Principal Component and Linear Discriminant Analyses

Michael Ferko
Dept. of Electrical and Computer Engineering
University of Louisville
Louisville, USA
Email: Mike.W.Ferko@gmail.com

***Abstract*—A two-dimensional data set is a matrix composed of rows and columns where each row may represent a sample, and each column may represent a feature from the data set. To deal with the "curse of dimensionality" features may be reduced using an unsupervised approach called Principal Component Analysis (PCA) or Linear Discriminant Analysis (LDA). Feature reduction is a domain mapping from higher dimensionality to lower dimensionality in which Bayesian Classification of the data will yield similar results on similar train-test splitting of the data set.**

Keywords—PCA, LDA, Dimensionality Reduction, Data, linear algebra, Eigen Decomposition, Pattern Recognition, Machine Intelligence, Probability & Statistics, Engineering, curse of dimensionality, Digital Image Processing, Bayes Classification

I. Introduction

The goal of this project is to introduce the reader to a special area of linear algebra called matrix decomposition that is very useful for data by comparing the Principal Component (PCA) and Linear Discriminant Analyses (LDA). And after reduction in dimensionality of the data; Bayesian Classification is still possible.

There are several types of matrix decomposition based on the eigen value of a matrix for solving a system of equations that should be mentioned. Eigen decomposition (vectors and values), Cholesky decomposition, Singular Value Decomposition (SVD), Scale-invariant decomposition (SVD, but invariant with respect to diagonal scaling) and many more. From the decomposition we are creating a factorization of a matrix into a product of matrices. For example, if our original data is 100 samples (100 rows) of 10 different features (columns i.e., elementary attributes, or salient features of the original data set) then we have a matrix that is 100 rows x 10 columns.

$Data\ Y\ for\ m-sample\ rows, n-feature\ columns$:

$$Y = \begin{bmatrix} Y_{11} & \cdots & Y_{m1} \\ \vdots & \ddots & \vdots \\ Y_{1n} & \cdots & Y_{mn} \end{bmatrix}$$

Next, we need to ask extremely important and fundamental questions.

1. Which parameters do we want to keep?

Answer: Keep the parameters that are independent of one another (i.e., uncorrelated, or low covariance).

2. How do we determine which parameters best express a data set?

Answer: The parameters that best represent our data will have a high Signal-to-Noise Ration/Ratio (SNR or ratio of variances). Traditionally, the SNR may be approximated as the ratio of the average signal value (μ_{signal}, mean of the signal) to the standard deviation of the signal (σ_{signal}).

$$SNR = \frac{\sigma_{signal}^2}{\sigma_{noise}^2} \approx \frac{\mu_{signal}}{\sigma_{signal}}$$

A high SNR (>> 1; much, much greater than one) indicates high precision of data, whereas a low SNR indicates noise contaminated data.

Along the diagonal elements of a covariance matrix, we find the variance of each feature and on the off diagonals we see the correlation between all possible pairs of the features. So, for our n=10 features data we have a 10-by-10 covariance matrix which may be calculated using (series summation indexing may vary between programming languages where the length of the series 0 to n-1 will be of length n and could just as well be indexed from 1 to n so long as the length of the indexing is intact):

$$S_Y = Cov(Y, Y^T) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} Cov\left(Y_i, Y_j^T\right) = \frac{1}{n-1} YY^T$$

When calculated, the covariance matrix S_Y (S for Sigma the standard deviation) may be visualized as the n-by-n length matrix as follows:

$$S_Y = \begin{bmatrix} S_{11} & \cdots & S_{n1} \\ \vdots & \ddots & \vdots \\ S_{1n} & \cdots & S_{nn} \end{bmatrix}$$

The S_{1n} and S_{n1} off diagonals indicate the correlation; where high valued off-diagonal elements indicate dependence between features whereas low valued off diagonal elements indicates independence, just like the variance. Correlation may also be calculated from covariance as follows:

$$\rho(Y, Y^T) = \frac{S_Y}{\sqrt{\sigma_Y^2 \sigma_{Y^T}^2}}$$

A salient feature (or elementary attribute, or as in statistics a factor) may be redundant if two factors are highly correlated and by removing one of the two will result in no loss of information. The correlation coefficient is also known as Pearson's product moment correlation coefficient (PPMCC) is also known as the bivariate correlation.

Our goal is to find a covariance matrix where we:

1. Minimize Redundancy: *Covariance off-diagonals* minimized (we would like each feature to co-vary as little as possible)
2. Maximize Signal Variance: Covariance diagonals

This means the optimal covariance matrix will be a diagonal matrix.

Decomposing the square n by n covariance matrix of our original data will allow us to create an eigen vector matrix where when we perform a Matrix Chain Multiplication (MCM is an optimizable algorithm using the iterative Dynamic Programming approach for a time complexity of $T(n) = O(n^3)$ where n is the length of a dimensions of decomposition array) of the decompositions and the square matrix, we will obtain the original data with the original dimensions once again.

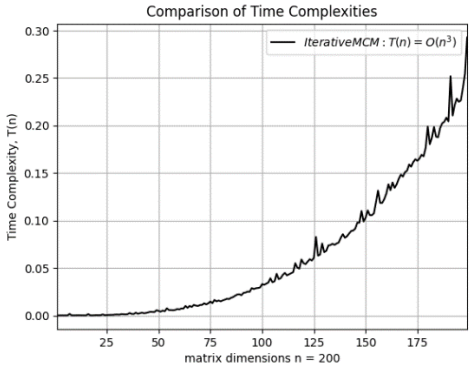

Figure 1: Iterative MCM: $T(n) = O(n^3)$

II. PART 1: PRINCIPAL COMPONENT ANALYSIS

The Principal Component Analysis (PCA) or Karhunen-Loeve Transform (KLT) is an unsupervised learning method of finding the "right" features from a data set. A geometric view of multidimensional data is to imagine clouds of points in m-dimensional sample space. That is to say that from the universe of possible measurable sets (σ_F, the sigma algebra); we are mapping a borel set from the sample space **Ω** within the universe of measurable events. when we take a measurement, we are mapping that measurable event to a data set. The sample space is a m-dimensional cloud of points, in which it is our job to analyze the sample space and determine key features (factors) which divide the data set by columns and then if those features are dependent upon one another (insignificant) we need to remove that feature from our data set via a dimensionality reduction such as PCA.

We need to find some orthonormal weight matrix P where the covariance matrix $S_Y = \frac{1}{n-1} YY^T$ is diagonalized. The rows of P are the principal components of Y which maps the data Y to the maximum variance direction (i.e., maximum eigen vector with mapping $Y = PX$). Now we will show the Process of Diagonalizing the covariance matrix:

$$S_Y = \frac{1}{n-1} YY^T = \frac{1}{n-1} (PX)(PX)^T = \frac{1}{n-1} P(XX^T)P^T$$
$$= \frac{1}{n-1} PAP^T; A = XX^T$$

We select the matrix P to be a matrix where each row is an eigenvector of $A = XX^T$. *Note: In linear algebra it may be shown that the inverse of an orthonormal matrix is its transpose (i.e., $P^{-1} = P^T$). A is a symmetric matrix which may be rewritten as $A = EDE^T$. Where E is a matrix of eigen vectors and D is some diagonal matrix which we want to isolate. By allowing the principal component vector matrix to be equal to the transposed Eigen vector matrix ($P = E^T$) we will achieve isolation of the diagonal matrix D. So, A can be rewritten as $A = EDE^T = P^T DP$.

$$S_Y = \frac{1}{n-1} PAP^T = \frac{1}{n-1} P(P^T DP)P^T$$
$$= \frac{1}{n-1} PP^{-1} DPP^{-1}$$
$$S_Y = \frac{1}{n-1} D$$

Through this manipulation we achieve our goal of minimizing redundancy and maximizing signal variance by diagonalizing the covariance matrix and selecting the diagonalization dimensions which comprise the largest ratio of the variance spread of the data. D can be n by n dimensional for the n eigen values or it can be 3-by-3 where only 3 eigen values are used.

We will see how the PCA implementation functions now that we understand the goals of dimensionality reduction.

Step 1: The first Step is to standardize the data. By standardize we are subtracting the mean and dividing by the standard deviation to achieve a zero mean with a standard deviation of 1 or converting the data to a standard normal distribution N(0,1). We want to understand this step and what these parameters are. However, since we are analyzing the variance/standard deviation we want to NOT divide by the standard deviation:

$$Z = \frac{x - \mu}{\sigma}$$

A mean vector μ is calculated by:

$$\mu = E[x] = \frac{1}{m} \sum_{k=0}^{m-1} x_k$$

For m number of samples where the E[x] is the expectation of x or the first moment in the moment generation function. And the standard deviation vector may be calculated as:

$$\sigma = \sqrt{E[x^2] - E[x]^2} = \sqrt{\frac{1}{m}\sum_{k=0}^{m-1}(x_k - \mu)^2}$$

Where $E[x-\mu]^2 = E[x^2] - E[x]^2$ is the variance or the 2nd moment in the moment generation function.

The standardized randomly generated data Y of dimensions 100-by-10 has a standard normal probability density function (pdf) of:

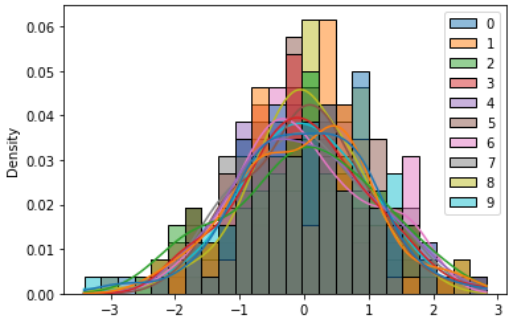

Figure 2: pdf of Standardized Y (100-by-10 Data)

The definition of eigen values and eigen vectors are as follows for a square matrix:

Define Eigen Vectors and Eigen Values

Let S be a square matrix. A non-zero vector **v** is an eigenvector for S with eigenvalue λ if

$$Sv = \lambda v$$

Rearranging the equation, we see that **v** is a solution of the homogeneous system of equations

$$(S - \lambda I)v = 0$$

where **I** is the identity matrix of size *n*. Non-trivial solutions exist only if the matrix $(S - \lambda I)$ is singular which means $det(S - \lambda I) = 0$. Therefore eigenvalues of S are roots of the characteristic polynomial

$$p(\lambda) = det(S - \lambda I) = 0$$

Figure 3: Definition of Eigen Vectors and Eigen Values

Step 2: Calculate the Covariance Matrix. The covariance matrix can be calculated two ways to achieve a square matrix. $S_1 = \frac{1}{n-1}YY^T$ will yield a m-by-m matrix and $S_2 = \frac{1}{n-1}Y^TY$ will yield a n-by-n matrix where Y is originally m-by-n. So, for our example we randomly generated 10 normally distributed features each with 100 samples to produce a Y matrix 100-by-10 in shape. Then we computed the zero mean data Y by transposing Y and subtracting the mean of the 100 points from each normal distribution. Now we can compute the eigen vector and eigen values from each covariance matrix. S_1 will have a shape of 100-by-100 and will yield 100 eigen values and a 100-by-100 eigen vectors matrix. Just the same way the S_2 covariance matrix will have shape 10-by-10 and will yield 10 eigen values and a 10-by-10 eigen vector matrix.

Step 3: Compute the eigen vectors and eigen values of the covariance matrix. The eigen values show the significance of each eigen vector. So, we need to organize the eigen vector matrices of the covariance matrix. The intuitive direction is to order the eigen vector matrix from largest to smallest where the first index position of the eigen values will be the largest and the last eigen value will be the smallest. And from the ordering of the eigen values the eigen vector columns correspond to each eigen value. So, the eigen vector matrix will be ordered from most significant to least significant in the eigen space. *NOTE: the eigen space dimensionality corresponds to the number of eigen values; so, for 100 eigen values we have an eigen space of 100-mutlidimensions and for 10 we have a 10-dimensional eigen space.

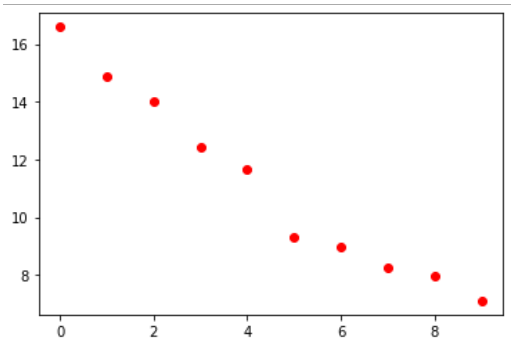

Figure 3: First 10 Eigen Values of S_1

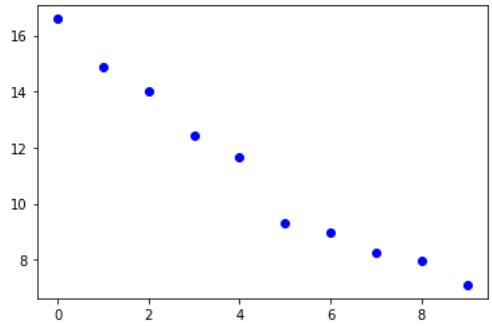

Figure 4: 10 Eigen Values of S_1

We can now take the first eigen vector column from the S_1 and S_2 eigen vector matrix which will be the most significant eigen vector based on our reordering. The comparison between the eigen vectors of S_1 and the eigen vectors of S_2 is that they can be made equal from the following manipulation (first eigen vector column of S_1 will be v_1, first eigen vector column of S_2 will be v_2:

$$v_1 = \frac{Yv_2}{\|Yv_2\|}$$

That is to say that the first n=10 eigen vectors of S_1 will be equal to the normalized 10-by-10 matrix Yv_2. Key word here is normalized which means that it is L_1 (from the L_p distance metric) or Manhattan distance normalized. And therefore, the Eigenvalues will be the same as shown:

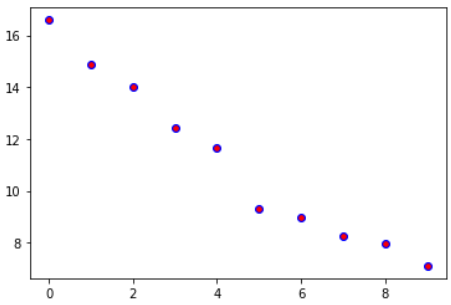

Figure 5: 10 Eigen Value Equivalence

III. Part 2: Principal Component Analysis

Now that we know we can approximate one covariance matrix with the other; let's see how this can be useful. We start with an image data set of 11 Aligned Fighter Jets. Each image is 256-by-256 binary pixels where the white represents the jet, and the black represents the background.

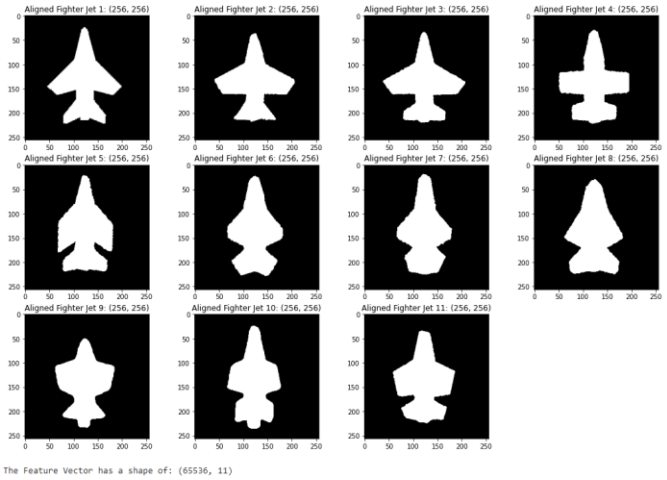

Figure 6: Image Data Set of 11 Aligned Fighter Jets

We will reshape the images so that we create a data frame where each column represents a different image, and each row represents a sample pixel in the image. So, we are still sticking to our m-sample rows by n-feature column data representation. We can see the reshaped dimensions at the bottom of the above figure is 256^2-by-11 = 65536-by-11. Again, each pixel is a sample. Each Feature is a different image of a jet.

Now here comes the real trick: If we compute the matrix $S_1 = \frac{1}{n-1} YY^T$ we would end up with a 65536-by-65536 covariance matrix which would require 32GB of storage space. This is completely possible to compute using row-by-column loading and unloading for hard storage of a table. But there is no need since we can approximate S_1 with S_2 as we found from the 10 Eigen Value Equivalence in Figure 5. So, we can use $S_2 = \frac{1}{n-1} Y^T Y$ to get a matrix of size 11-by-11 whereby we are looking at a covariance matrix covarying each feature to one another instead of covarying each sample to one another. And then approximating the first, and most significant, 11 eigen vectors of S_1 with all 11 eigen vectors of S_2.

Step 4: Reduce dimensionality and form the feature vector. We can ignore the principal components (PCs) of lesser significance. By doing so we lose some information, but if the eigen values are insignificantly small, you don't lose much information. For this step we need to determine how many eigen vectors should be used of the 11 that we have computed. Since the eigen values are ordered from most significant to least significant we will see that the largest variability in the data comes from the first few eigen values – and we have an equation to calculate this. If we let the percent of variance in data be represented by $\%\sigma^2$; then we can find the following equation follows from the significance of variance is extracted by the eigen value:

$$\%\sigma^2 = \frac{\sum_{i=0}^{r-1} \lambda_r}{\sum_{j=0}^{m-1} \lambda_m} * 100$$

Again, summation indexing may vary between programming languages. There is m=11 samples for which we have approximated $S_1 = \frac{1}{n-1} YY^T$. The bottom of the $\%\sigma^2$ equation will yield the total summation of all 11 eigen values. The top will yield a portion of the 11 eigen values summed together. We can create a code where when we adjust r we select a number of these eigen values to compute this % variance ($\%\sigma^2$). Let's show what this looks like for r = 3, 10 and 11:

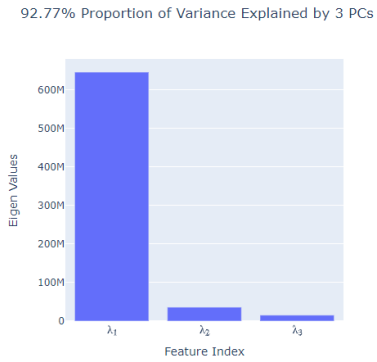

Figure 7: Eigen Value vs. Index for 3 PCs

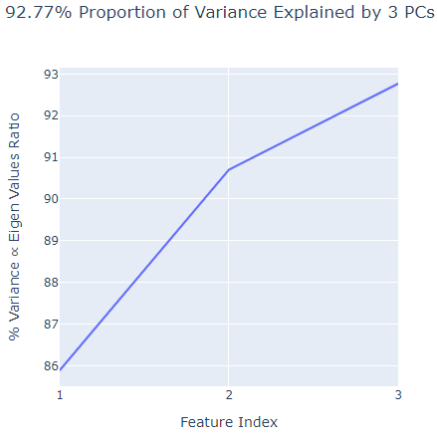

Figure 8: 92.77% of Variance Explained by 3 PCs

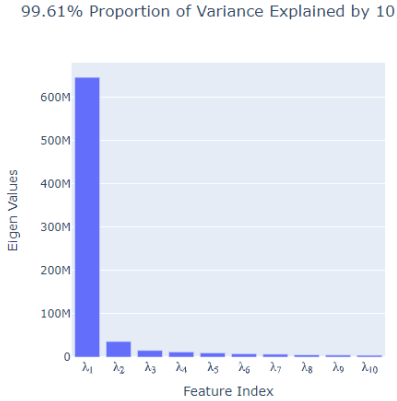

Figure 9: Eigen Value vs. Index for 10 PCs

Figure 10: 99.61% of Variance Explained by 10 PCs

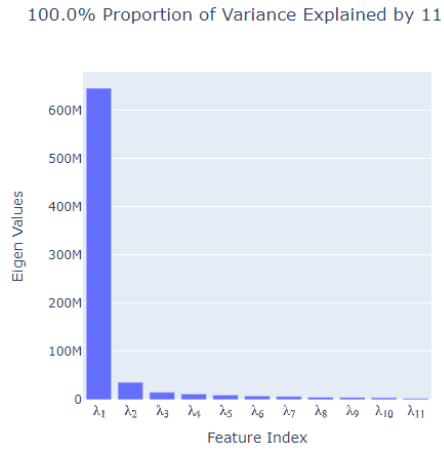

Figure 11: Eigen Value vs. Index for 11 PCs

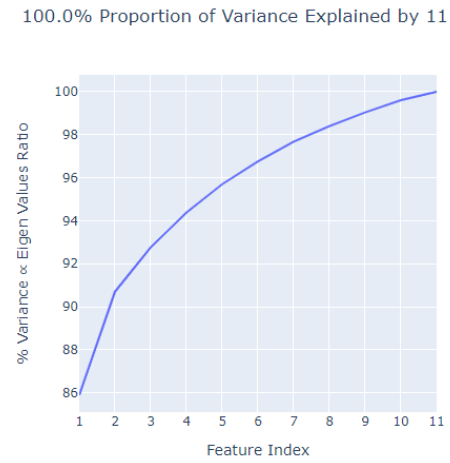

Figure 12: 100% of Variance Explained by 11 PCs

It is clear from Figures 8, 10 and 12 that getting rid of the last eigen value would only affect the data reconstruction by 0.39% so it is easy to eliminate to reduce our dimensionality. However, if we go even further and use the first 3 largest Principal Components (PCs or the first 3 eigen values) we still have 92.77% of the data intact. We only lose 7.33% of our total data!

So now let's create a feature vector from our 3 eigen vectors. First, we do the trick; multiply the first eigen vector of S_2 with the original data and normalize it to get the first eigen vector of S_1. Then, do the same for approximating v_2 and v_3 for $S_1 = \frac{1}{n-1} YY^T$. We should then have 3 Principal Component (PCs) feature columns which we append together to get a 65536-by-3 column vector and then transpose this to get a 3-by-65536 Row Feature Vector. The Row Feature Vector columns match the dimensionality of our original Zero Mean Data if we Transpose the Zero Mean data into a Row Zero Mean data matrix. Which we already have been transposing this to extract the means and standard deviation anyways. So, we can now matrix multiply the Row Feature vector (3-by-65536) with the Row Zero Mean Data (65536-by-11) to achieve the Final Data (3-by-11). If we plot this three-dimensional data, we can see the clustering of the aligned fighter jet images.

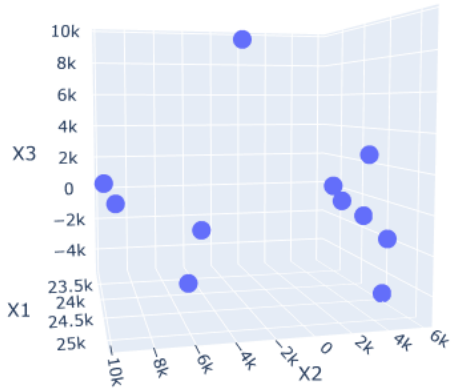

Figure 13: 3-D Final Data 11 Jets

When plotting various numbers of these fighter jets (i.e., 4,5,6,7,8,9,etc...) we see that the outlying point at the top of the graph is at the 9th Aligned Fighter Jet Feature. The 9th image in the aligned fighter jet image set is the outlier and the rest fit into two Classes. We can separate the two classes via Classical Bayes Classification.

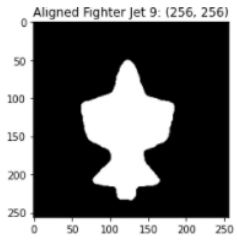

Figure 14: 9th Aligned Fighter Jet Image Outlier

Step 5: Derive the new data. We can now reconstruct the Zero Mean Original Data (65536-by-11) from multiplying the Feature Vector (65536-by-3) with the Final Data (3-by-11). Finally, we need to add the Original Mean back to the Zero mean Original Data to get the Row Reconstructed Original Data. Then transposing the Row Reconstructed Original Data and we achieve the 65536-by-11 Reconstructed Data. We can

now take thee 11 columns as each reconstructed image and reshape the pixels into 256 by 256 images.

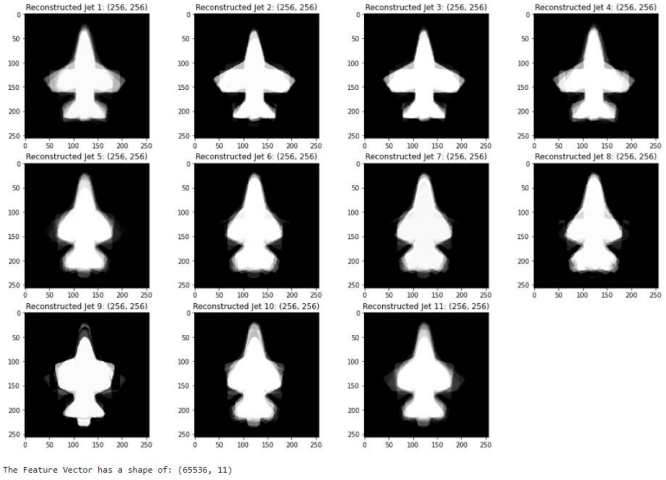

Figure 15: Reconstructed Jet Images from 3 PCs

If we did the same for 11 PCs, then we would just get 100% of the original image. However, since we found that 10 PCs encompasses 99.61% of the variance in the data Let's look at 10 PCs.

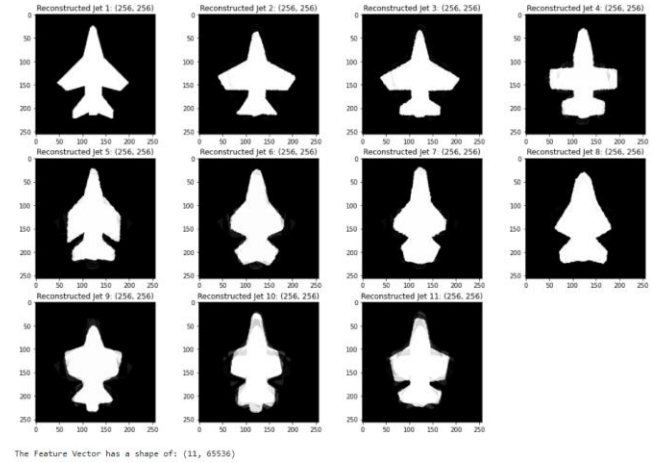

Figure 16: Reconstructed Jet Images from 10 PCs

It should be noted that the diagonalization of the covariance matrix came from the square eigen vectors matrix decomposition.

IV. Part 3: Fisher's Linear Discriminant Analysis

Sir Ronald Aylmer Fisher FRS. was a statistician and geneticists who developed the Linear Discriminant (aka., the Linear Discriminant Analysis – LDA) to find linear combinations of features that characterize (or separate) two or more classes of events. It is simply a ratio measure of the variance between two or more classes to the variance within two or more classes. From our previous discussion this could be viewed as the Signal-to-Noise Ratio (SNR) but in this case we want to maximize a function which represents the difference between the means, normalized by the scatter (or a measure of the within-class variability). For each class $C_i; i \in [1,2]$ we define a scatter as:

$$\tilde{s}_i^2 = \sum_{y \in C_i} (y - \tilde{\mu}_i)^2 \; ; i \in [1,2]$$

We can elaborate on the scatter if $y = w^T x$ then $\tilde{\mu}_i = w^T \mu_i$:

$$\tilde{s}_i^2 = \sum_{y \in C_i} (y - \tilde{\mu}_i)^2 = \sum_{x \in C_i} (w^T x - w^T \mu_i)^2$$
$$= \sum_{x \in C_i} w^T (x - \mu_i)(x - \mu_i)^T w = w^T S_i w$$

Elaborating on two classes we have:

$$\tilde{s}_1^2 + \tilde{s}_2^2 = w^T S_1 w + w^T S_2 w = w^T (S_1 + S_2) w = w^T S_W w$$
$$= \tilde{S}_W$$

Where $\tilde{S}_W$is the within-class scatter matrix of the projected samples y.

Similarly, we can find the projection of the means in the second dimension of the feature space where classes interact between each other and find the L_2 distance metric between those class means:

$$(\tilde{\mu}_1 - \tilde{\mu}_2)^2 = (w^T \mu_1 - w^T \mu_2)^2 = w^T (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T w$$
$$= w^T S_B w = \tilde{S}_B$$

Where $\tilde{S}_B$ is the between-class scatter matrix of the projected samples y. We can finally express the Fisher criterion, $J(w)$, in terms of $\tilde{S}_B$ and $\tilde{S}_W$ as:

$$J(w) = \frac{|\tilde{\mu}_1 - \tilde{\mu}_2|^2}{\tilde{s}_1^2 + \tilde{s}_2^2} = \frac{w^T S_W w}{w^T S_B w} = \frac{\tilde{S}_W}{\tilde{S}_B}$$

Fisher's criterion $J(w)$ is basically a similar measure as the SNR that we defined at the very beginning. To find the maximum of $J(w)$, we can differentiate and equate the derivative to zero:

$$\frac{d}{dw} J(w) = \frac{d}{dw} \left(\frac{w^T S_W w}{w^T S_B w} \right) = 0$$

By the derivative division rule:

$$\Rightarrow (w^T S_W w) \frac{d}{dw} (w^T S_B w) - (w^T S_B w) \frac{d}{dw} (w^T S_W w) = 0$$
$$\Rightarrow (w^T S_W w)(2 S_B w) - (w^T S_B w)(2 S_W w) = 0$$

Dividing by $2 w^T S_W w$:

$$\Rightarrow \left(\frac{w^T S_W w}{w^T S_W w} \right) (S_B w) - \left(\frac{w^T S_B w}{w^T S_W w} \right) (S_W w) = 0$$
$$\Rightarrow S_B w - J(w) S_W w = 0$$

Dividing by S_W:

$$\Rightarrow S_W^{-1} S_B w - J(w) w = 0$$

The above equation is reminiscent of the eigen decomposition where the solution is singular (i.e., $\det(S_W^{-1} S_B w - J(w) w) = 0$). In which case we can treat $J(w)$ as the eigen value and w would be our eigen vector. Therefore, we are trying to get the maximum eigenvectors that will encompass all the data.

Formally, we can express this maximization optimality as:

$$w^* = \arg\max_w J(w) = \arg\max_w \left(\frac{w^T S_B w}{w^T S_W w} \right) = S_W^{-1} (\mu_1 - \mu_2)$$

The above equation is known as Fisher's Linear Discriminant. The Discriminant can be used for Bayes Classification or it can be used for projection of the data down to one dimension. To obtain this projection we can use the equation $S_W^{-1}S_B w - J(w)w = 0$ where the covariance matrix $S_X = S_W^{-1}S_B$.

Now let's look at a real example of using Fisher's Discriminant to perform dimensionality reduction to C-1 dimensions (i.e., if we have 2 classes, we can reduce the feature space dimensionality to $C - 1 = 2 - 1 = 1$ dimension). Here are two gaussian normally distributed data sets where class 1, X_1, has two features $N(\mu, \sigma) = N(200,8)$, and $N(\mu, \sigma) = N(400, 30)$ each with 300 samples. Class 2, X_2 also has two features $N(\mu, \sigma) = N(100,50)$, and $N(\mu, \sigma) = N(-100,20)$. After working through LDA it was carefully decided upon to use standard deviation instead of variance because the wider spread of the data gave better results for projection.

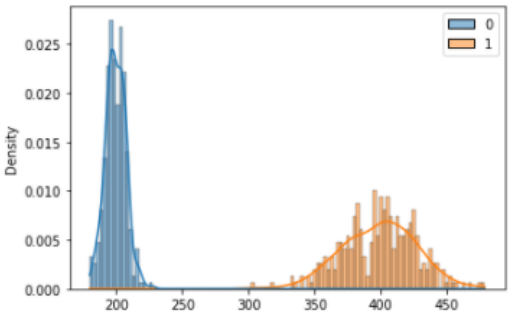

Figure 17: pdf of Class 1

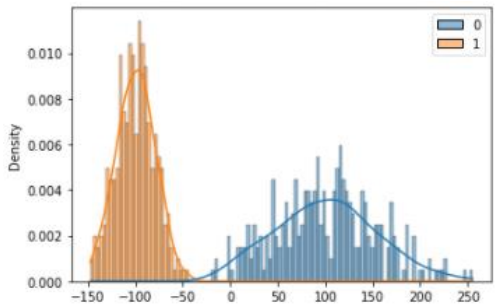

Figure 18: pdf of Class 2

Now we can scatter plot the two classes for the two features and we should see 300-points clouds in the feature space:

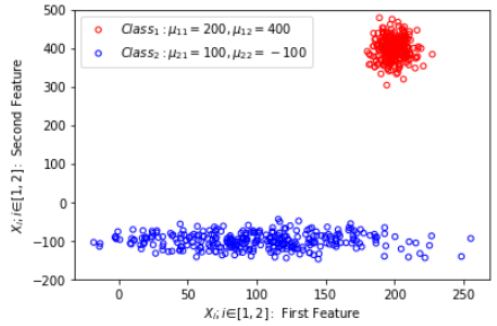

Figure 19: Feature Space for Class $C_i; i \in [1,2]$

The mean vector from each class in the first and second dimension of this feature space is computed here:

```
Mu1 = np.mean(X1, axis=1).reshape(2,1)
Mu2 = np.mean(X2,axis=1).reshape(2,1)

print(f"Mean Vector of Class 1: \nMu1 = \n{Mu1}\n")
print(f"Mean Vector of Class 2: \nMu2 = \n{Mu2}\n")
```

```
Mean Vector of Class 1:
Mu1 =
[[199.62739946]
 [398.72325334]]

Mean Vector of Class 2:
Mu2 =
[[100.78338173]
 [-99.66195622]]
```

Figure 20: Mean Vectors in Feature Space

Now we can compute the covariance of the first and second class:

```
# covariance of the first class
S1 = (X1 - Mu1) @ (X1 - Mu1).T  # or np.cov(X1)

# covariance of the second class
S2 = (X2 - Mu2) @ (X2 - Mu2).T # or np.cov(X2)

print(f"Covariance of Class 1: \nS1 = \n{S1}\n")
print(f"Covaraince of Class 2: \nS2 = \n{S2}\n")
```

```
Covariance of Class 1:
S1 =
[[ 17044.55094126   2544.68620914]
 [  2544.68620914 233322.88189857]]

Covaraince of Class 2:
S2 =
[[842815.06437333  -6908.30346067]
 [ -6908.30346067 119229.96886425]]
```

Figure 21: Covariance in Feature Space

Following the steps of our previously mentioned mathematics, we find the within and between Fisher scatter matrices:

```
# Within-Class Fisher Scatter matrix Sw
Sw = S1 + S2

print(f"Within-class Fisher Scatter matrix: \n\nSw = \n{Sw}\n")
```

```
Within-class Fisher Scatter matrix:

Sw =
[[859859.61531458  -4363.61725153]
 [ -4363.61725153 352552.85076282]]
```

```
# Between-Class Fisher Scatter matrix Sb
Sb = (Mu1 - Mu2) @ (Mu1 - Mu2).T

print(f"Between-class Fisher Scatter matrix: \n\nSb = \n{Sb}\n")
```

```
Between-class Fisher Scatter matrix:

Sb =
[[  9770.1398408   49262.39648966]
 [ 49262.39648966 248387.81710897]]
```

Figure 22: Computing Within and Between Scatter Matrices

Finally, we compute the LDA projection eigen vectors W_1 and W_2. Where W_1 corresponds to the largest eigen value (i.e., the first eigen vector extracted) which gives the axis onto which we should project our data for dimensionality reduction. W_2 corresponds to the projection eigen vector that can discriminate the two classes. Where anything above the discriminant line W_2 can be treated as Class 1 and anything below the discriminant can be treated as Class 2. That is to say W_2 is our discriminant for Bayesian Classification where as W_1 is our projection vector for dimensionality reduction. Two very useful but very different tools.

```python
# Compute the LDA projection
invSw = np.linalg.inv(Sw)

invSw_by_Sb = invSw @ Sb
# or
Sx = invSw @ Sb

# Getting the Eigen Value
[D, V] = np.linalg.eig(Sx) # unpacking eigen_value, eigen_vector

# Deal with the sorting issue just in case:
idx = D.argsort()[::-1]
D = D[idx]
V = V[idx]

# The Projection Vector W
W1 = V[:,0].reshape(2, 1)
W2 = V[:,1].reshape(2, 1)

# ** Note: Need non-zero eigen values for this...
print(f"Projection Vector: \n\nW1 = \n{W1}\n")
print(f"Projection Vector: \n\nW2 = \n{W2}\n")
print(f"Eigen Values: \n\nD = \n{D}\n")
```

```
Projection Vector:

W1 =
[[ 0.19453943]
 [-0.9808947 ]]

Projection Vector:

W2 =
[[-0.9962964 ]
 [-0.08598541]]

Eigen Values:

D =
[0.71736641 0.        ]
```

Figure 23: LDA Projection Computation

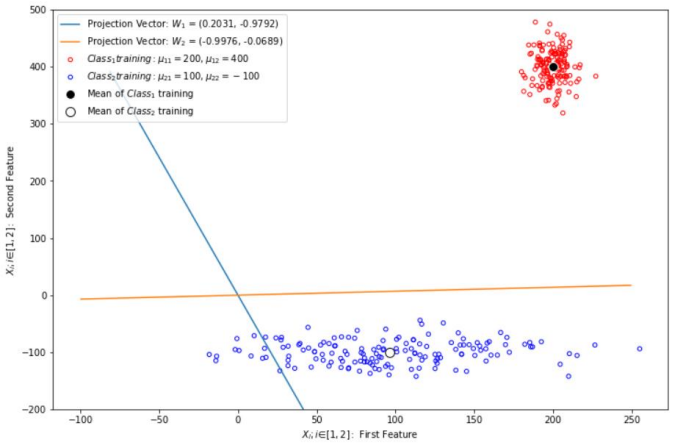

Figure 24: Feature Space with Linear Discriminant W_2 and projection vector W_1

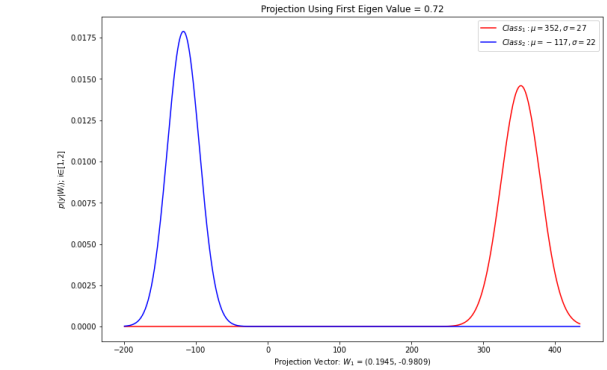

Figure 25: Projection onto W_1

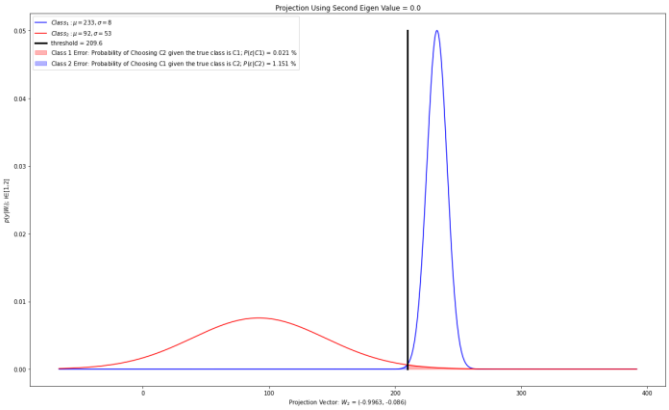

Figure 26: Projection onto W_2

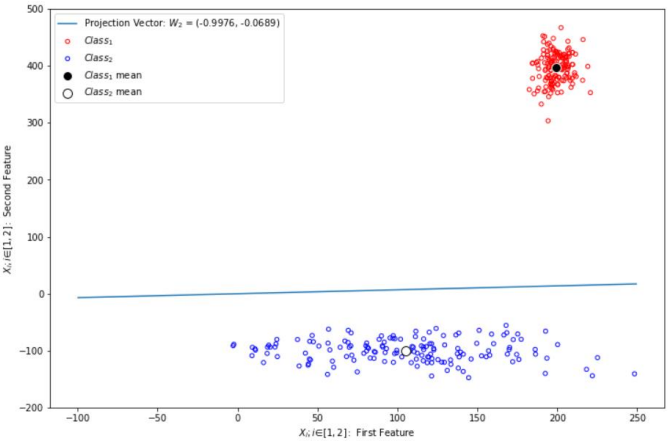

Figure 27: Bayesian Classification of Test Data

If we compare Figures 24 and 27, we see the training data perfectly classified the testing data with an error of classification of 0% for both Class 1 and Class 2:

```
Error of Classifying X test as Class 1 = 0 %
Error of Classifying X test as Class 2 = 0 %
```

Figure 28: Error in Classification

Clearly the Fisher's linear discriminant W_2 perfectly classifies the testing data in the feature space when the discriminant was made using only 50% of the original data.

And projection onto W_1 can retain all the classification information when projected and reduce our dimensionality from 2 to one feature per class.

V. Conclusions

First, we started with two fundamental questions:

1. Which parameters do we want to keep from a data set?

Answer: Keep the parameters that are independent of one another (i.e., uncorrelated, or low covariance).

2. How do we determine which parameters best express a data set?

Answer: The parameters that best represent our data will have a high Signal-to-Noise Ration/Ratio (SNR or ratio of variances). Traditionally, the SNR may be approximated as the ratio of the average signal value (μ_{signal}, mean of the signal) to the standard deviation of the signal (σ_{signal}).

$$SNR = \frac{\sigma_{signal}^2}{\sigma_{noise}^2} \approx \frac{\mu_{signal}}{\sigma_{signal}}$$

A high SNR (>> 1; much, much greater than one) indicates high precision of data, whereas a low SNR indicates noise contaminated data.

Then we moved into an understanding that we can approximate the high dimensionality of an m-by-m sample covariance matrix $S_1 = \frac{1}{n-1} YY^T$ with the eigen vectors of a much smaller n-by-n feature covariance matrix $S_2 = \frac{1}{n-1} Y^T Y$ by normalizing the vector projection of the ordered eigen vectors. then PCA was used to take the eigen decomposition of the covariance matrix and ordering the eigen vector based on the significance of the eigen value we can find Principal Components that best represent our data. The Principal Component % variance was analyzed by using the significance of the eigen value whereby we looked at the use 3, 10 and 11 eigen vector columns (the principal components - PCs). We solved the diagonalization of the covariance as a square singular eigen decomposition matrix. After diagonalizing the Covariance matrix, we reconstructed our data removing 8 of the 11 components when only using 3 PCs and reconstructing the data using 10 PCs. The reconstruction using 10 PCs demonstrate how little the 11th PC had any affect on the image reconstruction of the Jet image data set.

Finally, we found Fisher's Criterion $J(w)$ using the within and between class scatter matrices. The Criterion $J(w)$ was the diagonalized singular ordered eigen decomposition of the ratio between the within and between scatter matrices. Through this we achieved two eigen vectors – the maximum eigen value corresponded to the maximum eigen vector which projected the data without information loss and the second eigen vector was a perfect linear discriminant of the two classes. Analyzing the data with Bayesian Classification in the feature space we found perfect classification when the data was trained to create the linear discriminant and tested for values above and below the linear discriminant. When we mention perfect classification, it means all the test points that were above the computed linear discriminant were correctly classified as class 1. And all the test points below the linear discriminant were correctly classified as class 2.

From the goal we set it is clear we achieved the dimensionality reduction and Bayesian Classification utilizing these extremely powerful tools: PCA and LDA.

VI. References

[1] Farag, Introduction to Probability Theory with Engineering Applications, Cognella Academic Publishing, 2021.

[2] Farag, Biomedical Image Analysis Statistical and Variational Approaches.

[3] Golub, Gene Howard, and Van Loan Charles F. *Matrix Computations*. The Johns Hopkins University Press, 2013.

[4] Reuven Rubinstein and Dirk P. Kroese Simulation and the Monte Carlo Method, 3rd Edition, Wiley, New Jersey, 2017

[5] R. O. Duda, P. E. Hart and D. G. Stork: Pattern Classification, 2nd Edition, Wiley, 2000.

VII. Appendix: IPython Code:

The IPython Notebook is available at:
https://github.com/MichaelFerko