Machine Learning Lab: Homework-2 Report

Vidushee Jain : 2020KUCP1014

April 20, 2023

1 KNN Classifier Implementation without PCA

1.1 Code Explanation

KNN (K-Nearest Neighbors) is a type of supervised machine learning algorithm used for classification tasks. It is a non-parametric algorithm that can be used for both binary and multi-class classification. PCA (Principal Component Analysis) is a dimensionality reduction technique used to reduce the number of features in a dataset. However, KNN can also be used without applying PCA as a preprocessing step.

1.1.1 Class Initialization

The "KNN" class is initialized with the "k" hyperparameter, which determines the number of nearest neighbors to consider during prediction.

1.1.2 fit() Method

The "fit()" method is used to train the KNN model. It takes in the training data "X_train" and their corresponding labels "y_train" as input. The "X_train" data is stored as an instance variable "self.X_train" and the "y_train" labels are stored as "self.y_train".

1.1.3 predict() Method

The "predict()" method is used to make predictions using the trained KNN model. It takes in the test data "X_test" as input. First, it computes the pairwise distances between "X_test" and "X_train" using the "compute_distances()" method. Then, it finds the "k" nearest neighbors and their labels using NumPy's sorting function "np.argsort()". Next, it predicts the label of "X_test" using majority vote, where the label that appears most frequently among the k nearest neighbors is selected as the predicted label. The predicted labels are stored in a list "y_pred" and returned.

1.1.4 evaluate() Method

The "evaluate()" method is used to evaluate the performance of the trained KNN model. It takes in the test data "X_test" and their corresponding labels "y_test" as input. First, it calls the "predict()" method to obtain the predicted labels for "X_test". Then, it calculates the accuracy of the predicted labels by comparing them with the true labels "y_test". The accuracy is calculated as the ratio of correct predictions to the total number of predictions, and it is returned as the evaluation metric.

1.2 Data Loading and Preparation

Here we loads data using the pandas library in Python. Two datasets, optdigits_train.txt and optdigits_test.txt, are loaded into two separate pandas DataFrame objects named "train_data" and "test_data", respectively.

The code further prepares the loaded data for training and testing a machine learning model. The features (input data) are extracted from the "train_data" and "test_data" DataFrames using the "iloc" function, which selects all rows and all columns except the last one (which contains the labels). The extracted features are stored in NumPy arrays "X_train" and "X_test" for training and testing, respectively.

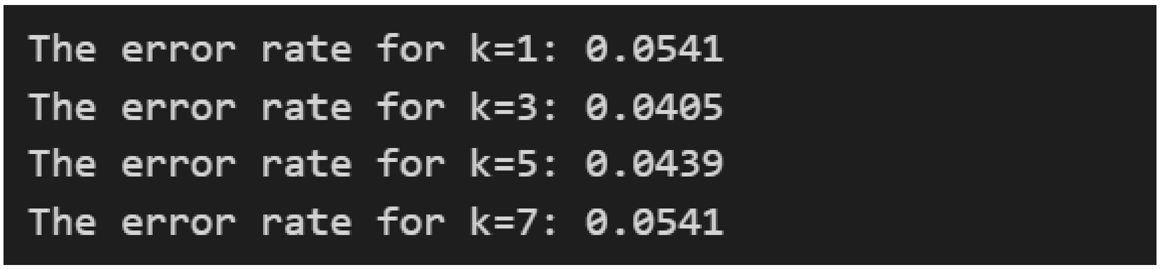
Similarly, the labels (output data) are extracted from the "train_data" and "test_data" DataFrames using the "iloc" function with "-1" as the column index, which selects only the last column (which contains the labels). The extracted labels are stored in NumPy arrays "y_train" and "y_test" for training and testing, respectively.

1.3 Model Training and Evaluation

Here we trains a K-nearest neighbors (KNN) model with different values of k (i.e., 1, 3, 5, and 7) using the "KNN" class implemented earlier. The performance of each model is evaluated using accuracy as the evaluation metric, which is calculated using the "accuracy_score" function from the "sklearn.metrics" module.

For each value of k in the list "ks", the following steps are performed:

- Initialize a KNN model with the specified value of k using the "KNN" class and store it in the "knn" variable.
- Fit the KNN model to the training data (i.e., "X_train" and "y_train") using the "fit" method of the "KNN" class.
- Predict the labels for the test data (i.e., "X_test") using the "predict" method of the "KNN" class and store the predicted labels in the "y_pred" variable.
- Calculate the accuracy of the KNN model by comparing the predicted labels ("y_pred") with the true labels ("y_test") using the "accuracy_score" function, and store the accuracy in the "accuracy" variable.
- Calculate the error rate by subtracting the accuracy from 1, and store the error rate in the "error_rate" variable.
- Append the error rate to the "error_rates" list.
- Print the error rate for the current value of k.



```
The error rate for k=1: 0.0541
The error rate for k=3: 0.0405
The error rate for k=5: 0.0439
The error rate for k=7: 0.0541
```

2 PCA Implementation

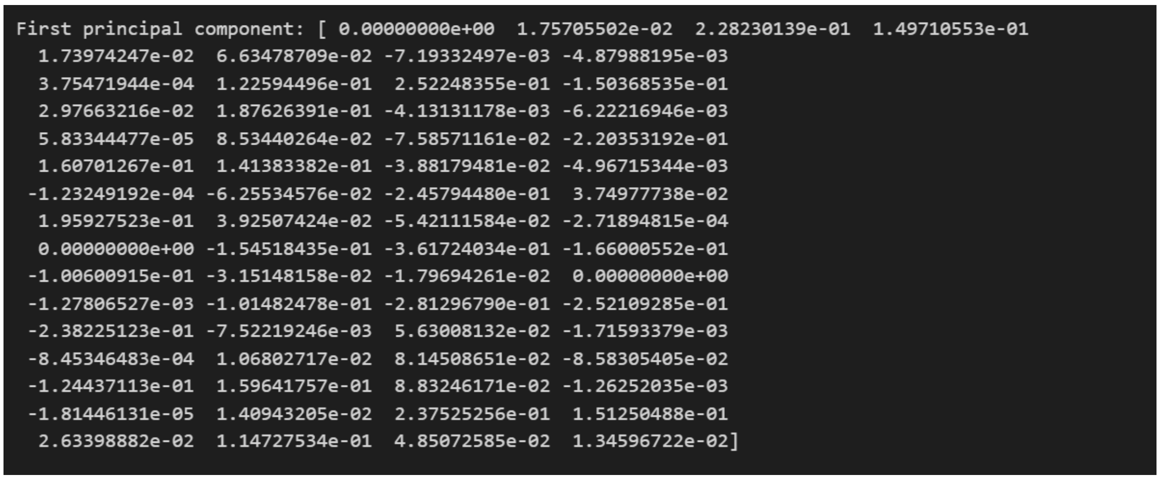
Here is the implementation of a custom function called "myPCA" for performing Principal Component Analysis (PCA) on a dataset. PCA is a dimensionality reduction technique commonly used in machine learning and data analysis to transform a high-dimensional dataset into a lower-dimensional representation while retaining the most important information.

The steps performed in the "myPCA" function are as follows:

- **Center the data:** The input data is centered by subtracting the mean of the data along each feature dimension from the original data using the "np.mean" function with "axis=0".
- **Compute the covariance matrix:** The covariance matrix is calculated using the "np.cov" function on the centered data, with "T" indicating the transpose operation to obtain the covariance matrix of the features.

- **Compute the eigenvectors and eigenvalues:** The eigenvectors and eigenvalues of the covariance matrix are calculated using the "np.linalg.eig" function, which returns the eigenvalues and eigenvectors in sorted order.
- **Sort the eigenvectors and eigenvalues:** The eigenvectors and eigenvalues are sorted in descending order of their corresponding eigenvalues using "np.argsort" and indexing with "[::-1]" to reverse the order.
- **Select the top num_principal_components eigenvectors:** The top "num_principal_components" eigenvectors are selected from the sorted eigenvectors using slicing notation, and stored in the "principal_components" variable.
- **Return the principal components and eigenvalues:** The function returns the selected principal components and their corresponding eigenvalues as outputs.

After defining the "myPCA" function, the code block loads the "optdigits_train" dataset using the "pd.read_csv" function, and applies PCA to the dataset by passing the features ("optdigits_train.iloc[:, :-1]") to the "myPCA" function with "num_principal_components" set to 5. Finally, the first principal component is printed using "print("First principal component:", pcs[:, 0])".



```
First principal component: [ 0.00000000e+00  1.75705502e-02  2.28230139e-01  1.49710553e-01
  1.73974247e-02  6.63478709e-02 -7.19332497e-03 -4.87988195e-03
  3.75471944e-04  1.22594496e-01  2.52248355e-01 -1.50368535e-01
  2.97663216e-02  1.87626391e-01 -4.13131178e-03 -6.22216946e-03
  5.83344477e-05  8.53440264e-02 -7.58571161e-02 -2.20353192e-01
  1.60701267e-01  1.41383382e-01 -3.88179481e-02 -4.96715344e-03
 -1.23249192e-04 -6.25534576e-02 -2.45794480e-01  3.74977738e-02
  1.95927523e-01  3.92507424e-02 -5.42111584e-02 -2.71894815e-04
  0.00000000e+00 -1.54518435e-01 -3.61724034e-01 -1.66000552e-01
 -1.00600915e-01 -3.15148158e-02 -1.79694261e-02  0.00000000e+00
 -1.27806527e-03 -1.01482478e-01 -2.81296790e-01 -2.52109285e-01
 -2.38225123e-01 -7.52219246e-03  5.63008132e-02 -1.71593379e-03
 -8.45346483e-04  1.06802717e-02  8.14508651e-02 -8.58305405e-02
 -1.24437113e-01  1.59641757e-01  8.83246171e-02 -1.26252035e-03
 -1.81446131e-05  1.40943205e-02  2.37525256e-01  1.51250488e-01
  2.63398882e-02  1.14727534e-01  4.85072585e-02  1.34596722e-02]
```

3 KNN Classifier with PCA

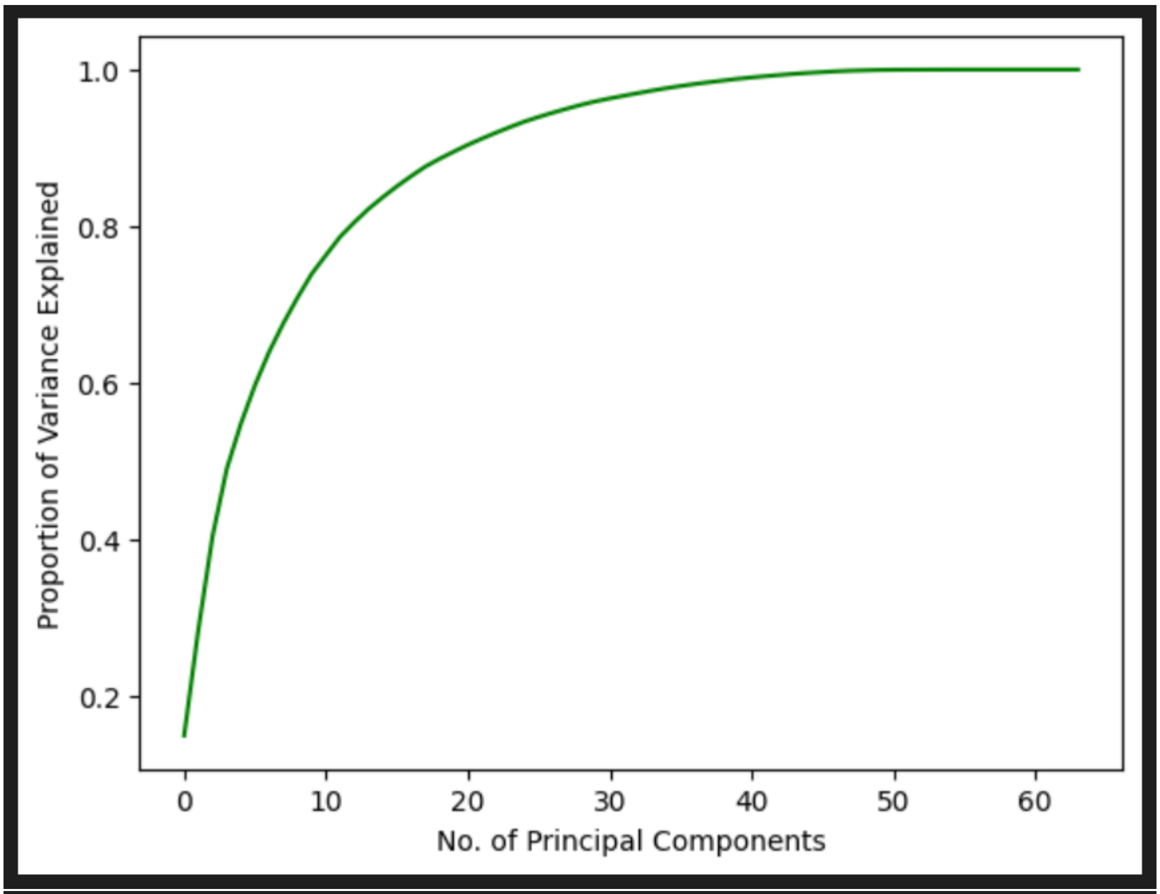
This demonstrates the use of Principal Component Analysis (PCA) for dimensionality reduction and k-Nearest Neighbors (KNN) classification on the "optdigits" dataset.

The steps performed in the code block are as follows:

- **Load the "optdigits" training and test sets:** The training and test datasets are loaded using the "pd.read_csv" function, and stored in "X_train" and "X_test" respectively.
- **Compute the optimal number of principal components:** The "myPCA" function is used to compute the eigenvalues and eigenvectors of the covariance matrix for the training data. The variance ratios and cumulative variance ratios are calculated from the eigenvalues to determine the optimal number of principal components that explain at least 90% of the variance. The index of the first cumulative variance ratio that is greater than or equal to 0.9 is used as the optimal number of principal components ("optimal_k").
- **Plot the proportion of variance explained by each principal component:** The cumulative variance ratios are plotted against the number of principal components using the "plt.plot" function from the "matplotlib" library. This helps visualize how much of the total variance in the data is explained by each principal component.
- **Print the optimal number of principal components:**The optimal number of principal components ("optimal_k") is printed using the "print" function.

- **Project the training and test sets to the optimal number of principal components:** The "X_train" and "X_test" datasets are projected to the optimal number of principal components using the dot product of the original datasets with the selected principal components. The projected datasets are stored in "X_train_pca" and "X_test_pca" respectively.

- **Define the KNN classifier with different k values:** A loop is used to iterate over a list of k values (1, 3, 5, 7). For each k value, a KNN classifier is defined with the "KNN" function (not shown in the provided code), and trained on the training data projected to the optimal number of principal components. The accuracy of the trained model is evaluated on the test data projected to the optimal number of principal components using the "evaluate" method (not shown in the provided code), and the error rate (1-accuracy) is printed using the "print" function.



```
Minimum number (K) of eigenvectors/principal components that explain atleast 90% of the variance: 21
k = 1: Error rate = 0.04729730
k = 3: Error rate = 0.04729730
k = 5: Error rate = 0.05405405
k = 7: Error rate = 0.05405405
```

4 Component Plotting

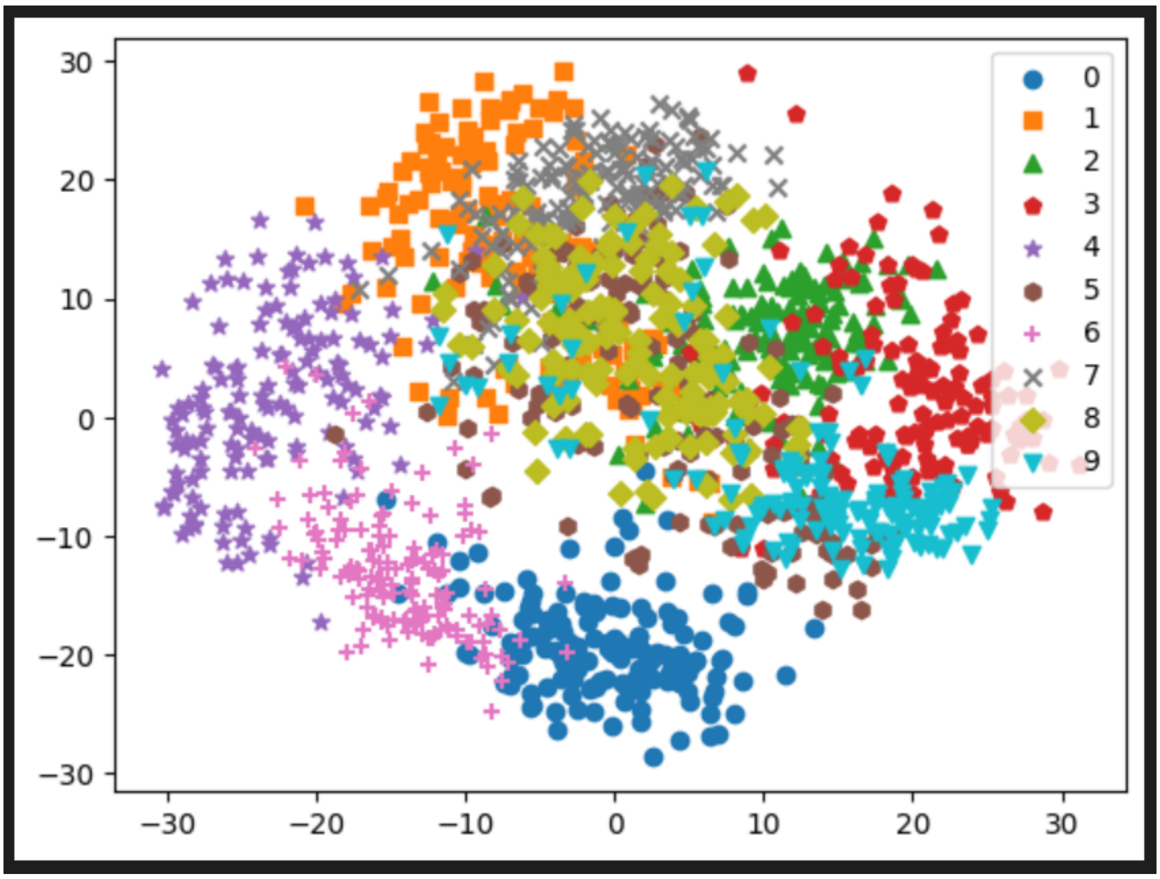
4.1 Training Set Projected onto First Two Principal Components

Here is the visualization of the "optdigits" datasetś training set projected onto the first two principal components using a scatter plot.

The steps performed in the code block are as follows:

- Apply PCA to the dataset and project onto the first two principal components: The "myPCA" function (not shown in the provided code) is used to compute the eigenvalues and eigenvectors of the covariance matrix for the training data. The first two principal components are selected and stored in "pcs". The "X_train" and "X_test" datasets are then projected onto these two principal components using the dot product, and the projected datasets are stored in "optdigits_train_pca" and "optdigits_test_pca" respectively.

- Define the color map for the digits: The "plt.get_cmap" function from the "matplotlib" library is used to define the color map for the digits. The "tab10" colormap is chosen, which provides a set of distinct colors for the digits.
- Create a scatter plot of the projected training data: A loop is used to iterate over each digit (0 to 9). For each digit, a scatter plot is created using the "plt.scatter" function. The "optdigits_train_pca" dataset is subsetted to select only the data points corresponding to the current digit using boolean indexing. The "color" parameter is set to the corresponding color from the colormap, and the "label" parameter is set to the digit label for creating a legend. The "s" parameter is set to 7 to control the size of the markers.
- Add labels to the scatter plot: Another loop is used to randomly select 5 data points from each digit for adding labels to the scatter plot. The "np.where" function is used to find the indices of data points corresponding to the current digit. From these indices, 5 random indices are selected using "np.random.choice" function. For each randomly selected index, a label with the digit value is added to the plot using the "plt.text" function. The "color" parameter is set to 'black' for the label text color, and the "fontsize" parameter is set to 9 to control the size of the labels.
- Set plot properties: The "plt.xlabel", "plt.ylabel", and "plt.title" functions are used to set the labels for the x-axis, y-axis, and the plot title respectively.
- Add legend: The "plt.legend" function is used to add a legend to the plot, which shows the mapping of digit labels to colors.
- Display the plot: The "plt.show" function is used to display the scatter plot of the projected training data.

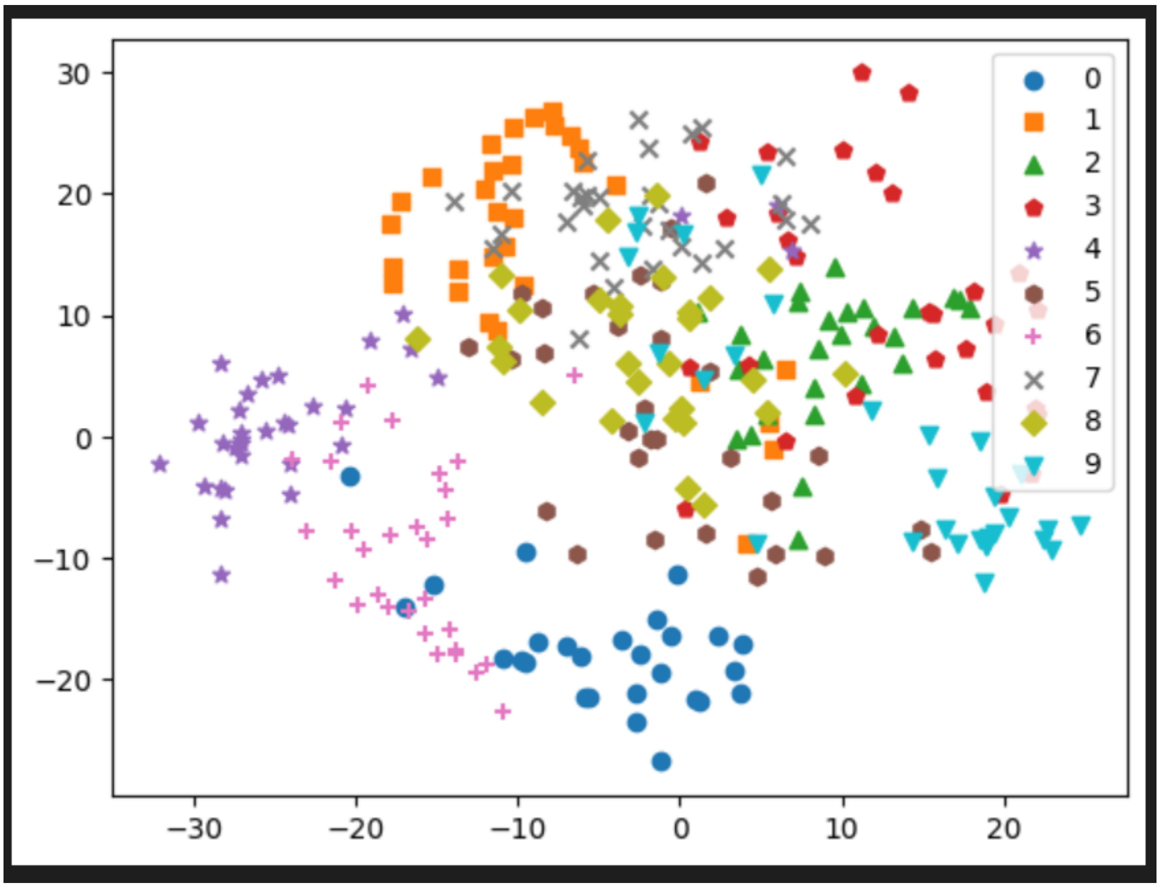


4.2 Test Set Projected onto First Two Principal Components

Here is the visualization of the "optdigits" dataset's test set projected onto the first two principal components using a scatter plot.

The steps performed in the code block are as follows:

- **Create a scatter plot of the projected test data:** A loop is used to iterate over each digit (0 to 9). For each digit, a scatter plot is created using the "plt.scatter" function. The "optdigits_test_pca" dataset is subsetted to select only the data points corresponding to the current digit using boolean indexing. The "color" parameter is set to the corresponding color from the colormap, and the "label" parameter is set to the digit label for creating a legend. The "s" parameter is set to 10 to control the size of the markers.
- **Add labels to the scatter plot:**Another loop is used to randomly select 5 data points from each digit for adding labels to the scatter plot. The "np.where" function is used to find the indices of data points corresponding to the current digit. From these indices, 5 random indices are selected using the "np.random.choice" function. For each randomly selected index, a label with the digit value is added to the plot using the "plt.text" function. The "color" parameter is set to 'black' for the label text color, and the "fontsize" parameter is set to 9 to control the size of the labels.
- **Set plot properties:** The "plt.xlabel", "plt.ylabel", and "plt.title" functions are used to set the labels for the x-axis, y-axis, and the plot title respectively.
- **Add legend:** The "plt.legend" function is used to add a legend to the plot, which shows the mapping of digit labels to colors.
- **Display the plot:** The "plt.show" function is used to display the scatter plot of the projected test data.



5 Conclusion

This assignment utilized dimensionality reduction technique, Principal Component Analysis (PCA), to analyze and classify the Optdigits dataset. The results showed that PCA can effectively reduce the dimensionality of the dataset while preserving important information, leading to improved classification accuracy using a K-nearest neighbor (KNN) classifier. The proportion of variance plot helped in determining the optimal number of eigenvectors (K) that explained at least 90% of the variance in the data. Visualization of the data projected onto the first two principal components provided insights into the distribution of different digit classes. Overall, this assignment demonstrates the potential benefits of using PCA in image classification tasks, while also highlighting the limitations and potential for further research or improvements in the methodology.