

# CS5785: Assignment #2

Due on Wednesday, September 27, 2017

*Prof. Serge Belongie*

**Travis Allen and Noshin Nisa**

September 28, 2017

# 1 Introduction

In this experiment, we complete two programming exercises and written exercises to explore Singular Value Decomposition (SVD) as well as various classification methods.

In our first programming exercise, we use Singular Value Decomposition to obtain "eigenfaces" from our training images; these can be thought of as feature faces that represent the important features from the training data, ordered based on importance. We use the results of this SVD to approximate our original training data, and also use the results to attempt to classify test data, and observe how these approximations and classifications are influenced by the parameter  $r$ ; this can be seen as how many of the "top" eigenfaces we use.

In the second programming exercise, we work with a training dataset of dishes and their ingredients, labeled based on their cuisine. We compare Naive Bayes Classifier (assuming both Gaussian and Bernoulli distributions, respectively) and Logistic Regression classifiers as models to predict cuisines based on ingredients, and explore why some are better than others with different type of data.

Lastly, we explore the mathematical concepts that underly our programming exercises: the Eigenvalue problem, Singular Value Decomposition, as well as the process by which many classification models are fit.

The methods and results of our experiments are discussed below.

## 2 Eigenface for face recognition.

(b) The code used for this task was borrowed from the sample code provided. This code goes through each line in the txt file, which each represent a path to an image and its corresponding label. The `scipy imread` method reads an image into a 50x50 array - [1], and the label is added to a separate labels array. This array is then flattened into an array of size 2500 using the `numpy reshape` method - [1], and appended to the 2D-array that represents the dataset we will use. This is done for both the training data and the test data. We use the `pylab` library - [2] to plot and display an arbitrary image from the training and test data; these are displayed as Figure 1a and Figure 1b, respectively.

(c) The `numpy .mean` method - [1] is used to calculate the average face. Passing the parameter `axis=0` to the method specifies that we wish to calculate the mean of the values of each column (which in this case represent a pixel), separately. The resultant array is a flat array with  $p$  entries, where each entry represents the mean for a given column. This average face is displayed in Figure 1c.

(d) The `numpy .subtract` method - [1] allows us to easily subtract the "average face" image from each image in the training data. When passed a 2D `numpy` array (our training data) and a 1D `numpy` array (the average face array), the 1D array is subtracted from each row in the 2D array and the resultant 2D array is returned. This centers our image data around our mean face. An arbitrary row in the resultant matrix is selected for display. This image is displayed in Figure 1d.

(e) To perform a Singular Value Decomposition on the training data, the numpy .svd method - [1] is called. This returns three objects: the left singular vectors, the singular values for every matrix, and the right singular vectors. When the singular values are converted into a diagonal matrix, matrix multiplication can be done to get the original matrix (in this case, the training data) back; this is shown in the code. Each row in the resultant right singular vectors matrix is called an eigenface. An example of an eigenface is displayed in Figure 1e.

(f) For every value of  $r$  between 1 and 200, we perform an  $r$ -rank approximation of the training data. This is done by first extracting the first  $r$  columns from the left singular vectors, the first  $r$  elements of the diagonal matrix, and the first  $r$  rows of the transposed right singular vectors, and calculating the dot product of the 3 (in that order). The resultant dot product is subtracted from the original training data set, and this difference is normalized using the numpy's linalg norm method [1], which can be passed the 'fro' parameter to calculate the Frobenius norm. For each  $r$ , this normalized difference between the training data and the  $r$ -rank approximation matrix is plotted against the  $r$  value, and is displayed using the matplotlib. [3], displayed in Figure 1f. This plot reveals that as you increase the  $r$  value, the normalized difference decreases; this indicates that the approximation gets closer and closer with a higher  $r$ .

(g) We create a method that accepts two parameters: the image set that we will use as our source, and the  $r$  value to specify the dimensions of the feature matrix that will result. We begin by extracting the first  $r$  rows from the training data's transposed right singular vectors. We then transpose this resultant matrix to obtain our feature matrix. This method was made generic so that it can be used for both test data and training data, as we will see in part (h).

(h) Our method calculates the accuracy with which we can train a logistic regression model on an  $r$ -dimensional feature matrix to classify an  $r$ -dimensional test matrix. To do this, we first use the method in part (g) to create  $r$ -dimensional feature matrices from the training and test data, respectively. We then instantiate scikitlearn's LogisticRegression model and specify that there will be more than just two classes and that we want to do a one-vs-rest classification (Standard Logistic Regression is used for binary data, so we specify that it should treat matches as "positive" and non-matches as "negative" as the two classes. [5] ) by passing the parameter multi\_class='ovr'. [4] We then fit the model on the training feature matrix and compute a score on the test feature matrix.

When  $r=10$ , our model performs with an 79% accuracy. We plot the classification accuracy as a function of  $r$  in Figure 1h. At  $r=100$ , this accuracy increase to 93%.

### 3 What's Cooking?

(a) The competition gives a set of training data and testing data. It provides us with a set of cuisine with their respective ingredients in it. We need to create a model which can detect

the cuisine based on the ingredients given to it.

(b) There are 39774 sample dishes in the training set with 20 different unique cuisines. All the cuisines combined have 6714 number of unique ingredients.

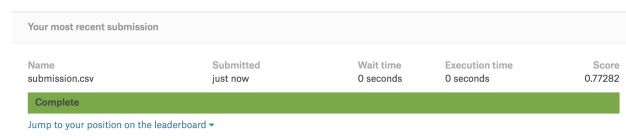
(c) Each dish is represented by a binary ingredient feature vector giving us a matrix of  $n \times d$  where  $n$  is the number of dishes. If the ingredient is present it is coded as 1 and if not it is coded as 0. This gives us a binary list of ingredients present and absent.

(d) Naive Bayes Classifier with 3 fold cross validation has been done on the matrix. This is a very simple probabilistic classifier which has strong independent assumption between different features. Gaussian distribution assumes features to be continuous, and Bernoulli distribution assumes them to be discrete binary features. For Gaussian average accuracy is 37.98% and for Bernoulli it is 68.35%.

(e) Bernoulli prior did better than Gaussian prior. Gaussian prior expects to receive continuous features and assumes there is a relation between the given features. In case of Bernoulli it assumes that the features are independent of each other. Since we have discrete values of 0 and 1, Bernoulli performs better in this scenario.

(f) After performing logistic regression on the training dataset using 3 fold cross validation, the average accuracy received is 77.55%

(g) Out of Bernoulli, gaussian and logistic classifiers, logistic regression performed the best with an average accuracy of 77.55%. Logistic regression assumes all the features are completely independent of each other and with discrete values it performed well. Even though Bernoulli assumes the same it still tries to find correlation between features. When the model was run against Kaggle training set it performed with an accuracy of 77.28%.



Your most recent submission				
Name	Submitted	Wait time	Execution time	Score
submission.csv	just now	0 seconds	0 seconds	0.77282
Complete				
<a href="#">Jump to your position on the leaderboard</a>				

## 4 Written Exercises

(1) The answers to question were written by hand and are attached below. This solution references equations provided in the textbook Elements of Statistical Learning. [6]

(2) The answers to question were written by hand and are attached below.

(a and b) Answers reference solutions found at [7]

(c, d, and e) Answers reference solutions found at [8]

(3) The answers that were calculated by hand are specified. Those answers that were calculated using python can be found in the "Written Exercise 3 Workbook" associated with this

report.

(a) The  $M^*M$  and  $MM^*$  dot products were calculated by hand. These are attached below.

(b and c)

The eigenvalues for  $M^*M$  (referred to as Matrix A) and  $MM^*$  (referred to as Matrix B) are calculated using numpy's linalg eig method. This method returns an array of eigenvalues and a matrix of eigenvectors; the  $i$ -th column of the resultant matrix corresponds to the eigenvector associated with the  $i$ -th element in the eigenvalues array. [1].

The numpy method used return eigenvalues (and corresponding eigenvectors) that were small enough to be considered 0. When Wolfram's eigenvalue calculator was used, these values were 0. As a result, we made a decision to remove these near-0 eigenvalues and the associated eigenvectors. Thus, we were left with eigenvalues in descending order, and a matrix of the associated eigenvectors.

The values are printed in the code, and are all used for the Singular Vector Decomposition that is done in part (d).

(d) The SVD was done by hand, using the values calculated in (a), (b), and (c). This work is displayed below.

(e) The one-dimensional approximation was calculated using numpy, and can be found in the attached workbook.

## 5 Analysis

Our experiments with the Face Image database demonstrate the power of Singular Value Decomposition in approximating data. As a test, we trained a logistic regression model on all of our training data and made predictions on the test data with a 95% accuracy. After performing an SVD on the training data and using the resultant eigenfaces to generate a feature matrix for training and testing, we calculate our classification accuracy. When the top 10 eigenfaces are selected, our score drops to 79%; when this is increased to 100, we achieve a 93% accuracy.

Our experiments with the cuisine data compare classification methods. We test Naive Bayes Classifiers using two different prior distribution assumptions: first Gaussian distribution, then Bernoulli distribution. The Naive Bayes Classifier performs better when assuming a Bernoulli distribution rather than Gaussian; this can be explained by the fact that Gaussian distributions are better-suited to represent continuous data rather than the discrete data that we have. Logistic Regression for classification exceeds both of these Naive Bayes methods. While Naive Bayes method assumes that all of the features are independent of one another, logistic regression does not make this assumption. In the case of ingredients, it seems intuitive that some features depend on one another, because use of one ingredient will influence whether another is used. It is therefore not surprising that Logistic Regression

performs the best of the classification models tested.

## References

- [1] *SciPy and numpy library documentation*, available at <https://scipy.org/>
- [2] *PyLab Documentation*, available at <https://scipy.github.io/old-wiki/pages/PyLab.html>
- [3] *Matplotlib's Pyplot documentation*, [https://matplotlib.org/api/pyplot\\_api.html](https://matplotlib.org/api/pyplot_api.html)
- [4] *scikit-learn docs*, found at <http://scikit-learn.org/stable/>
- [5] *Logistic Regression article*, found at [https://en.wikipedia.org/wiki/Logistic\\_regression](https://en.wikipedia.org/wiki/Logistic_regression)
- [6] *Elements of Statistical Learning* T. Hastie, R. Tibshirani and J. Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd edition), Springer-Verlag, 2008.
- [7] *John L. Weatherwax Solution Manual*, found at [http://www.waxworksmath.com/Authors/G\\_M/Hastie/WriteUp/Weatherwax\\_Epstein\\_Hastie\\_Solu](http://www.waxworksmath.com/Authors/G_M/Hastie/WriteUp/Weatherwax_Epstein_Hastie_Solu)
- [8] *Andrew Tulloch ESL partial solutions*, found at <http://ajtulloch.github.io/2012/elements-of-statistical-learning-chapter-4-solutions>

## 6 Figures

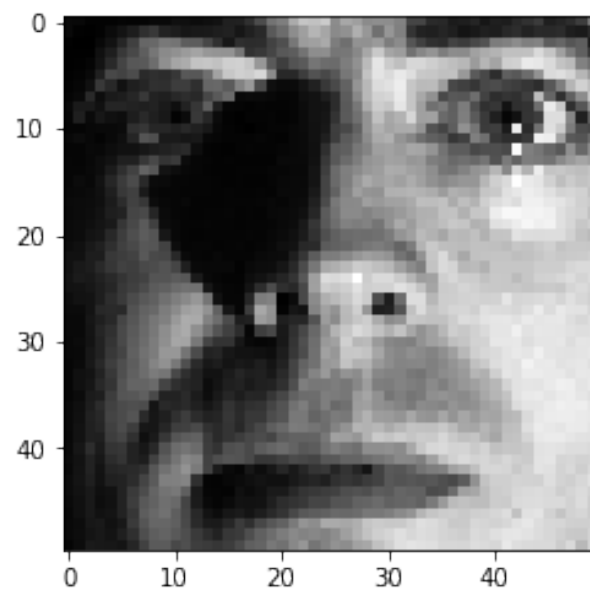


Figure 1: figure1a

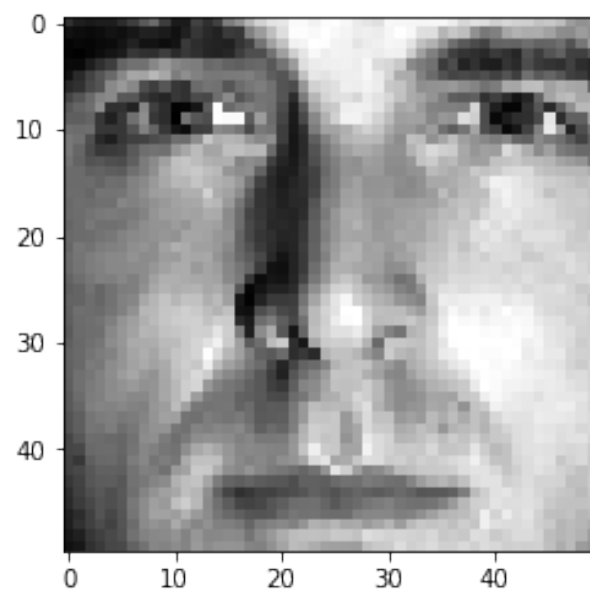


Figure 2: figure1b

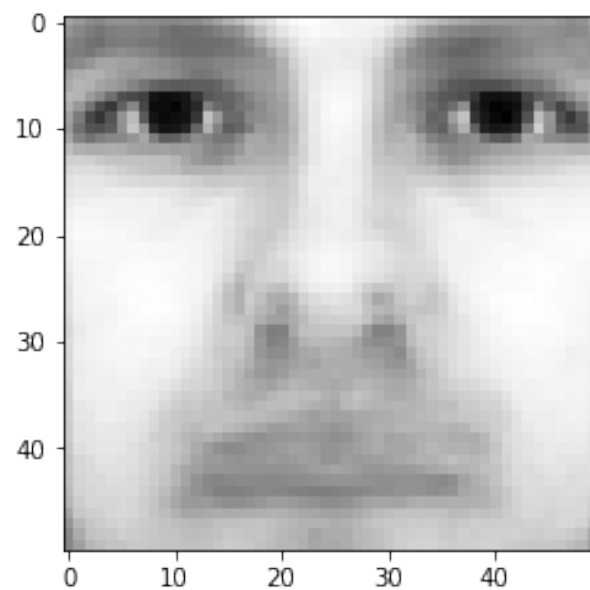


Figure 3: figure1c

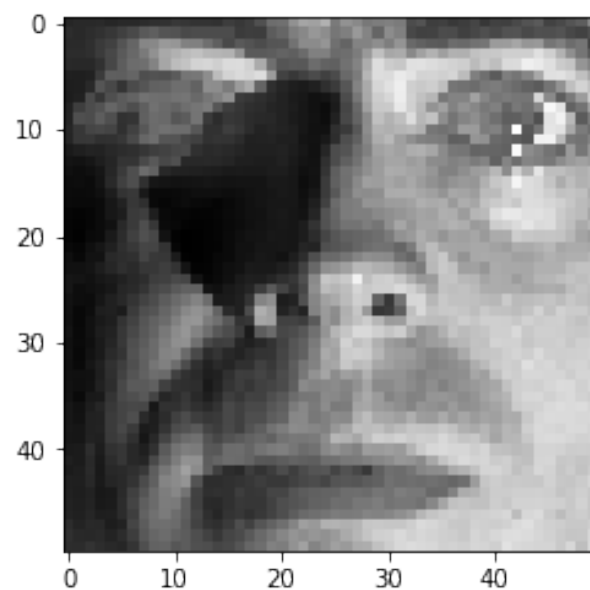


Figure 4: figure1d



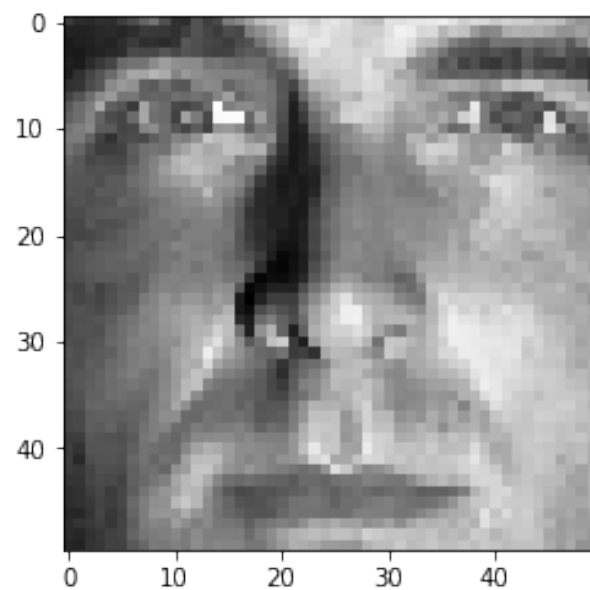


Figure 5: figure1e

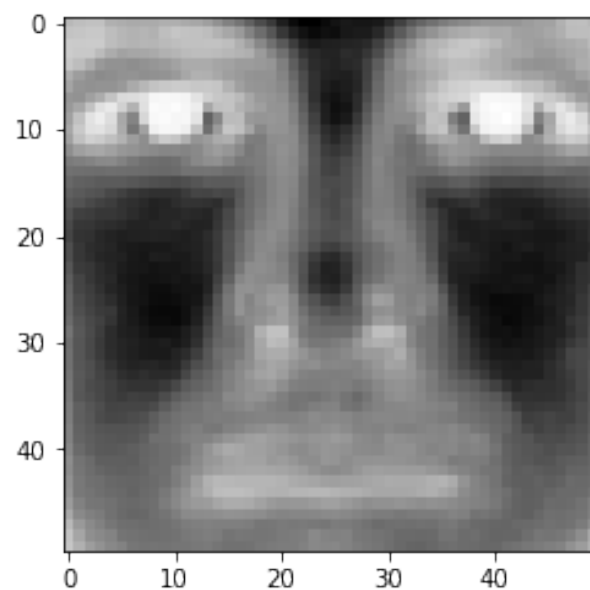


Figure 6: figure\_1f

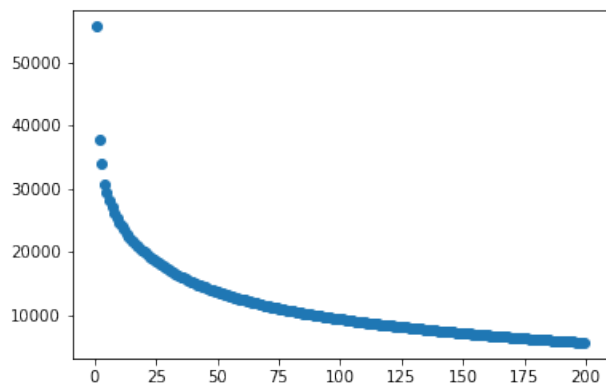


Figure 7: figure\_1g

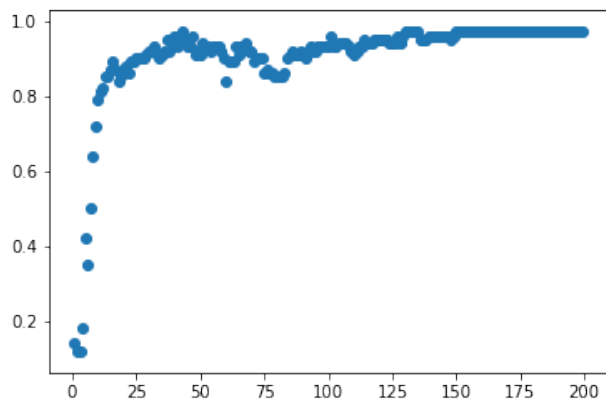


Figure 8: figure1h

1. According to Lagrange multiplier,

$$l(\lambda) = f(x) - \lambda g(x)$$

$$f(a) = a^T B a$$

$$g(a) = a^T W a - 1 = 0 \quad (\text{evaluating } g(a) = 0 \text{ according to the law})$$

$$l(\lambda) = a^T B a - \lambda (a^T W a - 1)$$

$$\frac{dl}{da} = \frac{d}{da} (a^T B a) - \lambda \frac{d}{da} (a^T W a - 1)$$

$$= (1 \times B a) + a^T (B \times 1) - \lambda (1 \times W a + a^T W \times 1 - 0)$$

$$= B a + a^T B - \lambda (W a + a^T W)$$

$$= B a + B^T a - \lambda (W a + W^T a) \quad [B \& W \text{ are co-variance \& symmetrical}]$$

$$= a(B + B^T) - \lambda (W + W^T) a$$

To find the maxima, let it equal to 0.

$$a(B + B^T) - \lambda a(W + W^T) = 0$$

$[(W + W^T)]^{-1} (B + B^T) a = \lambda a$ . Assuming  $W$  &  $B$  are symmetric, this is a standard eigenvalue problem.

2(a). According to equation 4.33 from the textbook for the LDA rule to classify to class 2, the ratio of posterior probability of  $N_2$  &  $N_1$  should be greater than 1.

$$\log \frac{Pr(G=K | X=x)}{Pr(G=K' | X=x)} > 0 \quad \text{where } K=N_2 \text{ \& } K'=N_1$$

$$\log \frac{N_2}{N_1} - \frac{1}{2} (\mu_2 + \mu_1)^T \Sigma^{-1} (\mu_2 - \mu_1) + x^T \Sigma^{-1} (\mu_2 - \mu_1) > 0$$

$$x^T \Sigma^{-1} (\mu_2 - \mu_1) > \frac{1}{2} (\mu_2 + \mu_1)^T \Sigma^{-1} (\mu_2 - \mu_1) - \log(N_2/N_1)$$

Therefore, substituting the target, we get

$$x^T \Sigma^{-1} (\mu_2 - \mu_1) > \frac{1}{2} \mu_2^T \Sigma^{-1} \mu_2 - \frac{1}{2} \mu_1^T \Sigma^{-1} \mu_1 + \log \frac{N_1}{N} - \log \frac{N_2}{N}$$

4.2(b) In order to minimize the given expression, it is important to satisfy the normal equation which is  $X^T X \begin{bmatrix} \beta_0 \\ \beta \end{bmatrix} = X^T y$  — eq(1)

Expanding the left hand side of the equation for  $X^T X$

$$\begin{bmatrix} 1 & 1 & 1 & \dots & 1 & 1 & 1 & \dots & 1 \\ x_1 & x_2 & x_3 & \dots & x_{N_1} & x_{N_1+1} & x_{N_1+2} & \dots & x_{N_1+N_2} \end{bmatrix} \begin{bmatrix} 1 & x_1^T \\ 1 & x_2^T \\ \vdots & \vdots \\ 1 & x_N^T \\ 1 & x_{N_1+1}^T \\ 1 & x_{N_1+2}^T \\ \vdots & \vdots \\ 1 & x_{N_1+N_2}^T \end{bmatrix}$$

Multiplying the two matrix we get:

$$\begin{bmatrix} N & \sum_{i=1}^N x_i^T \\ \sum_{i=1}^N x_i & \sum_{i=1}^N x_i x_i^T \end{bmatrix} \text{ — (2)}$$

when our response is coded as  $-N/N_1$  &  $+N/N_2$  for the classes, the right hand side of eq(1) can be written

as:

$$\begin{bmatrix} 1 & 1 & \dots & 1 & 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_{N_1} & x_{N_1+1} & x_{N_1+2} & \dots & x_{N_1+N_2} \end{bmatrix} \begin{bmatrix} -N/N_1 \\ -N/N_1 \\ \vdots \\ -N/N_1 \\ N/N_2 \\ N/N_2 \\ \vdots \\ N/N_2 \end{bmatrix}$$

After multiplying the two matrices we get:

$$\begin{bmatrix} N_1 \left( -\frac{N}{N_1} \right) + N_2 \left( \frac{N}{N_2} \right) \\ \left( \sum_{i=1}^{N_1} x_i \right) \left( -N/N_1 \right) + \left( \sum_{i=N_1+1}^N x_i \right) \left( N/N_2 \right) \end{bmatrix} = \begin{bmatrix} 0 \\ -N\mu_1 + N\mu_2 \end{bmatrix}$$

By introducing class specific means we get.

$$\sum_{i=1}^N x_i = \sum_{i=1}^{N_1} x_i + \sum_{i=N_1+1}^N x_i = N_1\mu_1 + N_2\mu_2 \quad (3)$$

By pooling all of the samples for both class,  $K=2$ , we can estimate the covariance matrix  $\hat{\Sigma}$

$$\begin{aligned} \hat{\Sigma} &= \frac{1}{N-K} \sum_{k=1}^K \sum_{i:g_i=k} (x_i - \mu_k)(x_i - \mu_k)^T \\ &= \frac{1}{N-2} \left[ \sum_{i:g_i=1} (x_i - \mu_1)(x_i - \mu_1)^T + \sum_{i:g_i=2} (x_i - \mu_2)(x_i - \mu_2)^T \right] \\ &= \frac{1}{N-2} \left[ \sum_{i:g_i=1} x_i x_i^T - N_1 \mu_1 \mu_1^T + \sum_{i:g_i=2} x_i x_i^T - N_2 \mu_2 \mu_2^T \right] \end{aligned}$$

Following equation (2)

$$\sum_{i=1}^N x_i x_i^T = (N-2) \hat{\Sigma} + N_1 \mu_1 \mu_1^T + N_2 \mu_2 \mu_2^T$$

Writing both side of (1) as linear system

$$X^T X \begin{bmatrix} \beta_0 \\ \beta \end{bmatrix} = X^T y$$

$$\begin{bmatrix} N & N_1 \mu_1^T + N_2 \mu_2^T \\ N_1 \mu_1 + N_2 \mu_2 & (N-2) \hat{\Sigma} + N_1 \mu_1 \mu_1^T + N_2 \mu_2 \mu_2^T \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta \end{bmatrix}$$

$$\stackrel{(3)}{=} \begin{bmatrix} 0 \\ -N\mu_2 + N\mu_1 \end{bmatrix} \quad \text{--- (4)}$$

$$\Rightarrow N\beta_0 + (N_1 \mu_1^T + N_2 \mu_2^T)\beta = 0$$

Solving for  $\beta_0$  in terms of  $\beta$ .

$$\beta_0 = \left( -\frac{N_1}{N} \mu_1^T - \frac{N_2}{N} \mu_2^T \right) \beta \quad \text{--- (5)}$$

Putting this in eq (4)

$$\begin{aligned} & (N_1 \mu_1 + N_2 \mu_2) \left( -\frac{N_1}{N} \mu_1^T - \frac{N_2}{N} \mu_2^T \right) \beta + \\ & \left( (N-2) \sum + N_1 \mu_1 \mu_1^T + N_2 \mu_2 \mu_2^T \right) \beta = N (\mu_2 - \mu_1) \end{aligned}$$

Outer product terms

$$\begin{aligned} & = -\frac{N_1^2}{N} \mu_1 \mu_1^T - \frac{2 N_1 N_2}{N} \mu_1 \mu_2^T - \\ & \quad - \frac{N_2^2}{N} \mu_2 \mu_2^T + N_1 \mu_1 \mu_2^T + N_2 \mu_2 \mu_1^T \\ & = \left( -\frac{N_1^2}{N} + N_1 \right) \mu_1 \mu_1^T - \frac{2 N_1 N_2}{N} \mu_1 \mu_2^T + \left( -\frac{N_2^2}{N} + N_2 \right) \mu_2 \mu_2^T \\ & = \frac{N_1}{N} (-N_1 + N) \mu_1 \mu_1^T - \frac{2 N_1 N_2}{N} \mu_1 \mu_2^T + \frac{N_2}{N} (-N_2 + N) \mu_2 \mu_2^T \\ & = \frac{N_1 N_2}{N} \mu_1 \mu_1^T - \frac{2 N_1 N_2}{N} \mu_1 \mu_2^T + \frac{N_2 N_1}{N} \mu_2 \mu_2^T \\ & = \frac{N_1 N_2}{N} (\mu_1 \mu_1^T - 2 \mu_1 \mu_2^T - \mu_2 \mu_2^T) = \frac{N_1 N_2}{N} (\mu_1 - \mu_2) (\mu_1 - \mu_2)^T \end{aligned}$$



Here we have  $N_1 + N_2 = N$ . If we introduce the matrix

$$\hat{\Sigma}_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T$$

Equation for  $\beta$ , 
$$\left[ (N-2)\hat{\Sigma} + \frac{N_1 N_2}{N} \hat{\Sigma}_B \right] \beta = N(\mu_2 - \mu_1)$$
 — (6)

---

2(c) Since  $\hat{\Sigma}_B \beta$  is  $(\mu_2 - \mu_1)(\mu_2 - \mu_1)^T \beta$ , and Product  $(\mu_2 - \mu_1)^T \beta$  is scalar, vector direction of  $\hat{\Sigma}_B \beta$  is given by  $\mu_2 - \mu_1$ , so as to as both the right hand side & the term

$\frac{N_1 N_2}{N} \hat{\Sigma}_B$  are in the direction of  $\mu_2 - \mu_1$ ,

the sol<sup>n</sup>  $\beta$  must be proportional to  $\hat{\Sigma}^{-1}(\mu_2 - \mu_1)$

---

2(d) If we use  $\alpha$  as arbitrary & defined, it proves the point.

2(e) If  $U_i$  is the  $n$  element vector with  $i^{\text{th}}$  element, our target value  $Y$  as  $t_1 U_1 + t_2 U_2$ .

$U_1 + U_2 = 1$ . Our estimates  $\hat{\mu}_1, \hat{\mu}_2$  as

$$x^T U_i = N_i \hat{\mu}_i \quad \& \quad x^T Y = t_1 N_1 \hat{\mu}_1 + t_2 N_2 \hat{\mu}_2$$

Solving for 2(a) we get.

$$\begin{aligned} \hat{\beta}_0 &= \frac{1}{N} 1^T (Y - x \hat{\beta}) \\ &= -\frac{1}{N} (N_1 \mu_1^T + N_2 \mu_2^T) \hat{\beta} \end{aligned}$$

We can then write our predicted value

$$\begin{aligned} \hat{f}(x) &= \hat{\beta}_0 + \hat{\beta}^T x \\ &= \frac{1}{N} (N x^T - N_1 \mu_1^T - N_2 \mu_2^T) \hat{\beta} \\ &= \frac{1}{N} (N x^T - N_1 \mu_1^T - N_2 \mu_2^T) \lambda \Sigma^{-1} (\hat{\mu}_2 - \hat{\mu}_1) \end{aligned}$$

for some  $\lambda \in \mathbb{R}$ , classification rule is  $\hat{f}(x) > 0$

$$\text{or, } N x^T \lambda \Sigma^{-1} (\hat{\mu}_2 - \hat{\mu}_1) > (N_1 \mu_1^T + N_2 \mu_2^T) \lambda \Sigma^{-1} (\hat{\mu}_2 - \hat{\mu}_1)$$

$$x^T \Sigma^{-1} (\hat{\mu}_2 - \hat{\mu}_1) > \frac{1}{N} (N_1 \mu_1^T + N_2 \mu_2^T) \Sigma^{-1} (\hat{\mu}_2 - \hat{\mu}_1)$$

$\therefore$  which different than LDA rule unless  $N_1 = N_2$

AML: 3

Rank: # of non-zero rows when you reduce a matrix to row-echelon form

①

$$M^T = \begin{bmatrix} 1 & 3 & 2 & 0 & 5 \\ 0 & 7 & -2 & -1 & 8 \\ 3 & 2 & 8 & 1 & 7 \end{bmatrix}$$

$$M^T M = \begin{bmatrix} 1 & 3 & 2 & 0 & 5 \\ 0 & 7 & -2 & -1 & 8 \\ 3 & 2 & 8 & 1 & 7 \end{bmatrix} \begin{bmatrix} 1 & 0 & 3 \\ 3 & 7 & 2 \\ 2 & -2 & 8 \\ 0 & -1 & 1 \\ 5 & 8 & 7 \end{bmatrix} = \begin{bmatrix} 39 & 57 & 60 \\ 57 & 118 & 53 \\ 60 & 53 & 127 \end{bmatrix}$$

$$M M^T = \begin{bmatrix} 1 & 0 & 3 \\ 3 & 7 & 2 \\ 2 & -2 & 8 \\ 0 & -1 & 1 \\ 5 & 8 & 7 \end{bmatrix} \begin{bmatrix} 1 & 3 & 2 & 0 & 5 \\ 0 & 7 & -2 & -1 & 8 \\ 3 & 2 & 8 & 1 & 7 \end{bmatrix} = \begin{bmatrix} 10 & 9 & 26 & 3 & 26 \\ 9 & 62 & 8 & -5 & 85 \\ 26 & 8 & 72 & 10 & 50 \\ 3 & -5 & 10 & 2 & -1 \\ 26 & 85 & 50 & -1 & 138 \end{bmatrix}$$

③ a) Singular values of  $M = \sqrt{\text{Eigenvalues of } MM^T}$

$$\therefore \text{Singular Values} = \left\{ \sqrt{214.670489196}, \sqrt{69.3295108039} \right\}$$

$$\therefore \Sigma = \begin{bmatrix} 14.651637765 & 0 \\ 0 & 8.32643445923 \end{bmatrix}$$

$V = \text{RIGHT SINGULAR VECTORS} = \text{Eigenvectors of } M^T M \text{ as columns}$

$$V = \begin{bmatrix} .42615127 & -.01460404 \\ .61500884 & -.72859799 \\ .66344497 & .68478587 \end{bmatrix}$$

$U = \text{Left Singular Vectors} = \text{Eigenvectors of } MM^T \text{ as columns}$

$$U = \begin{bmatrix} -.16492942 & .24497323 \\ -.47164732 & -.45330644 \\ -.33647055 & .82943965 \\ -.00330585 & .16974659 \\ -.79820031 & -.13310656 \end{bmatrix}$$

$$M = U \Sigma V^T$$

$$\therefore M = \begin{bmatrix} -.16492942 & .24497323 \\ -.47164732 & -.45330644 \\ -.33647055 & .82943965 \\ -.00330585 & .16974659 \\ -.79820031 & -.13310656 \end{bmatrix} \begin{bmatrix} 14.651637765 & 0 \\ 0 & 8.32643445923 \end{bmatrix} \begin{bmatrix} .42615127 & .61500884 & .66344497 \\ -.01460404 & -.72859799 & .68478587 \end{bmatrix}$$