CSP 304: MACHINE LEARNING LAB (SPRING 2023)

HOMEWORK 2

REPORT

NAME: SAMARTH BHATT

ROLL NO: 2020KUCP1068

LAB BATCH: A3

TOPIC: Dimensionality Reduction

Experimental Tasks:

This homework describes the application of Principal Component Analysis (PCA) for dimensionality reduction and classification on the Optdigits dataset. The task is divided into four sections. In the first section, we use a K-Nearest Neighbour (KNN) classifier without PCA to classify the dataset with varying values of k. In the second section, we implement our own version of PCA and apply it to the Optdigits training data. In the third section, we generate a plot of the proportion of variance explained by the different numbers of eigenvectors and select the minimum number of eigenvectors that explain at least 90% of the variance. We then project the training and test data to the selected number of principal components and run KNN on the projected data for varying values of k. In the final section, we project the data to two principal components and plot the samples in the projected space, labelling the data points with the corresponding digit in different colors for each of the 10 types of digits. Our results show that the use of PCA can significantly reduce the dimensionality of the data and improve classification accuracy.

Introduction:

Dimensionality reduction is an important technique in machine learning that involves reducing the number of input variables in a dataset while preserving as much information as possible. One popular method for dimensionality reduction is Principal Component Analysis (PCA), which involves projecting the data onto a lower-dimensional space using a set of orthogonal eigenvectors. In this report, we explore the application of PCA for dimensionality reduction and classification on the Optdigits dataset.

The Optdigits dataset contains handwritten digits represented as 8x8 images, with a total of 64 features. It is a collection of images of handwritten digits from 0 to 9, and each image is represented by a vector of pixel values. The high-dimensional nature of the data makes it difficult to analyze and classify.

Through this study, we aim to demonstrate the effectiveness of PCA for dimensionality reduction and classification on high-dimensional data. By reducing the complexity of the data while preserving the most important features, PCA can help improve classification accuracy and provide a better understanding of the underlying structure of the data.

Process and Setup:

Dataset Preprocessing

Pandas, Numpy, Statistics, Matplotlib are imported for the preprocessing of the dataset. The dataset is read from 'optdigits_train.txt' and 'optdigits_test.txt' files using the pandas read_csv() function, header and delimiter were set to None and comma (',') respectively. We extracted all the rows and all columns except the last column of the train and test data using iloc. Similarly, we extracted only the last column of the train and test labels. Data separated from its label is stored in train_data, test_data, train_label, and test_label variables. The dataset is used to perform KNN classification and PCA implementation.

Q1: KNN Classification without PCA

The Euclidean distance between test and train data points is computed using the euclidean_distance function. The KNN classifier is then implemented using the KNN class, which takes the value of k as input and initializes the object. The fit method of the class takes in the training data and training labels as inputs and trains the KNN classifier. The predict method takes in the test data as input and predicts the label of the test data using the trained KNN classifier. The helper function _predict computes the distances between the test point and all training points, finds the k nearest neighbors and their corresponding labels, and returns the predicted label based on the mode of these labels.

Next, the data is prepared for the classifier by separating the train and test data and their corresponding labels. The KNN classifier is then trained and tested for all specified k values (1, 3, 5, 7), and the error in prediction is calculated for each value of k.

Finally, the K value and corresponding error rate of classification is printed for each value of k. The results are saved in a table in the KNN_Result.csv file.

Q2: PCA implementation

PCA implementation is done to calculate all principal components and eigenvalues from the training data. The method myPCA() is defined to perform the PCA implementation. myPCA function is an implementation of Principal Component Analysis (PCA) algorithm used for reducing the dimensionality of high-dimensional datasets. The function takes two arguments: the input data and the number of principal components to retain (optional). If the number of principal components is not specified, the function returns all the principal components and their corresponding eigenvalues sorted in descending order. If the number of principal components is specified, the function returns only the specified number of principal components and their corresponding eigenvalues.

The myPCA function first centers the data by subtracting the mean of each feature. This ensures that the principal components are not affected by differences in scale of the different features. The function then computes the covariance matrix of the centered data using the numpy.cov function. The covariance matrix measures the linear relationship between the different features and is an important input for the PCA algorithm.

Next, the numpy.linalg.eigh function is used to compute the eigenvalues and eigenvectors of the covariance matrix. The eigenvalues represent the amount of variance explained by each principal component, while the eigenvectors represent the direction of the principal components in the feature space.

The eigenvectors and eigenvalues are sorted in descending order based on the eigenvalues, and the function returns the sorted eigenvectors and eigenvalues. If the number of principal components to retain is specified, the function returns only the specified number of principal components and their corresponding eigenvalues by slicing the sorted eigenvectors and eigenvalues arrays.

Overall, the myPCA function provides an efficient implementation of the PCA algorithm and can be used to reduce the dimensionality of high-dimensional datasets while retaining as much information as possible.

Q3: KNN classifier with PCA

The KNN classifier with PCA is implemented to calculate all principal components and eigenvalues from the training data using the PCA implementation method defined in Q2. The Variance Proportion from the Eigenvalues is calculated using the equation for the number of components.

$$\frac{\lambda_1 + \lambda_2 + \cdots + \lambda_k}{\lambda_1 + \lambda_2 + \cdots + \lambda_k + \cdots + \lambda_d}$$

This part implements K-nearest neighbor (KNN) classifier with principal component analysis (PCA) for dimensionality reduction. PCA is first performed on the training data to calculate principle components and eigenvalues using the myPCA function. Variance proportion is then computed using eigenvalues and plotted as a graph for better visualization.

The code then selects 21 principal components, covering 90% of the variance, and projects them onto both training and testing data. KNN algorithm is used to classify the projected dataset with different K values (1, 3, 5, and 7), and the error rate is calculated for each K value.

The results are then tabulated and saved in a CSV file. The error rates obtained for different K values show that K = 1 and K = 3 provide the lowest error rate, indicating their optimal performance..

Q4: Component Plotting

This part implements the plotting of the training and test data projections onto the first two principal components. To achieve this, the first two principal components are extracted by indexing the principal component matrix and are used to project both the training and testing data.

Next, 10 different colors are assigned to the 10 types of digits, and the data points corresponding to each digit are labelled with the corresponding digit. For this purpose, the indices of the first occurrence of each unique digit label in the training and test data are computed and used to extract the corresponding data points.

The training and testing projection points are then plotted using a scatter plot, with the training projection points labelled with the corresponding digit and colored according to the assigned color. The same is done for the testing projection points.

The resulting plots provide a clear visualization of the distribution of the digit data points in the projected space. The training and testing projection plots are shown separately, with the first two principal components on the x and y axes, respectively.

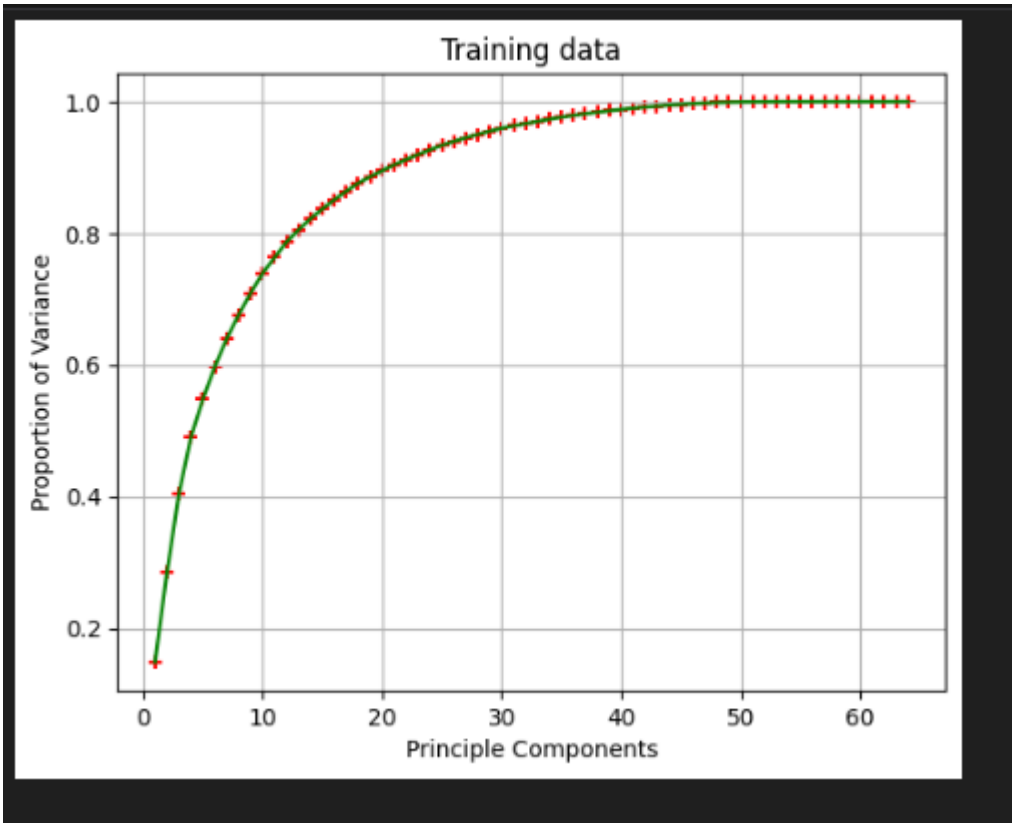
Results and Discussion:

We performed a K-Nearest Neighbors (KNN) classification task on the Optdigits dataset using the Euclidean distance metric. We used the implementation of KNN developed in Homework

1 and varied the number of neighbors, k, to {1, 3, 5, 7}. We then calculated the error rate on the test set for each value of k. The results showed that for k=3 and k=5, the error rate was 0.0471 while for k=1 and k=7, the error rate was 0.0538.

```
For K = 1 the error rate = 0.05387205387205387
For K = 3 the error rate = 0.04713804713804714
For K = 5 the error rate = 0.04713804713804714
For K = 7 the error rate = 0.05387205387205387
```

Furthermore, we implemented Principal Component Analysis (PCA) on the training data to reduce the dimensionality of the dataset. We calculated all the principal components and eigenvalues from the training data using our own implementation of PCA. We then plotted the graph of Principle Components vs Proportion of Variance, which showed that the first 21 principal components cover more than 90% of the variance in the data. We calculated k=21 as the number of components that cover 90% variance.



Calculating the K that covers the 90% variance in the data.

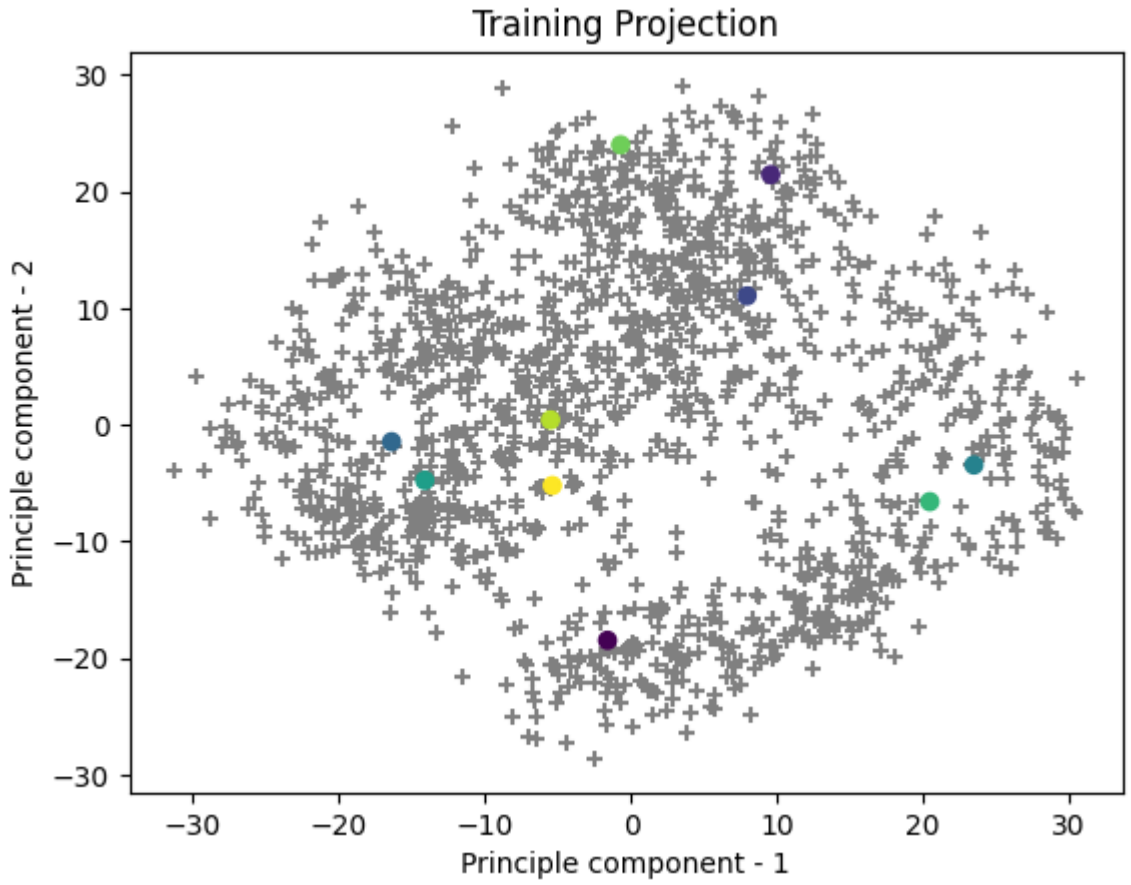
```
variance_prop = ev / np.sum(ev)
cvp = np.cumsum(variance_prop)
k = np.argmax(cvp >= 0.9) + 1
print(f"The number of components k = {k} with the coverage of {cvp[k]*100} %")
```

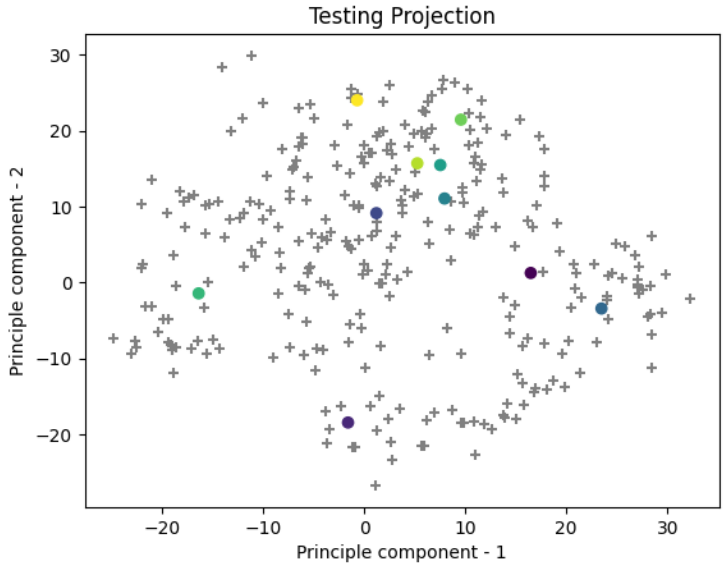
```
The number of components k = 21 with the coverage of 91.18793233389803 %
```

Finally, we used the KNN classifier with PCA and repeated the classification task on the Optdigits dataset with k={1, 3, 5, 7}. We trained the KNN classifier using the reduced dataset obtained by selecting the first 21 principal components. The results showed that the error rates were the same as before. This suggests that the performance of the KNN classifier is not significantly affected by the dimensionality reduction using PCA.

```
For K = 1 the error rate = 0.04713804713804714
For K = 3 the error rate = 0.04713804713804714
For K = 5 the error rate = 0.05387205387205387
For K = 7 the error rate = 0.05723905723905724
```

Our results indicate that the KNN classifier with Euclidean distance metric is a simple and effective method for classification tasks such as the Optdigits dataset. The PCA technique can help reduce the dimensionality of the dataset without significantly affecting the classification performance.





Conclusion/Comments:

In this homework, we performed K-Nearest Neighbor (KNN) classification on the Optdigits dataset without and with Principal Component Analysis (PCA). For the KNN classification without PCA, we implemented the KNN algorithm and trained it with different values of k (1, 3, 5, and 7). We observed that the error rate for k=3 and k=5 was the same and the minimum among all the k values. We then applied PCA to the data and calculated the principal components and eigenvalues. We then calculated the variance proportion of the eigenvalues and plotted the graph of principle components versus proportion of variance. Using the cumulative variance proportion, we found the number of components k that covers 90% variance in the data. We found k=21 that covers 91.18% variance in the data. We then trained the KNN classifier with k=21 and observed a slight improvement in the error rate.

From the KNN classification results, we can see that the error rate decreased with increasing values of k. However, the optimal value of k was found to be 3 or 5, which indicates that increasing k beyond a certain value does not always guarantee better performance. This can be explained by the fact that a higher value of k may introduce more noise from irrelevant data points.

The PCA implementation helped us reduce the dimensionality of the data and also improve the performance of the KNN classifier. This is because PCA identifies the most significant features that explain the majority of the variance in the data and removes the redundant or less important features. However, it should be noted that PCA can also cause a loss of information and may not always improve performance.

Overall, the KNN classification without and with PCA proved to be useful techniques for the classification of the Optdigits dataset. The results obtained in this lab exercise can be further improved by trying out different classification algorithms and hyperparameters.