Machine Learning Coursework

Liam Browne

Autumn Term, 2023

Contents

1 Maximum-Margin Classifiers - Support Vector Machines - Kernel Methods

1.1 Question 1.a.i)

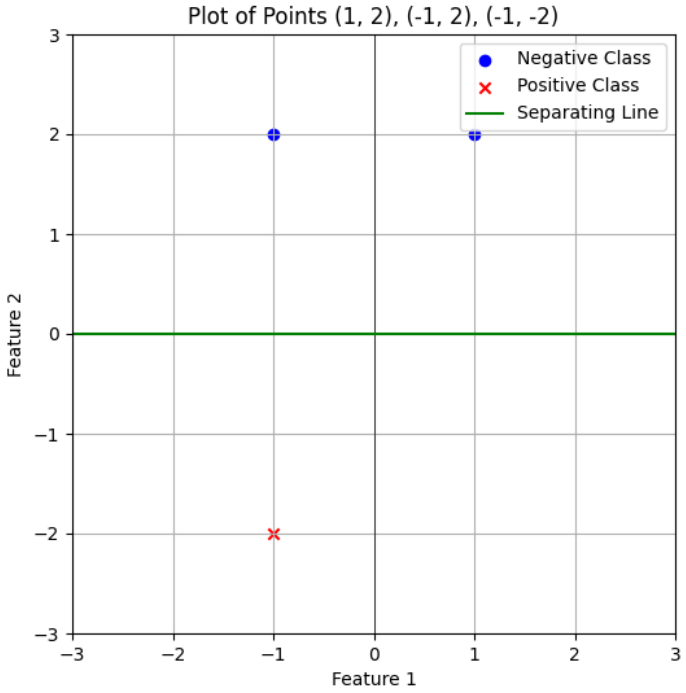

Figure 1.1.1: Plot displaying the linear separability of the negative and positive classes, given the 3 data points.

1.2 Question 1.a.ii)

By inspecting Figure 1.1.1, we can predict the support vectors to be:

$$x_2 = \begin{pmatrix} -1 \\ 2 \end{pmatrix}$$

$$x_3 = \begin{pmatrix} -1 \\ -2 \end{pmatrix}$$

1.3 Question 1.a.iii)

We consider the Lagrangian for a linear hard margin SVM:

$$L(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i [y_i(w \cdot x_i + b) - 1]$$

where w is the weights vector, y_i is the label of a certain point x_i, b is the bias and -1 comes from the condition.

We would like to minimise this equation. Hence, we find the first derivatives with respect to w and b, such that:

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{n} (\alpha_i y_i x_i)$$

$$\Rightarrow w = \sum_{i=1}^{n} (\alpha_i y_i x_i)$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{n} (\alpha_i y_i)$$

If we substitute these equations back into the original Lagrangian, we obtain an equation that is at the minimum with respect to w and b, such that we are left with an equation that is a function of α:

$$L(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j x_i^T x_j$$

Subject to $\alpha_i \geq 0$, $\sum_{i=1}^{n} \alpha_i y_i = 0$.

1.4 Question 1.a.iv)

Not attempted.

1.5 Question 1.b.i)

Given the function:

$$K_1(x, y) = \begin{cases} a, & \text{if } x = y \\ 0, & \text{otherwise} \end{cases}$$

We would like to show that it is a valid kernel by proving that it satisfies the Mercer conditions:

Symmetry:

$$x = y \Leftrightarrow y = x$$

Positive Semidefiniteness:

Let $x = x_i$, $y = x_j$ and note that K_1 will only have values, equal to a, along its diagonal, since for the rest of the matrix, $\forall$ is zero:

$$K_1(x_i, x_j) = \begin{pmatrix} a & 0 & \cdots & 0 \\ 0 & a & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a \end{pmatrix}$$

K_1 is positive semi-definite if $\alpha^T K_1 \alpha$ is positive, where α is an R^n dimensional vector, $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$, where $\alpha \in R$. Since K_1 is zero everywhere except on its diagonal, $\alpha^T K_1 \alpha$ becomes:

$$\Rightarrow \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j a$$

The above equation becomes:

$$\Rightarrow \sum_{i=1}^{n} \alpha_i^2 a$$

$\alpha_i^2 \geq 0 \ \forall \alpha_i$ and according to the question, $a \in R^+$, such that $\alpha^T K_1 \alpha$ is positive semidefinite.

1.6 Question 1.b.ii)

The kernel function $K_2(x, y)$ is defined as:

$$K_2(x, y) = x^T y$$

Symmetry:
The dot product is commutative, such that:

$$x^T y = y^T x$$

Positive Semidefiniteness:
K_2 is positive semidefinite if $\alpha^T K_2 \alpha$ is ≥ 0, where α is the same as in question 1.b.i). Furthermore, let $x = x_i$, $y = x_j$. We commence:

$$\alpha^T K_2 \alpha = \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j x_i^T x_j$$

$$\Rightarrow \left(\sum_{i=1}^{n} \alpha_i x_i^T \right) \left(\sum_{j=1}^{n} \alpha_j x_j \right)$$

$$\Rightarrow \left(\sum_{i=1}^{n} \alpha_i x_i \right)^T \left(\sum_{j=1}^{n} \alpha_j x_j \right)$$

Let $\sum_{i=1}^{n} \alpha_i x_i = v$, such that:

$$\Rightarrow v^T v$$

This is a vector's magnitude squared, which is always ≥ 0. Hence, $\alpha^T K_2 \alpha$ is ≥ 0, such that K_2 is positive semidefinite.

1.7 Question 1.b.iii)

The kernel function $K_3(x, y)$ is defined as:

$$K_3(x, y) = a x^T y + b$$

where $a, b \in R^+$.

Symmetry:

$$K_3(y, x) = ay^T x + b$$

Check if $K_3(y, x) = K_3(x, y)$:

$$K_3(y, x) = K_3(x, y)$$

$$\Rightarrow ay^T x + b = ax^T y + b$$
$$\Rightarrow y^T x = x^T y$$

The dot product is commutative, such that $y^T x = x^T y$. Hence, K_3 is symmetric.

Positive Semidefiniteness:

Let $x = x_i$, $y = x_j$, such that $K_3 = ax_i^T x_j + b$. K_3 is positive semidefinite if $\alpha^T K_3 \alpha$ is ≥ 0, where α is the same as in question 1.b.i). We begin with:

$$\alpha^T K_3 \alpha = \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j (a x_i^T x_j + b)$$

$$\Rightarrow \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j a x_i^T x_j + \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j b$$

$$\Rightarrow a \left(\sum_{i=1}^{n} \alpha_i x_i \right)^T \left(\sum_{j=1}^{n} \alpha_j x_j \right) + b \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j$$

Let $\sum_{i=1}^{n} \alpha_i x_j i = v$. Also, we note that since α is originally a column vector, $\sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j$ is, in fact, equal to $\sum_{i=1}^{n} \alpha_i^2$. Hence,

$$\Rightarrow av^T v + b \sum_{i=1}^{n} \alpha_i^2$$

We notice that $v^T v$ is a vector's magnitude squared, which is always ≥ 0. Hence, $\alpha^T K_3 \alpha$ is ≥ 0, such that K_3 is positive semidefinite.

2 k-Means Clustering, Principal Component Analysis

2.1 Question 2.a.i)

We have six data samples in R^2, such that:

$$X = \begin{pmatrix} -1 & -2 \\ 0 & -1 \\ 0 & 0 \\ 2 & 0 \\ 3 & 0 \\ 4 & 1 \end{pmatrix} = \begin{pmatrix} x_1^T \\ x_2^T \\ x_3^T \\ x_4^T \\ x_5^T \\ x_6^T \end{pmatrix}$$

Let A, B be the centres of the clusters, initialised to $(0, 0)$ and $(4, 1)$, respectively. The following process will consist of finding the distance squared of each x to A and B. We then select the x that are closest to A, relative to B, and the x that are closest to B relative to A. The new coordinates of A are then calculated using the mean of the points closest to it – similarly for B. Let D_A and D_B be 6×1 matrices that contain the distances from x_i to A or B. The matrix goes from x_1 to x_6 from top to bottom:

$$D_A = \begin{pmatrix} 5 \\ 1 \\ 0 \\ 4 \\ 9 \\ 17 \end{pmatrix}, \quad D_B = \begin{pmatrix} 34 \\ 20 \\ 17 \\ 5 \\ 2 \\ 0 \end{pmatrix}$$

Comparing each row of D_A to D_B, we find x_1, x_2, x_3 and x_4 to be closest to A and x_5, x_6 to be closest to B. Taking the mean of $\mathrm{x}_1 to x_4 and the mean of x_5 to x_6$, A and B become $(0.25, -0.75)$ and $(3.5, 0.5)$, respectively. We repeat the process, i.e., calculating the distances:

$$D_A = \begin{pmatrix} 3.125 \\ 0.125 \\ 0.625 \\ 3.625 \\ 8.125 \\ 17.125 \end{pmatrix}, \quad D_B = \begin{pmatrix} 26.5 \\ 14.5 \\ 12.5 \\ 2.5 \\ 0.5 \\ 0.5 \end{pmatrix}$$

Comparing rows, we find x_1, x_2, x_3 are closest to A and the rest of the points are closer to B. Calculating the respective means, we obtain $A = (-0.33, -1)$ and $B = (3, 0.33)$:

$$D_A = \begin{pmatrix} 1.45 \\ 0.12 \\ 1.11 \\ 6.43 \\ 12.09 \\ 22.75 \end{pmatrix}, \quad D_B = \begin{pmatrix} 21.42 \\ 10.76 \\ 9.11 \\ 1.11 \\ 0.11 \\ 1.45 \end{pmatrix}$$

x_1, x_2, x_3 are still closest to A and x_4, x_5, x_6 are closest to B. Since this is the same result as for the previous values of A and B, we conclude the algorithm here and report $A = (-0.33, -1)$ and $B = (3, 0.33)$ as the final cluster centres.

2.2 Question 2.a.ii)

We would like to find the solution to the optimisation problem:

$$\underset{\boldsymbol{y} \in R^d}{\operatorname{argmin}} \sum_{i=1}^{n} d(x_i, \boldsymbol{y})^2$$

where $d(\boldsymbol{x}, \boldsymbol{y})$ is the Euclidean distance, which is equal to:

$$d(\boldsymbol{x}, \boldsymbol{y}) = \left(\sum_{j=1}^{d} (x_j - y_j)^2 \right)^{\frac{1}{2}}$$

Hence,

$$\sum_{i=1}^{n} d(x_i, \boldsymbol{y})^2 = \sum_{i=1}^{n} \sum_{j=1}^{d} (x_{ij} - y_j)^2$$

Finding the derivative with respect to $\boldsymbol{y}$ and setting it equal to zero:

$$\frac{\partial}{\partial y} \left(\sum_{i=1}^{n} d(x_i, \boldsymbol{y})^2 \right) = -2 \sum_{i=1}^{n} \sum_{j=1}^{d} (x_{ij} - y_j)$$

$$\Rightarrow \sum_{i=1}^{n} \sum_{j=1}^{d} (x_{ij} - y_j) = 0$$

$$\Rightarrow \sum_{i=1}^{n} \sum_{j=1}^{d} x_{ij} = \sum_{i=1}^{n} \sum_{j=1}^{d} y_j$$

$$\Rightarrow \sum_{i=1}^{n} x_i = n \cdot \boldsymbol{y}$$

$$\Rightarrow \frac{1}{n} \sum_{i=1}^{n} x_i = \boldsymbol{y}$$

Hence,

$$\boldsymbol{y} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

2.3 Question 2.b.i)

$$X = \begin{pmatrix} 2 & 2 & 5 & 7 \\ 1 & 4 & 1 & 6 \end{pmatrix}$$

To find the principal components, i.e., the eigenvalues that encapsulate the most information of the matrix, we use:

$$\frac{1}{n} X X^T \boldsymbol{c}_1 = \alpha \boldsymbol{c}_1$$

Where X^T is the transpose of X, n is the number of data points, i.e. 4, $\boldsymbol{c}_1$ is an eigenvector of α and α is an eigenvalue. We begin with:

$$\frac{XX^T}{4} = \begin{pmatrix} 20.5 & 14.25 \\ 14.25 & 13.5 \end{pmatrix}$$

We omit the details of calculating the eigenvalues and eigenvectors as it is trivial. We obtain the eigenvalues in descending order:

$$\lambda_1 = 31.67, \quad \lambda_2 = 2.33$$

We choose λ_1 to calculate the eigenvector, since it has the largest value and therefore encapsulates the nature of the matrix X the most. Hence, the unit length eigenvector is:

$$\begin{pmatrix} 0.787 \\ 0.617 \end{pmatrix}$$

2.4 Question 2.b.ii)

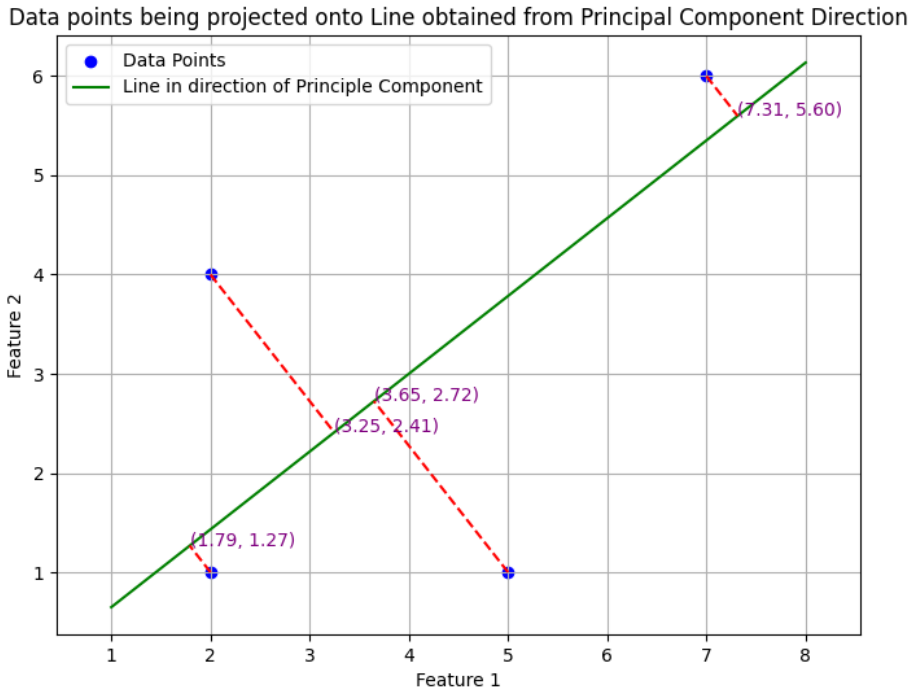

Figure 2.4.1: Plot displaying the coordinates of the projected points, from Question 2.b.i, in purple, on the line derived from the principle component's direction.

2.5 Question 2.b.iii)

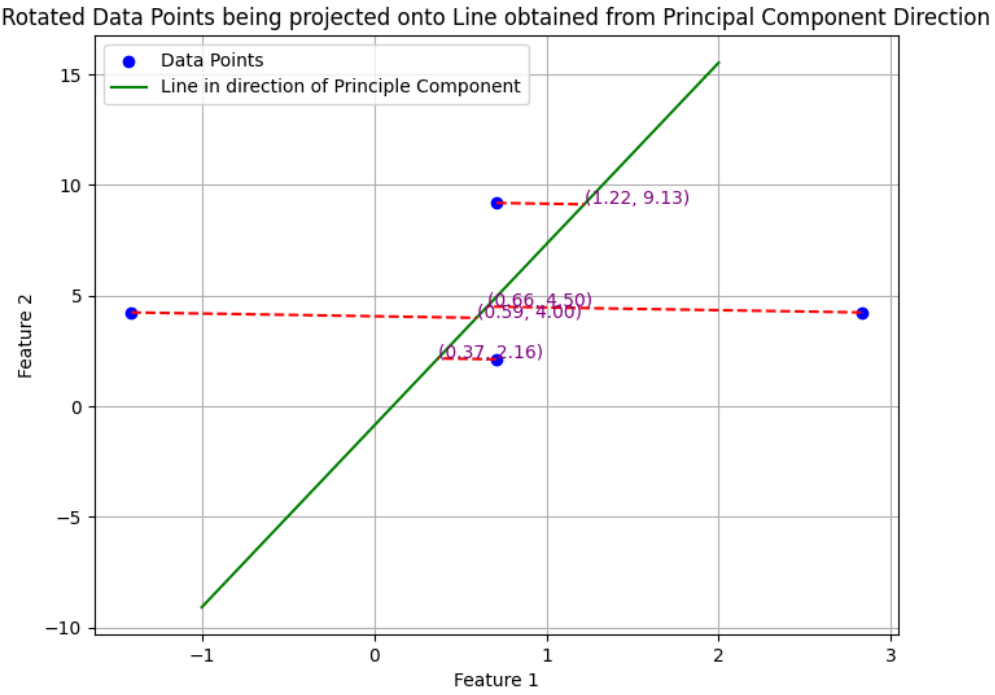

Figure 2.5.1: Plot displaying coordinates, in purple, of the projections of the counter clock-wise rotated points on the line derived from the principle component's direction.

The principal component's direction is $(-0.12, -0.99)$, while the projected points are $(0.32, 2.16)$, $(0.59, 4.00)$, $(0.66, 4.50)$, and $(1.22, 9.13)$.

3 Linear Regression

3.1 Question 3.a)

$$m = \frac{7-5}{2-1} = 2$$

$$b = \hat{y} - 2x$$

For $(1, 5)$:

$$\Rightarrow 5 - 2 = 3$$

Hence,

$$\hat{y} = mx + b = 2x + 3$$

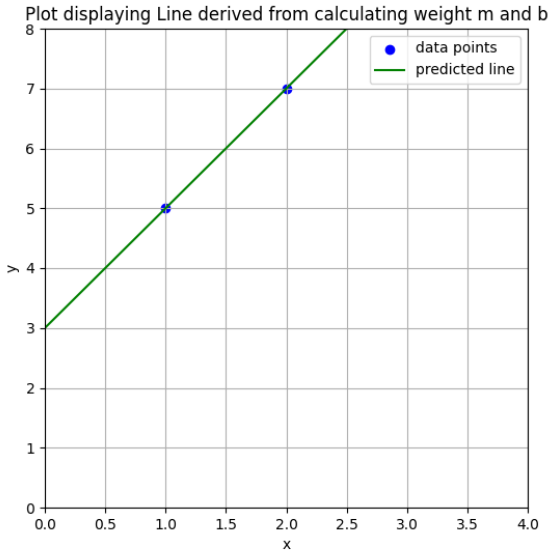

Figure 3.1.1: Display of line derived from calculating m and b.

3.2 Question 3.b)

Given the Mean Squared Error or MSE:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Gradient descent finds the derivative of the MSE with respect to w and b:

$$\frac{\partial \text{MSE}}{\partial w} = \frac{2}{n}\sum_{i=1}^{n} -x_i(y_i - \hat{y}_i)$$

$$\frac{\partial \text{MSE}}{\partial b} = \frac{2}{n}\sum_{i=1}^{n} -(y_i - \hat{y}_i)$$

Using these, we define the Update Rule as:

$$w := w - \alpha \cdot \frac{\partial \text{MSE}}{\partial w}$$

$$b := b - \alpha \cdot \frac{\partial \text{MSE}}{\partial b}$$

Where α is the learning rate. Given the points (1, 5) and (2, 7), as well as values to initialise m and b with, i.e., (9, 10), we can iteratively apply the update rule to ascertain final weights. We commence with $\alpha = 0.2$:

Iteration 1: $m = 9, b = 10, \text{MSE} = 1666$

$$\frac{\partial \text{MSE}}{\partial m} = 126$$

$$\frac{\partial \text{MSE}}{\partial b} = 70$$

Apply the Update Rule to obtain next weights:

$$m = -16.2, b = -4$$

Iteration 2: $m = -16.2, b = -4, \text{MSE} = 1259$

$$\frac{\partial \text{MSE}}{\partial m} = -112$$

$$\frac{\partial \text{MSE}}{\partial b} = -68.7$$

Apply the Update Rule to obtain final weights:

$$m = 6.2, b = 9.7, \text{MSE} = 174$$

3.3 Question 3.c)

A closed-form solution can be obtained using linear regression with the pseudo-inverse:

$$\boldsymbol{w} = (X^T X)^{-1} X^T \boldsymbol{y}$$

Note, the data points are (1, 5), (2, 7), where 1 and 2 correspond to the weight of feature_1; feature_0 is 1, since it is the bias. We define X and $\boldsymbol{y}$ as:

$$X = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}, \quad \boldsymbol{y} = \begin{pmatrix} 5 \\ 7 \end{pmatrix}$$

Computing $(X^T X)^{-1} X^T \boldsymbol{y}$:

$$\boldsymbol{w} = \begin{pmatrix} 3 \\ 2 \end{pmatrix} = \begin{pmatrix} w_0 \\ w_1 \end{pmatrix}$$

3.4 Question 3.d)

In ridge regression, i.e. L2 regularization, the loss function is given by:

$$\text{Loss function} = \sum_{i=1}^{n} (y_i - x_i^T \boldsymbol{w})^2 + \lambda \sum_{j=1}^{p} w_j^2$$

Minimizing the loss function yields:

$$\boldsymbol{w} = (X^T X + \lambda I)^{-1} X^T \boldsymbol{y}$$

For $\lambda = 1$ and using the data $x = (1, 1)$ and $(1, 2)$, $y = (5, 7)$, we obtain the solution as:

$$\boldsymbol{w} = \begin{pmatrix} 1.67 \\ 2.33 \end{pmatrix}$$

3.5 Question 3.e)

$$\phi(x) = (x, 3x - 1, 2x^2 + 2)$$

$$\Rightarrow \phi(1) = (1, 2, 4)$$

$$\Rightarrow \phi(2) = (2, 5, 10)$$

Model complexity should match the data resources. With only two data points, we should not use more complex models, especially $\phi(x)$, as it does not match our data resources. Hence, it could result in overfitting.

3.6 Question 3.f)

Not attempted.

4 Logistic Regression

4.1 Question 4.a)

Let t be the number of 'x's in an email, such that $P(t)$ is the probability that it is spam, and $P'(t) = 1 - P(t)$ is the probability that it is not spam.

Probability of email without 'x' to be a spam and not a spam email. We obtain the results graphically from the Figure in Question 4):

$$P(0) \approx 0.2$$

$$P'(0) \approx 0.8$$

4.2 Question 4.b)

After approximately 20 'x's, an email will be classified as spam with approximately 100% certainty.

4.3 Question 4.c)

Let $\hat{y}$ be a function that determines the binary classification, given certain probability thresholds:

$$\hat{y} = \begin{cases} 1, & \text{if } P(t) > 0.75 \\ 0, & \text{if } 0 \leq P(t) \leq 0.75 \end{cases}$$

4.4 Question 4.d)

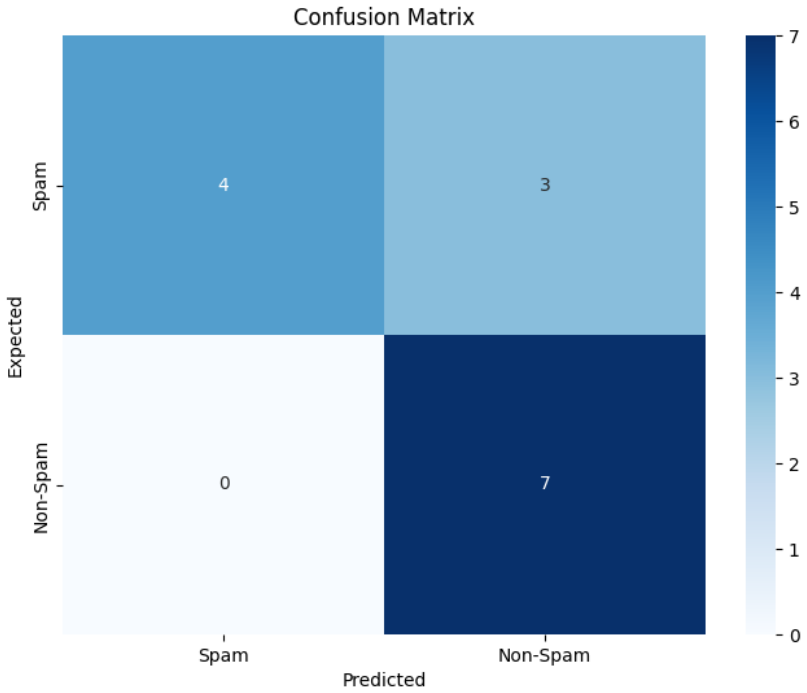

Figure 4.4.1: Confusion matrix for classifier defined in question 4.c).

Accuracy is given by the formula:

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{True Positive} + \text{True Negative} + \text{False Positive} + \text{False Negative}}$$

$$\Rightarrow \frac{11}{14} = 0.786$$

4.5 Question 4.e)

Precision is given by the formula:

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} = \frac{4}{4} = 1$$

Recall is given by the formula:

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} = \frac{4}{7} = 0.571$$

Precision measures how many spam emails were actually classified as spam emails. Since classifying non-spam emails as spam emails, i.e., a false positive, comes with a high penalty, we would like precision to remain high.

Recall measures how many spam emails were correctly labelled as spam. This does not have to equal 1 exactly, since seeing the occasional spam email is less detrimental than missing a non-spam email entirely. Hence, when minimising cost, we would like precision to remain as high as possible while we have some slack in terms of recall.

4.6 Question 4.f)

More effective parameters mean more degrees of freedom, implying higher complexity. A more complex model can fit the data better. However, if the amount of data we have does not increase as well, test error will increase, since we would need more data than we previously needed to generalise better. Hence, overfitting will occur without larger data resources. This is the equivalent of decreasing the bias, but increasing the variance, since we may have a better hypothesis in the hypothesis set, but we will not find it due to the larger variance and lack of data resources.

4.7 Question 4.g)

Not attempted.

5 Colab 1: Perceptron Learning Algorithm (PLA)

5.1 Task 1

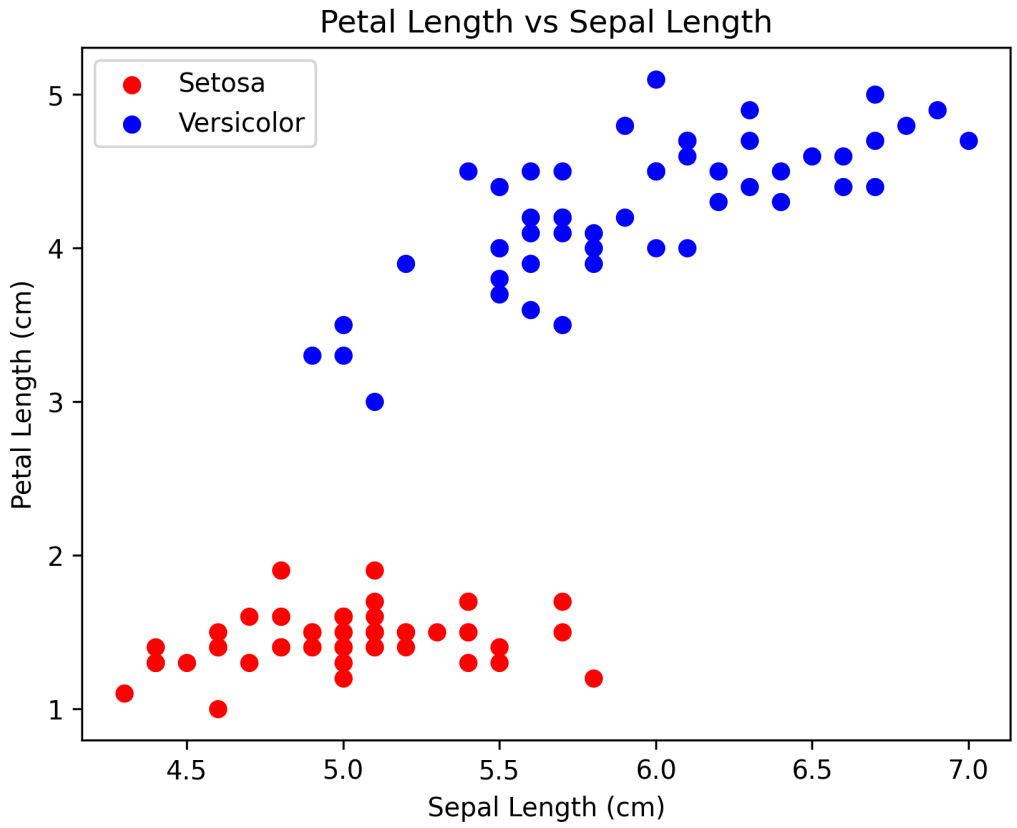

Figure 5.1.1: Plot of petal length vs sepal length demonstrating their linear separability.

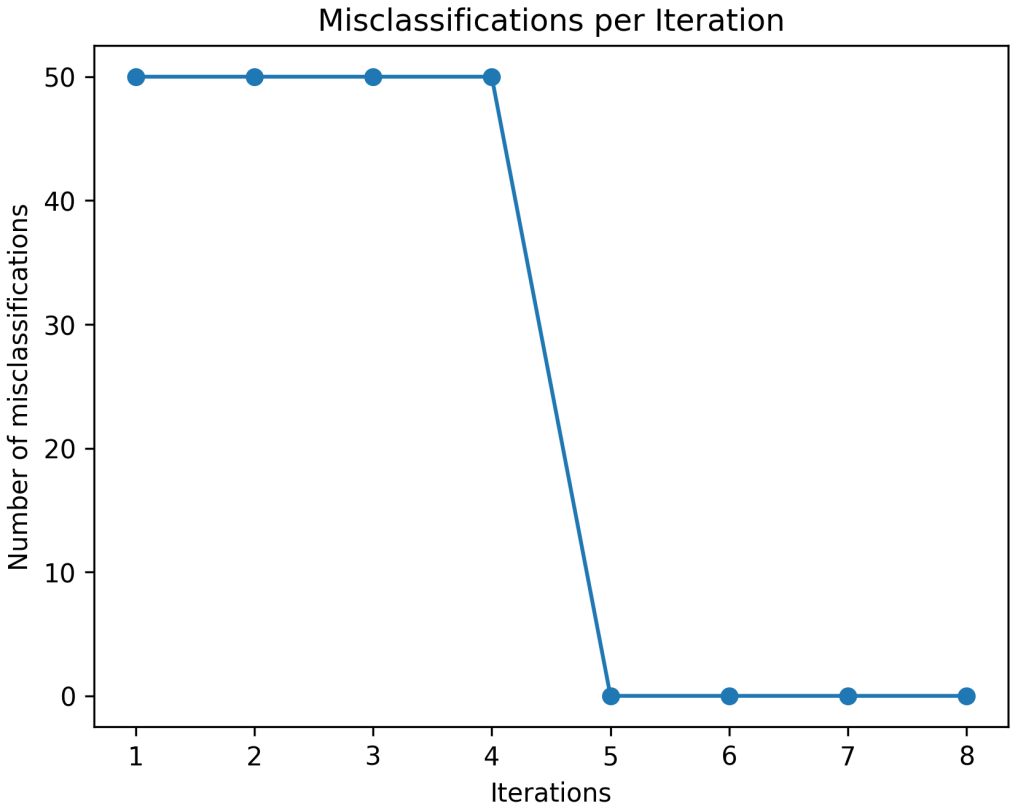

Figure 5.1.2: Figure displaying the change in number of misclassifications across iterations.

Final weights obtained:

$$\boldsymbol{W} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} -2 \\ -3.4 \\ 9.1 \end{pmatrix}$$

```
    X_setosa = iris_data.iloc[0:50, [0,2]]
    X_versicolor = iris_data.iloc[50:100, [0,2]]
    X = np.concatenate((X_setosa,X_versicolor))
```

Listing 1: Changing selected data from virginica to versicolor

Listing 1 displays how the code was adapted to suit the setosa vs versicolor classification by adjusting which data was selected from the Iris dataset. Furthermore, any names of virginica were replaced by versicolor throughout the code. Unlike for setosa and virginica, the petal vs sepal length data for setosa and versicolor are linearly separable, as seen in Figure 5.1.1.

5.2 Task 2

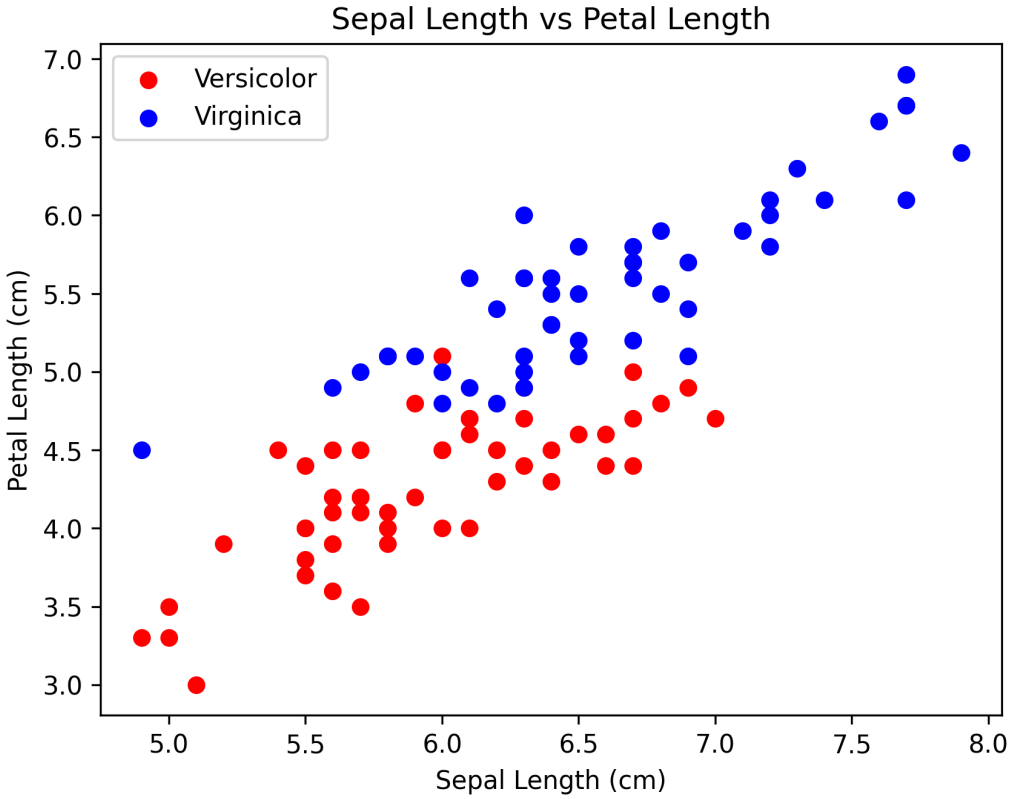

Figure 5.2.1: Plot of petal vs sepal length for the versicolor and virginica, demonstrating their lack of linear separability.

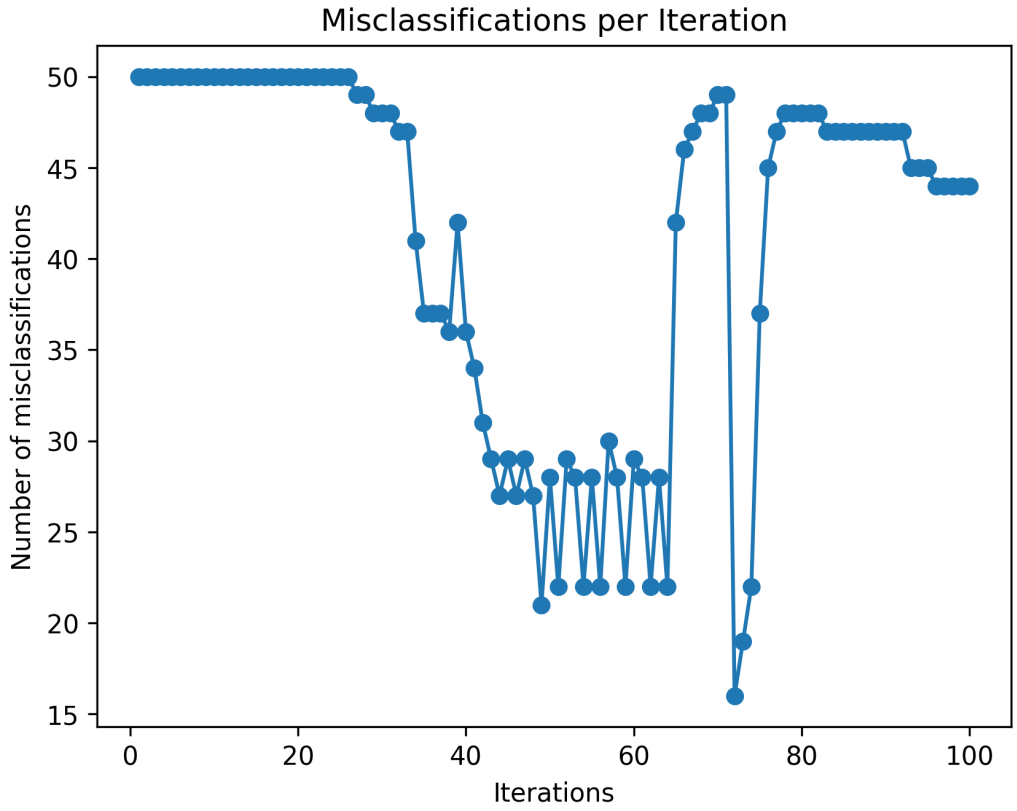

Figure 5.2.2: Plot displaying the evolution of misclassifications. Note, due to the lack of linear separability, the number of misclassified points fluctuate greatly across iterations.

Best-performing weights, obtained using the PLA Pocket Algorithm:

$$\boldsymbol{W} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 47.7 \\ -64.8 \end{pmatrix}$$

Choosing these weights as the final hypothesis results in 16 in-sample misclassifications.

```
    W_pocket = np.array([np.zeros(1 + X.shape[1]), np.full(1
   + X.shape[1], X.shape[0]/2)])
```

Listing 2: Initialises $\mathbf{W}_{\text{Pocket}}$, which stores the best-performing weights and their respective error.

Listing 2 generates a 2 by 3 matrix, where the first row intends to hold the best-performing weights across iterations, initialised to zero, and the second row holds the error, initialised to 50, i.e. the maximum number of misclassifications. Note, this makes two elements of the second row redundant, but was implemented to store the pocket weights and their respective error in the same array.

```
    if errors < W_pocket[1, 1]:
                W_pocket[0] = W
                W_pocket[1] = np.full(1 + X.shape[1],
                errors)
```

Listing 3: Checks if current weights should replace weights in $\mathbf{W}_{\text{Pocket}}$ by determining which of them has fewer misclassifications.

Due to the non-linear separability of the data, the number of misclassifications may increase or decrease across iterations, such that one cannot expect the final iteration to yield the best-performing weights as in the case of linearly separable data. Instead, the best-performing weights will occur at some point across the iterative evolution. Hence, listing 2 generates an array to store these weights and the error incurred with the best-performing weights that are to be stored. Furthermore, listing 3 tests if the weights of each consecutive iteration performs better than the best-performing weights until that point.

6 Colab 2: Linear Regression

6.1 Task 1

```
lr = 0.9
n_iter=10000
X = diabetes_data["bmi"]
Y = diabetes_data["DP"]
fit(n_iter,X,Y,lr)
X = diabetes_data["bp"]
Y = diabetes_data["DP"]
fit(n_iter,X,Y,lr)
```

Listing 4: Code for Task 1.

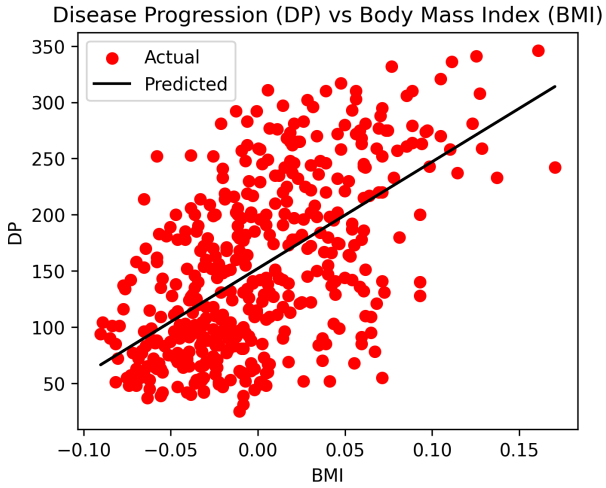

Figure 6.1.1: Disease Progression against Body Mass Index.

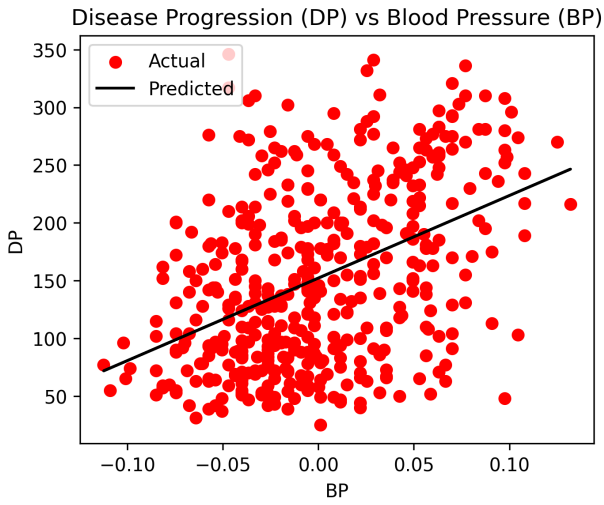

Figure 6.1.2: Disease Progression against Blood Pressure.

6.2 Task 2

```
def finding_W(X, Y):
    return np.dot(np.linalg.pinv(X), Y)

def closedFormFit(X,Y):
    # Incorporate the bias term
    X = np.column_stack((np.ones(len(X)),X))

    W = finding_W(X, Y)
    J = cost(X,W,Y)
    print(f"Cost = {J}, Weights = {W}")
    plot_line(X,W,Y)

                \centering
                \includegraphics[width=0.6\linewidth]{Figures
    /Lab2/TASK_1.2.png}
                \caption{Disease Progression against Blood
    Pressure.}
                \label{fig:myimage} % Label for referencing
    the figure in text
                \end{figure}
```

Listing 5: Closed-form linear regression using the pseudo-inverse.

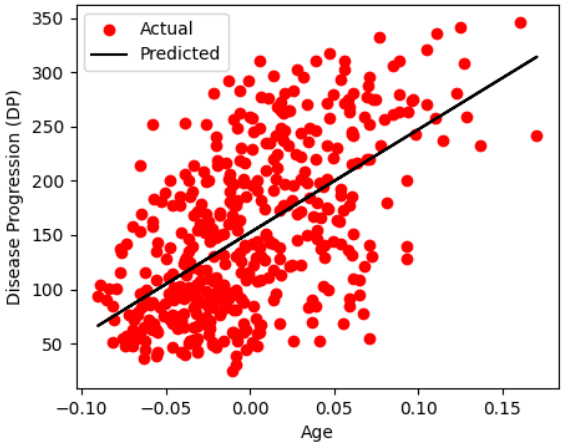

Figure 6.2.1: Disease Progression against Body Mass Index, using pseudo-inverse.

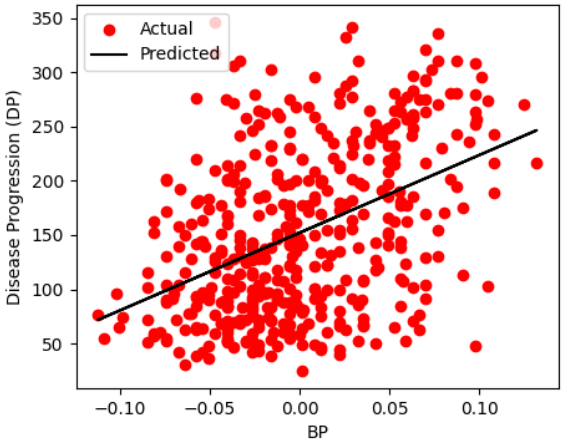

Figure 6.2.2: Disease Progression against Blood Pressure, using pseudo-inverse.

6.3 Task 3

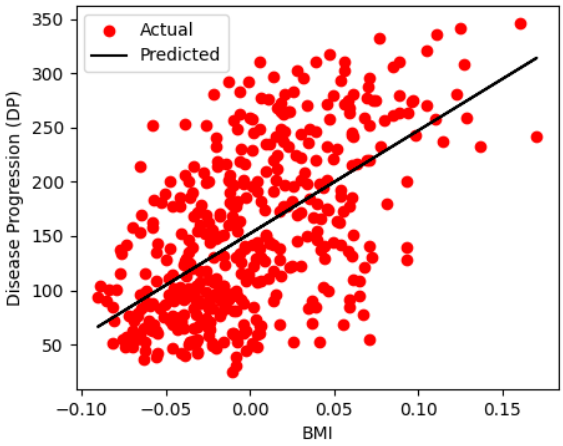

Figure 6.3.1: Disease Progression against Body Mass Index, using the Scikit Library.

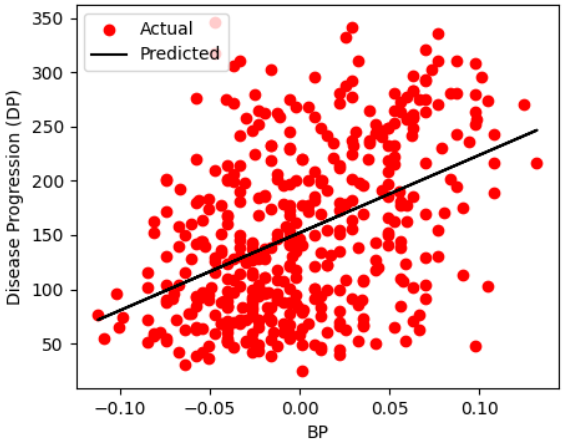

Figure 6.3.2: Disease Progression against Blood Pressure, using the Scikit Library.

	Cost	w_0	w_1
BMI_{GD}	3890	152	949
BP_{GD}	4774	152	714
BMI_{psuinv}	3890	152	945
BP_{psuinv}	4774	152	714
BMI_{Scikit}	3890	152	949
BP_{Scikit}	4779	152	714

Table 1: Table displaying the cost and final weights obtained from different linear regression methods. From top to bottom, gradient descent (GD), pseudo-inverse (pseuinv), Scikit Library (Scikit).

From table 1 we can conclude that all methods of linear regression achieve similar results. However, closed-form methods either by programming it oneself or using a library like Scikit, one step learning is faster than iterative learning, such as gradient descent.

6.4 Task 5

```
1
2  #Push data into matrices
3  Y = diabetes_data["DP"]
4  X = np.ones(len(diabetes_data["bp"]))
5  for i in diabetes.feature_names:
6      X = np.column_stack((X, diabetes_data[i]))
7
8  #Merge input/output data into single 442x12 matrix for
       reshuffling. Matrix is of form: w0..w10, y
9  XY = np.column_stack((X, Y))
10
11 #Shuffle rows
12 np.random.shuffle(XY)
13
14 #Separate input and output data
15 Y = XY[:, 11]
16 X = np.delete(XY, 11, 1)
17
18 #K is 2x10 matrix containing the indexing for cross-
       validation [0,44], [44,88] ... [396, 442]
19 K = np.empty((2, 10), dtype=int)
20 counter = 0
21 for i in range (0,10):
22     K[0, i] = counter
23     counter += 44
24     K[1, i] = counter
25 K[1,9] = 442
26
27 MSE_Array = []
28
29 #Apply linear regression to each fold
30 for i in range (0,10):
31     Xval = X[K[0,i]:K[1,i], :]
32     Yval = Y[K[0,i]:K[1,i]]
33     Xtrain = np.concatenate((X[:K[0,i]], X[K[1,i]:]), axis=0)
34     Ytrain = np.concatenate((Y[:K[0,i]], Y[K[1,i]:]), axis=0)
35     LR(Xval, Yval, Xtrain, Ytrain, MSE_Array)
36
37 MSE_AVG = np.sum(MSE_Array)/10
```

Listing 6: Code for linear regression cross-validation.

The average MSE was 3003.

7 Colab 3: Non-Linear Feature Transformation

7.1 Task 1

```
#Initialises non-linearly separable data
x1_transformed = np.empty((num_samples_per_class*2, 1))
x2_transformed = np.empty((num_samples_per_class*2, 1))

#Transform x1 and x2 into their quadratic counterparts
for i in range(0, len(x1)):
    x1_transformed[i] = (x1[i]**2)
    x2_transformed[i] = (x2[i]**2)

#Merges data into single matrix
X = np.concatenate((x1_transformed, x2_transformed), axis =
    1)

#PLA
pla(25, X, labels)
```

Listing 7: Transforms linear features into quadratic counterparts.

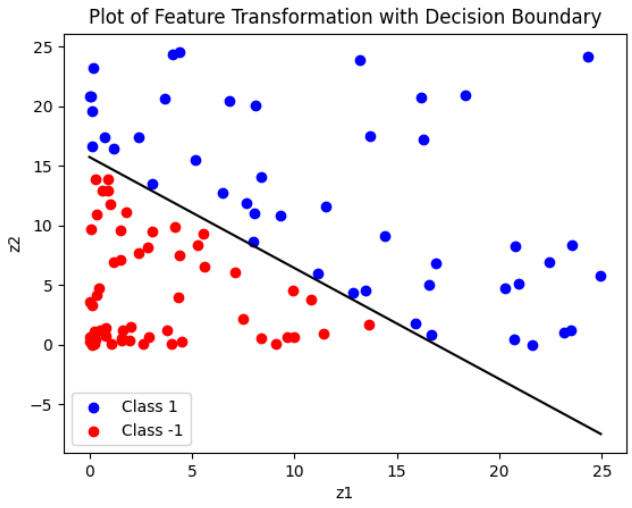

Figure 7.1.1: Plot of positive and negative class, showing linear separability of transformed data.

The final weights, i.e., those for which there were no more misclassifications, were achieved at iteration 16:

$$\boldsymbol{w} = \begin{pmatrix} -137 \\ 8.1 \\ 8.7 \end{pmatrix}$$

7.2 Task 2

```
def transform(x, d, include_bias):
    #Add column vector x to column of ones, i.e. the bias, if
    include_bias is toggled
    if include_bias:
      z = np.concatenate((np.ones((x.shape[0], 1)), x), axis
   =1)
    else:
      z = x

    #Transform the data x to z
    for i in range (2, d+1):
        z = np.concatenate((z,np.power(x, i)), axis=1)
    return z

def linearRegression(z, y):
    model = LinearRegression()
    model.fit(z, y)
    return model.coef_
```

Listing 8: Transforms x from z, i.e. from 1 dimension to d dimension, raising x to the power of 1 to d.

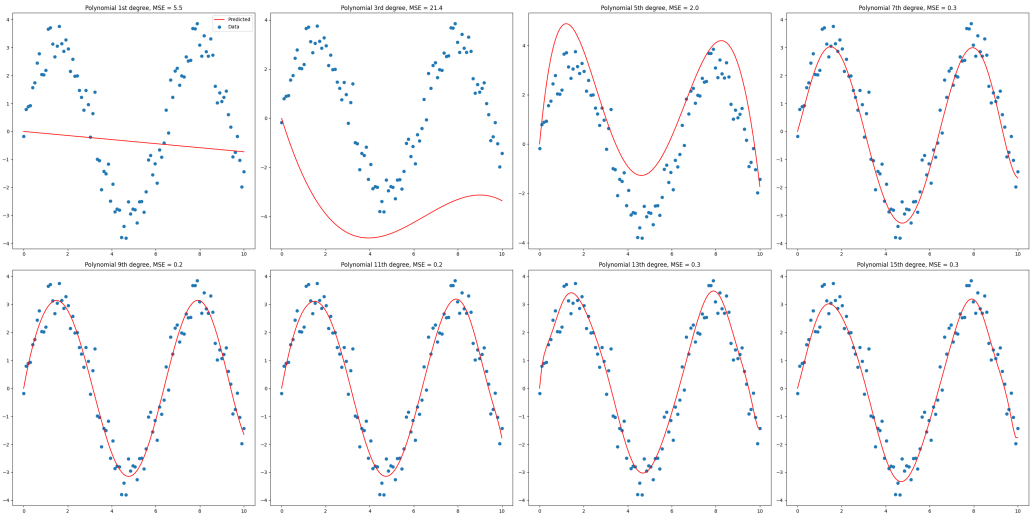

Figure 7.2.1: Plots displaying the effect of varying the model complexity on the MSE.

8 Colab 4: Logistic Regression

8.1 Task 1

The purpose of machine learning is to ascertain an in-sample estimate of the out-of-sample error and minimise this. When training a model on a certain data set, the chosen hypothesis will contain a deceptively optimistic bias for the data it was trained on. Hence, if the training data is used for testing, the out-of-sample error will be deceptively low. Any data used in the process of learning is said to become contaminated and will not yield an honest estimate of the out-of-sample error. Thus, we spare some data from learning, such that it is uncontaminated and unbiased during testing.

```
def shuffle(X, y):
    #Number of rows in X or y
    row_size = y.size
    #Generates array of size equal to number of rows,
   consisting of shuffled indices
    shuffled_indices = np.random.permutation(row_size)

    #Shuffle rows of X and y using shuffled indices,
   maintaining data integrity
    X_shuffled = X[shuffled_indices]
    y_shuffled = y[shuffled_indices]

    return X_shuffled, y_shuffled
```

Listing 9: Shuffles rows of a matrix.

```
def split_data(X, y):
    #Determine size of training and test data sets
    train_size = int(y.size*0.8)
    test_size = int(y.size*0.2)

    #Initilise empty matrices that will store training and
    test data
    #The column sizes are equal to those in X and y,
    respectively
    X_train = np.zeros((train_size, X.shape[1]))
    y_train = np.zeros((train_size))

    X_test = np.zeros((test_size, X.shape[1]))
    y_test = np.zeros((test_size))

    #Storing training and test data in their respective
    matrices,
    for i in range(0, train_size):
        X_train[i] = X[i]
        y_train[i] = y[i]

    for i in range(0, test_size):
        X_test[i] = X[train_size+i]
        y_test[i] = y[train_size+i]

    return X_train, y_train, X_test, y_test
```

Listing 10: Splits data into training and testing data.

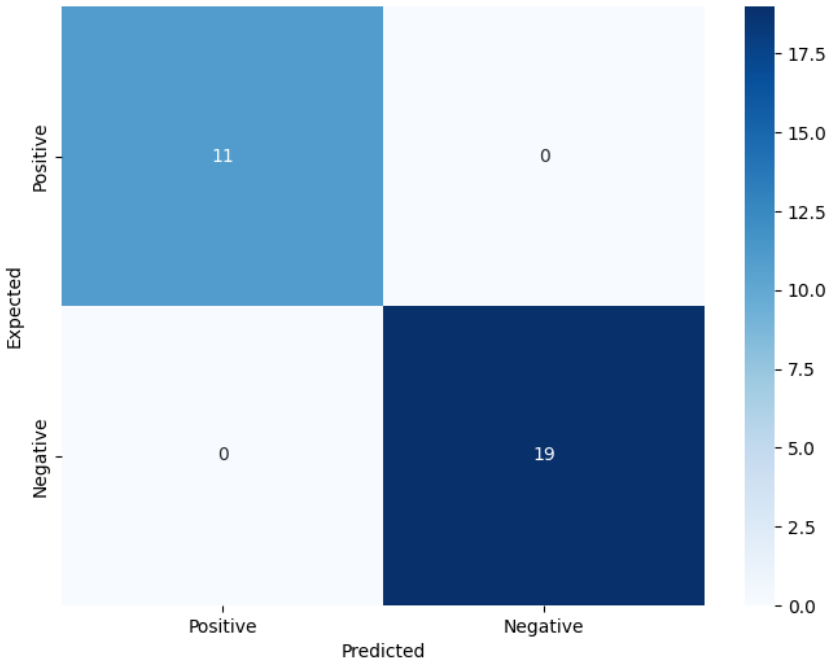

Figure 8.1.1: Confusion matrix for Setosa vs Others; accuracy = 1.

Logistic regression uses a soft threshold as opposed to the PLA hard threshold. In this case, Setosa vs Other is linearly separable, such that if a PLA can separate it perfectly, so will a soft threshold where the output is rounded to one class or the other.

8.2 Task 2

```
# Versicolor is assigned label 0, while all other classes
    will have a label of 1
y = np.where(labels == 1, 0, 1)
# Plot features in 2D plot - sepal length (x-axis) vs petal
    length (y-axis)
feat1, feat2 = 'sepal length (cm)', 'petal length (cm)'
X_target = iris_data[iris_data.target==1]
X_other = iris_data[iris_data.target!=1]
```

Listing 11: Code adjusted for Versicolor vs Others.

```
# Virginaca is assigned label 0, while all other classes will
    have a label of 1
y = np.where(labels == 2, 0, 1)
# Plot features in 2D plot - sepal length (x-axis) vs petal
    length (y-axis)
feat1, feat2 = 'sepal length (cm)', 'petal length (cm)'
X_target = iris_data[iris_data.target==2]
X_other = iris_data[iris_data.target!=2]
```

Listing 12: Code adjusted for Virginaca vs Others.

Final weights obtained for the separate cases:

$$\boldsymbol{w_{Versicolor}} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} -0.134 \\ 0.466 \\ -0.458 \end{pmatrix}$$

$$\boldsymbol{w_{Virginaca}} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 0.788 \\ 1.628 \\ -2.281 \end{pmatrix}$$

The weights for versicolor vs others has a 60% accuracy while those for virginica vs Others has a 90% accuracy. The data used for the confusion matrix is no longer a probability, but has been quantised using the .astype('int') method, i.e. the predicted output became binary classified. Hence, the analysis is virtually that for a binary classifier. The Iris-virginica vs Other model's accuracy is 90%; with the exception of a few points, it is close to linear separability. However, the Iris-versicolor vs Other has a model accuracy of 60%.

This is due to the Iris-versicolor's data being between two data clusters of the Other class. It is analogous to 3 points on a line and attempting to separate the middle point with a straight line – only 2 points, 66% of the data, will always be correctly classified.

8.3 Task 3

```
def plot_log_costs(loss_1, loss_2, loss_3, loss_4):
    #Function for plotting the costs vs iterations for
    different learning rates,


    eps = np.double(1e-15)


    #Generating x values, i.e. the log of iteration count
    x_axis_values = np.log(range(1, len(loss_1) + 1))


    #Plotting the log loss for different learning rates
    plt.plot(x_axis_values, np.log(loss_1+eps), label=f'   =
    0.0001, final cost = {"{:.{}e}".format(loss_1[-1], 2)}' ,
    color='red')
    plt.plot(x_axis_values, np.log(loss_2+eps), label=f'   =
    0.1, final cost = {"{:.{}e}".format(loss_2[-1], 2)}',
    color='blue')
    plt.plot(x_axis_values, np.log(loss_3+eps), label=f'   =
    1, final cost = {"{:.{}e}".format(loss_3[-1], 2)}', color=
    'green')
    plt.plot(x_axis_values, np.log(loss_4+eps), label=f'   =
    100, final cost = {"{:.{}e}".format(loss_4[-1], 2)}',
    color='purple')
    plt.xlabel('Log Iterations')
    plt.ylabel('Log Cost')
    # plt.savefig('plot.pdf')
    plt.legend(loc='lower right')
    plt.show()

def fit_adjusted(alpha, num_iter, X, y):
    #Identical to original fit function, but returns the loss
    list rather than plotting it


    # weights initialization
```

```
    W = np.zeros(X.shape[1])
    loss_list = []
    for i in range(num_iter):
        # Feed forward
        z = np.dot(X, W)
        Y = sigmoid(z)
        # Gradient
        W += alpha * (np.dot(X.T, (y-Y)) / y.size)
        loss = cost(Y, y)
        loss_list.append(loss)


    # This function will return the final weights as an array
    return W, loss_list
```

Listing 13: Functions to fit the data and plot the data on log scales.

```
num_iter = 3000

#Finding model weights and losses for different learning
    rates
alpha_1 = 0.0001
model_weights_1, loss_list_1 = fit_adjusted(alpha_1, num_iter
    , X_train, y_train)


alpha_2 = 0.1
model_weights_2, loss_list_2 = fit_adjusted(alpha_2, num_iter
    , X_train, y_train)


alpha_3 = 1
model_weights_3, loss_list_3 = fit_adjusted(alpha_3, num_iter
    , X_train, y_train)


alpha_4 = 100
model_weights_4, loss_list_4 = fit_adjusted(alpha_4, num_iter
    , X_train, y_train)


#Plots the costs on a logarithmic
plot_log_costs(loss_list_1, loss_list_2, loss_list_3,
    loss_list_4)

```

```
23
24 #Confusion matrix for each of the learning rates' model
      weights
25 confusion_matrix(X_test, y_test, model_weights_1)
26 confusion_matrix(X_test, y_test, model_weights_2)
27 confusion_matrix(X_test, y_test, model_weights_3)
28 confusion_matrix(X_test, y_test, model_weights_4)
```

Listing 14: Code used to execute the functions with the data.

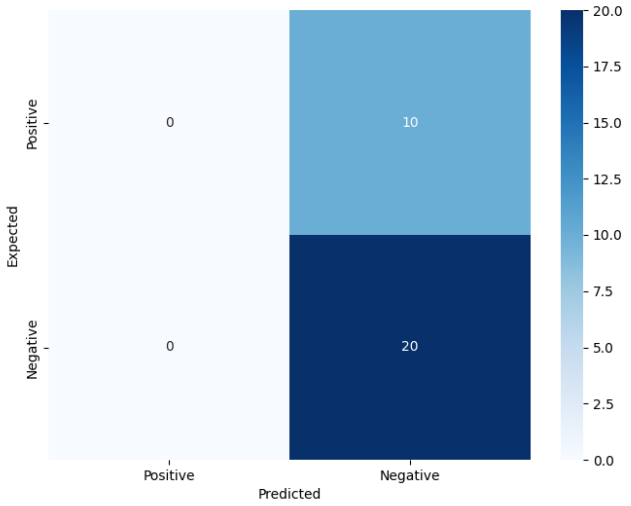

Figure 8.3.1: Confusion matrix for $\alpha = 0.0001$ accuracy is 66.67%.

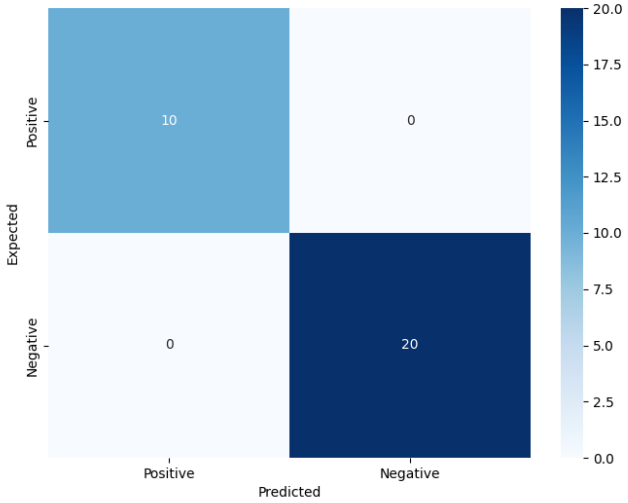

Figure 8.3.2: Confusion matrix for $\alpha = 0.1$, 1 and 1000, as they all have an identical confusion matrix. Accuracy is 100%.

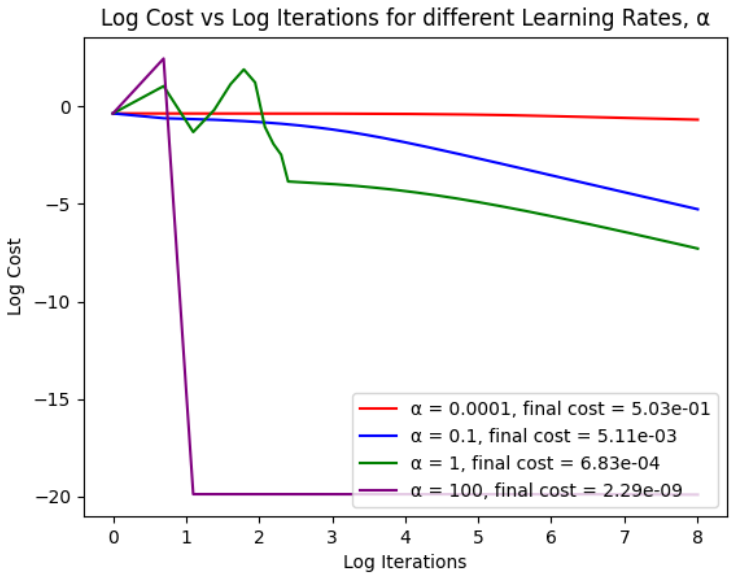

Figure 8.3.3: Log plot of Cost vs Iterations for different learning rates.

The learning rate, α, determines the step size in the direction of the minimum. If it is too small, the convergence speed is extremely low. This can be seen with α set to 0.0001 in Figure 8.3.3, where after 3000 iterations, the cost, or average error, is still at 50%. However, if α is too large, such as 1 or 100, the cost can actually increase, i.e. overshoot the local minimum, or even oscillate. This instability is undesirable, as we may oscillate without achieving a minimum effectively. In loss functions with multiple minima, one may even overshoot the optimal minimum.

9 Colab 5: Multi-Layer Perceptron

9.1 Task 1.A: Optimise L2 Regularisation

L2 Parameter	Training (%)	Validation(%)	Testing(%)	‖Training-Validation‖
10	34.02	34.26	34.08	0.24
1	88.74	87.92	89.33	0.82
0.1	95.49	95.32	95.45	0.17
0.01	98.46	97.18	97.40	1.28
0.001	98.75	97.24	97.46	1.51
0.0001	98.9	96.94	97.45	1.86

Table 2: Effect of L2 Parameter on Training, Validation and Testing Error.

L2 regularisation reduces the effects of overfitting by implementing a constraint on the squared sum of the weights, such that higher weights are penalised. Consequently, simpler hypotheses are favoured. A high L2 corresponds to a tighter constraint and vice versa. Hence, when L2 is 10, the weights are extremely constrained, such that we are underfitting the data – the best hypothesis is too simple. Decreasing L2 loosens the constraint, underfitting decreases, increasing overall accuracy. However, if we loosen it too much, we risk overfitting, i.e. we 'memorise' the training data, including stochastic noise. This results in worse generalisation, as shown by the increasing difference in validation and training accuracy. For these reasons, 0.1 was chosen as the optimal L2 parameter.

9.2 Task 1.B: Explore Activation Functions

Activation Function	Training (%)	Validation(%)	Testing(%)
Sigmoid	95.52	95.42	95.47
Tanh	71.16	70.80	70.88
ReLU	93.18	91.86	92.6

Table 3: Errors for different activation functions.

It was observed that the sigmoid activation function converges slowly, but with little generalisation error. The tanh activation function increases

rapidly with good generalisation, but then quickly decreases to a relatively low accuracy of 70.88%, as seen in Table 3. The ReLU activation function increases swiftly, yet its generalisation breaks down just as swiftly.

9.3 Task 2.A: Confusion Matrix

```
def compute_confusion_matrix(true, pred):

    #Number of classes
    num_cls = len(np.unique(true))
    result = np.zeros((num_cls, num_cls))

    for i in range(len(true)):
        result[true[i]][pred[i]] += 1

    return result
```

Listing 15: Code to calculate the confusion matrix.

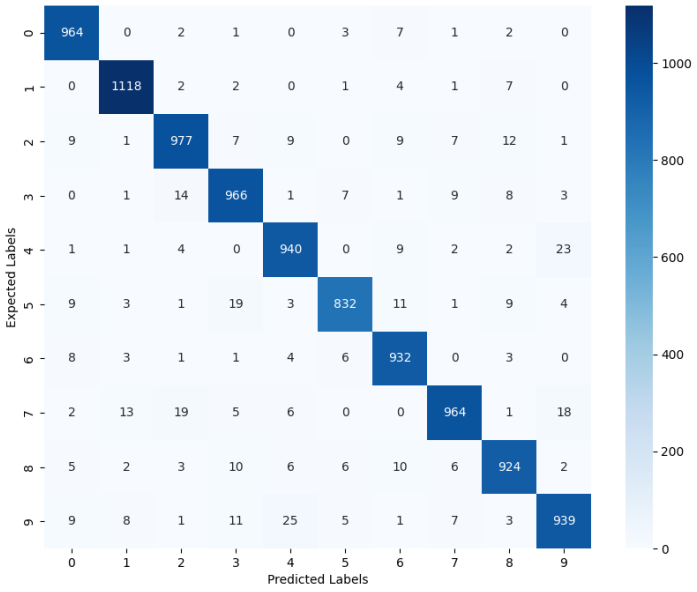

Figure 9.3.1: 10x10 Confusion Matrix for each Digit from 0 to 9.

9.4 Task B: Precision, Recall and F1-Score Calculation

```
def analysis(matrix):
    #Function to extract Accuracy, Precision, Recall and F1-Score, for each class, from confusion matrix


    #Number of classes
    num_cls = conf_matrix.shape[0]
    print("Class: ", "Accuracy: ", "Precision: ", "Recall: ", "F1-Score: " )

    #Compute for each class
    for i in range(0, num_cls):
        #Compute True Positive, False Positive, False Negative and True Negative
        TP = matrix[i, i]
        FP = np.sum(matrix[:, i]) - TP
        FN = np.sum(matrix[i, :]) - TP
        TN = matrix.sum() - FN - FP - TP

        Acc = (TN + TP)/(matrix.sum())
        Pre = TP/(TP+FP)
        Recall = TP/(TP+FN)
        F1 = (2*Pre*Recall)/(Pre+Recall)

        print(f"Digit {i}: ", round(Acc, 4), round(Pre, 4), round(Recall, 4), round(F1, 4))
```

Listing 16: Calculates accuracy, precision, recall and F1-score.

```
Class:  Accuracy:  Precision:  Recall:  F1-Score:
Digit 0:  0.9941 0.9573 0.9837 0.9703
Digit 1:  0.9951 0.9722 0.985 0.9786
Digit 2:  0.9898 0.9541 0.9467 0.9504
Digit 3:  0.99 0.9452 0.9564 0.9508
Digit 4:  0.9904 0.9457 0.9572 0.9514
Digit 5:  0.9912 0.9674 0.9327 0.9498
Digit 6:  0.9922 0.9472 0.9729 0.9598
Digit 7:  0.9902 0.9659 0.9377 0.9516
Digit 8:  0.9903 0.9516 0.9487 0.9501
Digit 9:  0.9879 0.9485 0.9306 0.9395
```

Listing 17: Output if Listing 11 is executed with sigmoid activated test data.

10 Colab 6: Support Vector Machines

10.1 Task 1

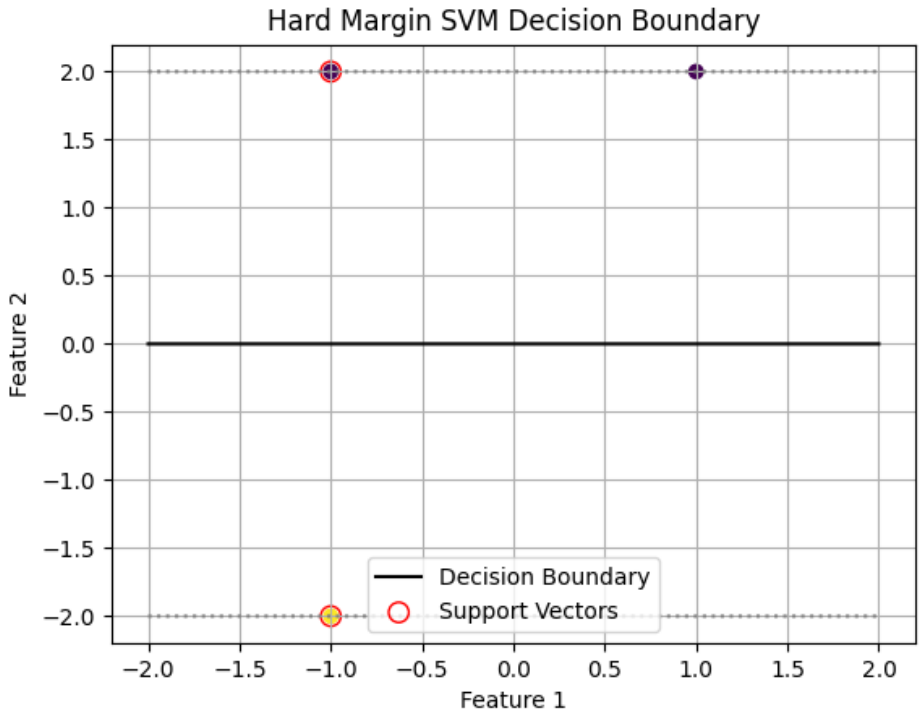

Figure 10.1.1: Plot of 3 points with lines illustrating the hard margin SVM's decision boundary and margins.

Optimal value of $\boldsymbol{w}$:

$$\boldsymbol{w} = \begin{pmatrix} 0 \\ -0.5 \end{pmatrix}$$

Optimal value of b:

$$b = 0$$

Non-zero $\boldsymbol{\alpha}$:

$$\boldsymbol{\alpha} = \begin{pmatrix} 0.125 \\ 0.125 \end{pmatrix}$$

Support vectors:

$$\begin{pmatrix} -1 \\ -2 \end{pmatrix}, \quad \begin{pmatrix} -1 \\ 2 \end{pmatrix}$$

The fact that the only non-zero feature weight is w_2 is unsurprising, since we can observe from Figure 10.1.1 that the main feature of what separates the points is their height, i.e. feature 2 – all points of the same class lie on the same horizontal line, hence why w_1 is 0.

10.2 Task 2

```
def soft_margin(X, y, C, L):
    m, n = X.shape
    # Define the SVM optimization problem
    zeta = cp.Variable(m , value=np.ones(m)) #Slack
    W = cp.Variable(n, value=np.zeros(n))  # Weights
    b = cp.Variable(value=0)               # Bias
    objective = cp.Minimize(0.5 * cp.square(cp.norm(W)) + C*cp
    .sum(zeta))
    constraints = [y[i] * (X[i] @ W + b)  >= 1 - zeta[i] for i
     in range(m)]  # Margin constraints
    constraints += [zeta[i] >= 0 for i in range(m)]
```

Listing 18: Code adjusted for the soft margin SVM. Note, this does not contain the rest of the code in the cell that was kept the same.

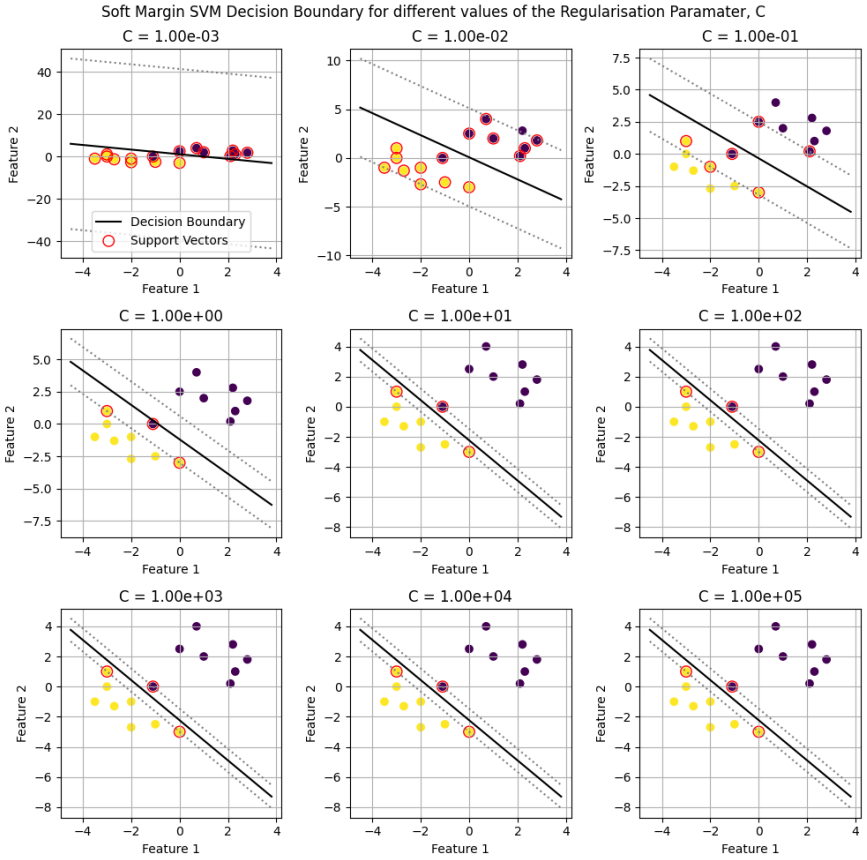

Figure 10.2.1: Soft Margin SVM Decision Boundaries for different values of the Regularisation Parameter C.

Observations and Analysis: Figure 10.2.1 illustrates how increasing the regularisation parameter's value, C, decreases the margins' width and therefore the number of support vectors. Conversely, lower values support a 'softer', wider margin, allowing for more misclassifications. This can mitigate overfitting, as a simpler model is being used that might generalise better than a more complex model, i.e. a higher C value. However, if C is too low, underfitting can occur, as seen for C=1e-3, where the margins become redundant as they capture all the training data. If C is too large, the soft margin becomes a hard margin, such that increasing C further is superfluous, as in Figure 10.2.1 where all graphs for C¿=10 are virtually identical. In

these cases, there are no points within the margins, which is a key difference between soft and hard margin SVMs.

Conclusion: When determining the optimal value for C, it is important to remember that the soft margin allows for some misclassifications, since they may be noisy. In Figure 10.2.1, the purple point closest to the cluster of yellow points seems to be noisy, as it is furthest away from the purple cluster of points. Hence, any value of C that puts this noisy point exactly on the margin may result in poor generalisation. C=1 seems most optimal, since although there are points being misclassified, the decision boundary separates the clusters where the purple and yellow points are most concentrated best. Regardless, further testing is required, for instance using cross validation, to be certain which value of C generalises best.

11 Colab 7: k-Nearest Neighbour

11.1 Task 1

```python
def manhattan_distance(p1, p2):
    m_distance = 0.0
    for i in range(len(p1)):
        m_distance += abs(p1[i] - p2[i])
    return m_distance


def minkowski_distance(p1, p2, p):
    sum_ = 0.0
    for i in range(len(p1)):
        sum_ += abs(p1[i] - p2[i])**p
    return sum_**(1/p)
```

Listing 19: Code implementing manhattan and minkowski distance. Note, one could only implement minkowski distance, as manhattan distance is a special case of minkowski distance.

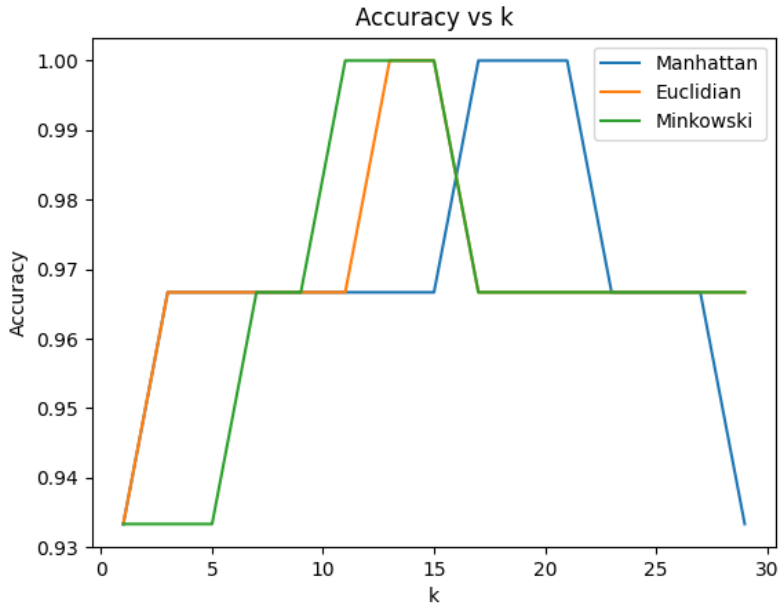

Figure 11.1.1: Accuracy vs number of nearest neighbours considered, k.

When using different methods to measure distance in K-NN, one is changing the interpretation of which points are closest to the unknown point being tested for. Hence, the optimal measurement for distance will depend on the data's characteristics. For the Iris data, all distance measurements yield full accuracy for the tested data at some value k, illustrated in Figure 11.1.1. However, since for each input all training examples are iterated over, this process can become computationally expensive for larger datasets and more so for larger k. Thus, the Minkowski distance measurement is most optimal for the given Iris data, as it achieves perfect accuracy with the smallest k.

11.2 Task 2

```
from sklearn.neighbors import KNeighborsClassifier


accuracy_list_1 = []
accuracy_list_2 = []
accuracy_list_3 = []
k_list = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27,
    29]


#K-NN using Manhattan Distance
for k in k_list:
    knn_classifier = KNeighborsClassifier(k,p=1)
    y_pred = []
    knn_classifier.fit(X_train, y_train)
    y_pred = knn_classifier.predict(X_test)
    acc = accuracy(y_pred, y_test)
    print("For k = {}, Accuracy {} %".format(k,acc))
    accuracy_list_1.append(acc)


#K-NN using Euclidian Distance
for k in k_list:
    knn_classifier = KNeighborsClassifier(k, p=2)
    y_pred = []
    knn_classifier.fit(X_train, y_train)
    y_pred = knn_classifier.predict(X_test)
    acc = accuracy(y_pred, y_test)
    print("For k = {}, Accuracy {} %".format(k,acc))
    accuracy_list_2.append(acc)

```

```
31
32 #K-NN using Minkowski Distance with p=3
33 for k in k_list:
34    knn_classifier = KNeighborsClassifier(k, p=3)
35    y_pred = []
36    knn_classifier.fit(X_train, y_train)
37    y_pred = knn_classifier.predict(X_test)
38    acc = accuracy(y_pred, y_test)
39    print("For k = {}, Accuracy {} %".format(k,acc))
40    accuracy_list_3.append(acc)
41
42
43 plt.plot(k_list, accuracy_list_1, label="Manhattan")
44 plt.plot(k_list, accuracy_list_2, label="Euclidian")
45 plt.plot(k_list, accuracy_list_3, label="Minkowski")
46
47
48 plt.xlabel('k')
49 plt.ylabel('Accuracy')
50 plt.title('Accuracy vs k, using Scikit-Learn')
51 plt.legend()
52 plt.savefig('Accuracyvsk.png')
53
54
55 plt.show()
```

Listing 20: Implementing k-NN using the Sci-kit library.

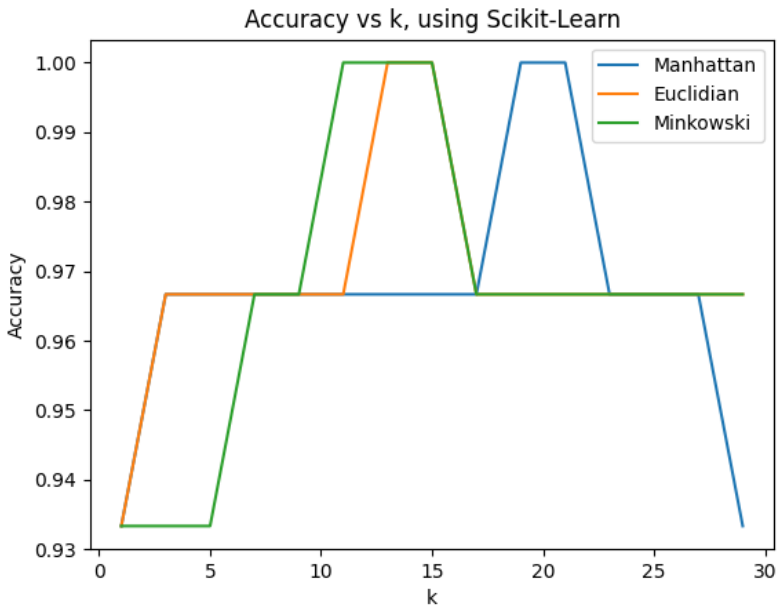

Figure 11.2.1: Accuracy vs number of nearest neighbours considered, k, using the Scikit Library.

Figure 11.2.1 is identical to Figure 11.1.1, except for the Scikit-Learn Manhattan distance accuracy being less than 100% for fewer k than in the case for Figure 11.1.1, obtained using the code for Task 1. This could be due to differences in the way the Scikit-Learn library processes its data or due to its Manhattan distance implementation being optimised for certain cases.

12 Colab 8: Clustering

12.1 Task 1

```
import statistics

def correspond_labels(labels):
   #Ensures the Iris data and K-means labelling correspond
   for all x in X


   #Finds mode of each section and maps this as the
   equivalent 0, 1 or 2 in the Iris data labels
   iris_zero = statistics.mode(labels[0:50])
   iris_one = statistics.mode(labels[50:100])
   iris_two = statistics.mode(labels[100:150])


   true_labels = np.zeros_like(labels)


   #Corresponds the K-means labelling with that of the Iris
   data
   #Example; Iris data's 0 could be a 2 in the K-means
   labelling, this converts all 2s in the K-means labelling
   to a 0
   for i in range(0, labels.shape[0]):
       if labels[i] == iris_zero:
           true_labels[i] = 0
       elif labels[i] == iris_one:
           true_labels[i] = 1
       else:
           true_labels[i] = 2


   return true_labels
```

Listing 21: Code that assigns the clusters obtained from K-means correct labels that correspond with those of the Iris data.

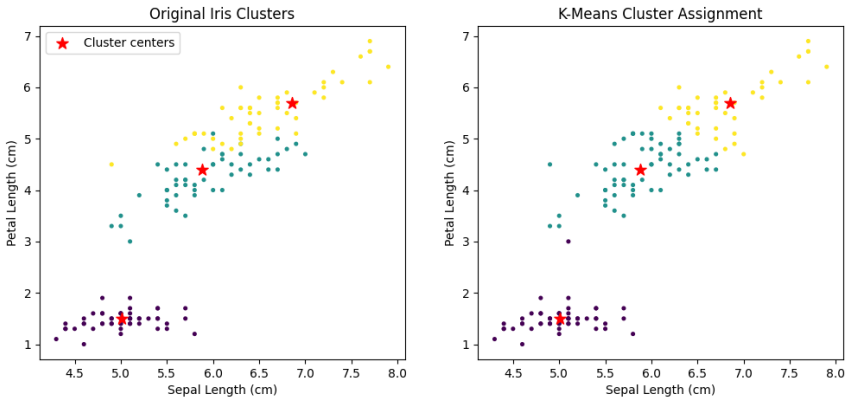

Figure 12.1.1: Comparison of the original iris labelling with that obtained by the K-Means assignment.

The total number of misclassifications in the K-Means assignment was 19. Comparing the two graphs in Figure 12.1.1, one will notice how the K-Means clusters do not have their points in other clusters, while the original data does – there is a clear divisioning. It is these points that will most-likely end up as misclassified.

12.2 Task 2

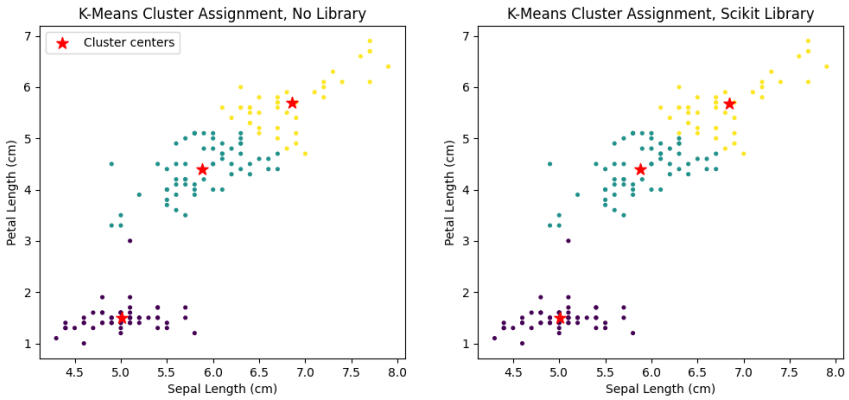

Figure 12.2.1: Comparison of the previously-obtained K-means assigment with that obtained using the Scikit library.

The assignments obtained by the Scikit library had 18 misclassifications, as opposed to the 19 we had in Task 1. The one point that was correctly classified in the Scikit version, relative to the assignment obtained with out code, can be seen on the border between the yellow and green cluster when comparing the points on the outside of the clusters in the left and right plots of Figure 12.2.1. As this is a borderline case, it may be due to the way the Scikit code rounds its numbers or the number of iterations it runs for.

12.3 Task 3

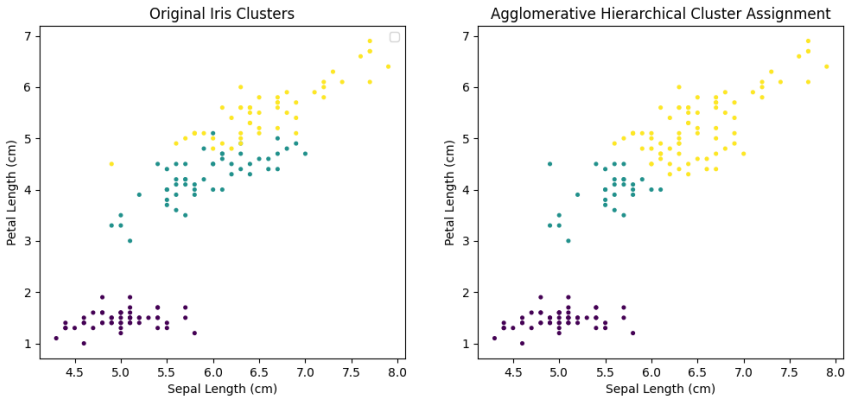

Figure 12.3.1: Comparison of the original iris labelling with that obtained by the Agglomerative Hierarchical Clustering.

The agglomerative heirarchical clustering had 24 total misclassifications.