AMATH 582: ASSIGNMENT 2

JON STAGGS

Applied Mathematics M.S. Student, University of Washington, Seattle, WA
jrstaggs@uw.edu

Abstract. Given the MNIST handwritten digit data set we utilize the Principal Component Analysis to obtain a low dimensional approximation to our data. With this approximation we train a model with Linear Regression to carry out a binary classification task on pairs of handwritten digits. We compare the accuracy of our model for different pairs of digits and multiple PCA modes.

1. Introduction and Overview

Our goal is to carry out a binary classification on pairs of handwritten digit images. We perform a Principal Component Analysis by projecting our data onto a basis with reduced dimension. We then can construct an approximation of our data. With the approximated data we use Linear Regression on a set of training data and with the regression analysis we test our reduced linear model on a separate set of test data. Using the notation from class, when we write a matrix, X, it is understood that its i^{th} column is $\underline{x}_i$ where $\underline{x_i}$ is an instance of data such as an image [4].

2. Theoretical Background

We use the Principal Component Analysis which is an application of the Singular Value Decomposition. We will discuss the theory of these concepts and show how to approximate data with a lower dimensional structure. We also discuss the formulation of the linear regression problem.

2.1. Singular Value Decomposition.

The Singular Value Decomposition (SVD) is a change of basis of a matrix A of the form,

$$AV = \Sigma U \tag{1}$$

where V is unitary, U has orthonormal columns, and Σ is a diagonal matrix where the elements are ordered, $\sigma_i > \sigma_{i+1} > 0$, where σ_j is the diagonal element in the j^{th} [9].

Consider the common geometric interpretation of the SVD. Let S be the unit n-sphere with radial vectors v_i. The change of basis takes S into a hyperellipse where v_i is mapped to a unit vector u_i whose direction denotes the semi-principle axes of the hyperellipse with length σ_i. We find this example helpful as it motivates the SVD's ability to provide a low rank approximation. Consider the question: If you have a 3D hyperellipse, but you can only approximate with two directions and two lengths- which directions would you choose? The 2 largest axes as these would contain the most information about the hyperellipse. In fact, we can bound the difference of our matrix and its low rank approximation by the following theorem from [9]:

Date: February 11, 2022.

Theorem 1. *Let A be a matrix of rank r then for any ν with $0 \leq \nu \leq r$, define*

$$A_\nu = \sum_{j=1}^{\nu} \sigma_j u_j v_j^*, \tag{2}$$

then A_ν satisfies

$$||A - A_\nu||_F = \inf_{\substack{B \in \mathbb{C}^{m\times n} \\ rank(B) \leq \nu}} ||A - B||_F = \sqrt{\sigma_{\nu+1}^2 + \cdots + \sigma_r^2}. \tag{3}$$

2.2. **Principal Component Analysis.** The goal of PCA is to make a reduction in dimension by making a low rank approximation with a Gaussian and specific covariance matrix [4]. We consider a covariance matrix of our data as it allows us to determine which variables are redundant, which are nearly independent, and which contain the most information. If we have a matrix of data, X, then it's Covariance matrix is,

$$C_X = \frac{1}{n-1} X X^T. \tag{4}$$

In the covariance matrix, the diagonal is the variance of a specific data point and the off diagonal is the covariance between two data points [6]. That means large off diagonal entries represent essentially similar information and small off diagonal points represent more independent information. [6] also remarks that diagonals with large magnitude, meaning large variance, are the variables of interest as they contain large amounts of information about the data. It seems natural that our Σ matrix is the ideal covariance matrix since it would contain independent variables and we easily identify the variables with large variances. We are then motivated to perform this change of basis on C_X,

$$C_X = \frac{1}{N-1} X^T X = \frac{1}{N-1} V^* \Sigma U U^T \Sigma V = \frac{1}{N-1} V^* \Sigma^2 V. \tag{5}$$

The columns of V are called the Principal Components of C_X. These coincide with the eigenvectors of C_X [4].

2.2.1. *Statistical Connection.* This section follows along with [4]. We consider a Gaussian random variable which are determined by there mean, μ, and covariance matrix, C_X. If we have a mean zero Gaussian then our random variable is solely determined by its covariance matrix. This leads us to trying to find a low rank Gaussian that approximates X with covariance matrix, C_X. In other words, we would like the solution to

$$\text{minimize}_{\substack{M \in \mathbb{R}^{d\times d} \\ \text{rank}(M) \leq r}} ||M - C_X||_F. \tag{6}$$

This can be achieved by taking an SVD approximation of C_X as in eqn. (2). Utilizing a lemma stated in [4] that says Gaussians are closed under affine transformations. If we have $z \sim \mathcal{N}(0, I)$ then if X has covariance matrix, C_X, we define $x = C_X z$. From this lemma if we use the SVD of C_X, it follows that we can write our random variable x as

$$x = V^* \Sigma \xi, \;\; \xi \sim \mathcal{N}(0, I) \tag{7}$$

where the equality is in a distributional sense. Hence our data can be represented as a linear combination of rank 1 Gaussians with variance σ_j,

$$\underline{x} = \sum_{j=0}^{d-1} \xi_j \sigma_j \underline{v}_j^*, \;\; \xi_j \sim \mathcal{N}(0, 1). \tag{8}$$

2.3. **Linear Regression.** Linear Regression is a method of supervised learning which fits a function, f with the real solution Y. Linear Regression takes f to be a hyperplane,

$$f(\underline{x}) = \beta_0 + \sum_{j=1}^{d} \beta_{j+1} x_j. \tag{9}$$

This function seeks to be the solution to the problem,

$$f = \text{argmin}_{\beta \in \mathbb{R}^d} \frac{1}{2\sigma^2} ||A\beta - Y|| \tag{10}$$

with $A = [1|X^T]$ where X contains our data as defined in Sec. 1. Eqn. (10) can be solved via the normal equation. For more information see [4]

3. Algorithm Implementation and Development

3.1. **Set up.** We use Python version 3.8 [10] with SciKit-learn [7], NumPy [1], and MatplotLib [5] libraries and Sypder as our IDE [8]. Then general outline is to perform a PCA on a training set, determine reasonable number of PCA modes to use for approximation, construct a linear regression with the training data, and predict the digit's labels on the test data.

3.1.1. *Scikit-learn.* We describe the Scikit-learn functions used and explain what they compute. The main tool is the Scikit-learn class *Decomposition* and it's method *PCA*. *PCA* centers the data and uses SVD to decompose the data. We use the *fit_transform*, *transform*, and *inverse_transform* methods associated with *PCA*. First consider *fit_transform* (or *fit* then *transform*). The *fit* method centers the data and computes the PCA modes (among other things like singular values). The *transform* method projects the data onto our lower dimensional basis. The *inverse_transform* performs a change of basis back into your original basis.

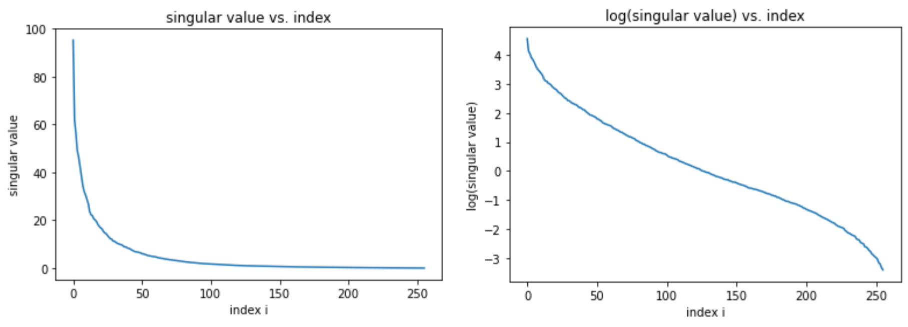
We also use the *linear_model* class and its *LinearRegression* method. The *fit* method of *LinearRegression* computes the coefficients and intercept of a hyperplane that minimizes the sum of squares between the linear model and our data as in Sec. 2.3.

To compute the mean squared error of the difference between the actual values and approximated values we use the Scikit-learn's *metrics* class and the *mean_squared_error* method.

3.2. **Implementation.** We load in our testing and training data of size $(500, 256)$ and $(2000, 256)$, respectively, along with their corresponding label sets. The data is shaped where a row is an image of 16×16 pixel images reshaped into a 1×256 array.

Using NumPy's *linalg* class and *svd* method we compute and plot the singular values of our training data to determine an appropriate rank (See Sec. 4.1). In order to carry out the binary classification we relabel digits 1 or -1 corresponding to our desired digit pairs as suggested by [3]. We use a *for* loop and *if* statement to loop through our test and training labels to identify the desired digit pair indices then extract them to create a new array of data and labels for our digit pair test and training sets. In the *PCA* method we provide the argument of number of PCA modes. We use *fit_transform* to project our digit pair training data onto the PCA basis. We then use *LinearRegression*'s *fit* method with arguments being the projected digit training data and corresponding labels to compute the coefficients and intercept of our hyperplane. Now we project our digit pair testing data onto the basis formed by the training data by using *PCA*'s *transform* method with the digit pair testing data as it's argument.

To classify, we take our equation for our hyperplane and provide it with the projected data for the test and training sets and we round the values of the predicted label to ± 1 depending on the predicted sign. We compute the mean square error with the *mean _square _error* method with the predicted labels (i.e. output of the hyperplane equation acting on the projected data) and the actual labels.



(A) σ_i vs i (B) $\log(\sigma_i)$ vs i

FIGURE 1. Singular Values, σ_i vs. Corresponding Index i

# of coefficients, ν	$1 - \frac{\|\|X - X_\nu\|\|_F}{\|\|X\|\|_F}$	# of coefficients, ν	$\frac{\|\|X_\nu\|\|_F}{\|\|X\|\|_F}$
5	34.5%	1	50.01%
16	60.56%	3	66.66%
38	80.27%	7	81.27%
65	90.02%	14	90.52%

TABLE 1. Bounds for low rank approximations to training data matrix.

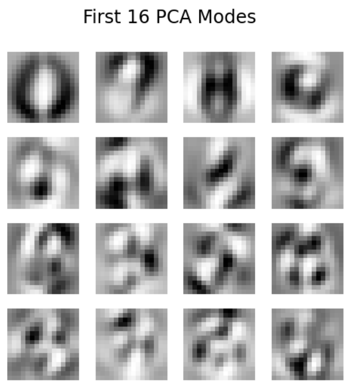
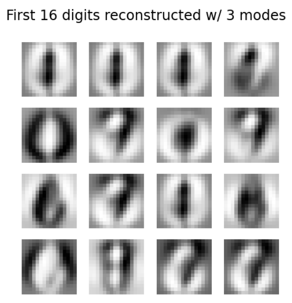


FIGURE 2. First 16 PCA Modes of training data matrix.

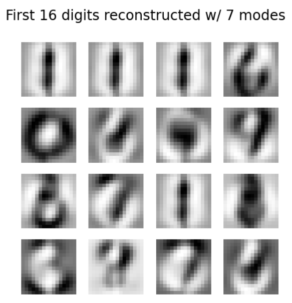
4. COMPUTATIONAL RESULTS

4.1. **Dimension Reduction and PCA.** Our first task is to determine the number of PCA modes needed for a reasonable low rank approximation. In Fig. 1 we plot the singular values vs their corresponding index on it's natural and logarithmic scale. In Fig. 1a we see a large decrease just before $i = 50$ so we might expect somewhere in this area would be a reasonable since this contains most of the variance of the data. In order to determine how well our low rank matrix, X_ν, approximates our data we have two different ways of viewing this. We can use Thm. 1 to compute the norm of the difference of our training matrix, X, and it's low rank approximation which is shown in the left side of table Table 1. We can also look at the variance explained that is retained by our low rank approximation which is on the right side of Table 1. These provide us various PCA modes to try approximating our data with and give us a rough idea about how accurate we should expect them to be.

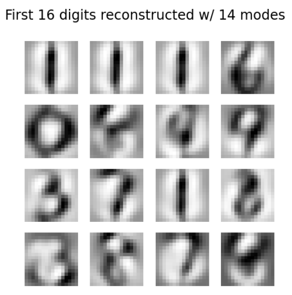
In Fig. 2, we plot the first 16 PCA modes which correspond to the left singular vectors of our training data matrix equivalently the eigenvectors of our covariance matrix. We would now like to use these modes to reconstruct our data. In Fig. 3, we present the first 16 digits of our training set which are reconstructed with multiple PCA modes. We notice that for modes less than 14 it is difficult to tell what digit it is since we've ditched a lot of the structure. For modes larger than 16 we see that the structure of the image isn't changing,



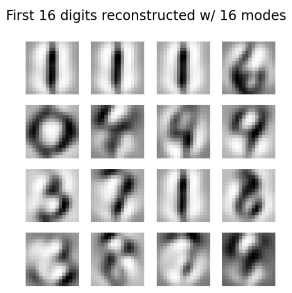
(A) 3 PCA modes



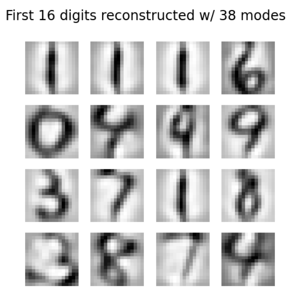
(B) 7 PCA modes



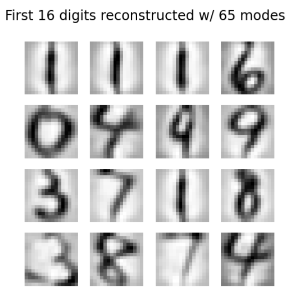
(C) 14 PCA modes



(D) 16 PCA modes



(E) 38 PCA modes



(F) 65 PCA modes

FIGURE 3. First 16 digits of the training set reconstructed with with various number of PCA modes.

# of Modes	Digit Pair	Train MSE	Test MSE
3	(1,8)	.05275	.08333
3	(3,8)	.19373	.77778
3	(2,7)	.09117	.20408
7	(1,8)	.03516	.08333
7	(3,8)	.13675	.55555
7	(2,7)	.022792	.04082
16	(1,8)	.00879	.041667
16	(3,8)	.06838	.5
16	(2,7)	.04558	.04082
38	(1,8)	0	.041667
38	(3,8)	.079772	.44444
38	(2,7)	.0114	.04082

TABLE 2. Mean Squared Error for testing and training sets with different pair of digits for various amounts of PCA modes used to approximate the training data.

but the becoming crisper. We expect this as we are adding more dimensions and this extra information isn't related to the underlying structure.

4.2. **Binary Classifcation.** We now wish to test the ability of our dimension reduction by having it classify data in the test set which it has not seen. We perform the PCA and construct a lower dimensional basis. We take our reassigned labels with the corresponding training data and construct a hyperplane with a linear regression. We then project our test data onto our training sets PCA basis and our linear regression computes a predicted label. In both cases of training and testing we measure the linear regressions ability to correctly assign a label by using the mean squared error between the computed label and the actual label. Table 2 reports these errors for different pairs and different number of PCA modes used.

We order the accuracy of the test set from best to worse as: (1,8),(2,7),(3,8). A reasonable explanation is that 3 and 8 have similar structure. We expect the pair (1,8) would have the lowest error since they have little similarity in their handwritten structure. The general trend is that error decreases with more modes. The increase is strict for the (3,8) pair. This is because we need more modes to pick up the smaller differences between them to correctly classify them. Since their structure is similar these small differences are more important. For the (1,8) pair we see that the test MSE begins to level out after 16 modes We observe this same trend in the (2,7) pair. This may be due to the fact that we are not able to capture any more relevant information and we may be picking up redundant information in the low rank structure or possibly picking up noise/information that is not related to the true structure. For reference, here is the equation of the hyperplane for 3 PCA modes on the (1,8) pair:

$$Y_p = .413501x_1 - .100768x_2 - .278289x_3 - 0.221978 \tag{11}$$

where Y_p is the predicted value and x_i is the entry in the i^{th} column for a particular row representing a low rank image.

5. Summary and Conclusions

Using the Singular Value Decomposition and the Principal component analysis we have shown that we approximate (in 2 different senses) up to a given accuracy with different numbers of PCA modes. With only 16 PCA modes we can reconstruct the images of the handwritten digits that are readable to the human eye. If we use 38 PCA modes we can use linear regression to classify pairs of handwritten digits with reasonable accuracy (in mean squared error sense). We showed that structural features such as how 3 and 8 are written can impact the accuracy of our classifier and need a greater number of modes to yield a better prediction.

Future directions that we would like to consider are other classification tasks such as clustering with the MNIST data set. We would also like to experiment with using the Proper Orthogonal Decomposition application of the SVD to look at low rank structure of dynamic data.

Acknowledgements

The author thanks fellow classmate Daniel Leon for helpful discussions about theoretical aspects about evaluating the amount of approximation from low rank structure and referring the author to additional relevant literature. The author also thanks Dr. Hosseini for the nice plotting function in [2].

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. Amath 582 assignment 2 helper code. Code to start assignment 2 of Amath 582.
[3] B. Hosseini. Amath 582 assignment 2 sheet. Sheet that contains relevant information for Assignment 2.
[4] B. Hosseini. Amath 582 course notes. Notes for Computational Methods for Data Analysis.
[5] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.
[6] J. N. Kutz. *Data-Driven Modeling and Scientific Computation.* Oxford University Press, 2013.
[7] 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.
[8] P. Raybaut. Spyder-documentation. *Available online at: spyder-ide.org*, 2009.
[9] L. N. Trefethen and D. Bau III. *Numerical linear algebra*, volume 50. Siam, 1997.
[10] G. Van Rossum and F. L. Drake. *Python 3 Reference Manual.* CreateSpace, Scotts Valley, CA, 2009.