AMATH 582: Homework 3

Kutay Demiralay

Aeronautics and Astronautics Department Department, University of Washington, Seattle, WA

In this homework assignment, our objective is to train classifiers for the recognition of images depicting handwritten digits, using the well-known MNIST dataset. We employed Ridge, KNN, and LDA classifiers on various subsets and sets, comparing their respective performances.

Introduction and Overview

The MNIST training dataset comprises 60,000 samples, each representing a distinct handwritten digit changing from 0 to 9 randomly. Each digit is characterized by 784 features, denoting individual pixels in a 28x28 resolution image. This dataset is utilized for training our algorithm. Additionally, the test dataset consists of 10,000 sample images, each with the same 784 features, and is employed for validating the performance of the algorithm. Different classification methods are applied to analyze and classify this dataset using python programming language.

Theoretical Background

Principal Component Analysis (PCA) is a mathematical technique employed for reducing the dimensionality of data sets while preserving essential patterns and variations. It operates by identifying the principal components within the dataset, which are the directions where the data shows the most significant variability. The data is linearly transformed onto a new coordinate system. PCA helps identify the most influential aspects, sort of like finding the most critical features or trends that capture the essence of the data. Basis for PCA can be obtained using Singular Value Decomposition shown in eq.(1), Where the U matrix contains the left singular vectors, Σ is a diagonal matrix of singular values and *V^T* contains the right singular vectors.

$$X=U\Sigma V^{T} \quad (1)$$

Obtaining the projection of a dataset onto k-PC modes involves three steps:

-First, we have to center the data by subtracting the mean of each column from the corresponding values in the dataset.

-Secondly, apply SVD on centered data to acquire U ,Σ , V^T matrices.

-Thirdly, the principal components are represented by the columns of matrix U. To obtain the first k principal components, select the first k columns of matrix U, and set the remaining columns to zero. Then, multiply the transpose of this truncated matrix with the dataset to create a new approximated dataset.

$$X_{\text{projected}_k} = U^{T}{}_{k\text{-Truncated}}\, X \quad (2)$$

The Energy can be computer by the square of Frobenius Norm of the matrix, which is also equal to multiplication of all the singular values square, as it can be seen in eq.(3)

$$E= ||A||_F^{2}=\sum_{j=1}^{\min(m,n)} \sigma_j^{2} \quad (3)$$

Maximum Likelihood Estimation is a statistical method used to estimate the parameters of a statistical model. The idea behind MLE is to find the values of the model parameters that maximize the likelihood function, which measures how well the model explains the observed data. eq.(4) is the formulation for the Maximum Likelihood Estimation where f is our model, x represents code samples, and y represents the labels.

$$f_{\text{MLE}}= \underset{f}{\text{argmin}}||f(x)\text{-}Y||^{2} \quad (4)$$

With the model we are trying to decrease our Mean Square Error in our classification as possible, shown in eq (5).

$$\text{MSE}\,(\hat{f}(x),y) = \frac{1}{N}\sum_{n=1}^{N}|\hat{f}(x)\text{-}Y|^{2} \quad (5)$$

Ridge Regression is a variant of Linear Regression that incorporates a regularization term as a parameter, aiming to prevent overfitting and enhance the model's generalization capabilities. Regularization is employed to impose penalties on higher-order terms, aiming to minimize both the loss function and the risk of overfitting in the model. P is equal to 2 most of the time and is equal to 1 sometimes in eq(6) .We recast classification method using regularization. This regularization helps us to avoid singular matrices in the computation of Beta. Choosing a proper lambda will be crucial for a proper classifier. Too big regularization lambda will make the classification lose information and too small lambda value will bring it closer to singularity.

$$\beta_{\text{MLE}} = \hat{\beta} = \underset{\beta}{\text{argmin}} = ||A\beta - x||^2 + \frac{\lambda}{2}||\beta||_p^p \qquad (6)$$

Cross-validation is a statistical technique used to evaluate the performance of a machine learning model. It involves partitioning the dataset into subsets, training the model on some of these subsets, and evaluating its performance on the remaining data. The primary purpose is to assess how well the model generalizes to an independent dataset. It increases consistency by tuning the lambda hyper parameter.

Another classification method, LDA, employs a logistic regression model represented by a sigmoid function as seen on Eq(7). We use this sigmoid model to compute mean square error and or Maximum Likelihood Estimation.

$$f(x) = \frac{1}{e^{-(\beta_0 + \beta_1 x)}} \qquad (7)$$

The KNN classifier, short for k-nearest neighbor classifier, is a machine learning algorithm that classifies a data point based on the majority class of its k-nearest neighbors in the feature space So, for each new point, the KNN algorithm computes the distance to all existing points, essentially creating a ball or circle in 2D (or a higher-dimensional space). This process continues until it includes the k-nearest neighbors. For classification, we consider the majority or maximum occurrence of features within this ball as the classification feature for the new point. In other words, the class label assigned to the new point is determined by the most prevalent class among its k-nearest neighbors

Algorithm Implementation and Development

All coding was done in the python programming language. Numpy, Matplotlib, Scipy,, Math, Seaborn and Sklearn libraries were used.

First Step: Downloading HW3_Helper.jpyn

This Code sample helped me to reshape each image into a vector and stack the vectors into matrices Xtrain and Xtest respectively. After assigning the correct paths for the training and test data, along with their corresponding labels, the necessary data preparation steps were already implemented within the provided code

Second Step: Performing PCA analysis

Using the PCA command from sklearn.decomposition, I was able to perform Principal Component Analysis of the digit images in the train set. PCA command shifts the data mean to be zero and scales the data for me so I don't have to worry about that part. First 16 modes were plotted as 28x28 images.

```
Xtrain = traindata_imgs.reshape((traindata_imgs.shape[0], -1))
pca = PCA(n_components=16)  # You can adjust the number of components as needed
pca.fit(Xtrain)
```

Third Step: Cumulative Energy Inspection

By utilizing np.cumsum(pca.explained_variance_ratio_), I effortlessly determined the cumulative energy of singular values spanning from the first to the 784th principal component in PCA. Then the number of PC mods needed to approximate %85 of the energy is found using this vector of cumulative energy, which is 59.

```
pca = PCA()
pca.fit(Xtrain)
cumulative_variance_ratio = np.cumsum(pca.explained_variance_ratio_) # Calculate cumulative explained variance ratio
```

Fourth Step: Inspecting Several Approximated digit images reconstructed using k-PCA projection.
Pca.transform command, transforms the original image into the PCA space.Then, we use pca.inverse_transform to reconstruct the image using only the first k principal components, which is 59 in this case. The reconstructed image is reshaped back to its original dimensions.

```
num_samples = 5 # Choose a few samples for visualization
sample_indices = np.random.choice(Xtrain.shape[0], num_samples, replace=False)
fig, axes = plt.subplots(num_samples, 2, figsize=(8, 2*num_samples))# Visualize approximated digit images
for idx, sample_idx in enumerate(sample_indices):
  original_image = Xtrain[sample_idx].reshape(28, 28)
    # Reconstruct the image using the truncated number of principal components
  reconstructed_image = pca.inverse_transform(pca.transform(Xtrain[sample_idx].reshape(1, -1))[:, 
:num_components_85_percent])
  reconstructed_image = reconstructed_image.reshape(28, 28)
```

Fifth Step: Generating a subset from the training and testing datasets, containing only the desired digits.
Write a function that selects a subset of specific digits (comprising all samples of those digits) from *Xtrain*, *ytrain*, *Xtest*, and *ytest*, returning the subset as new matrices *Xsubtrain*, *ysubtain*, *Xsubtest*, and *ysubtest*.

```
def select_digit_subset(digit, Xtrain, ytrain, Xtest, ytest):
  train_indices = np.where(ytrain == digit)[0]
  test_indices = np.where(ytest == digit)[0]
  Xsubset_train = Xtrain[:, train_indices]
  ysubset_train = ytrain[train_indices]
  Xsubset_test = Xtest[:, test_indices]
  ysubset_test = ytest[test_indices]
  return Xsubset_train, ysubset_train, Xsubset_test, ysubset_test
```

Sixth Step: Apply the Ridge classifier to the specified dataset projected onto k-PC modes, utilize the approximated data, and conduct cross-validation.

```
RidgeCL = RidgeClassifierCV() #Assuming data's already been projected into k-PC modes.
RidgeCL.fit(X_combined_test_approx, y_combined_test)
scores = cross_val_score(RidgeCL, X_combined_test_approx, y_combined_test, cv=5)
```

Seventh Step: Apply the KNN classifier to the specified dataset projected onto k-PC modes, utilize the approximated data, and conduct cross-validation.
The cross validation will be exactly the same as the Ridge Classifier case. The k number is the number that will give us the highest accuracy scores. It must be found using iteration.

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_combined_train_approxall)
X_test = scaler.transform(X_combined_test_approxall)
KNNCL = KNeighborsClassifier(n_neighbors=k)
```

```
KNNCL.fit(X_train, y_combined_trainall)
KNNCL.fit(X_test, y_combined_testall)
```

Eighth Step: Apply the LDA classifier to the specified dataset projected onto k-PC modes, utilize the approximated data, and conduct cross-validation.

```
LDACL = LinearDiscriminantAnalysis()  #The cross validation will be exactly the same as the Ridge Classifier case.
LDACL.fit(X_combined_train_approxall,y_combined_trainall)
```

Computational Results

I conducted a PCA analysis on the digit images within the training set. As part of this analysis, I visualized the first 16 Principal Component (PC) modes as 28 × 28 images. As shown in the figure (1), the images are blurry and non readable.

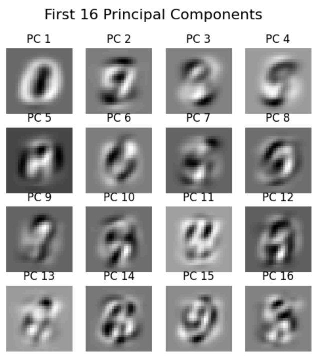
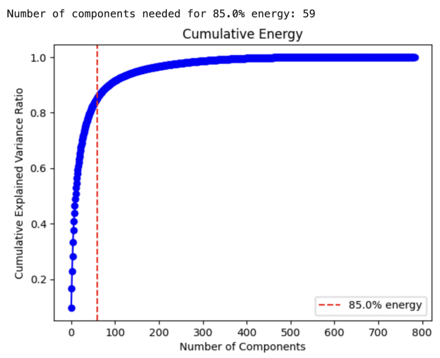


Figure 1: Plot of first 16 PCA modes

I examined the cumulative energy of the singular values and identified *k* as the minimum number of Principal Component (PC) modes required to capture at least 85% of the energy. In this case, I determined *k* to be 59. I visualized this finding by plotting the cumulative energy against the number of components, with a highlighted red line passing through 59 for a clear representation of the 85% threshold, as shown in figure (2).



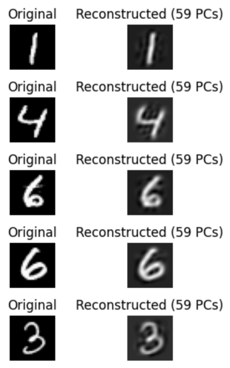


Figure 2: Cumulative Energy vs. Number of PCA Modes   Figure 3:Original and. Reconstructed Images of 5 Random Digits

I further examined five randomly selected approximated digit images reconstructed using *k*=59 truncated Principal Component (PC) modes. I plotted these approximated images to ensure that the image reconstruction using the truncated modes appears satisfactory and retains visual clarity when displayed in a 28x28 resolution, along with original images of each one of them respectively, as can be seen on figure (3). With 59 modes, the reconstructed images seem reasonable, one can easily classify these images of digits. They are just a bit blurry than the original images.

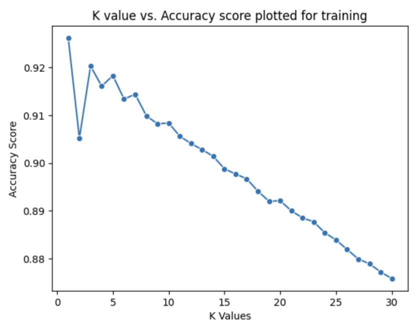
Accuracy scores and standard deviations were obtained using the Ridge classification method for both training and testing subsets, each containing distinct digits, following cross-validation with cv=5, presented in Table 1.

Digits included in subset	Training Accuracy Score	Training Standard Deviation	Testing Accuracy Score	Testing Standard Deviation	Difference between Training and Testing in terms of Standard D.	Training Accuracy Score before cross validation
1-8	**0.9644249**	**0.0039948**	**0.9725017**	**0.0118525**	**2.0215965**	**0.9647423**
3-8	**0.9583547**	**0.0055410**	**0.9576660**	**0.0112958**	**0.1242868**	**0.9608579**
2-7	**0.9804470**	**0.0027408**	**0.9660194**	**0.0124710**	**5.2639144**	**0.9809375**

Table 1. Accuracy Scores and Standard Deviations Obtained from Ridge Classification for Subsets Containing Specific Digits.

- Lowest Training and Testing accuracy scores are obtained from a subset containing 3-8 digits. This is because digits 3 and 8 look similar and are harder to distinguish then digits 2-7 and 1-8.
-The most significant disparity in training and testing scores was observed in the subset encompassing digits 2 to 7. This discrepancy indicates a potential issue of overfitting, where the training subset exhibits a considerably higher accuracy score, exceeding five times the standard deviation of the training set, compared to the accuracy score of the testing subset. Enhancing the training and testing datasets by incorporating a broader range of scenarios could lead to more reliable results.
-Highest Training scores are obtained from the subset containing 2-7 digits. Suggesting that these digits 2 and 7 are easiest to distunguish.
-The accuracy scores of all subsets experienced a decrease following cross-validation, emphasizing the significance of cross-validation in enhancing the classifier's adaptability to real-world applications.

While determining the cross-validated testing and training accuracy scores for the subset containing all digits using the k-nearest neighbors (KNN) classification method, I manually selected the value of k for the k-nearest neighbor algorithm. I plotted k values ranging from 1 to 30 against accuracy scores and chose the k value corresponding to the highest accuracy score for both testing and training datasets. The below figure (4,5) are K values vs. Accuracy Scores plotted for both training and testing. It is evident that k=1 gives the highest accuracy score for both testing and training.



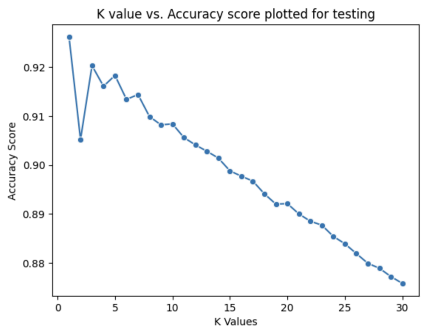


Figure 4: K value vs. Accuracy Score Plot for Training   **Figure 5: K value vs. Accuracy Score Plot for Testing**

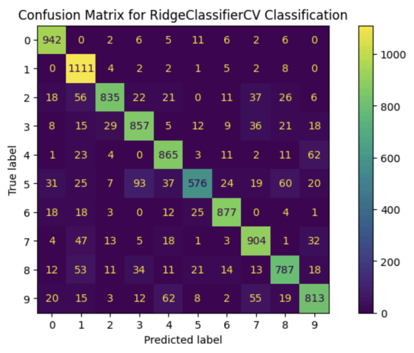
Classification Method	Training Accuracy Score	Training Standard Deviation	Testing Accuracy Score	Testing Standard Deviation	Difference between Training and Testing in terms of Standard D.	Training Accuracy Score before cross validation
Ridge	**0.8440666**	**0.0095985**	**0.8417999**	**0.0304837**	**0.2361514**	**0.8455000**
KNN	**0.9615**	**0.0018752**	**0.9262**	**0.0226773**	**18.824658**	**0.9828500**
LDA	**0.8653666**	**0.0085569**	**0.8641666**	**0.0294438**	**0.140237**	**0.8668833**

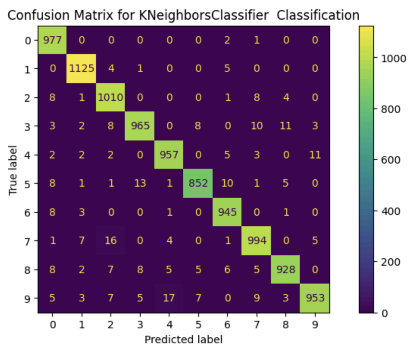
Table 2. Accuracy Scores and Standard Deviations obtained from Ridge, KNN, and LDA Classification Methods

Accuracy scores and standard deviations obtained through the Ridge, KNN, LDA classification methods for both training and testing subsets, This time each subset is containing all digits ranging from 0 to 9 .Cross-validation was performed in the final step with cv=5. Results are presented in table 2.

-For both training and testing, KNN classification has the highest accuracy score, followed by LDA and then Ridge, suggesting KNN classification performs the best as classification method for our data set.
-But KNN gives the highest difference between Training and Testing in terms of its Standard deviation, this shows that training and testing solutions are inconsistent. There is overfitting since the Training accuracy score is higher than the testing accuracy score. The data sets must be improved to capture a wider variety of scenarios.
-The accuracy scores of all subsets experienced a decrease following cross-validation for all methods, but KNN experienced the highest decrease rate, suggesting it is really sensitive to how the data is trained.
-Ridge and LDA methods have less than 1 difference between Training and Testing in terms of their Standard deviation. This shows the training and testing are consistent and trustworthy.

The confusion Matrices for 3 methods are shown in the below figure (6,7,8) The non diagonal elements represent confusion within classification. Notice how KNN confusion matrix has the least non diagonal element with respect to other two classification methods confusion matrices. This further verifies our point we made previously.





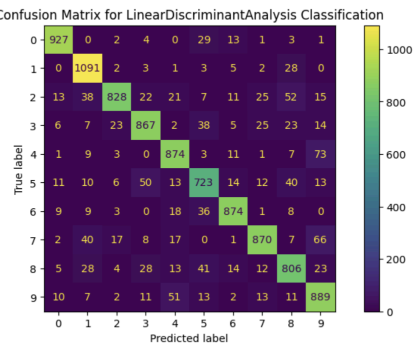


Figure 6:Ridge Classification Confusion Matrix **Figure 7: KNN C. Confusion Matrix** **Figure 8: LDA C. Confusion Matrix**

Summary and Conclusion

I successfully developed an algorithm to train classifiers for recognizing images of handwritten digits using the well-known MNIST dataset, achieving high accuracy rates. When applying the Ridge classifier to various subsets, including only specified digits, I observed that the classifier faced challenges and exhibited lower accuracy when distinguishing between similar-looking handwritten digits, such as 3 and 8, compared to other pairs like 2 and 7 or 1 and 8. After employing three different classification methods (Ridge, KNN, LDA) on my dataset projected onto k-PC modes, encompassing all the digits, I found that the KNN method with k=1 yielded the highest accuracy scores for both testing and training. However, the dataset experienced overfitting with KNN classification, leading to inconclusive testing and training accuracy scores when compared to other classification methods. I further validated my findings using confusion matrices and I also recognized the importance of cross-validation in every subset I utilized and for every classification method.

Acknowledgements

I would like to thank my classmates Po, Shavey, Emoji for useful discussions and Professor Eli for his notes, lectures, Office Hours and code examples

References

[1] Eli Shlizerman (2024). 582 KutzBook. AMATH 582 Wi 24: Computational Methods For Data Analysis.