COMPUTATIONAL METHODS FOR DATA ANALYSIS CLASSIFYING DIGITS

AVERY LEE

Applied Mathematics Department, University of Washington, Seattle, WA
leeave22@uw.edu
02/11/2022

Abstract. Classifying Digits creates a Machine Learning model that reads an input image of a digit from 0 to 9 and classifies the digit. It projects each image of the digit to the Principal Component Analysis (PCA) space, then uses a training dataset that contains images of digits and their labels to train the model, and a test dataset that contains another set of images to put into the trained model to test its accuracy. This paper will go through the mathematical theories behind the calculation methods, the algorithms used to create the predictor model, the final computational results, and conclusions.

1. Introduction and Overview

Machine Learning is a field of study that has been expansively growing due to its endless applications; it can be used to predict, classify, approximate, or even cluster any type of dataset [1]. This paper will cover an example of supervised learning that predicts and classifies data given a training and testing dataset, combined with unsupervised learning such as Principal Component Analysis that does dimensionality reduction [2].

In this context, there is a training dataset of two thousand 16x16-pixel images of digits from 0 to 9 as well as their true label. The model created will study a pair of digits at a time from the training set. Then using the testing dataset of five hundred images, the model will predict which of the two digits the input image portrays. During this process, the dimensionality of the training set, minimum number of PCA modes to reach different amounts of accuracy in the Frobenius norm, and the models' mean squared error will be studied [3].

Section 2 of this paper will go over the mathematical background necessary to understand the computations. Section 3 will cover the algorithms and Python software packages. Then Section 4 will summarize the results from the computations, and Section 5 will summarize the learnings.

2. Theoretical Background

2.1. **Machine Learning.**

Broadly speaking, Machine Learning (ML) can be categorized into two parts: supervised learning and unsupervised learning. The supervised learning portion will be done by creating a classification model that predicts and classifies data using training and testing sets. Unsupervised learning is modeling to find meaningful structure in a dataset to cluster or perform dimensionality reduction such as PCA.

As a simpler model that only studies a pair of digits at a time, the label "1" and "-1" will be used to label each digit, instead of the actual number value itself. This is common practice when observing two values, and more detail will be explained in Section 3.

An important part of any modeling procedure is observing the error of the outputs for both training and testing sets. For this case, it can be done by studying the mean squared error, calculated with this equation where b represents the labels as "1" or "-1", A is PCA coefficients, and $\hat{\beta}$ is coefficients of the fitted regression model:

$$MSE_{train} = \frac{1}{\text{length of } b_{train}} ||A_{train}\hat{\beta} - b_{train}||_2^2$$

$$MSE_{test} = \frac{1}{\text{length of } b_{test}} ||A_{test}\hat{\beta} - b_{test}||_2^2$$

A very low training MSE means nothing if the testing MSE is high. This means that the model fits the training data, but it fits too well to that particular dataset that it is not a good fit for a completely new testing set. So both MSEs must be observed.

2.2. **Principal Component Analysis (PCA).**

PCA is a dimensionality reduction technique that makes high dimensional data to low dimensional representation using fewer coefficients. This is done through Singular Value Decomposition (SVD) to create an orthogonal frame of reference.

Before working on the dataset, it should have a mean of 0, so the mean of the data will be subtracted from the dataset. The left singular vectors/values are the principal components which are the optimal basis. The covariance $Cov(X) = C_X$ is non-negative definite symmetric (NDS) and has the formula:

$$C_X = \frac{1}{N-1} XX^T$$

Since it is NDS, it has an eigen decomposition where the eigenvectors are the PCA modes.

$$C_X = Q\Lambda Q^T$$

If X is written as $X = U\Sigma U^T$, then this can be substituted to say that

$$C_X = \frac{1}{N-1} U\Sigma^2 U^T$$

The columns of U are the left singular vectors of X which are the principal components of C_X.

2.3. **Frobenius Norm.**

The Frobenius Norm goes by the function below and will be used in this example to test how many PCA modes are needed to approximate the training data up to 60%, 80%, and 90% in the Frobenius norm.

$$||B||_F^2 = \sum_{j=1}^{min\{m,n\}} \sigma_j(B)^2, \quad B \in \mathbb{R}^{m\times n}$$

3. Algorithm Implementation and Development

3.1. Python Packages and Notable Functions.

Python was used to compute the algorithms and produce plots. Below is a list of packages used.

Python Package or Function	Description
`google.colab, drive`	Gives access to the user's Google Drive account where they can access the data.
`scipy`	Helps with scientific computations and gives access to various built-in algorithms.
`scikit-learn [4]`	Provides tools for machine learning algorithms including linear_model and PCA.
`numpy [5]`	Allows usage of multidimensional arrays and functions to manipulate those arrays.
`matplotlib.pyplot [6]`	Useful for graphics manipulation.
`PCA()`	Performs PCA with linear dimensionality reduction using SVD.
`fit()`	Applies the dimensionality reduction to input values.
`transform()`	Applies the dimensionality reduction to input values.
`predict()`	Predicts the output using the linear model created.
`np.linalg.norm()`	Computes the norm of a matrix or vector.

Table 1. Python Packages and Functions

3.2. Setup.

The given data includes a training dataset X_{train} with 256 columns representing the pixels of an image, and 2000 rows each representing a whole image. The Y_{train} dataset is a vector of 2000 values, each representing the label of the digit from 0 to 9. The testing datasets X_{test} and Y_{test} are the same but only 500 images.

3.3. Dimensionality of the Training Dataset.

To study the dimensionality of X_{train}, it can be fit into a PCA model with all 256 components kept. Then, the singular values can be studied to see which of the 256 are important dimensions. A range of this can be done by plotting these singular values, and investigating where the values start to taper off. The values before the tapering off will tell what the dimensionality of X_{train} is.

To see things on a more relative scale, this can be normalized by finding the norm of the singular values, then using that to calculate $\frac{(\text{singular values})^2}{(\text{norm of singular values})^2}$. This is used to study the Frob Norm.

Also, images with just the first 16 PCA modes can be shown using the components of a PCA with 16 modes instead of 256. It should look more blurry than the original image.

3.4. Frobenius Norm.

Referring back to the Frobenius Norm equation in Section 2, different amount of PCA modes are needed to find different amount of accuracies in the Frobenius norm; more PCA modes are required for more accuracy in the Frobenius norm.

Getting the amount of PCA modes needed for 60%, 80%, and 90% in the Frobenius norm is simple as Python already has a function to find it with the percentage as the input value. Number of components of the X_{train} fit will be the number of PCA modes required. One thing to note is that

since the equation's left hand side states $||B||_F^2$ instead of $||B||_F$, the input parameter percentage must be $0.6^2, 0.8^2, 0.9^2$ instead of 0.6, 0.8, 0.9.

To double check if the output values make sense, a plot of the normalized PCA singular values mentioned in the previous section can be plotted to see if the 0.6, 0.8, and 0.9 marks match with the number of PCA modes.

3.5. **Training, Testing, and MSE.**

Now, in order to actually create the classifier model, a model has to be trained and the test dataset needs to go through that model. First, as mentioned in Section 2, the values will be studied in pairs. Two numbers are picked for each model, meaning a subset of the training data is extracted containing only these values. The labels Y_{train} is transformed into "-1" and "1". Using the PCA model created earlier using only 16 PCA modes, the X_{train} is transformed into the PCA space with just 16 modes. Using this, a regression model is created. One thing to note is that when creating the β vector mentioned earlier, the intercept of the regression model goes in the front, then the following coefficients.

The same processes is used for the testing dataset X_{test}. Except this time, instead of creating a regression model that fits the testing set, the same regression model created with the training set is used to predict the output with the testing set's input.

The MSE's are calculated with the formula mentioned in Section 2. Then the training and testing MSE's can be investigated to see if it is a good model with relatively low MSE. Of course it is technically possible that the model works for just a particular set of numbers, say 1 and 8, so more pairs of numbers can be tested to see if the model works with any type of pair.

The specific results of this process will be covered in the next section.

4. Computational Results

4.1. **Dimensionality.**

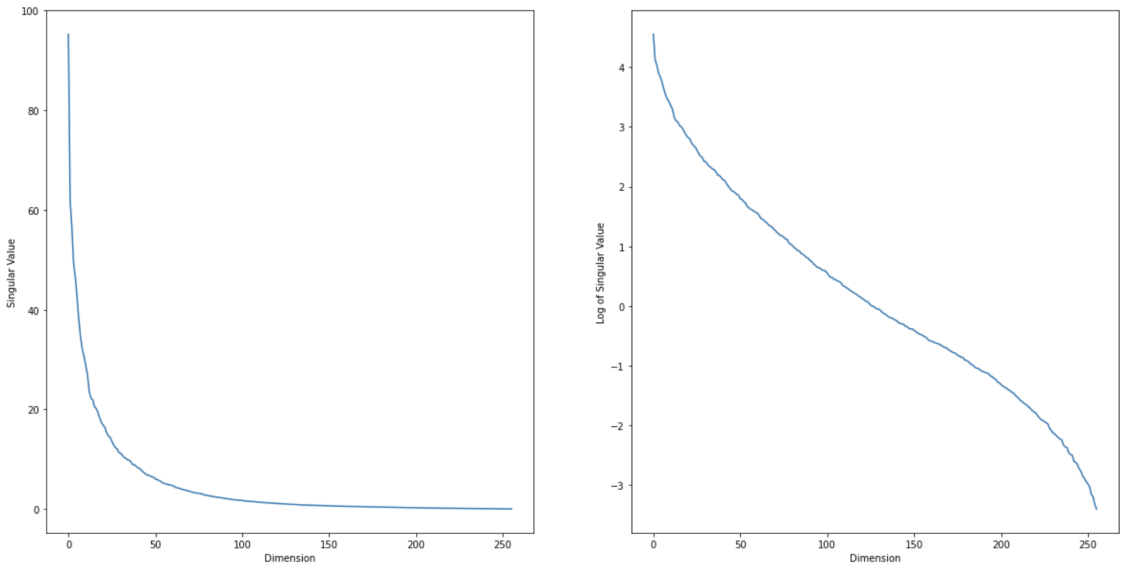

Figure 1. Singular Values of the Training Dataset

As shown in Figure 1, the log of the singular values do not have a clear tapering off point, so the raw values can be observed. From observation, it seems the values taper off approximately 30-45, meaning X_{train} can have a dimensionality of that much.

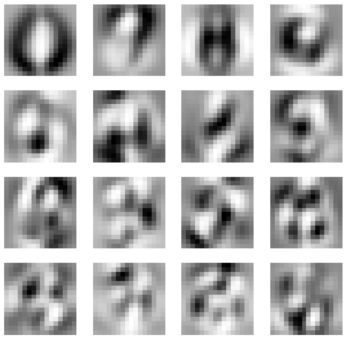

FIGURE 2. First 16 PCA Modes as 16×16 Images

Using the first 16 PCA modes, these images can be plotted. Instead of using all 256, this time only 16 are used. The images are blurrier than the original, as expected.

4.2. **Frobenius Norm.**

Using the calculation methods mentioned in Section 3, 3 PCA modes are required for 60%, 7 PCA modes are required for 80%, and 14 PCA modes are required for 90% in the Frobenius norm. Just to double check these results, this plot can be shown.

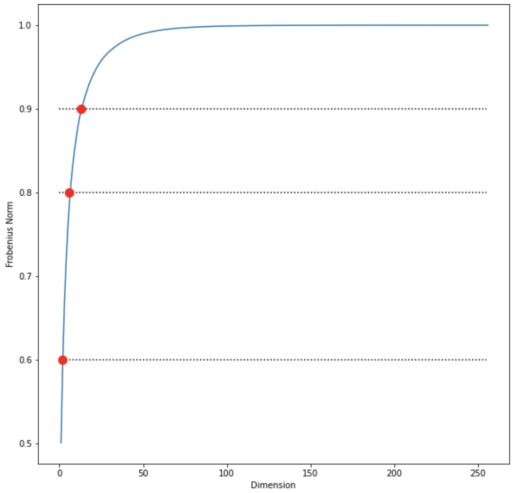

FIGURE 3. Frobenius Norm of Training Dataset

The red points mark the spot where the Frobenius norm reaches 60%, 80%, and 90%. Looking at the dimensions, it seems 3, 7, and 14 match up. To look closer, the below are the first 15 norms.

Nodes	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Frob Norm	0.50	0.59	0.66	0.71	0.75	0.78	0.81	0.83	0.84	0.86	0.87	0.88	0.89	0.90	0.91

TABLE 2. First 15 Frobenius Norms of Training Dataset

The 3rd node is where the Frobenius norm passes 0.6. The 7th node is where it passes 0.8. The 14th node is where it passes 0.9. So the results match. Since all 3, 7, and 14 are below 256, the entire 16x16 image is not needed for each datapoint.

4.3. **Mean Squared Errors of the Classifier Model.**

Using the model created with the training dataset, the MSE values can be found below.

Pair of Digits	Training MSE	Testing MSE
(1, 8)	0.0746	0.0832
(3, 8)	0.1804	0.2581
(2, 7)	0.0917	0.1364

TABLE 3. Mean Squared Error with Pairs (1,8), (3,8), (2,7)

For this example, the testing MSE is generally greater than the training MSE, which makes sense as the regression model is based on the training set, although this is not necessarily the case for any model. It can be predicted that the MSE differs between each pair of values because of the shape of the numbers themselves. For example, (1, 8) had a lower MSE because 1 is a straight vertical line whereas 8 is made of 2 circles on top of each other. (3, 8) had a greater MSE since 3 is shaped like the right half of an 8.

5. Summary and Conclusions

In this paper, PCA and Machine Learning techniques were used to create a classification model that observed images of digits and predicted the value of that digit. This model itself can be used to identify more than just digits. Images of a pair of anything can be used to train the model, then classify it. Of course, it won't be able to identify what exactly the object is. For example, it won't tell you it is a dog or a cat, but it will be able to say that this image is probably something, and this other image is not that thing. There are infinite potential uses for this simple model alone.

Acknowledgements

The author is thankful to TA Katherine Owens for the general rundown and clarifications on calculations and code, as well as classmates such as Pranav Natarajan and Ava Mistry that reconfirmed my data results and Python syntax. Last but not least, the author is thankful to Professor Bamdad Hosseini for teaching most of the mathematical concepts mentioned in the paper.

References

[1] Bamdad Hosseini. Introduction to machine learning. University of Washington (LOW 216), Jan 2022.
[2] Bamdad Hosseini. Principal component analysis. University of Washington (LOW 216), Jan 2022.
[3] Bamdad Hosseini. Evaluating supervised learning models. University of Washington (LOW 216), Jan 2022.
[4] 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] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.
[6] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.