AMATH 582: HOMEWORK 2

CASSIA CAI

School of Oceanography, University of Washington, Seattle, WA
`fmc2855@uw.edu`

Abstract. Digit classification using the MNIST dataset is often used as the foundation for developing and evaluating machine learning methods. Using a subset of the MNIST dataset, we train a classifier to distinguish images of handwritten digits. We can approximate the training data images up to 90% in the Frobenius norm using the first 14 PCA modes. However, after applying StandardScaler, which standardizes by removing the mean and scaling to unit variance, 90% is approximated using 16 modes. To train our classifier, we use scikit-learn's Ridge regression function for model fitting. The training and testing mean squared errors are compared for sets of 2-digit combinations. Although the classifier performs worse with the pair (3, 8), it generally performs with reasonable accuracy. An extension is to test our classifier performance using different model fitting methods, like K-nearest neighbors, random forest, and gradient boosted trees, and support vector machines.

1. Introduction

Given images of handwritten digits from the Modified National Institute of Standards and Technology (MNIST) dataset, we aim to train a classifier to distinguish the images. We use only a subset of the MNIST dataset, which is significantly larger, containing 60,000 training images and 10,000 testing images. The training data we use is a subset of the MNIST dataset containing 2000 handwritten digits. The "features," which we denote as X_{train} are the 16x16 black and white training images while "labels," which we denote as Y_{train}, are the digits. Our testing data is smaller, containing only 500 handwritten digits, but has the same attributes, which we denote as X_{test} and Y_{test}. The first 64 training features are visualized in Figure 1.

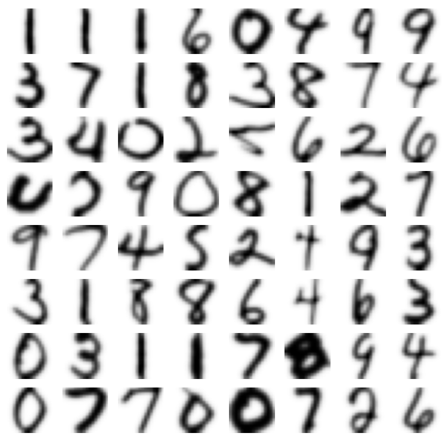

Figure 1. First 64 training features

Date: February 10 2022.

Using Principal Component Analysis (PCA), we investigate the dimensionality of X_{train}. We determine the number of PCA modes to keep in order to approximate X_{train} up to 60%, 80%, and 90% in the Frobenius norm. We do not need the whole 16x16 image for each data point. To approximate up to 90%, we keep the first 16 PCA modes (if we standardize X_{train}). Without standardizing, we only need the first 14 PCA modes. We then train a classifier to distinguish two digits by first extracting the features and labeling of those two digits from the training data set, then projecting those features on the first 16 PCA modes of X_{train}. After reassigning the digits to either -1 or 1, we use Ridge Regression to train a predictor for this new matrix of -1 and 1. We assess our classifier by comparing the training and testing mean squared errors (MSE) of the classifier. We are indeed able to train a classifier that is able to classify the two digits with reasonable accuracy. This method can be extended to classify multiple numbers as well as handwritten letters [1, 2]. Moreover, there are multiple ways to construct a classifier, such as using neuromorphic nanowire networks among others.

2. Theoretical Background

Principal Component Analysis (PCA): PCA is an approach to decomposing signals/images/data, finding c_k, the coefficients, and ψ_k [3]. PCA is viewed as a dimensionality reduction technique, which means that high dimensional data can be represented using a few coefficients. From a linear algebra point of view, PCA is an application of singular value decomposition (SVD), which produces characteristic features determined by C_X, the covariance matrix of data $\mathbf{X}$.

The eigenvectors of C_X are the principal components. Defining C_X if $X = \mathbb{U}\Sigma\mathbb{V}^T$, then we have:

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

where the columns of $\mathbb{U}$ are the PCA modes of C_X. In the equation above, we have switched from working with $\mathbf{X}$ to a new variable. We can think about it like this: $Y = U^*X$ where U is a SVD unitary matrix and $X = U\Sigma V^*$

Our new covariance is:

$$C_Y = \frac{1}{n-1}\Sigma^2$$

where $\Sigma^2 = \lambda$ where λ is a diagonal matrix containing the eigenvalues of XX^T [4].

Frobenius Norm: We can use the Frobenius Norm, or the Euclidean norm, for data fitting. The Frobenius Norm is useful because of its centrality. The Frobenius Norm is defined as:

$$\|B^2\|_F^2 = \sum_{minm,n}^{j=1} \sigma_j(B^2), B \in \mathbb{R}^{mxn}$$

Ridge Regression Model: We can use a simple linear regression for supervised learning [5].

$$\hat{f}(\underline{\mathrm{x}}) = \hat{\beta}_0 + \sum_{j=0}^{d-1}\hat{\beta}_j x_j$$

$$\hat{\beta} = argmin\|\hat{f}(x) - \underline{\mathrm{y}}\|^2$$

The regression solution is found by solving:

$$\underline{\hat{\beta}} = argmin\frac{1}{2\sigma^2}\|A\underline{\beta} - \underline{\mathrm{Y}}\|^2 + \frac{\lambda}{2}\|\underline{\beta}\|^2$$

where $\hat{\beta}$ are the predicting coefficient vectors and λ is the regularization/penalization parameter.

Mean Squared Error: If we define the trained regression model as:

$$\hat{f}(\underline{x}) = \sum_{j=1}^{J} \hat{\beta}_j \psi_j(\underline{x})$$

Then we define the training MSE as:

$$MSE_{\text{train}}(\hat{f}, \underline{Y}) = \frac{1}{N} \sum_{n=0}^{N-1} |\hat{f}(\underline{x}_n) - y_n|^2 \quad for \underline{x}_n \in \mathcal{X}, y_n \in \mathcal{Y}$$

Similarly, we can calculate the testing MSE. After calculating the testing and training MSEs, we can assess the performance of the classifier.

Ridge Regression with Cross Validation: Both training and testing errors are dependent on the choice of λ, which is a model parameter. A high λ would result in a model that is too simple while a low λ results in a model that is too complicated. This then affect our classifier performance, leading to variations in the MSEs between the test and training datasets [5]. To address this, we could test different λ values and calculate corresponding training and testing MSEs. Alternatively, we could use K-folder Cross Validation (CV). K-folder CV splits the training data (both features and labels) into K-subsets (some as training and others as testing), iterating over k = 0, ..., K -1, removing the K-th subset, and fitting the model to the training data. We use K-fold CV in this paper.

3. Algorithm Implementation and Development

The provided MNIST data is split into training and testing data. To classify digits, we investigate the dimensionality of X_{train}, which has a maximum number of components of 256, using scikit-learn's PCA function and plot the first 16 PCA modes as 16x16 images. Scikit-learn's PCA function has a number of useful attributes, such as explained variance, explained variance ratio, and singular values. The steps undertaken to create a classifier to distinguish between two digits in the MNIST dataset are outlined in the pseudo-code below. I coded Python and used NumPy and scikit-learn for mathematical calculations and Matplotlib for visualizations [6, 7, 8].

```
**Pseudo-code:** Determine the number of PCA modes and train a classifier
  Step 1: Investigate the dimensionality of $X_{train}$
      a. From scikit-learn, apply PCA fit to $X_{train}$
      b. Using singular values (SV), plot the explained variance ratio (EVR) vs. PCA component
          i. DEF: EVR is the square of the SV divided by the nuclear norm (the sum of SV)
  Step 2: Determine the # of PCA modes to approximate $X_{train}$ to % in the Frobenius norm
  Step 3: Train a classifier to distinguish two digits (example: 1 and 8)
      a. Extract features and labels of 1 and 8 from the training dataset to get $X_{(1,8)}$ and $Y_{(1,8)}$
      b. Project $X_{(1,8)}$ on the number of PCA modes determined from Step 2 to get $A_{train}$
          i. $A_{train}$ columns correspond to PCA coefficients and rows correspond to 1s and 8s
      c. Assign label -1 to the images of digit 1 and -1 to the images of digit 8 to get $b_{train}$
      d. Use Ridge Regression to train a predictor for $b_{train}$ using the fit function and $A_{train}$
      e. Report the training and testing MSE of the classifier
```

We also determine the number of PCA modes to keep in order to approximate X_{train} up to 60%, 80%, and 90% in the Frobenius norm because we may not need the entire 16x16 image for each data point. We do this for both X_{train} and standardized X_{train}. We standardize using Standard Scaler. After determining the number of PCA modes to keep, we can perform PCA reconstruction of digits' images. We can indeed create a classifier that distinguishes two digits by first extracting the labels and features of those digits, projecting the new X_{train}, which contains only those two digits, on the first 16 modes of X_{train}. We then relabel the digits to either 1 or -1 before performing ridge regression to train a predictor. We then calculate the MSE of the predicted and actual labels.

4. Computational Results

Using PCA to investigate the dimensionality of X_{train}, we observe that the image can be approximated with fewer than 256 modes, which is the maximum number of modes (Figure 2). Far fewer than 256 modes are needed. We observe that 90% of the cumulative explained variance can be approximated with only 22 PCA modes. It is important to note that the number of PCA modes needed to approximate 90% in the Frobenius norm is not 22 but rather 16 PCA modes (Table 1). To approximate 60% and 80% of the cumulative explained variance, we need 5 and 12 PCA modes respectively. The first 16 PCA modes as 16x16 images is shown in Figure 3.

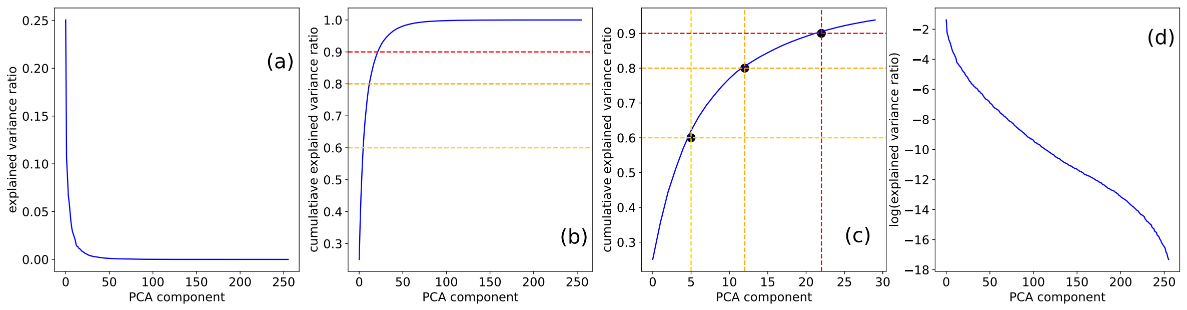

Figure 2. **(a)** Explained variance ratio vs. PCA component **(b)** Cumulative explained variance ratio vs. PCA component. The dashed red line signifies where the cumulative variance ratio is 0.9. Similarly, orange is 0.8 and yellow is 0.6. **(c)** Same as **(b)** but zoomed in. The circular markers indicate the number of PCA components at the 0.9, 0.8, and 0.6 cumulative explained variance ratio levels. **(d)** log of the explained variance ratio vs. PCA component

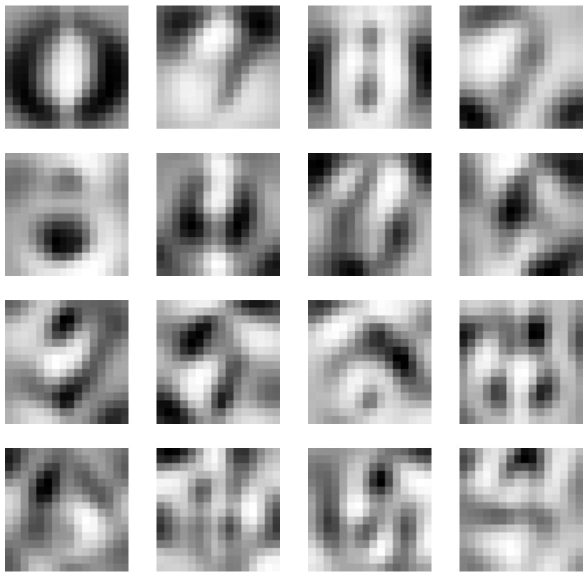

Figure 3. First 16 PCA modes

To approximate X_{train} up to 60%, 80%, and 90% in the Frobenius norm, we keep 3, 7, and 17 PCA modes respectively (Table 1). If we apply StandardScaler to X_train, then we need 3, 8, and 16 PCA modes to approximate 60%, 80%, and 90% in the Frobenius norm. It is clear that we do not need the entire 16x16 image for each data point (Figure 4). We re-iterate that approximating X_{train} in the Frobenius norm is different from approximating the cumulative explained variance.

% of Frobenius norm	# of modes w/o StandardScaler	# of modes w/ StandardScaler
60	3	3
80	7	8
90	14	16

TABLE 1. The number of PCA modes to keep in order to approximate the training feature data up to 60%, 80%, and 90% of the Frobenius norm calculated with and without applying StandardScaler to X_{train}

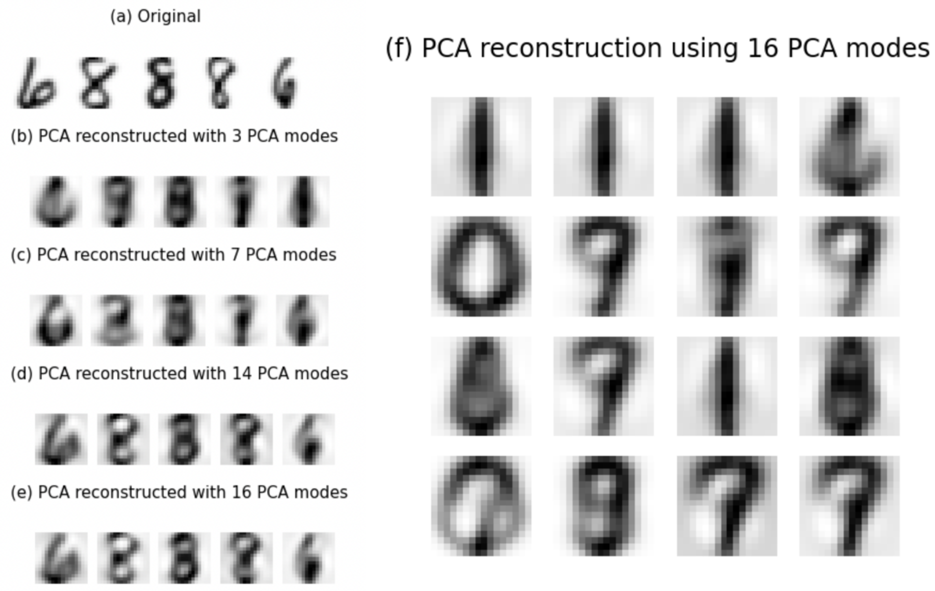

FIGURE 4. Confusion matrices to describe the performance of our classifier using three sets of digit combinations **(a)** (1,8), **(b)** (3,8), **(c)** (2,7).

After training our classifier, we report the training and testing MSEs for three sets of digit combinations (Table 2). Comparing the MSEs of these three cases, we observe that the testing MSEs are slightly higher than training MSEs, which we expect. Both training and testing MSEs for the digit combination (3,8) are higher than those for (1,8) and (2,7).

A possible reason is geometric similarity, where 3 and 8 look more similar than 1 and 8, thereby making classifying these two digits more difficult. However, to check whether this reasoning is plot a confusion matrix with different digit combinations. Confusion matrices for the three sets of digit combinations are shown in Figure 5, where we indeed see that 3 and 8 are particularly difficult.

Digit Combination	Training MSE	Testing MSE
(1, 8)	0.075	0.083
(3, 8)	0.180	0.258
(2, 7)	0.092	0.136

TABLE 2. Training and testing MSE of classifier on sets of digit combinations

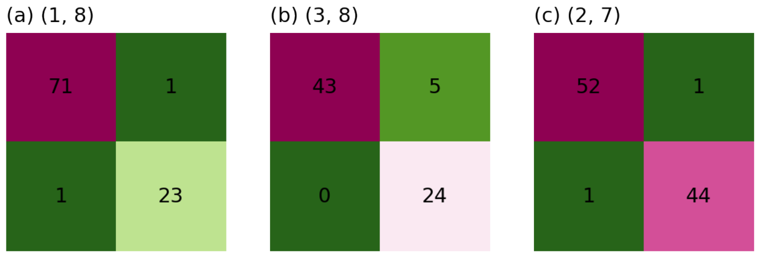

FIGURE 5. Confusion matrices to describe the performance of our classifier using three sets of digit combinations **(a)** (1,8), **(b)** (3,8), **(c)** (2,7).

5. Summary and Conclusions

We built a digit classifier to distinguish images of handwritten digits from the MNIST dataset. We used PCA to investigate the dimensionality of the features of the training data, which allowed us conduct dimension reduction. After applying StandardScaler on X_train, we found that to approximate 90%, 80%, and 60% in the Frobenius norm, we need the first 16, 8, and 3 PCA modes. Without applying StandardScaler, to 90%, 80%, and 60% in the Frobenius norm, we need the first 14, 7, and 3 PCA modes. We decided to use the first 16 PCA modes. After extracting the features of our target digits and re-labeling to either -1 and 1, we use Ridge regression model with cross validation to train a predictor, which we apply to the test set. Comparing the training and testing MSEs of our classifier, we find that testing MSEs are higher than the training MSEs. There is some variation in MSE for the three sets of digit combinations we tested, which we hypothesize to be due to geometric differences in the digit pairs. An extension would be to use different model fitting methods because different methods differ in how predictive they are on the test dataset.

Acknowledgements and Code Availability Statement

The author is thankful for the algorithm-implementation suggestions from peers in AMATH 482-582. The author is thankful for theoretical background discussions with Joyce Cai and to Prof. Bamdad Hosseini for providing the problem set-up and data as well as AMATH 482-582 Teaching Assistant Katherine Owens for productive discussions about setting up the classifier.

The code is publicly available on this Github repository after the submission date.

References

[1] Alejandro Baldominos, Yago Saez, and Pedro Isasi. A survey of handwritten character recognition with mnist and emnist. *Applied Sciences*, 9(15):3169, 2019.

[2] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre van Schaik. Emnist: Extending mnist to handwritten letters. *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017.

[3] J. Nathan Kutz. *Data-Driven Modeling amp; Scientific Computation: Methods for Complex Systems amp; Big Data*. Oxford University Press, Inc., USA, 2013.

[4] Bamdad Hosseini. Principal component analysis. University of Washington-Seattle (LOW 216), Feb 2022. AMATH 482/582.

[5] Bamdad Hosseini. Evaluating sl models. University of Washington-Seattle (LOW 216), Feb 2022. AMATH 482/582.

[6] 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.

[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.