AMATH 482/582 HOMEWORK 2 - CLASSIFYING DIGITS

PHILIP WESTPHAL

Department of Applied Mathematics, University of Washington, Seattle, WA
philw16@uw.edu

Abstract. In this report, the use of Principal Component Analysis (PCA) to reduce the dimensionality of images from the MNIST digit dataset is explored. The amount of information captured by keeping various numbers of principal components is discussed, and ridge regression models fit using a 16 component approximation of the full dataset were used to train a binary predictor for three sets of digit pairs. The Mean Square Error (MSE) of these models were compared, and additionally the MSE of each model keeping a varying number of principal components was also explored.

1. Introduction and Overview

The goal of this problem was to explore the impact of dimensionality reduction on the MNIST dataset using Principal Component Analysis (PCA). A training dataset X_{train}, $y_{train} \in \mathbb{R}^{2000,256}$, $\mathbb{R}^{2000}$ and a test dataset X_{test}, $y_{train} \in \mathbb{R}^{500,256}$, $\mathbb{R}^{500}$ were provided. Each entry of $X_{train},\ X_{test}$ represents a $16x16$ pixel image of a digit from 0 to 9 flattened to 256 features, and $y_{train},\ y_{test}$ are the digit labels. Using PCA, the number of features can be reduced from 256 by projecting the $X_{train},\ X_{test}$ datasets down to a specified number of principal components that capture a majority of the data variance.

Principal Component Analysis (PCA) is extremely useful in machine learning applications, as reducing the feature space while still approximation the full dataset well can reduce train time while still producing an accurate model. In this report, PCA is used to reduce the number of features from the flattened MNIST dataset down from 256 features to 16 for use in a ridge regression model. While the full MNIST dataset contains digits from 0 to 9, the ridge regression models in this report are binary and built using only two specified digits.

2. Theoretical Background

2.1. Singular Value Decomposition & PCA

For every $X \in \mathbb{R}^{n,m}$, there exists a matrix factorization known as the Singular Value Decomposition (SVD) given as

$$X = U\Sigma V^T \tag{1}$$

where the columns of $U \in \mathbb{R}^{n,n}$ represents the left singular vectors, the columns of $V \in \mathbb{R}^{m,m}$ represent the right singular vectors, and $\Sigma \in \mathbb{R}^{n,m}$ is a diagonal matrix with non-negative entries of descending magnitude [1]. The entries of Σ are the singular values of X, with the number of non-zero entries being equivalent to the rank of X. Considering X as a linear transformation, the singular values $\sigma_j \in \Sigma$ represent a scaling factor of the transformation $u_j v_j^T$ where $u_j \in U$ is a basis vector and $v_j \in V$ is a rotation vector [6]. The largest singular values therefore represent the majority of the transformation, and a small number of them can reasonably approximate X. Considering only $k < m$ singular values, we obtain a low rank approximation of X [1], given as

$$X \approx U_{red}\Sigma_{red}V_{red}^T \tag{2}$$

$$U_{red} \in \mathbb{R}^{n,k}, \Sigma_{red} \in \mathbb{R}^{k,k}, V_{red} \in \mathbb{R}^{m,k}$$

Date: February 11, 2022.

Principal Component Analysis (PCA) is a technique utilizing SVD to reduce the number of dimensions in a dataset while maintaining as much of the existing random variations as possible [5]. The variance between variables in a matrix X of N columns can be expressed via the covariance matrix as

$$Cov(X) = \frac{1}{N-1}\sum_{j=0}^{N-1}(x_j - \bar{x_j})(x_j - \bar{x_j}) = \frac{1}{N-1}\tilde{X}\tilde{X}^T \tag{3}$$

with $\bar{x_j}$ representing the mean of column j [9], and $\tilde{X}$ representing the mean centered data of X accounting for all j. Applying SVD to the covariance matrix, we derive the following

$$\begin{aligned}\frac{1}{N-1}\tilde{X}\tilde{X}^T &= \frac{1}{N-1}\tilde{U}\tilde{\Sigma}\tilde{V}^T(\tilde{U}\tilde{\Sigma}\tilde{V}^T)\\ &= \frac{1}{N-1}\tilde{U}\tilde{\Sigma}\tilde{V}^T\tilde{V}\tilde{\Sigma}\tilde{U}^T\\ &= \frac{1}{N-1}\tilde{U}\tilde{\Sigma}^2\tilde{U}^T\end{aligned}$$

Thus, by equation 2 we know the majority of the variation in $\tilde{X}$ can be represented using the largest singular values of $\tilde{\Sigma}$ to scale the left singular vectors of $\tilde{U}$. The Frobenius Norm of matrix A keeping k singular values can be calculated as

$$\|A_k\|_F = \sqrt{\sum_{j=0}^{k-1}\sigma_j^2} \tag{4}$$

To obtain a cumulative percentage captured of the Frobenius Norm, the truncated sum can be divided by the full Frobenius Norm of the A using all σ, taking $k = N-1$.

2.2. **Maximum Likelihood Estimation & Ridge Regression**

Given a feature x_j and corresponding output y_j, there exists a function f^+ that maps x_j to y_j such that

$$y_j = f^+(X_j) + \epsilon_j$$

where ϵ_j is noise of feature j. Assuming Gaussian noise of variance σ and accounting for all features $x_j \in X$, a probability function of outputs y given X can be estimated as

$$P(y|X) \approx \exp\left(\frac{-1}{2\sigma^2}\left\|f^+(X) - y\right\|_2^2\right) \tag{5}$$

Maximizing equation 5 is equivalent to finding a function f that minimizes the exponential term [3]. This function is known as the Maximum Likelihood Estimator (MLE), given by

$$f_{MLE} = \underset{f}{\operatorname{argmin}}\frac{1}{2\sigma^2}\left\|f(X) - y\right\|_2^2 \tag{6}$$

Further assuming that the solution to equation 6 is linear where f is an affine transformation of X,

$$f(X) = \beta_0 + \sum_{j=0}^{N-1}\beta_j X_j$$

a linear regression problem can be defined where we are attempting to solve for the coefficient vector β that minimizes

$$\beta_{MLE} = \underset{\beta}{\operatorname{argmin}}\frac{1}{2\sigma^2}\left\|A\beta - y\right\|_2^2,\ A = [\vec{1}\,X] \in \mathbb{R}^{n,m+1},\ \beta \in \mathbb{R}^{m+1} \tag{7}$$

A unique solution to equation 7 is only guaranteed if the matrix A^TA is invertible, which is not always the case. To fix this, a regularization term can be added to manipulate the problem such that a unique solution can be obtained. One example is a ridge regression model, which adds an $L2$ regularization of β. It is given as follows, where λ is a regularization parameter

$$\beta_{MLE} = \underset{\beta}{\operatorname{argmin}}\frac{1}{2\sigma^2}\left\|A\beta - y\right\|_2^2 + \frac{\lambda}{2}\|\beta\|_2^2 \tag{8}$$

To measure the performance of the ridge regression model, we can calculate the mean square error (MSE) using the optimal β from equation 8

$$MSE = \frac{1}{\text{length } y}\|A\beta_{MLE} - y\|_2^2 \tag{9}$$

Cross validation is used to select the optimal value of λ in equation 8. This technique involves breaking the training dataset into a number of pieces, then training the model multiple times with each piece being treated as the test set once [4]. The MSE is then aggregated across all runs, and the optimal model is the one with the lowest total error. For the ridge regression problem, cross validation is done using K dataset pieces for a range of λ to find the optimal value

$$\lambda_{opt} = \underset{\lambda \in \mathbb{R}}{\text{argmin}} \sum_{k=0}^{K-1} MSE(X_{-k}, \lambda) \tag{10}$$

The $-k$ index in equation 10 signifies the piece k of the dataset is being held out for training.

3. Algorithm Implementation and Development

Implementation of PCA was done using the `PCA()` function from `sklearn.decomposition`, which centers the input data when reducing as required per equation 3 [8]. Algorithm 1 outlines the code used to load the train and test data, generate a PCA projection, and approximate a dataset given N components to keep

Algorithm 1 PCA Application and Approximation

Load $X_{train}, y_{train}, X_{test}, y_{test}$ ▷ X are features, y are labels
pca = `PCA()`
Fit PCA on X_{train} ▷ Only fit on training data
function dataModeApproximation(pca, X_{data}, N)
  X_{pca} = PCA transform of X_{data}
  $X_{reduced}$ = reduced X_{pca} using first N components
  Zero out remaining components after N
  X_{approx} = inverse PCA transform of $X_{projected}$
  return $X_{reduced}$, X_{approx}
end function

Algorithm 2 was used to extract the data relating to two specified digits from the full dataset, and convert the labels to a form $\{-1, 1\}$ for use in a regression model

Algorithm 2 Reduction of Dataset to Two Specified Digits

function reducedBinaryDigitSet(pca model, N, X data, y data, digits)
  Find indices in y data corresponding to specified digits ▷ These are the labels
  X_{digits}, y_{digits} = X_{data}[indices], y_{data}[indices]
  y_{digits} = **if** $y_{data} < 0$ **then** -1 **else** $= 0$
  $X_{reduced}$ = dataModeApproximation(pca model, X_{digits}, N) ▷ From algorithm 1
  return $X_{reduced}$, y_{digits}
end function

A cross validating ridge regression was built by providing a range of possible λ values to the `RidgeCV()` model from `sklearn.linear_model` [8]. By default `RidgeCV()` uses leave-one-out cross validation, which takes $K = N$ in equation 10. Fitting this model on a dataset identifies the optimal λ, which can then be used by the model to predict output y given input X. Athough we are using a regression model, a classification accuracy can still be obtained by taking $y_{pred} < 0$ as -1 and $y_{pred} > 0$ as 1.

Algorithm 3 Run Regression Model

lambdas $= 2^n,\ n = [-4, ..., 20] \in \mathbb{R}^{1000}$
function RIDGECLASSIFIERFORDIGITS(train data, test data, pca, N, digits, lambdas)
X_{train}, y_{train} = train data; X_{test}, y_{test} = test data ▷ Extract data
Apply algorithm 2 to obtain $X_{train\ reduced}, y_{train\ red}, X_{test\ red}$, $y_{test\ red}$ ▷ Reduce data
`RidgeCV()` fit on $X_{train\ red}, y_{train\ red}$ to find optimal λ
$y_{train\ pred}$ = `RidgeCV()` predict on $X_{train\ red}$
$y_{test\ pred}$ = `RidgeCV()` predict on $X_{test\ red}$
Calculate accuracy$_{train}$, accuracy$_{test}$ for $y_{train\ pred}$, $y_{test\ pred}$
Calculate MSE_{train}, MSE_{test} for $y_{train\ pred}$, $y_{test\ pred}$
return MSE_{train}, accuracy$_{train}$, MSE_{test}, accuracy$_{test}$, `Ridge()`
end function

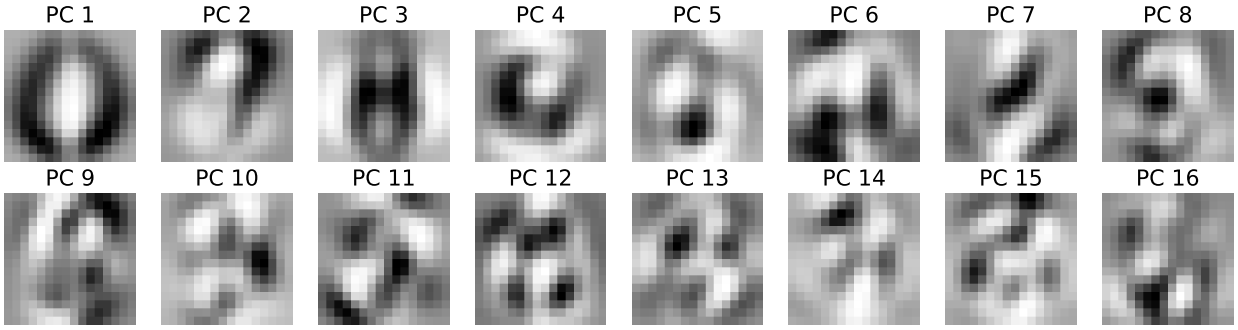

FIGURE 1. First 16 Principle Components of PCA fit on X_{train}

4. COMPUTATIONAL RESULTS

Figure 1 shows the first 16 principle components from the PCA fit on X_{train} obtained with algorithm 1. These represent the 16 most prominent component vectors of U that we can project our data on to reduce the dataset dimensionality.

The log of the singular values of X_{train} are shown in figure 2. While there is no clear breakpoint on the curve, it begins approach a linear rate of decrease quickly around 25 principal components. Around 150 principal components, there is an inflection point where the convexity of the curve changes.

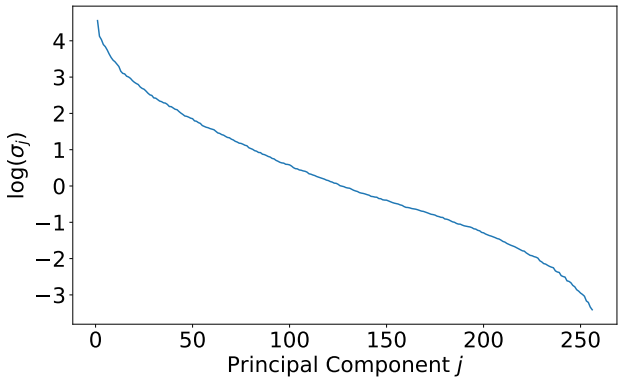

FIGURE 2. Singular Value Spectrum of X_{train}

The first 16 images of X_{train} were plotted using a the full dataset as well as 16, 32, and 64 mode approximations. Figure 3 shows that even at 16 modes the digits can clearly be identified, though there is a noticeable amount of background distortion. By 64 modes (25% of the full data dimensionality) the digits

are extremely clear and a majority of the background distortion is gone. This indicates the full $16x16$ image of a digit is not needed as a low rank approximation provides a strong estimation of the full dataset.

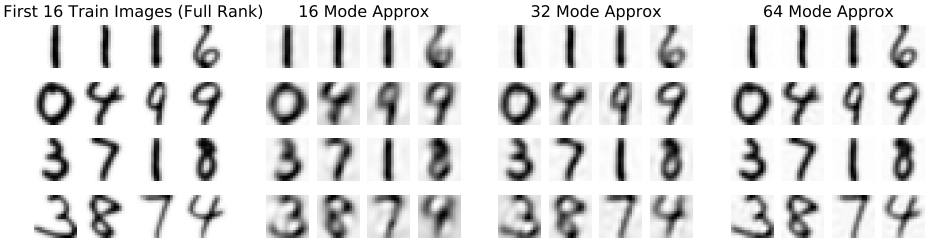

FIGURE 3. Modal Approximations of First 16 Images in X_{train}

Figure 4 shows the percentage of the Frobenius Norm captured and the normalized singular value magnitude of the of the first 70 principal components. The data show we can capture 80% of the Frobenius Norm with only the first seven components, and 90% from the first 14 components. Additional components yield diminishing returns, as keeping 23 components only accounts for an additional 5% of the Frobenius Norm. From the plot of the normalized singular values, we see an elbow point occurring between 15 and 30 principal components where the curve begins to decrease in a nearly linear fashion. Considering both of these facts, a good dimensionality for approximating the dataset is likely between 14 and 25 principal components.

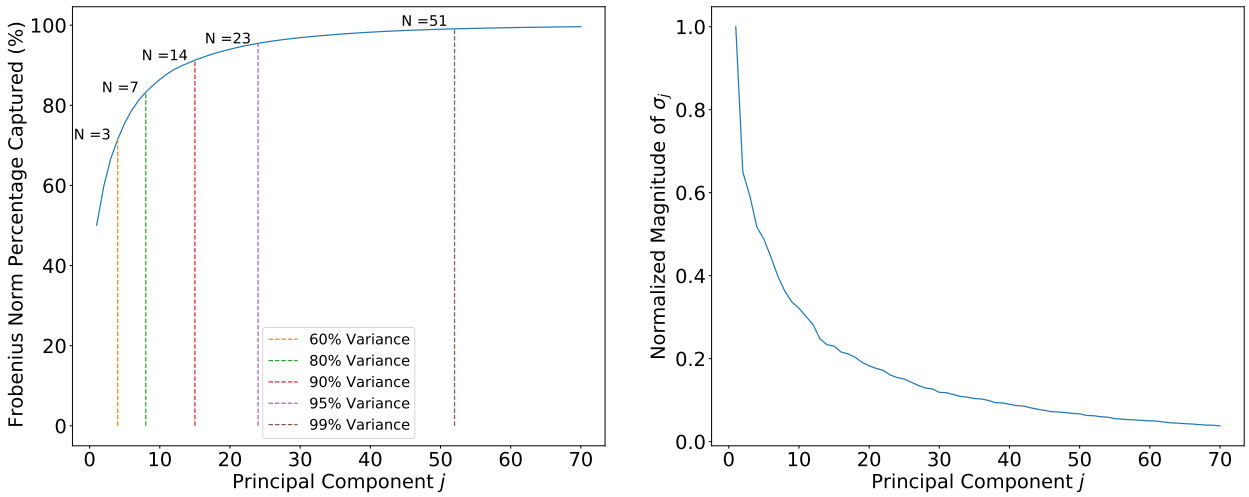

FIGURE 4. Frobenius Norm Percentage Captured & Normalized Singular Value Magnitude of First 70 Principal Components

Table 1 shows the train and test MSE for the regression model described in algorithm 3 evaluated on three sets of digit pairs

Digit Pairs	Training Set MSE	Test Set MSE	Training Set Accuracy	Test Set Accuracy
1, 8	0.0754	0.0826	99.12%	97.92%
3, 8	0.1820	0.2600	98.29%	91.67%
2, 7	0.0924	0.1325	99.43%	97.96%

TABLE 1. Ridge Regression MSE on Train & Test Datasets for Two Digit Pairings

The data show that the test set MSE is higher than the training set MSE in all cases. This makes sense, as the regression model has not seen the test set whereas it was optimized using the training set. The data also show that the MSE values for the digit pairs $(3, 8)$, and $(2, 7)$ are higher than that of the digit pair

$(1, 8)$. The digits $(3, 8)$ share similar shape and curvature, which could make distinguishing between the two difficult with a lower rank dataset approximation. The digits $(2, 7)$ also have similar curvature in the middle but have more distinguishable edge features than 3 and 8, which could contribute to their better MSE values. Figure 5 shows the train and test MSE of each digit pair keeping a variable number of principal components. The $(2, 7)$ and $(3, 8)$ pairings have a higher steady state MSE than the $(1, 8)$ pairing and require more principal components to reach lower MSE values.

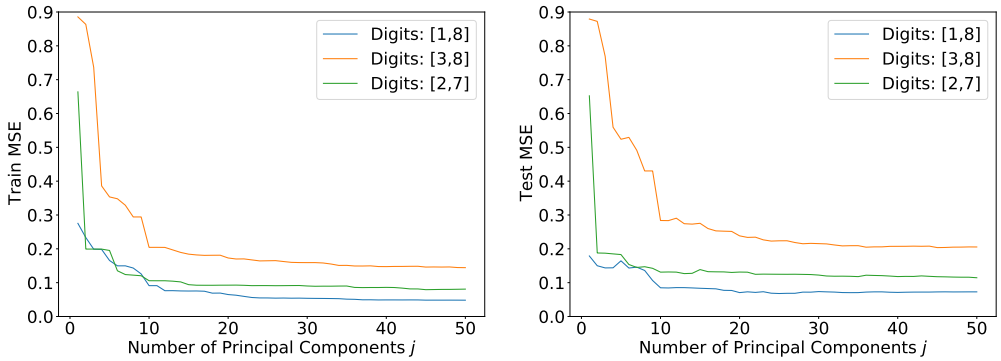

FIGURE 5. Train & Test Set MSE of Two Digit Pairs for the first 50 Principal Components

5. Summary and Conclusions

In solving this problem, we made use of Principal Component Analysis to reduce the dimensionality of a dataset while still building accurate regression models. Table 1 shows the train and test of three sets of digit pairs, while figure 5 visualizes the train and test error of each digit pair with a varying number of principal components. The data show that certain digit pairs perform better than others with a specified number of components, and also that certain digit pairs such as $(1, 8)$ are able to produce more accurate models with fewer principle components. In future experiments, it would be interesting to train a classifier with additional pairs of digits as well as the full MNIST dataset. The impact of the number of principle components kept on the train and test MSE could then be explored.

Acknowledgements

The author would like to thank Professor Bamdad Hosseini for covering the background concepts of PCA and Ridge Regression in lecture, as well as Katherine Owens for clarifying questions regarding calculating the Frobenius Norm and Mean Squared Error. Collaboration with other classmates in Discord was also useful, and the author thanks those in the Discord chat who discussed how to plot PCA modes and methodologies for calculating the variance captured by components using the Frobenius Norm. In addition to sklearn, the numpy [2] and matplotlib [7] packages in python were useful for processing data and generating visualizations.

References

[1] S. Brunton and J. N. Kutz. Singular value decomposition (svd). In *Data Driven Science & Engineering*.
[2] 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.
[3] B. Hosseini. Introduction to machine learning. University of Washington, Jan 2022. AMATH 482/582.
[4] B. Hosseini. Model tuning with cross validation. University of Washington, Feb 2022. AMATH 482/582.
[5] B. Hosseini. Principle component analysis. University of Washington, Jan 2022. AMATH 482/582.
[6] B. Hosseini. Review of linear algebra. University of Washington, Jan 2022. AMATH 482/582.
[7] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.
[8] 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.
[9] E. W. Weisstein. Covariance.