Math 521: Final Project Report

Daniel Lam

May 13, 2022

1 Introduction and Overview of the Problem

Cats and dogs are very common household pets people have at homes. It would be nice to see how many people take pictures of cats or dogs. Also, it is interesting to know whether more pictures of cat are taken than pictures of dogs (or vice versa). To tackle this problem, we examine a couple of different classification algorithms to determine whether an animal in an image is a cat or a dog.

2 Theoretical Background

For this project, we classify 38 probe images with a training set containing 80 cats and 80 dogs. So, we will discuss our classification algorithms based on this training and probe set. For mathematical purposes, we organize this data into a matrix of size 4096×160 where each of the 160 column vectors represents a vectorized version of a 64×64 image. We will consider this matrix as X. The first classification algorithm is Principal Component Analysis (PCA). This algorithm uses the idea of best basis to determine whether a probe point is closer to one class or another. To find a best basis, we look at how we find principal vectors and how many of them are optimal enough to represent our data. The principal vectors are found by computing the eigenvectors of the covariance matrix, which is XX^T. Alternatively, the left singular vectors of X is mathematically equivalent to the eigenvectors of the covariance matrix XX^T. To find out why the eigenvectors of the covariance matrix and the left singular vectors are equivalent, it is suggested to read the Chang textbook provided in the resources section. Because it is more efficient to compute left singular vectors by SVD, we use the alternative method of finding left singular vectors as our principal vectors.

Now, we would like to compute an optimal number of basis vectors ideally less than 160 so that we can approximate our data matrix X. This optimal number, call it D, is determined by computing the singular values of the matrix X and determine when the ratio of the Dth singular value to the biggest singular value is less than a tolerance (usually 0.01 is the best empirical tolerance). Thus, our optimal basis is made with D basis vectors and is usually significantly smaller than 80.

Hence, we have our optimal basis of D basis vectors. Furthermore, these basis vectors are listed corresponding to the sorted singular values from largest to smallest. For instance, the first principal vector is associated with the largest singular value and so forth. Ideally, our principal vectors are listed this way because they represent the directions of how much our data varies. The singular values represents the magnitude of the variation of each principal component vector and is why we consider the first D singular values to be significant enough for our basis.

The approach for PCA classification is to compare what happens when a probe point is expanded among a basis of one class versus another class. Suppose these classes are the cat and dog classes. To determine what class a probe point belongs to, if the probe vector expanded by the cat basis is closer to itself than the probe point expanded by the dog basis, then that probe point is classified as a cat. Otherwise, it is classified as a dog.

The second classification algorithm considered is called K Nearest Neighbors (KNN). This algorithm classifies test points by considering the k closest points and is classified by the most frequent class among the k points. To determine how close each test point is from another point, there are different distance measures to consider

which can drastically change which points are considered *nearest*. Some distance measures that are considered for KNN are the p-norms specifically for cases p=1 (Taxicab) and p=2 (Euclidean). Using these distance measures all provide different ways to measure which points are nearest to a test point. Ultimately, finding the best distance measure for KNN is done empirically and one way of doing this is to go through testing half of the training set against its other half. As for determining the optimal value of k, this is done empirically as their is no closed form method that provides an optimal k in this algorithm.

3 Algorithm Implementation

To do PCA, we separate our training data of 160 images into two distinct classes where one class has 80 dogs and the other class has 80 cats. This is done so that we will have two different bases for each animal. Both the cat class and dog class are represented as matrices with size 4096×80. Next, we perform a mean subtraction, otherwise known as the *caricature*, which removes background noise and focuses more on the character of each image. This is done where the cat and dog class are mean subtracted by the mean of the cat and dog gallery concatenated as an entire matrix.

The remaining procedures are done to both the cat and dog classes separately. Now, we begin with performing SVD on each class to gather our principal vectors and the optimal number of principal vectors. We first calculate the optimal number of principal vectors D for each class. To do this, we first calculate all of the singular values and sort them in descending order. Next, we heuristically determine the best value of D by examining when the ratio of the Dth singular value to the first singular value is less than the tolerance (empirically set to 0.01). Then, we compute the left singular vectors ordered based on the associated descending singular values and reserve only the first D left singular vectors. Using the SVD function on MATLAB allows us to find the left singular vectors (the principal vectors) and the singular values, where they are listed in descending order. Furthermore, the singular vectors are also scaled so that each of them are normalized and are also orthogonal to one another. These singular vectors and singular values are essential to constructing a different basis of our cat and dog classes where we then keep only the most important principal vectors corresponding to the largest eigenvalues.

After computing SVD, we now consider the new probe images in vectorized form and expand each of them in both the cat and dog basis. To expand each probe image in a basis, we find the coefficient expansion A of each probe image by calculating $U^T y$ where U is the optimal basis based on that class and y is the probe image. Next, we compute the approximated probe vector using that coefficient expansion by evaluating UA. Finally, we compute the error by taking the Euclidean norm of the difference between the original probe vector and the approximated probe vector expanded by the cat basis. If this error is the smallest under a cat basis compared to the error under a dog basis, then the probe image is classified as a cat and, otherwise, the probe image is classified as a dog.

For KNN, the algorithm is more intuitive than PCA. Instead of splitting the training set, we will keep the training set as it is in vector form and treat each of the column vectors as points in their resolution dimension. Just like PCA, we will mean subtract each column vector by the mean of the entire training set, i.e. evaluating the caricature of the training set. Afterwards, we find the k nearest neighbors of that probe point based on the Euclidean norm. It is desirable for k to be an odd number to avoid tiebreakers when two classes have an equal number of points among the k nearest neighbors. Among those points, we classify the probe point based on what class is the majority of the k nearest neighbors.

4 Computational Results

PCA gave very accurate results with classifying cats and dogs using a full training set (full gallery). Using a full training set of 80 dogs and 80 cats, we have the best accuracy of about 89%. On the other hand, using a smaller training set was much worse in terms of accuracy with an accuracy of 55%. It may seem counter-intuitive, but using filters that blur or enhance both the training set and the probe set does not decrease the accuracy.

To show this, we tested smaller training sets with 20 cats and 20 dogs as the training set (quarter gallery) and another with 40 cats and 40 dogs (half gallery). The following accuracy table displays the accuracy of not using a filter and for each of the four filters used.

	Quarter Gallery	Half Gallery	Full Gallery
No Filter	0.5526	0.6842	**0.8947**
Median Filter	0.5789	**0.7895**	**0.8947**
Gaussian Filter	0.5526	0.7632	**0.8947**
Box Filter	**0.6053**	**0.7895**	0.8684
Mean Filter	0.5789	**0.7895**	0.8684

The bolded accuracies indicate that it is better to use a box filter when dealing with smaller training sets. To note, a box filter is similar to a mean filter except that it does not pad a matrix by zeros when performing the filter and instead pads the matrix with the closest pixel values. However, using a full training set indicates that there was no need to have a filter in order to gain a better accuracy. Utilizing the best accuracy with a full gallery with no filter, the following table describes how many cats and dogs were classified correctly as well as how many dogs are misclassified as cats and vice versa.

	Dog	Cat
Dog	17	2
Cat	2	17

The following probe images shown in fig. 1 and fig. 2 are misclassified by our PCA algorithm.

Figure 1 – These cats were classified as dogs according to PCA

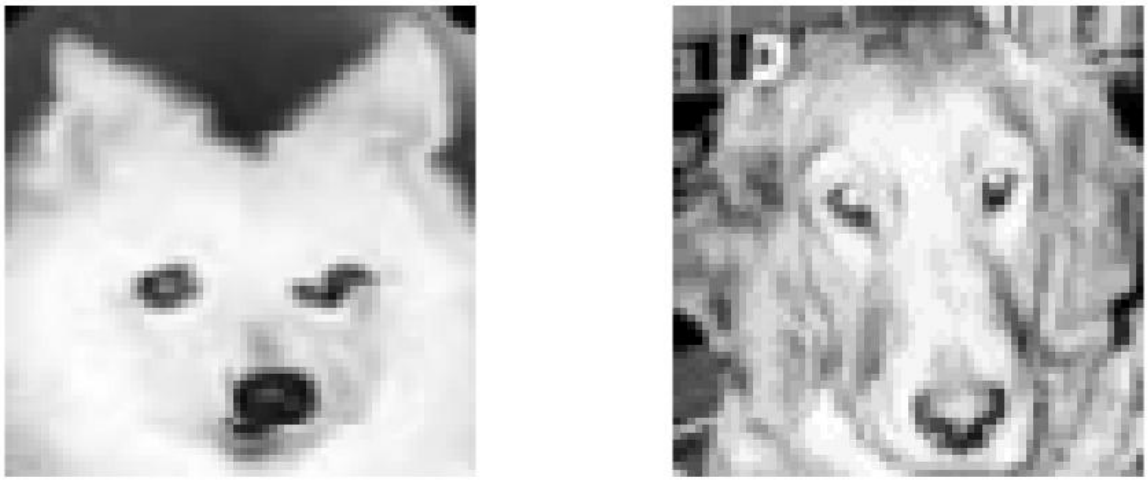

Figure 2 – These dogs were classified as cats according to PCA

For KNN, our classification accuracy of the probe set is best without any filters applied to the training and probe sets. The optimal k is found to be 32 nearest points with the distance metric as the Euclidean norm. The accuracy here is roughly 66% (25/38).

5 Discussion

The PCA algorithm is a powerful classification technique that uses a minimal number of principal vectors as the optimal basis that closely represents the original training set and uses that basis to represent the new probe vector as a linear combination of the new optimal basis. Then, after extracting the coefficients of that linear combination, they are used to recreate the probe vector based on the principal vectors made by the cat basis and the principal vectors made by the dog basis. From there, the errors are calculated based off of the Euclidean distance between the recreated probe vector and the actual probe vector we originally started with. Finally, the probe vector is classified based off of which animal basis has the smallest error.

This algorithm worked very well in comparison to KNN because it identifies the details of every picture in the training set and considers the principal components in an efficient manner whereas KNN classifies purely on distance between two vectorized images. In other words, PCA has a higher attention to detail than KNN. To explain the counter-intuitive result of filters providing better accuracies for smaller training sets, it can often be easier for algorithms to detect blurred pictures because a filtered image can have more uniform colors in certain areas which can highlight a pattern of a dog or cat.

One improvement that can be made for PCA is to use a different method of classifying probe vectors such as using the technique of KNN after computing the dog and cat basis. That is, after converting the probe vectors as coefficient vectors in the cat and dog basis, use KNN to classify the probe vector by comparing it to an entire gallery of cats which each cat image is expanded by the cat basis and similarly with dogs.

The KNN algorithm simply seeks for a number of nearest points based on Euclidean distance and seeks for the class majority among those nearest points. Since it is a simplistic algorithm, it does not perform as great in accuracy compared to PCA. A big drawback to KNN is that it can fail whenever two classes of data are intertwined. In other words, KNN can be as worst as a coin flip if the two classes are not randomly dispersed where there is no discernible pattern for class 1 and class 2. Another disadvantage to KNN is that classifying a probe point by its nearest neighbors is not always accurate. However, one advantages KNN has over PCA is that KNN is a quick and

simplistic algorithm to run when one has an optimal k value.

If this project were to be done again, I would try to repeat the PCA algorithm but randomly pick different images for a small training set rather than only picking the first N preceding images to determine accuracy. For the KNN algorithm, there may be other distance measures to consider such as using the cosine measure when comparing which points are nearest to a probe vector where the cosine of the angle between two vectors is the dot product of the two vectors divided by their magnitudes. Some algorithms that I could take a further look into are Linear Discriminant Analysis (LDA) and Principal Angles. LDA is the process of projecting data points onto an optimal projector w where these projected data are ideally separated by a point called α. The optimal projector w is found such that the distance between the mean of class 1 and mean of class 2 are maximized while the projected data points of the same class have minimal variation. Principal angles can tie in with KNN because principal angles uses the idea of how far apart the directions of two vectors are instead of the magnitudes. So, principal angles can provide a better accuracy for KNN.

6 Resources

Chang, Jen-Mei. *Matrix Methods for Geometric Data Analysis and Pattern Recognition*, 2014.

A Code for PCA

```
1
2  loadtifftraining
3  clearvars -except training_labels X;
4
5  load('PatternRecAns.mat'); % contains TestSet and hiddenlabels
6  X = double(X); % X has 80 cats and 80 dogs for our training set
7
8  % Filter X before doing PCA
9  % filter(~,"none") means no filter on X where "none" means a filter
10 % name not specified in the user made function filter.
11
12 for i=1:size(X,2)
13 reshapedColX = reshape(X(:,i),64,64);
14 filteredColX = filter(reshapedColX, "none");
15 X(:,i) = reshape(filteredColX,size(X,1),1);
16 end
17
18 % Filter TestSet and rename as probes
19 for i=1:size(TestSet,2)
20 reshapedColTestSet = reshape(TestSet(:,i),64,64);
21 filteredColTestSet = filter(reshapedColTestSet, "none");
22 probes(:,i) = reshape(filteredColTestSet,size(TestSet,1),1);
23 end
24
25 % Using all 160 images as your gallery (full gallery)
26 catGallery = X(:,1:80); %all cats
27 dogGallery = X(:,81:160); % all dogs
28
29 answers = hiddenlabels'; % answers for testSet
30 galleryMean = mean(X,2);
31
32 catGallery = catGallery - galleryMean;
33 dogGallery = dogGallery - galleryMean;
34
35 % Begin PCA
36 D_cat = optimalD(catGallery);
37 % D_cat = 80;
38 D_dog = optimalD(dogGallery);
39 % D_dog = 80;
40
41 [U_cat] = U_truncate(catGallery,D_cat);
42 [U_dog] = U_truncate(dogGallery,D_dog);
43
44 correct = 0; incorrect = 0;
```

```
catsCorrect = 0; dogsCorrect = 0;
catsIncorrect = 0; dogsIncorrect = 0;
guesses = zeros(size(probes,2),1);
for i=1:size(probes,2)
guesses(i) = predictor(probes(:,i),U_cat,U_dog);
% Compare guess to actual answer from hiddenlabels
if (guesses(i)==answers(i))
correct = correct + 1;
if (answers(i)==1)
catsCorrect = catsCorrect + 1;
else
dogsCorrect = dogsCorrect + 1;
end

else
incorrect = incorrect + 1;
if (answers(i)==1)
catsIncorrect = catsIncorrect + 1;
else
dogsIncorrect = dogsIncorrect + 1;
end

end
end

confusionMatrix = [dogsCorrect dogsIncorrect; catsIncorrect catsCorrect];
TP = dogsCorrect; FN = dogsIncorrect;
FP = catsIncorrect; TN = catsCorrect;

accuracy = (TP+TN)/sum(confusionMatrix,'all');
precision = TP/(TP+FP);
specificity = TN/(TN+FP);
negativePredictedValue = TN/(TN+FN);

% List of User-Defined Functions

function [U] = U_truncate(X,D)
[U,~,~] = svd(X, "econ");
U = U(:,1:D); % keep first D columns of U
end

function [D] = optimalD(X)
% Calculate optimal D value based on ratio of squared singular values
% or ratio of eigenvalues of XX^T
[~,S,~] = svd(X,0);
sigma = diag(S);
evals = sigma.^2;
```

```
ratioOfEvals = evals ./ evals(1);
tolerance = 0.01;
for i=1:rank(X)
if (ratioOfEvals(i) < tolerance)
D = i;
break;
end
end
D = i;

end

function [guess] = predictor(y, U_cat, U_dog)
newACat = U_cat'*y; % gather coefficients for cat basis
newADog = U_dog'*y; % gather coefficients for dog basis

newYCat = U_cat * newACat; % y expanded by cat basis (catY)
newYDog = U_dog * newADog; % y expanded by dog basis (dogY)

errorCat = norm(newYCat - y); % difference between y and catY
errorDog = norm(newYDog - y); % difference between y and dogY

% guess cat when the expansion of cat basis is closer to y
if (errorCat < errorDog)
%         disp("CAT")
guess = 1;
else % otherwise guess dog
%         disp("DOG")
guess = 0;
end

end

% filter each column vector of X by nameOfFilter with 3x3 shifted window
function newImage = filter(X,nameOfFilter)
% Apply mean filter with 3x3 shifting window
if (nameOfFilter == "mean")
newImage = meanFilter(X);
% The remaining filters are built in by MATLAB.
% Apply median filter with 3x3 shifting window
elseif (nameOfFilter == "median")
newImage = medfilt2(X);
% Apply 2D Gaussian filter
elseif (nameOfFilter == "gaussian")
newImage = imgaussfilt(X);
% Apply Sharpening filter using unsharp filtering
elseif (nameOfFilter == "sharpen")
```

```
newImage = imsharpen(X);
% Apply 2D Box Filtering of Images
elseif (nameOfFilter == "box")
newImage = imboxfilt(X);
else
% Apply no filter since nameOfFilter argument is not recognized
newImage = X;
end

end

% Self-Made Mean Filter algorithm
function mean_filtered = meanFilter(I)
I_padded = padarray(I,[1,1]);
mean_filtered = zeros(size(I)); % this is preset as a double instead of uint8
for i=1:size(I,1)
for j=1:size(I,2)
local = I_padded(i:i+2,j:j+2);
mean_filtered(i,j)=mean(local(:)); % local(:) is 9x1
end
end
end


```

B Code for KNN Function

```
% Courtesy of Simon Bar-On
function[classified] = KNN(training, testing, training_labels, k)
%this function takes in a set of training data, testing data, labels for
%the training data, and a natural number k, and then performs k nearest
%neighbors classification

m = size(testing,2);
n = size(training,2);
%in this array I will store the indices of the k nearest
%training members as well as their distance from the current test
%data point
closest_individual = zeros(k,2);

classified = zeros(m,1);

for i = 1:m
%this loop iterates through each of the test points
for j = 1:n % this loop iterates through the training points
%if we haven't checked k items yet, we just include the first k and their distances
if (j<=k)
closest_individual(j,1) = j;
closest_individual(j,2) = norm(training(:,j)-testing(:,i));%this does euclidean distance
else
temp = norm(training(:,j)-testing(:,i),3);
largest = max(closest_individual(:,2));
%alright if this is true, we need to replace the index of largest with the current
if temp<largest
[row, ~, ~] = find(closest_individual == largest);
closest_individual(row,1) = j;
closest_individual(row,2) = temp;%now we have succesfully replaced the least close
%neighbor with the new closer neighbor
end
end
end
% ok now ideally we have found the k nearest neighbors of test point i
neighbor_labels= zeros(k,1);
for p = 1:k
neighbor_labels(p) = training_labels(closest_individual(p));
end
classified(i) = mode(neighbor_labels);
end
end
```

C Code for KNN Project

```
% Courtesy of Simon Bar-On
whole = load('PatternRecData.mat');
train_images = double(load('tiffstudentdata_raw_vectorized').X);
train_labels = whole.sublabels;
testers = load('PatternRecAns.mat');
test_images = testers.TestSet;
test_labels = testers.hiddenlabels;

for i=1:size(train_images,2)
reshapedColX = reshape(train_images(:,i),64,64);
filteredColX = filter(reshapedColX, "box");
train_images(:,i) = reshape(filteredColX,size(train_images,1),1);
end

for i=1:size(test_images,2)
reshapedColX = reshape(test_images(:,i),64,64);
filteredColX = filter(reshapedColX, "box");
test_images(:,i) = reshape(filteredColX,size(test_images,1),1);
end

MEGA = [train_images,test_images];
MEGA_average = mean(MEGA,2);

average_train = mean(train_images,2);
average_test = mean(test_images,2);

% Mean subtracted
train_images = train_images-MEGA_average;
test_images = test_images-MEGA_average;

testlabels = KNN(train_images, test_images, train_labels, 32);

% testlabels = KNN(train_current, test_current, labels(1:size(train_current,2)), 32);

%next thing is for tuning optimal k
optimal = 0;
min = 10^15; % make sure min is super high so that it won't be the initial value
for k = 1:50
testlabels = KNN(train_images, test_images, train_labels, k);

%the indiced of labels must be changed according to which data is test and train
current = sum(abs(testlabels'-test_labels));
if(current<min)
min = current;
```

```
optimal = k;
end
end

testlabels = KNN(train_images, test_images, train_labels, optimal);
current = sum(abs(testlabels'-test_labels));
C = testlabels' == test_labels;
sum(abs(C))

function newImage = filter(X,nameOfFilter)
% Apply mean filter with 3x3 shifting window
if (nameOfFilter == "mean")
newImage = meanFilter(X);
% The remaining filters are built in by MATLAB.
% Apply median filter with 3x3 shifting window
elseif (nameOfFilter == "median")
newImage = medfilt2(X);
% Apply 2D Gaussian filter
elseif (nameOfFilter == "gaussian")
newImage = imgaussfilt(X);
% Apply Sharpening filter using unsharp filtering
elseif (nameOfFilter == "sharpen")
newImage = imsharpen(X);
% Apply 2D Box Filtering of Images
elseif (nameOfFilter == "box")
newImage = imboxfilt(X);
else
% Apply no filter since nameOfFilter argument is not recognized
newImage = X;
end
end

% Self-Made Mean Filter algorithm
function mean_filtered = meanFilter(I)
I_padded = padarray(I,[1,1]);
mean_filtered = zeros(size(I)); % this is preset as a double instead of uint8
for i=1:size(I,1)
for j=1:size(I,2)
local = I_padded(i:i+2,j:j+2);
mean_filtered(i,j)=mean(local(:)); % local(:) is 9x1
end
end
end
```