Implementing Various Classifiers to Achieve Facial Recognition

Omar H. Abdelkader[1]

[1]University of Maryland, College Park
Electrical and Computer Engineering – A.V. Williams Building
8223 Paint Branch Drive – College Park, MD – 20740

oabdelka@umd.edu

Abstract. *The purpose of this technical report is to describe the performance of various classifier implementations in the context of facial recognition. In this report, the naive Bayes' and K-nearest neighbors classifiers are considered. In addition, Principal Component Analysis (PCA) and Fisher's Multiple Discriminant Analysis (MDA) are evaluated as dimension reduction techniques. The aforementioned methods have been tested on three different datasets, including images with various pose, expression, and illumination variations.*
Keywords*: Bayesian decision theory, maximum likelihood, k-nearest neighbors, PCA, MDA.*

1. Introduction

Thus far, we have covered two classification methodologies. In the *bottom-up* approach, the training data is first fit to some known distribution using a procedure such as maximum likelihood estimation. The discriminant functions are computed from the distribution, to which the testing data can then be classified. The Bayesian classifier, otherwise known as the minimum error rate classifier, best falls into this category. Conversely, the *top-down* approach begins with choosing a classifier geometry, and aims to fit the data to that geometry afterwords. The top-down approach lends itself nicely to data that does not appear to fit a known distribution, but still has an intuitive and clear separation between the classes. The K-Nearest Neighbor Rule is a non-parametric classifier that is often used in these cases.

In addition to classification techniques, we have also examined the importance of dimension reduction. In some cases, many features can be overlooked while still maintaining the accuracy of a particular classifier. Dimension reduction techniques are vital for both reducing computation time and reducing the number of samples required to create a suitable classifier. The first technique covered in this course was Principal Component Analysis. PCA works to represent the given data in a lower dimension space such that the representation captures as much energy of the data as possible. In some cases, though, projecting onto a lower dimension space produced by PCA destroys the class separation. Fisher's Multiple Discriminant Analysis aims to fix this flaw by maximizing between class scatter in conjunction with minimizing within class scatter.

2. Theoretical Framework

In this project, I examined three `.mat` datasets. The *face* dataset is composed of cropped images of 200 subjects, three images each. The first image is a neutral face, the second

has a facial expression, and the third has illumination variations. The *pose* dataset is composed of cropped images of 68 subjects under 13 different poses, and the *illumination* dataset is composed of cropped images of 68 subjects under different 21 illuminations.

Table 1. Dataset Breakdown

	Num. of Classes	*Samples per Class*	*Image Size*
Face	200	3	24 x 21
Pose	68	13	48 x 40
Illumination	68	21	48 x 40

The number of subjects in each dataset can be thought of as the number of classes. In addition, since each pixel is considered to be a unique feature, the image size directly corresponds to the number of dimensions for the given model. Two photos from the *pose* dataset are presented below. The images are of the same subject, but under different poses.

Figure 1. Sample 1, Pose 1

Figure 2. Sample 1, Pose 2

3. Methodology

Prior to conducting any analysis or manipulation of the data, I first reshaped the image matrices into feature vectors. Fortunately, this is trivial to do in MATLAB, and the *illumination* dataset is already provided in vector form. Once reshaped, each dataset was divided into two disjoint sets, one for training, and another for testing. For the purposes of this paper, I divided the *face* dataset into $\frac{2}{3}$ training to $\frac{1}{3}$ testing, and the *pose* and *illumination* datasets to $\frac{3}{4}$ training to $\frac{1}{4}$ testing.

3.1. Bayes' Classifier

In order to correctly use the Bayes' classifier, the data must first be fit to a known distribution. Fortunately, in this project, we are told that the underlying distribution of the images is Gaussian. Estimating the Gaussian parameters $\hat{\mu}$ and $\hat{\Sigma}$ is trivial, as we have done the maximum likelihood estimation derivation for the multivariate Gaussian in class.

$$\hat{\mu} = \frac{1}{n}\sum_{k=1}^{n}\vec{x}_k \tag{1}$$

$$\hat{\Sigma} = \frac{1}{n}\sum_{k=1}^{n}(\vec{x}_k - \hat{\mu})(\vec{x}_k - \hat{\mu})^T \tag{2}$$

With $\hat{\mu}$ and $\hat{\Sigma}$ from equations 1 and 2, computing the posterior probability should be simple. Since the prior and evidence probabilities are equal across all classes, it is safe to compute and classify using class conditional probability values.

$$P(\omega_i|\vec{x}) = \frac{1}{(2\pi)^{\frac{d}{2}}|\hat{\Sigma}_i|^{\frac{1}{2}}} e^{-\frac{1}{2}(\vec{x}-\hat{\mu}_i)^T\hat{\Sigma}_i^{-1}(\vec{x}-\hat{\mu}_i)} \tag{3}$$

In general, any positive monotonic transformation of the discriminant function will classify exactly the same as the discriminant itself. So, to simplify even further, we can take the natural log of the class conditional probability and rearrange the terms. The boxed term in equation 4 below can be dropped because it is constant for all i.

$$\begin{aligned} g_i(\vec{x}) &= \ln P(\omega_i|\vec{x}) \\ &= -\frac{1}{2}(\vec{x}-\hat{\mu}_i)^T\hat{\Sigma}_i^{-1}(\vec{x}-\hat{\mu}_i) - \boxed{\frac{d}{2}\ln 2\pi} - \frac{1}{2}\ln|\Sigma_i| \qquad (4) \\ &= -\frac{1}{2}(\vec{x}^T\hat{\Sigma}_i^{-1}\vec{x} - 2\hat{\mu}_i^T\hat{\Sigma}_i^{-1}\vec{x} + \hat{\mu}_i^T\hat{\Sigma}_i^{-1}\hat{\mu}_i) - \frac{1}{2}\ln|\hat{\Sigma}_i| \\ &= \vec{x}^T(-\frac{1}{2}\hat{\Sigma}_i^{-1})\vec{x} + \hat{\mu}_i^T\hat{\Sigma}_i^{-1}\vec{x} + (-\frac{1}{2}\hat{\mu}_i^T\hat{\Sigma}_i^{-1}\hat{\mu}_i - \frac{1}{2}\ln|\hat{\Sigma}_i|) \\ &= \vec{x}^T W_i\vec{x} + w_i^T\vec{x} + w_{0i} \qquad (5) \end{aligned}$$

In this case, however, computing $g_i(\vec{x})$ is not as trivial plugging in and evaluating equation 5. For any $\vec{x} \in \mathbb{R}^d$, where d is greater than the number of samples for a given class, n, the estimate of the covariance matrix $\hat{\Sigma}$ will always be singular. This means $\hat{\Sigma}$ is non-invertable and has a determinant of zero.

Examining the procedure for computing $\hat{\Sigma}$ reveals why this happens. The multiplication of a vector with its transpose will always be a matrix of rank one because the columns of the resulting matrix are scaled versions of the original vector. For some vector, $\vec{x}$

$$\vec{x}\cdot\vec{x}^T = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \begin{bmatrix} x_1 & x_2 & \dots & x_d \end{bmatrix} = \begin{bmatrix} x_1x_1 & x_2x_1 & \dots & x_dx_1 \\ x_1x_2 & x_2x_2 & \dots & x_dx_2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1x_d & x_2x_d & \dots & x_dx_d \end{bmatrix}$$

Adding n matrices of this form will increase the rank one by one up to n, however if $n < d$, there will not be enough pivot columns in the end $d \times d$ matrix. Taking the inverse of $\hat{\Sigma}$ is what is known as an *ill-posed* inference problem. A common fix for regularizing problems of this kind is to use shrinkage estimation, whereby a naive or raw estimate is improved by combining it with other information. In this case, $\hat{\Sigma}$ becomes positive-definite by adding a small value to all the diagonal elements, namely the identity matrix.

3.2. K-Nearest Neighbors

Unlike the Bayesian classifier, the K-Nearest Neighbors classifier is non-parametric. The algorithm for classifying according to the k-nearest neighbors rule is as follows

Data: Gather n labeled samples $\{x_1, \ldots, x_n\}$ and choose a tiebreaker mode
for *a new sample* $\vec{x}$ **do**
    Find k-nearest neighbors, $\vec{x}'_1, \ldots, \vec{x}'_k \in \{\vec{x}_1, \ldots, \vec{x}_n\}$, closest to $\vec{x}$;
    Assign $\vec{x}$ label of majority of it's k-nearest neighbors;
    if *multiple k-nearest neighbors* **then**
        Use the tiebreaking mode provided;
    end
end

Algorithm 1: K-Nearest Neighbors Algorithm

In the event of a tie (i.e. two or more classes have the same number of closest neighbors to the sample), a tiebreaking mode of the user's choice is used to classify. The following tiebreaking methods are investigated in this report.

Table 2. K-Nearest Neighbors Tiebreaking Methods

Tiebreaker Mode	*Description*
Discard	Samples with a tie are discarded
Retry	Repeat with K-1 neighbors until the tie is broken
Closest	Of the tied classes, select the one with the minimum total distance
Random	Of the tied classes, select one randomly

3.3. Principal Component Analysis

Principal Component Analysis is a technique for reducing a given dataset to a lower dimension. PCA works by projecting the original data onto a subset of the feature vectors, known as the principal components. In order to compute which directions to project onto, one computes the scatter matrix of the centered data and solves for its eigenvalues and eigenvectors. By projecting onto the eigenvectors corresponding to the largest eigenvalues, we are guaranteed to maintain as much of the original data's variance as possible. The extent to which how much data is eliminated is determined by how much energy the operator is willing to sacrifice. This value is expressed by the parameter, α.

For example, let d be the number of features for a particular dataset. Let m be the number of features we wish to reduce our data to, where $m < d$. Then, the energy preserved by projecting onto the eigenvectors corresponding to the m largest eigenvalues can be expressed as R_m

$$R_m = \frac{\sum_{i=1}^{m} \lambda_i}{\sum_{i=1}^{d} \lambda_i} \tag{6}$$

$$= \frac{\lambda_1 + \lambda_2 + \cdots + \lambda_m}{\lambda_1 + \lambda_2 + \cdots + \lambda_d} = 1 - \alpha$$

It is important to note that the correct way to use PCA is to run it on the training set, save resulting the principal components, and then use them to transform the testing set. This

is the only way to guarantee that both sets end up in the same space without using any knowledge about the test set during training.

3.4. Fisher's Multiple Discriminant Analysis

A major shortcoming of PCA is that it does not preserve between-class scatter very well. In other words, samples that that appear to belong to distinct classes in a higher dimension space appear to belong to the same class when projected onto a lower dimension space. Fisher's Linear Discriminant Analysis projects a 2-class dataset of d dimensions onto a line for which the projected samples are well separated.

Generalizing this procedure to c classes is known as Multiple Discriminant Analysis (MDA). Naturally, this involves $c-1$ discriminant functions; thus, the projection is from a d-dimensional space to a $(c-1)$-dimensional space. We seek a transformation matrix $\mathbf{W}$ that maximizes the ratio of the between-class scatter to the within-class scatter. Using this measure, we define a criterion function, $J(\mathbf{W})$

$$J(\mathbf{W}) = \frac{|\mathbf{W}^T \mathbf{S}_W \mathbf{W}|}{|\mathbf{W}^T \mathbf{S}_B \mathbf{W}|} \tag{7}$$

The columns of an optimal $\mathbf{W}$ that maximize equation 7 are the generalized eigenvectors corresponding to the $(c-1)$-largest eigenvalues in

$$\mathbf{S}_W w_i = \lambda_i \mathbf{S}_W w_i \tag{8}$$

Just as in PCA, MDA should be run solely on the training data. The transformation matrix that is generated can then be used to reduce the testing data to $c-1$ classes.

4. Results

I divided each of the three datasets into the following training and testing sets according to the ratios defined in Section 3.

Table 3. Training-Testing Split

	Training Samples	*Testing Samples*
Face	{1, 2}	{3}
Pose	{1-10}	{11-13}
Illumination	{1-16}	{17-21}

4.1. Bayes' Classifier

As expected, out of the three datasets considered, the Bayesian classifier performed poorest on the *face* dataset. Due to the fact that there were only two training samples available for every class, I did not have very high expectations for any of the classifiers to perform well on this datset. Despite this, it is quite astonishing that the Bayesian classifier managed to correctly label 64% of the testing data. On average, randomly assigning a label

of the 200 possible classes would have yielded 0.5% accuracy. The fact that the Bayesian classifier performed 128 times better than randomly guessing on a dataset with only two training samples per class really speaks to the power of its classification ability.

Table 4. Bayesian Classifier Performance

	Correct Classifications	*Misclassifications*	*Accuracy*
Face	128	72	64%
Pose	139	65	68.1%
Illumination	340	0	100%

The same explanation cannot be made for the results from the *pose* dataset. In spite of having 10 training samples per class, the Bayesian classifier performed nearly just as poorly as on the *face* dataset. On the other hand, the Bayesian classifier correctly classified all of the testing samples in the *illumination* dataset.

According to these results, the Bayesian classifier is certainly illumination invariant, but the same cannot be said for pose variations. Perhaps a dataset with more samples would yield results more indicative of the classifier's performance. The covariance matrices in all three datasets had to be regularized in order to be used in any computation, meaning that the data was not well-formed for this kind of problem.

4.2. K-Nearest Neighbors

For all three datasets, the K-Nearest Neighbors classifier performed equal-to, or worse than the Bayesian classifier. For $K = 1$, the *face* and *pose* datasets were 4.5% and 6.3% less accurate compared to the Bayesian classifier, respectively. The nearest neighbor classifier, however, was just as accurate for the *illumination* dataset, correctly classifying every sample in the test set.

Due to the limited number of samples, both per class and overall, I only varied K from one to five. Intuitively, it would make sense that corroborating the label assignment against more neighbors from the training set would improve the labeling accuracy of the model. Surprisingly, however, increasing K seemed to hurt the overall precision of the model.

Despite the Bayesian classifier being more accurate across all the datasets, it is worth mentioning the price paid in execution time to achieve the modest increase in accuracy. Take the *illumination* testing set, for example. It took 293.87 seconds to compute the Bayesian classifier predictions, whereas it took only 4.94 seconds to compute the K-Nearest Neighbor predictions ($K = 1$). For high dimension data, taking the inverse and determinant of the covariance matrix can be quite expensive. As a result, sacrificing a few percentage points in accuracy could be worth using a non-parametric classifier such as K-Nearest Neighbors.

4.2.1. Tiebreaker Strategies

The `discard` tiebreaker strategy is not a very viable classification technique in practice. This is because samples in the testing set with numerous nearest neighbors end up unla-

beled. As a result, when overlooking unclassified samples, this adaptation of the nearest neighbors has high accuracy. It performs rather poorly, however, if the unclassified samples are counted as misclassifications. In the tables below, I chose to report the number of discarded samples as opposed to the accuracy of the `discard` method.

The `retry` and `closest` tiebreaking implementations produced nearly identical results of one another. Samples that tie in the $K = 2$ variant are always classified as they would have been had $K = 1$. As a result, the $K = 2$ accuracy is equivalent to the $K = 1$ accuracy for both tiebreaking modes. In addition, these modes saw the least precipitous drop in accuracy as K is increased. This outcome is likely due to the fact that these two strategies are the only ones that use information about the initial run's nearest neighbors when attempting to reclassify.

Table 5. Face Dataset Results

`Discard`

	Correct	*Incorrect*	*Discarded*
K = 1	119	81	0
K = 2	53	3	144
K = 3	67	8	125
K = 4	68	19	113
K = 5	69	28	103

`Retry`

	Correct	*Incorrect*	*Accuracy*
K = 1	119	81	59.5%
K = 2	119	81	59.5%
K = 3	123	77	61.5%
K = 4	120	80	60%
K = 5	119	81	59.5%

`Closest`

	Correct	*Incorrect*	*Accuracy*
K = 1	119	81	59.5%
K = 2	119	81	59.5%
K = 3	123	77	61.5%
K = 4	120	80	60%
K = 5	119	81	59.5%

`Random`

	Correct	*Incorrect*	*Accuracy*
K = 1	119	81	59.5%
K = 2	98	102	49%
K = 3	94	106	47%
K = 4	91	109	45.5%
K = 5	91	109	45.5%

The final tiebreaking strategy, `random`, was initially implemented in anticipation of the other strategies requiring much more execution time to arrive at a result. As it so happens, however, benchmarking the strategies showed no discernible reduction in wait time when using `random` over `retry` or `closest`. Of course, it is important to note that this behavior can be partially attributed to the unusually small size of the datasets being used.

In practice, where the goal is to correctly classify each sample in the testing set as best as possible, a strategy like `retry` or `closest` would most likely be used to break a tie. Because `closest` outperformed `retry` in my results, albeit somewhat marginally, I chose to use `closest` for the remaining computations pertaining to K-Nearest-Neighbors that do not explicitly state a specific tiebreaker mode for the rest of the report.

Table 6. Pose Dataset Results

`Discard`

	Correct	*Incorrect*	*Discarded*
K = 1	126	78	0
K = 2	29	4	171
K = 3	46	22	136
K = 4	48	40	116
K = 5	56	50	98

`Retry`

	Correct	*Incorrect*	*Accuracy*
K = 1	126	78	61.8%
K = 2	126	78	61.8%
K = 3	120	84	58.8%
K = 4	114	90	55.9%
K = 5	110	94	53.9%

`Closest`

	Correct	*Incorrect*	*Accuracy*
K = 1	126	78	61.8%
K = 2	126	78	61.8%
K = 3	120	84	58.8%
K = 4	115	89	56.3%
K = 5	111	93	54.4%

`Random`

	Correct	*Incorrect*	*Accuracy*
K = 1	126	78	61.8%
K = 2	87	117	42.7%
K = 3	74	130	36.3%
K = 4	71	133	34.8%
K = 5	72	132	35.3%

Table 7. Illumination Dataset Results

`Discard`

	Correct	*Incorrect*	*Discarded*
K = 1	340	0	0
K = 2	287	0	53
K = 3	290	7	43
K = 4	265	11	64
K = 5	250	17	73

`Retry`

	Correct	*Incorrect*	*Accuracy*
K = 1	340	0	100%
K = 2	340	0	100%
K = 3	333	7	97.9%
K = 4	327	13	96.2%
K = 5	316	24	92.9%

`Closest`

	Correct	*Incorrect*	*Accuracy*
K = 1	340	0	100%
K = 2	340	0	100%
K = 3	333	7	97.9%
K = 4	329	11	96.8%
K = 5	322	18	94.7%

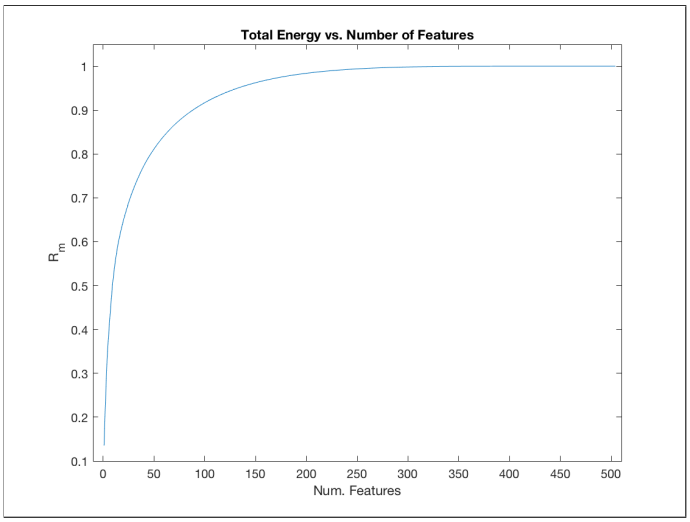
`Random`

	Correct	*Incorrect*	*Accuracy*
K = 1	340	0	100%
K = 2	98	102	49%
K = 3	94	106	47%
K = 4	91	109	45.5%
K = 5	91	109	45.5%

4.3. Principal Component Analysis

In the graph below, I plot R_m as a function of the number of retained features in the *face* dataset. The images from this dataset are originally of size 24 x 21 pixels, meaning that the feature vector, $\vec{x}$, is of size 504. Recall, PCA can be used to reduce the feature vector size to any arbitrary size m, so long as $m < d$, where $\vec{x} \in \mathbb{R}^d$.

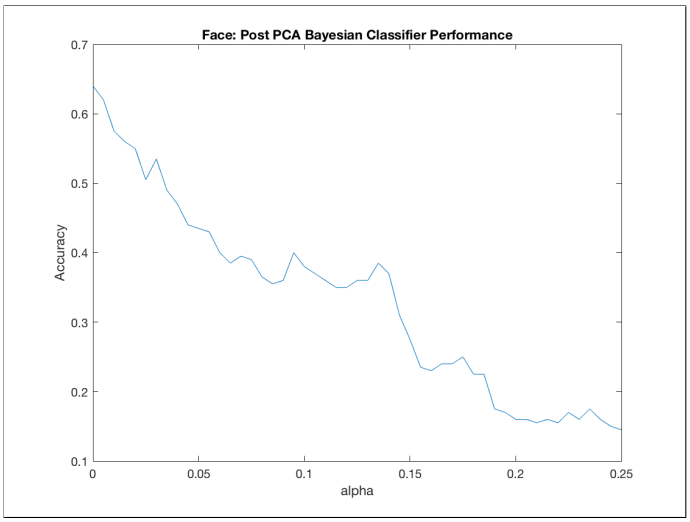
For example, reducing the *face* feature set in half (onto the 252 largest-variance principal components) would only cost 0.57% of the original energy. In fact, projecting onto 89 principal components would only lose 10% of the original energy.



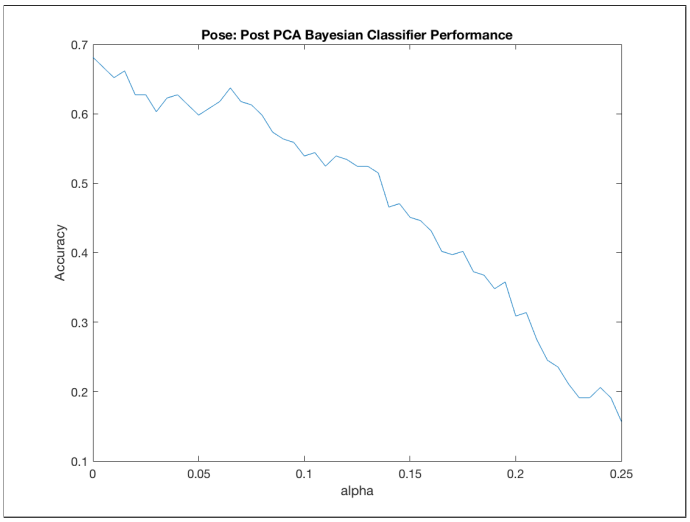
This same procedure can be used to reduce the feature set of the *pose* and *illumination* datasets.

4.3.1. Post-PCA Bayes'

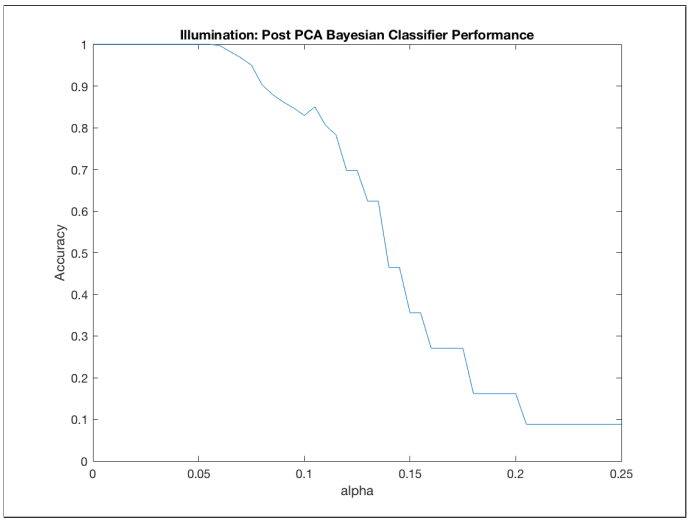
The following graphs showcase the performance of the post-PCA Bayesian classifier as a function of varying α values.



Projecting the *face* dataset onto the 190-highest-variant dimensions ($\alpha = 0.05$) resulted in a loss of accuracy amounting to slightly more than 20%. This sharp drop-off in accuracy directly correlates to the low number of training samples used in this dataset.



Conversely, PCA was much more effective on the *pose* and *illumination* datasets. Using $\alpha = 0.05$ reduced the feature vector size of the *pose* dataset from 1920 to 180, at a cost of 8% accuracy. Using the same value of α for the illumination dataset reduced the feature vector size from 1920 to 39, at no cost to the accuracy of the model. In other words, the Bayesian classifier still correctly classified all of the testing samples using only 2% of the original features. Recall, without PCA, the Bayesian classifier took 293.87 seconds to classify every test sample. After reducing the testing samples onto 39 dimensions, this same feat only took 0.2049 seconds.



It would appear that many of the features in the *illumination* dataset are extremely redundant. Fortunately, as one can see from the sample photos immediately on the following page, this checks out. Many of the pixels in these photos covary with the illumination alterations in the same manner (i.e. the pixels carry redundant information.)

Figure 3. Sample 1, Illumination 1

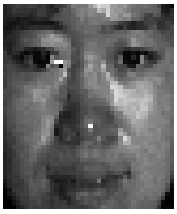
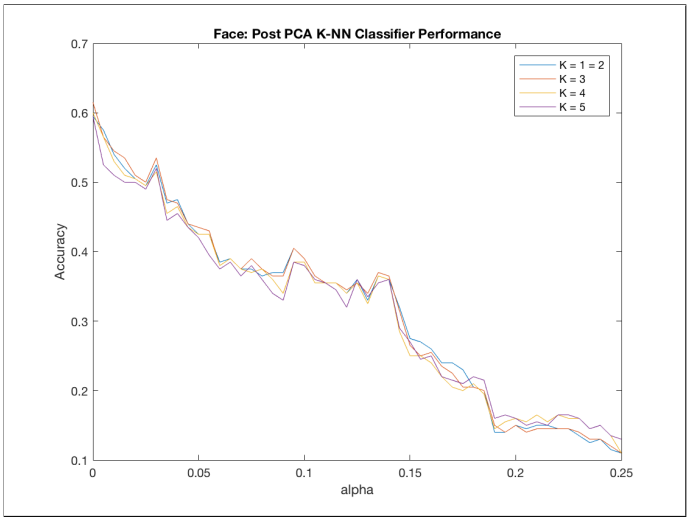


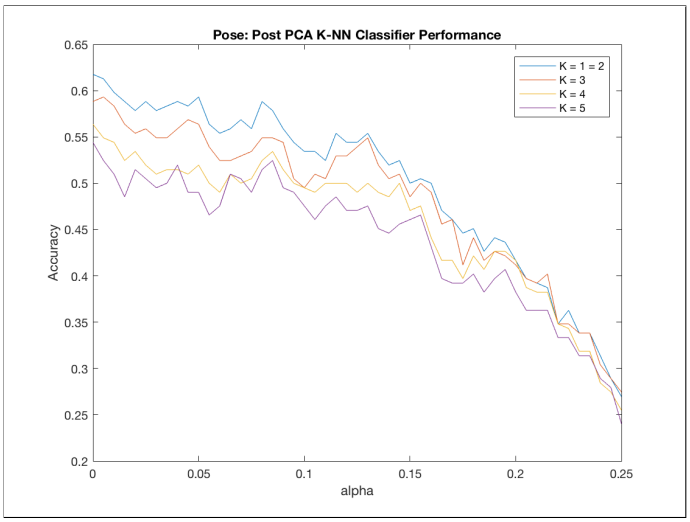
Figure 4. Sample 1, Illumination 5

4.3.2. Post-PCA K-Nearest Neighbors

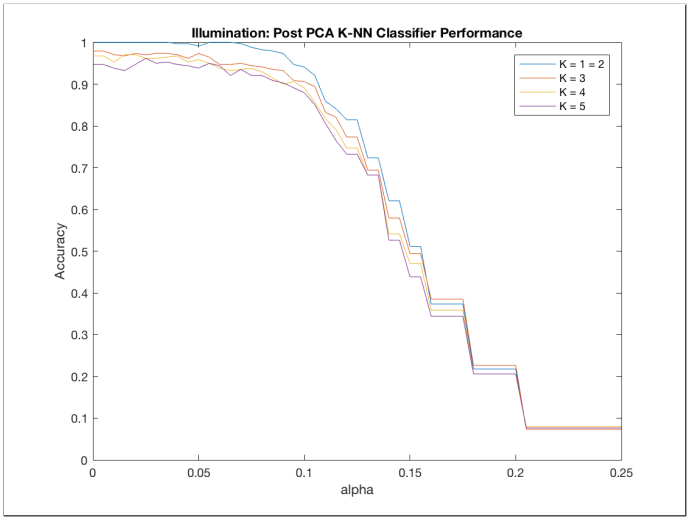
This section investigates the effect of varying the PCA parameter, α, on the K-Nearest Neighbors classification performance. As one can tell rather quickly, the shapes of the graphs for each particular dataset bear a striking resemblance to their post-PCA Bayesian classifier counterparts.



Across all three datasets, the trend of higher K values performing worse also continued. As before, this is most likely because of the sparse number of samples in the training set.



Interestingly, the discrepancy in performance between the different K-variants seemed to converge as α increased. This makes sense due to the shrinkage of the between-class scatter that creates class separation.



4.4. Fisher's Multiple Discriminant Analysis

When used as a dimension reduction routine, Fisher's Multiple Discriminant Analysis projects a dataset onto $c - 1$ dimensions. For the *face* dataset, this would requires projecting from 504 dimensions to 199. And for the others, it involves projecting from 1920 features to 67.

4.4.1. Post-MDA Bayes'

It would seem that the MDA projection, as with the PCA project, managed to remove many of the unnecessary features from the *illumination* dataset, while still preserving the integrity of the model. The same could not be said for the other two datasets. As stated before, these models do not appear to be expression or pose invariant, regardless of whether any sort of dimension reduction has been done beforehand.

Table 8. Post-MDA Bayesian Classifier Performance

	Correct Classifications	*Misclassifications*	*Accuracy*
Face	4	196	2%
Pose	47	157	23%
Illumination	340	0	100%

The Bayesian classifier performance on the *face* and *pose* datasets post-MDA was abysmal. If there were more samples available for training, I surmise that at the very least, the performance on the *face* dataset would improve.

4.4.2. Post-MDA K-Nearest Neighbors

As in previous sections, the same patterns seemed to surface. In general, as K increases, the performance of the classifier decreased. While performance was better than randomly guessing for all three datasets, only the *illumination* dataset had comparable performance to pre-MDA classification.

Table 9. Face: Post-MDA K-Nearest Neighbors Classifier Performance

	Correct	*Incorrect*	*Accuracy*
K = 1	4	196	2%
K = 2	4	196	2%
K = 3	4	196	2%
K = 4	5	195	2.5%
K = 5	6	194	3%

Table 10. Pose: Post-MDA K-Nearest Neighbors Classifier Performance

	Correct	*Incorrect*	*Accuracy*
K = 1	34	170	16.7%
K = 2	34	170	16.7%
K = 3	33	171	16.2%
K = 4	30	174	14.7%
K = 5	29	175	14.2%

Table 11. Illumination: Post-MDA K-Nearest Neighbors Classifier Performance

	Correct	*Incorrect*	*Accuracy*
K = 1	318	22	93.5%
K = 2	318	22	93.5%
K = 3	302	38	88.8%
K = 4	293	47	86.2%
K = 5	284	56	83.5%

5. Conclusion

It is clear that the number of training samples has a massive impact on the performance of any particular classifier, whether it be Bayesian, or non-parametric like the K-Nearest

Neighbors. In circumstances where this condition holds true, dimension reduction techniques are ineffective, and classification performance in general tends to suffer.

In addition, simple machine learning algorithms like the ones explored in this report tend to perform well on datasets where many of the features tend to covary. This explains the great success on the *illumination* dataset compared to the *pose* dataset, despite the fact that there were a similar number of training samples.

Finally, it is clear that an effective tiebreaking technique for the K-Nearest Neighbors algorithm should use the information from the previous run in order to break the tie. Of the implementations tested in this report, the `closest` method consistently performed the best, and if implemented properly, can break the tie very quickly.