University of Maryland, College Park

Final Term Project

A Comparative Study Of Representation Methods For Face Recognition

Chandramani

May 2, 2023

Table of Contents

Chapter 1

Introduction

1.1 Abstract

In this paper, we present a comparative analysis of the benefits of using representation methods for classification on the AT&T Face Database. We consider four different representation methods: raw data, PCA (Principal Component Analysis), kPCA (kernel PCA), and LDA (Linear Discriminant Analysis). I consider a nonlinear classifier: KNN (k-nearest neighbours) for the analysis. I evaluate the performance of each representation method and classifier on a 50/50 and 70/30 train/test split of the dataset. **Keywords**: representation methods, classification, AT&T Face Database, PCA, kPCA, LDA, KNN

1.2 Data Representation

Data representation is the process of transforming data into a new form that is more suitable for analysis or machine learning. This can be done by reducing the dimensionality of the data, transforming the data into a new feature space, or both.

There are many different data representation methods, but some of the most common include:

- **Principal component analysis (PCA)**: PCA is a linear dimensionality reduction method that finds a set of uncorrelated features that capture the most variance in the data.
- **Kernel PCA (kPCA)**: kPCA is a nonlinear dimensionality reduction method that uses kernel functions to map the data into a higher-dimensional space, where it can be linearly reduced..
- **Linear discriminant analysis (LDA)**: LDA is a supervised dimensionality reduction method that finds a set of features that best separates two or more classes of data..

Data representation is important in Data Analysis and Machine Learning for a number of reasons. First, it can help to improve the interpretability of the data. Second, it can help to improve the accuracy of machine learning models. Third, it can help to reduce the computational cost of machine learning algorithms.

For example, PCA can be used to reduce the dimensionality of a dataset of images, making it easier to visualize the data and to train machine learning models to classify the images. kPCA can be used to find nonlinear patterns in data, which can be useful for tasks such as image recognition and natural language processing. LDA can be used to improve the accuracy of machine learning models that are used to classify data, such as spam filters and medical diagnosis systems.

Data representation is a powerful tool that can be used to improve the performance of Data Analysis and Machine Learning algorithms. By transforming the data into a new form, it can make the data easier to understand, more accurate, and more efficient to process.

1.2.1 Principle Component Analysis

Principal component analysis (PCA) is a dimensionality reduction technique that finds a set of uncorrelated features that capture the most variance in the data. PCA is often used to reduce the dimensionality of data sets before performing other analyses, such as clustering or classification.

The PCA algorithm works by first finding the covariance matrix of the data. The covariance matrix is a square matrix that measures the linear relationships between the features of the data. The covariance matrix is then decomposed into its eigenvectors and eigenvalues. The eigenvectors are the principal components, and the eigenvalues are the variances of the principal components.

The principal components are sorted in decreasing order of their eigenvalues. This means that the first principal component captures the most variance in the data, the second principal component captures the second most variance, and so on.

The PCA algorithm can be used to reduce the dimensionality of the data by projecting the data onto the first few principal components. This is done by multiplying the data matrix by the matrix of eigenvectors and then keeping only the first few columns of the resulting matrix. The mathematical equation of the PCA is given by :

$$\mathbf{X} = \mathbf{USV}^\top \tag{1.1}$$

where X is the data matrix, U is the matrix of eigenvectors, S is the diagonal matrix of eigenvalues, and V is the matrix of principal components.

1.2.2 Kernal PCA

Kernel PCA (kPCA) is a nonlinear dimensionality reduction method that uses kernel functions to map the data into a higher-dimensional space, where it can be linearly reduced. The kernel function is a mathematical function that measures the similarity between two data points.

$$\mathbf{X}_k = \mathbf{KUSV}^\top \tag{1.2}$$

where X_k is the data matrix in the kernel space, K is the kernel matrix, U is the matrix of eigenvectors, S is the diagonal matrix of eigenvalues, and V is the matrix of principal components.

The kernel matrix is calculated using the **kernel function**. The most common kernel function is the Gaussian kernel, which is defined as follows:

$$k(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2\sigma^2}\right) \tag{1.3}$$

where σ is the kernel width. kPCA can be used to reduce the dimensionality of data by projecting the data onto a lower-dimensional subspace spanned by the first few principal components in the

kernel space. This can be useful for visualization, machine learning, and other applications where it is desirable to reduce the dimensionality of the data without losing too much information.

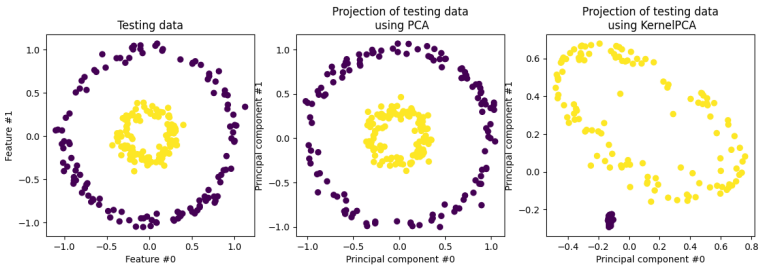


Figure 1.1: PCA and Kernal PCA representation

1.2.3 Linear Discriminant Analysis

LDA is a supervised dimensionality reduction method that finds a set of features that best separates two or more classes of data. The LDA algorithm finds a projection of the data that maximizes the between-class variance and minimizes the within-class variance. This projection is called the **discriminant axis**. LDA is a powerful tool that can be used to improve the performance of machine learning algorithms. By finding a projection of the data that maximizes the between-class variance and minimizes the within-class variance, LDA can help to improve the accuracy of classification tasks.

$$\mathbf{W} = \arg\max_{\mathbf{W}} \frac{\mathbf{W}^\top \mathbf{S}_B \mathbf{W}}{\mathbf{W}^\top \mathbf{S}_W \mathbf{W}} \tag{1.4}$$

where $S_B$is the between-class covariance matrix and $S_W$is the within-class covariance matrix.

LDA can be used to improve the accuracy of machine learning models that are used to classify data, such as spam filters and medical diagnosis systems. For example, LDA can be used to improve the accuracy of a spam filter by projecting the email messages into a lower-dimensional space where the spam and ham messages are more easily separated.

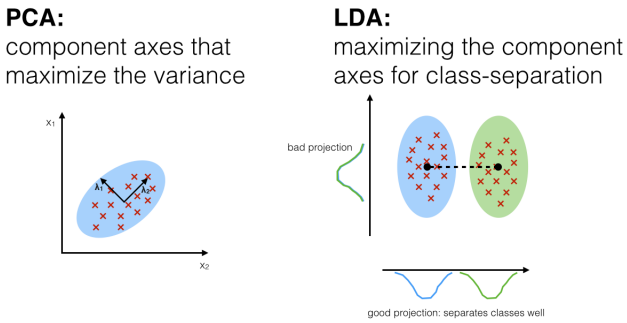


Figure 1.2: Difference between PCA and LDA

1.3 Dataset

. The **AT&T dataset** is a collection of 400 images of 10 people. Each image is of size 92 x 112 pixels and is stored in a separate directory. The directories are named s1, s2, and so on up to s40, where the number indicates the subject number. Each directory contains 10 images of the corresponding subject. The images are named y1.pgm, y2.pgm, and so on up to y10.pgm, where the number indicates the image number.

The AT&T dataset is a classic dataset in the field of face recognition. It has been used in many studies to evaluate the performance of face recognition algorithms. The dataset is relatively small, but it is still challenging, as the images are of low quality and the subjects are wearing different expressions and headwear.

The AT&T dataset is a valuable resource for researchers in the field of face recognition. It is a freely available dataset that can be used to train and evaluate face recognition algorithm

Figure 1.3: Example of AT&T Face Database

Chapter 2

Methodology

2.1 Data Splitting

As mentioned the AT&T dataset is a collection of 400 images of 10 people. Each image is of size 92 x 112 pixels. The dataset can be split into training and testing sets in a number of ways. I split the dataset into two ways :

- One Way is to split into 50-50 with 200 images in each set.
- Another common way is to split the dataset 70-30, with 280 images in the training set and 120 images in the testing set

The choice of how to split the dataset depends on a number of factors, including the size of the dataset and the desired accuracy of the model. If the dataset is small, it is important to use a 50-50 split to avoid overfitting the model to the training data. If the dataset is large, it is possible to use a 70-30 split without overfitting the model.

2.2 Algorithms

As mentioned in the introduction for the data representation apart from the raw data I used the following algorithms :

- PCA
- kPCA and
- LDA

For classification, I used the following classification algorithms :

- **KNN**: KNN (k-nearest neighbours) is a non-parametric, lazy learning algorithm that classifies a new data point by finding the k most similar data points in the training set and then assigning the new data point to the class that is most common among the k nearest neighbours.

Chapter 3

Data Representation

3.1 Raw Data

One of the images from the training data is represented below in greyscale. The image is loaded from the training_set array and reshaped to a 112x92 array. The image is then displayed in grayscale using the cmap=*'grey'* argument.

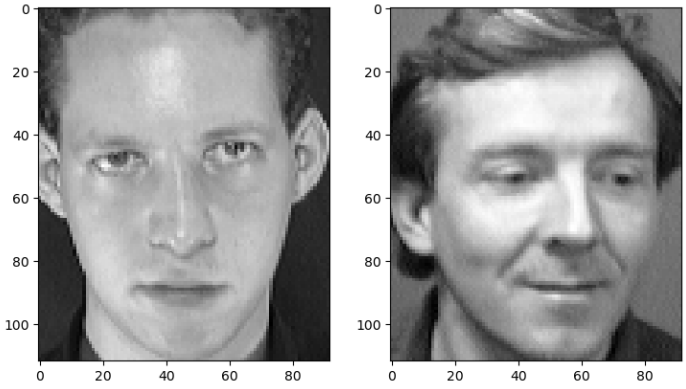


Figure 3.1: Raw Data Representation

3.2 PCA

To represent the data in PCA, I first compute the means of the training and test sets. The means are then used to centre the data, which means subtracting the means from each data point. The centred data is then used to compute the covariance matrix. The covariance matrix is a square matrix that stores the covariances between all pairs of features. The covariance matrix is then decomposed into its **eigenvectors and eigenvalues**. The eigenvectors are the principal components, and the eigenvalues are the variances of the principal components. The principal components are ordered by decreasing variance, so the first principal component captures the most variance in the data, the

second principal component captures the second most variance, and so on.

I then sorted the eigenvectors and eigenvalues in descending order of eigenvalues. The sorted eigenvectors are then used to project the data onto a lower-dimensional space. The lower-dimensional space is spanned by the first few principal components. Here I used the first fifty principal components to represent the data :

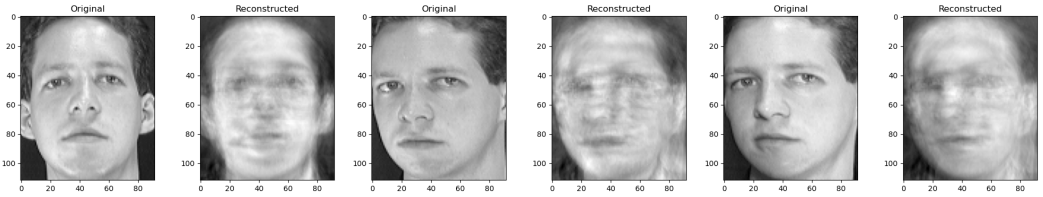


Figure 3.2: PCA Data Representation

3.3 kPCA

To represent the data in kPCA, I fitted kernel PCA with radial basis function (RBF) kernel. Unlike PCA, kPCA uses a kernel function to map the data into a higher dimensional space, where it may be more easily separable. The transformed dataset can then be projected onto a lower-dimensional space using the first few kernel principal components. Here I used the first fifty kernel principal components to represent the data.

After transforming the dataset, I reconstructed the images from the transformed dataset using the inverse_transform() method. However, since kPCA is a non-linear dimensionality reduction technique, the fit_inverse_transform parameter must be set to True when instantiating kPCA, to enable the inverse transform.

Finally, I plotted the original and reconstructed images using matplotlib. The images were displayed side by side, with the original image on the left and the reconstructed image on the right. I displayed ten images in total, but this can be changed by modifying the num_images variable in the code.

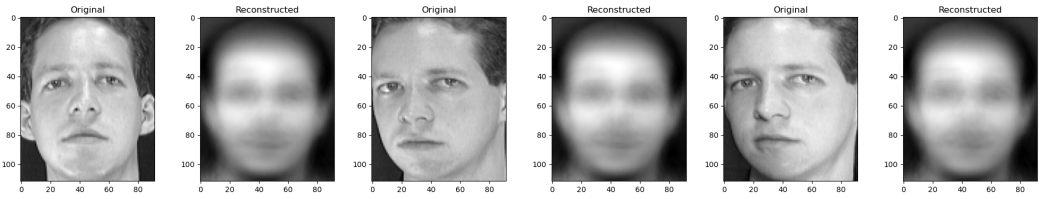


Figure 3.3: kPCA Data Representation

3.4 LDA

In LDA, the first step is to compute the overall mean of the data set. Next, I compute the between-class and within-class scatter matrices. The between-class scatter matrix measures the variance

between different classes, while the within-class scatter matrix measures the variance within each class. We then solve a generalized eigenvalue problem to obtain the eigenvectors and eigenvalues of the matrix product between the inverse of the within-class scatter matrix and the between-class scatter matrix.

Once we have obtained the eigenvectors and eigenvalues, I sort them in descending order of eigenvalues. We then take the dominant eigenvectors to project the data onto a lower-dimensional space. Here, we take all the eigenvectors since we are interested in a linear transformation.

We then project the test set onto the new space and plot the projected images. As with PCA, the lower-dimensional space captures the most important features of the data, and the projected images may look different from the original images. However, in LDA, the lower-dimensional space is designed to maximize the separation between classes, so the projected images may be more useful for classification tasks.

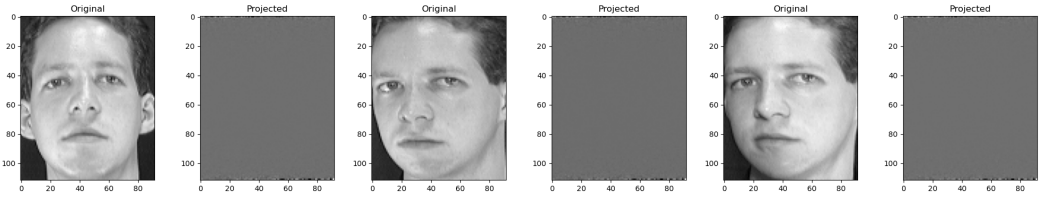


Figure 3.4: LDA Data Representation

Chapter 4

Results of Classification of Images using KNN

For image classification, I have opted to use a popular machine learning algorithm, namely K-Nearest Neighbor (KNN). To evaluate the performance of the models, I have conducted experiments using two different training set sizes, 50% and 70% of the available data. In each section, I have analyzed the results obtained from each of the data representations, including the raw method.

4.1 Classification on Raw Data

KNN

The KNN classifier is trained with different values of k ranging from 1 to 5. It was applied to a dataset with two different training sizes of 50% and 70%. For the 50% training size, the classifier achieved the highest accuracy of 90.5% with k=1, followed by 80.5% for k=2 and 80% for k=3. The accuracy decreased for k=4 and k=5, with 77% and 76% respectively. The plot of accuracy versus k shows a clear trend of decreasing accuracy with increasing k.

For the 70% training size, the classifier achieved a significantly higher accuracy of 97.5% with k=1, followed by 92.5% for k=2 and 90.8% for k=3. Again, the accuracy decreased for k=4 and k=5, with 87.5% and 86.7% respectively. The plot of accuracy versus k also shows a clear trend of decreasing accuracy with increasing k, but the decrease is more gradual than in the 50% training size case.

Overall, the results suggest that increasing the size of the training set can significantly improve the accuracy of the KNN algorithm. The choice of k is also an important parameter that affects the accuracy of the algorithm. In this case, the optimal value of k depends on the training size, with k=1 achieving the highest accuracy for both sizes. The results are discussed in the below table.

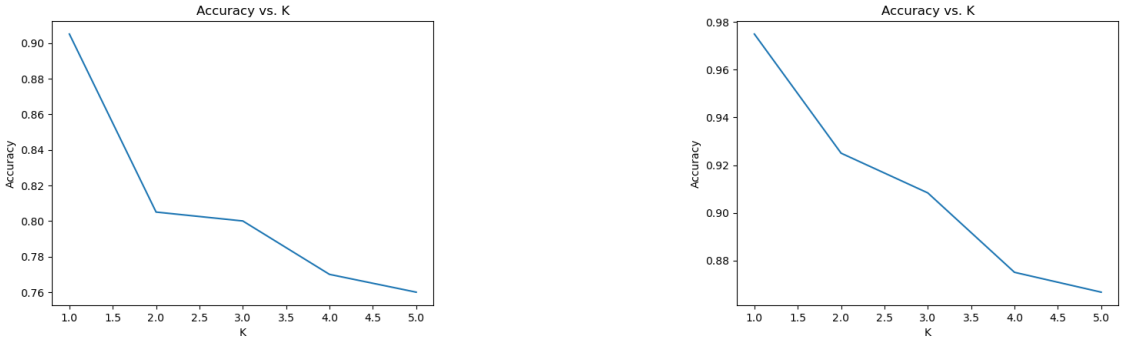
K	$\textbf{accuracy}_1$
1	90.5%
2	80.5%
3	80%
4	77%
5	76%

Table 4.1: 50% training data

K	$\textbf{accuracy}_2$
1	97.5%
2	92.5%
3	90.8%
4	87.5%
5	86.7%

Table 4.2: 70% training data

The plot of the accuracy vs K is shown below



(a) 50% data

(b) 70% data

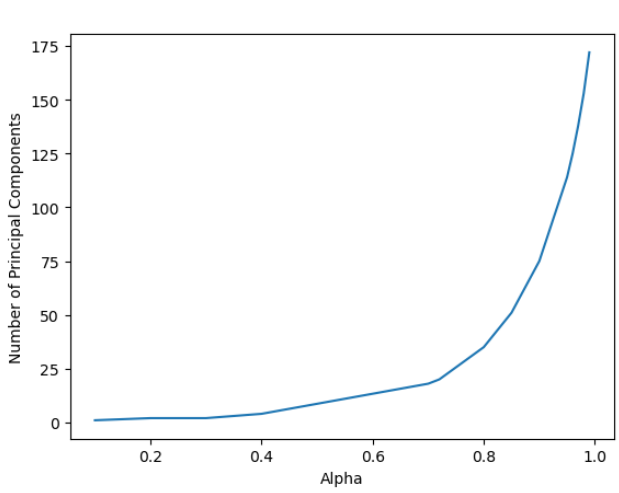
Figure 4.1: Accuracy vs K for KNN on Raw Data

4.2 Classification using PCA and KNN

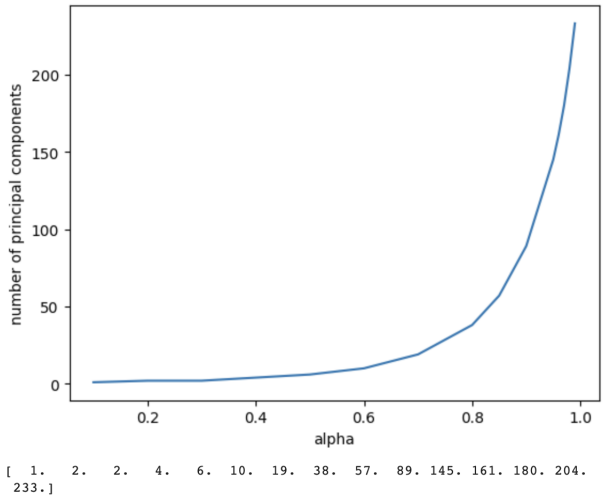
I performed Principal Component Analysis (PCA) on a given dataset to reduce the dimensionality of the dataset. I used the alpha values ranging from 0.7 to 0.99 which is used to determine the number of Principal Components (PCs) that should be retained to explain the variance in the data.

Initially, the code processes the training and test data by applying the mean centring technique and calculates the sorted eigenvalues and eigenvectors for the centred dataset. Then the sorted eigenvalues, eigenvectors, and the alpha value as inputs to determine the number of PCs required to explain the alpha per cent of the total variance.

Then I proceed to loop through each alpha value and calculate the number of PCs and the new space for each value. The new space is used to project the centred training and test sets onto a lower-dimensional subspace. K-Nearest Neighbors (KNN) classifier with one neighbour (**I took neighbour as 1 from the result of the raw data**)is used to classify the test data, and the classification scores are computed for each alpha value. Finally, the classification scores for each alpha value are appended to a list named scores, and the value of PC for each alpha is stored in the pcs list. This approach provides an idea of how much variance is captured by a specific number of PCs and helps in determining the optimal number of PCs for the given dataset. The no of principle components for 50% and 70% training data set for each alpha is shown in the below plot.



(a) 50% data

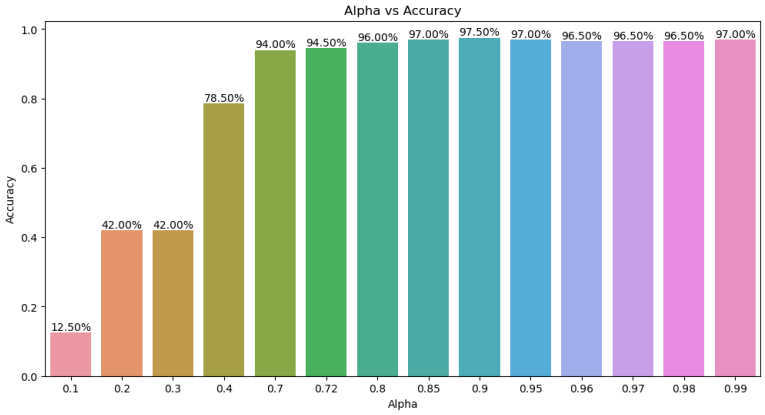


(b) 70% data

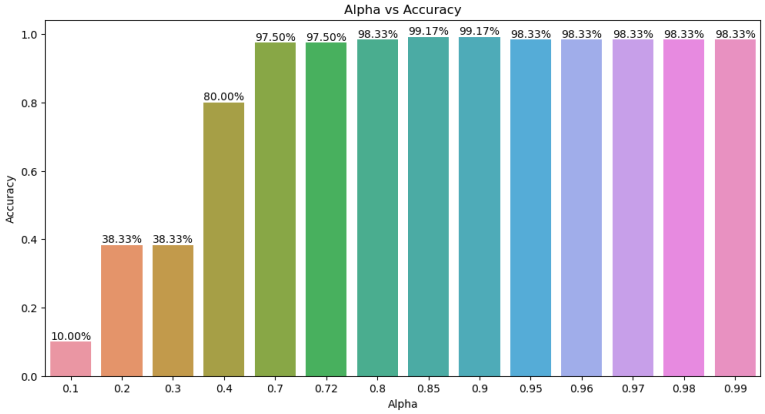
Figure 4.2: No of Principle Components for each alpha

The experiment shows the accuracy of the KNN algorithm with different alpha values ranging from 0.1 to 0.99 for 50% and 70% training data. The accuracy values range from 0.125 to 0.97 for 50% data and from 0.1 to 0.9916 for 70% data. **It is observed that as the alpha value increases, the accuracy of the model increases**. For 50% data, the accuracy values start from a very low value of 0.125 and keep increasing up to 0.97 with the increase in alpha value. Similarly, for 70% data, the accuracy values also show an increasing trend as the alpha value increases, but the starting value of accuracy is relatively higher than that of 50

The alpha value in the KNN-PCA algorithm represents the number of principal components to keep. Increasing the number of principal components leads to an increase in accuracy because the algorithm can capture more variance in the data, which is important for distinguishing between different classes. However, when alpha is very large, the model may suffer from overfitting, where it becomes too specialized to the training data and fails to generalize to new data. This is because the model is fitting to noise and other irrelevant features in the training data, rather than capturing the important underlying patterns that distinguish between classes. Therefore, there is a trade-off between capturing enough variance in the data to achieve good accuracy and avoiding overfitting. In the code provided, we can see that as alpha increases, the accuracy generally improves, but after a certain point, the accuracy begins to plateau or even decrease slightly, indicating that overfitting may be occurring.



(a) 50% data



(b) 70% data

Figure 4.3: Alpha vs Accuracy for dataset

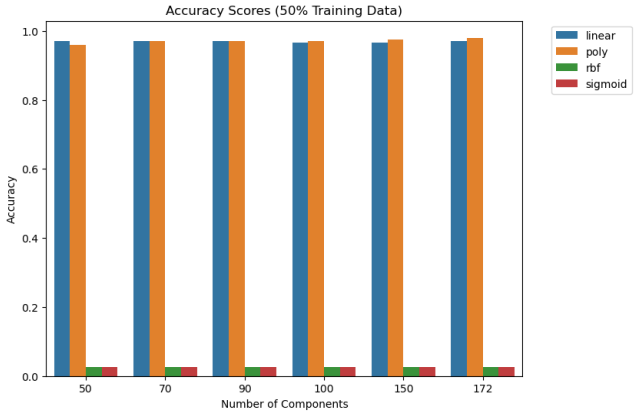
4.3 Classification using kPCA and KNN

I perform kernel principal component analysis (kPCA) on a given training set using different kernel functions such as linear, polynomial, radial basis function (RBF), and sigmoid kernels. The function takes a number of components and a kernel function as input and returns the accuracy of the kNN classifier trained on the transformed data. I then called this function for different values of n_components and kernels to experiment and stored the accuracy in a pandas data frame. I iterated over the n_components and kernel lists and filled the results data frame with the accurate results for each combination of n_components and kernel. The result for different components for

No of Components	kernal	Accuracy (50% training)	Accuracy (70% training)
50	linear	0.970	0.991667
50	poly	0.960	0.966667
50	RBF	0.025	0.025000
50	sigmoid	0.025	0.025000
70	linear	0.970	0.991667
70	poly	0.970	0.966667
70	RBF	0.025	0.025000
70	sigmoid	0.025	0.025000
90	linear	0.970	0.991667
90	poly	0.970	0.958333
90	RBF	0.025	0.025000
90	sigmoid	0.025	0.025000
100	linear	0.965	0.991667
100	poly	0.970	0.958333
100	RBF	0.025	0.025000
100	sigmoid	0.025	0.025000
150	linear	0.965	0.983333
150	poly	0.975	0.958333
150	RBF	0.025	0.025000
150	sigmoid	0.025	0.025000
172	linear	0.970	0.983333
172	poly	0.980	0.958333
172	RBF	0.025	0.025000
172	sigmoid	0.025	0.02500

Table 4.3: Accuracy scores for different values of components and kernal

The table above shows the accuracy scores for different values of n_components and kernel using two different training dataset sizes: 50% and 70%. The results indicate that the linear kernel consistently performs better than the other kernels across all values of n_components, achieving accuracy scores of 0.97 or higher on both training set sizes. The polynomial kernel performs well on some values of n_components, achieving scores of 0.96 to 0.98 on the 50% training set and 0.96 to 0.96 on the 70% training set. However, the accuracy drops significantly for higher values of n_components. On the other hand, the RBF and sigmoid kernels have very low accuracy scores across all values of n_components and both training set sizes, indicating that they are not suitable for this dataset. Overall, the results suggest that using a linear kernel with a moderate value of n_components, such as 70 or 90, is a good choice for achieving high accuracy on this dataset.



(a) 50% data

(b) 70% data

Figure 4.4: Accuracy scores for different values of n_components and kernel

4.4 Classification using LDA and KNN

I took the training set as an input parameter and computes the between-class scatter matrix and within-class scatter matrix to compute the eigenvectors and eigenvalues of the generalized eigenvalue problem. The function then returns the sorted eigenvectors.

The dominant eigenvectors are then selected using a list of indices, and the dominant eigenvectors are extracted from the sorted eigenvectors using slicing. These dominant eigenvectors are used to transform the training and test sets into a new space using the dot product of the training and test sets with the dominant eigenvectors.

Finally, the K-nearest neighbour (KNN) algorithm with k=1 is trained on the new transformed training set and tested on the transformed test set. The accuracy of the KNN classifier is then printed for each value of dominant eigenvectors. Overall, this code is implementing an LDA-based face recognition algorithm that utilizes the KNN classifier to predict the classes of new face images.

Dominant Eigen Vector	Accuracy (50%)	Accuracy (70%)
10.0	0.895	0.9083
40.0	0.955	0.975
80.0	0.955	0.983
100.0	0.950	0.975
200.0	0.960	0.975
250.0	0.960	0.975
300.0	0.960	0.975
500.0	0.960	0.975
800.0	0.965	0.975
1000.0	0.965	0.975
2000.0	0.970	0.975
4000.0	0.975	0.975
8000.0	0.965	0.975
10000.0	0.950	0.966
10300.0	0.955	0.975

Table 4.4: Accuracy with different dominant eigenvectors on LDA dataset space

The table represents the accuracy of the K-Nearest Neighbors (KNN) classifier using LDA with varying dominant eigenvectors for both 50% and 70% of the training data. As expected, the accuracy of the KNN classifier improves as the number of dominant eigenvectors increases. With the dominant eigenvectors less than 100, the accuracy for both 50% and 70% training data increases and stabilizes between 0.95 and 0.96. The accuracy further improves slightly and then remains constant for dominant eigenvectors greater than 100.

The dominant eigenvectors play a crucial role in LDA because they represent the directions that maximize the separation between the different classes. By selecting the top dominant eigenvectors, we project the original data into a lower-dimensional subspace that captures the most important discriminatory information between the different classes. The number of dominant eigenvectors that need to be retained depends on the size of the dataset, the number of classes, and the amount of variation in the dataset.

The accuracy remains constant for 70% training data because increasing the amount of training data allows the KNN classifier to better capture the underlying structure of the dataset, thereby reducing the generalization error. Since LDA is a supervised learning technique, it requires a sufficient amount of labeled data to be effective. With more labeled data, the model can capture more of the variations within each class and improve the discrimination between different classes. As a result, the model can generalize better to unseen data, which leads to a constant accuracy for the 70% training data.

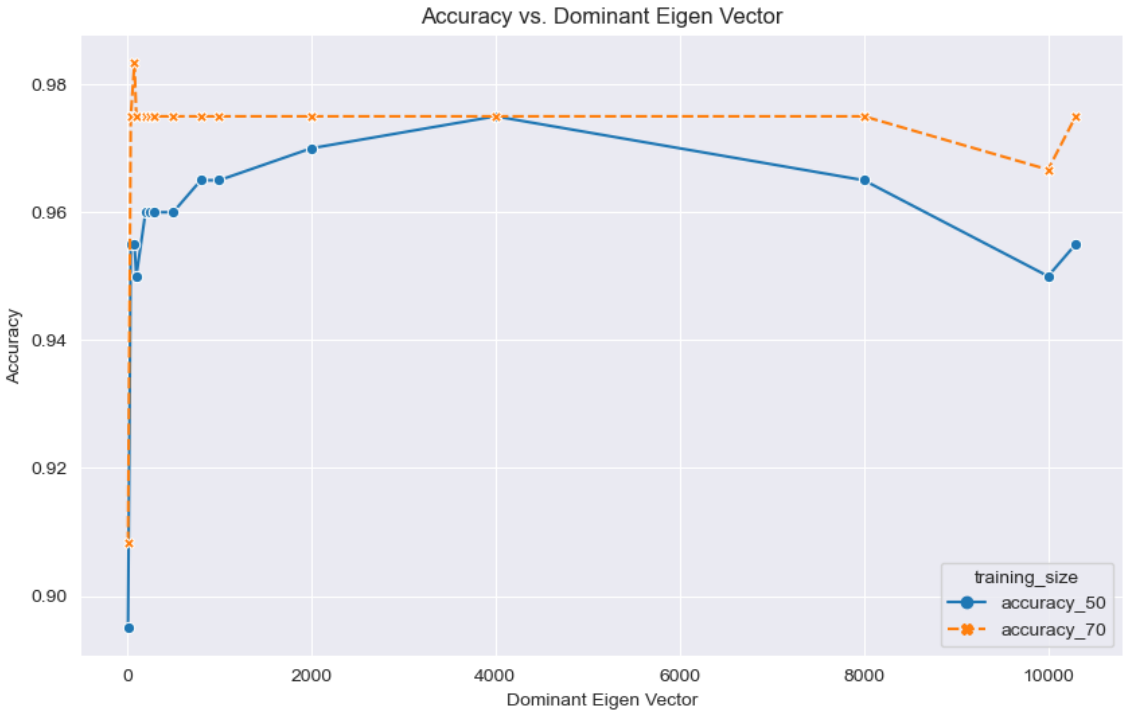


Figure 4.5: Accuracy vs. Dominant Eigen Vector

Chapter 5

Conclusion

Based on the results obtained from the project of image classification, we can conclude that the accuracy of image classification greatly depends on the pre-processing techniques applied on the training data.

When using raw data, the accuracy achieved for 50% and 70% training sets is 90.5% and 97.5% respectively, which is relatively low compared to the accuracies achieved through the pre-processing techniques.

When the training set is projected using PCA, it was observed that the highest accuracy achieved was 97% for 50% data and 99.17% for 70% data. The highest accuracies were achieved when 172 components were used for 50% data and 114 components for 70% data.

In the case of kPCA, the highest accuracy achieved was 98% for 50% training data and 99.16% for 70% training data. The accuracies were obtained using poly kernel for 50% training data and linear kernels on components 70, 90, and 100 for 70% training data.

When the training set is projected using LDA, the highest accuracy achieved was 97.5% for both 50% and 70% training data. This was achieved by using 4000 dominant eigenvectors.

In the case of image classification, where the data is represented as high-dimensional vectors of pixel intensities, PCA and kPCA can be used to reduce the dimensionality of the data while still preserving the most important information needed for classification. This is because images are highly redundant and contain a lot of irrelevant information.

On the other hand, Linear Discriminant Analysis (LDA) is a supervised dimensionality reduction technique that attempts to find the best linear combination of features that can separate different classes of data. In image classification, LDA may not perform well when the classes are not well-separated, or when the number of classes is high. Here there are 40 labels that may have come into factor.

In the case of the ATA&T dataset, which contains images of faces, PCA and kPCA may perform better than LDA and raw data because they are able to capture the most important features of the images while ignoring irrelevant information. PCA and kPCA reduce the dimensionality of the data, making it easier to classify, and they are able to capture the underlying structure of the data by preserving the maximum variance.

In summary, the highest accuracies were achieved when the training set was pre-processed using PCA and kPCA techniques. However, LDA also provided a high accuracy of 97.5%.

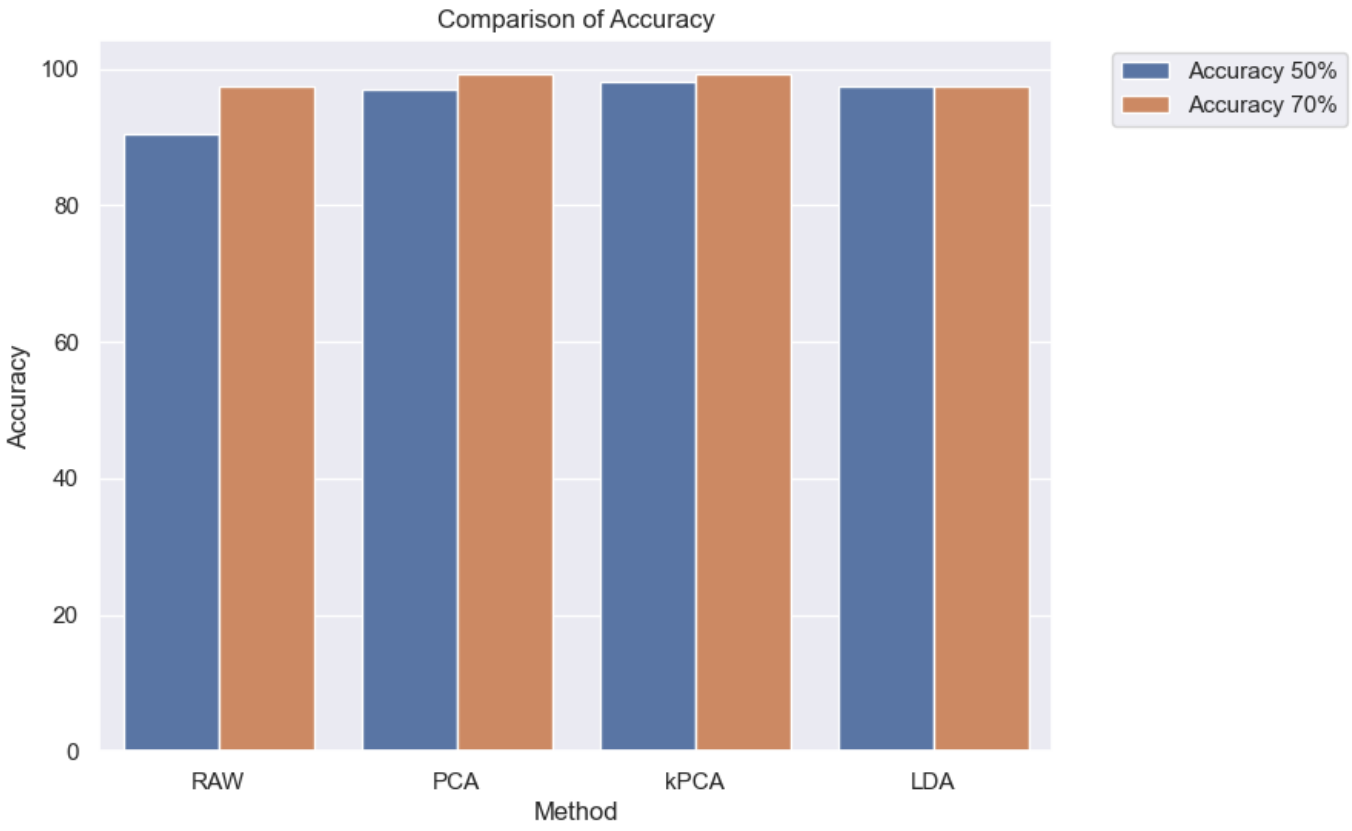


Figure 5.1: Comparison of Accuracy