# **Homework 2**

AMATH 482

Christian Diangco

## **Abstract**

This paper covers the theory and use of Principal Component Analysis and Supervised Machine Learning techniques in order to train a classifier to distinguish images of handwritten digits.The process is discussed in Sec. 3. Algorithm Implementation and Development, after which results and visualizations are displayed in Sec. 4. Computational Results.

## **Sec. 1. Introduction and Overview**

In this assignment, we are tasked with training a classifier that distinguishes images of handwritten digits. The data comes from the MNIST data set. We utilize Principal Component Analysis to accomplish this task. We look at the number of PCA modes needed to approximate the training data up to certain percentages. Ridge regression is then used to train a predictor and then obtain the Mean Squared Error of the classifier for both the training and test data.

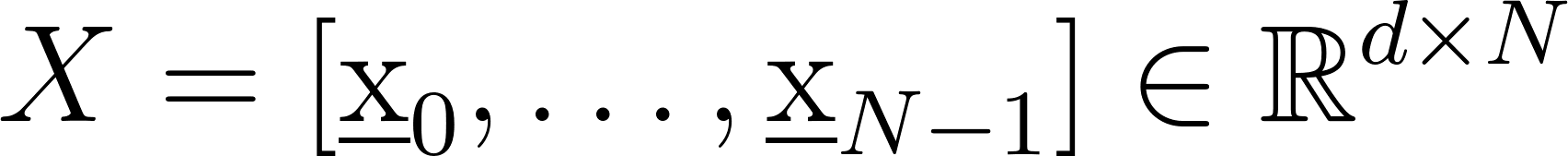
## **Sec. 2. Theoretical Background**

When decomposing data, whether it be signals, images, etc. we aim to create linear combinations of simpler pieces. If we have an image , then the image can be approximated as

,

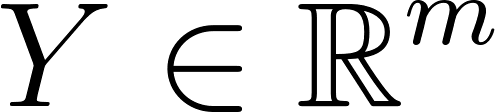
Where are real-value coefficients and are chosen functions. Principal Component Analysis (PCA) is one approach that aims to find the coefficients and at the same time. PCA is also seen as a technique for dimensionality reduction, which lets us represent high dimensional data with fewer coefficients.

If we have a dataset

[](https://latex-staging.easygenerator.com/eqneditor/editor.php?latex=X%3D%5C%5B%5Cunderbar%7Bx%7D_0%2C%5Chdots%2C%5Cunderbar%7Bx%7D_%7BN-1%7D%5C%5D%20%5Cin%5Cmathbb%7BR%7D%5E%7Bd%5Ctimes%20N%7D#0),

The left singular vectors of X are the principal components of the data [](https://latex-staging.easygenerator.com/eqneditor/editor.php?latex=%5Cunderbar%7Bx%7D_j#0) and the optimal basis in which the [](https://latex-staging.easygenerator.com/eqneditor/editor.php?latex=%5Cunderbar%7Bx%7D_j#0) can be represented.

In addition to PCA, this homework also requires us to utilize supervised machine learning techniques, In standard machine learning terminology, we consider a set of points of features or inputs, where [](https://latex-staging.easygenerator.com/eqneditor/editor.php?latex=X%20%3D%20%5Cmathbb%7BR%7D%5Ed#0). We can thus again think of as a matrix [](https://latex-staging.easygenerator.com/eqneditor/editor.php?latex=X%20%5Cin%20%5Cmathbb%7BR%7D%5E%7Bd%20%5Ctimes%20N%7D#0).

Associated with these features/inputs are a collection of outputs/responses . is often thought of as either a Euclidean space [](https://latex-staging.easygenerator.com/eqneditor/editor.php?latex=Y%20%5Cin%20%5Cmathbb%7BR%7D%5Em#0) or space of categories e.g. . The pair of input and output sets is referred to as a training dataset. The goal of supervised learning is, given the training dataset, predict responses/output for any new input/feature (.

## **Sec. 3. Algorithm Implementation and Development**

After loading both the training and test data, we save the features and labels of each to variables X\_train, Y\_train, X\_test, and Y\_test. The features are stored in the X variables and the labels are stored in the Y variables.

We create a PCA object (pca) using sklearn and fit the model using X\_train. We then use pca.singular\_values\_ and pca.components\_ to extract the singular values and principal components to create the visualizations for task 1.

For task 2, we first calculate the Frobenius norm as follows:

f\_norm = np.sqrt(sum(pca.singular\_values\_\*\*2)).

In order to find the number of PCA modes needed to approximate X\_train up to 60%, 80% and 90% of f\_norm we define a function like so:

function find\_num\_PCA(p):

pca = # Create PCA object with 1 component

# Fit pca with X\_train

i = 1

while Frobenius norm of PCA object < f\_norm \* p:

i += 1

pca = # New PCA object with i components

# Fit pca with X\_train

return i

We can then pass .6, .8, and .9 into this function in order to obtain the number of modes needed for each.

For task 3, we first create a function to extract the features and labels for specific digits (n1, n2) from the X and Y variables (either training or test). This function will also project the subset of X for n1 and n2 on the first 16 PCA modes of X\_train, assigning it to the variable A\_train. In addition, the function will assign label -1 to the images of digit n1 and +1 to the images of the digit n2. This will be stored in vector b\_train. A\_train and b\_train are returned as output.

function classify(X, Y, n1, n2):

X\_filtered = X[indices where Y = n1 or n2]

Y\_filtered = Y[indices where Y = n1 or n2]

pca16 = PCA object with 16 components

# Fit pca16 with X\_train

A\_train = projection of X\_filtered onto modes of pca16

b\_train = vector of zeros the same length as Y\_filtered

for i in range(length(Y\_train)):

if y\_filtered[i] = n1

b\_train[i] = -1

else

b\_train[i] = 1

return A\_train, b\_train

Thus, we can create A\_train and b\_train for digits 1 and 8 by calling

A\_train, b\_train = classify(X\_train, Y\_train, 1, 8)

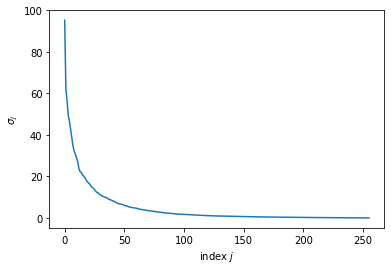
We can then use Ridge regression to train a predictor for b\_train by linearly combining the columns of A\_train. We create a Ridge regression object, reg using sklearn.linear\_model.Ridge(alpha=1.0). We then fit the model using A\_train and b\_train. We can then find the training Mean Squared Error (MSE) of the classifier using

predictions = reg.predictions(A\_train)

sklearn.metrics.mean\_squared\_error(predictions, b\_train)

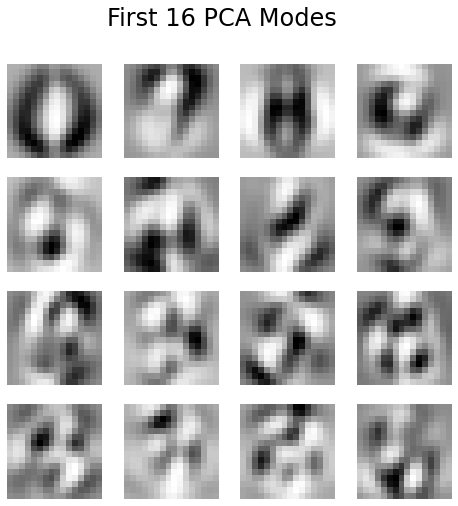
We can find the testing MSE by instead using A\_test and b\_test, which can be created in the same way as A\_train and b\_train, but instead using X\_test and Y\_test. This process can then be repeated for digits 3 and 8, digits 2 and 7, and other pairings.

## **Sec. 4. Computational Results**



(Figure 1) Plot of the singular values from the PCA object fitted with X\_train.

We see from this plot that the values for decrease rapidly from index 0 - 50, with the differences past that range being negligible.



(Figure 2) Plot of the first 16 PCA modes as 16x16 images.

My value for the Frobenius norm is **190.27893628308198**. For the number of PCA modes needed to approximate up to a certain percentage I obtained the following:

|  | Number of PCA modes |
| --- | --- |
| 60% | 3 |
| 80% | 7 |
| 90% | 14 |

Thus, we do not need the entire 16x16 image for each data point.

These are my values for the different test and training MSE’s:

Training MSE for digits 1 and 8: 0.07461177974622323

Test MSE for digits 1 and 8: 0.0832959946666643

Training MSE for digits 3 and 8: 0.18040371928152915

Test MSE for digits 3 and 8: 0.2580998572477193

Training MSE for digits 2 and 7: 0.09179209345042728

Test MSE for digits 2 and 7: 0.13649806147705723

I think that the differences in performance are due to the differences in similarity between the pairs of digits. 1 and 8 are less similar visually than 3 and 8, which could explain why the latter has a higher MSE than the former.

## **Sec. 5. Summary and Conclusions**

Utilizing Principal Component Analysis, I was able to train a classifier to distinguish handwritten digits. I found that we do not need the whole 16x16 image for each data point. I also found the MSE of the classifier for different number pairings.

## **Acknowledgments**

I would like to acknowledge the students in the AMATH 582/482 Discord who allowed me to have fruitful discussion about this assignment.

## **References**

Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature

585, 357–362 (2020). <https://doi.org/10.1038/s41586-020-2649-2>

J. D. Hunter, "Matplotlib: A 2D Graphics Environment," in Computing in Science &

Engineering, vol. 9, no. 3, pp. 90-95, May-June 2007, doi: 10.1109/MCSE.2007.55.

Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.