Matrix Methods for Geometric Data Analysis and Pattern Recognition
Cats and Dogs Classification Project

Ryan de Vera
Hector Valdez
Torin Gerhart
Michael de Guzman

May 20, 2012

Abstract

In this project we were given 80 images of cats and 80 images of dogs which was to be used as a training set to classify another set of 38 images containing cats and dogs. In this project four different methods were used for classification. The first method uses the Average and Laplacian filter followed by Linear Discriminant Analysis. The second method uses an k-nearest neighbors search with an adaptive metric. The third method uses the Voronoi Diagram. Finally, the last method used the SVM (Support Vector Machine) method. The classification rates are 97.37%, 94.73%, 89.47%, and 86.84%, respectively.

1 Introduction

The human mind is an incredible machine. The human mind has the ability to recognize people, animals, and places. The goal of this project is to develop a mathematical method that can reproduce this process of image recognition. Everyday we take in large amounts of data from different sources all around us. The mind then uses all of this data and experience to build a basis for recognizing a plethora of different objects. In our case we want to create a basis for recognizing images of cats and dogs. If one were to look at the images below;

Figure 1: Image of a Cat

Figure 2: Image of a Dog

one could easily decipher that the first image is a cat and the second image is a dog. The viewer would know this based on experience and training. One of the main factors of experience in the deduction of cats and dogs are from specific features in both animals that come from shapes and distinct edges of each animal. In general, cats have pointed ears and a flatter face. Cats also have round eyes. Dogs usually have a larger snout. Dogs eyes are also much different from that of a cat. Also, in general, dogs have ears that do not point upward.

In this project we want to recreate the image recognition process of the mind. In particular we want to verify whether a given image is either a cat or a dog. We are given a training set of 160 cats and dogs. The training set consists of exactly 80 cats and exactly 80 dogs. With this information we want to create an algorithm that will help deduce the differences between cats and dogs and be able to classify a probe set containing 38 images of cats and dogs as either a cat of a dog by comparison with the testing set.

2 Theoretical Background

2.1 Average and Laplacian Filter

The averaging filter is a smoothing spatial filter. An important application of spatial averaging is to blur an image for the purpose of getting a gross representation of objects of interest, such that the intensity of smaller objects blends with the background and larger images become ”bloblike” and easy to detect. The size of the filter/mask establishes the relative size of the objects that will be blended with the background. A typical averaging filter has the form

$$AF = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

Another way to look at this process is convolution as filters. Convolution is considered a form of spatial filtering in manipulating images.

We use the Laplacian to sharpen (extract) edges. The Laplacian is a derivative operator. Therefore, it highlights intensity discontinuities in an image and deemphasizes regions with slowly varying intensity levels. This will tend to produce images that have grayish edge lines and other discontinuities, all superimposed on a dark, featureless background. Background features can be ”recovered” while still preserving the sharpening effect of the Laplacian simply by adding the Laplacian image to the original. By doing this it restores the overall intensity variations in the image, with the Laplacian increasing the contrast at the locations of intensity discontinuities. The final result is an image in which small details were enhanced and the background was reasonably preserved [1]. An example of the Laplacian filter is given by

$$\nabla^2 F = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}.$$

2.2 Principal Component Analysis

Principal Component Analysis is an important tool when working with large data sets because it lets us effectively reduced the dimensions of a large data set while successfully retaining the most important information of the original data. The main step of PCA is the extraction of the eigenvalues and eigenvectors using Singular Value Decomposition. Once we have extracted the eigenvalues and eigenvectors, we can use a plot of the eigenvalues to determine the eigenvectors that contain the most prominent information. We then use the left singular vectors as a KL basis, which is well-known as an optimal basis that achieves dimensionality reduction of the data matrix.

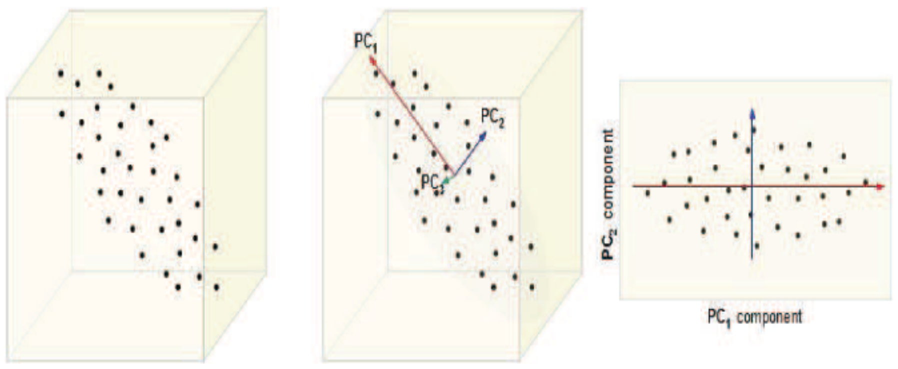

Figure 3: Principal Component Analysis

2.3 Linear Discriminant Analysis

The goal of LDA is to find a vector w, which is an optimal direction of projection. We want to find this w because when this optimal projection is found, two distinct classes can be separated without error. So given a point x, the idea behind FDA is to

1) Project x onto the line spanned by a certain unit vector w, thereby reducing the dimension of x.

2) Select the class of x based on whether $w^T x$ is above or below a certain scalar critical value α_c.

With that, optimally seeking w and α_c will be our main focus.

Now, in order to find the optimal projection we have to separate data in distinct classes as far as possible while pull the data within the same class as close as possible. In other words LDA explicitly attempts to model the difference between the classes of data [1]. This idea is displayed graphically below.

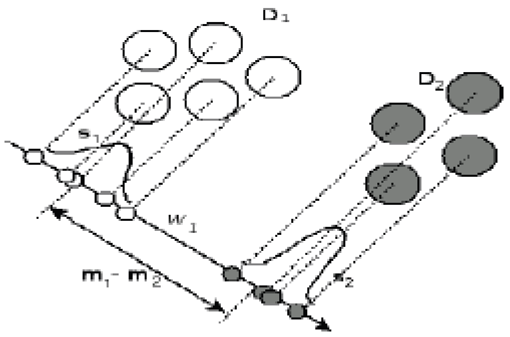

Figure 4: This is an image of an optimal direction vector w, that maximizes the between-class scatter and minimizes the within-class scatter.

The Fisher criterion that we must satisfy is calculated to be;

$$J(w) = \frac{w^T S_B w}{w^T S_W w},$$

which is also commonly known as the generalized Rayleigh Quotient. Setting $\nabla J(w) = 0$, a necessary condition for a global maximum, gives the generalized eigenvalue problem

$$S_B w = \lambda S_W w.$$

We need to decipher which of the eigenvectors to this generalized eigenvector problem will give an optimal projection, w. From the equation above we notice that

$$J(w) = \lambda.$$

Since we want to maximize $J(w) = \lambda$ we will find w as the generalized eigenvector corresponding to the largest generalized eigenvalue solving $S_B w = \lambda S_W w$.

Finally, we have to find the value α_c. One way of finding such a scalar can be calculated by taking the average of the projected means. The equation for this is given by,

$$\alpha_c = \frac{\tilde{m}_1 + \tilde{m}_2}{2}.$$

The generalized eigenvector problem can be solved with more than one method. The method presented in this document is by method of Principal Component Analysis (PCA) and then solving the eigenvalue problem using Singular Value Decomposition.

Reducing the dimensionality of the problem is the main benefit from using PCA. You might have classes which contain a lot of data but not all of this data may be important for finding a solution. Using PCA we can find the dimensions with the most information and reduce the dimensionality of the classes therefore making computations easier and more memory efficient when using numerical methods. From before we know that the left eigenvectors, U, are an optimal basis for our matrix which contains the data from both classes. PCA also calculates the expansion coefficients, A. Where, A is also a matrix of reduced dimension which has the expansion coefficients as its elements. The expansion coefficients represent the projections of the data onto the optimal spatial eigenvectors. Therefore, we may use the reduced expansion coefficients to calculate the between-class and within-class scatter matrices. Futhermore, this makes sense geometrically because our goal is to separate data in distinct classes as far as possible while pulling the data from the same class as close as possible. So the scatter matrices can be calculated using the optimal spatial dimensions. After the calculations using PCA we can now solve the eigenvector problem

$$(S_W)^{-1}S_Bw = \lambda w.$$

Why couldn't we have just solved this problem to begin with? In general, S_W is not an invertible matrix. However, after we reduce the dimensions of the problem we obtain the dimensions which contain the most data. Therefore in a reduced dimension S_W is invertible. Note here that the inverse operation is the pseudoinverse because we are not guaranteed to have a square matrix when we perform this calculation. The scalar α_c is calculated easily at this point by taking the average of the projected means.

2.4 Adaptive k-Nearest Neighbor Search

The k-nearest neighbor algorithm was first proposed in the 1950's as a means to classify data. It relies on classifying a new data point by assigning it the most frequently occurring class of the nearest k points. Since classification depends on a majority vote, k should be odd. If k is even, then a tie-breaking scheme is necessary. The traditional k-nearest neighbor search preforms poorly when the data is not clearly separated, or is highly mixed. This happens to be the case with images of cats and dogs. However, there is a simple way to improve the algorithms performance by slightly modifying the metric used to calculate distances. Since classification is based on smallest distances, by introducing a weighting system between classes the misclassification rate will go down. For each point in a particular class, the smallest distance to another class is computed. That is, r_i is found such that,

$$r_i = \min_{k:Y_k \neq Y_i} d(x_i, x_k) - \epsilon \tag{1}$$

where Y_i stands for the class of x_i. Weighting the distance by $1r_i$ will make the minimum distance between classes 1. This will give the nearest points in the same class an advantage and provide an opportunity for that class to be assigned[3]. The new metric becomes,

$$d_{new}(x,\ x_i) = \frac{d(x,\ x_i)}{r_i} \tag{2}$$

There are a few things to note about this metric. First, it is not a true metric as symmetry is lost ($d(x, y) \neq d(y, x)$). However, this is of little importance from an implementation point of view, since this property of metrics is not used in the algorithm. Also, it is essentially a metric specific to each individual point. That is, each point of training data has a specific distance function associated with it.

2.5 Discrete Haar Wavelet Transform

In *Geometric Data Analysis* by Michael Kirby, wavelet expansions are defined as

$$f(x) = \sum_{j=-\infty}^{\infty}\sum_{k=-\infty}^{\infty} d_k^j \phi_k^j(x),$$

where the $\left\{d_k^j\right\}$ are referred to as the wavelet coefficients. The functions

$$\phi_k^j(x) = 2^{-j/2}\phi_k^j(x-k)$$

are generated by the translations and dilations of the mother wavelet $\phi(x)$ on the dyadic grid [2]. The integers $j, k \in \mathbb{Z}$ correspond to the scale and location of the center of the function, respectively. The wavelet expansion is local in the sense that only a few of the expansion coefficients $\left\{d_k^j(x)\right\}$ contribute to the sum of the series $\sum_{j,k} d_k^j \phi_k^j(x)$ around any given point $x = x_0$.

The scaling coefficients for the Haar wavelet function are $h_0 = h_1 = 1/\sqrt(2)$. Thus

$$\phi(x) = \sqrt{2}(\frac{1}{\sqrt{2}}\phi(2x) + \frac{1}{\sqrt{2}}\phi(2x-1)) = \phi(2x) + \phi(2x-1).$$

The Haar scaling function has compact support, meaning that it is 0 everywhere outside a finite interval called the support. The width of the support is 1. Thus, the Haar wavelet function is

$$\psi(x) = \begin{cases} 1 & 0 \leq x < \ 1/2 \\ -1 & 1/2 \leq x < \ 1 \\ 0 & \textit{otherwise.} \end{cases}$$

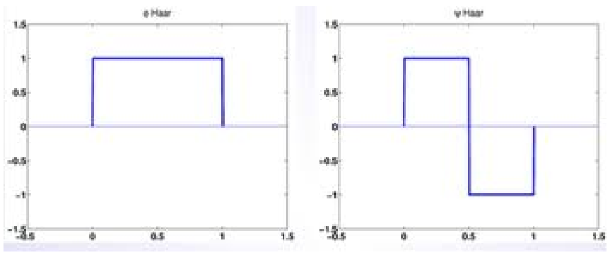

Figure 5: Figure on the left is the scaling coefficient for the Haar wavelet function while the figure on the right is the Haar wavelet function. Both have compact support from the interval [0,1].

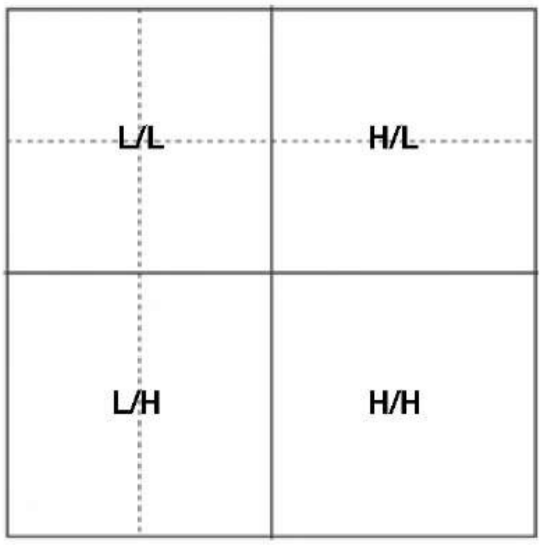

Figure 6: Pyramidal decomposition. Upper right partition would be the image reduced to 25 percent of it's original image. Upper right partition is the high-low pass which yields a horizontal edge extraction. Lower left partition is the low-high pass which yields a vertical edge extraction. Lower right partition is a high-high pass.

2.6 Voronoi Tessellation

Definition: Let X be $\mathbb{R}^n$. Given a set of points $\mathbf{x}_i \in X$, there exists a natural decomposition of X with $\mathbf{x}_i \in V_i \subset X$, $i = 1, 2, ..., n$, such that $\bigcup_{i=1}^{n} V_i = X$ and $V_i \bigcap V_j = \emptyset$, $i \neq j$. Each point $\mathbf{x}_i$ is called a *center* and each V_i of X is called a *region* of $\mathbf{x}_i$. Every point p in V_i satisfies the condition

$$||p - \mathbf{x}_i|| < ||p - \mathbf{x}_j||, \quad \text{where} \quad i \neq j.$$

The resulting decomposition of X is called a *Voronoi Tessellation*.

Consider a new point $\tilde{\mathbf{x}} \in X$. If $\tilde{\mathbf{x}}$ lies in the set V_i, then $\tilde{\mathbf{x}}_j$ is nearest to $\mathbf{x}_i$. We call $\mathbf{x}_i$ *the winning center.*

Example: Let $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^2$. The set of all points that are equal distance from the two centers is clearly a straight line. Then the set of points closet to $\mathbf{x}_1$ is given by the half plane

$$V_1 = \left\{ x \in \mathbb{R}^2 \mid ||x - \mathbf{x}_1|| < ||x - \mathbf{x}_2|| \right\}$$

with the other half of the plane being the set of points

$$V_2 = \left\{ x \in \mathbb{R}^2 \mid ||x - \mathbf{x}_2|| < ||x - \mathbf{x}_1|| \right\}$$

that have $\mathbf{x}_2$ as their winning center. The addition of a third center would result in three regions. As more centers are added the defined regions become finite in the interior, thus decomposing the space into polygons, i.e. Voronoi regions.

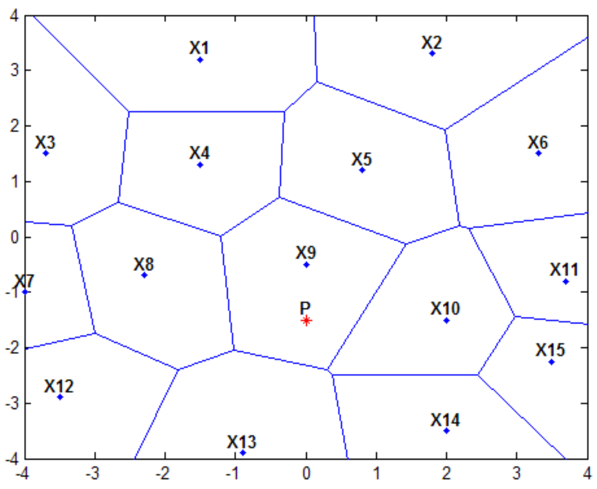

Figure 7: The above Voronoi Tessellation contains 15 regions corresponding to 15 centers. The point P is in $X9$, therefore it is nearest to the center of $X9$ than any other center of the Voronoi Tessellation.

2.7 Linear Learning Machines and The Maximal Margin Classifier

In *Supervised Learning*, a learning machine is given a *training set* (inputs) with associated known labels (output) values, also denoted as *supervised* labels. Customarily, values are in the form of vectors so that the *input space* is a subset of $\mathbb{R}^n$.

Learning/Training means a decision rule can be found that explains the training set well. Clearly, this is easily done since training sets are known labels of the binary classification problem.

Definition. *Let X be the input space and Y be the output domain. The set $X \subset \mathbb{R}^n$, and the binary classification $Y = \{-1, 1\}$. In the case of* m*-class classification $Y = \{1, 2, ..., m\}$. A* **training set** *is a collection of training examples/instances, which are also called training data usually denoted as*

$$S = ((\mathbf{x_1}, y_1), ..., (\mathbf{x}_{\mathscr{L}}, y_{\mathscr{L}})) \subset (X \times Y)^{\mathscr{L}},$$

such that $\mathscr{L}$ is the number of instances, where $\mathbf{x_i}$ are the instances and y_i their associated labels. Again, it is common that if X is a vector space, then the input vectors are column vectors.

Rosenblatt's Perceptron

The first kind of iterative algorithm for learning linear classifier is the **Rosenblatt Perceptron** algorithm. Rosenblatt explored different types of learning machines to include neural networks, or perceptrons. A perceptron consists of connected neurons, where each neuron implements a separating hyperplane, so a perceptron as a whole generates a piecewise linear separating surface. If there exists such a hyperplane that classifies the training data, then the data is said to be **linear separable**.

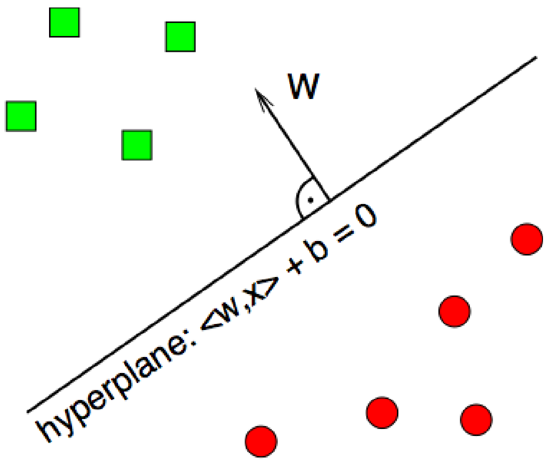

Figure 8: Linearly Separable

Definition 1. *The* **functional** *margin of an instance* $(\mathbf{x}_i, y_i)$ *with respect to a hyperplane* $(\mathbf{w}, b)$ *is denoted*

$$\gamma_i = \gamma_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b)$$

For a correct classification of $(\mathbf{x}_i, y_i)$, $\gamma_i > 0$. From both definitions, if a functional margin is substituted for a *geometric margin*, the result is a normalized linear function $(\frac{1}{||\mathbf{w}||}\mathbf{w}, \frac{1}{||\mathbf{w}||}b)$ measuring Euclidean distances of the points from the decision boundary in the input space. This margin of the training set is now the *maximal margin hyperplane*. The margin is positive for a linearly separable set.

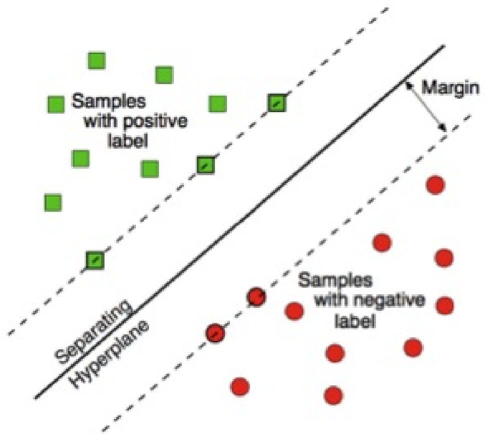

Figure 9: Linearly Separable

Definition. Support Vectors *are the points nearest to the separating hyperplane. They determine the position of the hyperplane, while all other points are independent and do not influence the hyperplane.*

The weighted sum of these support vectors is the normal vector of the hyperplane. For the linear maximal margin hyperplane γ, called H in the figure below is defined as $\langle \mathbf{x}_i, \mathbf{w} \rangle + b \geq +1$, when $y_i = +1$ and $\langle \mathbf{x}_i, \mathbf{w} \rangle + b \leq -1$, when $y_i = -1$. The points that satisfy the equalities are exactly the support vectors that define the maximal margin hyperplane. The maximal distance between the support vector planes H_1 and H_2 is $\frac{2}{||w||}$. In order to maximize the margin, $||w||$ needs to be minimized with the condition that no data lies between H_1 and H_2. That is, again, the combination of $\langle \mathbf{x}_i, \mathbf{w} \rangle + b \geq +1$, when $y_i = +1$ and $\langle \mathbf{x}_i, \mathbf{w} \rangle + b \leq -1$, result in $y_i \langle x_i, w \rangle \geq 1$. Thus, as stated previously, γ needs to be positive. There are a number of optimization methods to maximize $||w||$ to include the Lagrangian method, Quadratic Programming.

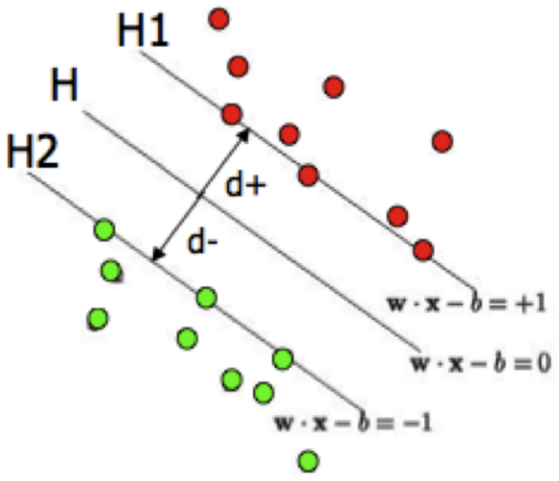
H1
H
H2
d+
d-
$\mathbf{w}\cdot\mathbf{x}-b=+1$
$\mathbf{w}\cdot\mathbf{x}-b=0$
$\mathbf{w}\cdot\mathbf{x}-b=-1$

Figure 10: Support Vectors

By Principle Component Analysis, applied two of the strongest features, the two highest eigenvectors from each observation, from both cats and dogs. The implementation of Principle Component Analysis and Linear Support Vector Machine Train and Classify on MATLAB 'svmtrain' and 'svmclassify' yielded non-linearly separable data. That is, the function γ was not positive. As seen below on the graph, support vectors for training yielded on both sides of the hyperplane.

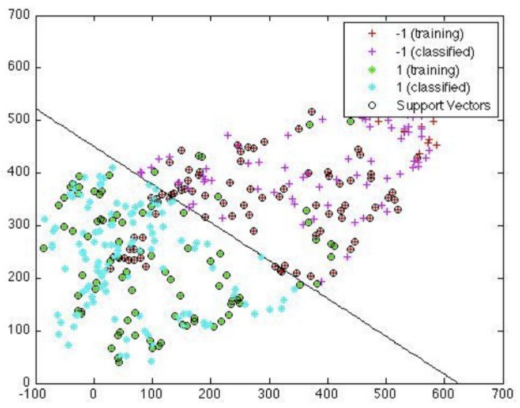

Figure 11: Non-Linearly Separable

2.8 Kernel Induced Feature Spaces

Kernel Induced Feature Spaces

Definition. *Let X be the input space. Then, the suitable representation for the data quantities referred to as features is chosen by a mapping $\varphi : X \rightarrow F$. The space $F = \{\varphi(\mathbf{x}) : \mathbf{x} \in X\}$ is called* **the feature space**.

If classification is easily done on a higher dimensional space, then one needs to construct a maximal hyperplane in this new space. The construction of this hyperplane depends on inner products that will be evaluated in higher dimensions. Computationally, this can become costly, if the dimensions are high. However, there exists a loophole. That is, apply a Kernel that resides in lower dimensions but behaves like an inner product in higher dimensions.

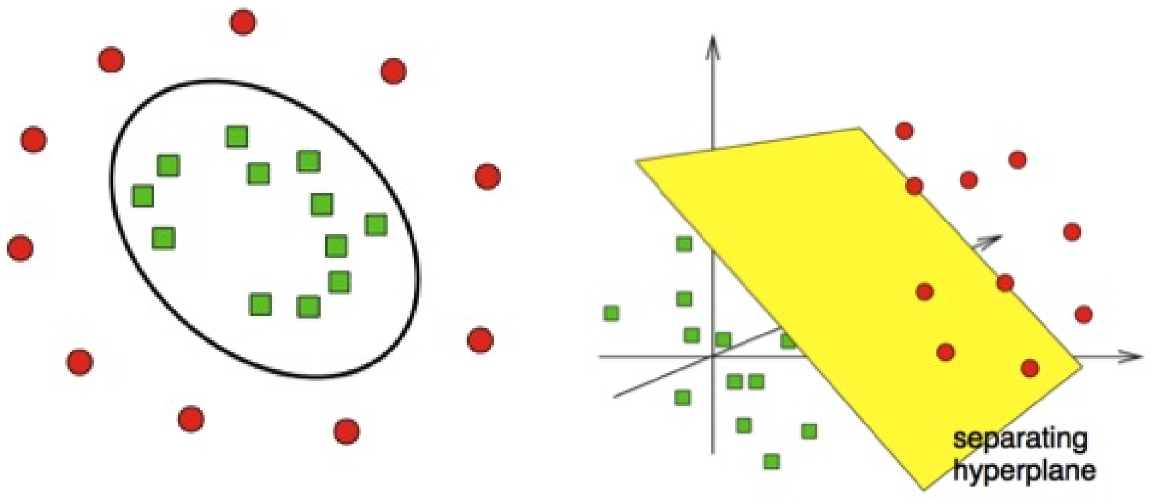

Figure 12: Non-Linearly Separable

Kernel Functions

Definition. *The* **Kernel** *is a function* $\mathcal{K}$ *such that* $\mathcal{K}(\mathbf{x},\mathbf{y}) = \langle\varphi(\mathbf{x},\varphi(\mathbf{y})\rangle$ *for all* $\mathbf{x},\mathbf{y} \in X$. *The function* φ *maps from the input space* X *to the inner product feature space* F.

Given a Kernel function, the decision rule is now

$$f(\mathbf{x}) = \sum_{i=0}^{\mathscr{L}} \alpha_i y_i \mathcal{K}(\mathbf{x}_i, \mathbf{x}) + b$$

for $\mathscr{L}$ iterations of the kernel. Thus, a maximal margin plane is generated by the Kernel Function in the input space.

Kernel Examples

Kernel Examples	
Linear	$\mathcal{K}(\mathbf{x},\mathbf{y}) =< x, y >$
Polynomial	$\mathcal{K}(\mathbf{x},\mathbf{y}) = \gamma(< x, y > +c_\circ)^d$
Radial Basis Function	$\mathcal{K}(\mathbf{x},\mathbf{y}) = \exp\left(-\gamma\,\|\|x-y\|\,\|^2\right)$

3 Implementations

3.1 Using PCA and LDA

The first step of this method is to use the averaging filter followed by the Laplacian filter. The averaging filter will blur our image to make small inconsistencies blend with the background while making the larger images of focus easier to detect. The next step of the filtering process is to use the Laplacian filter for edge detection. After implementation of the filtering process we can see that edges are more apparent. The entire image is placed upon a featureless background so the edges are now more prominent in the image. One can see that an important feature of the cat image is its ears. While an important feature of the dog image is its longer snout. Two images of both a cat and a dog are shown below.

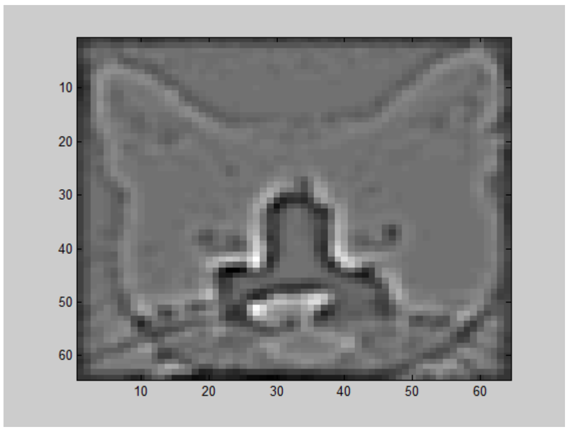
Figure 13: Image of Cat00 after Averaging and Laplacian filter

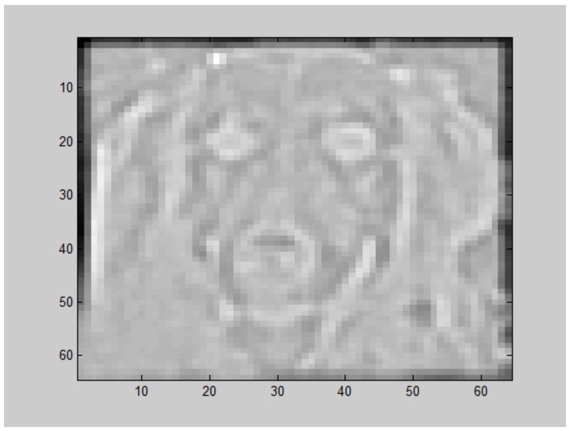
Figure 14: Image of Dog01 after Averaging and Laplacian filter

We want consistencies in the coloring of the image so the results of the Linear Discriminant Analysis are consistent and more precise. A remedy for this is to map the color intensity from 0 to 255. This can be done in MATLAB using *wcodemat*. The results of this process are displayed below.

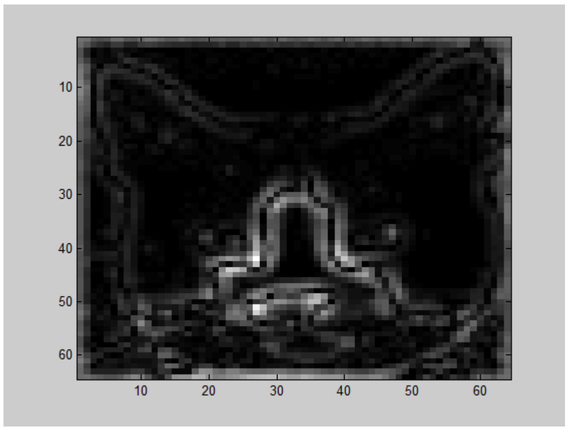
Figure 15: Image of Cat00 after Averaging and Laplacian filter and color re-scale

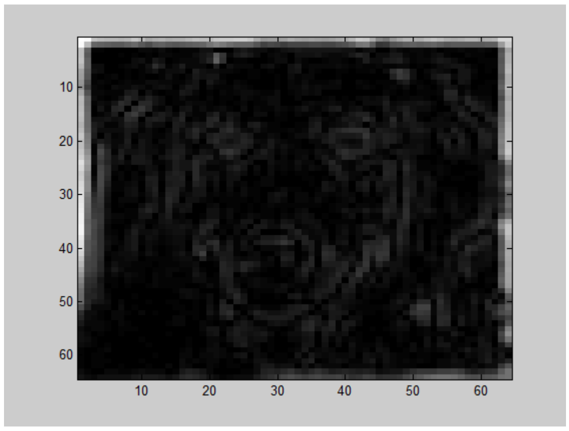
Figure 16: Image of Dog01 after Averaging and Laplacian filter and color re-scale

Before we use Linear Discriminant Analysis we will use Principal Component Analysis. Using Principal Component Analysis is beneficial because it will allow us to find an optimal basis which gives us a lower dimensionality for our data set. This allows for less computation while retaining efficiency. To perform Principal Component Analysis we use Singular Value Decomposition. After Singular Value Decomposition we use the singular values and the concept of cumulative energy to determine a coefficient, D, that will capture the most information from our optimal basis. In this case the cumulative energy was set to 95% to find the best D value. The optimal D value that was found is 121.

Finally, the classification step comes from Linear Discriminant Analysis. Using the method of LDA we find the optimal projection direction, w, to project the data onto the real line.

Once the cats and dogs data has been projected onto the real line we can then find a threshold value and create a plot. The projection of the data is shown below.

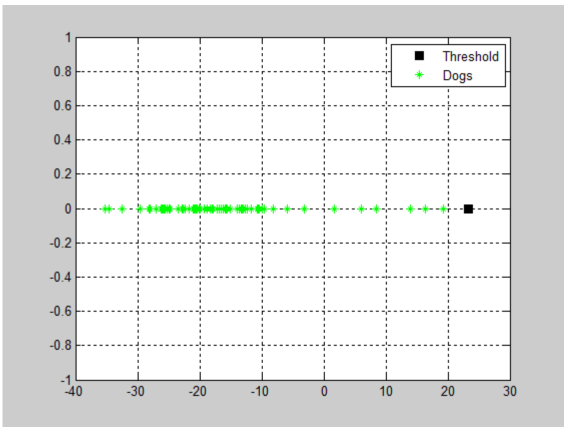

Figure 17: Projection of dogs data onto the real line.

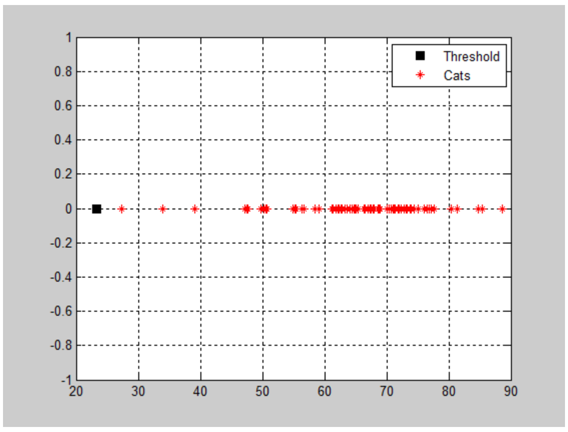

Figure 18: Projection of cats data onto the real line.

A third plot was generated using the probe data. The probe data was also projected onto the optimal direction vector, w. There is one data point that lies on the threshold which makes its particularly hard to classify as either a cat or dog. The results are shown below.

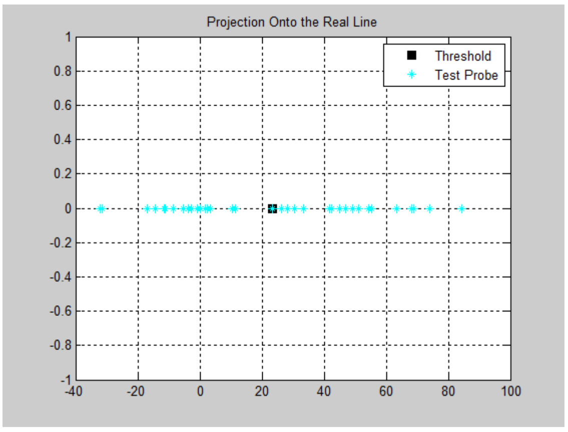

Figure 19: Projection of the testing probe onto the real line.

3.2 Using k-nearest neighbors

Prior to starting the classification process, the images were put through some preprocessing in order to improve results. The images were put through a contrast enhancement filter and a Sobel filter to improve the edges. Figure 20 shows an image before and after filtering. Principle Components Analysis was done on the set of enhanced images in order to reduce the dimensionality of the data. PCA was set to use 120 of the KL-basis vectors, which amounts to about 90% of the energy. This value was chosen after a few trial runs for best performance. After this preprocessing, the adaptive k-nearest neighbor algorithm was used to classify the dimensionality reduced images. In order to provide comparisons in performance the algorithm was run with different k values and different metrics. The metrics used were:

$$\text{The Euclidean metric,} \quad d(x,y) = (\sum_{i=1}^{n} |x_i - y_i|^2)^{\frac{1}{2}}$$

$$\text{The cosine metric,} \quad d(x,y) = 1 - \frac{x^T y}{\|x\| \, \|y\|}$$

$$\text{The correlation metric,} \quad d(X,Y) = 1 - \frac{(x-\bar{x})^T (y-\bar{y})}{\|(x-\bar{x})\| \, \|(y-\bar{y})\|}$$

The different metrics provide a way to interoperate the data geometrically. The Euclidean distance computes the straight line distance between two points. The cosine metric computes distance by comparing the directions two vectors are pointing. That is, two vectors are "close" if the point in a similar direction.

The correlation metric uses the correlation coefficient between two datasets to measure distance between vectors.

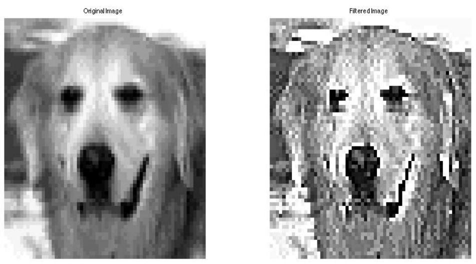

Figure 20: The original image (left) and the filtered image (right)

3.3 Using Haar Wavelet and Voronoi Tessellation

Our main task is to classify images of dogs and cats. Applying a classification method produces good results, but we can improve the classification rate by applying a filter to each image and extract edges to better classify. We will implement one iteration of the Discrete Haar Wavelet Transform on both the training images and testing images. The decomposed sections that contain the extracted edges are the high-low pass and the low-high pass. Then the addition of both the high-low pass and low-high pass sections are the best edge extraction images after the Haar filter.

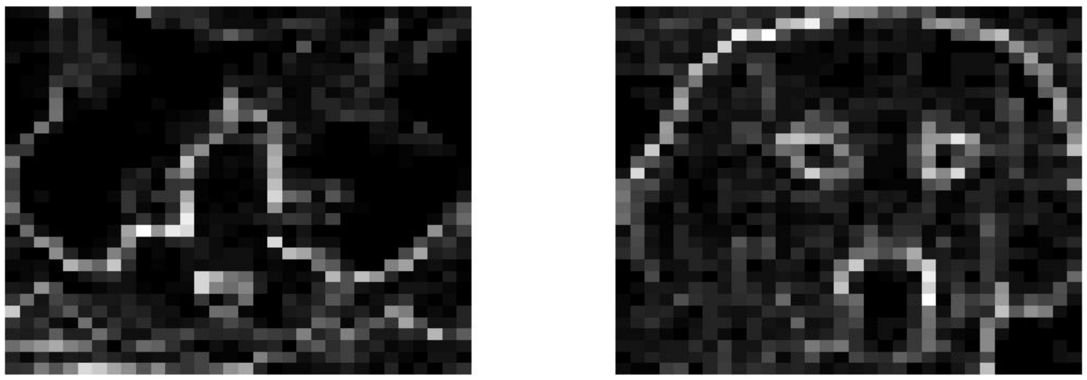

Figure 21: The original images are of size 64×64, but after one iteration of the Discrete Haar Wavelet Transform the decomposed images are of size 34×34. The above images are the edges extracted from two images of the training set.

In order to represent a concatenated image $\mathbf{I}\in\mathbb{R}^{1024}$ in $\mathbb{R}^2$, we must reduce the dimensions significantly while preserving a majority of the information. This is accomplished by the use of *Singular Value Decomposition.*

Theorem: Singular Value Decomposition(SVD). Let A be a real $m\times n$ matrix and $d = \min\{m,n\}$. There exists orthogonal matrices U and V such that

$$A = U\Sigma V^T,$$

where $U\in\mathbb{R}^{n\times n}$, and $\Sigma = \text{diag}(\sigma^{(1)},...,\sigma^{(d)})\in\mathbb{R}^{m\times n}$. The entries of Σ are ordered according to

$$\sigma^{(1)}\geq\sigma^{(2)}\geq\ ...\ \geq\sigma^{(d)}\geq 0$$

and are the singular values of A.

We use the two eigenvectors U_1 and U_2 corresponding to the two largest eigenvalues which are defined as

$$\lambda^{(1)} = (\sigma^{(1)})^2 \quad \text{and} \quad \lambda^{(2)} = (\sigma^{(2)})^2.$$

Let I_i be an image following the edge extraction via the wavelet such that it is a concatenated column vector in $\mathbb{R}^{1024}$. Let $A = \{I_i | i = 1, ..., 160\}$. The matrix A is mean subtracted to yield a matrix B. The SVD of B is performed and the eigenvectors are U_i. Let $\mathbf{U} = [U_1 \; U_2]$. Each I_i is then projected onto these two eigenvectors

$$P_i = \mathbf{U}^T I_i.$$

The column vector P_i is now a point in $\mathbb{R}^2$. Thus, we are able to represent every image as a point and construct a Voronoi Tessellation.

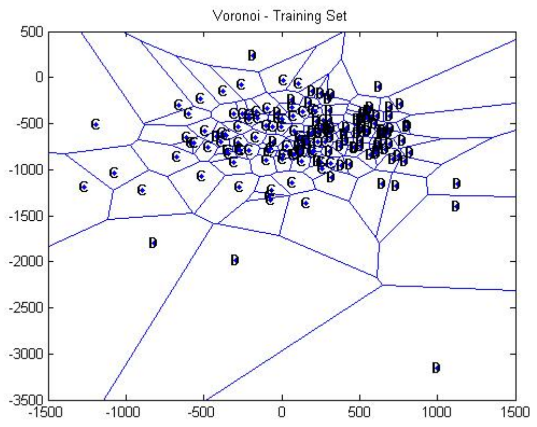

Figure 22: Above is a Voronoi Tessellation constructed by the training set of points. Each cat image is represented as a C and each dog image is represented as a D.

Let I_{test} be an image from the testing set following the edge extraction via the wavelet. The process of dimensionality reduction is performed on I_{test}. Thus, $I_{test} \in \mathbb{R}^2$. The point is introduced to the Voronoi Tessellation and the winning center is located by which region the point is contained in.

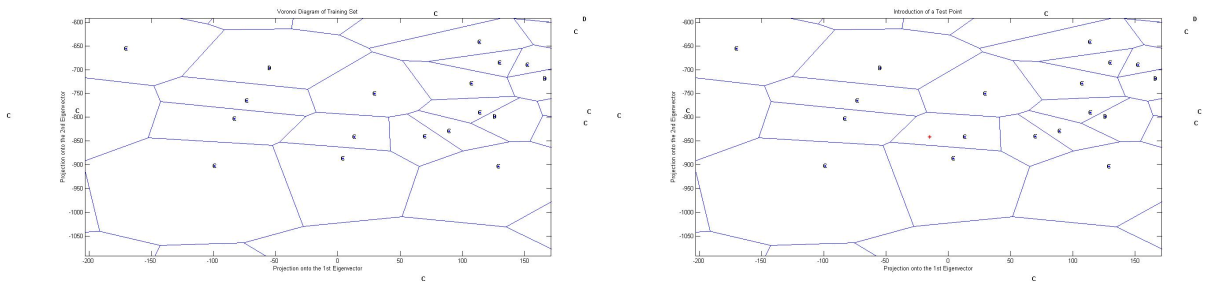

Figure 23: Figure on the left is a zoomed in image of the Voronoi Tessellation of the training set. Figure on the right is identical to the left figure with the introduction of a test point. This test point is nearest the winning center of a cat, and is therefore, classified as a cat.

Every image in the testing set is now reduced to a point in $\mathbb{R}^2$ and classified according to the winning center.

A minor subroutine was added in order for the classification to improve. The subroutine is an adaptive technique which recalculates the Voronoi Tessellation after a testing point is introduced and classified.

3.4 Using Support Vector Machine

Like the linear case, applied Principle Component analysis and using ninety-five percent cumulative energy, extracted the two most prominent features in each observation for the 'svmtrain' and 'svmclassify' in MATLAB. In 'svmtrain', the Kernel 'rbf' with a γ margin of 0.1 and 0.2 and a 'boxconstraint' for the maximum error bound was used, after finding that default resulted in a less accurate hyperplane. The confusion matrix below shows the resulting nonlinearly separable data method Principle Component Analysis Support Vector Machine PCA-SVM. The figure below also shows graphically, the projected space onto polar coordinates of the separation. Again, in the training the ones are the labels of cats and zeros are the signed labels of dogs, as with the testing set. The black circles denote the misclassified 'svmclassify' set, while the red circles denote the properly classified testing set. Support vectors are the smaller circled points near the hyperplane. For future work, one may be able to use a wavelet transform to filter and extract more refined features before PCA and SVM rbf for a better classification rate. Also, one may be able to use the theoretical data to use different optimization methods and adjust parameters for better results in the process of the method itself.

4 Results

4.1 Results of Laplacian and LDA Method

How does the method of Laplacian and LDA measure up? The method works very well. The table of confusion is shown below.

Table of Confusion

	Cat	Dog
Cat	19	0
Dog	1	18

All cats were perfectly classified while only one dog was classified as a cat. The classification rate for this method is 97.37 %. Which dog was misclassified?

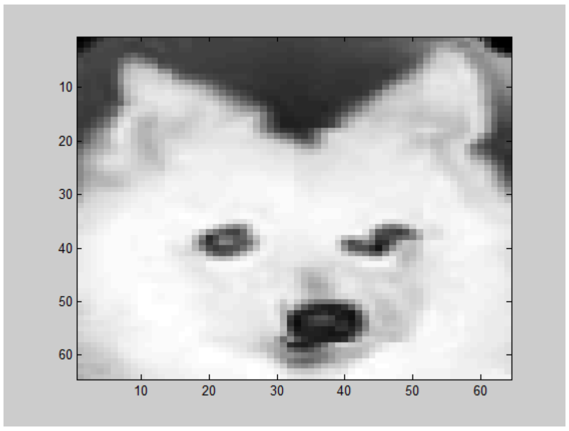

Figure 24: The misclassified canine.

The dog displayed above was the only misclassified canine. Why might this dog may have been misclassified? One of the most prominent features of the cat is its pointy ears. This dog in particular certainly shares that feature with the cats. Another feature that can be seen from the cats are their rotund eyes. This dog also shares that features with the cats. These are of few of the features that may confuse the computer into thinking that this dog is indeed a cat.

4.2 Results of k-nearest neighbor search

Figure 25 shows the results of the original and adaptive k-nearest neighbor search performed on the cat and dog image data. It is clear that as the value of k increased, the misclassification increased. The is most likely due to the similarity of the cat and dog images. Also, the training data may be too sparse for high values of k. If there were say, 1000 points of training data, higher values of k may be more accurate. The best results were obtained for k=1. Using the Euclidean metric, there were 2 dogs misclassified as cats. Using the cosine or correlation metric, there were 3 dogs misclassified as cats. If the adaptive metric had not been used, the Euclidean metric would have misclassified 9 images for k=1.

KNN Table of Confusion

	Cat	Dog
Cat	19	0
Dog	2	17

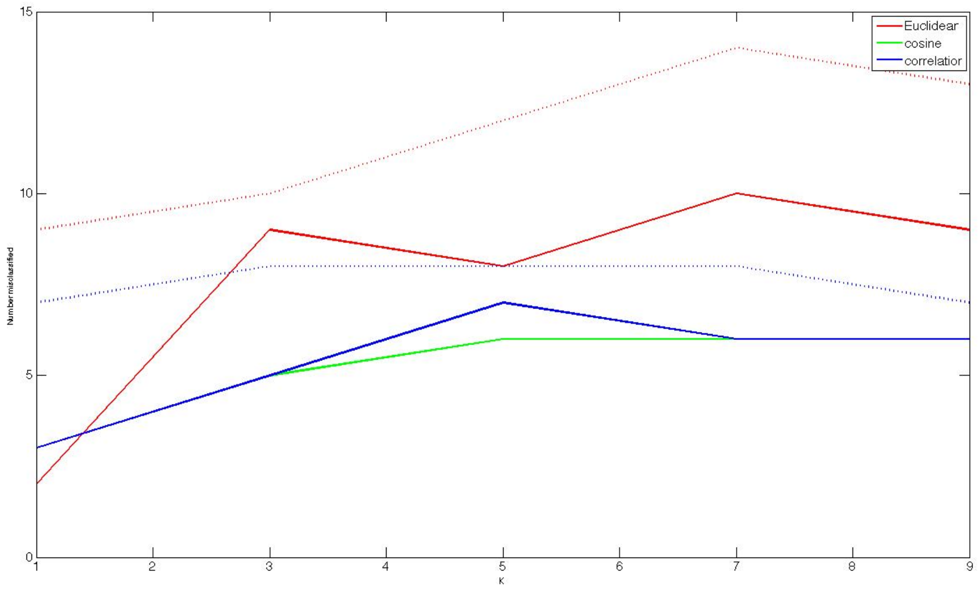

Figure 25: Plot of the number of misclassifications vs. the value of k. The dashed lines represent the original k-nearest neighbor algorithm and the solid lines represent the use of the adaptive metric. The colors signify the different metric used: Euclidean (red), cosine (green), and correlation (blue).

4.3 Results of Haar Wavelet and Voronoi Tesselation

Classification Rate: Given a testing set of 38 images of unknown cats and dogs, the performance of this classification method had a 89.5 percent classification rate.

	DOGS	CATS
Classified	17	17
Misclassified	2	2

4.4 Results of Support Vector Machine Method

Given a testing set of 38 images of unknown cats and dogs, the performance of this classification method had a 86.84 percent classification rate.

	CATS	DOGS
Classified	16	14
Misclassified	3	5

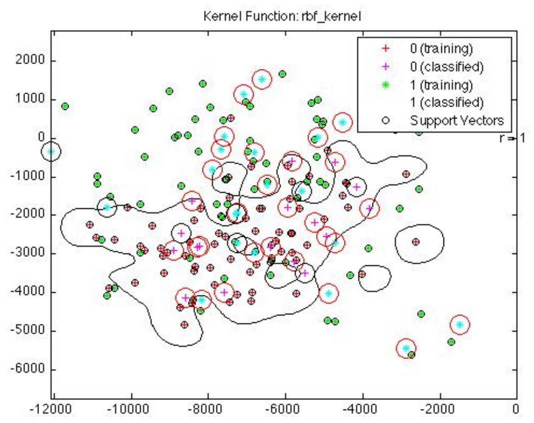

Figure 26: SVM-rbf

5 Summary and Conclusion

In summary, to classify the images of cats and dogs, four different methods were used. The first method used the Laplacian filter and Linear Discriminant Analysis. The second method of classification used k-nearest neighbor search. The third method used the Haar Wavelet and the Voronoi Diagram. Finally, the last method that was used for classification of the cats and dogs images was the SVM (Support Vector Machine) method. In conclusion, the first method using the Laplacian and LDA method which resulted in a 97% classification rate. The second method that used the k-nearest neighbor search resulted in a 94% classification rate. The third method that used the Haar Wavelet and Voronoi Diagram resulted in a 89% classification rate. Lastly, the fourth method of Support Vector Machine resulted in a 86% classification rate. In the case of classifying images of cats and dogs, it seems that the more simplistic methods resulted in a higher classification. Simplistic methods refereing to basic operators and functions. In this case, the derivative operator and Euclidean metric. However, the cats and dogs recognition problem is still an open problem since no algorithm has classified the images perfectly. Since the cats and dogs images have yet to be classified perfectly, then his leaves each of the methods used above open to improvement.

MATLAB

These are the MatLab codes written and commented on for reference.

Laplacian Filter

```
function  [ L ]  = Laplacian3(X)
% A function that performs both the averaging filter and the Laplacian for
% edge detection.
% The Weighted Average Filter
AF = (1/16)*[1 2 1; 2 4 2; 1 2 1];

% The Laplacian Filter
Laplacian = [1 1 1; 1 -8 1; 1 1 1];
AF2 = conv2(X,AF,'same');
M = conv2(AF2,Laplacian,'same');

color = size(colormap(gray),1);
L = wcodemat(M,color);

L1 = L+X;


% This computes the averages out our image and then computes the laplacian
% of our image and also computes the wavelet of the sharpened image.

end
```

Categorize Algorithm (Laplacian and LDA)

```
function [C,R] = categorize( cats, dogs, probe, energy )

%Classify cats and dogs

nc = size(cats,2);
nd = size(dogs,2);
np = size(probe,2);

%Reshape the images
for k=1:nc
    cat(:,:,k)=reshape(cats(:,k),64,64);
end
for k=1:nd
    dog(:,:,k)=reshape(dogs(:,k),64,64);
end
for k=1:np
    probe1(:,:,k)=reshape(probe(:,k),64,64);
end

%Apply the Filters and Wavelet to extract the images
for k=1:nc
    L=Laplacian3(cat(:,:,k));
    cats2(:,k)=reshape(L,size(L,1)*size(L,2),1);
end
for k=1:nd
    L=Laplacian3(dog(:,:,k));
    dogs2(:,k)=reshape(L,size(L,1)*size(L,2),1);
end
for k=1:np
    L=Laplacian3(probe1(:,:,k));
    probes2(:,k)=reshape(L,size(L,1)*size(L,2),1);
end
cats2 = real(cats2);
dogs2 = real(dogs2);
probes2 = real(probes2);

% We now have a new matrix consisting of images which have been transformed
% by the Laplacian and Wavelet
N = [cats2, dogs2];
N = N-repmat(mean(N,2),[1,nc+nd]);
[U,S,V] = svd(N,0);

% Find the cumulative energy to find the optimal D for a best basis
D=diag(S).^2; %Squares each diagonal entry of the singular values array
cum_energy = 100.*cumsum(D)./sum(D); % computes the cumulative energy of an
    image.
%cumsum is the sum of the K preceeding values while sum give the sum of all
%values
I=find(cum_energy > energy); %searches and returns the indcies of values
    greater than
% 95%
R = I(1) ;

% Principal Component Analysis
UR = U(:,1:R);
```

```
qcats = UR'*cats2;
qdogs = UR'*dogs2;
qprobe = UR'*probes2;

% Fisher Discriminant Analysis — Compute the optimal projection direction, w
A = S(1:R,1:R)*V(:,1:R)';
ncat = A(:,1:nc);
ndog = A(:,nc+1:nc+nd);

mean1 = mean(ncat,2);
mean2 = mean(ndog,2);

Sb = (mean2-mean1)*(mean2-mean1)';
M1 = repmat(mean1,[1,nc]);
M2 = repmat(mean2,[1,nd]);

SW1 = (A(:,1:nc)-M1)*(A(:,1:nc)-M1)';
SW2 = (A(:,nc+1:nc+nd)-M2)*(A(:,nc+1:nc+nd))';
Sw = SW1+SW2;

SwI = pinv(Sw);
S=SwI*Sb;
[Unew,Sigma,V] = svd(S,0);
w = Unew(:,1);

%Project Data onto the Real Line
rcat = w'*qcats;
rdog = w'*qdogs;
rprobe = w'*qprobe;

if mean(rdog) > mean(rcat)
    w=-w;
    rdog = -rdog;
    rcat = -rcat;
    rprobe = -rprobe;
end

sortDog = sort(rdog);
sortCat = sort(rcat);

l = length(sortDog);
l2 = 1;
while sortDog(l) > sortCat(l2)
    l = l-1;
    l2 = l2+1;
end
threshold = (sortDog(l)+sortCat(l2))/2;
C = zeros(1,length(rprobe));
for k = 1:length(rprobe)
    if rprobe(k) > threshold
        C(k) = 1;
    end
end

clf
figure(1),
plot(threshold,0,'sk','markerfacecolor','k')
hold on
plot(rcat,0,'*r'); grid on
legend('Threshold', 'Cats')

figure(2)
plot(threshold,0,'sk','markerfacecolor','k')
```

```
hold on
h1 = plot(rdog,0,'*g'); grid on
legend('Threshold', 'Dogs')

figure(3)
plot(threshold,0,'sk','markerfacecolor','k')
hold on
plot(rprobe,0,'*c'), grid on
title('Projection Onto the Real Line'),
legend('Threshold', 'Test Probe')

cats = cats - repmat(mean(cats,2),1,80);
[U1,S1,V1] = svd(cats,0);


for k = 1:5
    figure(k+3),imagesc(reshape(U1(:,k),64,64)), colormap gray
end


end
```

Image enhancement

```

function [ ImgsSharpened ] = SharpenImages( Imgs )

    [ Rows Number ] = size(Imgs);

    ImgsSharpened = zeros(4096, Number);

    H = fspecial('unsharp');

    for k=1:Number

        D_Img = reshape(Imgs(:,k), 64, 64);

        D_Sharp = imfilter(D_Img, H, 'replicate');

        D_Edge = edge(D_Sharp,'sobel');
        D_Img = D_Sharp + D_Edge;

        D_Adj = imadjust(D_Img);

        ImgsSharpened(:,k) = D_Adj(:);
    end

end
```

Construct the vector r_i for the adaptive metric

```

function [ R ] = BuildR ( Input, Classes, dist )

    [ Rows Total ] = size(Input);

    Data1 = Input(:, Classes == 0);
    [ Rows Num1 ] = size(Data1);
    R1 = zeros(1, Num1);
    Data2 = Input(:, Classes == 1);
    [ Rows Num2 ] = size(Data2);
    R2 = zeros(1, Num2);

    Distances = zeros(1, Num1);
    for j=1:Num1

        Distances = pdist2(Data2', Data1(:, j)', dist);

        R1(j) = min(Distances);
    end


    Distances = zeros(1, Num2);
    for j=1:Num2

        Distances = pdist2(Data1', Data2(:, j)', dist);

        R2(j) = min(Distances);
    end

    R = [ R1 R2 ];

    R(R==0) = 0.0000001;

end
```

Adaptive k-nearest neighbors

```
function [ Classified MinDistances ] = AdaptiveKNN( Input, Unknowns, Classes, K
    , dist )

    [ Rows InputTotal ] = size(Input);


    DogIndex = Classes==0;
    CatIndex = Classes==1;

    Dogs = Input(:, DogIndex);
    Cats = Input(:, CatIndex);


    R = BuildR(Input, Classes, dist);

    [ Rows UnknownTotal ] = size(Unknowns);

    MinIndexes = zeros(UnknownTotal, K-1);
    MinDistances = zeros(UnknownTotal, K-1);

    Classified = zeros(UnknownTotal, 1);
    Distances = zeros(1, InputTotal);

    for j=1:UnknownTotal

        Distances = pdist2(Input', Unknowns(:, j)', dist);
        Distances = (1./R) .* Distances';

        for i=1:K

            M = min(Distances);
            indx = find(Distances==M, 1);

            MinDistances(j, i) = M;
            MinIndexes(j, i) = indx;

            Distances(indx) = [];
        end

        Res = Classes(MinIndexes(j,:));

        Classified(j) = mode( Res );
    end

end
```

Code to run Voronoi Classification

```
% This is the main script file which yields the array of correctly
% classified dogs and cats.
%
% For the testing set of 38 dogs and cats, the output should be 1 in the
% first 19 values and 0 in the remaining 19 values.

clear
[CATS,DOGS] = LoadProjectFileTif();
[group] = Perform2DWavelet([CATS,DOGS])
[group] = LapFilter(group);
[group] = ZeroOutNeg(group);

[CATS,DOGS] = LoadTestTIFF();
[wavtest] = Perform2DWavelet([CATS,DOGS])
[wavtest] = LapFilter(wavtest);
[wavtest] = ZeroOutNeg(wavtest);

[class,U,S,V] = CatDogClassifier2(wavtest,group);
```

Haar Wavelet

```
function [ wavCATDOG ] = Perform2DWavelet(X)
% performs 2D Haar wavelet on one or a set of images
% output is a matrix of concatenated edge extracted images

M = size(X,2);
nbcol = size(colormap(gray),1);

for i = 1:M
    A = reshape(X(:,i),64,64);
    [UL,UR,LL,LR] = dwt2(A,'Haar','zpd','sym');
    UR = wcodemat(UR,nbcol);
    LL = wcodemat(LL,nbcol);
    B = UR + LL;
    wavCATDOG(:,i) = reshape(B,32*32,1);
end
end
```

Laplacian Filter (Voronoi Diagram)

```
function [LapImages] = LapFilter(A)
% performs the Laplacian filter on a set of images

Lap_Mask = [-1 -1 -1; -1 8 -1; -1 -1 -1];
c = 1; % intensity
[N,M] = size(A); dim = sqrt(N);
for i = 1:M
    Img = reshape(A(:,i),dim,dim);
    % apply Laplacian mask on the blurred image
    Lap_img = conv2(Img,Lap_Mask,'same');
    LapImage = Img + c*Lap_img;      % sharpen image
    LapImages(:,i) = reshape(LapImage,dim*dim,1);
end

end
```

Classification (Voronoi Diagram)

```
function [class,U,S,V] = CatDogClassifier2(testSet,group)
% classify images using the Voronoi Tessellation
% testSet = testing images; group = training set
% outputs are:
% class = array of ones and zeros
% U = eigenvectors from mean subtracted data
% S = singular values; V = Voronoi data

col = size(group,2); Mm = size(testSet,2); avg = mean(group,2);
C_tilde = group - repmat(avg,1,col);
[U,S,V] = svd(C_tilde); projection = U(:,1:2)'*group;

% plot the initial Voronoi data
Vdata = [projection'];
figure(4), voronoi(Vdata(:,1),Vdata(:,2)),
M = size(Vdata,1);
plabelC = arrayfun(@(n) {sprintf('C', n)}, (1:M/2)');
plabelD = arrayfun(@(n) {sprintf('D', n)}, (M/2+1:M)');
hold on
Hpl1 = text(Vdata(1:M/2,1), Vdata(1:M/2,2), plabelC, 'FontWeight', ...
    'bold', 'HorizontalAlignment','center', ...
    'BackgroundColor', 'none');
Hpl2 = text(Vdata(M/2+1:M,1), Vdata(M/2+1:M,2), plabelD, 'FontWeight', ...
    'bold', 'HorizontalAlignment','center', ...
    'BackgroundColor', 'none');
hold off

figure(5), voronoi(Vdata(:,1),Vdata(:,2)),
M = size(Vdata,1);
plabelC = arrayfun(@(n) {sprintf('C', n)}, (1:M/2)');
plabelD = arrayfun(@(n) {sprintf('D', n)}, (M/2+1:M)');
hold on
Hpl1 = text(Vdata(1:M/2,1), Vdata(1:M/2,2), plabelC, 'FontWeight', ...
    'bold', 'HorizontalAlignment','center', ...
    'BackgroundColor', 'none');
Hpl2 = text(Vdata(M/2+1:M,1), Vdata(M/2+1:M,2), plabelD, 'FontWeight', ...
    'bold', 'HorizontalAlignment','center', ...
    'BackgroundColor', 'none');

cnt = 0; index = 0;
for i = 1:Mm
    projection2 = U(:,1:2)'*testSet(:,i);
    [v,c] = voronoin(projection');
    dt = DelaunayTri(projection');
    [Vv Rr] = dt.voronoiDiagram();
    % begin classification of testing set
    tid = dt.nearestNeighbor(projection2(1),projection2(2));
    plot(projection2(1),projection2(2), 'r*');
    if tid < col/2 + 1 + index
        class(i) = 1; index = index + 1;
        projection = [projection2 projection];
    else
        class(i) = 0;
        projection = [projection projection2];
    end
end
hold off

% plot the updated Voronoi data
Vdata = [projection'];
figure(6), voronoi(Vdata(:,1),Vdata(:,2)),
M = size(Vdata,1);
plabelC = arrayfun(@(n) {sprintf('C', n)}, (1:M/2)');
plabelD = arrayfun(@(n) {sprintf('D', n)}, (M/2+1:M)');
hold on
Hpl1 = text(Vdata(1:M/2,1), Vdata(1:M/2,2), plabelC, 'FontWeight', ...
    'bold', 'HorizontalAlignment','center', ...
    'BackgroundColor', 'none');
Hpl2 = text(Vdata(M/2+1:M,1), Vdata(M/2+1:M,2), plabelD, 'FontWeight', ...
    'bold', 'HorizontalAlignment','center', ...
    'BackgroundColor', 'none');
hold off
end
```

PCA (SVM)

```
function [U,A,D] = PCA(X, percent)
%-------------------------------------------------------------------------
% Principal Component Analysis
% Purpose:
% To reduce the dimension of a collection of X high-resolution image files
% to the best D-element subspace of X matrix

% INPUT:
% X - a collection of high-resolution image files
%     (after concatinating each image file into a column vector)
% percent - % of cumulative energy of X to retain
%
% OUTPUT:
% U - basis in the best D-element subspace
% A - coefficient matrix
% D - the best D value
%-------------------------------------------------------------------------

X_Ensemble_Avg = mean(X,2);
% mean-subtracted to remove insignificant data
X_tilde = X - repmat(X_Ensemble_Avg,1,size(X,2));

% Finds the best D-value for X
[U,Sigma,V] = svd(X_tilde,0);
lambda = dot(Sigma,Sigma);

% Cumulative Energy
cumulative_energy = 100.*(cumsum(lambda)./ sum(lambda));
```

```
I = find(cumulative_energy > percent);
D = I(1);
% U - basis in D-element subspace, A - coefficient matrix
U = U(:,1:D); A = Sigma(1:D,1:D)*V(:,1:D)';
```

PCA with Support Vector Machine

```
% PCA and Linear SVM
% April 30, 2012 update

clear all; clc;
load CATS; load DOGS;

m = 4096; n = 80; p = 70;
Xi = CATS(1:m, 1:p); Yi = DOGS(1:m, 1:p);
Group = [Xi, Yi];

% PCA, take transpose since svm takes rows as observations
% [U, Y, latent] = princomp(Group');
[U, Y, D] = PCA(Group, 98);

% reduced subspace and Traning matrix for svmtrain
new_CATS = U'*Xi; new_DOGS = U'*Yi;
Training = [new_CATS'; new_DOGS'];

% Group matrix for svmtrain
CATS_train = ones(size(new_CATS));
DOGS_train = -ones(size(new_DOGS));
Train_Classes = [CATS_train'; DOGS_train'];

% Use a linear support vector machine classifier
svmStruct = svmtrain(Training(:,1:2), Train_Classes(:,1),'showplot',true);

% Add a title to the plot
title(sprintf('Kernel Function: %s',...
              func2str(svmStruct.KernelFunction)),...
              'interpreter','none');

% Classify the test set using svmclassify
% test sample one cat and try another vector of different cat
% and see the efficiency
% remember needs to match as svmtrain
classes = svmclassify(svmStruct, new_CATS(:,1:2),'showplot',true);

% See how well the classifier performed
% cp = classperf(Train_Classes(:,1));
% classperf(cp,classes,test);
% cp.CorrectRate
```

SVM with Radial Basis Function

```
% Principle Comp Analysis and Support Vector Machine
% Radial_Basis_Function as Kernel
% May 9, 2012 last updated



close all; clear all; clc;
load CATS; load DOGS;
load TestCats; load TestDogs;

% [ TestCats TestDogs ] = LoadTestTIFF();

m = 4096; n = 80; p = 70; c = 19;
Xi = CATS(1:m, 1:n); Yi = DOGS(1:m, 1:n);
Xt = TestCats; Yt = TestDogs;
Train_Group = [Xi, Yi];
Train_Group2 = [Xt, Yt];
TG = [Train_Group, Train_Group2];
%%
% PCA, take transpose since svm takes rows as observations
[U, Y, D] = PCA(TG, 98);
% [V, Z, B] = PCA(Train_Group2, 98);
% projected subspace and Traning and Testing matrices for svmtrain and
% svmclassify
new_CATS = U'*Xi(1:m,1:p); new_DOGS = U'*Yi(1:m,1:p);

test_CATS = U'*Xt(1:m,1:c); test_DOGS = U'*Yt(1:m,1:c);

Training = [new_CATS'; new_DOGS'];

Testing = [test_CATS'; test_DOGS'];

% Group Matrix of zeros and ones for svmtrain and check
CATS_train = ones(size(new_CATS));
DOGS_train = zeros(size(new_DOGS));

Train_Classes = [CATS_train'; DOGS_train'];

CATS_test = ones(size(test_CATS));
DOGS_test = zeros(size(test_DOGS));

Test_Classes = [CATS_test'; DOGS_test'];

%%
% Use a Gaussian Radial Basis Function vector machine classifier
svmStruct = svmtrain(Training(:,1:2), Train_Classes(:,1), ...
    'Kernel_Function','rbf','RBF_Sigma', 0.2,...
    'BoxConstraint', 0.8,'showplot',true);
hold on
axis equal
ezpolar(@(x)1)
title(sprintf('Kernel Function: %s',...
             func2str(svmStruct.KernelFunction)),...
             'interpreter','none');
hold off



%%
% svmclassifier

[row_Test, col_Test] = size(Testing);

hold on;
for i = 1:row_Test
    v(i,:) = svmclassify(svmStruct, Testing(i,1:2),'showplot',true);
end
hold off;
v

hold on;
```

```
    mydiff = (v == Test_Classes(:,1)); % classified correctly
    for j = mydiff
        plot(Testing(j,1), Testing(j,2),'ro','MarkerSize',15)
    end
    for j = not(mydiff) % check plot black circles misclass
        plot(Testing(j,1), Testing(j,2),'ko','MarkerSize',15)
    end
hold off;
```

```
function [B] = ZeroOutNeg(A)
% zero out any negative values in a matrix

[N,M] = size(A);

for i = 1:N
    for j = 1:M
        if A(i,j) < 0
            B(i,j) = 0;
        else
            B(i,j) = A(i,j);
    end
end
end
```

References

[1] Dr. Jen-Mei Chang. *Matrix Methods for Geometric Data Aanalysis and Pattern Recognition.* 2011.

[2] Michael Kirby. *Geometric Data Analysis*, pages 220–240. 2001.

[3] Neskovic P. Cooper L. Wang, J. Improving Nearest Neighbor Rule with a Simple Adaptive Distance Measure. *Pattern Recognition Letters*, 28:207–213, 2007.