CS5785: Applied Machine Learning **(Due: 10/10/19)**

Homework 2

Instructor: Serge Belongie *Names:* Eva Esteban Velasco, Simran Rajpal, *Netids:* epe26, sr2258

1 Programming Exercises

1.1 Eigenface for Face Recognition

Set Up

For this assignment we imported the following libraries: numpy [1], scipy [2], matplotlib [3], sklearn [4] and imageio [5].

(a) Download The Faces Dataset

We downloaded and unzipped The Face Dataset that contains the training and testing images, and analyzed them to understand the content.

(b) Load Train/Test Dataset and Show Examples

We used Jupyter Notebook for this assignment. Using a modification of the code provided, we loaded the training set into a matrix *train_data* of dimensions 540x2500. We picked the image in index 10 and displayed it using the *imshow*() function with the parameter $cmap = cm.Greys\_r$. We also loaded the testing images into a matrix *test_data* of dimensions 100x2500. Again, we picked the image in index 10 and displayed it using the *imshow*() function. These images can be found in Figure 1.

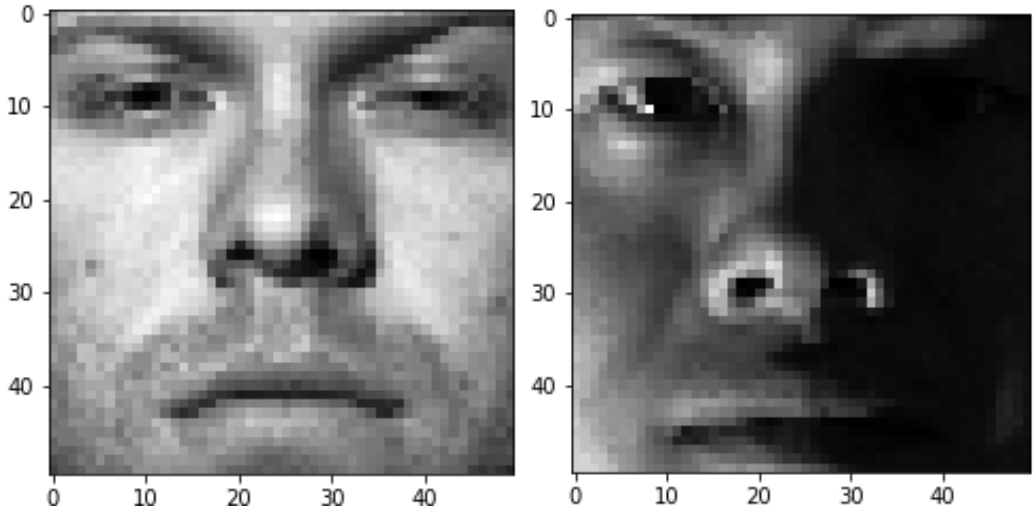

Figure 1: Training face image (left) and testing face image (right).

(c) Average Face

To compute the average face, we summed all the image values and divided them by the total number of images, which is 540. We then converted the image to grayscale and displayed it using the function *imshow*() with the parameter $cmap = cm.Greys\_r$. The average face image can be found on Figure 2.

(d) Mean Subtraction

We subtracted the average face image matrix from the training images array, and displayed the result in grayscale using *imshow*() with the parameter $cmap = cm.Greys\_r$. We followed the same steps for the testing data. The resulting images are shown on Figure 3.

(e) Eigenface

To perform Singular Value Decomposition (SVD) we used the function *linalg.svd*() from the scipy library. We then displayed the first 10 images in V^T (the first 10 eigenfaces) using *imshow*() with the parameter $cmap = cm.Greys\_r$. The results can be found on Figure 4.

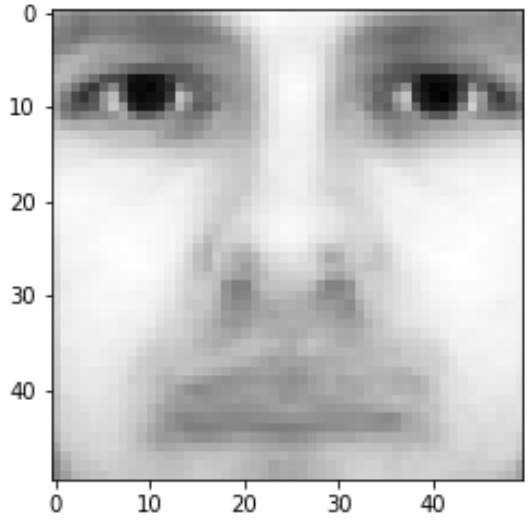

Figure 2: Grayscale average face image.

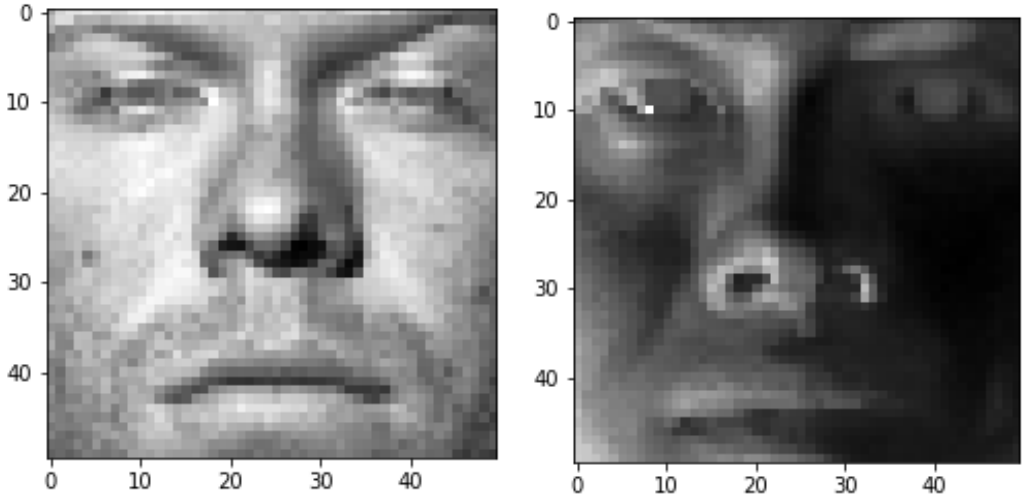

Figure 3: Training data minus average image (left) and testing data minus average image (right).

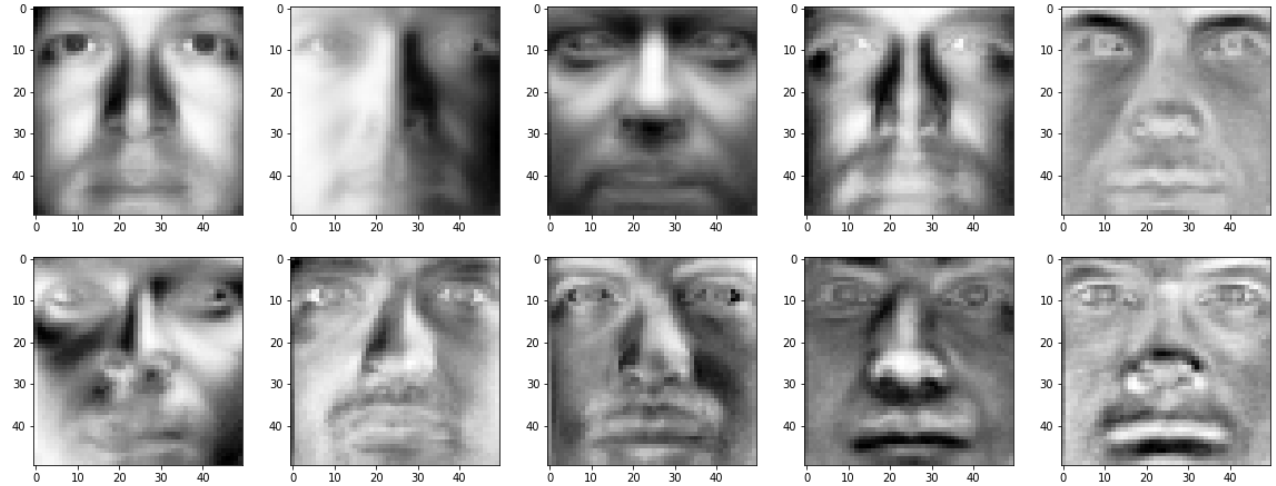

Figure 4: First 10 SVD images.

(f) Low-rank Approximation

To calculate the Low-rank Approximation, we looped from r=1 to r=200, calculating the product between the first r diagonal elements in Σ with the first r columns in U and the first r rows in V. This gave a rank-r approximation of the *train_data* matrix, which we then subtracted from *train_data*. We then computed the Frobenius norm of this difference to get the rank-r approximation error. Figure 5 shows a plot of the approximation error for every r value. As we can see, the approximation error decreases exponentially as the parameter r increases. This is expected because for higher values of r we use a higher number of dimensions to reconstruct the image.

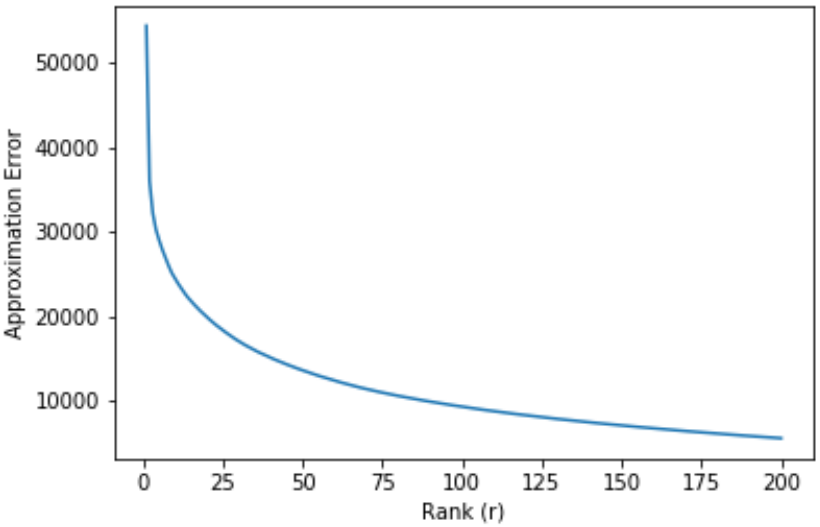

Figure 5: Rank-r Approximation Error plot.

(g) Eigenface Feature

We wrote a function named *generateFMatrix* that takes in the training images, the testing images, and r as parameters. It returns the Eigenface Feature matrices *F_training* and *F_testing* for the training and testing data, respectively. It calculates F by multiplying the training or testing data by the transpose of the first r rows of V^T. The complete code for this function can be found on the IPythonNotebook file 'Homework2_Code'. We run the function on some example data and checked the matrices dimensions to ensure the correct functioning of the code.

(h) Face Recognition

We extracted the training and testing features for r=10 using the *generateFMatrix* function, and trained a Logistic Regression model with the training data F matrix. To implement 'one-vs-rest' logistic regression, we set logistic regression to 'ovr' mode. We then calculated our model's predictions for the testing data F matrix for r=10. The accuracy of our model was 79%.

We repeated this process for every r value in the range 1 to 200 to obtain the Rank-r Classification Accuracy. The results can be found on Figure 6. It can be seen that the plot follows an increasing logarithmic trend. The accuracy of the classifier increases fast from almost 0 to around 90 % between the r values of 20 and 35 and increases much slower afterwards, almost reaching 100 % at r=200. The code to generate this and the previous graph can be found on Appendix A.

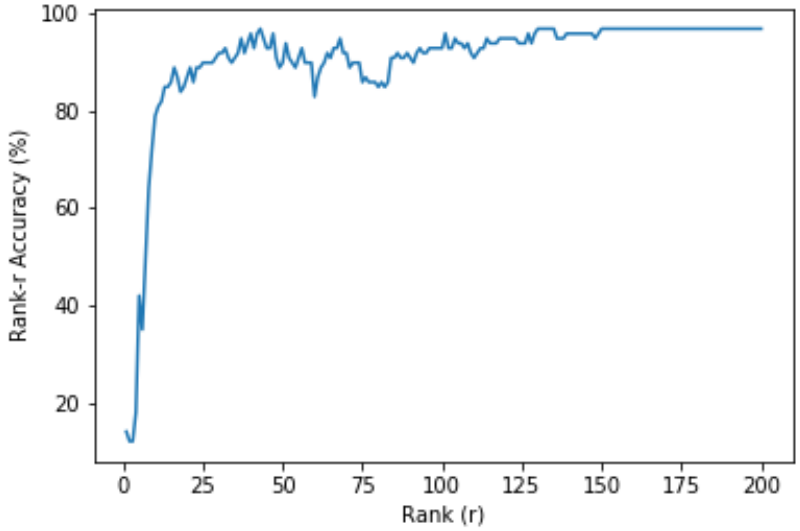

Figure 6: Rank-r Classification Accuracy plot.

1.2 What's Cooking?

(a) Join the Competition

We joined the Kaggle competition What's Cooking?

(b) The Data

We loaded in the json training and testing data files and determined some information about the training data by counting the number dishes, the number of unique cuisine, and the number of unique ingredients included in the sample set. We found that the training set had 39,774 samples (dishes). Of these dishes there were 20 different cuisines. Within these dishes there were 6,714 different ingredients.

(c) Dish Representation by Ingredients

For each sample (dish) we created a 6,714 value vector, where each item in the vector represented an ingredient. If the dish contained the ingredient, the ingredient's value in the vector was 1, and if the dish did not contain the ingredient, the ingredient's value in the vector was 0. We put all of these vectors into a feature matrix and this feature matrix had the shape (39774, 6714). We also created a feature matrix for the test data similarly. The test feature matrix had shape (9944, 6714).

(d) Naive Bayes Classification

We performed Naive Bayes on the data using two different priors, Gaussian and Bernoulli. To create these classifiers, fit them, and use them to make predictions we used the sklearn.naive_bayes methods GaussianNB and BernoulliNB [4]. We performed 3-fold crossvalidation on the training data with both models and found the mean accuracy. The accuracy for the Naive Bayes Model with Gaussian prior was 37.76%. The accuracy for the Naive Bayes Model with Bernoulli prior was 68.69%.

(e) Which Was Better?

The Bernoulli prior performed better. This is likely because the data is not distributed in the shape of a Gaussian. Rather, the data must have more of a multimodal distribution. This would make sense if there are multiple cuisines that are strongly represented in the data and this is possible because there are 20 cuisines in a dataset of almost 40,000 dishes.

(f) Logistic Regression

We used Logistic Regression from sklearn on the data [REFERENCE]. We used 3-fold crossvalidation on the training data using this model to determine an average accuracy. The accuracy was 77.87%.

(g) Use the Best Model on Test Data

Since the best performing model was Logistic Regression, we performed logistic regression on the whole test data set and submitted the results to the Kaggle competition. On the Kaggle competition we achieved accuracy of about 78.34%.

2 Written Exercises

2.1 Exercise 1 - Ex. 4.1 in [6].

Show how to solve the generalized eigenvalue problem $\max a^T\mathbf{B}a$ subject to $a^T\mathbf{W}a = 1$ by transforming to a standard eigenvalue problem.

We want to maximize $a^T\mathbf{B}a$ and we know that this maximization is subject to $a^T\mathbf{W}a - 1 = 0$. This makes it clear that we need to use the Lagrange Multiplier defined as $l(a) = f(a) - \lambda g(a)$ [7]. Thus, we have $f(a) = a^T\mathbf{B}a$ and $g(a) = a^T\mathbf{W}a - 1$.

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

To maximize we need to take the derivative and set it equal to 0:

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

Let $u_1 = \mathbf{B}a$, $u_2^T = a^T\mathbf{B}$, $v_1 = \mathbf{W}a$, $v_2^T = a^T\mathbf{W}$, where these are treated as constant while performing the derivative with product rule.

$\frac{dl}{da} = \frac{d}{da}a^Tu_1 + \frac{d}{da}u_2^Ta - \lambda(\frac{d}{da}a^Tv_1 + \frac{d}{da}v_2^Ta)$

$\frac{dl}{da} = u_1^T + u_2^T - \lambda(v_1^T + v_1^T) = a^T\mathbf{B}^T + a^T\mathbf{B} - \lambda(a^T\mathbf{W}^T + a^T\mathbf{W})$

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

Assume the matrices $\mathbf{W}$ and $\mathbf{B}$ are symmetric. If two matrices are symmetric then they are equal to their transpose. Therefore, adding a symmetric matrix to its transpose is just scaling. So we have:

$2\mathbf{B}^Ta^T - \lambda 2\mathbf{W}^Ta^T = 0$

To solve the eigenvalue problem lets find what λa is equal to:

$2\mathbf{B}^Ta^T = \lambda 2\mathbf{W}^Ta^T$

$\mathbf{B}a = \lambda\mathbf{W}a$

$\mathbf{B}\mathbf{W}^{-1}a = \lambda a$

This is the solution to a standard eigenvalue problem.

2.2 Exercise 2 - Ex. 4.2 in [6].

(a) The LDA rule will classify to class 2 if $\delta_2(x) > \delta_1(x)$. This gives:

$x^T\Sigma^{-1}\mu_2 - \frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2 + log\pi_2 > x^T\Sigma^{-1}\mu_1 - \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + log\pi_1$

$x^T\Sigma^{-1}\mu_2 - x^T\Sigma^{-1}\mu_1 - \frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2 + \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + log\pi_2 - log\pi_1 > 0$

$x^T\Sigma^{-1}\mu_2 - x^T\Sigma^{-1}\mu_1 - \frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2 + \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + log\frac{\pi_2}{\pi_1} > 0$

$x^T\Sigma^{-1}\mu_2 - x^T\Sigma^{-1}\mu_1 - \frac{1}{2}\Sigma^{-1}(\mu_2^T\mu_2 - \mu_1^T\mu_1) + log\frac{\pi_2}{\pi_1} > 0$ Eq. 1

Since $\hat{\pi}_k = \frac{N_k}{N}$, we get $\hat{\pi}_1 = \frac{N_1}{N}$ and $\hat{\pi}_2 = \frac{N_2}{N}$. Substituting in Eq. 1 we get:

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

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

We can simplify the original equation provided in the question to:

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

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

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

$x^T\hat{\Sigma}^{-1}\hat{\mu_2} - x^T\hat{\Sigma}^{-1}\hat{\mu_1} - \frac{1}{2}\hat{\Sigma}^{-1}(\hat{\mu_2}^T\hat{\mu_2} - \hat{\mu_1}^T\hat{\mu_1}) + log\frac{N_2}{N_1} > 0$ Eq. 3

Since the prediction in Eq. 3 is equivalent to the values in Eq. 2, the LDA rule will classify to class 2 if the equation in the question is true. Otherwise it will classify to class 1 i.e the LDA rule will classify to class 1 if $\delta_1(x) > \delta_2(x)$.

(b) First, we want to minimize the least squares criterion (LSC).

To minimize this we need to take the derivative with respect to β and β_0.

$\frac{d(LSC)}{d\beta} = -2\sum_{i=1}^{N} x_i(y_i - \beta_0 - \beta^T x_i) = 0$ Eq. 1

$\frac{d(LSC)}{d\beta_0} = -2\sum_{i=1}^{N} (y_i - \beta_0 - \beta^T x_i) = 0$ Eq. 2

From equation 2 we can derive the following:

$0 = \sum_{i=1}^{N} (y_i - \beta^T x_i) - N\beta_0$

$\beta_0 = \frac{1}{N}\sum_{i=1}^{N} (y_i - \beta^T x_i)$ Eq. 3

Then, from Eq. 1 and Eq. 3 we have:

$\sum_{i=1}^{N} x_i(y_i - (\frac{1}{N}\sum_{j=1}^{N} (y_j - \beta^T x_j)) - \beta^T x_i) = 0$

Let X be the sum of all x_j from j=1 to j=N.

$\frac{1}{N}\sum_{i=1}^{N} \beta^T x_i X - \sum_{i=1}^{N} \beta^T x_i x_i + \sum_{i=1}^{N} x_i(y_i - (\frac{1}{N}\sum_{j=1}^{N} y_j)) = 0$

$\sum_{i=1}^{N} \beta^T x_i x_i - \frac{1}{N}\sum_{i=1}^{N} \beta^T x_i X = \sum_{i=1}^{N} x_i(y_i - (\frac{1}{N}\sum_{j=1}^{N} y_j))$ Eq. 4

Let's solve the right hand side (RHS) of Eq.4 and make it equal the right hand side of the final equation (i.e. $N(\hat{\mu}_2 - \hat{\mu}_1)$).
We know we have the two classes with class sizes N_1 and N_2. Therefore we know the following.

$\frac{1}{N}\sum_{j=1}^{N} y_j = \frac{N_1 y_1 + N_2 y_2}{N}$ Eq. 5

Combining the RHS of Eq. 4 and Eq. 5.

$RHS = \sum_{i=1}^{N} x_i(y_i - \frac{N_1 y_1 + N_2 y_2}{N})$

We know that there are 2 classes that we can represent with k and we know from the book pg. 109 that $\hat{\mu}_k = \sum_{g_i=k} x_i/N_k$. So, $\sum_{g_i=k} x_i = N_k\hat{\mu}_k =$ Therefore, we can simplify using this and the idea of k=2 classes.

$RHS = \sum_{k=1}^{2} N_k\hat{\mu}_k(y_k - \frac{N_1 y_1 + N_2 y_2}{N})$

$= \sum_{k=1}^{2} N_k\hat{\mu}_k(\frac{Ny_k - N_1 y_1 - N_2 y_2}{N})$

$= N_1\hat{\mu}_1(\frac{Ny_1 - N_1 y_1 - N_2 y_2}{N}) + N_2\hat{\mu}_2(\frac{Ny_2 - N_1 y_1 - N_2 y_2}{N})$

$= \frac{N_1 N_2}{N}(\hat{\mu}_1(\frac{(N-N_1)y_1 - N_2 y_2}{N_2}) + \hat{\mu}_2(\frac{(N-N_2)y_2 - N_1 y_1}{N_1}))$

$= \frac{N_1 N_2}{N}(\hat{\mu}_1(\frac{(N-N_1)y_1}{N_2} - y_2) + \hat{\mu}_2(\frac{(N-N_2)y_2}{N_1} - y_1))$

$= \frac{N_1 N_2}{N}(\hat{\mu}_1(y_1 - y_2) + \hat{\mu}_2(y_2 - y_1))$

$= \frac{N_1 N_2}{N}(y_2 - y_1)(\hat{\mu}_2 - \hat{\mu}_1)$

We know that $y_1 = -N/N_1$ and $y_2 = N/N_2$.

$= \frac{N_1 N_2}{N}(\frac{N}{N_2} + \frac{N}{N_1})(\hat{\mu}_2 - \hat{\mu}_1)$

$RHS = N(\hat{\mu}_2 - \hat{\mu}_1)$ Eq. 6

Eq. 6 is exactly the RHS of the equation we are trying to prove that $\hat{\beta}$ satisfies.

Now we can solve the left hand side (LHS) of Eq. 4.

We know that β, x_i and X (the sum of all x_i's) are vectors, so the transpose can be switched between them.

$\sum_{i=1}^N \beta^T x_i x_i - \frac{1}{N}\sum_{i=1}^N \beta^T x_i X = \sum_{i=1}^N \beta x_i x_i^T - \frac{1}{N}\sum_{i=1}^N \beta x_i X^T$

$LHS = (\sum_{i=1}^N x_i x_i^T - \frac{X^T}{N}\sum_{i=1}^N x_i)\beta$ Eq. 7

We know from before that we have a 2 class class and $\sum_{g_i=k} x_i = N_k \hat{\mu}_k$. Thus we can simplify Eq. 7.

$LHS = (\sum_{i=1}^N x_i x_i^T - \frac{X^T}{N}\sum_{k=1}^2 \sum_{g_i=k} x_i)\beta$

$= (\sum_{i=1}^N x_i x_i^T - \frac{X^T}{N}(N_1\hat{\mu}_1 + N_2\hat{\mu}_2))\beta$

Recall that X was just the sum of all x_j's from j = 1 to N.

Therefore, $X^T = (\sum_{j=1}^N x_j)^T = (\sum_{k=1}^2 \sum_{g_i=k} x_j)^T = (\sum_{k=1}^2 N_k\hat{\mu}_k)^T$.

$LHS = (\sum_{i=1}^N x_i x_i^T - \frac{1}{N}(N_1\hat{\mu}_1 + N_2\hat{\mu}_2)(N_1\hat{\mu}_1 + N_2\hat{\mu}_2)^T)\beta$

$= (\sum_{i=1}^N x_i x_i^T - \frac{1}{N}(N_1^2\hat{\mu}_1\hat{\mu}_1^T + N_1N_2\hat{\mu}_1\hat{\mu}_2^T + N_1N_2\hat{\mu}_2\hat{\mu}_1^T N_2^2\hat{\mu}_2\hat{\mu}_2^T)\beta$

$LHS = (\sum_{i=1}^N x_i x_i^T - \frac{N_1N_2}{N}(\frac{N_1}{N_2}\hat{\mu}_1\hat{\mu}_1^T + \hat{\mu}_1\hat{\mu}_2^T + \hat{\mu}_2\hat{\mu}_1^T + \frac{N_2}{N_1}\hat{\mu}_2\hat{\mu}_2^T)\beta$ Eq. 8

We need Eq. 8 to equal the left hand side of the resulting equation which is $[(N-2)\hat{\Sigma} + N\hat{\Sigma}_B]\beta$. We need to simplify this expression.

Let's start with the first term in the parenthesis $(N-2)\hat{\Sigma}$.

Let's use the definition of $\hat{\Sigma} = \sum_{k=1}^K \sum_{g_i=k}(x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T/(N-K)$, where K=2.

$(N-2)\hat{\Sigma} = \sum_{k=1}^2 \sum_{g_i=k}(x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T$

$= \sum_{k=1}^2 \sum_{g_i=k} x_i x_i^T - x_i\hat{\mu}_k^T - x_i^T\hat{\mu}_k + \hat{\mu}_k\hat{\mu}_k^T$

$= \sum_{k=1}^2 \sum_{g_i=k} x_i x_i^T - 2x_i\hat{\mu}_k^T + \hat{\mu}_k\hat{\mu}_k^T$

$= \sum_{i=1}^N x_i x_i^T - 2\sum_{k=1}^2 \hat{\mu}_k^T \sum_{g_i=k} x_i + \sum_{k=1}^2 N_k\hat{\mu}_k\hat{\mu}_k^T$ We know $\sum_{g_i=k} x_i = N_k\hat{\mu}_k$

$= \sum_{i=1}^N x_i x_i^T - 2\sum_{k=1}^2 N_k\hat{\mu}_k\hat{\mu}_k^T + \sum_{k=1}^2 N_k\hat{\mu}_k\hat{\mu}_k^T$

$= \sum_{i=1}^N x_i x_i^T - \sum_{k=1}^2 N_k\hat{\mu}_k\hat{\mu}_k^T$

$= \sum_{i=1}^N x_i x_i^T - (N_1\hat{\mu}_1\hat{\mu}_1^T + N_2\hat{\mu}_2\hat{\mu}_2^T)$

$(N-2)\hat{\Sigma} = \sum_{i=1}^{N} x_i x_i^T - \frac{N_1 N_2}{N}(\frac{N}{N_2}\hat{\mu}_1\hat{\mu}_1^T + \frac{N}{N_1}\hat{\mu}_2\hat{\mu}_2^T)$ Eq. 9

Let's simplify the second term in the parenthesis $N\hat{\Sigma}_B$.

Let's use the definition of $\hat{\Sigma}_B = \frac{N_1 N_2}{N^2}(\hat{\mu}_2 - \hat{\mu}_1)(\hat{\mu}_2 - \hat{\mu}_1)^T$.

$N\hat{\Sigma}_B = \frac{N_1 N_2}{N}(\hat{\mu}_2 - \hat{\mu}_1)(\hat{\mu}_2 - \hat{\mu}_1)^T$

$N\hat{\Sigma}_B = \frac{N_1 N_2}{N}(\hat{\mu}_2\hat{\mu}_2^T - \hat{\mu}_1\hat{\mu}_2^T - \hat{\mu}_2\hat{\mu}_1^T + \hat{\mu}_1\hat{\mu}_1^T)$ Eq. 10

Let's combine Eq. 9 and 10 to form the simplified version of the left hand side of the resulting equation.

$[(N-2)\hat{\Sigma} + N\hat{\Sigma}_B]\beta = [\sum_{i=1}^{N} x_i x_i^T - \frac{N_1 N_2}{N}(\frac{N}{N_2}\hat{\mu}_1\hat{\mu}_1^T + \frac{N}{N_1}\hat{\mu}_2\hat{\mu}_2^T) + \frac{N_1 N_2}{N}(\hat{\mu}_2\hat{\mu}_2^T - \hat{\mu}_1\hat{\mu}_2^T - \hat{\mu}_2\hat{\mu}_1^T + \hat{\mu}_1\hat{\mu}_1^T)]\beta$

$= [\sum_{i=1}^{N} x_i x_i^T - \frac{N_1 N_2}{N}(\frac{N_1+N_2}{N_2}\hat{\mu}_1\hat{\mu}_1^T + \frac{N_1+N_2}{N_1}\hat{\mu}_2\hat{\mu}_2^T - \hat{\mu}_2\hat{\mu}_2^T + \hat{\mu}_1\hat{\mu}_2^T + \hat{\mu}_2\hat{\mu}_1^T - \hat{\mu}_1\hat{\mu}_1^T)]\beta$

$[(N-2)\hat{\Sigma} + N\hat{\Sigma}_B]\beta = [\sum_{i=1}^{N} x_i x_i^T - \frac{N_1 N_2}{N}(\frac{N_1}{N_2}\hat{\mu}_1\hat{\mu}_1^T + \frac{N_2}{N_1}\hat{\mu}_2\hat{\mu}_2^T + \hat{\mu}_1\hat{\mu}_2^T + \hat{\mu}_2\hat{\mu}_1^T]\beta$ Eq. 11

Combining Eq. 8 and Eq. 11.

$LHS = [(N-2)\hat{\Sigma} + N\hat{\Sigma}_B]\beta$ Eq. 12

Eq. 12 is exactly the LHS of the equation we are trying to prove that $\hat{\beta}$ satisfies.

Thus, we have proven that $\hat{\beta}$ satisfies the equation $[(N-2)\hat{\Sigma} + N\hat{\Sigma}_B]\beta = N(\hat{\mu}_2 - \hat{\mu}_1)$

(c) We know that $\hat{\Sigma}_B\beta = \frac{N_1 N_2}{N^2}(\hat{\mu}_2 - \hat{\mu}_1)(\hat{\mu}_2 - \hat{\mu}_1)^T\beta$.

We can summarize several of the terms in the above equation into a constant α which is a real number.

$\hat{\Sigma}_B\beta = \alpha(\hat{\mu}_2 - \hat{\mu}_1)$

Therefore, we know that $\hat{\Sigma}_B\beta$ is in the direction of $(\hat{\mu}_2 - \hat{\mu}_1)$. Since α is a scalar.

We know that the solution $\hat{\beta}$ satisfies the equation in (b).

$\hat{\beta} = \frac{N(\hat{\mu}_2 - \hat{\mu}_1)}{(N-2)\hat{\Sigma} + N\hat{\Sigma}_B} = \frac{N(\hat{\mu}_2 - \hat{\mu}_1)\hat{\Sigma}^{-1}}{[(N-2)\hat{\Sigma} + N\hat{\Sigma}_B]\hat{\Sigma}^{-1}} = \hat{\Sigma}^{-1}\frac{N}{(N-2) + \frac{N_1 N_2}{N}}(\hat{\mu}_2 - \hat{\mu}_1)$

So ...

$\hat{\beta} \propto \hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1)$

(d) All calculations are done with N_1 and N_2, which are the class sizes of the class with label 1 and the class with label 2, respectively. Therefore, as long as the coding is distinct and allows us to differentiate between classes, the equations will hold because all that matters is the size of the full dataset and the size of each class.

(e) Find the solution β_0.

$\hat{\beta}_0 = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{\beta}^T x_i)$

$\hat{\beta}_0 = \frac{1}{N}(N_1 y_1 + N_2 y_2 - \hat{\beta}^T X)$ Eq. 1

We also know that to classify to class 2 it must be true that:

$\hat{f}(x) > 0$

$\hat{\beta}_0 + x^T\hat{\beta} > 0$ Eq. 2

Combining Eq. 1 and Eq. 2 we get:

$\frac{1}{N}(N_1 y_1 + N_2 y_2 - \hat{\beta}^T X) + x^T\hat{\beta} > 0$ Eq. 3

We know that $y_1 = -N/N_1$ and $y_2 = N/N_2$, so we get:

$\frac{1}{N}(N_1(\frac{-N}{N_1}) + N_2(\frac{N}{N_2}) - \hat{\beta}^T X) + x^T\hat{\beta} > 0$

$\frac{1}{N}(-N + N - \hat{\beta}^T X) + x^T\hat{\beta} > 0$

$-\frac{1}{N}(\hat{\beta}^T X) + x^T\hat{\beta} > 0$ Eq. 4

We also know that $X = \sum_{i=1}^{N} x_i$ and thus substituting in Eq. 4 we get:

$-\frac{1}{N}(\hat{\beta}^T \sum_{i=1}^{N} x_i) + x^T\hat{\beta} > 0$

Since $\sum_{g_i=k} x_i = N_k\hat{\mu}_k$, we obtain:

$-\frac{1}{N}(\hat{\beta}^T[N_1\hat{\mu_1} + N_2\hat{\mu_2}]) + x^T\hat{\beta} > 0$

We also know that $\hat{\beta}$, $\hat{\mu_1}$ and $\hat{\mu_2}$ are 1D vectors, so the transpose can be interchanged, giving:

$-\frac{1}{N}(\hat{\beta}[N_1\hat{\mu_1}^T + N_2\hat{\mu_2}^T]) + x^T\hat{\beta} > 0$ Eq. 5

From (c) we know that $\hat{\beta} = \hat{\Sigma}^{-1}\frac{N}{(N-2)+\frac{N_1N_2}{N}}(\hat{\mu}_2 - \hat{\mu}_1)$, which substituted in Eq. 5 gives:

$-\frac{1}{N}([\hat{\Sigma}^{-1}\frac{N}{(N-2)+\frac{N_1N_2}{N}}(\hat{\mu}_2 - \hat{\mu}_1)][N_1\hat{\mu_1}^T + N_2\hat{\mu_2}^T]) + x^T[\hat{\Sigma}^{-1}\frac{N}{(N-2)+\frac{N_1N_2}{N}}(\hat{\mu}_2 - \hat{\mu}_1)] > 0$ Eq. 5

$x^T[\hat{\Sigma}^{-1}\frac{N}{(N-2)+\frac{N_1N_2}{N}}(\hat{\mu}_2 - \hat{\mu}_1)] > \frac{1}{N}([\hat{\Sigma}^{-1}\frac{N}{(N-2)+\frac{N_1N_2}{N}}(\hat{\mu}_2 - \hat{\mu}_1)][N_1\hat{\mu_1}^T + N_2\hat{\mu_2}^T])$

Simplifying the expression, we get:

$x^T\hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1) > \frac{1}{N}\hat{\Sigma}^{-1}[N_1\hat{\mu_1}^T + N_2\hat{\mu_2}^T](\hat{\mu}_2 - \hat{\mu}_1)$ Eq. 6

The LDA expression is:

$x^T\hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1) - \frac{1}{2}\hat{\Sigma}^{-1}(\hat{\mu_2}^T\hat{\mu_2} - \hat{\mu_1}^T\hat{\mu_1}) + log\frac{N_2}{N_1} > 0$

The Eq. 6 is different to the LDA expression in all the cases except for $N_1 = N_2$, in which case we get $N_1 = \frac{N}{2}$ and $N_2 = \frac{N}{2}$. This gives $log\frac{N_2}{N_1} = log1 = 0$, making the LDA expression:

$x^T\hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1) - \frac{1}{2}\hat{\Sigma}^{-1}(\hat{\mu_2}^T\hat{\mu_2} - \hat{\mu_1}^T\hat{\mu_1}) > 0$

For $N_1 = \frac{N}{2}$ and $N_2 = \frac{N}{2}$ in Eq. 6 we obtain:

$x^T\hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1) > \frac{1}{N}\hat{\Sigma}^{-1}[\frac{N}{2}\hat{\mu_1}^T + \frac{N}{2}\hat{\mu_2}^T](\hat{\mu}_2 - \hat{\mu}_1)$

$x^T\hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1) > \frac{1}{2}\hat{\Sigma}^{-1}[\hat{\mu_1}^T + \hat{\mu_2}^T](\hat{\mu}_2 - \hat{\mu}_1)$

$x^T\hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1) - \frac{1}{2}\hat{\Sigma}^{-1}[\hat{\mu_1}^T + \hat{\mu_2}^T](\hat{\mu}_2 - \hat{\mu}_1) > 0$ Eq. 7

Thus, Eq.7 is equal to the LDA expression.

2.3 Exercise 3

(a) Find M^TM and MM^T.

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

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

(b) Find eigenvalues.

To solve for the eigenvalues we need to find the characteristic equation for the matrix. Let A be the matrix we are trying to find the eigenvalues of.

$\mathbf{A}v = \lambda v \rightarrow \mathbf{A}v - \lambda v = 0 \rightarrow (\mathbf{A} - \lambda\mathbf{I})v = 0$

The characteristic equation is: $|\mathbf{A} - \lambda\mathbf{I}| = 0$

Let's find the eigenvalues for $M^T M$:

$$\left|\begin{bmatrix}39 & 57 & 60\\ 57 & 118 & 53\\ 60 & 53 & 127\end{bmatrix} - \begin{bmatrix}\lambda & 0 & 0\\ 0 & \lambda & 0\\ 0 & 0 & \lambda\end{bmatrix}\right| = \left|\begin{bmatrix}39-\lambda & 57 & 60\\ 57 & 118-\lambda & 53\\ 60 & 53 & 127-\lambda\end{bmatrix}\right| = 0$$

$-\lambda^3 + 284\lambda^2 - 14883\lambda = 0$

$\lambda_1 = 0$

$\lambda_2 = 142 - \sqrt{5281}$

$\lambda_3 = 142 + \sqrt{5281}$

Let's find the eigenvalues for MM^T:

$$\left|\begin{bmatrix}10 & 9 & 26 & 3 & 26\\ 9 & 62 & 8 & -5 & 85\\ 26 & 8 & 72 & 10 & 50\\ 3 & -5 & 10 & 2 & -1\\ 26 & 85 & 50 & -1 & 138\end{bmatrix} - \begin{bmatrix}\lambda & 0 & 0 & 0 & 0\\ 0 & \lambda & 0 & 0 & 0\\ 0 & 0 & \lambda & 0 & 0\\ 0 & 0 & 0 & \lambda & 0\\ 0 & 0 & 0 & 0 & \lambda\end{bmatrix}\right| = \left|\begin{bmatrix}10-\lambda & 9 & 26 & 3 & 26\\ 9 & 62-\lambda & 8 & -5 & 85\\ 26 & 8 & 72-\lambda & 10 & 50\\ 3 & -5 & 10 & 2-\lambda & -1\\ 26 & 85 & 50 & -1 & 138-\lambda\end{bmatrix}\right| = 0$$

$-\lambda^5 + 284\lambda^4 - 14883\lambda^3 = 0$

$\lambda_1 = 0$

$\lambda_2 = 142 - \sqrt{5281}$

$\lambda_3 = 142 + \sqrt{5281}$

(c) Find eigenvectors.

Let's find the eigenvectors for $M^T M$.

Convert the matrices of $(\mathbf{A} - \lambda\mathbf{I})$ to row-echelon form.

For $\lambda_1 = 0$:

$$\begin{bmatrix}39 & 57 & 60 & 0\\ 57 & 118 & 53 & 0\\ 60 & 53 & 127 & 0\end{bmatrix} \rightarrow \begin{bmatrix}1 & 0 & 3 & 0\\ 0 & 1 & -1 & 0\\ 0 & 0 & 0 & 0\end{bmatrix}$$

$x_1 + 3x_3 = 0$

$x_2 - x_3 = 0$

For $\lambda_2 = (142 - \sqrt{5281}) = 69.3295$:

$$\begin{bmatrix}39 - (142 - \sqrt{5281}) & 57 & 60 & 0\\ 57 & 118 - (142 - \sqrt{5281}) & 53 & 0\\ 60 & 53 & 127 - (142 - \sqrt{5281}) & 0\end{bmatrix}$$

$$= \begin{bmatrix}-30.33 & 57 & 60 & 0\\ 57 & 48.67 & 53 & 0\\ 60 & 53 & 57.67 & 0\end{bmatrix} \rightarrow \begin{bmatrix}1 & 0 & 0.0213 & 0\\ 0 & 1 & 1.064 & 0\\ 0 & 0 & 0 & 0\end{bmatrix}$$

$x_1 + 0.0213x_3 = 0$

$x_2 + 1.064x_3 = 0$

For $\lambda_3 = (142 + \sqrt{5281}) = 214.67$:

$$\begin{bmatrix} 39 - (142 + \sqrt{5281}) & 57 & 60 & 0 \\ 57 & 118 - (142 + \sqrt{5281}) & 53 & 0 \\ 60 & 53 & 127 - (142 + \sqrt{5281}) & 0 \end{bmatrix}$$

$$= \begin{bmatrix} -175.67 & 57 & 60 & 0 \\ 57 & -96.67 & 53 & 0 \\ 60 & 53 & -87.67 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & -0.642 & 0 \\ 0 & 1 & -0.927 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$x_1 - 0.642x_3 = 0$

$x_2 - 0.927x_3 = 0$

The normalized eigenvectors from the above results are:

$$v_1 = \begin{pmatrix} -0.904 \\ -0.301 \\ 0.301 \end{pmatrix}$$

$$v_2 = \begin{pmatrix} -0.0145 \\ -0.728 \\ 0.684 \end{pmatrix}$$

$$v_3 = \begin{pmatrix} 0.426 \\ 0.615 \\ 0.663 \end{pmatrix}$$

Let's find the eigenvectors for MM^T:

Convert the matrices of $(\mathbf{A} - \lambda\mathbf{I})$ to row-echelon form.

For $\lambda_1 = 0$:

$$\begin{bmatrix} 10 & 9 & 26 & 3 & 26 & 0 \\ 9 & 62 & 8 & -5 & 85 & 0 \\ 26 & 8 & 72 & 10 & 50 & 0 \\ 3 & -5 & 10 & 2 & -1 & 0 \\ 26 & 85 & 50 & -1 & 138 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & \frac{20}{7} & \frac{3}{7} & \frac{11}{7} & 0 \\ 0 & 1 & \frac{-2}{7} & \frac{-1}{7} & \frac{8}{7} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$x_1 + \frac{20}{7}x_3 + \frac{3}{7}x_4 + \frac{11}{7}x_5 = 0$

$x_2 + \frac{-2}{7}x_3 + \frac{-1}{7}x_4 + \frac{8}{7}x_5 = 0$

For $\lambda_2 = (142 - \sqrt{5281}) = 69.3295$:

$$\begin{bmatrix} 10 - 69.3295 & 9 & 26 & 3 & 26 & 0 \\ 9 & 62 - 69.3295 & 8 & -5 & 85 & 0 \\ 26 & 8 & 72 - 69.3295 & 10 & 50 & 0 \\ 3 & -5 & 10 & 2 - 69.3295 & -1 & 0 \\ 26 & 85 & 50 & -1 & 138 - 69.3295 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} -59.3295 & 9 & 26 & 3 & 26 & 0 \\ 9 & -7.3295 & 8 & -5 & 85 & 0 \\ 26 & 8 & 2.6705 & 10 & 50 & 0 \\ 3 & -5 & 10 & -67.3295 & -1 & 0 \\ 26 & 85 & 50 & -1 & 68.6705 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 & 1.84 & 0 \\ 0 & 1 & 0 & 0 & -3.41 & 0 \\ 0 & 0 & 1 & 0 & 6.23 & 0 \\ 0 & 0 & 0 & 1 & 1.28 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$x_1 + 1.84x_5 = 0$

$x_2 - 3.41x_5 = 0$

$x_3 + 6.23x_5 = 0$

$x_4 + 1.28x_5 = 0$

$x_5 = 0$

For $\lambda_3 = (142 + \sqrt{5281}) = 214.67$:

$$\begin{bmatrix} 10-214.67 & 9 & 26 & 3 & 26 & 0 \\ 9 & 62-214.67 & 8 & -5 & 85 & 0 \\ 26 & 8 & 72-214.67 & 10 & 50 & 0 \\ 3 & -5 & 10 & 2-214.67 & -1 & 0 \\ 26 & 85 & 50 & -1 & 138-214.67 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} -204.67 & 9 & 26 & 3 & 26 & 0 \\ 9 & -152.67 & 8 & 5 & 85 & 0 \\ 26 & 8 & -142.67 & 10 & 50 & 0 \\ 3 & -5 & 10 & -212.67 & -1 & 0 \\ 26 & 85 & 50 & -1 & -76.67 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 & -0.207 & 0 \\ 0 & 1 & 0 & 0 & -0.591 & 0 \\ 0 & 0 & 1 & 0 & -0.422 & 0 \\ 0 & 0 & 0 & 1 & -0.004 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$x_1 - 0.207x_5 = 0$

$x_2 - 0.591x_5 = 0$

$x_3 - 0.422x_5 = 0$

$x_4 - 0.004x_5 = 0$

$x_5 = 0$

The normalized eigenvectors from the above results are:

$$v_1 = \begin{pmatrix} -0.719 \\ -0.523 \\ 0 \\ 0 \\ 0.458 \end{pmatrix}$$

$$v_2 = \begin{pmatrix} -0.245 \\ 0.453 \\ -0.829 \\ -0.170 \\ 0.133 \end{pmatrix}$$

$$v_3 = \begin{pmatrix} 0.165 \\ 0.471 \\ 0.336 \\ 0.003 \\ 0.798 \end{pmatrix}$$

(d) Perform SVD

Given a matrix $\mathbf{A} \in \mathbb{R}^{N \times p}$ its SVD is given by $\mathbf{A} = \mathbf{UDV}^\top$. The columns of $\mathbf{U}$ consist of the eigenvectors of AA^T for non-zero eigenvalues and the columns of $\mathbf{V}$ consist of the eigenvectors of A^TA for non-zero eigenvalues. The diagonal matrix $\mathbf{D}$ contains the square root of the eigenvalues from AA^T or A^TA. Following this, the SVD of M is:

$$M = \begin{pmatrix} -0.245 & 0.165 \\ 0.453 & 0.471 \\ -0.829 & 0.336 \\ -0.170 & 0.003 \\ 0.133 & 0.798 \end{pmatrix} \begin{pmatrix} \sqrt{214.67} & 0 \\ 0 & \sqrt{69.3295} \end{pmatrix} \begin{pmatrix} -0.0145 & -0.728 & 0.684 \\ 0.426 & 0.615 & 0.663 \end{pmatrix}$$

(e) Obtain one-dimensional approximation of M

If we set the smallest singular value to be 0 instead of $\sqrt{69.3295}$ we obtain an one-dimensional approximation of M as follows:

$$M = \begin{pmatrix} -0.245 & 0.165 \\ 0.453 & 0.471 \\ -0.829 & 0.336 \\ -0.170 & 0.003 \\ 0.133 & 0.798 \end{pmatrix} \begin{pmatrix} \sqrt{214.67} & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} -0.0145 & -0.728 & 0.684 \\ 0.426 & 0.615 & 0.663 \end{pmatrix} = \begin{pmatrix} 0.052 & 2.634 & -2.479 \\ -0.096 & -4.874 & 4.588 \\ 0.177 & 8.916 & -8.393 \\ 0.036 & -1.823 & -1.716 \\ 0 & 0 & 0 \end{pmatrix}$$

References

[1] T. E. Oliphant, *A guide to NumPy*. Trelgol Publishing USA, 2006, vol. 1.

[2] E. Jones, T. Oliphant, P. Peterson *et al.* (2001) SciPy: Open source scientific tools for Python. "http://www.scipy.org/"[Accessed 2019-06-10].

[3] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science Engineering*, 2007.

[4] A. G. e. a. Fabian Pedregosa, Gaël Varoquaux, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, 2011.

[5] pypi.org. (2019) imageio. "https://pypi.org/project/imageio/"[Accessed 2019-06-10].

[6] J. F. Trevor Hastie, Robert Tibshirani, *The Elements of Statistical Learning*, 2nd ed. Springer, 2017.

[7] Lagrange multiplier. "https://en.wikipedia.org/wiki/Lagrange_multiplier".

A Code to Generate Graphs

A.1 Code for Low-Rank Approximation (1f)

```
# r is r=1,2,...,200
rValues = [n for n in range(1, 201)]
# list to store Frobenius Norm error values
errList = []

# calculate the low rank approximation error for every rank in r=[1,200]
for r in rValues:
    # calculate matrix products
    SVt_product = np.dot(np.diag(S[:r]), VTrans[:r,:])
    Xr = np.dot(U[:,:r], SVt_product)
    # calculate difference between X and Xr
    diff = np.subtract(trainMinusAv, Xr) # change to train_data????
    # calculate Frobenius Norm of the difference
    errList.append(np.linalg.norm(diff, 'fro'))

# plot r against error
plt.plot(rValues, errList, label='Rank-r Approximation Error')
plt.xlabel("Rank (r)")
plt.ylabel("Approximation Error");

# save images for report
plt.savefig('Low_Rank.png')
```

A.2 Code for Logistic Regression Classification Accuracy (1h)

```
# Get training and test features for r = 10
train10, test10 = generateFMatrix(trainMinusAv, testMinusAv, 10)

# create a logistic regression model
lr = LogisticRegression(multi_class = 'ovr').fit(train10, train_labels)
# get predictions
preds = lr.predict(test10)

# calculate and print model accuracy
accuracy = metrics.accuracy_score(preds, test_labels) * 100
print('The accuracy of our model for r = 10 is', accuracy, '%')

accuracies = [] # list to store accuracies for r = 1,....,200

for r in rValues:
    # get matrices
    trainF, testF = generateFMatrix(trainMinusAv, testMinusAv, r)
    # create logistics regression model
    lr = LogisticRegression(multi_class = 'ovr').fit(trainF, train_labels)
    # predict
    preds = lr.predict(testF)
    # find accuracy
    accuracies.append(metrics.accuracy_score(test_labels, preds)*100)

# create accuracy plot, plotting r against accuracy
plt.plot(rValues, accuracies, label='Classification Accuracy')
plt.xlabel("Rank (r)")
plt.ylabel("Rank-r Accuracy (%)");

# save images for report
plt.savefig('Accuracy.png')
```