AMATH 482/582: HOMEWORK 2

JONATHAN BEAUBIEN

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

Abstract. A classic problem in machine learning is classifying handwritten digits from the MNIST data set. By using Principle Component Analysis (PCA) to reduce the dimension of our image data, we were able to then build a classifier, using the first 16 PCA components, to distinguish between any set of two integers (from 0-9). We used Ridge regression to train the classifier and reported its mean standard error (MSE) as a measure of its efficacy.

1. Introduction and Overview

The classic problem of classifying handwritten digits has been implemented many times. The original MNIST dataset contains 60000 grayscale images of size 28×28. However, for our purposes we have only 2500 grayscale images of size 16×16. We will split up the data into a training and testing set with 2000 and 500 images respectively. Each image will be represented as a flattened vector of size 256 (16^2).

Our first order of business will be to reduce the dimensionality of the images. To do this, we can use Principle Component Analysis (PCA) to reduce our dimensions without losing information and keeping uncorrelated variables that maximize variance. This will change the basis of our data by projecting the original images onto only a certain set of principle components. We will choose how many components to keep based off of how much of the Frobenius norm we can maintain between the original and approximated data.

Once we have a basis for dimension reduction, we can then start to create a classifier. The first attempt at classification will be to classify between images of 1s and 8s. To do this, we will project all of the 1s and 8s on the basis we found in the first section. Then, by using the singular value coefficients, we can feed the training data of 1s and 8s into a Ridge regression function that will fit a linear model. We can then feed the testing data into the model to determine its accuracy. The accuracy measure we will use will be the Mean Squared Error (MSE) of the prediction versus the true value.

Lastly, we will train classifiers for other pairs of digits: $(3, 8)$ and $(2, 7)$ and report the MSE for each.

2. Theoretical Background

PCA rests on Singular Value Decomposition (SVD). In essense, PCA is just an application of SVD. SVD can be summed up by this one theorem:

Theorem: Let $A \in \mathbb{R}^{n\times m}$. Then there exist unitary matrices $\mathbb{U} \in \mathbb{R}^{n\times n}$ & $\mathbb{V} \in \mathbb{R}^{m\times m}$ along with a diagonal matrix $\Sigma \in \mathbb{R}^{n\times m}$, with positive entries, so that [4]

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

Some more useful notation and conventions include [4]:

Date: February 11, 2022.

- The $\{\underline{\mathrm{u}}_j\}$ columns of $\mathbb{U}$ are called the **left singular vectors** of A and form a basis for $\mathbb{R}^n$
- The $\{\underline{\mathrm{v}}_j\}$ columns of $\mathbb{V}$ are called the **right singular vectors** of A and form a basis for $\mathbb{R}^m$
- **The $\{\sigma_j\}$ are non-negative, (typically in descending order) $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_m \geq 0$ and are called the singular values of A**

We can then define PCA to be **the left singular vectors of X are the principal components of the data $\underline{\mathrm{x}}_j$ and the optimal basis in which the $\underline{\mathrm{x}}_j$ can be represented**. [3]

The larger σ_j represent larger variance in the direction of the vector $\underline{\mathrm{u}}_j$. Hence, the PCA modes corresponding to the larger σ_j are indeed more significant as they represent the directions of maximum variance in the data [3]. This means we will keep the singular vectors that correspond to larger (or the n largest) singular values. For example, we can use the number of modes that preserves a certain percent of the Frobenius norm of our training matrix. The Frobenius norm is defined as [4]:

$$A \in \mathbb{R}^{n\times m} \quad \|A\|_F = \left(\sum_{j=1} m\|a_j\|_2^2\right)^{\frac{1}{2}} \tag{2}$$

After we have reduced dimensionality, we want to train a classifier. The equation for Ridge regression is as follows [2]:

$$\hat{\underline{\beta}} = \underset{\beta}{\operatorname{argmin}} \frac{1}{2\sigma^2}\|A\underline{\beta} - \underline{\mathrm{Y}}\|^2 + \underline{\lambda}_2\|\underline{\beta}\|^2 \tag{3}$$

Where $\underline{\mathrm{Y}}$ is our known labels, A is our known features, and $\underline{\lambda}$ is the regulation/penalty parameter (also sometimes called 'alpha').

After we determine our $\hat{\underline{\beta}}$, we can compute the Mean Standard Error (MSE) of each set of data (training and test). If $\hat{f} \equiv \hat{\underline{\beta}}$ and $\{X, Y\} \rightarrow$ training set and $\{X', Y'\} \rightarrow$ test set then our MSE equations are as follows [2]:

$$MSE_{\text{train}} = \frac{1}{N}\sum_{n=0}^{N-1} |\hat{f}(\underline{\mathrm{x}}_n) - y_n|^2, \quad \underline{\mathrm{x}}_j \in \mathcal{X}, y_j \in \mathcal{Y} \tag{4}$$

$$MSE_{\text{test}} = \frac{1}{N}\sum_{n=0}^{N-1} |\hat{f}(\underline{\mathrm{x}}_n) - y_n|^2, \quad \underline{\mathrm{x}}_j \in \mathcal{X}', y_j \in \mathcal{Y}' \tag{5}$$

3. Algorithm Implementation and Development

The scikit-learn [5] implementations of PCA and Ridge regression were used in a Python environment. Plotly was used for creating plots for visualizations. The MSE was retrieved from the Ridge regression classifier attributes. NumPy [1] was also used for matrix and array manipulation.

The feature data that we obtained was in two arrays with dimensions 2000×256 and 500×256 for the training features and testing features respectively. Similarly, the label data was in two vectors of length 2000 and 500. We used scikit-learn's function $PCA()$ to reduce the dimensions of our handwritten digits. We also use scikit-learn's $RidgeCV()$ function to implement Ridge regression with cross-validation.

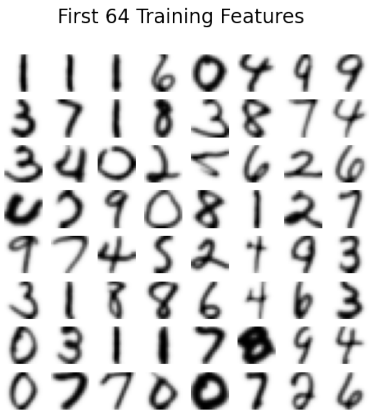
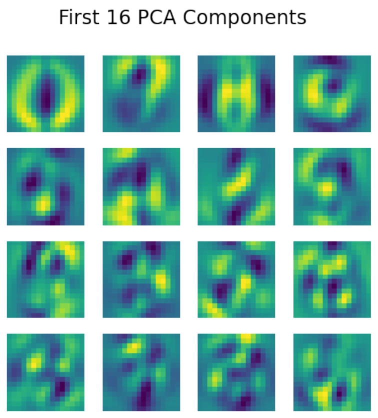


FIGURE 1. Here are the first 64 training features. As you can see, there are digits 0-9 and most are decipherable with the human eye

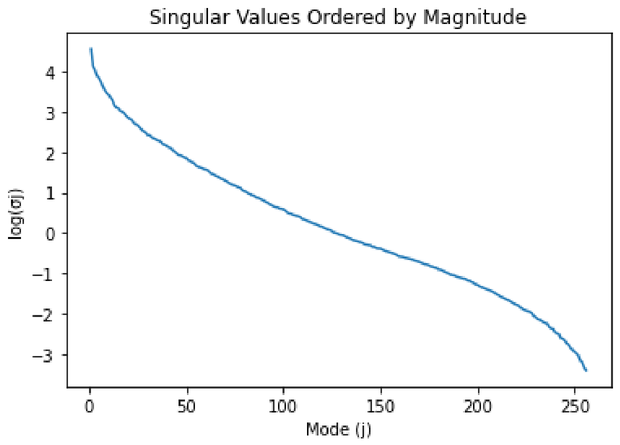
4. Computational Results

To start off, let us visualize some of our handwritten digits to see what they look like.

Our next step will be to take the PCA of all the training data. This will help us reduce our dimensions. Lets us take a look at what the most important principal component modes look like by graphing them. These are the modes that explain the most variance within the data. For our purposes, I will graph the 16 modes by descending singular value. I will also graph the singular values (σ_j) ordered by magnitude (and index, they mean the same thing).



(A) The more important modes are sort of decipherable but as their singular value decreases, the image becomes more 'random'



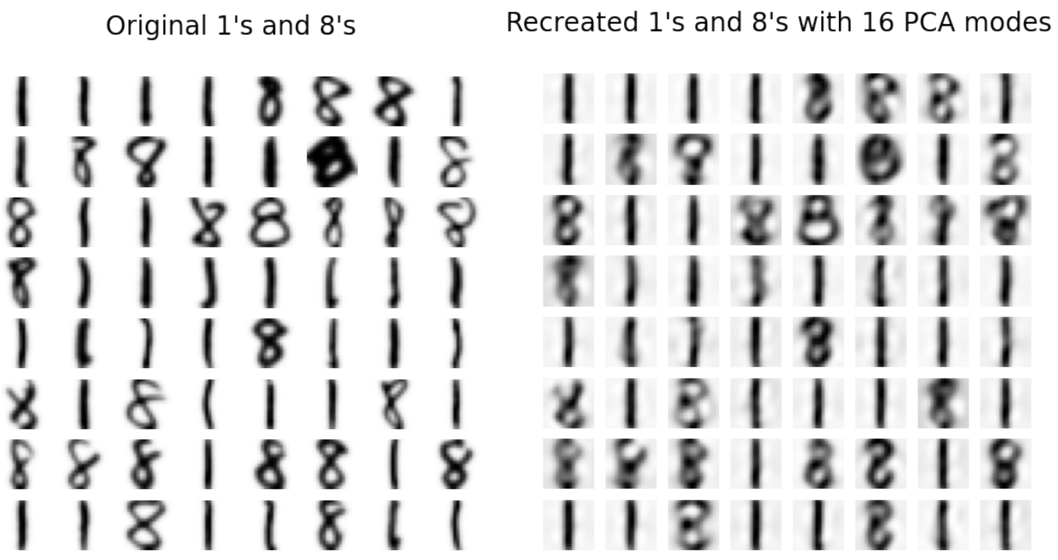
(B) The logarithm of the singular value is graphed to show the sharp drop-off after 50 modes

Next, we will examine how much of our Frobenius norm we can preserve by using lower rank approximations for our training matrix. In the next table, we can see that there is a surprising amount of our Frobenius norm that can be preserved with a fairly low rank approximation. If we want 60% of our Frobenius norm preserved, we only have to save 3 PCA modes and the same is true for 90% and 14 modes.

For our classifier, we will start by trying to classify the digits 1 and 8. To reduce the dimensions, we will project all 1s and 8s on to the 16 PCA modes we calculated in previous section. Instead of

% of Frobenius Norm Preserved	# of PCA modes
66.65	3
81.26	7
90.52	14

storing 256 numbers (one for each pixel), we can now store only 16 while maintaining a high level of accuracy. Here is a comparison of the original data to the dimension-reduced data.



(A) (B)

FIGURE 3. The recreated data looks similar to the original without needing all of the original pixel data

Next we will fit a linear Ridge regression with cross-validation and obtain a MSE for both the training and testing data. We have used $\underline{\lambda} = \{0.1, 1, 10\}$ in our Ridge regression equation (eq. 3). We will translate our labels from human readable (i.e. 1 or 8) to a binary of $\{-1, 1\}$ where digit $1 \to -1$ and $8 \to 1$. We can generalize this entire process to any pair of digits and we will do so for $(2, 7)$ and $(3, 8)$.

	$(1,8)$ Classifier		$(3,8)$ Classifier		$(2,7)$ Classifier	
lambda	$\text{MSE}_{\text{train}}$	MSE_{test}	$\text{MSE}_{\text{train}}$	MSE_{test}	$\text{MSE}_{\text{train}}$	MSE_{test}
0.1	0.08602229	0.08856919	0.20338317	0.26621684	0.10663727	0.16292203
1	0.08581233	0.08613802	0.20310093	0.2568409	0.10643401	0.15656563
10	0.08509995	0.0891674	0.20159898	0.2428258	0.10589419	0.14718207

5. Summary and Conclusions

We were able to train multiple classifiers for different pairs of digits. We also saw how we can use PCA for dimension reduction on image data. After building multiple classifiers with Ridge regression, we saw the MSE of each. The worst classifier is the $(3, 8)$ classifier with the largest MSE of any of them. This is most likely because 3s and 8s look very similar to each other and therefore live close together in our reduced dimension space. We also reported how many modes are needed to preserve at least $60\%, 80\%$, and 90% of the Frobenius norm.

Further investigation could be done by using a different linear regression technique (such as least-squares) or a non-linear regression technique. We didn't try every combination of digits for classification but that could also be done in a further paper.

Acknowledgements

The author is thankful to Katherine Owens for her advice on calculating MSE and general tips for graphing. The author is also thankful to Ava Mistry for her help regarding citations and relevant equations.

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 supervised learning models. University of Washington-Seattle (LOW 216), Feb 2022. AMATH 482/582.

[3] B. Hosseini. Principal component analysis. University of Washington-Seattle (LOW 216), Jan 2022. AMATH 482/582.

[4] B. Hosseini. Singular value decomposition and linear algebra review. University of Washington (LOW 216), Jan 2022.

[5] 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.