HW2: Classifying Digits

Jessica Jones

Feb. 11, 2022

Abstract

In this work, we use the MNIST dataset of handwritten digits to analyze different types of classifiers. Preprocessing the data consisted of decomposition via Singular Value Decomposition (SVD) and Principal Component analysis (PCA). Classification was done linearly via Ridge Regression and assessed by mean-square error (MSE) of training and testing session.

1. Introduction and Overview

We are given a list of images containing handwritten digits and are tasked with developing a classifier to recognize the digits. We perform the singular value decomposition which helps us determine the rank of the data. Then, we project the data onto 16 principal component modes to help visualize the dataset. Once we have done this, we perform Ridge Regression Analysis by projecting our data onto the training set. We then test out various combinations of digits seeing how accurate our classifier is at discriminating between pairs of digits..

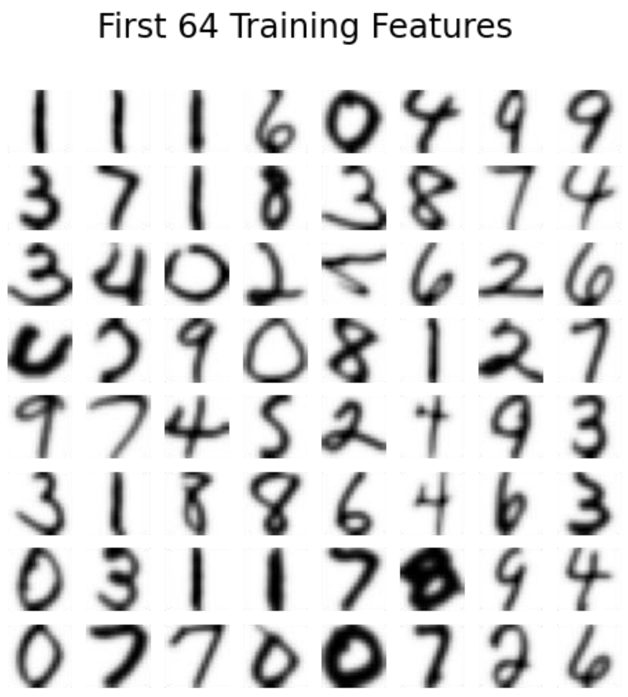

Figure 1: Images of the First 64 Training Features

2. Theoretical Background

2.1. SVD

The Singular Value Decomposition (SVD) is a matrix factorization. If A is an $n \times m$ matrix, then we may write A as a product of three factors:

$$A = U\Sigma\, \mathrm{V}^{\mathrm{T}}$$

(1)

where U is an orthogonal $n \times n$ matrix, V is an orthogonal $m \times m$ matrix, V^T is the transpose of V, and σ is an $n \times m$ matrix that has all zeros except for its diagonal entries, which are nonnegative real numbers.

Using the SVD to perform PCA makes much better sense numerically than forming the covariance matrix to begin with

2.2. Singular Values

Let A be an $m \times n$ matrix. Consider the matrix $A^T A$. This is a symmetric $n \times n$ matrix, so its eigenvalues are real. Let $\lambda_1, \ldots, \lambda_n$ denote the eigenvalues of $A^T A$, with repetitions. Order these so that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$. Let $\sigma_i = \lambda_i$, so that

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0.$$

(2)

The numbers $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$ defined above are called the **singular values** of A. The number of nonzero singular values of A equals the rank of A. The largest singular value is equal to the operator norm

2.3. Frobenius norm

Matrix norms and singular values have special relationships. The Frobenius norm is a matrix norm of a matrix defined as the square root of the sum of the absolute squares of its elements. The Frobenius norm of a matrix A is also equal to L2 norm of singular values

$$||\mathrm{A}||_F = \sqrt{\sigma_1^2 \ \ldots \ \sigma_r^2}$$

(3)

where σ are singular values of A, i.e. diagonal elements of σ in the SVD. In the context of this assignment, the identity for the Frobenius Norm is given as such:

$$||\mathrm{A}||^2{}_F = \sum_{j=1}^{min\{m,n\}} \sigma_j\,(A)^2 \ , \ \mathrm{A} \in \mathbb{R}^{mxn}$$

(4)

2.4. PCA

PCA is given by the same SVD when the data are centered. This method reduces the dimensionality of a data set consisting of large number of interrelated variables, while retaining as much as possible of the variation present in the data set. This is achieved by transforming to a new set of variables, principal components (PCs), which are uncorrelated and ordered so that the first few retains most of the variation present.

If A is a square matrix then a nonzero vector v is an eigenvector of A. Assume that v is an eigenvector of A. Then $v \neq 0$, and we have:

$$Av = \lambda v$$

(5)

for some scalar $\lambda \in R$. Let $c \in R$ be any non-zero scalar. Then cv is a non-zero vector, and we have:

$$A(cv) = cAv = c\lambda v = \lambda(cv).$$

(6)

This shows that cv is an eigenvector of A (with eigenvalue λ).

3. Algorithm Implementation and Development

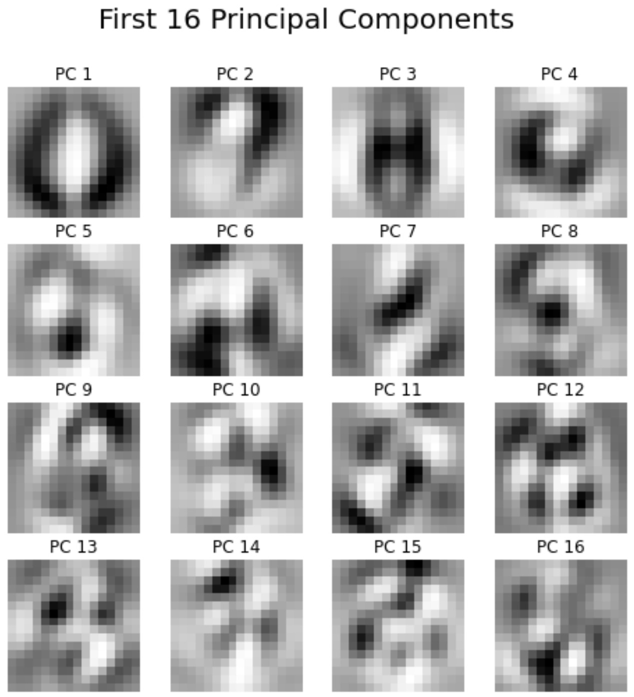

Figure 2: First 16 Principal Components Visualized.

3.1. Preprocessing

Our first step is to load the MNIST training and test datasets and labels. This was done via np.load() function and indexed the appropriate training and test datasets. This gave matrices

$$X_{train} \in \mathbb{R}^{2000\,X\,256},\ Y_{train} \in \mathbb{R}^{2000}$$

to denote the matrix of training features and the vector of training labels *respectively*. Likewise,

$$X_{test} \in \mathbb{R}^{500\,X\,256}, \qquad Y_{test} \in \mathbb{R}^{500}$$

denote the matrices of test features and the vector of testing labels *respectively*.

We next used PCA to investigate the dimensionality of X_{train} and plotted the first 16 PCA modes as 16 X 16 images shown in Figure 2.

The first could be interpreted as a zero or an eight, whereas the second seems to look like a two, and the third principal component could also be an eight. It becomes harder to identify digits as the components go on.

We next asked how many PCA modes do we need to keep approximating X_{train} up to 60%, 80% and 90% in the Frobenius norm. Recalling the identity seen in $Equation$ (4), we found that to approximate up to 60%, 80% and 90% of the images in the Frobenius norm, we only require **3, 7, and 14 principal components** to approximate X_{train} *respectively*, seen in the right plot in Figure 3. I compared this with doing eigen decomposition of transformation matrix (Covariance matrix created using X_{train}). To do this, I:

1. Scaled the dataset, which is very important before you apply PCA
2. Instantiated PCA using `PCA()` using `sklearn`
3. Determined transformed features using `pca.fit_tranform()`
4. Determine explained variance `using explained_variance_ration_` attribute
5. Determined the cumulative sum of eigenvalues; This will be used for visualizing the variance explained by each principal component.

The results are shown in the left plot in Figure 3. This method yielded **7, 15, and 26 principal components** to approximate X_{train} up to 60%, 80% and 90%, *respectively*.

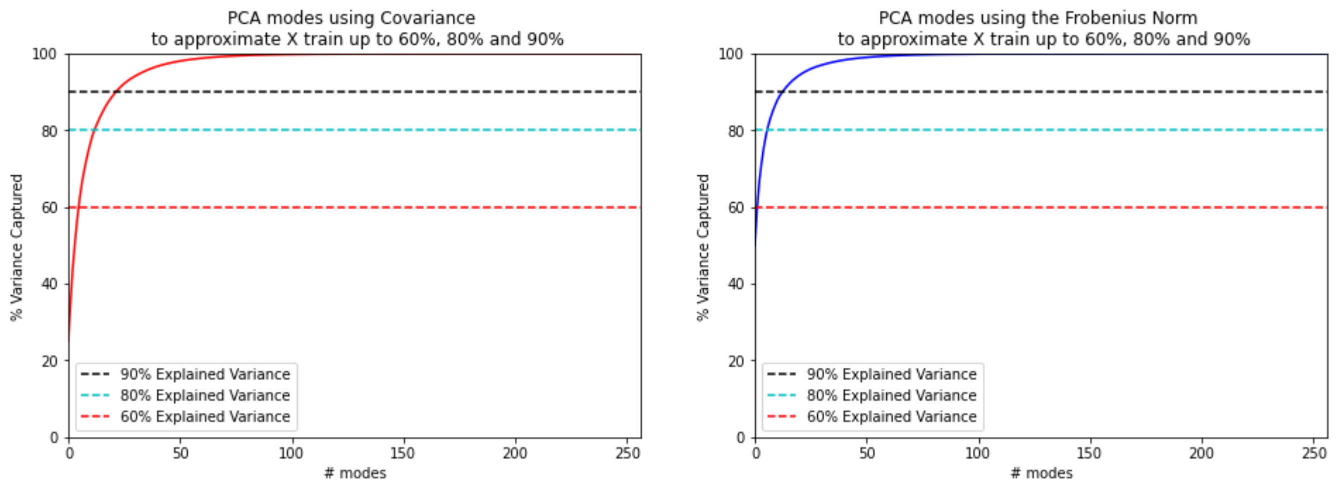

Figure 3: PCA modes using Covariance and Frobenius Norm methods to Approximate X_{train}

3.2. Digit Classification

We then trained a classifier to distinguish the digits 1 and 8 via the following steps:

1. Wrote a function `grabDigits()` that extracts the features and labels of the digits 1 and 8 from the training data set. Let us call these $X(1,8)$ and $Y(1,8)$.
2. Wrote a function `reduceData()` that projected $X(1,8)$ on the first 16 PCA modes of X_{train} computed in step 1. This gave a matrix A_{train} which has 16 columns corresponding to the PCA coefficients of each feature and 455 rows corresponding to the total number of 1's and 8's in the training set.
3. Wrote a function `ridgeModel()`. In it I first assigned label -1 to the images of the digit 1 and label +1 to the images of the digit 8. This should result in a vector $b_{train} \in \{-1,+1\}^{455}$.
4. I then used Ridge regression to train a predictor for the vector b_{train} by linearly combining the columns of A_{train}. I reported the training and testing mean squared error (MSE) of this classifier

4. Computational Results

I wanted to find out which digits were easiest and most difficult to differentiate. To start, I ran my classifier on $X(1,8)$, $X(3,8)$, and $X(2,7)$. I used 16 principal components for training in all three cases.

After training the two-digit classifier, the most difficult digits to differentiate were 3 and 8, with Training MSE of 18.04% and Test MSE of 57%. Training and Test Percentage Correct between $X(3,8)$, were 98.6% and 83.3%, *respectively*. I suspect this is due to 3 and 8 looking almost identical in nature, with similar curvature and vertices.

$X(1,8)$ were easily differentiated, with Train MSE = 7.4%, Train Percent Correct = 99.5%, Test MSE = 13.5%, and Test Correct = 97%.

$X(2,7)$ performed similarly, with Train MSE = 9.2%, Train Percent Correct = 99.4%, and Test Correct = 96%. Test MSE for $X(2,7)$ was higher, however, with 37.4%. I suspect that this is because 2 and 7 look similar in design.

To further investigate performance, I plotted the Prediction Accuracy in each test. This visualized b_{test} in comparison to the predicted value of b_{test}. Its visually apparent that the $X(3,8)$ classifier had a harder time differentiating the two digits.

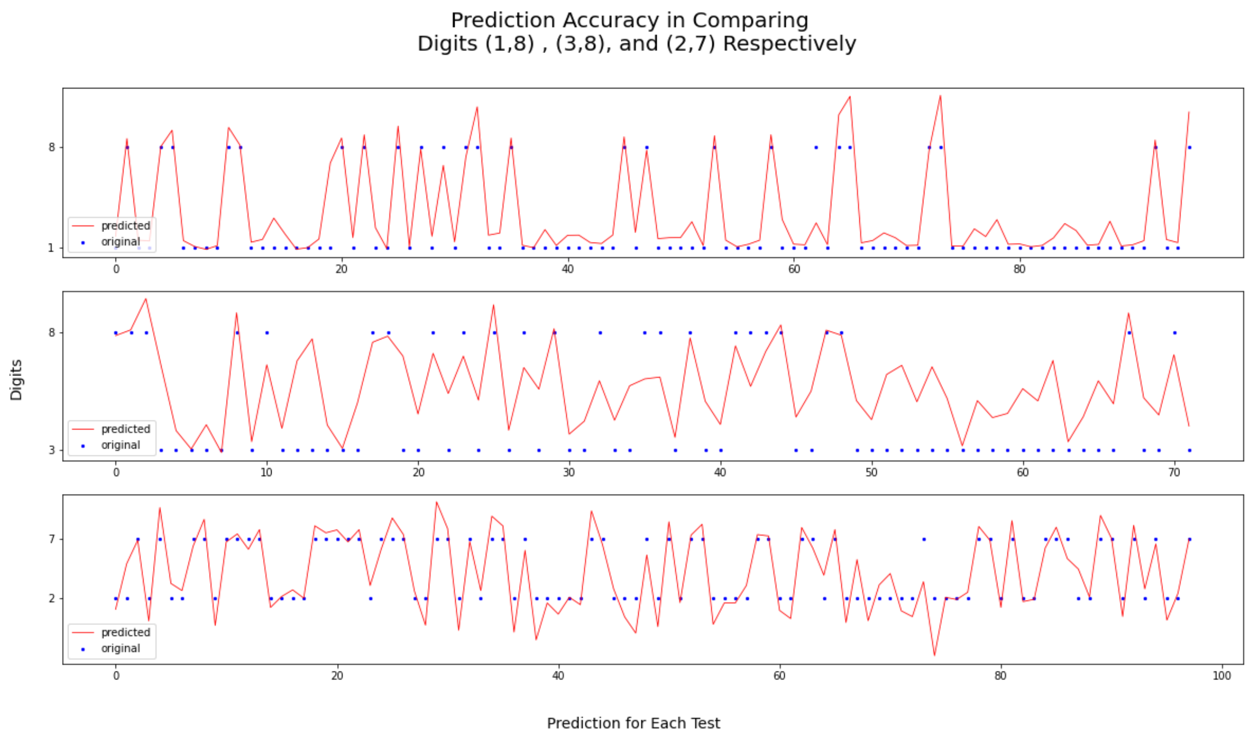

Figure 4: Prediction Accuracy from Ridge Regression Analysis When Comparing Three Sets of Two-Digit Combinations

5. Summary and Conclusions

Classifying two digits with very few components using Ridge Regression from PCA still yielded decent training and test accuracies, but the classifier breaks down when comparing very similar digits (in this case, 3 and 8). It would be worthwhile to tune the alpha values associated with the model to better fit the data, and to also try another classification method. For example, instead of PCA, the classifier might do better using Linear discriminant analysis (LDA), which is a technique where data is projected onto a new basis, usually a line, where it can more easily be analyzed. Similarly, a decision tree classifier could be used as a supervised machine learning algorithm that classifies examples by continually splitting the data according to some parameters.

6. Acknowledgements

Thank you Stack Overflow (https://stackoverflow.com), Anna Li for emotional support, and other anon students for clarifying questions.

References

Lay, D. C., & Kent Nagle, R. (2012). Linear Algebra & Differential Equations, Second Custom Edition for University of California, Berkeley (Math 54 - 2013). Pearson; 2nd Edition (January 1, 2012). https://math.berkeley.edu/~hutching/teach/54-2017/svd-notes.pdf