trees. These classifiers are chosen because they only choose a small subset of the features. This is in contrast to, for example, SVM, where linear combinations of all the features are considered.

As one might expect, both the classification tree and AdaBoost end up choosing the same feature first. This feature can be visualized in figure 4.12. On the right side, red and blue indicate subtracting and adding the voxels intensity from the feature value, respectively. The feature type is all blue except for one red corner. It is difficult to see, but the red area is close to the very center of the subwindow, thus almost directly over where a nodule would be. Perhaps one would expect that the top and bottom corner should be red, instead of only the top. Such a feature would probably be chosen with a larger, more complete training set.

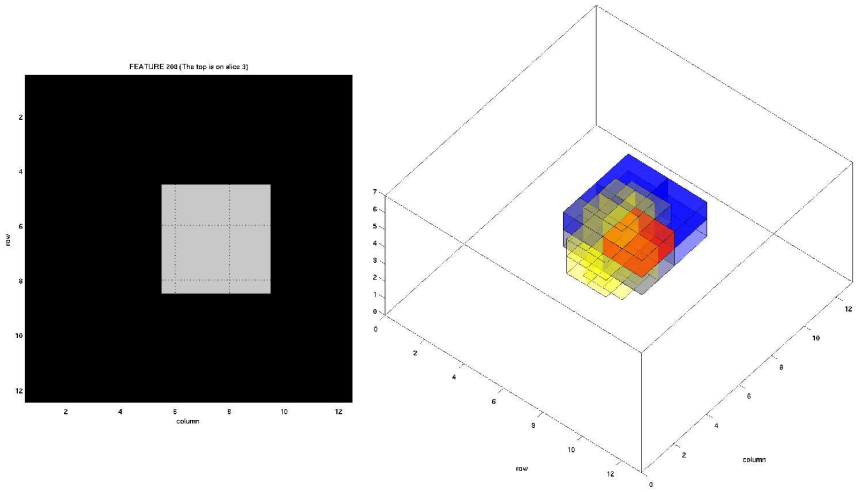

Figure 4.12: The first feature chosen by both AdaBoost and the classification tree. On the left is a top view of the subwindow to see where the feature is placed. On the right, the feature is overlayed onto a spherical yellow blob in the center of the subwindow. Here, the blue indicates the voxel's intensity will be added into the feature value and red indicates the voxel's intensity will be subtracted. (Since this is hard to see, figure 4.13 below may give a more clear picture of the yellow nodule.)

4.7.1 Classification Tree

While the parent node of the tree can be seen in figure 4.12, the first two children can be seen in figures 4.13 and 4.14. It is important to note that nodules are different sizes and not always perfectly centered in subwindows. Thus, for example, even though it does not look like the feature in figure 4.13 touches the nodule, it will overlap with many of the actual nodules.

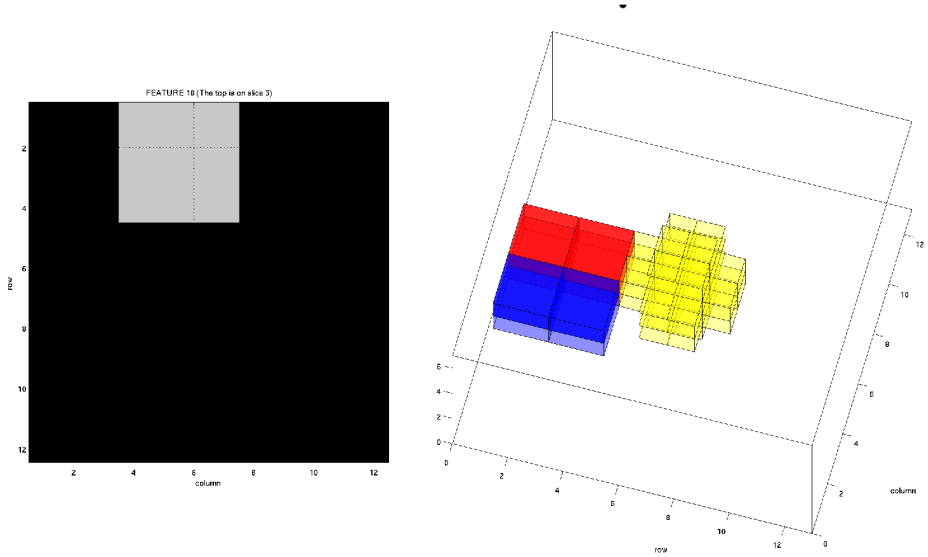

Figure 4.13: The left child node directly under the node based on the feature in figure 4.12.

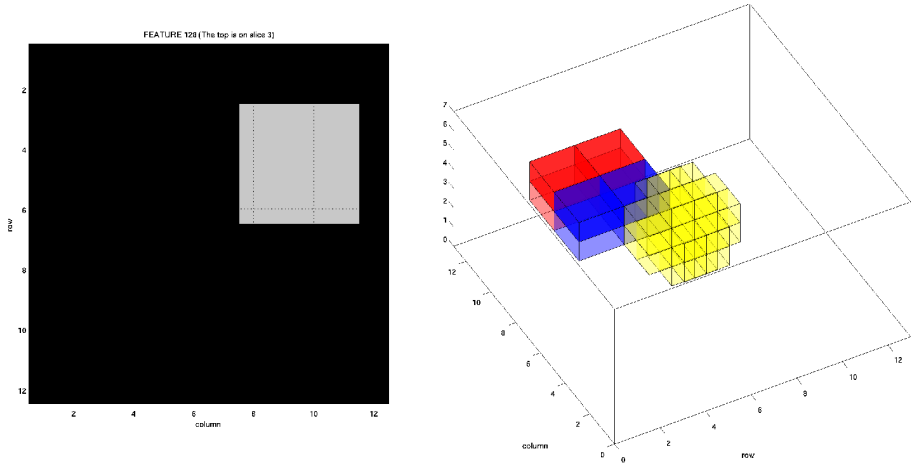

Figure 4.14: The right child node directly under the node based on the feature in figure 4.12.

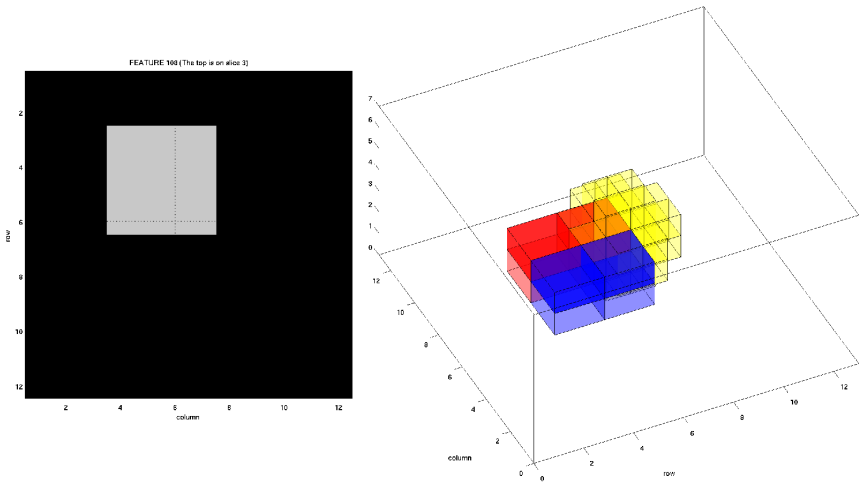

Figure 4.15: The second feature chosen by AdaBoost, where the first feature is shown in figure 4.12. Note that the image on the right has been rotated upside down for better visualization.

Haar-Like Features/Classification Tree		
	Classified as Nodule	Classified as Non-nodule
Nodules	153	17
Non-nodules	17	499

This is a 90.0% detection rate and a 3.3% false positive rate.

4.7.2 AdaBoost

The second feature chosen by AdaBoost can be seen if figure 4.15. Intuitively, the red area is in the center of the subwindow.

Haar-Like Features/AdaBoost		
	Classified as Nodule	Classified as Non-nodule
Nodules	164	6
Non-nodules	12	504

This is a 96.5% detection rate and a 2.3% false positive rate.

4.7.3 Summary of Haar-Like Feature Methods

Below is a summary of the Haar-like features methods, ordered by appearance.

Haar-Like Feature Methods		
Classifier	Detection Rate (%)	False Positive Rate (%)
Classification Trees	90.0	3.3
AdaBoost	96.5	2.3

4.8 Summary of the Best Performing Methods

A summary of the best performing methods will be organized by false positive rate for detection rates above 90% and false positive rates below 5%. This summary is given in table 4.1

The first thing that one might notice from this table is that the AdaBoost classifier almost always outperforms the other classifiers that were considered. This speaks to the flexibility of the AdaBoost classifier to partition the feature space properly. Classification trees partition the feature space in a similar way, but are more susceptible to overfitting, which explains this classifiers role in table 4.1. The SVM classifier performs satisfactorily in some instances. However, in the necessary step of choosing the kernel, much of the flexibility of the classifier is lost as the kernel determines how the feature space is partitioned.

Another immediate observation is the existence of each of the following feature extractions outlined in this paper in table 4.1. This supports the claim that these feature spaces are roughly separable, and shows that there is large amount of potential for these feature extraction methods to be utilized in this problem.

Finally, as expected, the parametric methods discussed in this paper do not appear in this summary of best performing methods. These feature spaces are too complicated

Best Performing Methods by False Positive Rate		
Feature/Classifier	Detection Rate (%)	False Positive Rate (%)
Fourier Modes / AdaBoost	100.0	1.0
Tensor Features / AdaBoost	95.3	1.7
Haar-Like Features / AdaBoost	96.5	2.3
SVD / AdaBoost	95.9	2.7
Modified SVD / Classification Trees	93.5	2.7
Modified SVD / SVM (R.B.F. Kernel)	97.1	2.9
Modified SVD / AdaBoost	95.9	2.9
Haar-Like Features / Classification Trees	90.0	3.3
SVD / SVM (R.B.F. Kernel)	95.3	3.5
Modified SVD / SVM (Linear Kernel)	94.0	4.3
SVD / SVM (Polynomial Kernel)	97.1	4.8

Table 4.1: Summary of results

to explain with a statistical distribution without an in depth statistical analysis.

Chapter 5

Discussion

5.1 A Summary of the System

This work concentrates on a method for computer aided detection (CAD) of lung nodules. Detecting nodules is a very time consuming task for physicians. Such a CAD system has great potential as an aid for physicians or a second reader.

Previous CAD systems have been developed for this task, but there is much room for improvement. Most previous systems rely on segmentation of all structures in the lung, extracting features from these structures, and feeding to a classifier. However, problems with the segmentation algorithm, failure to explain the entire structure with the ad-hoc extracted features, and other issues result in undesirable detection rates. This thesis attempts to eliminate the above problems.

The CAD algorithm in this thesis avoids the segmentation step by sweeping a three-dimensional subwindow throughout the entire image. At each location, the subwindow is directly classified as either containing a nodule or not containing a nodule. There is no segmentation step necessary.

To extract features from subwindows, rather than hand pick features to extract in an ad-hoc manner, this thesis describes several methods for learning a basis on the subwindow in such a way that the subwindows for nodules and non-nodules are separated in the feature space. By doing so, no information is lost in feature extraction, as it may be when hand picking features from structures. Several methods for doing this feature

extraction were experimented with in this work, including the SVD, a modification of the SVD, tensor decompositions, Haar-like features, and Fourier modes.

5.2 Interpretation of the Results

The experimental results show several interesting things. They show that the considered feature spaces do a good job of separating the two classes of subwindows. Also, it is clear that parametric classifiers will not work without a better understanding of the complicated distributions in feature space. Finally, the results show that the subwindow approach has potential to yield great performance on the problem of detection lung nodules.

5.2.1 Success in Feature Extraction

This thesis experimented with the following methods to extract features: the SVD, a modification of the SVD, tensor decompositions, Haar-like features, and Fourier modes. These feature spaces were all able to be roughly separated by classifiers into the two classes, nodules and non-nodules, well enough to obtain desirable detection and false positive rates.

The comparable results over all of these feature spaces reflect their ability to preserve most of the information included in the original subwindows. This shows that, while there is room for improvement in some of the feature extractions, it may be the classifier that is the limiting factor. A better understanding of these feature spaces may lead to better classifier and thus better results.

5.2.2 Poor Performance with Parametric Methods

Two parametric classifiers were studied in this thesis, an ordinary likelihood classifier and Probabilistic D-Clustering. Likelihood classifiers use likelihood to choose which class an observation belongs to and Probabilistic D-Clustering uses distance to the class centers to choose a class label.

The clustering algorithm relies on a distance function which determines the distribution of the data. Since this choice is somewhat arbitrary, the assumed distribution and actual distribution are very different. This problem caused the method to fail to converge to the class centers, making it useless as a classifier.

The likelihood method is simple; the likelihood that an observation is in each class is computed and the class with the highest likelihood is chosen as the conclusion. However, to compute likelihood, the statistical distribution must be known. In the case of this lung nodule detection problem, the distribution is not known. Many times, a normal approximation works well. However, such approximations used in this thesis produce undesirable results. It is clear that parametric methods will not be useful until there is a better understanding of each of the feature spaces.

5.2.3 Potential of Subwindow Approach

Paired with a suitable classifier, each feature extraction method succeeded in obtaining above a 95% detection rate with under a 3% false positive rate. This is in comparison to detection rates of under 85% for previous methods mentioned in this thesis. (The false positive rate is not immediately comparable.) Clearly this approach has great potential in automated lung nodule detection.

5.3 Future Directions

The methods discussed in this thesis produce good results. However, there is much room for improvement. This leads to the following areas of future research.

5.3.1 Statistical Analysis

When a statistical distribution is known, the best classifier will always be parametric. However, the distribution of each class in each of these feature spaces is not known. For this reason, the non-parametric classifiers outperform the parametric classifiers in the experimental results. However, there is potential to improve the accuracy of detection by

better understanding the statistical distributions of the classes, and use this information to construct a parametric classifier.

5.3.2 Training Data

As mentioned earlier, training data is very important. The database used in this paper only contained 50 CT images from 50 different subjects. Each subject has a number of nodules ranging from about one to thirty.

This database gives a good amount of data, but not nearly enough considering how many pixels are in one subwindow, 864. While a sufficient number of non-nodule subwindows can be extracted from this database for training, there are only about 80 nodule subwindows that can be used. This is simply not enough data to understand the class of nodule subwindows. More data would also allow for a better understanding of the statistical distributions as discussed above.

5.3.3 Improving the Haar-Like Feature Approach

Using only Haar-like features of size $4 \times 4 \times 2$, an AdaBoost classifier was able to detect 96.5% of the nodule subwindows with only a 2.3% false positive rate. This was one of the best methods used in this paper in terms of the results. However, the implementation used was very limited in both feature size and feature type.

By implementing every combination of size, orientation, and type, the feature and classifier pair of Haar-like features and AdaBoost could outperform the other methods. The experiment above only chooses features from a pool of about $9,720$ distinct features. This is a large number of features, but only a small percentage of the features of all different sizes, orientations, and types. Including all of these Haar-like features yields the possibility of improving results while not increasing the runtime of the detection process.

5.3.4 Tuning Parameters and Classifiers

The goal of this research was to investigate the subwindow approach to lung nodule detection. However, there is room to improve each of the implementations in this paper. Tuning parameters, such as number of singular vectors to use, rank of tensor decomposition, and weights on the terms in the modification of the SVD, could be better learned by a cross-validation with a finer mesh than the one used in this research.

There is some additional work that can be done in choosing the classifiers. AdaBoost and classification trees require the user to specify the level of complexity. This too could be achieved with a cross-validation scheme. Also, there are many kernels that can be used with an SVM classifier, not just linear, polynomial, and radial basis functions. An in depth comparison of kernels could yield a much better classifier in many of the feature spaces discussed above.

5.3.5 Nodules on Walls

As mentioned previously, the methods discussed above only apply to isolated and juxta-vascular nodules. Juxta-pleural nodules are not considered. For a complete lung nodule detection system, a system would have to be developed to handle this class of nodules.

5.3.6 Complete System

Finally, it is difficult to completely understand the benefits and shortfalls of this system at this time. For such a study, a complete lung nodule detection system would have to be developed. This is because subwindows overlap, and neighboring subwindows contain information about each other.

For example, suppose you have a chain of subwindows identified as nodules running from the top slice of the lung to the bottom slice of the lung. This chain of subwindows is most likely a blood vessel that has continually been identified as a nodule in each of the subwindows. However, using the proximity of the subwindows in this chain, a conclusion could be made that they are, in fact, identifying parts of a blood vessel.

On the other hand, suppose you have a group of six subwindows identified as nodules which are clustered in a small area. This group most likely corresponds to a nodule that the overlapping subwindows all classified correctly. Thus more confidence should be put on these classifications, while a single isolated subwindow identified as a nodule without any neighboring subwindows identified as nodules gives evidence that there is not really a nodule present.

Such a system would increase the detection rates produced by the above experiments and decrease the false positive rates. It would also produce false positive rates on a per case basis, allowing a fair comparison between this method and many other methods in the literature.

Bibliography

[1] Yongbum Lee, Takeshi Hara, Hiroshi Fujita, Shigeki Itoh, and Takeo Ishigaki. Automated detection of pulmonary nodules in helical CT images based on an improved template-matching technique. *IEEE Trans. Medical Imaging*, 20:595–604, 2001.

[2] Christopher J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[3] A. Jemal, R. Siegel, E. Ward, Y. Hao, J. Xu, T. Murray, and M. J. Thun. Cancer statistics, 2008. *CA: a cancer journal for clinicians*, 58(2):71–96, 2008.

[4] European Society for Medical Oncology. Early detection of lung cancer. *ScienceDaily*, May 2009.

[5] LungCancer.org. `http://www.lungcancer.org`, Accessed July 2009.

[6] Maryellen Lissak Giger, Kunio Doi, and Heber MacMahon. Image feature analysis and computer-aided diagnosis in digital radiography. 3. Automated detection of nodules in peripheral lung fields. *Medical Physics*, 15(2):158–166, 1988.

[7] Michael Aschenbeck. Learning Techniques for Segmentation, Recognition, and Diagnostics in Medical Imaging. Plan B Project Report, University of Minnesota. Available from the author, 2008.

[8] Peter E. Hart, Richard O. Duda, and David G. Stork. *Pattern Classification*. Wiley, 2nd edition, 2001.

[9] Jane P. Ko and Margrit Betke. Chest CT: Automated nodule detection and assessment of change over time–preliminary experience. *Radiology*, 218(1):267–273, 2001.

[10] Metin N. Gurcan, Berkman Sahiner, Nicholas Petrick, Heang-Ping Chan, Ella A. Kazerooni, Philip N. Cascade, and Lubomir Hadjiiski. Lung nodule detection on thoracic computed tomography images: Preliminary evaluation of a computer-aided diagnosis system. *Medical Physics*, 29(11):2552–2558, 2002.

[11] S Calvard and TW Ridler. Picture thresholding using an iterative selection method. *IEEE Trans Syst Man Cybern*, 8(8), 1978.

[12] Binsheng Zhao. Automatic detection of small lung nodules on CT utilizing a local density maximum algorithm. *Journal of Applied Clinical Medical Physics*, 4(3), 2003.

[13] Matthew S. Brown, Jonathan G. Goldin, Robert D. Suh, Michael F. McNitt-Gray, James W. Sayre, and Denise R. Aberle. Lung micronodules: Automated method for detection at thin-section CT–initial experience. *Radiology*, 226(1):256–262, 2003.

[14] V Hlavac, R Boyle, and M Sonka. *Image processing, analysis and machine vision.* Chapman & Hall Computing, 1993.

[15] C.C. McCulloch, R.A. Kaucic, P.R. Mendonca, D.J. Walter, and R.S. Avila. Model-based detection of lung nodules in computed tomography examps. *Academic Radiology*, 11(3):258–266, 20004.

[16] David S. Paik, Christopher F. Beaulieu, Geoffrey D. Rubin, Burak Acar, R. Brooke Jeffrey Jr., Judy Yee, Joyoni Dey, and Sandy Napel. Surface normal overlap: a computer-aided detection algorithm with application to colonic polyps and lung nodules in helical ct. *IEEE Trans. Med. Imaging*, 23(6):661–675, 2004.

[17] Kyongtae T. Bae, Jin-Sung Kim, Yong-Hum Na, Kwang Gi Kim, and Jin-Hwan Kim. Pulmonary nodules: Automated detection on CT images with morphologic matching algorithm–preliminary results. *Radiology*, 236(1):286–293, 2005.

[18] Zhanyu Ge, Berkman Sahiner, Heang-Ping Chan, Lubomir M. Hadjiiski, Philip N. Cascade, Naama Bogot, Ella A. Kazerooni, Jun Wei, and Chuan Zhou. Computer-aided detection of lung nodules: False positive reduction using a 3D gradient field method and 3D ellipsoid fitting. *Medical Physics*, 32(8):2443–2454, 2005.

[19] Paul Viola and Michael Jones. Robust real-time object detection. In *International Journal of Computer Vision*, 2001.

[20] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting, 1997.

[21] Lloyd N. Trefethen and David Bau. *Numerical Linear Algebra*. pub-SIAM, pub-SIAM:adr, 1997.

[22] Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3), September 2009 (to appear).

[23] Ledyard Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, September 1966.

[24] Charles K. Chui. *An Introduction to Wavelets*. Academic Press Professional, Inc., San Diego, CA, USA, 1992.

[25] Adi Ben-Israel and Cem Iyigun. Probabilistic D-clustering. *Journal of Classification*, 25(1):5–26, 2008.

[26] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[27] L. Escauriaza. The taylor series of the gaussian kernel. *Journal of Nonlinear and Convex Analysis*, 7(3):405–410, 2006.

[28] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, August 2001.

[29] Elcap public lung image database. http://www.via.cornell.edu/databases/lungdb.html, Accessed July 2009.

[30] Alexander Vezhnevets. GML AdaBoost MATLAB toolbox. http://research.graphicon.ru/machine-learning/gml-adaboost-{MATLAB}-toolbox.html, Accessed July 2009.

[31] D. R. Cox and N. J. H. Small. Testing multivariate normality. *Biometrika*, 65(2):263–272, 1978.

[32] Hongcheng Wang and Narendra Ahuja. Rank-R approximation of tensors: Using image-as-matrix representation. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2*, pages 346–353, Washington, DC, USA, 2005. IEEE Computer Society.

[33] C.A. Andersson and R. Bro. The N-way toolbox for MATLAB. *Chemometrics & Intelligent Laboratory Systems*, 52(1):1–4, 2000.