AMATH 482/582: HOMEWORK 2

SATHVIK CHINTA

Abstract. Using PCA analysis with Ridge Regression, we were able to identify handwritten digits within the MNIST dataset to a very high degree of accuracy.

1. Introduction and Overview

This time, we have a very interesting (and fun) problem to solve. We want to create a model that can identify hand-written digits! The data itself is already split into training and test data, so we won't have to worry about that. Taking a look at the data itself, we can see that there are 2000 images in the train data and 500 in the test, each with 256 modes. Knowing that we most likely won't need to use all the modes to represent our data, we can think about using PCA analysis in order to figure out how many modes we truly need.

2. Theoretical Background

Imagine we are given some dataset in a very high dimensional space. We want to apply a model to identify classes on said dataset, but the complexity of multiple higher dimensions makes this task very difficult. This is where PCA analysis comes in. We check the covariance of each dimension to the other dimensions and then find the eigenvalues and eigenvectors of the covariance matrix. We then sort the eigenvalues in descending order and use the corresponding eigenvectors to project our dataset into a lower dimensional space. This is called PCA analysis. It allows us to also know the variance of each dimension and how much of the variance is explained by each dimension. So, if we know that the variance of some dimension is 20 percent of the total variance, we can say that 20 percent of the data is explained by that dimension. Knowing this, we can remove the dimensions that are not important to our model. This is called dimensionality reduction. Our current dataset has 256 modes of data, so we should perform PCA analysis and see how many of these dimensions we can remove and still have a good model.

We can attempt to reduce dimensionality according to the Frobenius norm, which is the square root of the sum of the squares of the eigenvalues. This is usually a good metric for determining how many dimensions we should keep, and we can pass in how much variance we would like to keep in order to do so.

Once we do this, we can take two numbers from our dataset (we will take the numbers (1,8), (3, 8), and (2, 7)), assign either +1 or -1 to each of them, and then use the model to predict the class of the new data. We can then compare the predicted class to the actual class

Once we have done so, we can use Ridge regression to find the best weights for our model. We can then use the weights to predict the class of a new data point within our dataset. In Ridge regression, we estimate the weights and assume that the dimensions are highly correlated with each other. This is why it's so important to perform PCA analysis before hand. In ridge regression, we attempt to minimize the following equation:

$$minimize_{\beta}||A\beta - Y||^2 + \lambda||\beta||^2$$

Date: February 11, 2022.

Where A is our prediction, y is the true value and λ is our regularization parameter.

We can then see how well our model does on the train data, and apply that same model to the test data to see how well it does on unseen data.

3. Algorithm Implementation and Development

I used Python for this project. The major package that I used was sklearn, but I also used numPy and matplotlib.

Within sklearn I used the following functions:

`sklearn.decomposition.PCA()` to perform PCA analysis.
`sklearn.singular values` to find the singular values of the data.
`sklearn.decomposition.PCA().fit()` to fit the PCA model to the training data.
`sklearn.linearModel.Ridge()` to initialize the ridge regression model.
`sklearn.metrics.meanSquaredError()` to find the mean squared error of the model.
`sklearn.linearModel.Ridge().fit()` to fit the ridge regression model to the training data.
`sklearn.linearModel.Ridge().predict()` to predict the class of a new data point.

Within numPy I used the following functions:

`np.load()` to load in both the train and test data.
`np.cumsum()` to get the cumulative sum while looking at the total variance explained by each dimension.

Within matplotlib I used the following functions:

`plt.plot()` to plot all graphs/data

4. Computational Results

When we graph out the singular values after doing the PCA analysis, we can see the following fraction of the total variance:

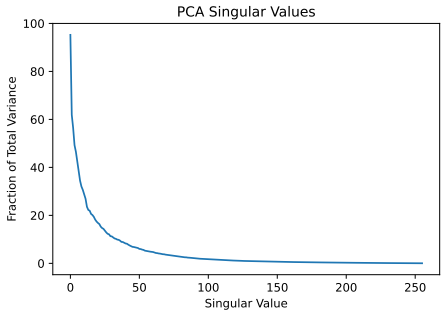

Figure 1. The fraction of the total variance explained by each dimension.

We can see that the first few dimensions are the most important. It looks like after 150 dimensions, the variance is tapered off. This means that the data is not very well represented by the last dimensions. We can paint a better picture by looking at the cumulative sum of the variance as well

Our original analysis seems to be correct. We can see that the first few dimensions are the most important in terms of their increase. Once we hut 100 principal components, we're above 80 percent of the variance explained.

Looking at the first 16 modes (which according to our graph should be somewhere between 20-40 percent variance), we get the following

We can see that these images aren't very decipherable. The third image from the top looks to be an 8, but the rest of the images are murky. We should keep in mind that when we reduce dimensionality, we lose information.

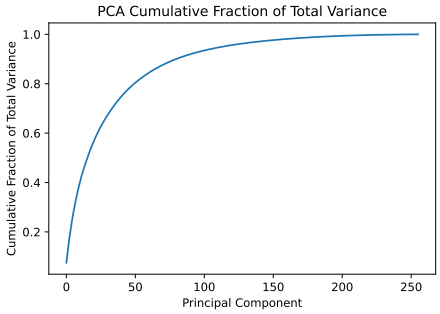

FIGURE 2. The cumulative fraction of the total variance explained by each dimension.

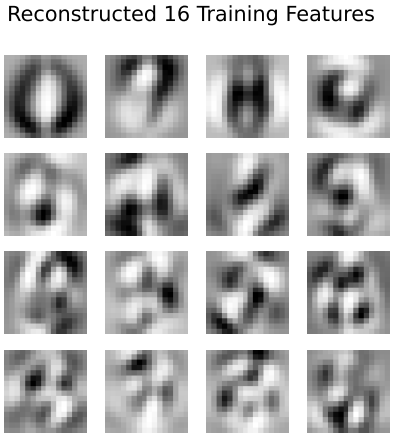

FIGURE 3. The first 16 modes of the training data, applied on 16 sample images.

Now, we can try to explain the variance according to the Frobenius Norm. I got the following values:

Variance Explained by the Frobenius Norm	Number of Modes Required
60 percent	3
80 percent	7
90 percent	14

TABLE 1. Variance Explained by the Frobenius Norm and their associated Modes Required.

We can see that the number of nodes required to explain the variance is increasing with the increase in variance. Surpisingly, we only need 14 modes to get to 90 percent explained variance. Let's use the first 16 modes to see how accurate of a model we can train.

When identifying the digits, I got the following results:

We can see that the lowest MSE is for the 1 and 8 pair for both training and test data. This means that the model is very accurate. We may think at first that the model is overfitting, but we can see that the train and test data are very close for the individual pairs. This means that we are actually not overfitting, but there is a difference between the pairs.

Digits	MSE
1 and 8 Train	0.0745963381938405
1 and 8 Test	0.08347453537941685
3 and 8 Train	0.18039681693239826
3 and 8 Test	0.2581632921805704
2 and 7 Train	0.09177944633086439
2 and 7 Test	0.13722034799081978

TABLE 2. Mean squared error of the model on the training and test data for different pairs of digits

One possible explanation may be due to the similarity between the way that digits are written. Recall from earlier that when we reduce dimensions, we lose information. This means that some digits may look very similar when compared to one another. For instance, 3 and 8 are very similar in the way that they are written. This is why they have the highest MSE out of all the pairs. 2 and 7 both have a diagonal line in the middle of them, so they are also similar in that fasion. This similarity is not as clear as that between 3 and 8, but the MSE is also not as high as that of 3 and 8. 1 and 8 are not very similar, so they have the lowest MSE.

5. Summary and Conclusions

Even with just a simple PCA analysis model with Ridge Regression, we can make a model to classify digits to a very high degree of accuracy. Further improvements to the model can be to classify all digits at once instead of just pairs, as well as to introduce more complicated models such as convolutional neural networks.

Acknowledgements

I am thankful to Professor Hosseini for introducing us to the concept of dimensionality reduction, PCA analysis, and different regression models. I am also grateful to the YouTube Channel "StatQuest with Josh Starmer" by Josh Starmer for providing a visual explanation for the intuition behind PCA dimensionality reduction, and how we can use it to reduce the dimensionality of high-dimensional data.

I am very thankful to my peers taking the class alongside me, they have helped me understand the material as well as provide a reference to compare my results against. I interacted with them both through Canvas discussion boards as well as Discord chat.

References

[1] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020.

[2] B. Hosseini. Evaluating sl models. University of Washington (LOW 216), Jan 2022. AMATH 482/582.

[3] B. Hosseini. Introduction to machine learning. University of Washington (LOW 216), Jan 2022. AMATH 482/582.

[4] B. Hosseini. Principle component analysis. University of Washington (LOW 216), Jan 2022. AMATH 482/582.

[5] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[5] [1] [4] [3] [2] [6]