TWO-CLASS CLASSIFICATION: AN APPLICATION

MELISSA DOMINGUEZ, KELLY GIANG, & ERICH BROWNLOW

Abstract. This report summarizes findings on solving the 2-class classification problem. Given a set of dog and cat images, together known as the "training" set, come up with classification algorithms that accurately classify unknown images as either "cat" or "dog". This report discusses the methods used, the algorithms implemented, and the results of different evaluations of the classifiers.

1. Introduction

Training a computer to recognize and correctly classify images is not an easy task to accomplish. This paper discusses three different approaches to solving this problem. To accomplish this task, we create a problem and propose different solutions to it.

Given images of dogs and cats, develop a classifier that can accurately classify new unknown images as either "cats" or "dogs". Eighty images of cats and eighty images of dogs are given (called a training set) where we can tell the computer that the cats are cats and that the dogs are dogs. We then develop methodology for creating classifiers that can (hopefully) accurately classify new unknown images (called a test set).

In the Methods section, we discuss two kinds of computer algorithms: methods to classify unknown images, and methods used by the classifiers to improve the correct-classification rate.

In the Implementation section we discuss how the methods are used to create classifying algorithms. We discuss in detail how each classifying algorithm works, and briefly discuss its development and intuition.

In the Results section, we report on the performance of the classifiers using two methods for evaluation: a leave-one-out-classify cross-validation on the training set and the classification of the test set that was kept out of the training set during the development of the classifiers.

Finally, in the Conclusion, we discuss our impressions of the classifiers,

Date: May 20, 2010.

2. Methods

In this section, we discuss two kinds of methods: methods used for classifying unknown images, and methods used by the classifiers to improve correct-classification.

Given a set of cat-images, a set of dog-images, and a unknown probe image, three methods were developed for classifying the probe image: Mean Vector Norm Difference (MVND), Fisher's Discriminant Analysis (FDA), and Principal Angles (PA).

Fisher's Discriminant Analysis for Two Classes (FDA) was used as a classifier for the unknown cat-dog images. FDA considers two classes: Cats and Dogs, of data points in $\mathbb{R}^N$. Once FDA is applied to the data in the n space, it will generate a line that separates and classifies the cats and dogs.

The Mean Vector Norm Difference (MVND) method computes the "mean" dog-vector, subtracts the probe vector from it, and computes the norm of the result. The same is done with the "mean" cat-vector. The probe vector is classified as which ever norm-difference is smaller, i.e., if the norm difference of the probe vector and the mean cat-vector is smaller than the norm difference of the probe vector and the mean dog-vector, then the probe vector is classified as a cat. The idea is that each image represents a point in n-space, and MVND calculates the "mean cat image" and the "mean dog image" and classifies the probe as whichever class the probe vector is closest to.

The Principal Angles (PA) method computes the principal angle between the probe and the cat-images and computes the principal angle between the probe and the dog-images. Whichever principal angle is smaller, the probe-image is classifed as that kind of image, i.e., if the principal angle between the probe and the cat-images is smaller than the principal angle between the probe and the dog-images, then the probe is classifed as a cat-image. The idea is that principal angles provide information about the relative position of two subspaces in Euclidean space. If the principal angles between two subspaces is small, then they are "close together".

The following methods were used in the implementation to improve the correct-classification rate of the classifying methods. They are: 2D Convolution, Principal Components Analysis (PCA), and Wavelet Decomposition.

Two-dimensional Convolution is used in two ways. An image (training or probe) can be convoluted with either an averaging filter, which will smooth the details out of the image, or can be convoluted with a 2D-Laplacian filter, which will extract the details of the image. The resulting images may help a classifier by either extracting the details that make cat images cat images, or possibly smoothing out images to make the dog-images all look alike, and screen out minor details that appear in each image. See Figure 1.

Principal Component Analysis is used to reduce the dimensionality of a set of vectors, while keeping the maximum amount of information (or variance) contained in them. When reducing the dimensionality of an image, we hope that the difference between cat images gets reduced, while the difference between cat and dog

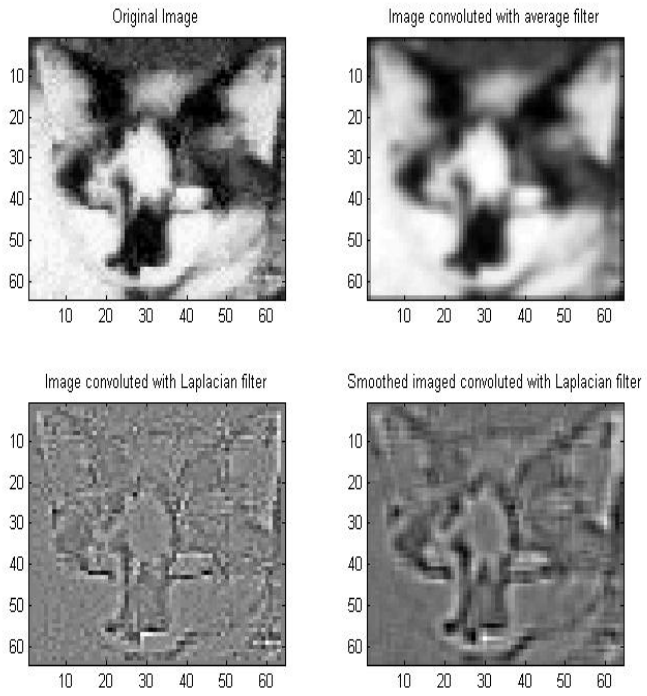

FIGURE 1. Example of 2D-Convolutions of an image

images is retained.

Wavelet Decomposition takes an image and decomposes it into the approximation (cA), horizontal (cH), vertical(cV), and diagonal (cD) details.

3. Implementation

In Melissa's classifier, FDA for Two Classes is used to classify the unknown cat-dog probe images. Wavelet decomposition and principal component analysis were used to improve the efficiency of the algorithm. The classification algorithm begins by extracting the details from the wavelet decomposition,and then decomposing the the cat-dog data with the use of SVD. After SVD is carried out, the projection matrix is obtrained, we use the rank of U^T which contains the features, to determine the number of principal components to use. Afterward, the FDA is carried out and gives us our decision rule for the classification of cats and dogs. In Melissa's algorithm, she used 20 principal components. Since feature=20, was specified in the algorithm, it will suggest that 18.219 is the decision threshold value that will separate the cat and dogs in the following manner: dogs<threshold<cats. Once the LDA projection is implemented and stored as the variable pval the algorithm uses the following rule (pval>threshold) to create a row vector called probe labels,

that gives us our classification of the cat and dogs.

In Kelly's classifier, MVND is used to classify an unknown vector. Kelly tried to improve the accuracy of the classifier using 2D-Convolutions and PCA, but these did not improve the accuracy. Thus the classifier simply uses the MVND method described above.

In Erich's classifier, PAs are used to classify an unknown vector and 2D-Convolutions are used to help improve its accuracy. The classifier starts by applying an average filter to all images (training and probe), applying a 2D-Laplacian (feature-extraction) filter to all images (training and probe), and applying a averaging filter to the detail images of the 2D-Laplacian. For each of the 3 sets of images (averaged, features, averaged-features), use Principal Angles to classify each of the probe vectors. Each probe vector has been classifed 3 ways, each of the ways votes, and majority rules, i.e., if two methods vote "cat" and one votes "dog", then the probe vector is classified as a cat.

4. Results

To evaluate the algorithms, two procedures were conducted. First, a test set of 38 images was separated from the beginning, and kept separate during the development of the classifying algorithms. Once finished, the classifiers classified these images, and the correct classification rate can be computed. Second, a Leave-one-out-classify cross-validation (LOOC) procedure on the training set was conducted. In this procedure, each of the training images is separated from the rest, and the classifier uses the rest of the training images to classify the separated image.

For Melissa's Wavelet FDA classifier, LOOC on the original training data (80 cats & 80 dogs), the confusion matrix is:

Truth \ Classified as:	Cat	Dog
Cat	74	10
Dog	6	70

Which gives a correct classification rate of 90%.

For the test data set, (38 "unknown" images), the confusion matrix is:

Truth \ Classified as:	Cat	Dog
Cat	19	2
Dog	19	17

Which gives a correct classification rate of 94.74%.

For Kelly's MVND classifier, LOOC on the original training data, the confusion matrix is:

Truth \ Classified as:	Cat	Dog
Cat	57	16
Dog	23	64

Which gives a correct classification rate of 75.63%.

For the test data set the confusion matrix is:

Truth \ Classified as:	Cat	Dog
Cat	12	7
Dog	7	12

Which gives a correct classification rate of 63.16%.

The MVND classifier is not very effective when classifying brightr and light probe images. See Figure 2 and Figure 3. Kelly attempted to use 2D-convolution filters to improve the accuracy, but those methods did not change the accuracy.

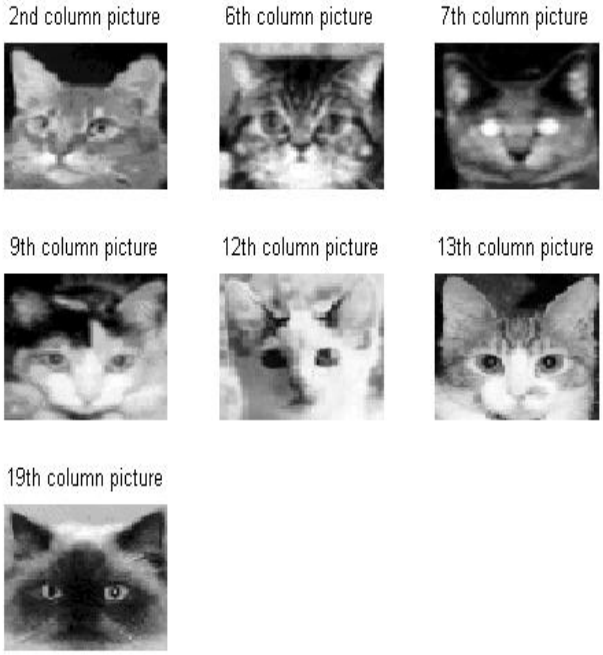

FIGURE 2. MVND Misclassified Images in Test Set

For Erich's PA classifier, LOOC on the original training data, the confusion matrix is:

Truth \ Classified as:	Cat	Dog
Cat	79	1
Dog	7	73

Which gives a correct classification rate of 95%. See Figure 4.

For the test data set (38 "unknown" images) the confusion matrix is:

Truth \ Classified as:	Cat	Dog
Cat	18	1
Dog	1	18

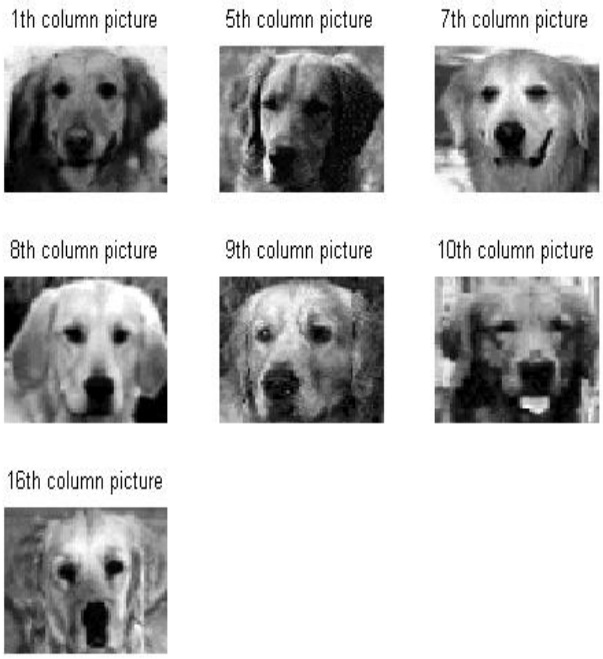

FIGURE 3. MVND Misclassified Images in Test Set

Which gives a correct classification rate of 94.74%. See Figure 5 and Figure 6.

5. Conclusions

While Kelly's MVND classifier did not have the highest correct classification rate, it was the fastest method. Erich's PA classifier was the most correct but was very slow, and is not appropriate when fast answers are needed. Melissa's Wavelet FDA classifier had a high correct-classification rate, and was fast, and has the best balance of accuracy and speed.

6. Appendix A MATLAB codes

Melissa's algorithm uses 3 functions: [Dominguez_classifier.m], [dc_trainer.m], and [dc_wavelet.m]

```
% Melissa Dominguez
% Math 695

% Classifier call
% for Final Project
%
```

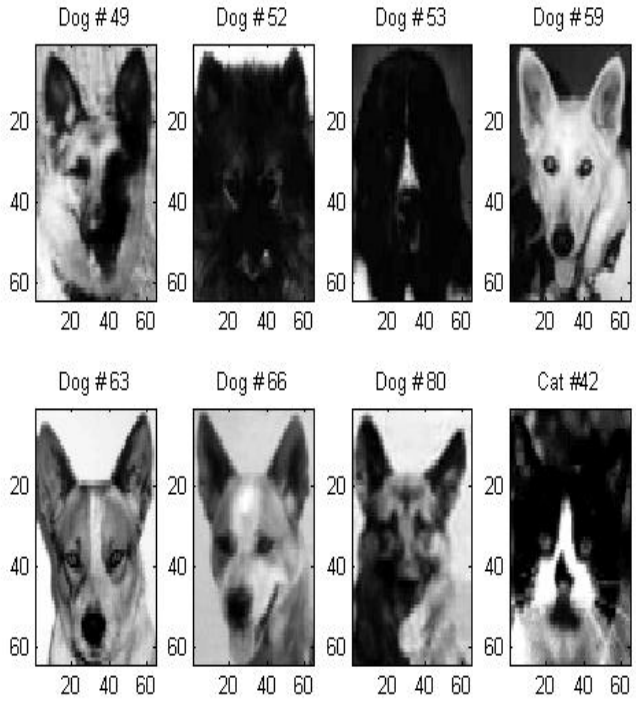

FIGURE 4. PA Misclassified images in LOOC

```
load cats_dogs
function()=Dominguez_classifier(TestSet)
dog_wave = dc_wavelet(dogs);
cat_wave = dc_wavelet(cats);
feature=20; % 1<feature<80
[result,w,U,S,V,threshold] = dc_trainer(dog_wave,cat_wave,feature);
Test_wave = dc_wavelet(TestSet); % wavelet transformation
TestMat = U'*Test_wave; % SVD projection
pval = w'*TestMat; % LDA projection

%cat = 1, dog = 0
probe_labels = (pval>threshold)%Decision Rule


%Melissa Dominguez
% Final Project
% Math 695
%Dc trainer file

function [result,w,U,S,V,threshold]=dc_trainer(dog0,cat0,feature)
nd=length(dog0(1,:)); nc=length(cat0(1,:));
```

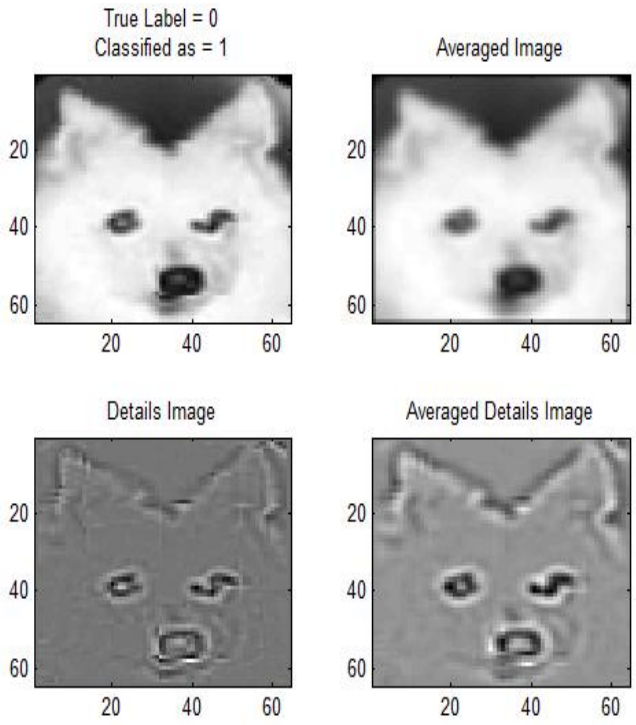

FIGURE 5. PA Misclassified Test Set Dog

```
[U,S,V] = svd([dog0,cat0],0); % reduced SVD
animals = S*V';
U = U(:,1:feature);
dogs1 = animals(1:feature,1:nd);
cats1 = animals(1:feature,nd+1:nd+nc);

for j=1:9
colormap(gray)
title(['Principal Component( ',int2str(j-1),' )'])
subplot(3,3,j)
ut1=reshape(U(:,j),32,32);
ut2=ut1(32:-1:1,:);
pcolor(ut2)
set(gca,'Xtick',[],'Ytick',[])
end
% Projection of the first 40 individual cats and dogs onto the first three
% POD modes as described by the SVD matrix V.
% The left column is the first 40
% dogs while the right column is the first 40 cats.
```

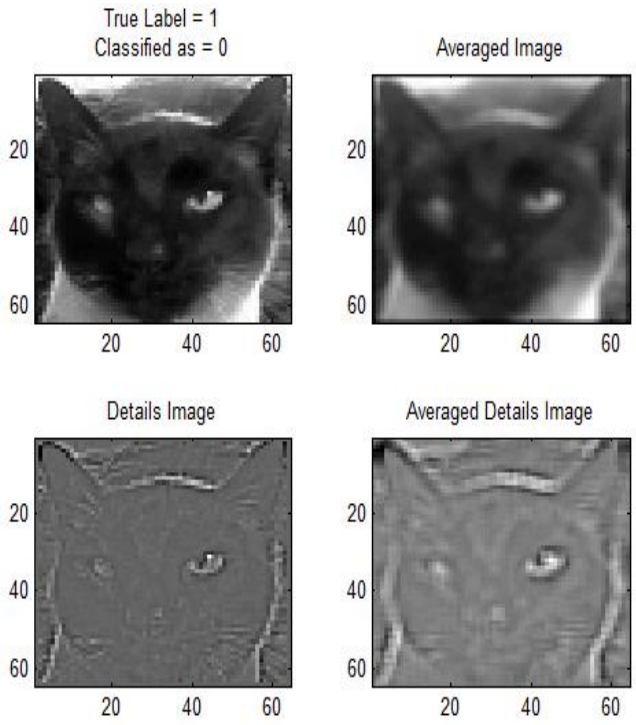

FIGURE 6. PA Misclassified Test Set Cat

```
% The three rows of figures are
% the projections on to the first three POD modes.

% subplot(2,1,1)
% plot(diag(S),'ko','Linewidth',2)
% set(gca,'Fontsize',14,'Xlim',[0 80])
% subplot(2,1,2)
% semilogy(diag(S),'ko','Linewidth',2)
% set(gca,'Fontsize',14,'Xlim',[0 80])
% figure(3)
% for j=1:3
% subplot(3,2,2*j-1)
% plot(1:40,V(1:40,j),'ko-')
% subplot(3,2,2*j)
% plot(81:120,V(81:120,j),'ko-')
% end
% subplot(3,2,1), set(gca,'Ylim',[-.15 0],'Fontsize',14), title('Dogs')
% subplot(3,2,2), set(gca,'Ylim',[-.15 0],'Fontsize',14), title('Cats')
% subplot(3,2,3), set(gca,'Ylim',[-.2 0.2],'Fontsize',14)
%
% subplot(3,2,4), set(gca,'Ylim',[-.2 0.2],'Fontsize',14)
```

```
% subplot(3,2,5), set(gca,'Ylim',[-.2 0.2],'Fontsize',14)
% subplot(3,2,6), set(gca,'Ylim',[-.2 0.2],'Fontsize',14)

%Performing Linear Discriminant Analysis while using SVD to find
%the optimal projection matrix

md = mean(dogs1,2);
mc = mean(cats1,2);
Sw=0; % within class variances
for i=1:nd
Sw = Sw + (dogs1(:,i)-md)*(dogs1(:,i)-md)';
end
for i=1:nc
Sw = Sw + (cats1(:,i)-mc)*(cats1(:,i)-mc)';
end
Sb = (md-mc)*(md-mc)'; % between class
[V2,D] = eig(Sb,Sw); % linear discriminant analysis
[lambda,ind] = max(abs(diag(D)));
w = V2(:,ind); w = w/norm(w,2);
vdog = w'*dogs1; vcat = w'*cats1;
result = [vdog,vcat];
if mean(vdog)>mean(vcat)
w = -w;
vdog = -vdog;
vcat = -vcat;
end
% dog < threshold < cat
sortdog = sort(vdog);
sortcat = sort(vcat);
t1 = length(sortdog);
t2 = 1;
while sortdog(t1)>sortcat(t2)
t1 = t1-1;
t2 = t2+1;
end
threshold = (sortdog(t1)+sortcat(t2))/2;

figure(4)
subplot(2,2,1)
hist(sortdog,30); hold on, plot([18.22 18.22],[0 10],'r')
set(gca,'Xlim',[-200 200],'Ylim',[0 10],'Fontsize',14)
title('Dogs')
subplot(2,2,2)
hist(sortcat,30,'r'); hold on, plot([18.22 18.22],[0 10],'r')
set(gca,'Xlim',[-200 200],'Ylim',[0 10],'Fontsize',14)
title('Cats')
```

```
%Melissa Dominguez
% Math 695
% Final Project
%Dc trainer file



%This file does the wavelet decomposition and extracts the details for our
%analysis
function dcData = dc_wavelet(dcfile);
[m,n]=size(dcfile); % 4096 x 80
nw=32*32; % wavelet resolution
nbcol = size(colormap(gray),1);
for i=1:n
X=double(reshape(dcfile(:,i),64, 64));
[cA,cH,cV,cD]=dwt2(X,'haar');
cod_cH1 = wcodemat(cH,nbcol);
cod_cV1 = wcodemat(cV,nbcol);
cod_edge=cod_cH1+cod_cV1;
dcData(:,i)=reshape(cod_edge,nw,1);
end
```

Kelly's classifier uses 3 functions: [Cener_method.m], [classify_centerMethod.m], and [Probe_checking.m]

```
function[r_1]=Center_Method(ccats,ddogs,Dr_Guai_Matrix)


[ nrows, nCats ] = size( ccats );
[ nrows, nDogs ] = size( ddogs );


%% Center Method:

% r_1 = [];

%cats_correct = zeros( 1, nCats );



for i = 1:size(Dr_Guai_Matrix,2)




    testCol = Dr_Guai_Matrix(:,i);

    cat = ccats;
```

```
    dog = ddogs;
%  ********************************************************************


    %% Kelly Classification Method:


    %cat = [1 0 2;-2 1 0;3 1 1];

    center_cat = mean(cat,2);

    %dog = [5 0 1;-1 3 2;2 7 -3];

    center_dog = mean(dog,2);

     % distance from testCol to center_cat:
    distance_centerCat_testCol = norm(center_cat - testCol);

    % distance from testCol to center_dog:
    distance_centerDog_testCol = norm(center_dog - testCol);



     if (distance_centerCat_testCol <  distance_centerDog_testCol)
        result = 1;  % 1 stands for cat
        r_1(i) = result;
    else
        result = 0;          % 0 stands for dog
        r_1(i) = result;

     end


end

r_1;









%% for conclusion:
```

```
% % when testing column is cat:  ----------------------------------------------
% Percent_correct_cat = (sum(r)/length(r))*100
% Percent_correct_dog = ((length(r) - sum(r))/length(r))*100
function [ probe_labels ] = classify_centerMethod( training, training_labels, probe )
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Assuming there are only 2 distinct values in [training_lables]:
unique_labels = sort( unique(training_labels) );
label1 = unique_labels(1);
label2 = unique_labels(2);

find( training_labels == label1 );

training1 = training( :, find(training_labels == label1) );
training2 = training( :, find(training_labels == label2) );


[r_1]=Center_Method(training2, training1, probe );


probe_labels = r_1


end

readRawImageFiles
cats_dogs = [ cats dogs ];     % cats_dogs is 4096 x 160
cats_dogs_labels = [ ones(1,length(cats(1,:))) zeros(1,length(dogs(1,:))) ];     %  cats_dogs

load PatternRecAns
for i = 1:size(TestSet,2)

    probe_labels(i)  = classify_centerMethod( cats_dogs, cats_dogs_labels, TestSet(:,i) );
end


classfied = abs(hiddenlabels - probe_labels);
accuracy = 100-100*sum(classfied)/size(TestSet,2)
```

Erich's classifier uses 4 functions: [averageImages.m], [classify_PA.m], [computePrincipalAngles.m], and [featuresImages.m]

```
function [ X_averaged ] = averageImages( X )
%
%   [ X_averaged ] = averageImages( X )
%
```

```
%   Each column of [X] should be 4096-by-1 vector that
%   corresponds to a 64-by-64 image. This function convolutes
%   every image with an average filter to smooth out the details
%   of each picture.
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    X_averaged = zeros( size(X) );
    average_filter = ones( 3, 3 );
    nVectors = length( X(1,:) );
    for i = 1:nVectors
        % Grab and reshape the current image:
        curr_image = reshape( X(:,i), 64, 64 );
        smoothed_image = conv2( curr_image, average_filter, 'same');
        % Reshape [smoothed_image] and hold on to it:
        X_averaged(:,i) = reshape( smoothed_image, 64*64, 1 );
    end
end
%%
%%
%%

function [ probe_labels ] = classify_PA( training, training_labels, probe )
%
%   [ probe_labels ] = classify_PA( training, training_labels, probe )
%
%   Uses principle angles to classify the columns of [probe] using the
%   columns of [training] and [training_labels] as a training set.
%
%   [training] and [probe] must contain the same number of rows, but may
%   contain different number of columns. [training] must have the same
%   number of columns as [training_labels]. [training_labels] is
%   used to denote class membership of the columns of [training].
%   This function is rudimentary in that it expects only 2
%   classes (2 unique vales in [labels]) for class membership.
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Get [nProbeVectors], the number of things we need to classify:
    [nRows, nProbeVectors] = size( probe );

    probe_labels = zeros(1, nProbeVectors );

    % Split training vectors into 2 classes, using
    % [training_labels] to determine class membership:
    unique_labels = sort(unique( training_labels ));

    % I'm expecting that unique_labels will have 2 elements:
```

```
label1 = unique_labels(1);
label2 = unique_labels(2);

training1 = training( :, find(training_labels==label1) );
training2 = training( :, find(training_labels==label2) );

% Applying filters for classification:
[ a_training1 ] = averageImages( training1 );
[ a_training2 ] = averageImages( training2 );
[ a_probe ] = averageImages( probe );

[ f_training1 ] = featuresImages( training1 );
[ f_training2 ] = featuresImages( training2 );
[ f_probe ] = featuresImages( probe );

[ a_f_training1 ] = averageImages( f_training1 );
[ a_f_training2 ] = averageImages( f_training2 );
[ a_f_probe ] = averageImages( f_probe );

fprintf(1,'Classifying...\n');
for iter = 1:nProbeVectors
    iter/nProbeVectors

    % Get the current probe vector (in all its forms):
    curr_probe = probe(:, iter );
    a_curr_probe = a_probe(:, iter);
    f_curr_probe = f_probe(:, iter);
    a_f_curr_probe = a_f_probe(:, iter);

    % Classify the curr_probe:

    % Find all relevant principal angles:

    [ theta1_1 ] = computePrincipalAngles( a_training1, a_curr_probe );
    [ theta2_1 ] = computePrincipalAngles( a_training2, a_curr_probe );

    [ theta1_2 ] = computePrincipalAngles( f_training1, f_curr_probe );
    [ theta2_2 ] = computePrincipalAngles( f_training2, f_curr_probe );

    [ theta1_3 ] = computePrincipalAngles( a_f_training1, a_f_curr_probe );
    [ theta2_3 ] = computePrincipalAngles( a_f_training2, a_f_curr_probe );

    % Everyone votes on what the curr_probe is:
    votes = [ (theta1_1 < theta2_1)  (theta1_2 < theta2_2) (theta1_3 < theta2_3) ];

    % Majority decision on the votes:
    if( sum(votes) >= 2)
        % Votes want to say that curr_probe is a training1 member:
```

```
            probe_labels(iter) = label1;
        else
            % sum(votes) <= 1
            % Votes want to say that curr_probe is a training2 memeber:
            probe_labels(iter) = label2;
        end
    end
end

%%
%%
%%

function [ theta ] = computePrincipalAngles( X, Y )
%
%   [ theta ] = computePrincipleAngles( X, Y )
%
%   Computes the principle angles between 2 subspaces given by
%   the real matrices [X] and [Y], where X is n-by-p and Y is
%   n-by-q. Principle angles are defined to be between 0 and pi/2
%   and listed in ascending order.
%
%   INPUTS:
%
%   X
%   X is a real-values n-by-p matrix
%
%   Y
%   Y is a real-valued n-by-q matrix
%
%
%   OUTPUTS:
%
%   theta
%   theta is a list of the principal angles between the subspaces
%   R(X) and R(Y).
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    [n,p] = size(X);

    [n,q] = size(Y);

    % Find the singular value decomposition of X:\
    [Ux,Sx,Vx] = svd(X,0);
    Qx = Ux * Vx';
```

```
    % Find the singular value decomposition of Y:
    [Uy,Sy,Vy] =  svd(Y,0);
    Qy = Uy * Vy';

    M = Qx' * Qy;
    [Um, Sm, Vm] = svd( M, 0 );
    sigma = diag(Sm);

    if( rank(Qx) >= rank(Qy) )
        Y1 = Qy - Qx*(Qx' * Qy);
    else
        Y1 = Qx - Qy*(Qy' * Qx);
    end

    [H,MU,Z] = svd( Y1, 0 );
    mu = diag(MU);

    % the values of [mu] are in decreasing order, when they need to be in
    % increasing order:
    mu = sort(mu);

    q = min( rank(X), rank(Y) );

    theta = zeros( q, 1 );
    for( k = 1:q )
       if sigma(k)^2 < (1/2 )
           theta(k) = acos(sigma(k));
       end
       if mu(k)^2 <= (1/2)
           theta(k) = asin(mu(k));
       end
    end

end
%%
%%
%%


function [ X_features ] = featuresImages( X )
%
%   [ X_features ] = featuresImages( X )
%
%   Each column of [X] should be 4096-by-1 vector that
%   corresponds to a 64-by-64 image. This function convolutes
%   every image with an Laplacian filter to extract out the
%   details of each picture.
%
%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    X_features = zeros( size(X) );
Laplacian_filter_2 = [  -1  -1  -1; ...
                        -1  8   -1; ...
                        -1  -1  -1  ];

    nVectors = length( X(1,:) );
    Laplacian_filter = Laplacian_filter_2;
    filter_size = length(Laplacian_filter(:,1));
    pad_number = floor(filter_size/2);

    for i = 1:nVectors

        % Grab and reshape the current image:
        curr_image = reshape( X(:,i), 64, 64 );

        smoothed_image = conv2( curr_image, Laplacian_filter, 'valid');

        % The above function call cut down the size of the image to a
        % 62-by-62, pad with zeros to get it back to a 64-by64:
        [nrows,ncols] = size( smoothed_image );

        % Pad the rows:
        smoothed_image = [ zeros(pad_number,ncols); smoothed_image; zeros(pad_number,ncols) ]

        [nrows, ncols] = size( smoothed_image );

        % Pad the cols:
        %size( zeros(nrows,pad_number) )
        smoothed_image = [ zeros(nrows,pad_number) smoothed_image zeros(nrows,pad_number) ];

        % Reshape [smoothed_image] and hold on to it:
        X_features(:,i) = reshape( smoothed_image, 64*64, 1 );
    end

end
%%
%%
%%
```

6.1. **References.**

(1) A special thanks to the Image Recognition Article by **J.N. Kutz** and **Dr. Jen-Mei Chang**.