Project 3: Dimension Reduction & Classification

EGR 190 Fundamentals of Data Analysis and Decision Science
Instructors: Dr. Stacy Tantum and Dr. Paul Bendich

Dima Fayyad
Kyle Newman
Feroze Mohideen
Maddie Wilkinson

30 April 2018

Problem Statement

The final project aims to explore the ways that reducing the dimensions of data can help with binary classification. Oftentimes, data from sensors or networks can be recorded with very high dimensions. This high dimensional data takes up large amounts of storage and has a high processing time. To help alleviate this problem, the data can be reduced into lower dimensions. An overview of how this is done is by projecting the high dimensional data down onto a lower set of principal components. Principal component analysis attempts to find a basis of principle component vectors that help account for as much variability as possible from the data. This basis is then used for the project from the higher dimensional data.

The other aspect of the final project is to investigate the ways that dimension reduction can be used to help classify data. Oftentimes classification in higher dimensions works out better, since none of the variability is lost due to dimension reduction. But working in higher dimensions increases runtime for the classifier and increases the amount of space needed to store data. Through this project, we investigate the advantages and disadvantages of classifying in lower dimensions through dimension reduction.

The specific task at hand is to take a set of images of faces or not faces, break them down into their principal components, and use those principal components to classify the images based on certain parameters. This project explored two ways to reduce the dimension of the images, eigenfaces and fisher faces, as well as two different types of classification, linear discriminant analysis (LDA) and K nearest neighbor (KNN). This process can be used as a basic model for facial recognition software. By reducing the dimensions of the images, the software can process

images faster and use less data to store the images. This would help in reducing overhead for a security company or police department when trying to use multiple images to identify a suspect at the scene of the crime.

Another example in which dimension reduction can be important is modelling the neuron's probability of generating an action potential when responding to certain external stimuli. An action potential is potential difference created by the neuron in order to signal signals to other neurons in response to an external stimulus. Data is gathered by subjecting the neuron to white noise, which triggers different action potential responses and those reposes are then recorded. To help determine what from the white noise cause the action potential shift dimension reduction of that white noise and the resulting response is applied. Due to the nature of the first principal component accounting for the highest variance and the next component accounting for the next highest variance of the data, the first few principal components can approximate what from the white noise caused the resulting change in the neuron.

Dimension Reduction

The image data of faces that was given to us was in the form of four hundred 112x92 matrices with individual elements that represented pixels in RGB space. Each matrix, therefore, could be thought of as a single vector in 10,304 dimensions. The classification methods covered throughout the semester were only practiced in up to two dimensions; however, we had the tools in place to extend their use to as high of a dimension as R^10,304. As such, our first plan of action was to confirm that both LDA and kNN were able to handle this higher-dimensional data in the classification problems listed below. The results of this analysis are also discussed below.

Although it was eventually decided that the kNN classifier was effective in the high dimension specified by the image vectors, we next wished to observe the results of the same classifiers given a different dataset, one generated by a method also discussed in class—that is, principal components analysis (PCA). The ultimate aim of PCA is to reduce the dimension of a dataset while still capturing the high variances in some dimensions, and ignoring the dimensions in which there is relatively low variance. This is a hugely important statistical tool in easier data visualization, but can also be extended to other real-world applications like compression algorithms and neural networks.

Rather than scripting the computationally intensive formal mathematical way of calculating the principal components, we instead took a shortcut and used MATLAB's `PCA` function for the same result. Initially, we were confused as to why the function only returned 399 principal components from the aggregate 400x10,304 image face matrix - we expected at least 400, or even 10,304. However, upon brushing up on how exactly PCA works, we were satisfied with the result. In general, PCA shifts the dataset to the centroid of the data, then scales and rotates axes on which to project the data such that the projections capture as much of the variance of the data in each dimension as possible. For example, in a dataset with 2 points in 3 dimensions, after mean-centering, the two points would be connected with a line, and the resulting subspace would be one-dimensional. As a result, mean-centering of the data in effect removes a dimension, such that the original 400 vectors would create 399 non-zero eigenvectors. Generalizing, N points determine an $N-1$ Euclidean space.

Our goal was to reduce the image dataset into its principal components, then compare the results of the classifier under a number of those principal components with the results of the classifier under the unmodified, higher dimensional dataset. One way of deciding how many principal components is enough to use when modeling a dataset is to create a percentage of variance (PoV) plot of the eigenvalues from PCA, which represent the variance of the data in each subset of dimensions, from 1 to 10,304. Figure 1 shows this PoV plot.

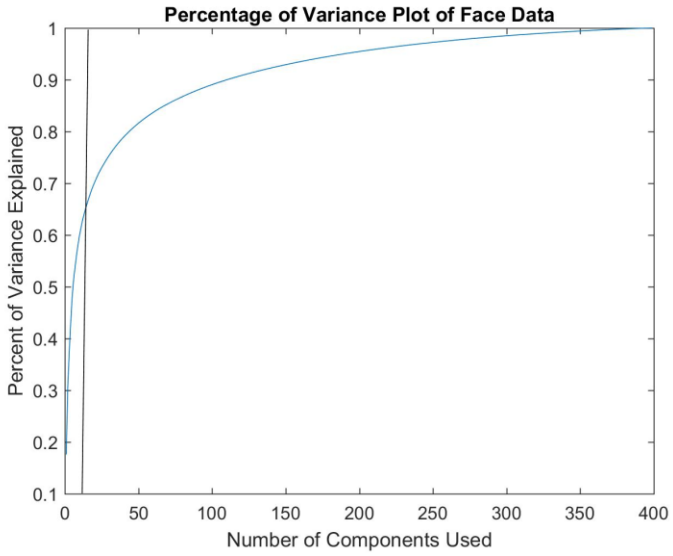

Figure 1: Percentage of Variance Explained vs. Number of Principal Components

In addition to the above method, we were also able to test the classification problems detailed below using a certain number of principal components of the data, and assessing their accuracy from there. This assessment was determined by the area-under-curve (AUC) of the ROC curve generated using different training and test data for a classifier with 5-fold cross-validation. The AUC was calculated for each subset of principal components, from 1 to all 399. The result of this computation is graphed in Figure 2, with kNN as the classifier of choice and the ‘glasses’ problem as the classifying factor:

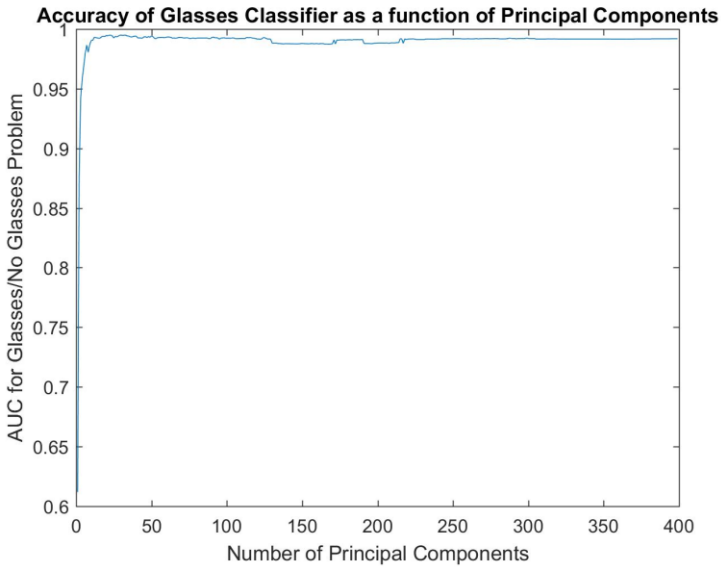

Figure 2: The accuracy of the KNN classifier for the glasses problem for between 0 and 400 principal components

The above plot is also shown zoomed in below:

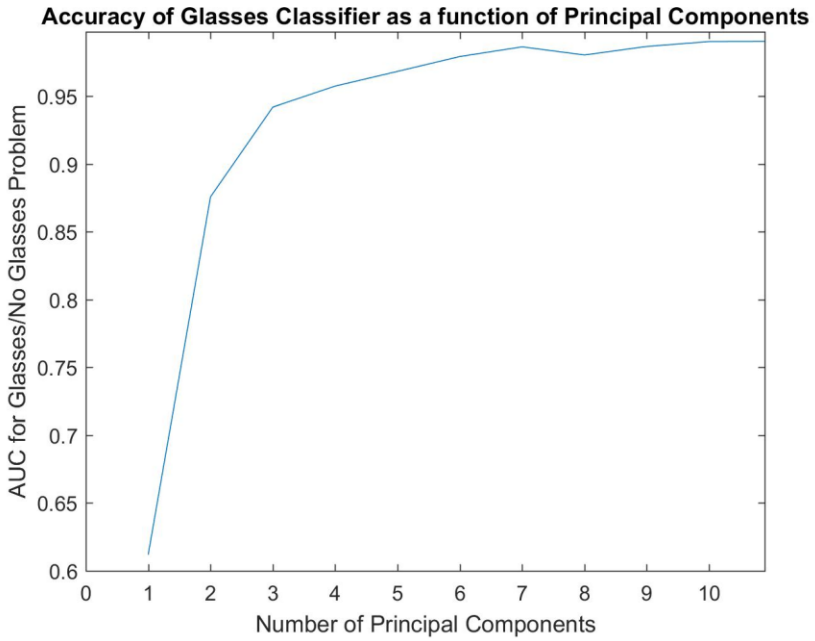

Figure 3: The accuracy of the KNN classifier for the glasses problem using the first 1 to 10 principal components

As shown in Figure 3, there is not much difference between using the first 8 principal components and any number higher than that in terms of classifier accuracy. This result is also confirmed in Figure 2 plot, where the first 8 principal components capture almost 60 percent of variance of the dataset (60% isn't too high - however, lacking a real 'knee' in the PoV plot and in the interest of computational power, 8 dimensions was deemed sufficient). As a result, we decided to reduced the dataset into around 8 dimensions, since we were confident that 8 dimensions was enough to capture enough of the variance in face data; however, this number changes slightly given the classification problem and classifier, which is discussed later in the report.

Another interesting question regarding dimension reduction is whether the first *n* principal components are the best *n* principal components to project the dataset onto. Although for much of the classification we assumed that this was the case, we discuss alternatives to this approach in the following section - namely, using the optimal subset of principal components to generate a better classifier.

Dimension Reduction Extended: Forward-Searching Principal Components

As mentioned previously, a question we sought to answer in the dimension reduction problem was not only how many principal components are needed to best answer our classification

problem, but after deciding on a number, which principal components were the best to use. Previous classification relied on simply choosing the first *n* principal components from the set of eigenvectors to the covariance matrix; however, we wished to learn whether that was the best decision policy, or whether a different combination of *n* principal components was better in terms of classification accuracy.

The classification problem chosen to test the validity of decisions was the glasses classification problem, for no particular reason; the following analyses could have been extended to any classification problem. Firstly, we tested the accuracy of the classifier at different start indices, but using the same number of principal components (in this case, 8). In other words, we tested the classifier using principal components 1:8, then 2:9, 3:10, and so on. A plot showing the AUC (which was used as a measure of classifier accuracy) as a function of these different start indices is shown in Figure 4:

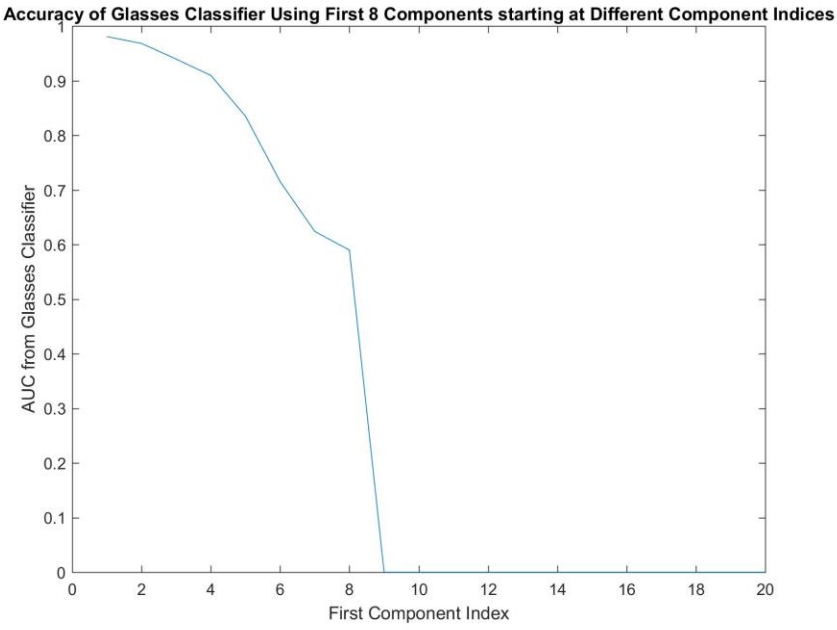

Figure 4: The accuracy of the KNN classifier for the glasses problem given different PC start indices

We chose to omit the start indices past 20, since as shown in Figure 4, the AUC dramatically decreases past using principal components 8:15. We wondered why that may have happened, and we decided that the latter principal components are associated with eigenvalues that don't explain much variance in the dataset (compared to the initial ones). Therefore, that may explain why the classifier may be inaccurate to a certain extent, since there is not much variance in the dataset to allow classes to be distinguishable enough.

Afterwards, we tested combinations of 8 eigenvalues that were not consecutive. We wrote a script in MATLAB that sequentially runs through the entire list of principal components and performs classification, then reports back an AUC. The next principal component that yields the highest AUC from the remaining principal components is chosen as the optimal next principal component, and the cycle continues until a set of 8 (or however many) principal components are chosen. We were interested to see whether this set of 8 would end up being simply the first 8 principal components, or a different set.

However, there were some caveats to this approach. It turned out that the algorithm (or at least our implementation) was extremely computationally heavy and took much time to generate the optimal set. Therefore, for the purposes of this analysis, we chose to limit the scope of principal components to the first 50, instead of the full 399. We found that our algorithm, when asked for 8 principal components, did not simply return the first 8, but a set that included principal components **[1 2 6 3 9 12 18 17]**. We validated this result by comparing the AUC of this classifier with those principal components with the AUC resulting from the first 8 principal components, and finding that the former value was slightly higher.

This result is shown visually in Figures 5 and 6 through two different ROC curves, though with a different number of principal components than previously (in this case, 3):

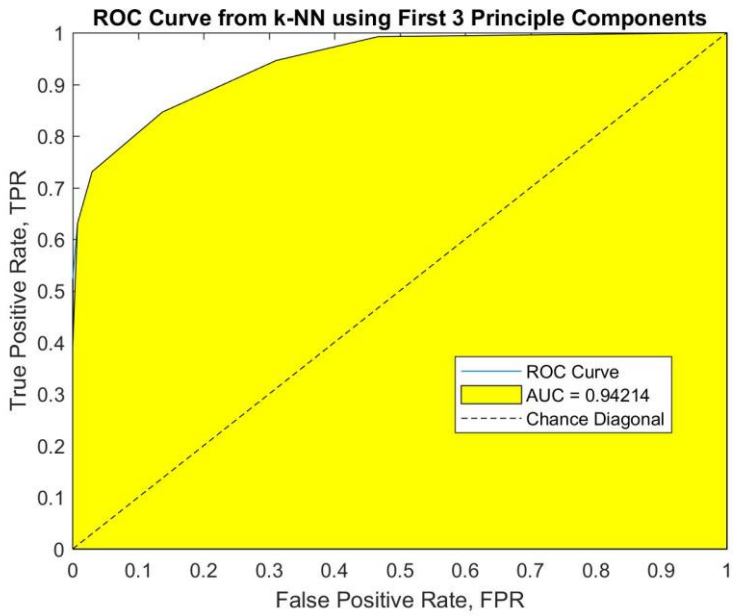

Figure 5: ROC Curve from the Glasses Problem with KNN and first 3 PC's

Figure 6 shows classification using principal components **[1 2 6]** instead, which indeed has a (slightly) higher AUC:

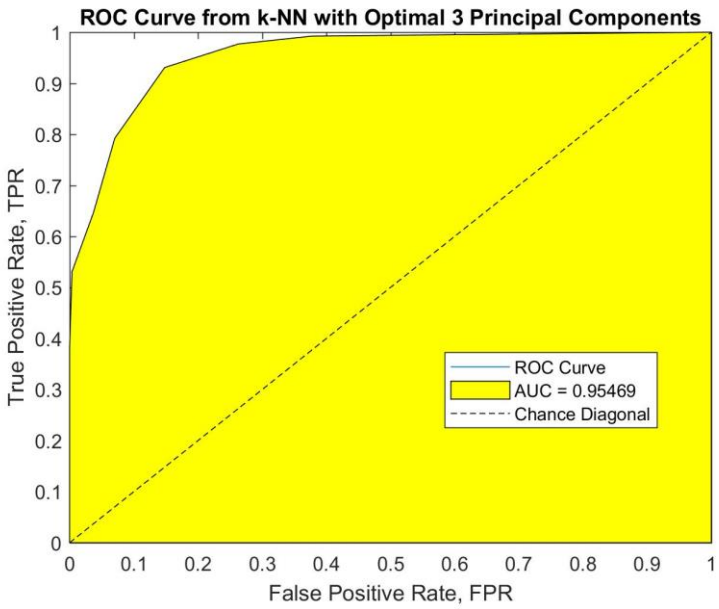

Figure 6: ROC Curve from the Glasses Problem with KNN and optimal 3 PC's

Reconstructed Faces Visualized

As a part of understanding how an image can be reconstructed using the projections onto the principal components (basis vectors) chosen, we visualized them using the imagesc command in Matlab. We compared these reconstructions using different choices of principal components to the original image.

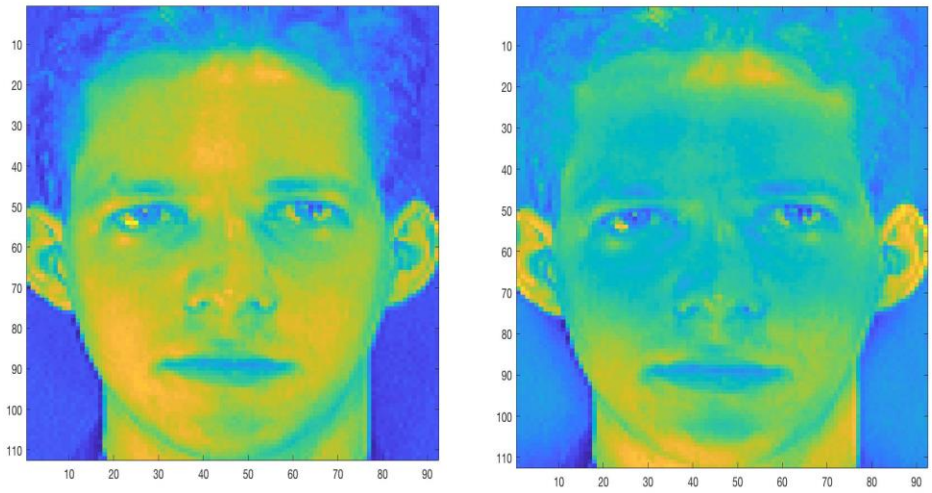

Figure 7: Original image displayed using Imagesc (left) and reconstructed image using all 399 principal components (right)

Figure 7 shows the original image side by side with the same image reconstructed using all available principal components. There are some very obvious differences between the two images in terms of shading on the face. The same two images are shown below in black and white (Figure 8), which highlights this change in shadowing on the face. What remains remarkably similar between the two photos is the quality of the reconstruction: despite the differences in shadowing, the two images are clearly the same photo of the same person.

Figure 8: Original image displayed (left) and reconstructed image using all 399 principal components (right) in black and white

The reason that the image reconstructed using all non-zero principal components is not identical to the original image is because the principal components are generated from the matrix of all 400 photos. Had the principal components been generated using only the matrix corresponding to this particular photo, the reconstructed photo would be identical. Had the principal components been generated using the matrix corresponding to all ten available photos of this particular person, the reconstructed photo would be very close to the original.

As the number of principal components used to reconstruct the image is decreased, the quality of the reconstructed image and its similarity to the original photo deteriorates quite rapidly. As figure 9 below demonstrates, the image reconstructed using 20 principal components is a much worse reconstruction than that made using all principal components, but still looks somewhat sharper than those reconstructed using only eight principal components. There is no visible difference between the images reconstructed using the best 8 principal components and the first 8

principal components. It makes sense that the differences that are present cannot be determined visually, especially for so few principal components.

Figure 9: (Left) Image reconstructed using first 20 principal components. (Center) Image reconstructed using first 8 principal components. (Right) Image reconstructed using best 8 principal components.

Fisherfaces

The Method:

One way in which Principal Component Analysis (PCA) falls short, is that it does not make use of all the information available. For this problem in particular, we were given four-hundred photos, and were told that every ten photos correspond to one person. This meant that the labels for the data are known, which is information that was not used when performing PCA that could prove useful for classification. Fisher's Linear Discriminant (FLD) is a similar method of dimension reduction that makes use of the class-specific labels available. The critical difference between FLD and PCA occurs when solving the eigenvalue problem that allows us to find a new, lower dimensional basis onto which we may project the data. While PCA solves the general eigenvalue problem for the covariance matrix of the entire data pooled together, FLD solves the eigenvalue problem for the between class scatter matrix S_b and the within-class scatter matrix S_w:

$$S_b W = \lambda S_w W$$

Fisher's Linear Discriminant maximizes the ratio between the projected between class scatter S_b and within class scatter S_w.

The scatter matrices are defined to be:

$$S_{\square} = \sum_{i=1}^{c} \quad \sum_{x_k \epsilon X_i} \quad (x_k - \mu_i)(x_k - \mu_i)^T$$

And

$$S_{\square} = \sum_{i=1}^{c} \quad N_i(\mu_i - \mu \ )(\mu_i - \mu \ )^T$$

Where c is the number of unique classes in the labels, $\mu_{\square}$ is the mean of the i^{th} class, μ is the total mean, and $N_{\square}$ is the number of samples in the i^{th} class. Since we have fewer photos in the training set than we do pixels, calculating S_w produces some problems. Since the rank of S_w can only be less than or equal to the number of photos in the training set minus the number of classes, then S_w can be singular. To avoid this problem, we first perform PCA on the data, project it into a lower dimension, and then perform FLD to further reduce dimension using class specific information.

KNN Classifier

The K nearest neighbor classifier works by looking at a set amount, which is determined by the user, of the nearest neighbors of the test point. It creates a decision statistic by comparing how many of neighbors of data type B are closest to the test point versus the total number of neighbors being compared (K). To carry this out a MATLAB script was created that generated the distances from all of the training data to the test data point, choosing the K nearest neighbors, and then assigning a predicted label depending on the decision statistic. For this classifier a decision statistic of 0.5 was chosen, which means that the predicted label was chosen to be the label of the majority of nearest neighbors. To avoid the cases of ties, the value of K was chosen to be an odd value. When testing the classifier with the given faces, 5 fold cross validation is used to generate the receiver operating characteristic (ROC) curve.

Classification Problem: Face versus Not a Face

The first to complete was to be able to use eigenfaces and fisher faces to distinguish between images of faces and not faces. The set of images that were not faces were comprised everyday objects (ie a spoon, a keyboard, etc.), which to a human are very easy to distinguish between face and not a face. Since there were both 400 images of faces and 400 images of non-faces, the labels were easy to assign, with faces receiving a ‘0’ and non-faces receiving a ‘1’. The classification method chosen was KNN. Below is the ROC generated when using 5 fold cross validation and K=3 with no dimension reduction along with the accompanying confusion matrix and AUC.

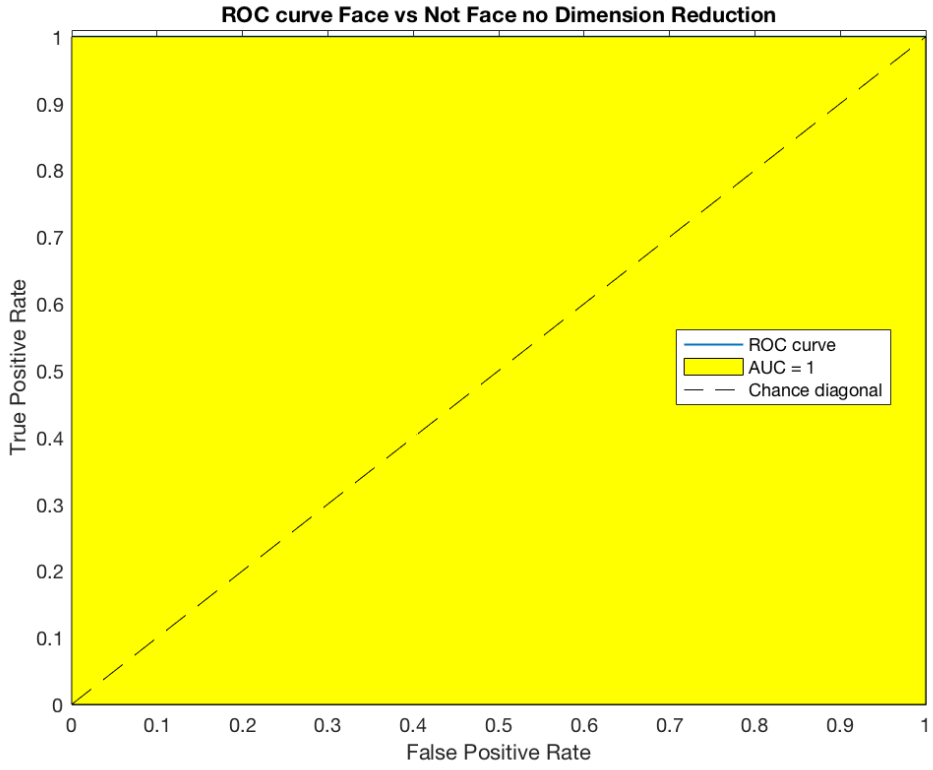

Figure 10: ROC curve for KNN with K=3 on faces versus non-faces without any dimension reduction

Confusion Matrix:

400 0

0 400

AUC: 1

Without any dimension reduction, the KNN classifier was able to successfully distinguish a face from not a face for all of the images. This result is not shocking because of the major differences in the not a face and face images. The next classification was carried out using the eigenface method to reduce the dimensions down to the first 8 principal components. The same 5-fold

cross validation and K=3 were used. Below is the accompanying ROC curve, confusion matrix, and AUC.

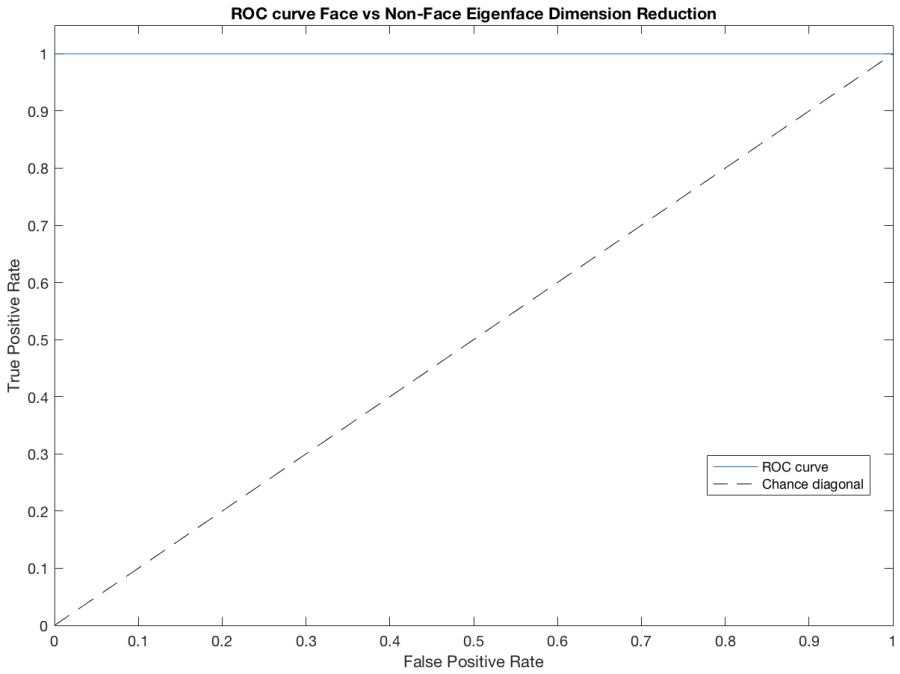

Figure 11: ROC curve for KNN with K=3 on faces versus non-faces with dimension reduction to the first 8 principle dimensions using PCA

Confusion Matrix:

399 1

0 400

AUC = 1

With the dimension reduction the KNN classifier work almost flawlessly, only incorrectly classifying one not a face as a face. This is a promising result because it means that this classification task can be completed with using a much lower dimensional set of data. Instead of having to store large amounts of data or dimensions for each of the images, a small subspace of

the original image can be used for the identification process. This will allow for more images to be processed in a shorter amount of time, without losing much accuracy in the process. Other values of K were explored and produced similar results. Below is a look at the confusion matrix and AUC that K = 5 and K = 1 produced.

K = 5 Confusion Matrix:

398 2

0 400

AUC: 1

K = 1 Confusion Matrix:

400 0

0 400

AUC: 1

For each of the different K values, they performed about the same, with K =1 having perfect performance on the given data. Each of the ROC curves looked the same as the ROC curve in Figure 11. Given the success when reduced down to 8 dimension, reduction down to 2 dimensions was also attempted. The same process of 5 fold cross validation and PCA dimension reduction were used. The K values used were 1,3,5,7. The ROC curve, confusion matrix and AUC can be seen below.

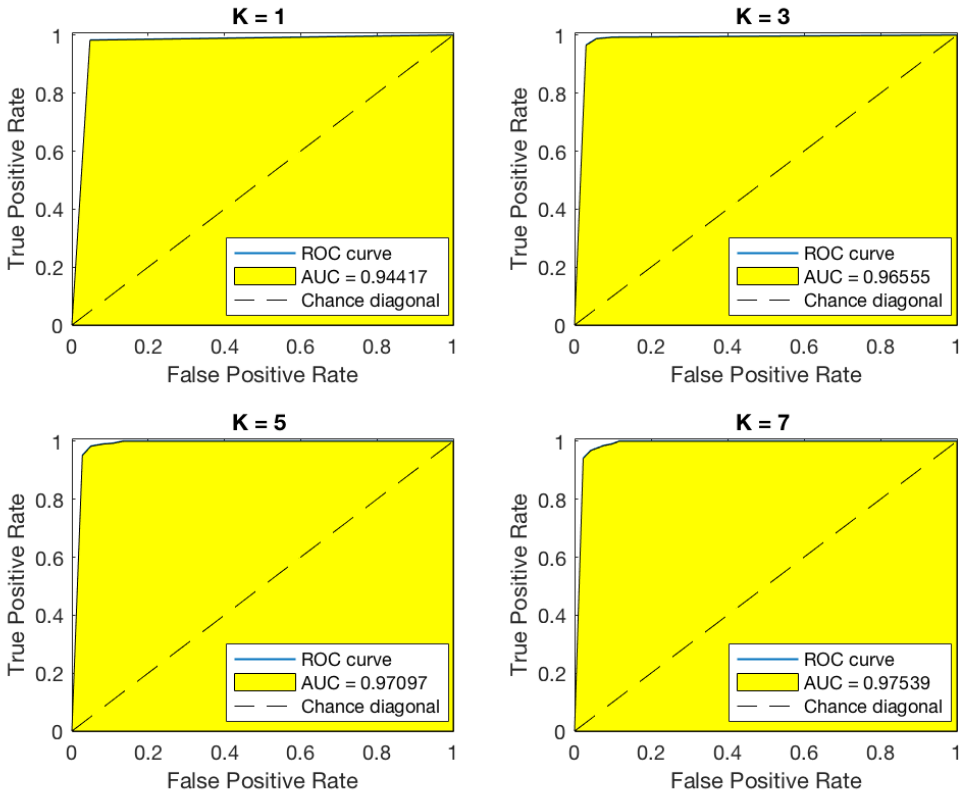

Figure 12: ROC curve for KNN with K=1,3,5,7 on faces versus non-faces with dimension reduction to the first 2 principle dimensions using PCA

Table 1: Confusion Matrices and AUC for the different K values

K Value	1	3	5	7
Confusion Matrix	381 19	377 23	367 33	361 39

	7 393	5 395	4 396	4 396
AUC	0.9442	0.9655	0.9710	0.9754

Overall with only 2 principal components used the classification works very well. There is a very high AUC value for each of the different K values, with K = 7 as the highest AUC. Overall, this performed slightly worse than the KNN with 8 principal components, but was able to reduce the amount of dimensions to ¼ the value. So depending on the needs of the program the decision can be made to reduce accuracy to reduce storage space of the data, or vice versa.

Classification Problem: Who Wears Glasses?

One of the most obvious distinguishing features between people in the dataset is glasses. The dataset consisted of an unbalanced number of photos of glasses-wearers and non-glasses wearers. Generating the vector of labels was very straightforward, so the binary classification problem seemed like a reasonable one. The labels for this problem were determined by going through the photos of the 40 people and marking them as non-glasses wearers ‘0’ or glasses wearers ‘1’.

The classification for this problem was first performed using K-NN on the data with no dimension reduction, on the data with dimension reduction using eigenfaces, and on dimension reduction using fisherfaces. The ROC curves and the corresponding confusion matrices are below.

No Dimension Reduction:

3-NN, 5 fold cross validation

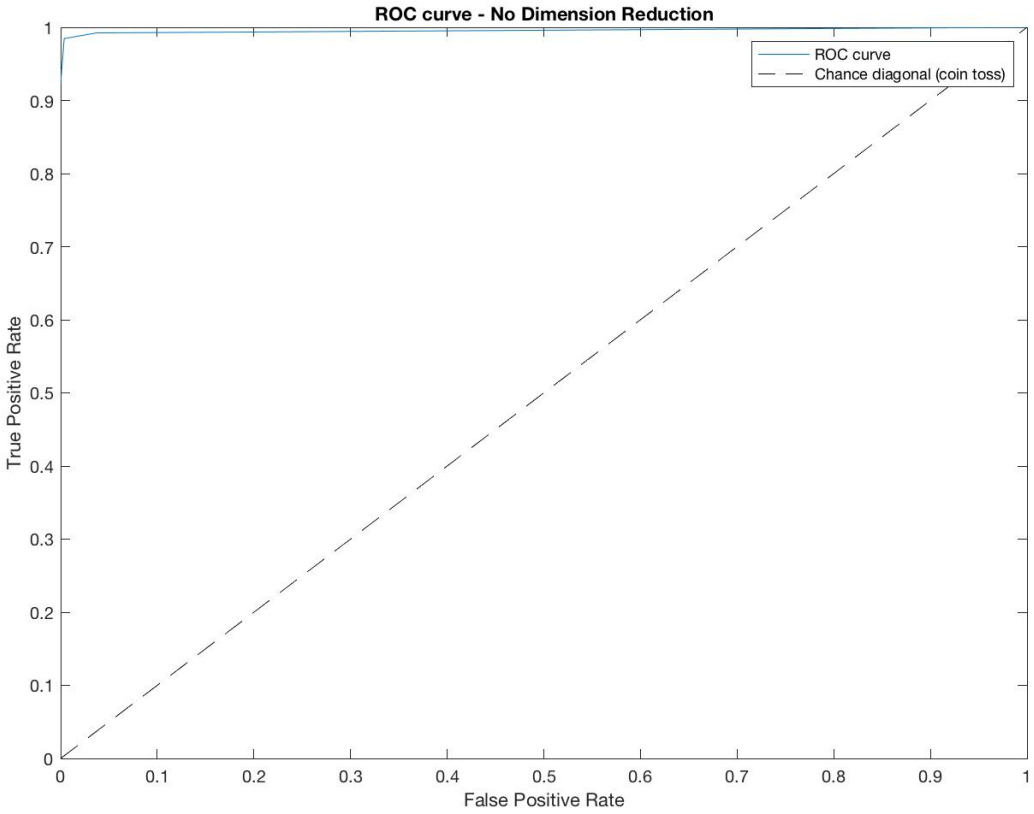

Figure 13: ROC curve for KNN with K=3 for glasses problem with no dimension reduction

Confusion Matrix =

269 1

2 128

AUC = 0.9957

Dimension Reduction:

1. Eigenfaces using 3-NN, 5 fold cross validation, 8 Principal Components

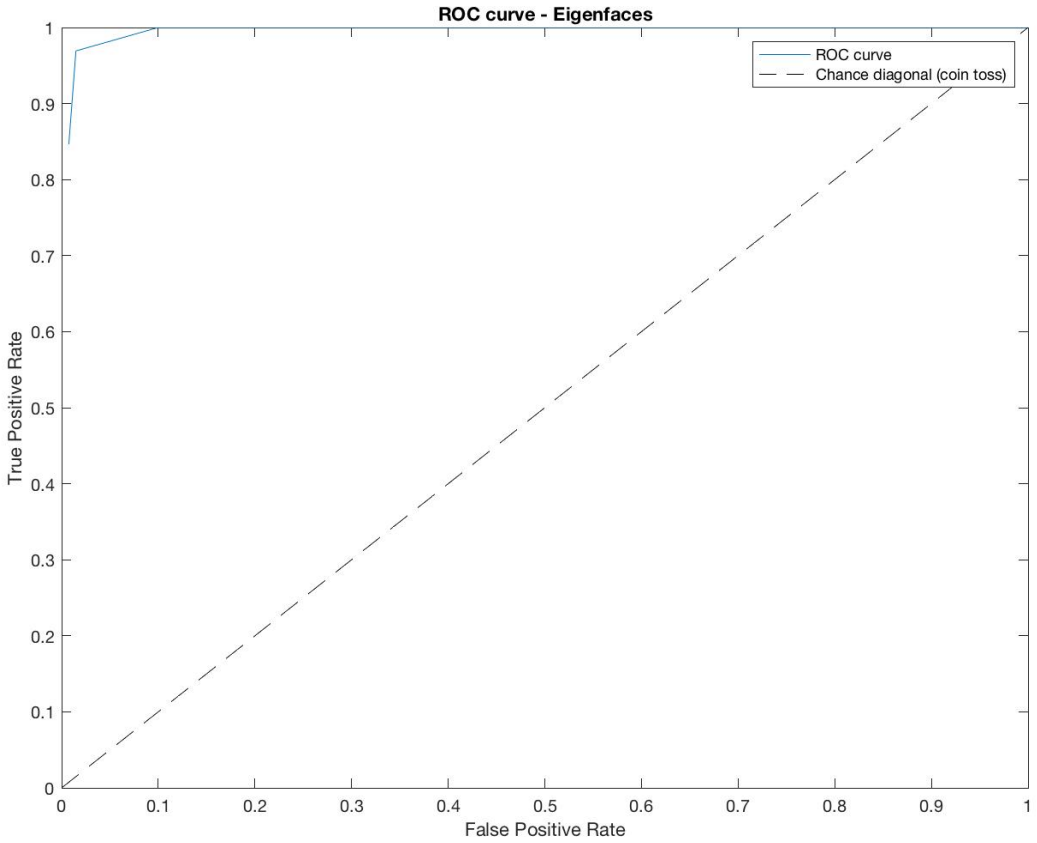

Figure 14: ROC curve for KNN with K=3 for glasses problem with dimension reduction to the first 8 principle dimensions using PCA

Confusion Matrix =

266 4

4 126

AUC = 0.9906

2. Fisherfaces using 3-NN, 5 fold cross validation, 8 Principal Components

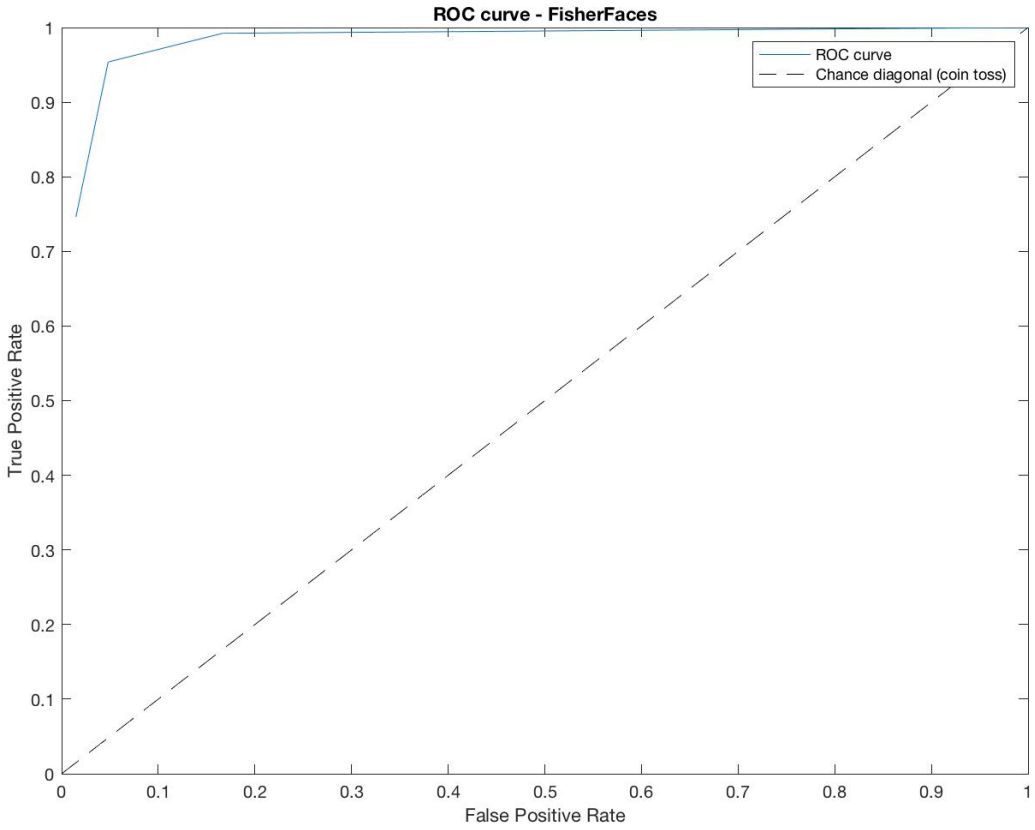

Figure 15: ROC curve for KNN with K=3 for glasses problem with dimension reduction to the first 8 principle dimensions using FLD

Confusion Matrix = 257 13

6 124

AUC = 0.9738

For this classification problem, we chose to project onto eight principal components. As explained in the dimension reduction section, this choice is supported by figure 3 which shows little increase in the AUC for more than eight principal components. The choice of nearest neighbors was a little bit trickier. One issue we had to take into account was the unbalance in each class components. The number of glasses wearers in the full data set is 13, compared to 27 non-glasses wearers. This meant that a large value for the number of nearest neighbors (K) would not be wise given the relative abundance of non-glasses wearers to glasses-wearers. We also chose to keep K an odd number in order to avoid "ties" which might occur with an even number of nearest neighbors; we would need to have some sort of rule to classify a point with half its nearest neighbors glasses-wearers and the other half non-glasses neighbors with an even-number K-NN. Since the improvement between K=1 and K=3 was significant enough to warrant using K=3, and the improvement between K=3 and K=5 not that drastic, K=3 was selected.

K-NN for the Glasses problem worked best without dimension reduction. However, the difference in the area under the ROC curve for the classification performed with the first eight principal components is only 0.005 less. Given that computation in lower dimensions is easier and not much is lost in terms of ability to classify in this lower dimension, dimension reduction can be appropriate in this situation. Classification using FLD actually performed worse than the classification with PCA alone despite the fact that FLD takes into account class-specific information.

Classification Problem: Man Or Woman?

Since the dataset consisted of photos of both men and women, the binary classification problem seemed like a reasonable one. The labels for this problem were determined by going through the photos of the 40 people and deciding as a team if each was a man ‘0’ or a woman ‘1’. Since this is information that would generally accompany a set of photographs in a database (ex. students, employees, citizens, criminals, etc.) we decided the classification problem was also a useful one.

The classification for this problem was first performed using K-NN on the data with no dimension reduction, on the data with dimension reduction using eigenfaces, and on dimension reduction using fisherfaces. The ROC curves and the corresponding confusion matrices are below.

No Dimension Reduction:

3-NN, 5 fold cross validation

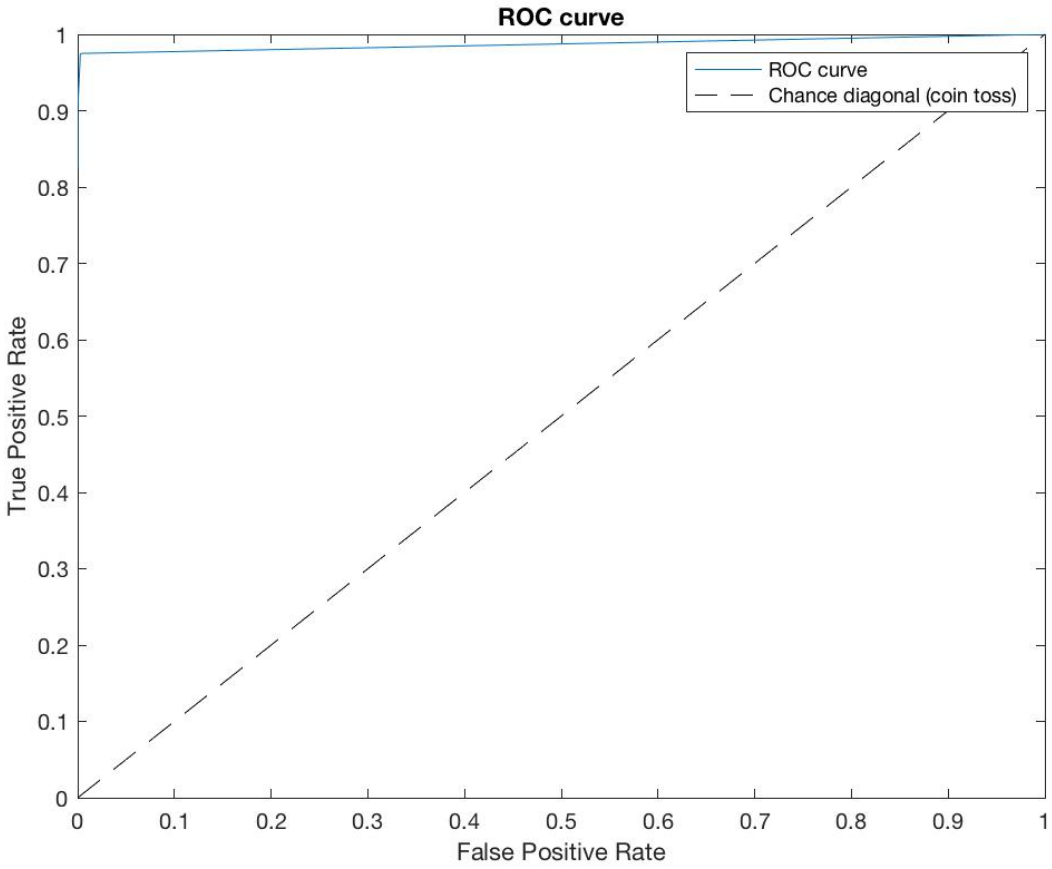

Figure 16: ROC curve for KNN with K=3 for man or woman problem with no dimension reduction

AUC = 0.9874

Confusion Matrix = 360 0

4 36

Dimension Reduction:

1. Eigenfaces: 3-NN, 5 fold cross validation, 4 Principal Components

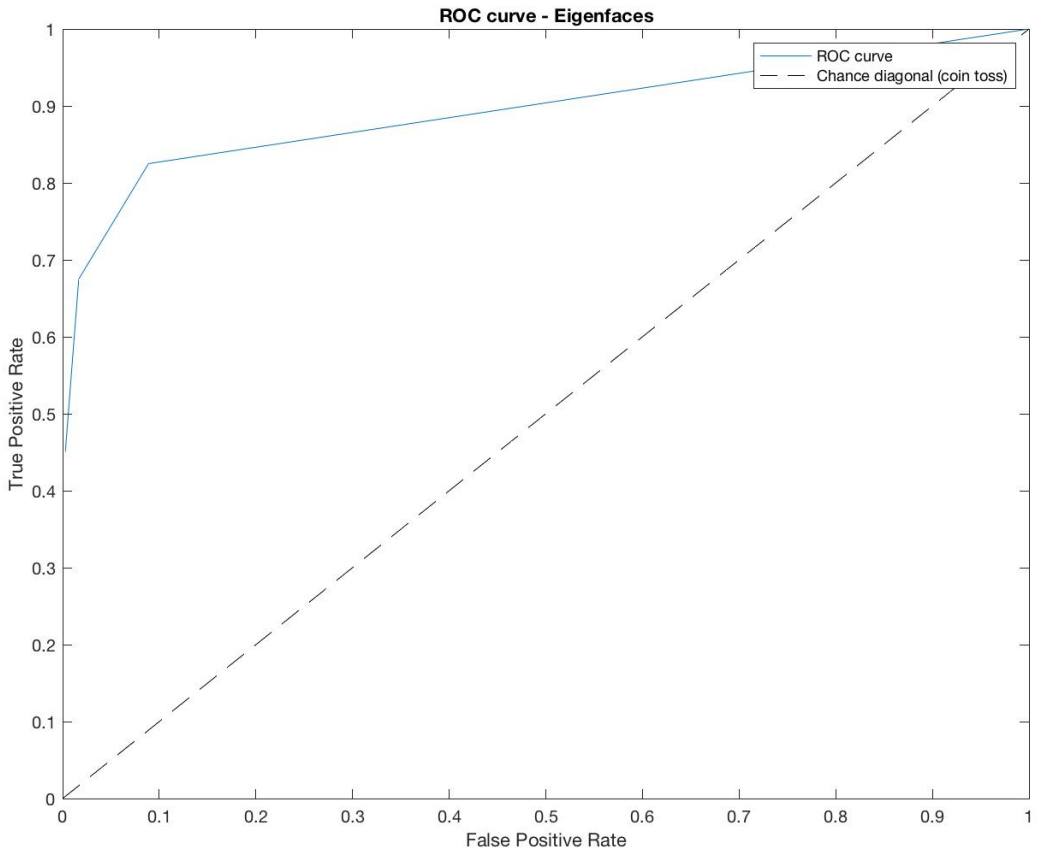

Figure 17: ROC curve for KNN with K=3 for glasses problem with dimension reduction to the first 4 principle dimensions using PCA

AUC = 0.8934

Confusion Matrix = 354 6

13 27

2. Fisherfaces: 3-NN, 5 fold cross validation, 4 Principal Components

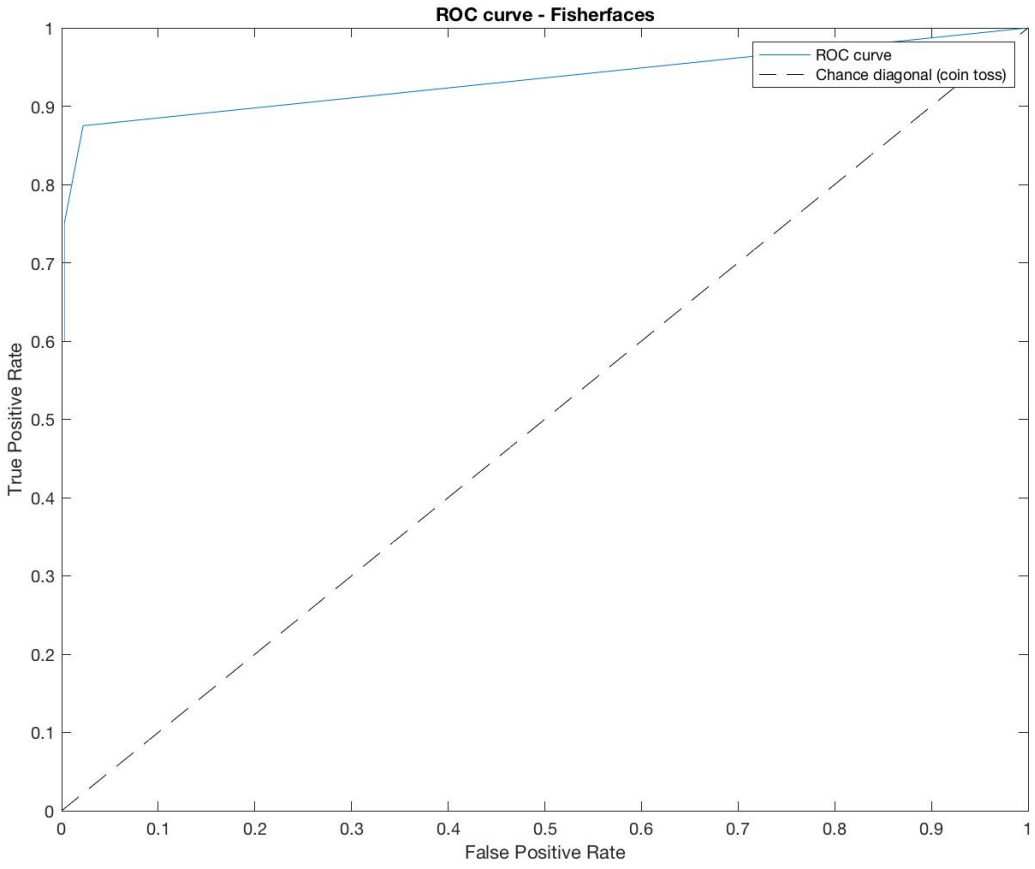

Figure 18: ROC curve for KNN with K=3 for glasses problem with dimension reduction to the first 4 principle dimensions using FLD

AUC = 0.9325

Confusion Matrix = 359 1

10 30

For this classification problem, we chose to project onto eight principal components. As explained in the dimension reduction section, there was little increase in the AUC for more than a few principal components. Similar to the Glasses Classification Problem, the choice of nearest

neighbors was a little bit trickier. One issue we had to take into account was the imbalance in class components. Here, this imbalance is even greater than it was for the Glasses problem. The number of women in the full data set is only 4, compared to 36 men. This meant that a large value for the number of nearest neighbors (K) would not be wise given the relative abundance of men to women in the data. We also chose to keep K an odd number in order to avoid "ties" which might occur with an even number of nearest neighbors; we would need to have some sort of rule to classify a point with half its nearest neighbors women and the other half men with an even-number K. K=3 was found to be optimal.

Performing classification on the data using K-NN before dimension reduction still produced the largest area under the curve at 0.9874 just as it did for the glasses classification problem. The next best AUC was given by classification using FLD, with an AUC of 0.9325 which, while notably lower the AUC with no dimension reduction, is significantly greater than the AUC achieved for classification using PCA which was 0.8934. As noted above, for the glasses classification problem, PCA alone performed better than FLD. This is most likely a result of the differences between the two classification problems. It would be reasonable to say that the differences between the men's and women's faces in this dataset are probably more subtle than the differences between a face with glasses and a face not wearing glasses. In this classification problem where differences are less obvious, the fact that FLD makes use of the labels is quite a significant advantage for the purposes of classification. Based on the results of the glasses classification problem and the man/women classification problem, the choice between using FLD or PCA alone for dimension reduction depends on the classification problem at hand.

Fisherface Results for KNN:

While we expected the fisherface method to produce better results than the eigenface method given that the fisherfaces use class-specific information, the results indicated that the depending on the classification problem at hand and using KNN as the classifier, the eigenface method could produce better results for classification than the fisherface method did, as in the case of the glasses problem, or the fisherface method could produce better results for classification than the eigenface method did, as in the case of the man/woman problem. One reason for this could be that performing PCA prior to performing FLD might cause the data to lose some of the distinguishing between-class variation. Another important point to take into account is that our methods operated under the assumption that the best eigenvectors will be those corresponding to the largest eigenvalues. These largest n eigenvectors may not be the best for our classification problem, as there may be some other combination of eigenvectors that form a better basis for dimension reduction with the goal of classification. For this reason, we later decided to look at search methods for finding the best combination of eigenvectors for dimension reduction. Overall, for the KNN-classifier, the highest dimensional data still produced the best classification results, with the highest AUC.

KNN Classification with Multiple Labels

The problem where multiple labels for classification could be identifying suspect A or B from a group of images with some containing faces, and some not containing faces. The four labels would be if the image contains suspect A (0), contains suspect B (1), contains suspect C (2), or contains a face that may later be a person of interest (3). To design the problem with the given faces, the first person was decided as suspect A, the second person was decided as suspect B, the third person was designated suspect C, and the rest of the people were not a suspect.

To perform KNN classification with multiple faces, the same general technique of finding the K closest neighbors and applying a decision rule to decide on a label. In this case, the label that had the highest number of neighbors closest to the test data point was chosen as the predicted label. In cases of ties, the lowest label was chosen. Below is a table of how KNN with varying K values (1,2,3,4) performed. The lower K values were chosen because there was not much training data to be able to use to classify the suspects. The following table shows the K values, and their associated percentages at correctly identifying if the face was suspect A, B, C, or not a suspect. 5 fold cross validation was used and the first 8 principal components were generated from PCA and then FLD.

Table 2: Comparison of Percent Correct for different dimension reduction techniques and different K values

K Value	1	2	3	4
PCA Percent Correct (%)	96.25	97.00	96.25	96.25
FLD Percent Correct (%)	91.50	85.75	92.25	91.00

Overall the classifier worked well choosing the correct labels for the given test image. Further inspection on the PCA and FLD values (K=2, K = 3 respectively) will be carried out. The confusion matrices with the predicted label being the row number and the column number being the actual label were generated for the K's with the highest percent correct for both PCA and FLD. The confusion matrices for the PCA with K = 2 and FLD with K = 3 are below.

Confusion Matrix PCA

9	0	0	2
0	10	0	3
0	0	8	4
1	0	2	361

Confusion Matrix FLD

2	0	0	2
0	4	0	2
0	0	2	5
8	6	8	361

For the PCA confusion matrix it shows that the classifier correctly identified the actual suspect 90% of the time and that it falsely accused someone of being a suspect 2.4% of the time. It did not incorrectly label a suspect as another suspect. The PDA confusion matrix shows that the

KNN classifier correctly identified the actual suspect 26.7% of the time and falsely accused someone of being a suspect 2.4% of the time. It also did not incorrectly label a suspect as another suspect. From this further analysis the PCA method does a very good job of identifying the suspects correctly and reducing the amount of false alarms. However, PDA does a horrible job of detecting a suspect, only getting it correct about a quarter of the time. The percent correct statistic from above is skewed by the amount of non suspect faces in the training data, especially for the PDA method. While it looks like both classifier could work well within the given task, upon further inspection the PDA method with K = 3 would not be recommended to use. The PCA method with K = 1 works well and if it was given more data, it would be able to better identify the suspect and reduce the amount of false alarms.

This method of classification could be improved by being given more training data. Also further testing could be done to alleviate the issue of a tie, by either assigning a higher priority to labels that are closer to the test point or by looking as the next closest neighbor until the tie is broken. The first method of weighting the labels due to their distance would probably result in better classification since it would prioritize the closer labels. The second method could actually produce worse results, especially if the data set is unevenly distributed. Often times as more labels are taken into consideration, the not suspect label with be the next closest one because there are more of those training points than the different suspect training points.

LDA Classifier

The LDA (Linear Discriminant Analysis) Classifier works essentially by generating a linear equation that divides two classes within a dataset. With two dimensional data, for simplicity, this can be visualized as a line that attempts to cut the dataset along the line that best divides the data into its differently labeled parts. In three dimensions, the dataset would be cut with a plane, and so on as the dimensionality increases. This linear equation is generated using the means, variances, and priors of the two classes of data, and it assumes that the variance of the classes is the same.

Classification Problem: Face versus Not a Face

First, LDA was used to solve the problem of distinguishing whether an image was of a face or not, also described above in the KNN section. The datasets used for these categories each comprised 400 images. Faces were given a label of 0 and non-faces were given a 1. The first ROC curve below is for the first 300 principal components used, which is approximately as much as the LDA function we used would support; larger numbers of features led to errors and warnings about singular/badly scaled matrices. (This is, however, reflective of the difficulty and potential inaccuracy of classifying in higher dimensions, something that this project is intended to demonstrate.)

Less Dimension Reduction:

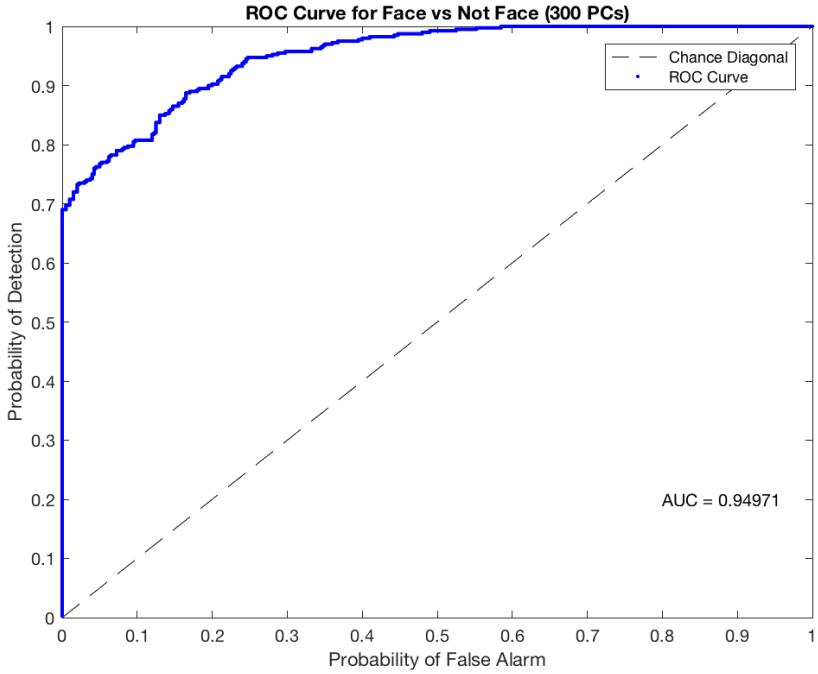

Figure 19: ROC curve for LDA for face problem with dimension reduction to the first 300 principle dimensions using PCA

LDA classifies faces and not-faces fairly well with the higher-dimensional data, as demonstrated by the high AUC of 0.94971. It seems that there is a sweet spot of sorts in terms of dimensionality that works best with LDA, as seen in the later classifications for this problem. Below are two ROC curves that use the only first 20 and the first 8 principal components, respectively.

Large Dimension Reduction:

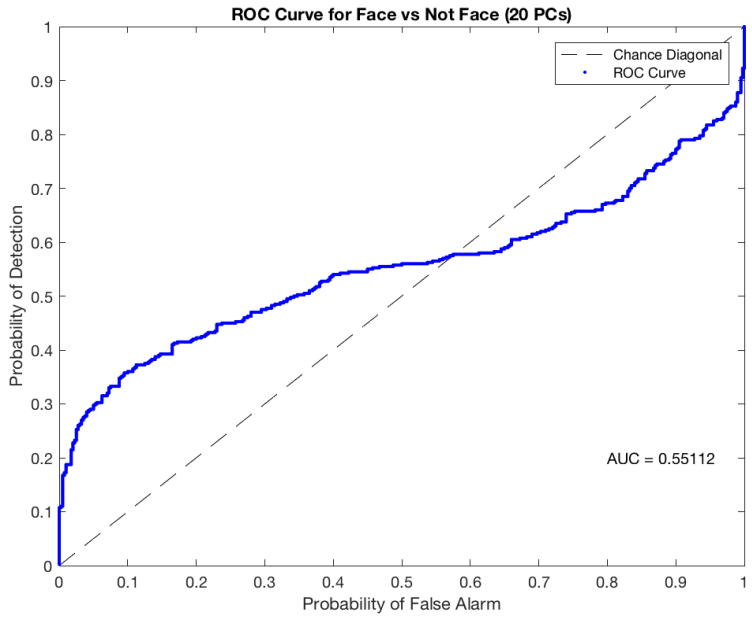

Figure 20: ROC curve for LDA for face problem with dimension reduction to the first 20 principle dimensions using PCA

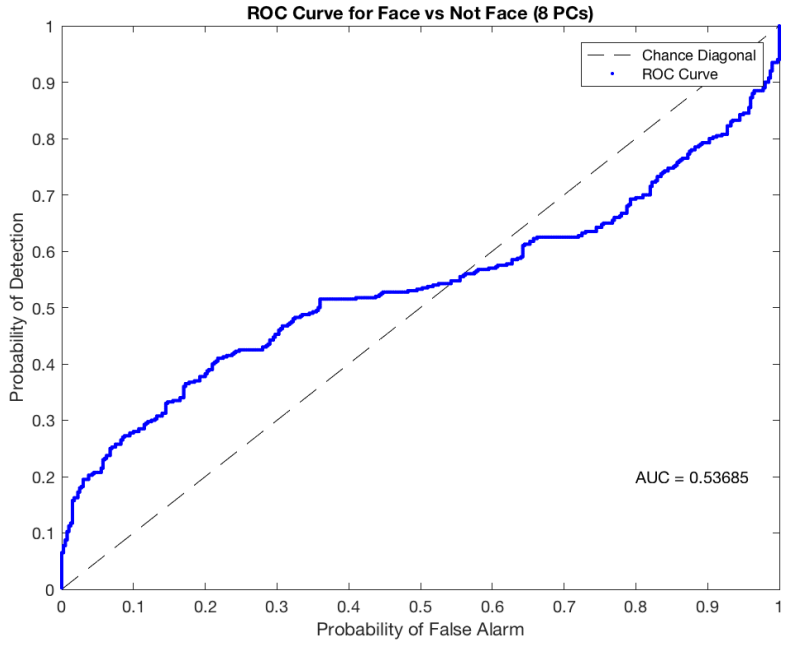

Figure 21: ROC curve for LDA for face problem with dimension reduction to the first 8 principle dimensions using PCA

The lower-dimensional classification of faces and not-faces with LDA does significantly worse than the classification in higher dimensions, with AUCs of 0.55112 and 0.53685. This is potentially due to the fact that the variances of the not-faces are different enough from those of the faces that smaller numbers of principal components cannot account for them as well. The figure below offers some insight into this, showing that the two largest variances in the different images tend to occur in different dimensions. This is a plausible hypothesis because LDA becomes less accurate if the variances of the two classes are different, because the calculation of the dividing line or plane makes the assumption that the variances are the same.

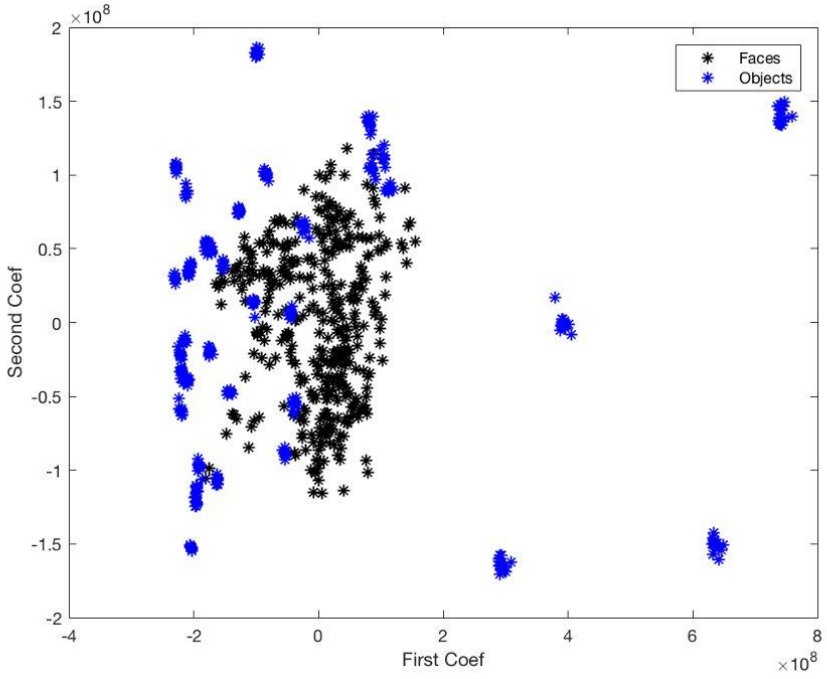

Figure 22: Plot of the coefficients of the first and second principal components

Classification Problem: Who Wears Glasses?

LDA was then used to distinguish between faces wearing glasses and faces that were not. There was an uneven number of the two classes of data in this classification problem. Glasses wearers were given a label of 1 and non-glasses wearers were given a 0. The first ROC curve below is again for the first 300 principal components that were used.

Less Dimension Reduction:

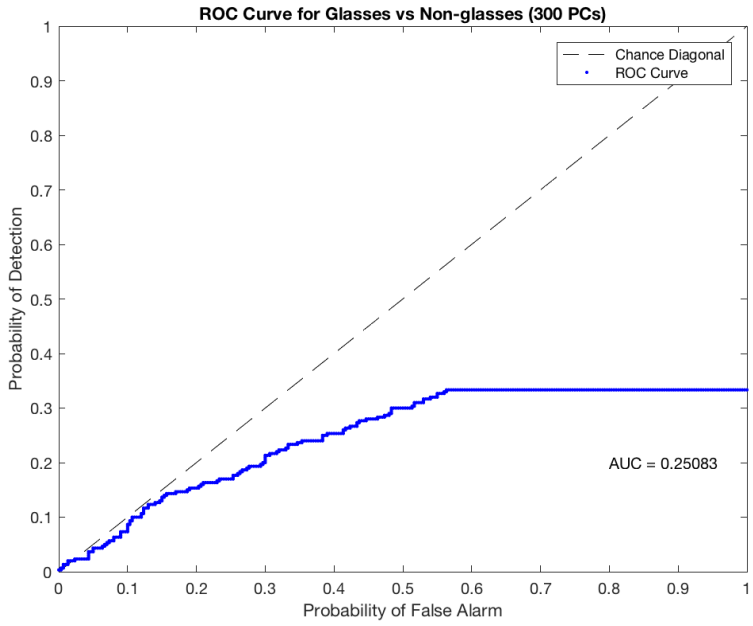

Figure 23: ROC curve for LDA for glasses problem with dimension reduction to the first 300 principle dimensions using PCA

Large Dimension Reduction:

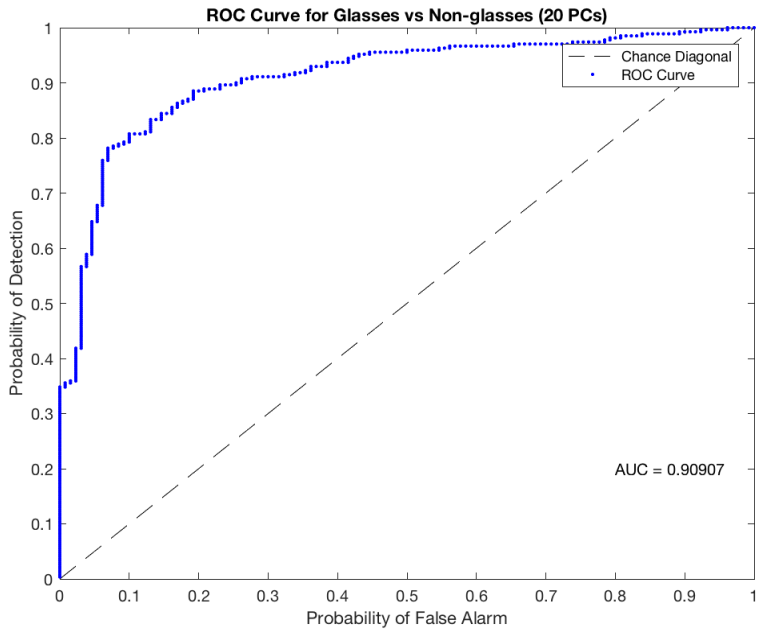

Figure 24: ROC curve for LDA for glasses problem with dimension reduction to the first 20 principle dimensions using PCA

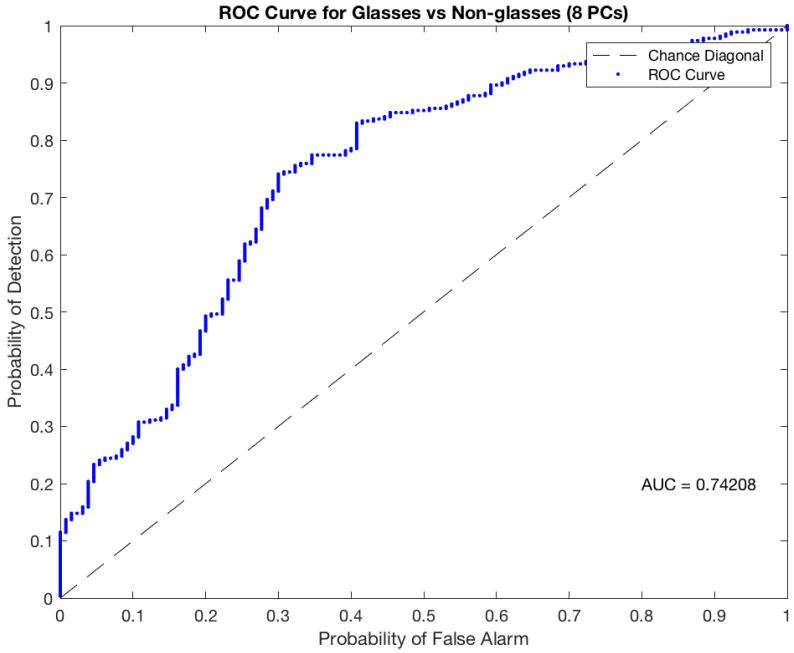

Figure 25: ROC curve for LDA for glasses problem with dimension reduction to the first 8 principle dimensions using PCA

Clearly, LDA performs far worse with 300 principal components for the glasses problem than it does for the face vs. not face problem, with a positively dismal AUC of 0.25083, while the lower-dimensional ROC curves have better AUCs of 0.90907 and 0.74208. One possible reason for this is that in the face vs. not a face problem, you are (obviously) comparing things that are faces with things that aren't faces. Thus, most of the differences between these images and the variation in the images is likely due to the fact that they are pictures of completely different things. On the other hand, in the glasses problem, faces are being compared to other faces, so a good deal of the variation is probably due to things like differences in lighting and face angle, and not necessarily because the picture is of a different face. Thus, when LDA is used on high dimensional face vs. not face data, it appropriately uses the differences in variation to classify the images, but when it is used on high dimensional glasses vs. no glasses data, the variation accounted for by the many principal components skews the results because it does not reflect a difference in the identity (or, in this case, glasses-wearing) of the face. This also explains the more accurate classification in lower dimensions, because the classifier is not overwhelmed by the meaningless data in higher dimensions and can instead use the relevant data from the most significant principal components. The fact that 20 dimensional data fares better than the 8 dimensional data supports the idea that there is a sweet spot of dimensionality where you have enough information to accurately classify, but not so much information that the data is meaningless. The same pattern in the ROC curves can also be seen in the man vs. woman problem below.

Classification Problem: Man Or Woman?

Lastly, we used LDA to determine whether the subject in the image was a man or a woman. This was the most unbalanced of the datasets, with 4 out of the 40 subjects being women. Men were given a label of 0, and women were given a label of 1. The first ROC curve below is again for the first 300 principal components that were used.

Less Dimension Reduction:

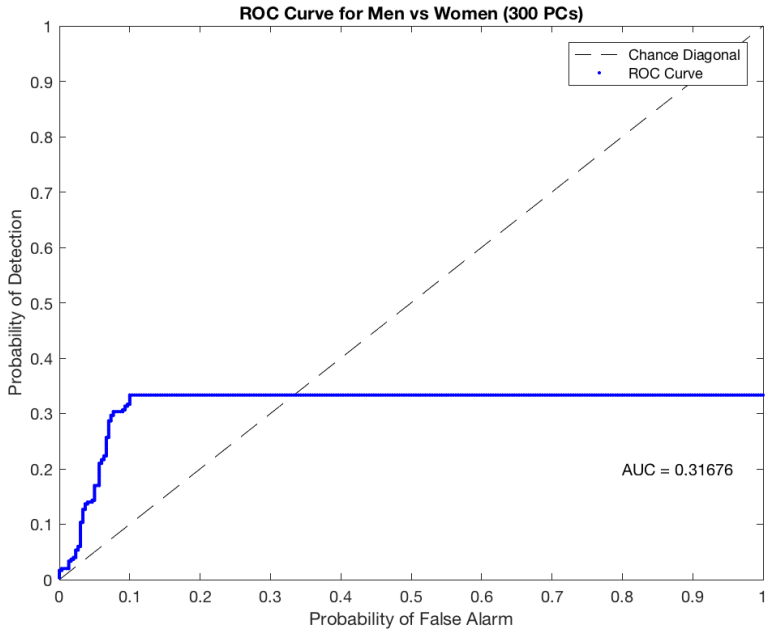

Figure 26: ROC curve for LDA for man or woman problem with dimension reduction to the first 300 principle dimensions using PCA

Large Dimension Reduction:

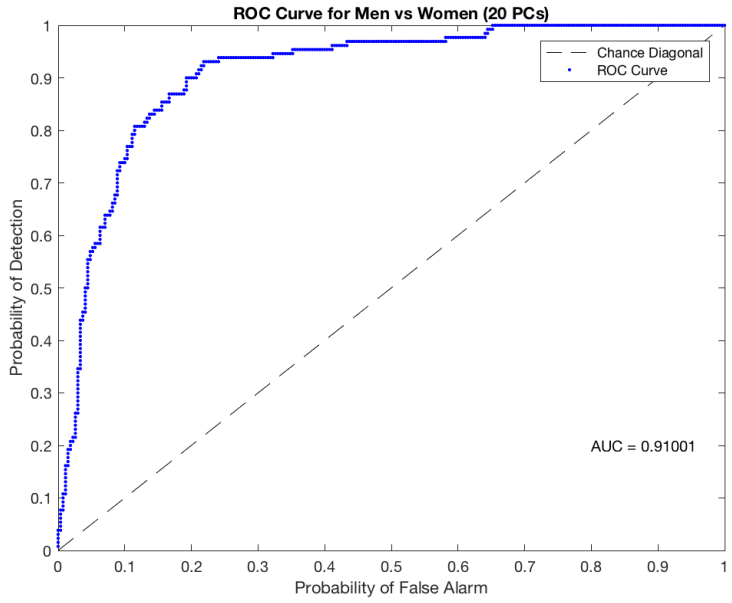

Figure 27: ROC curve for LDA for man or woman problem with dimension reduction to the first 20 principle dimensions using PCA

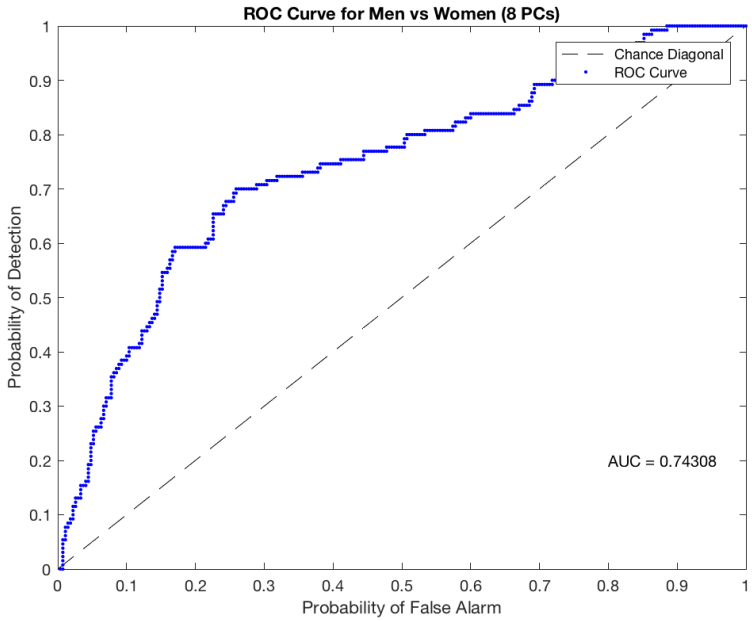

Figure 28: ROC curve for LDA for man or woman problem with dimension reduction to the first 8 principle dimensions using PCA

A very similar pattern to the one seen in the glasses problem is also visible here. As discussed above in the glasses problem, the poor performance in high dimensions and the better performance in lower dimensions is likely due to the irrelevance of the less significant principal components because faces are being compared with other faces, not different objects.

Conclusions

This project gave us a deeper understanding behind the same classification tools we had been using for most of the semester, as well as introduced to us a new concept in the form of dimension reduction. It was interesting to see how classification results changed in lower and higher dimensions given by principal component analysis. After viewing the results of normal classification, we extended our analyses in different directions by reconsidering dimension reduction. We were able to compare the results gathered by normal principal component analysis to those gathered by Fisher's Linear Discriminant. In addition, we improved our KNN classifier to account for multiple classes and learned how to select the best principal components to project onto given a classification problem.

Overall, we are grateful for the insight we gathered on modern facial recognition techniques and how the concepts we learned in class could be extended to some interesting real-world examples. This project has encouraged us to do some more independent research to further satisfy our curiosities.

References

1. IEEE Transactions On Pattern Analysis And Machine Intelligence, Vol. 19, No. 7, July 1997, and 711. “Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection.” *Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection*.
2. “Linear Discriminant Analysis.” *Statistical Pattern Recognition*, pp. 123–168., doi:10.1002/0470854774.ch4.
3. “Log-Likelihood-Ratio-Based Detection Ordering in V-BLAST.” *IEEE Transactions on Communications*, vol. 54, no. 1, 2006, pp. 179–179., doi:10.1109/tcomm.2005.861635.
4. Mai, Qing. “A Review of Discriminant Analysis in High Dimensions.” *Wiley Interdisciplinary Reviews: Computational Statistics*, Wiley-Blackwell, 9 Apr. 2013, onlinelibrary.wiley.com/doi/pdf/10.1002/wics.1257.
5. “Turbo Pascal Code for Optical Character Recognition Shell.” *Neural Network PC Tools*, 1990, pp. 367–373., doi:10.1016/b978-0-12-228640-7.50025-8.
6. “A Tutorial on Combinator Graph Reduction.” *An Architecture for Combinator Graph Reduction*, 1990, pp. 101–122., doi:10.1016/b978-0-12-419240-9.50015-x.
7. Wagner, Philipp. “Face Recognition with GNU Octave/MATLAB.”

Appendix

PCAFaces.m

```
clear;
load('not_faces.mat');
not_faces2 = not_faces;
load('faces.mat');
load('not_faces2.mat');

for k = 1:400
Test = faces(:,:,k);
TestT = Test';
TestAll(:,k) = TestT(:);

end
NotFaces = not_faces';
NotFaces2 = not_faces2';

% Mean center data
TestMean = TestAll - mean(mean(TestAll));
NotMean = NotFaces - mean(mean(NotFaces));
NotCov = NotMean'*NotMean;
NotMean2 = NotFaces2 - mean(mean(NotFaces2));
NotCov2 = NotMean2'*NotMean2;
Cov = TestMean'*TestMean;
[coeffs, scores] = pca(Cov);
[NotCo, NotSc] = pca(NotCov);
[NotCo2, NotSc2] = pca(NotCov2);
figure(1);clf;
plot(coeffs(:,1), coeffs(:,2), 'k*');
hold on
plot(NotCo(:,1), NotCo(:,2),'b*');
hold off
xlabel('First Coef');
ylabel('Second Coef');
legend('Faces','Objects')

labels = zeros(400,1);
labels2 = ones(400,1);

labels = [labels; labels2];
Data = [coeffs(:,1:2); NotCo(:,1:2)];
%Data = coeffs(:,1:4);

%labels(1:5:end) = [];

%% Knn to ROC curve
% requires all the code from the KNN lab
```

```
% arg1 = data
% arg2 = labels
% arg3 = number of nearest neighbors (K)
% arg4 = number of folds
% arg5 = figure number
for g = 1:4
    [X_ROC,Y_ROC,AUC(g)] = data_to_ROC(Data,labels,2*g-1,5,g);
    confusion_matrix = data_to_confusion_mat(Data,labels,2*g-1,5)
end
figure(7);clf;
x = [1,3,5,7];
bar(x,AUC)
xlabel('K value (Number of Nearest Neighbors)')
ylabel('AUC')
title('AUC for Different K Values')

%% Attempt to do more than 2 labels
load('FisherFaceScores.mat');
scorelda = scorelda';

NumDifLab = 4;
% D is the amount of Dimensions you want to use
D = 2;
% S is the starting point for the components
S = 1;
data = scores(:,S:S+D-1);
%lab = [ones(300,1); zeros(100,1); zeros(400,1) + 2];
lab = [zeros(10,1); zeros(10,1)+1;zeros(10,1)+2; zeros(370,1)+3];
for fold = 1:5
    set = [];
    %data = [data; NotCo(:, S:S+D-1)];
    for k = 1:40
        set = [set, (k-1)*10 + 2*(fold)-1,(k-1)*10 + 2*(fold)];
    end
    TrainData = data;
    FinalTrainData = [];
    for l = 1:D
        Temp = TrainData(:,l);
        Temp(set) = [];
        FinalTrainData = [FinalTrainData Temp];
    end
    TrainData = FinalTrainData;
    Trainlab = lab;
    Trainlab(set) = [];
    TestData = data(set,1:D);
    TestLab = lab(set,1);

    [predictions_test, K_nearest_indices,scoresKnn] = knn_from_scratch2...
        (TrainData,TestData,Trainlab,2);
```

```
    OnesCor = find(TestLab == predictions_test);
    PerCor(fold) = length(OnesCor)/length(TestLab);

    for p = 1:NumDifLab
        for l = 1 : NumDifLab
            Pred = find(predictions_test == p-1);
            Act = find(TestLab == l-1);
            Cor = intersect(Pred, Act);
            ConMat(p,l, fold) = length(Cor);
        end
    end
end
ConMat = sum(ConMat,3);
PerCorrect = mean(PerCor);
%% Plotting
% 108
% 137

figure(6);clf;
hold on
for p = 1:NumDifLab
    rng(231+p);
    Color = randi([0 250],1,3);
    Color = Color./255;
      WhatToPlotTrain = find(Trainlab == NumDifLab - p);
    WhatToPlotTest = find(predictions_test == NumDifLab - p);
    plot(TrainData(WhatToPlotTrain,1),TrainData(WhatToPlotTrain,2),'wo',...
        'MarkerFaceColor',Color);
    plot(TestData(WhatToPlotTest,1),TestData(WhatToPlotTest,2),'kd',...
        'MarkerFaceColor',Color);
end
legend('Train 3', 'Predict 3', 'Train 2', 'Predict 2', 'Train 1',...
    'Predict 1', 'Train 0', 'Predict 0', 'Location', 'Best')
hold off

% figure(4);clf;

% plot3(NotSc(:,1), NotSc(:,2), NotSc(:,3),'b*');
```

Knn_from_scratch2.m

```
function [predictions_test, K_nearest_indices,scores] =
knn_from_scratch2(data_train,data_test,labels_train,K)

% Author: Serge Assaad, Sep 23, 2017
% Assumptions:
```

```
% - training and test data stored in matrices with samples as rows
% - labels_test is a column vector
% - L2 norm for distance
if(~iscolumn(labels_train))
    labels_train = labels_train';
end

M = size(data_test,1);

distances = calc_distance_matrix(data_train,data_test);

[~, sort_indices] = sort(distances);
labels_train_matrix = labels_train*ones(1,M);
labels_train_sorted = labels_train_matrix(sort_indices);
K_nearest_indices = sort_indices(1:K,:);
K_nearest_labels = labels_train_sorted(1:K,:);
predictions_test = mode(K_nearest_labels,1)';
for l = 1:length(K_nearest_labels)
    labs = K_nearest_labels(:,l);
    numSame = length(find(labs == predictions_test(l)));
    scores(l) = numSame/K;
end
end

function distances_squared = calc_distance_matrix(data_train,data_test)
N = size(data_train,1);
M = size(data_test,1);

data_train_norm = diag(data_train*data_train');
data_test_norm = diag(data_test*data_test');

term1 = data_train_norm*ones(1,M);
term2 = data_test_norm*ones(1,N);
term3 = -2*data_train*data_test';

distances_squared = term1+term2'+term3;
end
```

Forwardsearch.m

```
function colvec = forwardsearch(nComp, scores, labels, K, fold, fignum,
numcomponents)
nComp = 8;
colvec = [1];
j = 1:50;
for i = 1:(numcomponents-1) %number of principal components to add
    maxauc = -1;
```

```
        for k = j %finds maximum AUC from remaining PC's
            [X_ROC,Y_ROC,AUC] = data_to_ROC(scores(:,[colvec
k]),labels,K,fold,fignum);
            if (AUC > maxauc)
                maxauc = AUC;
                nextindex = k;
            end
        end
        colvec = [colvec nextindex];
        j = j(j~=nextindex);
end
end
```

letstryLDA.m

```
clear;
load('faces.mat');
load('not_faces.mat');
load('not_faces2.mat');
nComp = 20;

%% making large data matrix of True Face Data (400, 10k)
Matrix = [];
%make limit=400 for full matrix
limit = 400;
for r = 1:limit
    newmat = faces(:,:,r);
    newvec = newmat(:)';
    Matrix(r,:) = newvec;
end
dimension = limit;

%Define my labels
labels = ones(10,1);
for cnt = 2:limit/10
    labels = [labels cnt*ones(10,1)];
end

%% run fisherfaces
[W, scorelda, D] = fisherfaces(Matrix ,labels, nComp);

%% let's examine our LDAScores
figure(5); clf;
hold on
%scoreslda is 20*400 meaning columns correspond to people and their photos
plot(scorelda(1,1:10),scorelda(2,1:10), '.', 'MarkerSize', 20)%person 1
plot(scorelda(1,11:20),scorelda(2,131:140), '.', 'MarkerSize', 20)%person 2
plot(scorelda(1,21:30),scorelda(2,21:30), '.', 'MarkerSize', 20)%person 3
plot(scorelda(1,31:40),scorelda(2,111:120), '.', 'MarkerSize', 20)%person 4
plot(scorelda(1,41:50),scorelda(2,141:150), '.', 'MarkerSize', 20)%person 5
```

```
hold off
%% POV Plot
POVk = [0]; %want first data point of (0,0) for visual satisfaction
sumL = sum(D)
eee = size(D,1)
for i= 1:eee
     sss = sum(D(1:i, :));
     POVk = [POVk sss./sumL];
 end

%Plot
figure(2); clf;
hold on
plot(linspace(0,eee,eee+1), POVk)
xlabel('lambda')
ylabel('POV(k)')
title('Plot of Percentage of Variance Explained by First K Principle
components')
hold off
```

fisherfaces.m

```
function [W, scorelda, D] = fisherfaces (X ,y ,nComp)
%This function will output fisherfaces W, scores (scorelda), and
%eigenvalues (D)

% number of samples
N = size(X);
N = N(:,1);

% number of classes
uniqlabels = unique(y);
numlabels = length(uniqlabels);

% get principal components
[eigpca, scores] = pca(X);
%
% %project onto princip components
Xhat = scores(:,1:nComp)*eigpca(:,1:nComp)';
%call LDA
%Choice to run 'lda' on: X, Xhat, pcascore
[Eiglda , scorelda, D] = lda(scores(:,1:nComp), y, nComp);
W = Eiglda;
end
```

lda.m

```
function [W , scorelda, D ] = lda (X ,y, ncomp)
%This function does FLD
% dimensions
[n, d] = size(X);
```

```
% find how many classes we have
labels = unique(y) ;
C = length(labels);
% scatter matrices
Sw = zeros(d,d);
Sb = zeros(d,d);
% total mean
mu = mean(X);
% calculate scatter matrices, for each class C
for i = 1: C
    Xi = X(find(y==labels(i)),:); % samples for desired class
    [n cols]=size(Xi);
    mu_i = mean(Xi); % mean of desired class
    %Xi = Xi - repmat(mu_i, n ,1); %changing mean
    Xi = bsxfun(@minus,Xi,mu_i);
    %Sw id within class scatter
    Sw = Sw + cov(Xi);
    %Sb is between class scatter
    prod = (mu_i-mu)'*(mu_i-mu);
    Sb = Sb + n.*prod;
end

% invswsb = inv(Sw)*Sb;
% % solve general eigenvalue problem
% [W, sc] = pca(invswsb);

% solve general eigenvalue problem
[W , D] = eig( Sb , Sw );
% sort eigenvectors
[D , i] = sort ( diag (D) , 'descend');
AllW = W (: , i );
% keep desired eigenvectors
W = AllW(: ,1:ncomp);
%A short note on matrix dimensions:
%find corresponding scores to W eigenvectors we want to keep wrt X
%originally 400*10k is X, we projected it onto PC first time to get
%400*nComp X (from scorespca). Then we get that W is nCOmp*nComp
%So  dimensions should line up
scorelda = (W'*W)\(W*X');

end
```

WhoHasGlasses.m

```
clear;
load('faces.mat');
load('not_faces.mat');
load('not_faces2.mat');

%I = mat2gray(faces(:,:,400));
```

```
%%show a face
%figure(1); clf;
%imshow(I);

%say indexing starts at 1. We have person 1-40
%Person 11-20 -> Person 2
%Person 31-40  -> Person 4
%person 51-60  -> Person 6
%person 121-130  -> Person 13
%person 131-140  -> Person 14
%person 161-170  -> Person 17
%person 181-190  -> Person 19
%person 191-200  -> Person 20
%person 261-270  -> Person 27
%person 271-280  -> Person 28
%person 301-310  -> Person 31
%person 331-340  -> Person 34
%person 361-370  -> Person 37

limit = 400;
%making large data matrix (400, 10k)
Matrix = [];
for r = 1:limit
    newmat = faces(:,:,r);
    newvec = newmat(:)';
    Matrix(r,:) = newvec;
end
dimension = 400;

%Define my labels: zero is no glasses, 1 is glasses
labels = zeros(400,1);
labels(11:20) = 1;
labels(31:40) = 1;
labels(51:60) = 1;
labels(121:130) = 1;
labels(131:140) = 1;
labels(161:170) = 1;
labels(181:190) = 1;
labels(191:200) = 1;
labels(261:270) = 1;
labels(271:280) = 1;
labels(301:310) = 1;
labels(331:340) = 1;
labels(361:370) = 1;

%find indexes of people with and without glasses
a = find(~labels)
b = find(labels==1)
%PCA - eigenfaces
```

```
[eigenvectors, scores] = pca(Matrix);

%Do a little plotting
figure(5); clf;
hold on
plot(scores(b,1),scores(b,2), '.', 'MarkerSize', 20)%Glasses 1
plot(scores(a,1),scores(a,2), '.', 'MarkerSize', 20) %No Glasses 0
hold off

%% LDA - fisherfaces
allpcas=8;
[W, scorelda, D] = fisherfaces(Matrix ,labels, allpcas);

%Do a little plotting
%fig 6 is 2 first LDA pC's
figure(6); clf;
hold on
plot(scorelda(1,b),scorelda(2,b), '.', 'MarkerSize', 20)%Glasses 1
plot(scorelda(1,a),scorelda(2,a), '.', 'MarkerSize', 20) %No Glasses 0
hold off

%% Run the Classifier of our choosing on pca
K=3;
fold = 5;
fignum = 1;
nComp = 8;
[ out ] = CrossValidateAndCLassify(scores(:,1:nComp),labels,fold );

%confusion matrix
confusion_matrix = data_to_confusion_mat(scores(:,1:nComp),labels,K,fold)

[X_ROC,Y_ROC,AUC] = data_to_ROC(scores(:,1:nComp),labels,K,fold,fignum);
Auc = AUC

%% Run Classifier of our choosing on fisherstuff
K=3;
fold = 5;
fignum = 2;
scoreldat = scorelda';

[ out ] = CrossValidateAndCLassify(scoreldat(:,1:nComp),labels,fold );

%confusion matrix
confusion_matrix = data_to_confusion_mat(scoreldat(:,1:nComp),labels,K,fold)

[X_ROC,Y_ROC,AUC] = data_to_ROC(scoreldat(:,1:nComp),labels,K,fold,fignum);
Auc2 = AUC
%% Run it Upstairs
K=3;
```

```
fold = 5;
fignum = 3;

[ out ] = CrossValidateAndCLassify(Matrix,labels,fold );

%confusion matrix
confusion_matrix = data_to_confusion_mat(Matrix,labels,K,fold)

[X_ROC,Y_ROC,AUC] = data_to_ROC(Matrix,labels,K,fold,fignum);
Auc = AUC
```

ManOrWoman.m

```
%Women Indices
%71-80
%91-100
%311-320
%341-350

%long hair dude: 111-120
clear;
load('faces.mat');
load('not_faces.mat');
load('not_faces2.mat');

limit = 400;
%making large data matrix (400, 10k)
Matrix = [];
for r = 1:limit
    newmat = faces(:,:,r);
    newvec = newmat(:)';
    Matrix(r,:) = newvec;
end
dimension = 400;

%Define my labels: zero is no glasses, 1 is glasses
labels = zeros(400,1);
labels(71:80) = 1;
labels(91:100) = 1;
labels(311:320) = 1;
labels(341:350) = 1;
%labels(111:120) = 1; %long hair boy

a = find(~labels);
b = find(labels==1);
%PCA - eigenfaces
[eigenvectors, scores] = pca(Matrix);

%% Do a little plotting
%fig 5 is 2 first PC's
```

```
figure(5); clf;
hold on
plot(scores(b,1),scores(b,2), '.', 'MarkerSize', 20) %Women 1
plot(scores(a,1),scores(a,2), '.', 'MarkerSize', 20) %Man 0
hold off

%% LDA - fisherfaces
nComp=8;
[W, scorelda, D] = fisherfaces(Matrix ,labels, nComp);

%Do a little plotting
%fig 6 is 2 first LDA pC's
figure(6); clf;
hold on
plot(scorelda(1,b),scorelda(2,b), '.', 'MarkerSize', 20) %Women 1
plot(scorelda(1,a),scorelda(2,a), '.', 'MarkerSize', 20) %Man 0
hold off

%% Run the Classifier of our choosing on pca

fold = 5;
fignum = 2;

[ out ] = CrossValidateAndCLassify(scores(:,1:nComp),labels,fold );

%confusion matrix
K=3;
confusion_matrix = data_to_confusion_mat(scores(:,1:nComp),labels,K,fold)

[X_ROC,Y_ROC,AUC] = data_to_ROC(scores(:,1:nComp),labels,K,fold,fignum);
Auc = AUC

%% Run Classifier of our choosing on fisherstuff
fold = 5;
fignum = 1;
scoreldat = scorelda';

[ out ] = CrossValidateAndCLassify(scoreldat(:,1:nComp),labels,fold );

%confusion matrix
confusion_matrix = data_to_confusion_mat(scoreldat(:,1:nComp),labels,K,fold)

[X_ROC,Y_ROC,AUC] = data_to_ROC(scoreldat(:,1:nComp),labels,K,fold,fignum);
Auc2 = AUC
%% Run it Upstairs

fold = 5;
fignum = 3;
```

```
[ out ] = CrossValidateAndCLassify(Matrix,labels,fold );

%confusion matrix
confusion_matrix = data_to_confusion_mat(Matrix,labels,K,fold)

[X_ROC,Y_ROC,AUC] = data_to_ROC(Matrix,labels,K,fold,fignum);
Auc = AUC
```

CrossValidateAndCLassify.m

```
function [ out ] = CrossValidateAndCLassify( data,labels, folds )
% Run KNN Classification with Cross Validation
%   Detailed explanation goes here
%5 folds
N=size(data,1)

keys = ones(N,1);
j=1;
while (j<N+1)
    f=1;
    while (f<folds)
        keys(j+(f)) = f+1;
        f=f+1;
    end
     j=j+folds;
end
keys = keys((1:N),:);
%make a keys vector length of x. goes 12345..n12345...n in col vector

K = folds; %num folds
Y = labels;
X = data;
lambda = nan(N,1);
for i=1:K
    Xtest = X(keys==i, :);
    Xtrain = X(keys~=i,:);
    Ytrain = Y(keys~=i, :);
    %% Create the model and generate lambdas

    [predictions_test, K_nearest_indices,scores] =
knn_from_scratch(Xtrain,Xtest,Ytrain,K);

% %     B = mnrfit(Xtrain, Ytrain+1);
% %     E = (B')*(vals');
% %     G = E./(1+E)
     lambda(keys==i, :) = predictions_test;
end
out = lambda;
```