HW3: MNIST DIGIT CLASSIFICATION

TIANBO ZHANG

University of Washington, Department of Applied Mathematics, Seattle

ABSTRACT. This report details the implementation of various machine learning algorithms, including Principal Component Analysis (PCA) for dimensionality reduction, followed by the application of several classifiers to distinguish between digits from the MNIST data set. We will test our algorithms and and show the predication performance at the end.

1. Introduction and Overview

Introduction: The MNIST dataset, comprising thousands of handwritten digit images, presents a benchmark challenge in the field of machine learning and pattern recognition. In this report we will outlines several machine learning approaches to classify these digits with high accuracy.

Overview: Our data set is the images of handwritten digits from the famous MNIST data set. The data set is split into training and test sets. We will train our classifiers using the training set while the test set is only used for validation/evaluation of the classifiers. We will perform our analysis with the following steps:

(1) Reshape each image into a vector and stack them into X_{train} and X_{test}. Then perform the PCA analysis on the train set and graph the first 16 PC modes.
(2) Inspect the cumulative energy and find the amount of PCA modes needed for 85 percent energy approximation.
(3) Write a function to selects the subset for particular digits and return the subsets as subtrain and subtest matrices.
(4) Apply Ridge classifier to distinguish between the following sets: $\{1,8\}$, $\{3,8\}$, $\{2,7\}$ and compare the results.
(5) Perform multi-class classification with Ridge, KNN and LDA classifiers on the full train set and validate with test set.
(6) Use Random Forest classifier with linear kernel to classify our data set and validate the result.

2. Theoretical Background

2.1. **Frobenius norm.** Frobenius norm is a matrix norm (not a induced norm) be defined as following:

$$||A||_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m}|a_{ij}|^2} = \sqrt{Tr(A^TA)} = \sqrt{Tr(AA^T)}$$

2.2. **Principle Component Analysis.** PCA aims to measure the variance of the data in each axis that is centralized(mean 0). It can also find a decent solution to the "least number of components to summarize data". We can start by considering the covariance matrix:

$$C_x = Cov(x) \cong \frac{1}{N-1}XX^T$$

Then plug in the SVD of X we have:

$$C_x \cong \frac{1}{N-1}U\Sigma V^T V\Sigma U^T = \frac{1}{N-1}U\Sigma^2 U^T$$

Then let $Y = UX$, we have:

$$C_Y = \frac{1}{N-1}YY^T = \frac{1}{n-1}\Sigma^2$$

Then note PC modes are eigenvectors of C_x and σ^2 are eigenvalues of C_x. There fore, PC modes indicate the "optimal" directions to represent variance of the data.

2.3. **Cross Validation.** Cross-validation is the process of separating the training data after shuffling and compute the average error/accuracy among them. The formal process is as following:

(1) Shuffle the data.
(2) Divide into k subsets.
(3) Train each subset and validate the other subsets using the trained model.

2.4. **Ridge Classifier.** Ridge classifier is a ML algorithm designed for multi-class classification tasks. It applies L2 regularization to penalize large coefficients in the model to prevent over-fitting, enhancing its generalization to new data. The classifier minimizes the squared loss with an added L2 penalty on the magnitude of the coefficients, given by the formula:

$$\hat{\beta} = argmin_{\beta\in\mathbb{R}^J}\frac{1}{2\sigma^2}||X\beta - y||^2 + \frac{\lambda}{2}||\beta||_p^p$$

2.5. **KNN Classifier.** KNN classifier is an algorithm operates on the principle that similar data points are often in close proximity to each other. For classification, kNN assigns a class to a new data point based on the majority class among its k nearest neighbors in the feature space. The distance is usually calculated using Euclidean distance as following:

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

2.6. **LDA Classifier.** LDA classifier is based on logistic regression. LDA aims to project the data onto a lower-dimensional space that maximizes the class separation by finding the projection that maximizes the ratio of between-class variance to within-class variance.

3. **Algorithm Implementation and Development**

3.1. **PC Modes Analysis.** First we will compute the PC modes for our data set. Then we will determine the amount of PC modes needed to approximate 85 percent energy.

Algorithm 1 PC Modes Determination

```
Compute the PC modes using PCA package from Sklearn
Compute the energy for each row by taking the square of each
singular value and divide by the sum of them
Set the cumulative energy to zero
for each PCA component's energy do
   Compute the cumulative energy up to this PCA mode
   if cumulative energy is bigger than the targeted percentage then
      Return the index of the PCA mode
   end if
end for
```

3.2. **Find Subsets.** In order to find the subsets for train and test data set, we can just take the samples with the targeted digits in the labels data set. We can implement the following algroithm:

Algorithm 2 Find subsets

```
Find the indices of the targeted digits in train and test labels
Collect the data with the calculated indices
Put all the collected train data and test data into two subsets as
$X_{subtrain}$ and $X_{subtest}$
Generate $y_{subtrain}$ and $y_{subtest}$ by taking the amount of the digits in
train labels and plu them back in
```

3.3. **Train Classifiers and Cross-Validation.** Before we train the classifiers we have to project our X data set onto PC modes space. Taking the amount of PC modes we found in Algorithm 1 and project X onto the k-PC modes space, we can then proceed to train our classifers:

Algorithm 3 Train Classifiers

```
Project $X_{train}$ and $X_{test}$ onto k-PC modes space using fit transform
function
Set up the classifier with model = some classifier from SKlearn
for each values in the hyper parameter do
   Fit the model with $X_{train}$ using the hyper parameter value
   Calculate the cross validation scores using function from
Sklearn
   Compute the mean and the standard deviation of the scores
   Report the results and append the results to the scores list
end for
```

4. **Computational Results**

4.1. **PCA Analysis.** After reshape and stack our data to get our X_{train} and X_{test} matrices we can calculate our PC modes for the data set. Then we can plot the first 16 PC modes as 28*28 images as following:

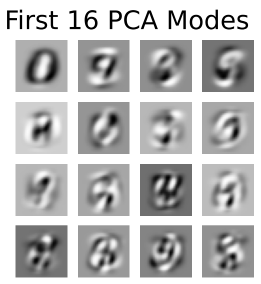


FIGURE 1. First 16 PCA modes as 28*28 images

Note from the figure 1 shows the most significant variations with in the MNISST data set's digit images. Each PC mode represents a specific pattern or feature that differentiates the images in the dataset, such as edges, curves, and specific digit shapes. For example for the first PC mode, it shows a somewhat round characteristic like the digit 0. Observing these digits we can see that these modes highlights the structures that contribute most significantly to their variation.

4.2. **Cumulative Energy.** Then after the computation of the PCA modes we can find the amount of PCA modes we needed in order to keep the approximate X_{train} up to a 85 percent by computing the energy for each mode and cumulatively add them up. Implementing algorithm 1, we will have the following graph.

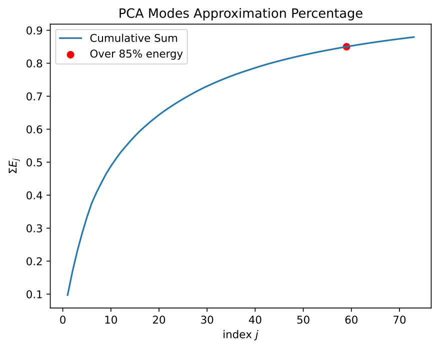


FIGURE 2. Cumulative energy for PCA modes

As we can see from figure 2 we can see that we need 59 PCA modes in order to keep the approximate energy up to 85 percent. Hence we will use 59 PC modes for the projection space of X_{train}, and use this projection for the training of several different classifiers.

4.3. **PCA Projections.** We can also check the PCA projections of X_train to make sure that it make sense to use our PC modes.

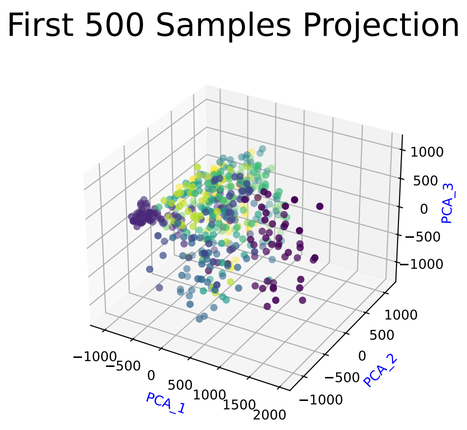
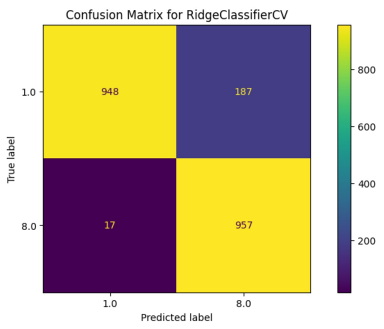


FIGURE 3. PCA projection onto 3-PC modes space

We can see that our data shows a clustering pattern in the 3 PC modes space. Hence we are on the right track. However, from the figure 3 we can see that we still need more PC modes to capture the structure of our data.

4.4. **Ridge Classification on Pairs.** After selecting subsets from our original data set using algorithm 2 and project them onto 59 PC modes space, we can then perform ridge classifications on them with algorithm 3. We have the following results:

Digits	Train Score	Test Score	Cross-Val
$[1,8]$	0.9042	0.9056	0.9021 with 0.1345 std
$[3,8]$	0.9379	0.9441	0.9366 with 0.0055 std
$[2,7]$	0.9563	0.9733	0.9555 with 0.0468 std



As we can see from the table that the digits [2,7] has the highest accuracy scores. One reason is that the structure difference between the digit 2 and 7 is captured the most by 59 PC modes. Hence, the result shows the highest accuracy for digits [2,7]. Then we can see that the accuracy score we got for digits [1,8] is not as high as the other digits pairs. One reason is that the digits 1 and 8 may have more similar features or overlap in their handwritten forms than other digit pairs, leading to higher confusion or misclassification rates.

4.5. **Classifiers on Whole Data Set.** Then we can perform KNN, Ridge and LDA classification on all digits. We will perform cross validation on all the different classifiers and try to find the optimal hyper-parameters for all the algorithms. Then we have the following tables:

Classifier	Train Score	Test Score	Highest Cross-Val
Ridge	0.8454	0.8561	0.8441 with 0.0098 std
KNN	0.9867	0.9705	0.9701 with 0.0012 std
LDA	0.8715	0.8729	0.8630 with 0.0073 std

Note from the table we can see that KNN classifier has the highest traning and test scores, as well as the cross-validation score. Thus KNN classifier performs the best out of the three methods. KNN's effectiveness here could be attributed to its ability to capture the local structure of the data, which can be particularly beneficial in a dataset with well-separated classes like the digits in MNIST.

LDA classifier shows a slightly better score compared to Ridge classifier. This suggests that LDA might be more effective in this multi-class classification context, potentially due to its ability to maximize class separability.

5. **Summary and Conclusions**

This report has detailed the execution of multi-class classification on the MNIST dataset using various machine learning algorithms, namely Ridge, k-Nearest Neighbors (KNN), and Linear Discriminant Analysis (LDA). The initial task involved the application of Principal Component Analysis (PCA) for dimensionality reduction, identifying the number of components required to capture 85 percent of the variance within the data. Subsequently, the reduced dataset was utilized to train different classifiers to recognize and differentiate between the ten digits (0-9).

In the classification tasks, the performance of the Ridge classifier was found to be satisfactory, with KNN demonstrating superior results in both the training and testing phases, as well as in cross-validation scores. LDA offered a slight improvement over the Ridge classifier, indicating its potential in maximizing class separability.

The study has also emphasized the importance of hyper-parameter tuning and model validation through cross-validation to prevent over-fitting and ensure the model's generalizability to new data. In the future, more classification methods could be studied to provide better feature extraction capabilities and classification accuracy.Additionally, investigating ensemble methods and deep learning approaches could offer further improvements in performance.

Through this comprehensive analysis, the report has contributed to the understanding of classification algorithms' behavior in high-dimensional spaces and set the stage for future advancements in the field of machine learning and digit recognition.

6. **Acknowledgements**

The author is thankful to Pro.Zhang from Beijing University of Post and Telecommunications for useful discussions about potential classification methods.

7. **References**

References

[1] J.N. Kutz (2013) *Methods for Integrating Dynamics of Complex Systems and Big Data*, Oxford.

[2] Alan V. Oppenheim, Alan S. Willsky (2022) *Signals and Systems*, 2nd Edition.

[3] Anshul Saini (2022) *An Introduction to Random Forest Algorithm for beginners*, Analytic Vidhya.

8. **Appendix (Extra Credit)**

For the alternative classifier we will implement the Random Forest Classifier which was not covered in class.

After train our model with the PCA projection of X_{train} we have the following result:

n estimators	Cross-Val	Standard Deviation	Time(second)
1	0.5246	0.0177	16.394
2	0.5334	0.0235	30.119
3	0.60803	0.0165	46.073
4	0.6683	0.0149	60.426

Note from the table we can see that the accuracy is indeed increasing as the hyper-parameter increases, however, the time used also increases. Thus we can conclude that although the accuracy may increases if we keep adding estimators, the algorithm is too time consuming.

One reason for this phenomenon is that The n estimators parameter controls the number of trees in the Random Forest. By increasing this number, you're essentially allowing the model to make decisions based on a larger ensemble of trees. This generally leads to better performance but also longer time.

In conclusion, although accuracy is important in machine learning algorithms, valuable resources such as time is also important.