University of Southampton, School of Electronics and Computer Science
Department

COMP3223 Coursework 1

Adrian Azzarelli-Bonnardel, 29901391, aab1g18@soton.ac.uk

December, 06, 2020

Date of Write-Up ..... December 05, 2020
Program Verification ..... December 06, 2020
Submission Made ..... December 06, 2020

Note the accompanying code is outlined in a file labeled "README.txt". Program names are formatted as such, "[section]_[subsection]_[part]_[additional information].py", and appropriate amount of comments have been made within each program in order for you to understand it.

Contents

1 Linear Regression with Non-Linear Functions **2**
1.1 Problem Outline 2
1.2 Performing Lin-Reg. with PBF and GRBF methods 3
1.3 Generalisation 5

2 Classification **8**
2.1 Problem Outline 8
2.2 Data I: Classify 2 Gaussians 9
2.3 Data II: Classify Iris Data 14

1 Linear Regression with Non-Linear Functions

1.1 Problem Outline

This sub-section serves the purpose of showing my understanding with regard to problem formatting

Consider a problem $\Phi(C, E, X, W, Y, V)$ of Linear Regression with a set of training data-points X_{Tr} which aims to ***fit*** equation, E, by using a set of (conditional) basis functions, C, to determine a list of directional weight vectors, W. Where a set of predicted y-values, Y, can be determined using $W \cdot X_{Te}$, where X_{Te} represents a set of testing data points. Fitted data is represented by a set of coordinate locations (*predictions*) represented by V.

$\Phi(C, E, X_{Tr}, W, X_{Te}, Y, V)$ where

$$\begin{aligned} C =& \{\phi_{PBF,1}(x_n), \ldots, \phi_{PBF,j}(x_n), \phi_{GRBF,1}(\mu_j; x_n), ;\phi_{GRBF,j}(\mu_j; x_n)\} \\ E =& e_1 \\ X =& \{x_{Tr,1}, x_{Tr,2}, \ldots, x_{Tr,N_0}, x_{Te,1}, x_{Te,2}, \ldots, x_{Te,N_1}\} \\ W =& \{w_1, w_2, \ldots, w_p\} \\ Y =& \{y_1, y_2, \ldots, y_{N1}\} \\ V =& \langle (x_{Te,1}, y_1), (x_{Te,2}, y_2), \ldots, (x_{Te,N_1}, y_{N1}) \rangle \end{aligned}$$

Where N_0 and N_1 represent the number of training and testing data-points[1], respectively; p represents the degree of weight vectors; PBF represents a *Polynomial Basis Function* and $GRBF$ represents a *Gaussian Radial Basis Function*; x_n represents a point in $\overline{x}$ of training data X_{Tr}[2], where $n = 1, \ldots, N_0$, and μ_j represents a single *centroid* in $\overline{\mu}$. The equation to fit, E, can be described by Equation 1.

In order to calculate $\overline{w}$, which represents W mathematically, a design matrix $\overline{A}$ must be determined. This is done using the $GRBF$ and PBF methods, which are described respectively by Equation 3 and 4. More precisely, $\overline{w}$ is calculated using Equation 2. By extension, Y can be calculated using $\overline{y} = \overline{w} \cdot \overline{x_{Te}}$. As a result we can determined V.

$$y(x) = sin(4\pi \cdot x) + \epsilon \tag{1}$$

$$\overline{w} = \overline{A^T}(\overline{A} \cdot \overline{A^T} + \lambda \mathbb{I}_N)^{-1}\overline{y} \tag{2}$$

$$\phi_j(\mu_j; x_n) = \exp\Big(-\frac{(\mu_j - x_n)^2}{2s^2}\Big) \tag{3}$$

$$\phi_j(x_n) = x_n^j \tag{4}$$

[1] Which is dependant on training-test split, discussed later

[2] $\overline{x}$ is a vector not a representation of the mean in x

1.2 Performing Lin-Reg. with PBF and GRBF methods

In section 2.1 of the coursework, we were first asked to construct the design matrix, $\overline{A}$. In order to do this I would need to randomly generate $N = 15$ data points and split the data into for $\overline{x_{Tr}}$ and $\overline{x_{Te}}$[3]. I used the given function, *make_design*, to determine design matrices, *design_A* and *design_B*, relating to PBF and GRBF respectively. In order to accomplish this I randomly generated a list of input-values, μ, for thre GRBF method and a list of degrees for the PBF method. See requirements in Equations 3 and 4.

Following this, we were asked to determine the set of weights, W. In order to execute this I implement Equation 2 as a function, labeled *weight_gen*, to determine two weight vectors, *w_A* and *w_B*, for PBF and GRBF, respectively.

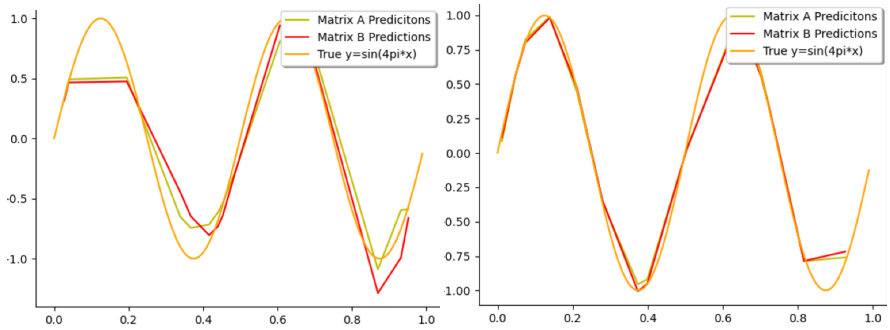
To ensure the implementation worked correctly I applied the weights to my testing data, $\overline{x_{Te}}$, in order to calculate y-predictions, $\overline{y_{Te}}$, done for both PBF and GRBF methods. Then I plot the (x,y) values against the true curve $y = sin(4\pi x)$.

Further Comments:

While developing this program I had to address several issues, (1) avoid over-fitting, (2) data-size issues and (3) tweaking values for the degree of weight-vector, p, and the strength of regularisation coefficient, λ.

My interpretation of 2.1 was to determine y-predictions by re-using training data as testing data. This would cause over-fitting, hence concern (1). There were two solutions, split N data points into training and testing sets, or simply generate new testing data values, under the same conditions as training data generation. I chose the latter, as the former would lead me to issue (2).

My concern regarding (2), is that 15-data points is small. In my eyes, it is not be enough to develop reliable results, and is not realistic when concerning real-application[4]. In comparison, take the other extreme, using 2,000-points. Figure 1 illustrates the quality difference between data-sizes.



(a) N=30 (b) N=2,000

Figure 1: Plots used for debugging

[3]I had preempted the implementation of data splitting so you will find simple split functionality within programs for section 2.1

[4]This is a naive comment. I do understand that linear-algorithms require less data that non-linear algorithms, though I still stand to say 15-points is not enough. I must also accept that $N_{Tr} >> N_{Te}$ could lead to over-fitting

Issue (3) was easily solved using the plotting functionality within my program, I determined good results. I assigned $\lambda = 10^{-7}$ and $p = N$.

1.3 Generalisation

Section 2.2 of the coursework focused on developing a working model and parameter tweaking.

In part (a) we were required to split data (condition, $N = 25$). I took a look at *scikit-learn*'s *train_test_split* function and recognised two main paths of data-splitting: (1) randomly shuffling all data and partitioning given a split ratio, (2) non-random shuffle and split. In lieu of this I developed two methods of my own: (1) random selection selection from a list of available data, (2) randomly generate training data and separately generate testing data under the same conditions. I have used (1) in order to emulate data-splitting with regards to real data[5].

In part (b)[6] we were asked to calculate the mean of square residuals (MSR) using the test-data and plot in comparison to degree variation, p, and strength of regularisation (SoR) coefficient, λ. Regarding p, I varied it under the condition $1 \leq p \leq 30$ and produced Figure 2(a). It is evident that the turning point exists between $6 \leq p \leq 10$ and I would be happy to say 8 may be the ideal value. You may be inclined to set $p = 30$, though this presents an issue of increased run-time due to developed mathematical complexity and higher likelihood of over-fitting[7]. It is also evident from the graph little-to-no disparity exists between PBF and GRBF regarding MSR values. Following this, I varied λ under the condition $10^{-20} \leq \lambda \leq 10^{10}$, with a step of 10^i. This allowed me to develop Figure 3(a). Evidently, there exists disparity between the PBF and GRBF methods, where GRBF MSR values are lower for lower values of λ. There also exists clear turning points, where MSR values are most-optimal between $10^{-14} \leq \lambda \leq 10^{-6}$ and $10^{-13} \leq \lambda \leq 10^{-2}$ for GRBF and RBF respectively.

Additionally, I will continue my comparison between low and high amounts of data. Figure 2(b) and Figure 3(b), show MSR values when $N = 1000$. For p-variation, there is little difference between higher and lower sums of data, though this is not the case for λ-variation, where there is more clarity as to what SoR coefficients are most optimal. To conclude, I am satisfied with my initial choice of $\lambda = 10^{-7}$ and p=10.

In part (c) we were asked to vary split along with the tweaking-variations applied in part (b). Figure 4 illustrates the plots from each variation regarding each method. Figure 4(a), shows that a 0.9 split-ratio is most ideal given the MSR value is consistently the lowest result (especially at our chosen value of p=10), whereas 0.3 is consistently worse which is surprising as I imagined a 0.1 split-ratio to produce worse results, given a lower amount of training data would most likely lead to under-fitting[8]. Figure 4(b) acts more as I had imagined, where a 0.1 split-ratio performs worse, though 0.5, 0.7 and 0.9 all perform similarly as optimal values. Figure 4(c), similar to (a) shows 0.9 split0-ratio being optimal. Figure 4(d) is the most surprising graph given there is no consistent "best" and the MSR plots are all similar.

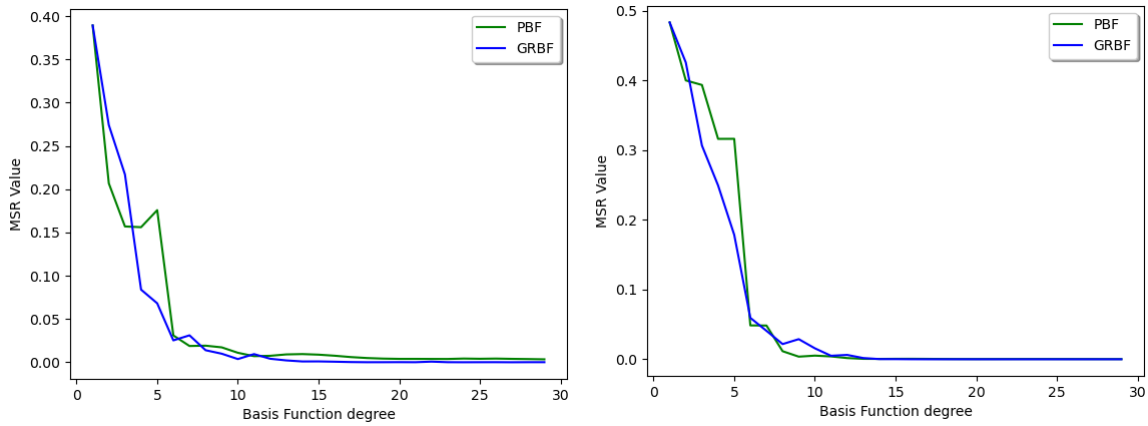
In my opinion, to rely on these results for an accurate conclusion is unwise. As we use a low amount of data, the difference between a 0.9 split and 0.7 split is low - 5 data points. It is evident this is not enough when you run the program several times and notice the graphs never produce constant shapes. Do understand I also implemented an 8-iteration loop to compute an average MSR, in order to reduce uncertainty; despite this there still exists a large variation is plot-shape every time I run the program. Again, this re-affirms my prior

[5] Not randomly generated

[6] Here I fixed split-ratio for training data at 0.9

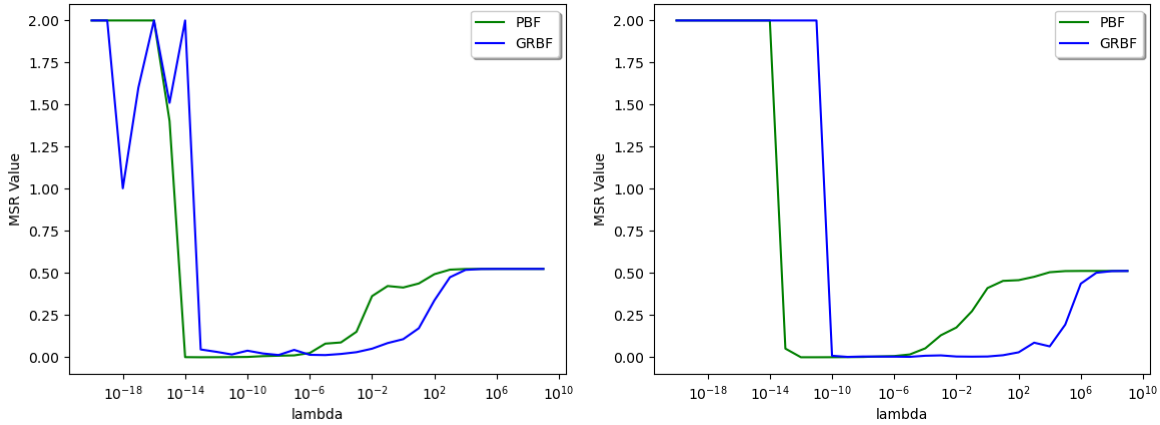
[7] If I increase the degree of my basis function, I need to calculate more associated weights, which is timely. Additionally, a higher degree causes a "strict" fit, thus over-fitting

[8] Which would mean predictions are less accurate



(a) N=25 (b) N=1000

Figure 2: Varying degree p to determine optimal value, (split was 0.9)



(a) N=25 (b) N=1000

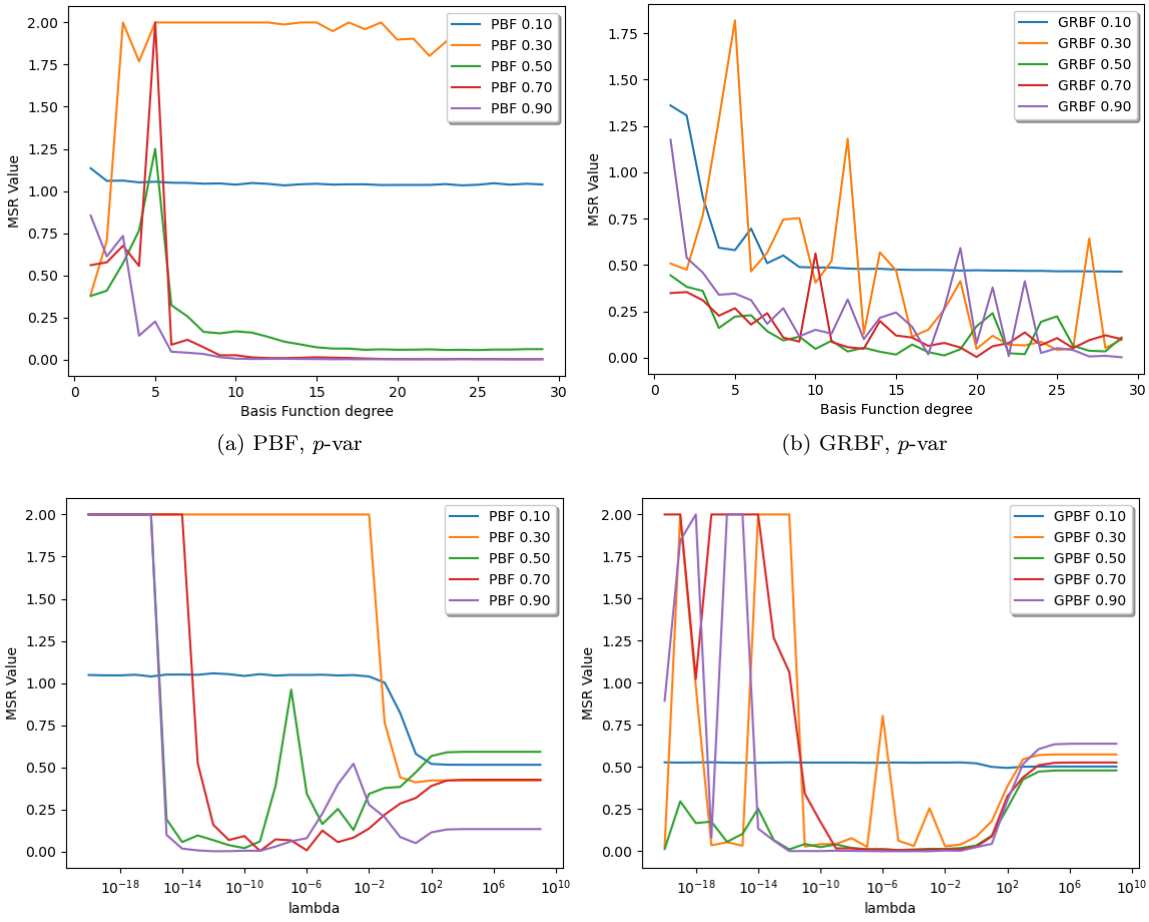
Figure 3: Varying λ to determine optimal value, (split was 0.9)

conclusion that higher volume of data is necessary to make more evident distinctions, as such.

Further Comments:

When $N_{Tr} > p$ there exists lower variance, hence better performance, whereas when $N_{Tr} = p$ there exists a large-er variance of results and where $N_{Tr} < p$, there is a high probability of forming a non-unique design-matrix, meaning solutions for ω using Equation 2 will tend to infinity. Alternatively we could use the method of gradient descent[9].

[9] Although I did not, I wanted to outline this concern as we are working with a low-volume data set, so possibility of $N_{Tr} < p$ is higher



(a) PBF, p-var

(b) GRBF, p-var

(c) PBF, λ-var

(d) GRBF, λ-var

Figure 4: Split and parameter tweak variation for PBF and GRBF methods

2 Classification

2.1 Problem Outline

In this section we are required to implement linear discriminant analysis (LDA) in order to solve a K-class classification problem.

In section 3.1 of the coursework documentation, we are required to determine two 2-Dimensional Gaussian distributions, which will allow us to randomly generate two sets of data, which will respectively fall under the two classes of distributions. To accomplish this I arbitrarily assigned a (2x1) mean matrix, μ_c (where "c" is the class label), and generated a (2x2) covariance matrix, Σ_c, by: (1) defining two independent Guassian random variables, as expressed in Equation 5, (2) applying the rv's to Equation 6, where α is a given constant, and (3) using Equation 7 to determine the final (2x2) covariance matrices. I did this twice applying two separate sets of 1-D distributions to produce two different 2-D Gaussian distributions. Otherwise there were several tasks that required us to build an LDA classification frame around our distributions and test/tweak parameters.

$$n_x \sim \mathcal{N}(\mu_x, \sigma_x),\ n_y \sim \mathcal{N}(\mu_y, \sigma_y) \tag{5}$$

$$X = n_x,\ Y = \alpha X + n_y \tag{6}$$

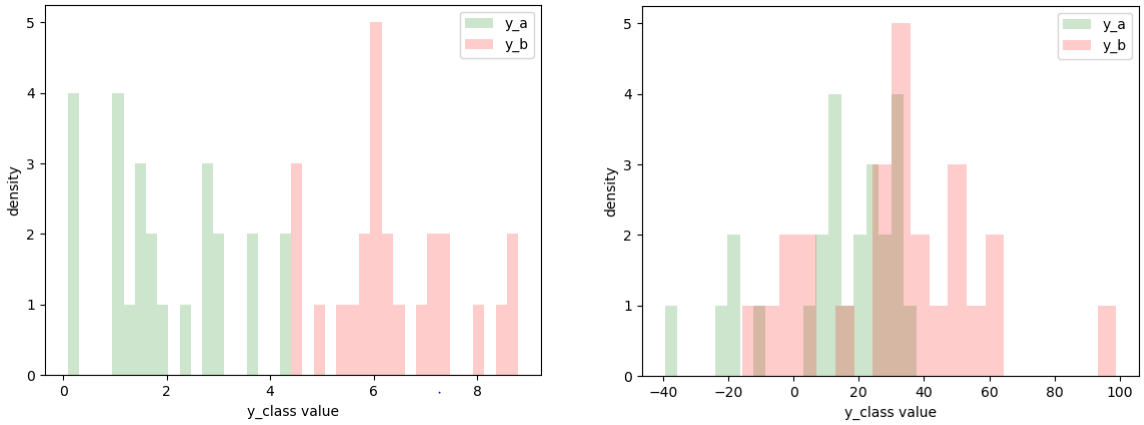
$$\Sigma = \begin{pmatrix} \mathbb{E}(X^2) & \mathbb{E}(XY) \\ \mathbb{E}(XY) & \mathbb{E}(Y^2) \end{pmatrix} \tag{7}$$

In section 3.2 of the coursework documentation, we applied Fishers LDA on the famous Iris data. Furthermore, we were asked to apply soft-max regression and an altered soft-max classifier using learned-weights to the iris data and compare classification methods. I also included additional comparison to principle component analysis (PCA), as this often comes hand-in-hand with LDA classification.

2.2 Data I: Classify 2 Gaussians

This section was split in two parts. In part 1 we visually analysed the consequence of projecting data onto a lower dimension. In part 2 we compared the estimate parameters (realistic) and the true parameters.

In part 1(a) we were asked to make two illustrative choices of ω and plot the corresponding histograms of projection-vectors y_a and y_b[10], from which I chose $\omega_1 = (1, 1)$ and $\omega_2 = (11, -1)$, and plotted Figure 5. Evidently, there is larger class separation for directional vector ω_1. This suggests ω_1 is a better vector for LDA classification, as the purpose of Fishers LDA is to maximise class separation.



(a) $\omega_1 = (1, 1)$ (b) $\omega_2 = (11, -1)$

Figure 5: Histogram of y_c projections given direction ω

Following this, part 1(b) asked us to present 2-4 values of Σ_a and Σ_b and demonstrate class separation using the directional vectors ω_1 and ω_2. In this part I simplified the definition of my cov. matrices (in comparison to my definition Σ_c in Section 2.1) for two reasons: (1) the focus of this part is on the comparison not generation, so simplifying the generation would not impact my results, and (2) I want to reduce the probability of generating a non-positive definite matrix[11]. I chose matrices (a) $\left(\begin{smallmatrix} 8 & 2 \\ 2 & 3 \end{smallmatrix}\right)$, (b) $\left(\begin{smallmatrix} 4 & 0 \\ 0 & 1 \end{smallmatrix}\right)$, and (c) $\left(\begin{smallmatrix} 8 & -2 \\ -2 & 3 \end{smallmatrix}\right)$, whose histograms for ω_1 and ω_2 projections are illustrated in Figure 6 and Figure 7. The shape is important but do not neglect the x-axis, which assigns objective values to class separation. From Figure 6 it is evident matrix (c) provides greater class separation whereas (a) provides the worse separation, so using ω_1 for (a) would be unwise. For Figure 7 there exists lower grade of class separation, Though this is expected given our prior analysis between ω_1 and ω_2. It is interesting to see that matrix (b) is shifted 50 places to the right, in comparison with (a) and (b). Though the distributions of projections in Figure 7 ar larger than Figure 6, (which can be seen as favourable), the class separation can not be excused, so clearly ω_1 is the preffered disrection matrix.

[10] My class labels are "a" and "b"

[11] I do understand that all covariance matrices are positive definite and can be part of a multivariate distribution, so suggesting it is non-positive definite does not make sense. However, what I mean by non-positive definite is a matrix having a singular solution, thus the inverse matrix (and precision matrix) does not exist. Hence, the need to avoid generating non-positive definite matrices

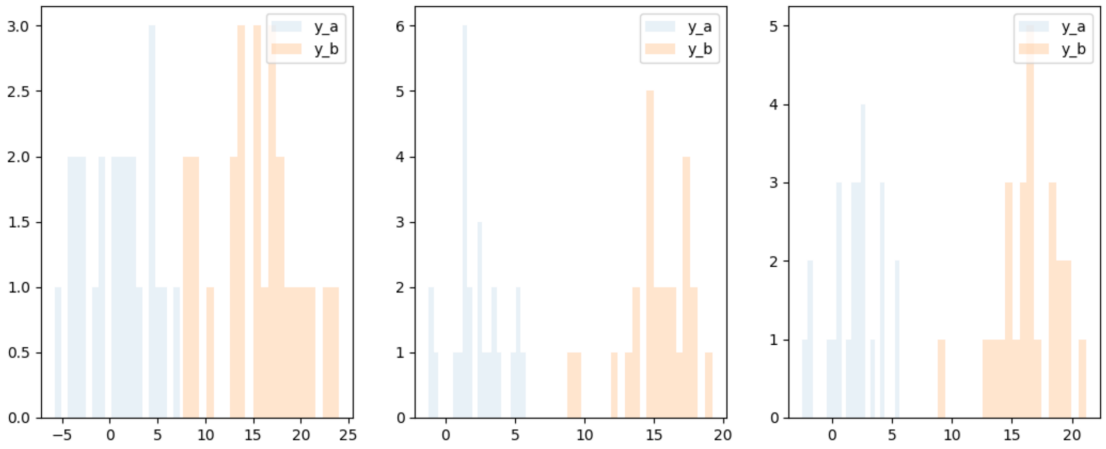


Figure 6: Histogram of y_c projections given direction ω_1 for matrices (a), (b), (c), respectively

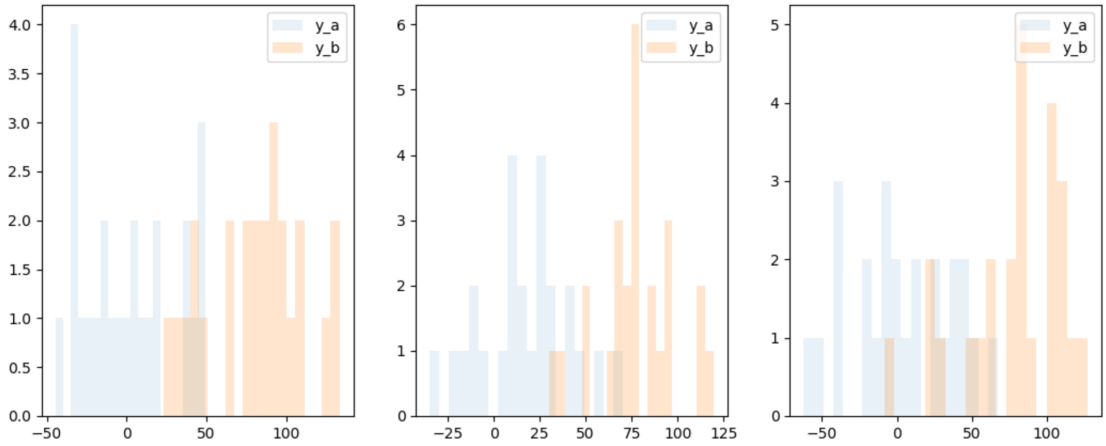


Figure 7: Histogram of y_c projections given direction ω_2 for matrices (a), (b), (c), respectively

In part(c) we were asked to define a directional weight vector ω_0 and apply the rotation matrix, $R(\theta) = \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix}$, to produce the vector for $\omega(\theta) = \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix} \cdot \omega_0$. Then we were asked to apply this to our training data, plot the Fisher score (calculation provided in coursework documentation) against the variation of θ and find the argmax of the Fisher score for, $argmaxF(\omega(\theta))$, which will allow us to determine the optimal directional vector[12] for projections $\omega^* = argmaxF(\omega(\theta))$. I implemented this by rotating through 100 equally spaced angles (around a 360Deg rotation). Figure 8 illustrates the Fisher score for the different angles rotation, θ, and outlined is the $w^* = (0.998, 1.001)$, where $\theta = 154Ded$ and $\omega_0 = (1, 1)$. The plot repeats every 180Deg. 180Deg translates to a flip in directions, which

[12]The reason I keep calling ω a directional vector is because we do not care for its magnitude, only its direction, hence the importance of θ

would not change the values of class projections, hence the same Fishers score is attained. Given I cycled through 360Deg, we end up with two values of θ which attain the the optimal Fishers score.

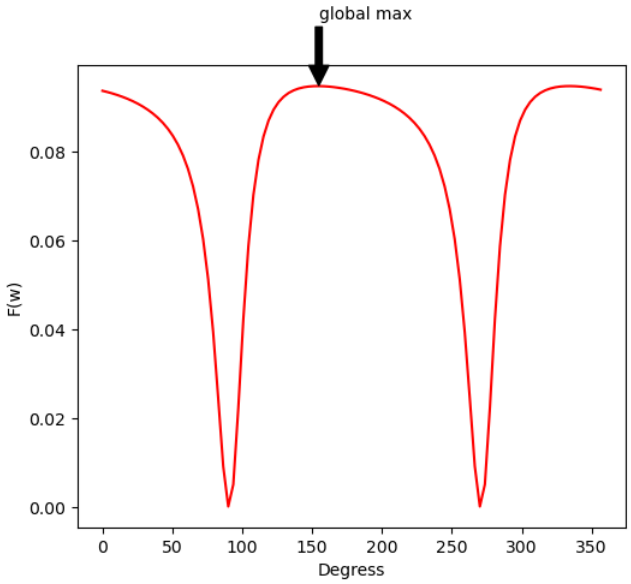


Figure 8: Measuring Fisher Score given rotations through θ, using $R(\theta)$

Part 2(a) asks us to derive estimates for the mean and covariances of the distributions we generated. The purpose of this is to emulate real-world application, where true mean and covariances are unknown. Thereafter, part 2(b) required us to use Baye's theorem to write down the log of the ratio of conditional probabilities (called log-odss) and compare decision boundaries of estimated parameters[13] and true parameters[14] for the conditions $\Sigma_a = \Sigma_b$ and $\Sigma_a \neq \Sigma_b$. My derivation of log-odds using Baye's theorem is depicted in Equation 8, and further calculation of log-odds dependant on covariances and means of classes is found in Equation 9. Notice that, since I determined the prior to be class-size dependant (hence, the class-size prior condition) and the class sizes are both the same the term is resolved as $\ln 1 = 0$

$$\ln \frac{P(c=a|x^n)}{P(c=b|x^n)} = \ln \frac{P(x^n|c=a)P(c=a)/P(x^n)}{P(x^n|c=b)P(c=b)/P(x^n)} = \ln \frac{P(x^n|c=a)}{P(x^n|c=b)} + \ln \frac{P(c=a)}{P(c=b)} \tag{8}$$

$$\ln \frac{P(x^n|c=a)}{P(x^n|c=b)} = \ln \frac{|\Lambda_a|}{|\Lambda_b|} + \frac{1}{2}(x-\mu_b)^T \Lambda_b (x-\mu_b) - \frac{1}{2}(x-\mu_a)^T \Lambda_a (x-\mu_a) \tag{9}$$

After having derived Equation 9, I had to implement two methods of calculating the decision boundary, based on our conditions $\Sigma_a = \Sigma_b$ and $\Sigma_a \neq \Sigma_b$. I began with $\Sigma_a = \Sigma_b$, which we already know produces a linear decision-boundary, as outlined in Equation 10.

[13]Estimated using the maximum likelihood esitmation method (MLE). If you are interested in the derivation of these estimates I implore you to visit: *https://stats.stackexchange.com/questions/351549/maximum-likelihood-estimators-multivariate-gaussian*

[14]The parameters we used to generate the distribution

I implemented this in a function I called *decision_bound_lin*, which returns y-values in the form $y = mx + c$, and can therefore be graphed along with the corresponding x-values. The plots for the MLE and true parameter decision boundaries are illustrated in Figure 9. There exists a difference between the two estimated decision boundary and true boundary - this behaviour is expected - though I would like to argue that the difference is minimal, give-or-take a couple data points, and the estimates would still work fine. Granted, as with my comments in section 2, I could have generated more data.

$$2(\mu_a - \mu_b)^T \Sigma^{-1} x + \mu_b^T \Lambda \mu_b - \mu_a^T \Sigma \mu_a \tag{10}$$

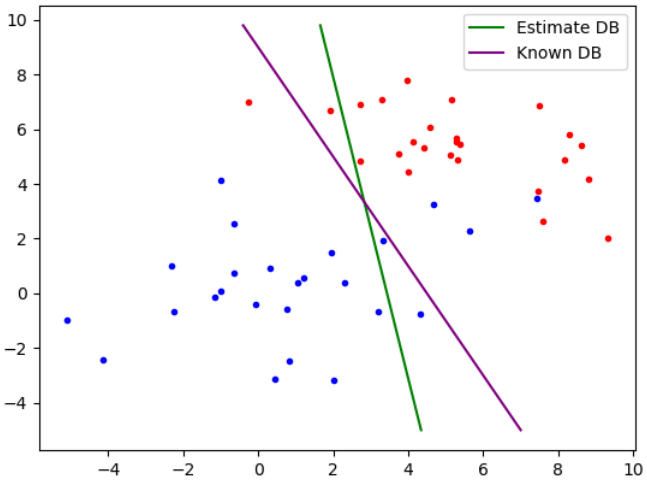


Figure 9: Decision boundary plot using linear log-odds equation

I then went onto the quadratic decision boundary implementation and as with the linear, I aimed to derive it mathematically. Unfortunately, as you know, a multi-variable polynomial cannot be solved for a unique solution unless one of the variables is know, which is not the case. Instead, I resorted to implementing the *mesh-grid* method; which creates a finite-element mesh ((i.e.) a grid of equally spaced coordinates), which I then applied to a function I called *LogOdds_grid* to. This function returned the coordinates where LogOdds appropriately approached zero[15], thus allowed me to plot Figure 10. From the illustration, again disparity is expected though, subjectively speaking, the boundaries are quite similar and I can confidently say the MLE parameters model the distributions well.

Further Comments:

I would like to close this section off by making further comments on estimating parameters. To enable reasonable comparisons to be drawn between true and estimated parameters I recruited two external estimation functions, *scikit-learn*'s *empirical_covariance* and *MinCovDet* functions. Tabulated in Table 1 and Table 2 are the determined estimates for class covariance and mean, respectively. My predictions and similarly those using the *empirical_covariance* function, produce relatively accurate results with resulting inaccuracies

[15] I am forced to implement the constraint "close-to-zero" and not "equal-to-zero" as the mesh-grid is finite is size. Therefore, the likelihood of actually getting a co-ordinate whose log-odds is zero is low, so we resort to plotting close-to-zero values

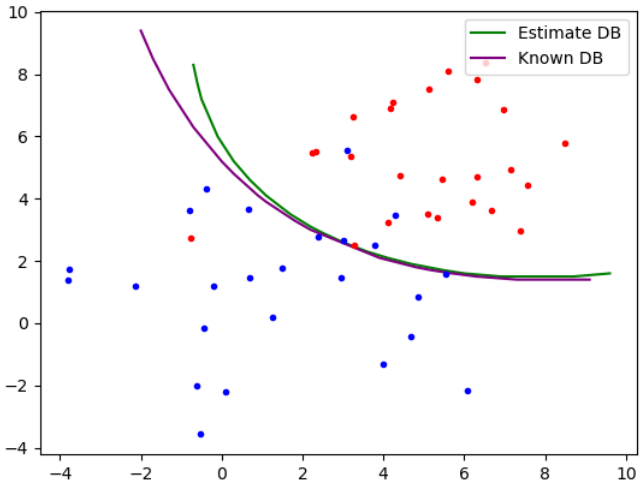


Figure 10: Decision boundary plot using quadratic log-odds function along with the mesh-grid method

ranging between 0.3 and 0.9 for Σ_c and 0.01 and 0.6 for μ_c, whereas for covariance the *MinCovDet* provides a much larger range of inaccuracy. In lieu of this and previous decision boundary plots, I can say that I have confidence in my estimations.

Covariance Estimates:	My Σ	empirical_covariance	MinCovDet
Result:	$\begin{pmatrix} 6.9 & 1.7 \\ 1.7 & 3.2 \end{pmatrix}$	$\begin{pmatrix} 6.9 & 1.7 \\ 1.7 & 3.2 \end{pmatrix}$	$\begin{pmatrix} 4.7 & 2.7 \\ 2.7 & 3.6 \end{pmatrix}$

Table 1: Estimations for Covariance (actual $\Sigma = \begin{pmatrix} 6 & 2 \\ 2 & 3 \end{pmatrix}, where \Sigma_a = \Sigma_b$)

Mean Estimates:	Class a	Class b
Result:	$\begin{pmatrix} 0.8 & 0.99 \end{pmatrix}$	$\begin{pmatrix} 3.6 & 2.9 \end{pmatrix}$

Table 2: Estimations for Mean (actual $\mu_a = \begin{pmatrix} 1 & 1 \end{pmatrix}$ and $\mu_b = \begin{pmatrix} 3 & 3 \end{pmatrix}$)

2.3 Data II: Classify Iris Data

This section is split into three parts, (1) solving an LDA eigenvalue problem in order to find the optimal direction vector ω^*, (2) displaying the 1-D projections on a histogram and (3) comparative analysis between Fisher'S LDA, soft-max (SM) and learned-weights soft-max (LWSM) classifications.

In part(1) we were required to extract data from the Iris data set, estimate the means and covariances of the classes and find the optimal directional vector, ω^*. I implemented the data-extraction algorithm in a separate file to the main body (for the aesthetic purpose of organised code), and passed the data (in dictionary format) to main-file[16], where the data was organised into the relevant classes, and estimates for means and covariances were outlined using MLE. Following this I needed to find a way of determining the optimal direction vector for projecting data. To accomplish this using Fisher's LDA, we have to calculate the partial derivative of the Fisher's score with respect to ω, set this to zero and solve for ω[17]. This transforms our problem into the eigenvalue problem described by Equation 11, where Σ_b, Σ_w are the between-class and within-class scatter matrices, respectively. Thus, my next step was to calculate the scatter-matrices. As the sum of training data for each class was equal I didn't have to scale the class dependencies, hence my use of Equations 12 and 13.

$$\Sigma_b \omega = \lambda \Sigma_w \omega \tag{11}$$

$$\Sigma_w = \sum_c \Sigma_c \tag{12}$$

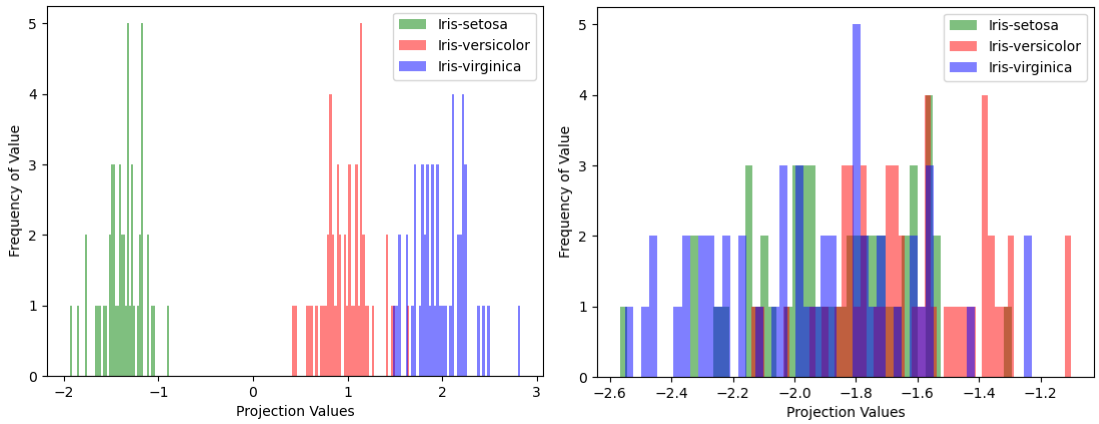
$$\Sigma_b = \sum_c \frac{n_c}{N} (\mu_c - \mu)(\mu_c - \mu)^T \tag{13}$$

Solving the eigenvalue problem brings me onto discussing the *Rank condition*, which states the number of linear discriminants (ergo, the number of eigenvalue solutions) is at most $c - 1$, constrained by c number of classes. This is dependant on Σ_b's rank, as Σ_b is the summation of the covariance matrices that pertain to each class, hence the rank of Σ_b is limited by the number of classes. In cases where perfect colienarity exists for Σ_b ((i.e.) rank 1) there exists only one solution for the eigenvalue problem (with a non-zero eigenvalue), though this is very rare in real-world application. In our case Σ_b holds rank 2, and we expect two solutions[18]. Since I have two eigenvalues, I need to determine which is most effective ((i.e.) which provides the least information loss when transforming 4-D data to 1-D projections). To accomplish this I calculated the explained variance as a percentage[19] and chose the eigenvalue with the highest percentage. This corresponds to 99.1% for an eigenvalue of 32.3 whose eigenvector is now the known optimal direction, ω^*. Figure 11 visualises the difference between each choice for eigenvector. It is evident that the eigenvalue of 32.3 produces the best projection given the superior class separation, refortifying my previous choice. This result reinforces the concept that the higher eigenvalues will produce the most ideal eigenvectors.

[16] The purpose of this is to simulate .json extraction from APIs

[17] You can visualise this using Figure8 where the optimal value exists at the peak of the curve, (i.e.) where the gradient is zero

[18] You'll discover within the code we actually receive four eigenvalues, though we can eliminate two as they hold a value of (close-to) 0. Within Python there exists floating-point imprecisions, so the values close to 0 are actually meant to be 0. We want to ignore zero-ed eigenvalue solutions. If your interested in understanding why we get solutions of zero, it can be explained using the rank-nullity theorem explained in: *https://math.stackexchange.com/questions/1349907/what-is-the-relation-between-rank-of-a-matrix-its-*



(a) Eigenvalue 32.3

(b) Eigenvalue 0.3

Figure 11: Histogram of Projections using given eigenvalues

In part(3) we are initially asked to perform SM regression on the same data set, which I did using the code that I wrote for Lab2a[20]. Following this, we were asked to find a way of representing data in a 2-Dimensional projection (no longer 1-D) using the learned weights. By applying $P(k = K|x = X) = e^{s_k(x)}/\sum_i e^{s_i(x)}$, where k represents a specific dimension in our projection (in this case k=1,2), to Equation 14 I was able to make calculations shown in Equation 15, where the first dimension ((i.e.) axis) is calculated using $\omega_1^T x - \omega_3^T x$ and the second is calculated using $\omega_2^T x - \omega_3^T x$, where ω_c corresponds to the set of learned weights from class c. Using both calculations we can then plot the 2-Dimensional graph.Additionally, given it is a SM classifier by nature we also get the added benefit of certainty classifications.

$$\ln \frac{P(k = 1|x = X)}{P(k = K|x = X)} = \beta_{10} + \beta_1^T x \tag{14}$$

$$\ln \frac{\frac{e^{s_x(1)}}{\sum_i e^{s_i(x)}}}{\frac{e^{s_k(x)}}{\sum_i e^{s_i(x)}}} = \ln \frac{e^{s_1(x)}}{e^{s_k(x)}} = s_1(x) - s_k(x) = \omega_1^T x - \omega_k^T x \tag{15}$$

As a result: Fisher's LDA enables us to optimise class separation using 1-D projections, (e.g.) for our Iris data in Figure 12(a); SM regression allows us to calculate certainties of a point pertaining to each classes, (e.g.) as done in Figure 12(b), where there exists 100% certainty of pertaining to class 1; and LWSM allows us to determine the uncertainties as well outlining 2-D projections based on the uncertainties, (e.g.) we can plot regions in Figure 12(c) for certainty of 50% or more[21].

To properly test these methods, I applied a test point to the methods. The outcomes are illustrated in Figure 12, where in (a) it is represented by a black bar present within the setosa class prediction, (b) it is the 100% certainty pertaining to class 1, the setosa class,

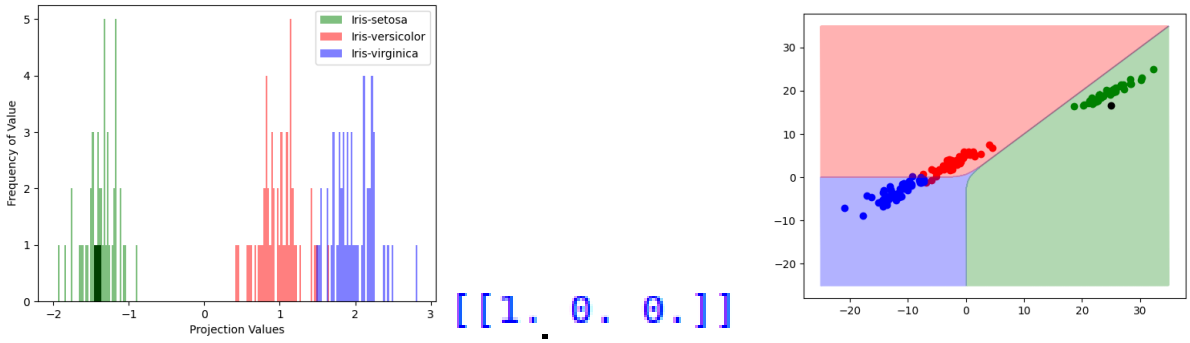
eigenvalues-and-eigenvectors

[19] An objective value used to determine how much information is transferred in 4-D to 1-D transformation

[20] A COMP3223 lab, performed earlier in the term

[21] I chose 50% as this is the threshold for confident classification

and (c) where it is represented by a black dot and a certainty of 99% of being part of the setosa class. They all agree it pertains to the setosa class, though there are some differences: Where 1-D projections are determined for LDA, while LWSM determines 2-D projections, which means more information is saved as a consequence of going from 4-D data point to 2-D projection; and clearly the improved modification of the SM classifier to provide the LWSM classifier.



(a) Fisher's LDA (b) SM Uncertainties (c) LWSM

Figure 12: Program outcomes for different classification techniques

Further Comments: In addition I wanted to draw one last comparison to the principle component analysis (PCA) method, which often follows LDA execution of dimensional reduction[22]. PCA is a common alternative to LDA, where LDA finds the axes (projection) that produce the largest class separation, PCA finds the axes that produce the largest vriance for the whole data-set. I used *scikit-learn*'s *PCA* function to fit the data and determined a set of 2-D projections shown in Figure 13. Again, our test point (black dot) falls within the setosa class.

My conclusion from these results is that LWSM classification provides the highest degree of information, so I in my opinion it is the superior classification method. Note that this statement is refortified by [1], who states, "It is generally felt that logistic regression is a safer, more robust bet than the LDA model, relying on fewer assumptions", (found in chapter 4.4.5), on the other hand [1] mentions that LDA estimates parameters more efficiently, which results in lower variance of results. Overall I stand by the choice of using LWSM.

[22]I.e. PCA provides 2-D projections, so reduced loss of information when calculating projections

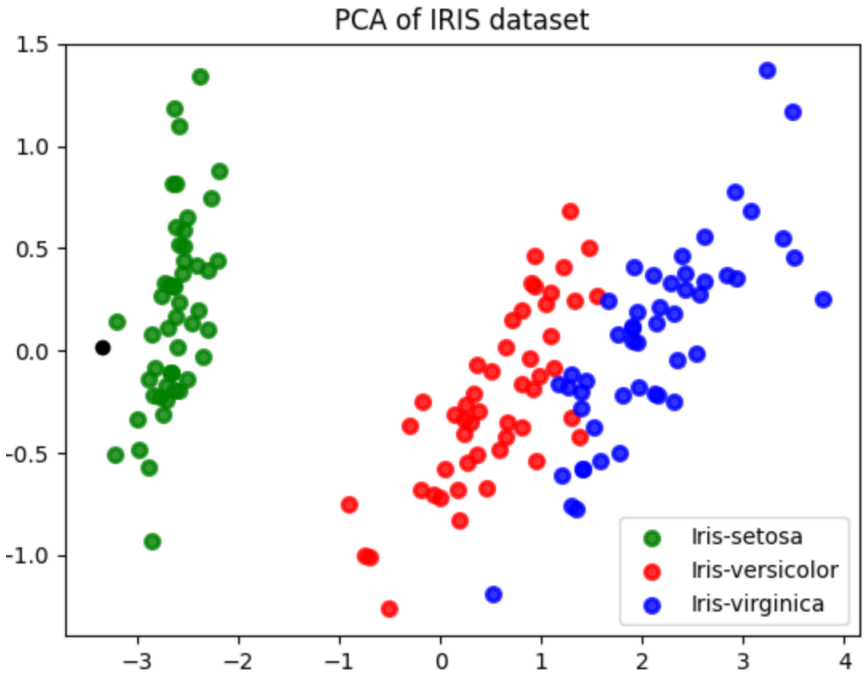


Figure 13: PCA 2-D projection plot

References

[1] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning.* Springer, Springer, New York, NY, 2009.