VIETNAM NATIONAL UNIVERSITY

VIETNAM JAPAN UNIVERSITY

_____

MID-TERM REPORT

SUBJECT: ARTIFICIAL INTELLIGENCE

Instructor : Dr. Bùi Huy Kiên

Dr. Akihiro Kishimoto

Student: Bùi Thế Trung

Phạm Quang Anh

Lê Hải Nam

Phạm Minh Tuấn

Nguyễn Duy Tùng

Hanoi, 12/2023

Information

Topic name:

Explore various projects suitable for applying the following four machine learning algorithms:

      I.   Linear Regression

     II.   K-means Clustering (KNN)

   III.   Logistic Regression

   IV.   Support Vector Machines (SVM)

Implementation time: December 8, 2023 - December 22, 2023

Project leader: Bui The Trung

Phone: 0373.104.304   Email: 21110108@st.vju.ac.vn

Program: Bachelor of Computer Science and Engineering

LIST OF MEMBERS PARTICIPATING IN IMPLEMENTING THE PROJECT

NO	Name	Work content
1	**Bùi Thế Trung**	Code analysis SVM, KNN. Summarize the report
2	**Phạm Quang Anh**	Mathematical analysis SVM, write report
3	**Lê Hải Nam**	Implementing Linear Regression
4	**Phạm Minh Tuấn**	Implementing Logistic Regression
5	**Nguyễn Duy Tùng**	Mathematical analysis KNN, write report

Table of contents

List of tables

List of shapes

Introduction

In the increasingly complex world of data, the application of machine learning algorithms to solve real-world problems has become essential. Faced with this need, our team has decided to focus on exploring and applying four leading machine learning algorithms.

We will approach these problems in two ways. Firstly, we will implement the algorithms using mathematical analysis and defining the related functions. Secondly, we will use the Scikit-learn library to apply these algorithms. We will compare the results from both methods to gain a better understanding of the performance and efficiency of our implementations.

We hope that through the execution of this project, we will not only provide solutions for the specific problems we have chosen but also contribute to expanding the understanding of how machine learning algorithms can be applied to solve real-world problems.

CHAPTER I: LINEAR REGRESSION

1. Mathematical analysis

- Suppose that we have a predictive function:

$$f(x) = y' = w1x1 + w2x2 + w3x3 + w4x4 \ldots + wnxn + w0$$

where:

+ [w1,w2,w3,...wn] is coefficient needed optimize

+ [x1,x2,x3,...xn] is vector containing parameter to train model

=> The objective of the model will be to find coefficients w1, w2, w3,...wn Such that:

y ≈ f(x) ≈ y'

- This means minimizing the error of the prediction function as much as possible.

- Let:

+ $[1, x1, x2, x3, \ldots xn] = X$ ($1*w0 = w0$)

+ $[wo, w1, w2, w3, \ldots wn]^T = W$ (The transpose of a matrix of predictive function)

- As mentioned earlier, our goal is to find coefficients w1, w2, w3,...wn such that the error between the actual value and the predicted value is minimized.

⇔ Minimize y - y'

- Using Least Ordinary least square, suppose:

$$e = y - y' \quad <=> \quad e^2 = (y - y')^2$$

- The prediction error of a value in the prediction function is given by:

$$\tfrac{1}{2} e^2 = \tfrac{1}{2}(y - y')^2 = \tfrac{1}{2}(y - XW)^2$$

- From there, we will have the loss function for all values in the model represented as follows:

$$L(W) = \frac{1}{2}\sum_{i=1}^{N}(y_i - X_i W)^2$$

- => The problem now is to find the coefficients w such that the value of this loss function is minimized. Let:
    + y = [y1, y2, ... yn]
    + X' = [X1, X2, X3, ... Xn]
  => The loss function at this point will be:

$$L(w) = \frac{1}{2} \cdot \|y - Xw\|_2^2 \quad (1)$$

- In which $\|f(x)\|2$ is referred to as the Euclidean Norm, defined as the sum of squares of each element of the function f(x).
- To minimize the loss function, we solve the equation by setting the derivative of the function at a certain value equal to zero.
  Solve (1):
- We have:

$$f(u) = \frac{1}{2}u$$

$$g(w) = \|y - Xw\|_2^2$$

As the chain rule:

$$\nabla_w f(g(w)) = \nabla_u f(u)\big|_{u = g(w)} \cdot \nabla_w g(w)$$

and g(w) can rewrite follow as:

$$g(w) = (y - Xw)^T (y - Xw)$$

- The derivative of loss function is:

$$\nabla_w \left(\frac{1}{2} \|y - Xw\|_2^2 \right) = \frac{1}{2} \cdot (-2X^T(y - Xw)) = X^T(Xw - y)$$

$$X^T(Xw - y) = 0$$

$$\Leftrightarrow X^T \cdot Xw = X^T \cdot y$$

$$\Leftrightarrow w = A^T b = (X^T X)^{-1} X^T y$$

Solve this linear equation we have w

2. Dataset

- I will use a dataset that explores the relationship between different marketing approaches and sales figures.
- Link dataset: Sales Prediction (Simple Linear Regression) | Kaggle
- The data consists of 4 fields: TV, Radio, Newspaper, Sales. These fields represent the proportional relationship between the amount spent on each marketing channel and the resulting product sales.
- In this project, I will use 2 variables, TV and Newspaper, to create a linear regression model. This model aims to provide the most visually descriptive representation after its implementation.

2.1. Implemented in Python

- After reading data from the csv file, split the dataset into two parts to train and test. In which, the test dataset accounts for 75%, and the training dataset accounts for 25%.

- X is array to save information about how much money pays for advertising on TV, Z is array to save information about how much money pays for advertising in Newspapers and Y is revenue of the production when using the two methods above.

- First, we will create the design matrix X, which contains the information for training. Then, we will compute the Pseudo-inverse matrix ($X^T X$) of X.

- Applying the formula mentioned above: $w = A^T b = (X^T X)^{-1} X^T y$ we obtain w.

2.2. Implemented in Scikit-learn Python

- Using the LinearRegression model of Scikit-learn with two parameters is X, Y.

- In which, X represents the variable that holds the matrix containing the feature information used for training the model. In this case, X is taken from the 'advertising' DataFrame and includes the columns 'TV' and

'Newspaper'. y stands for the variable containing the target values, which refers to the 'Sales' column in the 'advertising' DataFrame. This variable holds the values that the model tries to predict or understand the relationship with the features in the matrix X.

- After that we use model.fit() to train the model.

2.3. Result

y_test	y_predicted_algorithm	y_predicted_sklearn
11.6	14.312748	14.31274765
16.6	17.593739	17.59373887
20.6	16.803587	16.80358701
20.6	16.910956	16.91095559
3.2	6.545149	6.54514891
15.3	12.851677	12.85167705
10.1	15.216503	15.21650316
7.3	8.109419	8.10941934
12.9	14.503218	14.5032177
16.4	16.668567	16.66856669
13.3	12.359232	12.35923194

19.9	17.409094	17.4090942
18.0	15.493659	15.49365899
11.9	12.850968	12.85096812
16.9	21.691390	21.69138981
8.0	7.774484	7.77448375,
17.2	18.257453	18.25745342
17.1	19.840037	19.84003726
20.0	22.209175	22.20917455
8.4	9.474305	9.47430547
17.5	16.703875	16.70387517
7.6	7.736642	7.73664201
16.7	15.923073	15.9230725
16.5	18.946600	18.9466001
27.0	22.817657	22.81765669
20.2	22.817657	20.60599338
16.7	16.670024	16.67002391
16.8	22.284131	22.28413118
17.6	15.905272	15.90527217
15.5	15.134399	15.13439931

Table 1.1 : Result predicted algorithm & Sklearn

2.3.1. With algorithm python
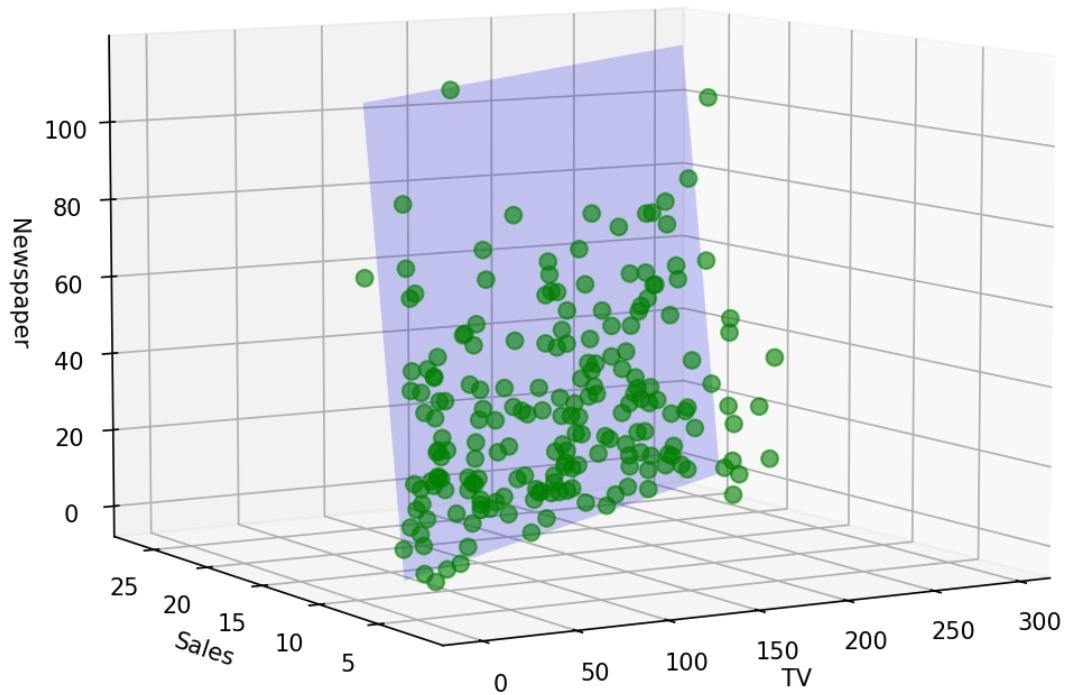

Figure 1.1: The linear regression model created after using the algorithm
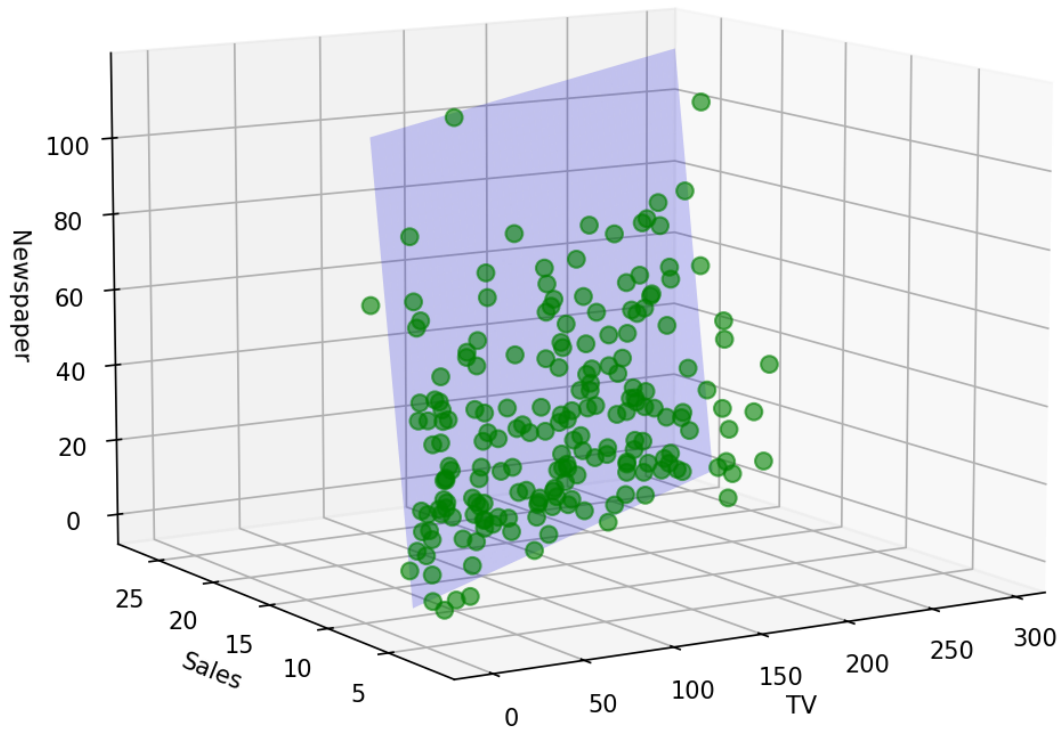
Figure 1.2: The linear regression model created after using Sklearn library

	Time	MSE	R-squared
Algorithm	0.0	5.93317101762521 7	0.789163746645646 4
Scikit-learn	0.0040371417999267 58	5.93317101762522 4	0.789163746645646

Table 1.2: Comparison between two method

3. Comparison two results:

- The two approaches yield similar results. This indicates that the algorithmic approach is accurate and somewhat superior in terms of computation time due to the use of calculation formulas

4. Conclusion

- Manual Linear Regression: Offers insights into underlying mathematical operations but might be less efficient and prone to implementation errors.
- Scikit-learn Linear Regression: Provides an efficient, optimized, and user-friendly interface for linear regression with reliable performance.

5. Recommendations

- For educational purposes or understanding the underlying mathematics, the manual implementation can be beneficial.
- For practical applications, Scikit-learn's implementation is recommended due to its efficiency, reliability, and built-in functionalities.

CHAPTER II: K-NEAREST NEIGHBOR (KNN)

1. Introduction

K-Nearest Neighbours is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining, and intrusion detection.

It is widely disposable in real-life scenarios since it is non-parametric, meaning, it does not make any underlying assumptions about the distribution of data (as opposed to other algorithms such as GMM, which assume a Gaussian distribution of the given data). We are given some prior data (also called training data), which classifies coordinates into groups identified by an attribute.

2. Ideas and algorithms

Idea: The main idea of the k-NN algorithm is to predict the label of a new data point (point a) based on the closest labeled data points to it.
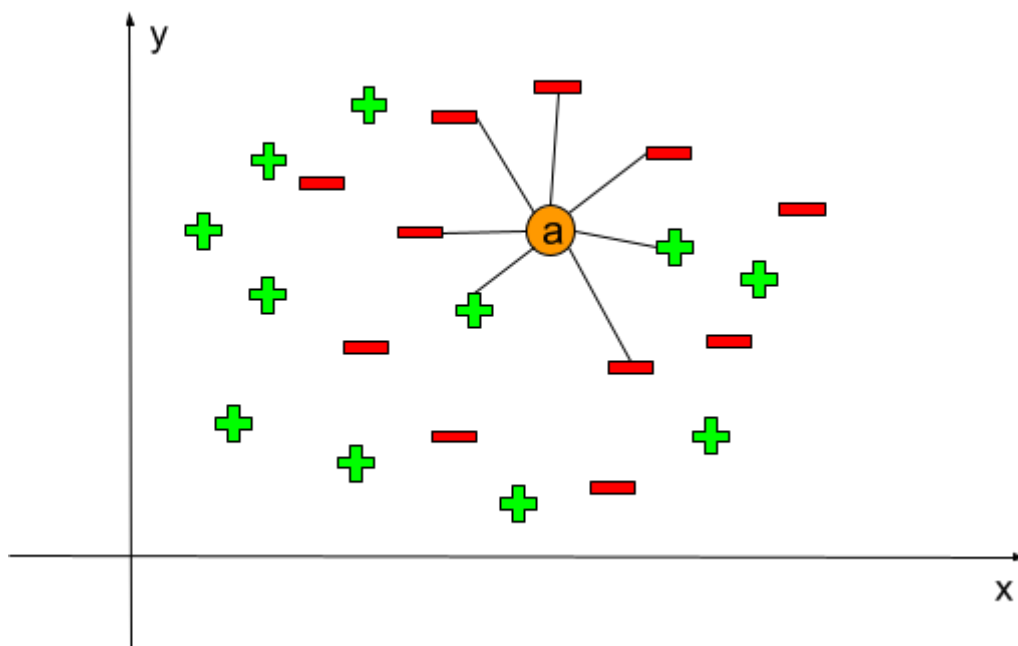
Step 1: Choose quantity k

We will choose a positive odd integer k_, the number of nearest neighbors that the algorithm will consider to make predictions. The value of k can affect the accuracy of the model.

Step 2: Measure the distance

One of the most commonly used distance calculations in the KNN algorithm is the Euclidean distance; You can also use other distance calculations such as Manhattan or Minkowski

Euclidean distance: Suppose there exists a point a with coordinates (x_1, x_2, x_3,..., x_n) and the given points have coordinates (y_1, y_2, y_3,..., y_n). We can calculate the distance between point a and given points using the formula

$$D = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Step 3: Find the k nearest neighbors

After measuring the distance of point a to the given points, we will be able to find out which k nearest neighbors to point a are.

Step 4: Make predictions

Once we have the k nearest neighbors to point a, we can determine the class of the new point based on the percentage of neighbors belonging to each class.

3. Implemented in Python and Scikit-learn Python

3.1. Implemented in Python

Data Loading and Preprocessing:

- The code uses the pandas library to read data from a CSV file named 'KNNDataset.csv'.
- The 'id' and 'diagnosis' columns are dropped from the dataset, and missing values are imputed with the mean using SimpleImputer.

Train-Test Split:

- The data is split into training and testing sets using train_test_split from scikit-learn. The split is 80% training and 20% testing, with a specified random seed (random_state=1234) for reproducibility.

Standardization:

- Features are standardized using StandardScaler from scikit-learn to ensure they have a mean of 0 and a standard deviation of 1.

Euclidean Distance Function:

- The function euclidean_distance calculates the Euclidean distance between two points in the feature space.

Prediction Functions:

- The function predict_label predicts the label for a data point in the test set based on its k-nearest neighbors using the Euclidean distance.
- The function knn_predict predicts labels for the entire test set using the predict_label function.

Number of Neighbors (k):

- The variable k_neighbors is set to 3, indicating that the model considers the three nearest neighbors.
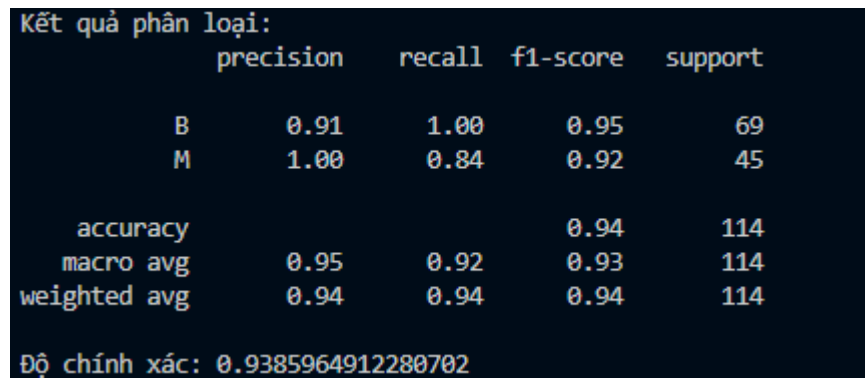
Prediction and Evaluation:

- Labels are predicted for the test set using the KNN algorithm with the specified number of neighbors.
- The classification report, which includes precision, recall, and F1-score for each class, is printed using classification_report from scikit-learn.

- Accuracy is calculated using accuracy_score and printed.

Output:

- The code prints the classification report and accuracy, providing a comprehensive evaluation of the K-Nearest Neighbors model on the test set.

Result:

```
Kết quả phân loại:
              precision    recall  f1-score   support

           B       0.91      1.00      0.95        69
           M       1.00      0.84      0.92        45

    accuracy                          0.94       114
   macro avg       0.95      0.92      0.93       114
weighted avg       0.94      0.94      0.94       114

Độ chính xác: 0.9385964912280702
```

Figure 2.1: Summary result KNN using algorithm

3.2. Implemented Scikit-learn Python

Data Loading:

- The code uses the pandas library to read data from a CSV file named 'KNNDataset.csv'.

Data Preprocessing:

- The 'id' and 'diagnosis' columns are dropped from the dataset, indicating that 'id' is not a relevant feature, and 'diagnosis' is the target variable.

- SimpleImputer is used to fill missing values (NaN) with the mean value. Other imputation strategies can be chosen based on the data characteristics.

Train-Test Split:

- The dataset is split into training and testing sets using train_test_split from scikit-learn. The split is 80% training and 20%

testing, with a specified random seed for reproducibility (random_state=1234).

KNN Model Training:

- An instance of the KNeighborsClassifier from scikit-learn is created with n_neighbors=3, indicating that the algorithm will consider 3 nearest neighbors.
- The model is trained on the training set using fit.

Prediction:

- The trained KNN model is used to predict labels for the test set using the predict method.

Evaluation:

- The classification report is printed using classification_report from scikit-learn, providing metrics like precision, recall, and F1-score for each class ('B' and 'M') and overall metrics.
- Accuracy is calculated using accuracy_score and printed.

Output:

- The code prints the classification report, which includes precision, recall, and F1-score for each class, and the overall accuracy of the model on the test set.

Result:


```
Kết quả phân loại:
              precision    recall  f1-score   support

           B       0.94      0.97      0.96        69
           M       0.95      0.91      0.93        45

    accuracy                           0.95       114
   macro avg       0.95      0.94      0.94       114
weighted avg       0.95      0.95      0.95       114

Độ chính xác: 0.9473684210526315
```

Figure 2.2: Summary result KNN using Scikit-learn

3.3 Conclusion

Machine Learning Approach (scikit-learn's KNeighborsClassifier):
- Pros:
    - Convenience: The scikit-learn implementation is easy to use and requires minimal code.
    - Optimization: scikit-learn's implementation is optimized for performance.
    - Flexibility: It provides various options and configurations for the KNN algorithm.
- Cons:
    - Black Box: The internal workings are abstracted, making it less transparent for customization.
    - Dependency: Requires an external library (scikit-learn).
    - Complexity: For beginners, understanding and customizing might be challenging.

Math-Based Approach (Manually Implemented KNN Algorithm):
- Pros:
    - Transparency: You have complete control over the implementation, making it transparent and customizable.
    - Learning: It's a good exercise for understanding the inner workings of the algorithm.
    - No Dependency: Doesn't rely on external libraries.
- Cons:
    - Performance: May not be as optimized as the scikit-learn version, especially for large datasets.
    - Code Length: The implementation can be longer and requires more effort.

- Error Handling: May require additional code for handling various scenarios, like missing values.

Comparison of Results:
- Both approaches provide reasonably high accuracy, but there are slight differences in precision, recall, and F1-score metrics.
- The scikit-learn version shows slightly better precision for class 'M' and slightly lower recall for class 'B'.
- The accuracy is very close, with the machine learning approach having a slightly higher accuracy.

Conclusion:
- For practical purposes, especially in production environments, using well-established libraries like scikit-learn is often preferred due to their optimization, ease of use, and reliability.
- Manually implementing algorithms can be beneficial for educational purposes or if you need specific customizations.

CHAPTER III: LOGISTIC REGRESSION

1. Introduction

1.1. About Logistic Regression

- **Logistic regression** is a supervised machine learning algorithm primarily used for binary classification. It employs the logistic function, also known as the sigmoid function, which takes an input as the independent variable and produces a probability value ranging from 0 to 1. For example, with two classes, Class 0 and Class 1, if the logistic function's output for an input is greater than 0.5 (the threshold), it belongs to Class 1; otherwise, it belongs to Class 0. It is called regression because it is an extension of linear regression but is mainly used for classification problems. The key difference between linear regression and logistic regression is that linear regression outputs continuous values, whereas logistic regression predicts the probability of an instance belonging to a certain class.

- **Logistic Function (Sigmoid Function):**
  - The sigmoid function is a mathematical function used to map prediction values to probabilities. It maps any real value to another value within the range of 0 and 1. The logistic regression output must be within the range of 0 to 1, forming an "S"-shaped curve.
  - In logistic regression, a threshold value is used to determine the probability as either 0 or 1. Values above the threshold tend to be classified as 1, and values below the threshold tend to be classified as 0.
  - The logistic regression model transforms the continuous output values of the linear regression function into binary class values using the sigmoid function, mapping any set of independent

variables with real values to a value between 0 and 1. This function is called the logistic function.

- Let's denote the independent input features as X

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1m} \\ x_{21} & \cdots & x_{2m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nm} \end{bmatrix}$$

and the dependent variable as Y, which takes binary values 0 or 1.

$$Y = 0 \text{ if class 1 [1 if class 2}$$

- Then, the linear function is applied to the input features X:

$$z = (\textstyle\sum n, i{=}1 \text{ wixi}) + b$$
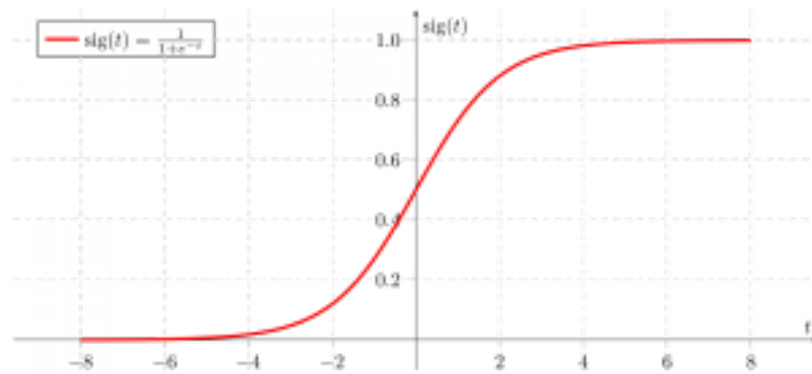
- Where xi is the ith observation of X, wi = (w1, w2, …, wn) is the weight or coefficient, and b is the bias term.
- Simplifying, this can be represented as

$$z = wx + b$$

- All the above discussion is linear regression.

1.2. Sigmoid Function

- Now, the sigmoid function is applied, where the input is z, and it outputs the predicted probability y : $\sigma(z) = \frac{1}{1-e^{-z}}$

Sigmoid function

- The sigmoid function transforms continuous data into probabilities, always bounded between 0 and 1.

    $\sigma(z)$

  · Tends to 1 because

    $z \to \infty$

    $\sigma(z)$

  · Tends to 0 because

    $z \to -\infty$

    $\sigma(z)$

  · Always limited between 0 and 1

● Where the probability of becoming a class can be measured as follows:

$$P(y = 1) = \sigma(z)$$
$$P(y = 0) = 1 - \sigma(z)$$

1.3. Logistic Regression Equation:

● Equation $\quad \dfrac{p(x)}{1-p(x)} = e^z$

- Strange is the ratio between something that happens and something that doesn't. It is different from probability because probability is the ratio of something that happens to everything that can happen.
- Apply natural logs on odd numbers. then the odd log will be:

$$\log\left[\frac{p(x)}{1-p(x)}\right] = z$$
$$\log\left[\frac{p(x)}{1-p(x)}\right] = w \cdot X + b$$

1.4. Logistic regression function

- The predicted probability will be p(X;b,w) = p(x) with y = 1 and with y = 0 the predicted probability will be 1-p(X;b,w) = 1-p(x)

$$L(b,w) = \prod_{i=1}^{n} p(x_i)^{y_i}(1 - p(x_i))^{1-y_i}$$

- Take natural logarithms on 2 sides:

$$
\begin{aligned}
l(b,w) = \log(L(b,w)) &= \sum_{i=1}^{n} y_i \log p(x_i) \;+\; (1-y_i)\log(1-p(x_i)) \\
&= \sum_{i=1}^{n} y_i \log p(x_i) + \log(1-p(x_i)) - y_i \log(1-p(x_i)) \\
&= \sum_{i=1}^{n} \log(1-p(x_i)) + \sum_{i=1}^{n} y_i \log \frac{p(x_i)}{1-p(x_i)} \\
&= \sum_{i=1}^{n} -\log 1 - e^{-(w \cdot x_i + b)} + \sum_{i=1}^{n} y_i (w \cdot x_i + b) \\
&= \sum_{i=1}^{n} -\log 1 + e^{w \cdot x_i + b} + \sum_{i=1}^{n} y_i (w \cdot x_i + b)
\end{aligned}
$$

2. Dataset

Dataset contains information about student outcomes based on two exam scores (represented by grade1 and grade2):

- grade1: The score of the student in the first exam.

- grade2: The score of the student in the second exam.
- label: The outcome label, which is a binary value (0 or 1). This could represent the final outcome, for example, whether a student was admitted to a university (label=1) or not admitted (label=0).

3. Implemented in Python

Data Loading and Preprocessing:

- The dataset is loaded from a CSV file using pandas.
- Input features (X) are selected, and Min-Max Scaling is applied to normalize them.
- The target variable (Y) is extracted.

Train-Test Split:

- The dataset is split into training and testing sets using the train_test_split function from scikit-learn.

Logistic Regression Model:

- A logistic regression model is defined as the Logistic_Regression function.
- The model is trained using the gradient descent optimization algorithm.
- The training process includes updating the model parameters (theta) iteratively, and the cost and theta values are printed at regular intervals.
- Declare_Winner function is called to evaluate the model.

Sigmoid Function:

- The sigmoid function (Sigmoid) is defined, which maps input values to a range between 0 and 1.

Gradient Descent Function:

- The Gradient_Descent function updates the model parameters using the derivative of the cost function with respect to each parameter.

Cost Function:

- The Cost_Function function calculates the cost of the model using the logistic regression cost function.

Hypothesis Function:

- The Hypothesis function calculates the hypothesis (predicted output) using the sigmoid function.

Declare Winner Function:

- The Declare_Winner function evaluates the model's performance by comparing predictions with actual outcomes.

Result:
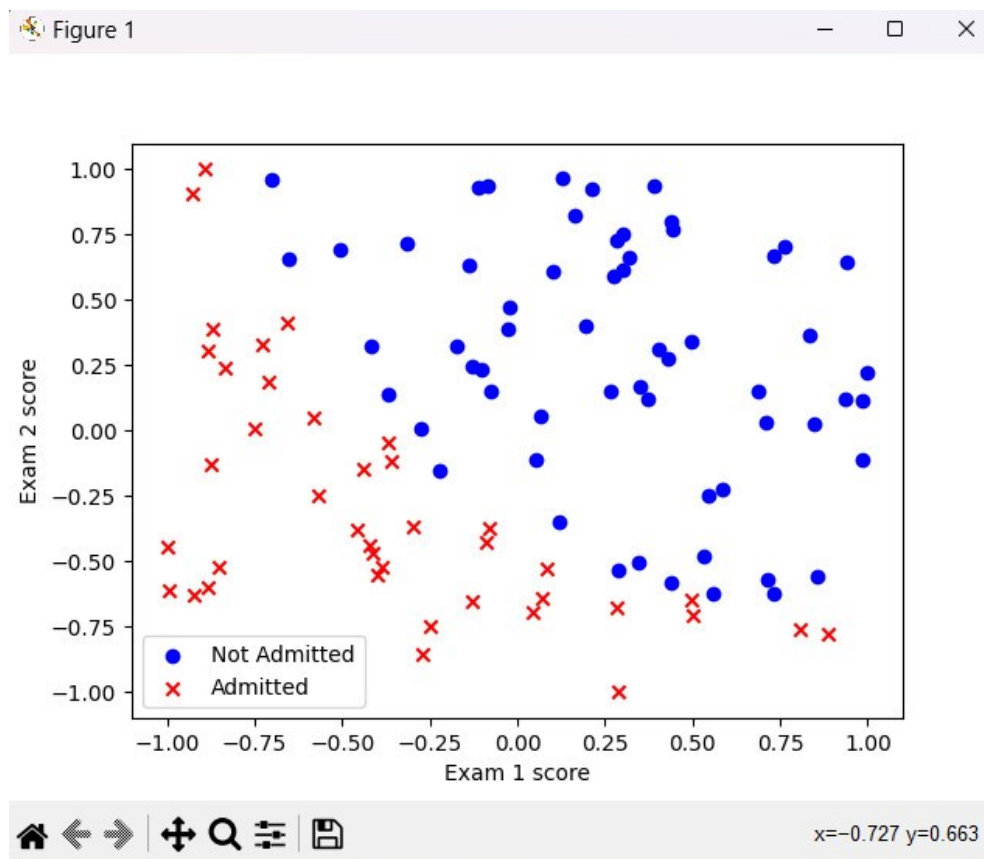
Figure 3.1: Summary result LogisticRegression using algorithm

4. Implemented in Scikit Learn

Data Preprocessing:

● Data from the CSV file is read using the pandas library.
● Input features are normalized using Min-Max Scaling to ensure they have similar scales.

Data Splitting:

● The train_test_split function from scikit-learn is used to split the dataset into a training set and a testing set.

Training Logistic Regression with scikit-learn:

● The LogisticRegression class from scikit-learn is used to train the model on the training set.
● The accuracy score on the test set is printed.

Data Visualization:

● Matplotlib is used to plot a graph and visualize the data.
● Each data point is represented by either an 'o' (Admitted) or 'x' (Not Admitted) on the graph.

Custom Logistic Regression Implementation:

- Functions such as sigmoid, hypothesis, cost function, and gradient descent are implemented based on the provided code.
- Parameters such as alpha (learning rate) and the number of iterations are set.
- The custom Logistic Regression algorithm is run, and the cost values and theta parameters are printed during training**.**

Result:
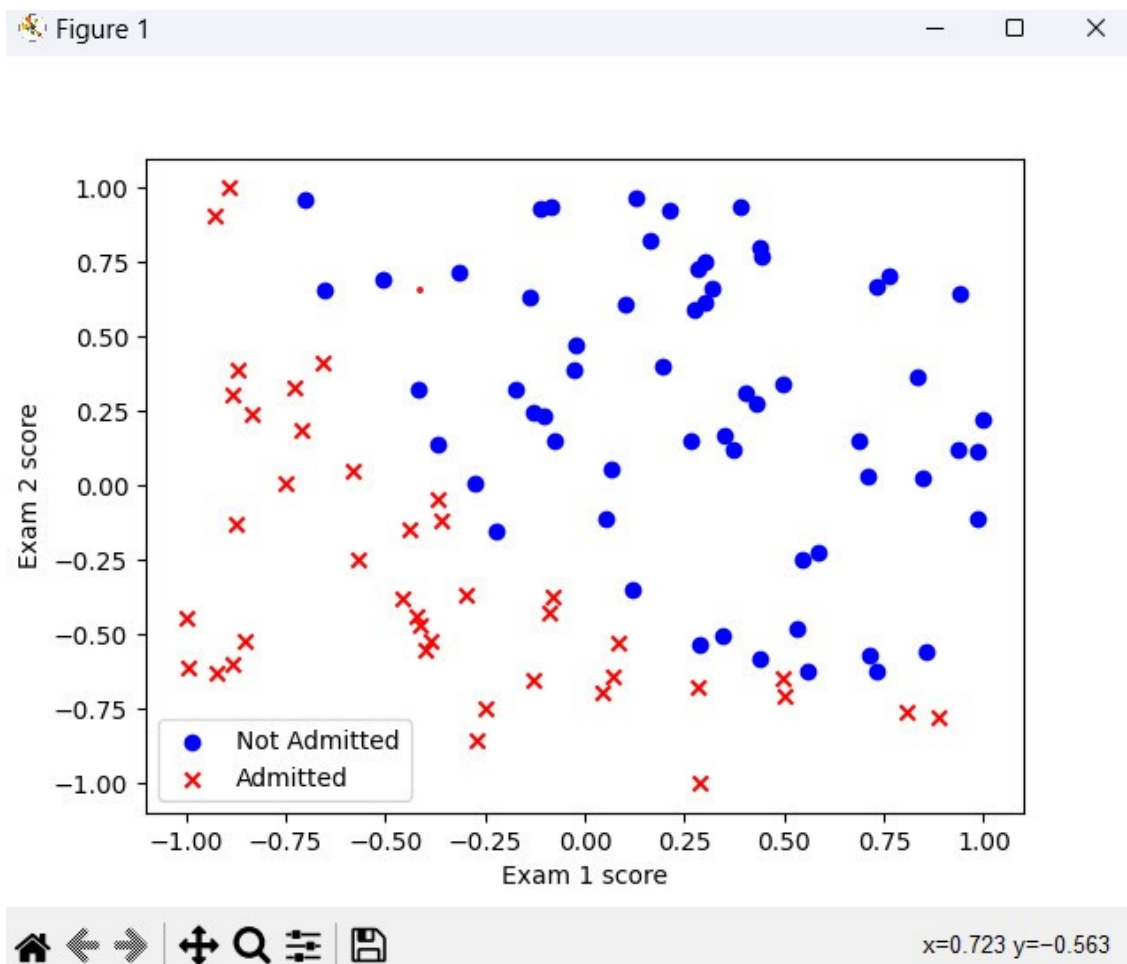
Figure 3.2: Summary result LogisticRegression using ScikitLearn

5. Conclusion

- **Logistic Regression in Python:**

Accuracy: 0.85 (rounded to two decimal places)

Precision: 0.95 (rounded to two decimal places)

Recall: 0.83 (rounded to two decimal places)

F1 Score: 0.88 (rounded to two decimal places)

Confusion Matrix:

[[9 4]

 [1 19]]

- **With Scikit-learn :**

Accuracy: 0.88 (rounded to two decimal places)

Precision: 0.85 (rounded to two decimal places)

Recall: 0.94 (rounded to two decimal places)

F1 Score: 0.89 (rounded to two decimal places)

Confusion Matrix:

[[12 3]

 [1 17]]

- **Comparison:**

The accuracy of your implementation is slightly lower compared to scikit-learn.

The precision of your implementation is higher, indicating a lower rate of false positives.

The recall of your implementation is lower, suggesting a higher rate of false negatives.

The F1 score, which balances precision and recall, is also slightly lower in your implementation.

In summary, while your implementation shows competitive performance, there are some differences in the precision, recall, and F1 score compared to scikit-learn. Fine-tuning your model or using different features might help improve the results.

CHAPTER IV: SUPPORT VECTOR MACHINE (SVM)

1. Introduction: Distance from a point to a hyperplane.

① In 2-dimensional space, the distance from a coordinate point $\left(x_0, y_0\right)$ to a straight line with an equation $\omega_1 x + \omega_2 y + b = 0$ is determined by:

$$\frac{|w_1 x_0 + w_2 y_0 + b|}{\sqrt{w_1^2 + w_2^2}}$$

② In 3-dimensional space, the distance from a coordinate point $\left(x_o, y_o, z_o\right)$ to a plane with an equation $\omega_1 x + \omega_2 y + \omega_3 z + b = 0$ is determined by:

$$\frac{|w_1 x_0 + w_2 y_0 + w_3 z_0 + b|}{\sqrt{w_1^2 + w_2^2 + w_3^2}}$$

The absolute value sign is affected by points on the line/plane:

+ The expression in the absolute value sign has a positive sign located on the same side (positive side of the line).

+ The expression in the absolute value sign has a negative sign on the same side (the negative side of the line).

+ The expression has a value of 0 , meaning the distance is 0.

③ In general, in n-dimensional space, the distance from a point (vector) with coordinates x_0 to a hyperplane (hyperplane) with equation determined by:

$$\frac{|\mathbf{w}^T \mathbf{x}_0 + b|}{||\mathbf{w}||_2}$$

Where $\|\omega\|_2 = \sqrt{\sum_{i=1}^{d} \omega_i^2}$ d is the number of dimensions of space.

2. SVM – Optimization problem

Problem: Given the data pairs of the Training set:

$$\left(x_1, y_1\right), \left(x_1, y_2\right), ..., \left(x_N, y_N\right) \text{with } \vec{x}_i \in R^d \text{the input representation of a}$$
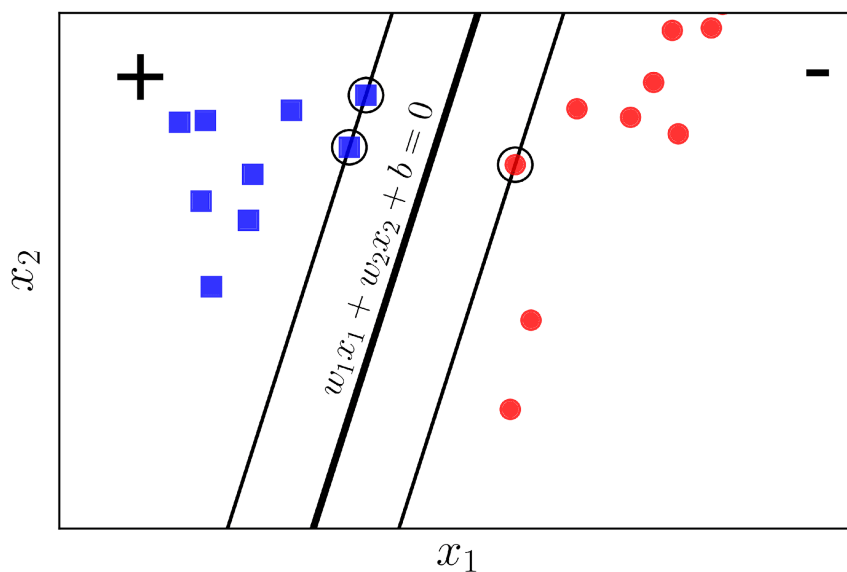
data point.

y_i is the label of that data point.

d is the number of dimensions of the data.

N is the data point.

Suppose the label of each data point is determined by $y_i = 1$(class 1) or

$y_i = -1$ (class 2)

Suppose the blue square points belong to class 1 (positive side), the red round points belong to class -1 (negative side) and the surface $\omega^T x + b = \omega_1 x_1 + \omega_2 x_2 + b = 0$ is the dividing surface between the two classes. We just need to change the signs of w and b to change the positions of the two sets belonging to the two dividing faces.

! Note: w and b are the coefficients to find.

④ For any pair of data $\left(x_n, y_n \right)$, the distance from that point to the dividing surface:

$$\frac{y_n(\mathbf{w}^T \mathbf{x}_n + b)}{||\mathbf{w}||_2}$$

Noticed, y_n always has the same sign as the side of $x_n \to y_n$ has the same sign as $\left(\omega^T x_n + b \right)$, and the numerator is always non-negative.

⑤ With the division pair as above, margin is calculated as the closest distance from a point to that surface (regardless of the point in the two classes):

$$\text{margin} = \min_n \frac{y_n(\mathbf{w}^T \mathbf{x}_n + b)}{||\mathbf{w}||_2}$$

→ ⑥ The optimization problem in SVM is the problem of finding w and b so that this margin reaches the maximum value:
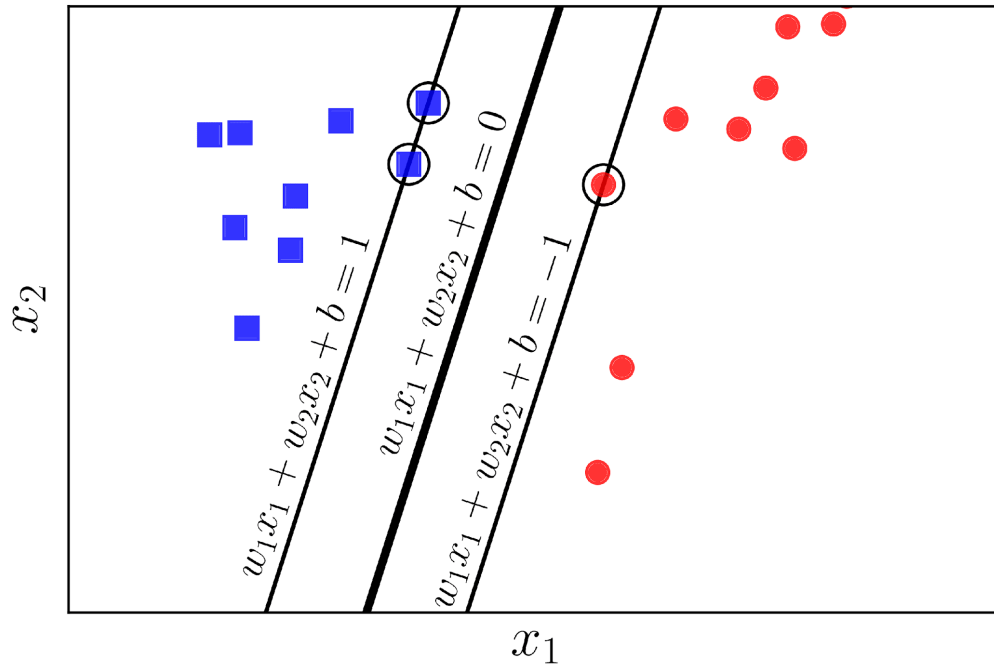
$$(\mathbf{w}, b) = \arg\max_{\mathbf{w},b} \left\{ \min_n \frac{y_n(\mathbf{w}^T \mathbf{x}_n + b)}{||\mathbf{w}||_2} \right\} = \arg\max_{\mathbf{w},b} \left\{ \frac{1}{||\mathbf{w}||_2} \min_n y_n(\mathbf{w}^T \mathbf{x}_n + b) \right\} \quad (1)$$

- Comment: Replace the coefficient vector w by kw and b by kb where k is a positive constant, then the dividing surface does not change, meaning

the distance from each point to the dividing surface remains unchanged, meaning the margin remains unchanged $\rightarrow$ Suppose :

$$y_n(\mathbf{w}^T\mathbf{x}_n + b) = 1$$

- For the points closest to the dividing surface:

For all n, we have:

$$y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1$$

So the optimization problem (1) can be reduced to the following constrained optimization problem;

$$(\mathbf{w}, b) = \arg\max_{\mathbf{w},b} \frac{1}{\|\mathbf{w}\|_2}$$

$$\text{subject to:} \quad y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1, \forall n = 1, 2, \ldots, N \quad (2)$$

Take the inverse of the objective function, square it to get a differentiable function, and multiply by ½ to get a nicer derivative expression

$$(\mathbf{w}, b) = \arg\min_{\mathbf{w}, b} \frac{1}{2}\|\mathbf{w}\|_2^2$$

$$\text{subject to:} \quad 1 - y_n(\mathbf{w}^T \mathbf{x}_n + b) \leq 0, \forall n = 1, 2, \ldots, N \quad (3)$$

Determine the class for a new data point, after finding the separation surface $\omega^T x + b = 0$:

$$\text{class}(\mathbf{x}) = \text{sgn}(\mathbf{w}^T \mathbf{x} + b)$$

The sgn function is a function that determines the sign, receiving the value 1 if the argument is zero and -1 otherwise.

3. SVM – Duality problem

3.1. Test the Slater criterion

a. Introduce

The Slater criterion requires that there exists a non-negative Langrange vector such that each constraint is satisfied and the sum of the Lagrangian product and the constraint value is zero.

b. Check the Slater condition

Considering the Slater condition with the optimization problem (3), we have: if there exists w, b satisfying:

$$1 - y_n(\mathbf{w}^T \mathbf{x}_n + b) < 0, \quad \forall n = 1, 2, \ldots, N$$

$\rightarrow$ Satisfies Strong Duality.

We see that there is always a plane separating two classes if those two classes are Linear Separable, meaning the problem has a solution, so the Feasible Set of the optimization problem (3) must be non-empty. That is, there always exists a pair (ω_0, b_0) such that:

$$1 - y_n(\mathbf{w}_0^T \mathbf{x}_n + b_0) \leq 0, \quad \forall n = 1, 2, \ldots, N$$
$$\Leftrightarrow 2 - y_n(2\mathbf{w}_0^T \mathbf{x}_n + 2b_0) \leq 0, \quad \forall n = 1, 2, \ldots, N$$

Choosing $\omega_1 = 2\omega_0$ and $b_1 = 2b_0$, we have:

$$1 - y_n(\mathbf{w}_1^T \mathbf{x}_n + b_1) \leq -1 < 0, \quad \forall n = 1, 2, \ldots, N$$

$\rightarrow$ Satisfies the Slater condition

3.2. Largrangian SVM problem

From equation (3):

$$\mathcal{L}(\mathbf{w}, b, \lambda) = \frac{1}{2}||\mathbf{w}||_2^2 + \sum_{n=1}^{N} \lambda_n (1 - y_n(\mathbf{w}^T \mathbf{x}_n + b)) \quad (4)$$

With $\lambda = \left[\lambda_1, \lambda_{21} \ldots, \lambda_N\right]^T$ and $\forall n = 1, 2, \ldots, N$

3.3. Lagrange dual function

The Lagrange dual function is defined:

$$g(\lambda) = \min_{\mathbf{w}, b} \mathcal{L}(\mathbf{w}, b, \lambda)$$

With $\lambda \geq 0$

The derivative of $L(w, b, \lambda)$ w and b is 0, we have:

$$\frac{\partial \mathcal{L}(\mathbf{w}, b, \lambda)}{\partial \mathbf{w}} = \mathbf{w} - \sum_{n=1}^{N} \lambda_n y_n \mathbf{x}_n = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^{N} \lambda_n y_n \mathbf{x}_n \quad (5)$$

$$\frac{\partial \mathcal{L}(\mathbf{w}, b, \lambda)}{\partial b} = -\sum_{n=1}^{N} \lambda_n y_n = 0 \quad (6)$$

Substituting (5) and (6) into (4) we have:

$$g(\lambda) = \sum_{n=1}^{N} \lambda_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n \lambda_m y_n y_m \mathbf{x}_n^T \mathbf{x}_m \quad (7)$$

Consider the matrix:

$$\mathbf{V} = [y_1 \mathbf{x}_1, y_2 \mathbf{x}_2, \ldots, y_N \mathbf{x}_N]$$

and vector $1 = [1, 1, \ldots, 1]^T$, we have:

$$g(\lambda) = -\frac{1}{2} \lambda^T \mathbf{V}^T \mathbf{V} \lambda + \mathbf{1}^T \lambda. \quad (8)$$

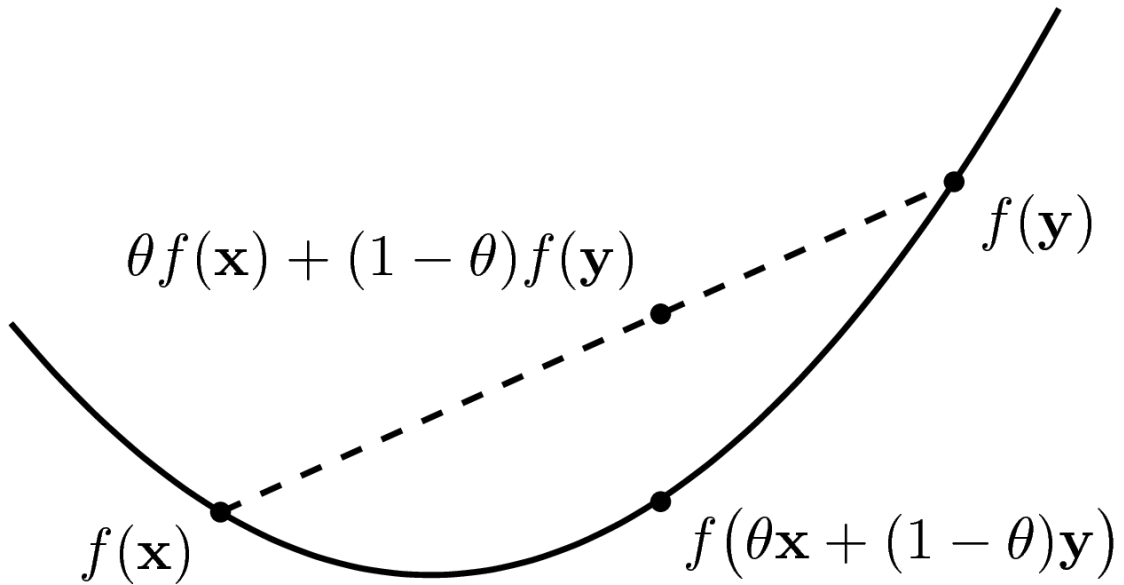Let $k = v^T v$k be a positive semidefinite matrix, for all λwe have:

$$\lambda^T \mathbf{K} \lambda = \lambda^T \mathbf{V}^T \mathbf{V} \lambda = ||\mathbf{V}\lambda||_2^2 \geq 0.$$

- **Definition of Convex Function:** A function $f: R^n \rightarrow R$is called a convex function if dom f is a convex set and:

$$f(\theta \mathbf{x} + (1 - \theta)\mathbf{y}) \leq \theta f(\mathbf{x}) + (1 - \theta)f(\mathbf{y})$$

$$\forall x, y \in f, 0 \leq \theta \leq 1$$

! Note: The condition that dom f is a convex set is very important, because without it, $f(\theta x + (1 - \theta)y)$.

$$\theta f(\mathbf{x}) + (1-\theta)f(\mathbf{y})$$

$$f(\mathbf{y})$$

$$f(\mathbf{x})$$

$$f\big(\theta\mathbf{x} + (1-\theta)\mathbf{y}\big)$$

$$\theta f(\mathbf{x}) + (1-\theta)f(\mathbf{y}) \geq f\big(\theta\mathbf{x} + (1-\theta)\mathbf{y}\big)$$

- **Definition of Concave Function:** A function f is called Concave if - f is Convex.

$\rightarrow y(\lambda) = -\frac{1}{2}\lambda^T k\lambda + 1^T\lambda$ is a concave function.

3.4. Lagrange duality problem

From equation (6) we have the second constraint:

$$\lambda = \arg\max_{\lambda} g(\lambda)$$
$$\text{subject to:} \quad \lambda \succeq 0 \qquad (9)$$
$$\sum_{n=1}^{N} \lambda_n y_n = 0$$

3.5. Economic zone conditions

The problem will satisfy the following EZ conditions:

$$1 - y_n(\mathbf{w}^T\mathbf{x}_n + b) \le 0, \ \forall n = 1, 2, \ldots, N \quad (10)$$

$$\lambda_n \ge 0, \ \forall n = 1, 2, \ldots, N$$

$$\lambda_n(1 - y_n(\mathbf{w}^T\mathbf{x}_n + b)) = 0, \ \forall n = 1, 2, \ldots, N \quad (11)$$

$$\mathbf{w} = \sum_{n=1}^{N} \lambda_n y_n \mathbf{x}_n \quad (12)$$

$$\sum_{n=1}^{N} \lambda_n y_n = 0 \quad (13)$$

From (11) for any n or $\lambda_n = 0$ or $1 - y_n(\omega^T x_n + b) = 0$, we have:

$$\mathbf{w}^T\mathbf{x}_n + b = y_n \quad (14)$$

! Attention: $y_n^2 = 1 \ \forall n$

The points that satisfy (14) are the points closest to the dividing surface, also known as Support Vectors. The number of these satisfaction points is usually a very small number in N.

Considering the problem with a small number of data points N, we can solve the KKT condition system above by considering the cases $\lambda_n = 0$ or $\lambda_n \ne 0$.

The total number of cases considered is 2^n(N > 50).

After finding λfrom (9), we can deduce w based on (12) and b based on (11) and (13) with $\lambda_n \ne 0$.

Calling the set $S = \{n : \lambda_n \ne 0\}$ and N_s the number of elements of set S with $\forall n \in S$, we have:

$$1 = y_n(\mathbf{w}^T\mathbf{x}_n + b) \Leftrightarrow b + \mathbf{w}^T\mathbf{x}_n = y_n$$

$$b = \frac{1}{N_S} \sum_{n \in \mathcal{S}} (y_n - \mathbf{w}^T \mathbf{x}_n) = \frac{1}{N_S} \sum_{n \in \mathcal{S}} \left(y_n - \sum_{m \in \mathcal{S}} \lambda_m y_m \mathbf{x}_m^T \mathbf{x}_n \right) \quad (15)$$

$$\mathbf{w} = \sum_{m \in \mathcal{S}} \lambda_m y_m \mathbf{x}_m \quad (16)$$

From (12), to determine which class a new point x belongs to, we need to determine the sign of the expression

$$\mathbf{w}^T \mathbf{x} + b = \sum_{m \in \mathcal{S}} \lambda_m y_m \mathbf{x}_m^T \mathbf{x} + \frac{1}{N_S} \sum_{n \in \mathcal{S}} \left(y_n - \sum_{m \in \mathcal{S}} \lambda_m y_m \mathbf{x}_m^T \mathbf{x}_n \right)$$

Depends on how to calculate the dot product between pairs of vectors x and $\forall x_n \in S$.

4. Implemented in Python and Scikit-learn Python

4.1. Implemented in Python

Data Loading and Preprocessing:
- The code starts by reading a diabetes dataset from a CSV file using Pandas (pd.read_csv).
- It handles missing values in certain columns ("Glucose", "BloodPressure", "SkinThickness") by replacing zeros with the mean of each respective column.

Data Splitting:
- The dataset is split into features (X) and labels (y) using the train_test_split function from scikit-learn. It reserves 20% of the data for testing (test_size=0.2).

Feature Scaling:

- Standardization of features is performed using the StandardScaler to ensure all features have a mean of 0 and a standard deviation of 1.

Model Training:

- An SVM model with a linear kernel is created using SVC(kernel='linear', C=1.0).
- The model is trained on the standardized training data, using only the first two features (X_train_scaled[:, :2]).

Prediction and Evaluation:

- The model is used to predict labels on the standardized test set (X_test_scaled[:, :2]).
- The accuracy of the model is computed using accuracy_score.

Visualization:

- The code generates a meshgrid to create a decision boundary plot for the SVM model.
- The decision boundary is plotted along with the test data points, and the regions are color-filled based on the predicted class.

Results Display:

- The accuracy of the model is printed to the console.
- A plot is displayed, showing the decision boundary and the distribution of test data points in the feature space.

In summary, the code performs data preprocessing, splits the data, scales features, trains an SVM model, evaluates its accuracy, and visualizes the decision boundary in a 2D feature space.
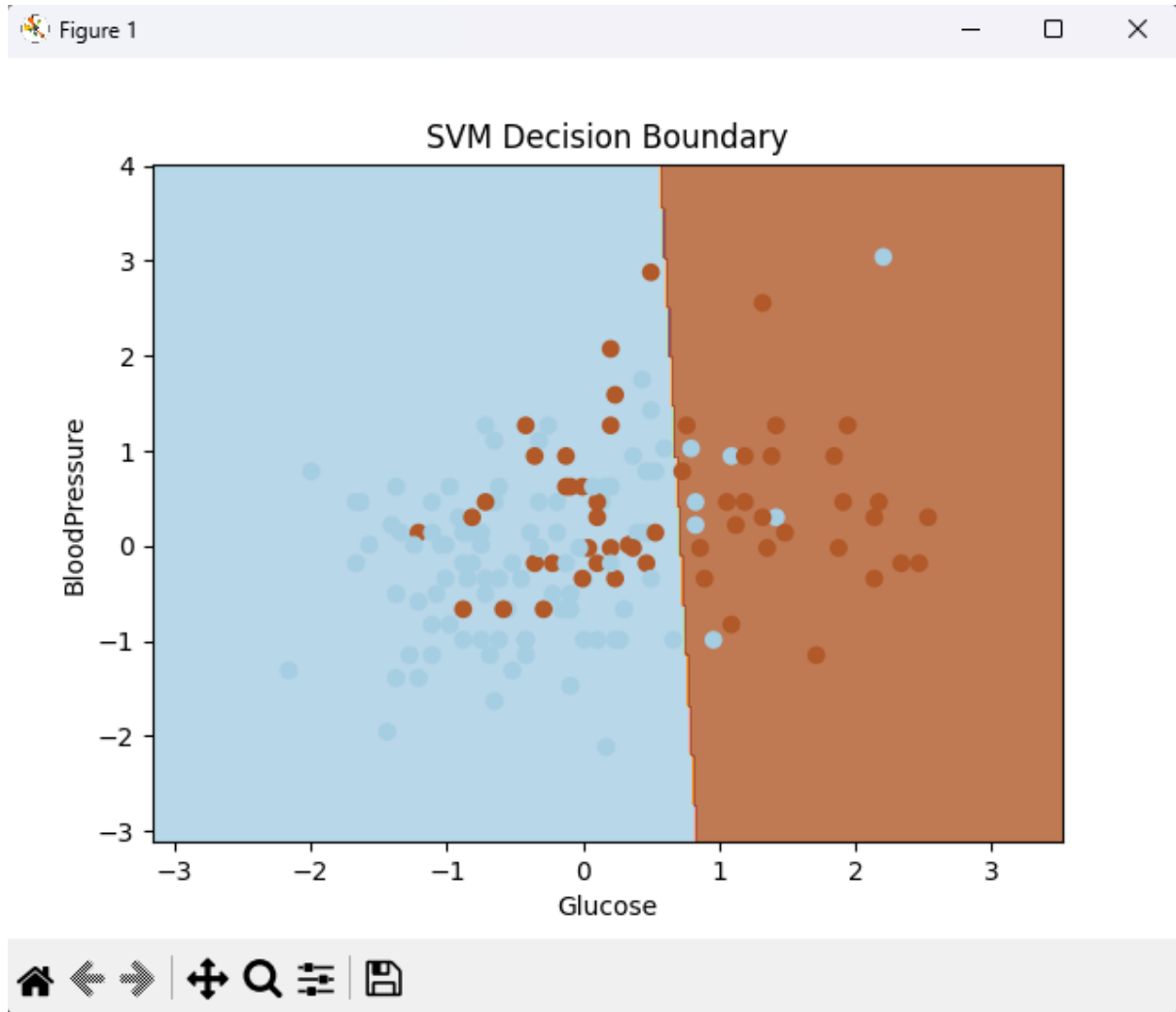
Result:

Accuracy: 0.7662337662337663

Figure 4.1.1: SVM Decision Boundary

4.2. Implemented in Scikit-learn Python

Data Loading and Preprocessing:

- The code starts by loading a diabetes dataset from a CSV file using the Pandas library (pd.read_csv).
- It handles missing values in specific columns ("Glucose", "BloodPressure", "SkinThickness") by replacing zeros with the mean of each respective column.

Data Splitting:

- The dataset is split into features (x) and labels (y) using the train_test_split function from scikit-learn. It reserves 20% of the data for testing (test_size=0.2).

Data Standardization:

- The features are standardized using the StandardScaler from scikit-learn. This step ensures that all features have a mean of 0 and a standard deviation of 1.

SVM Model Creation and Training:

- An SVM classifier with a radial basis function (RBF) kernel is created using SVC(kernel='rbf').
- The model is trained on the standardized training data (x_train_scaled) and corresponding labels (y_train) using the fit method.

Prediction on Test Set:

- The trained SVM model is used to make predictions on the standardized test set (x_test_scaled) using the predict method.

Model Evaluation:

- The code computes various evaluation metrics, including accuracy, confusion matrix, and classification report, using functions from scikit-learn (accuracy_score, confusion_matrix, classification_report).

Results Display:

- The accuracy, confusion matrix, and classification report are printed to the console.
- A heatmap of the confusion matrix is displayed using matplotlib and seaborn for a visual representation of model performance.

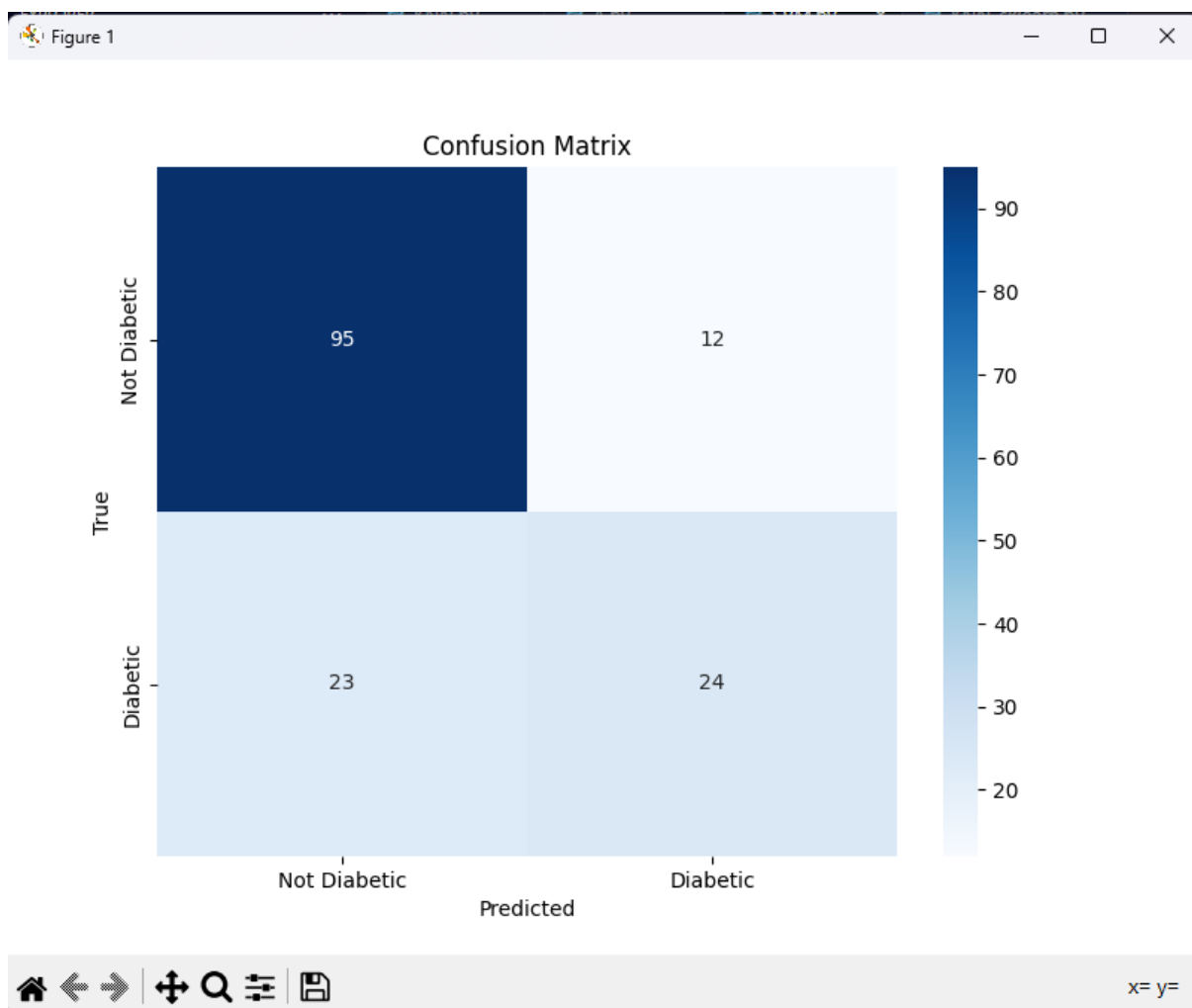Result:

Figure 4.2.1: SVM result

Figure 4.2.2: Confusion Matrix after using Sklearn library

4.3. Conclusion:

Mathematics in Python:

- Algorithm: Utilizes the SVM (Support Vector Machine) algorithm with a linear kernel.
- Feature Selection: Uses only the first two features ("Glucose" and "BloodPressure").
- Accuracy: 0.7662
- Decision Boundary: Plots the decision boundary on the graph, indicating the decision boundary of the SVM.

ML Model with Scikit-learn:

- Algorithm: Employs the SVM model with an RBF (Radial Basis Function) kernel.
- Feature Selection: Utilizes all available features in the dataset.
- Accuracy: 0.7727
- Comprehensive Evaluation: Includes a Confusion Matrix and a Classification Report.

Comparison:

- Feature Selection:
    - Mathematics in Python: Utilizes only the first two features.
    - ML with Scikit-learn: Utilizes all available features.
- Kernel Function:
    - Mathematics in Python: Uses a linear kernel.
    - ML with Scikit-learn: Uses an RBF kernel.
- Model Evaluation:
    - Mathematics in Python: Prints only the accuracy.
    - ML with Scikit-learn: Provides detailed information including Confusion Matrix, Precision, Recall, and F1-score.
- Number of Features:
    - Mathematics in Python: Uses two selected features.

- ML with Scikit-learn: Uses all available features.

Using Machine Learning (ML) models with the Scikit-learn library compared to implementing machine learning algorithms using Python mathematics has some important differences:

1. Simplicity and Convenience: Scikit-learn provides a simple and unified interface for using and tuning machine learning models. It includes many popular machine learning algorithms, from linear regression, SVM, to deep learning algorithms. On the other hand, implementing machine learning algorithms using Python mathematics requires deep knowledge of mathematics and programming, and may take more time.

2. Performance and Optimization: Scikit-learn is optimized to ensure the best performance, with the use of libraries such as NumPy and SciPy. Meanwhile, implementing machine learning algorithms using Python mathematics may not be as well optimized.

3. Flexibility: Implementing machine learning algorithms using Python mathematics allows you to customize the algorithm in your own way, which can be useful if you want to experiment with new methods. However, Scikit-learn also provides some options for tuning the model.

4. Community and Support: Scikit-learn has a large community and many support documents, helping to solve problems you may encounter. On the other hand, implementing machine learning algorithms using Python mathematics may require you to solve many problems on your own.

REFERENCES

1. Linear Regression:

- Kaggle. (2023). "Predicting Stock Prices." Kaggle. https://www.kaggle.com/rohitrox/healthcare-provider-fraud-detection-analysis
- Towards Data Science. (2023). "Linear Regression in Python." Towards Data Science. https://towardsdatascience.com/linear-regression-in-python-9a1f5f000606
- Kaggle. (2023). "Simple Linear Regression." Kaggle. https://www.kaggle.com/andyxie/simple-linear-regression.

2. K-means Clustering (KNN):

- Kaggle. (2023). "University Clustering." Kaggle. https://www.kaggle.com/mohansacharya/graduate-admissions
- Towards Data Science. (2023). "K-means Clustering: Applications in Python." Towards Data Science. https://towardsdatascience.com/k-means-clustering-applications-in-python-6a89e1f3a2a7
- Scikit-learn. (2023). "K-means Clustering." Scikit-learn. https://scikit-learn.org/stable/modules/clustering.html#k-means

3. Logistic Regression:

- Kaggle. (2023). "Loan Prediction." Kaggle. https://www.kaggle.com/altruistdelhite04/loan-prediction-problem-dataset

- Kaggle. (2023). "Logistic Regression." Kaggle.
  https://www.kaggle.com/mnassrib/titanic-logistic-regression-with-python
- Towards Data Science. (2023). "Real-world applications of Logistic
  Regression." Towards Data Science.
  https://towardsdatascience.com/real-world-implementation-of-logistic-regression-5136cefb8125

4. Support Vector Machines (SVM):

- Towards Data Science. (2023). "Support Vector Machines: A Simple
  Explanation." Towards Data Science.
  https://towardsdatascience.com/support-vector-machines-a-simple-explanation-4b3e61db32a3
- Scikit-learn. (2023). "Support Vector Machines." Scikit-learn.
  https://scikit-learn.org/stable/modules/svm.html
- Kaggle. (2023). "SVM Classification." Kaggle.
  https://www.kaggle.com/uciml/breast-cancer-wisconsin-data
- Medium. (2023). "Support Vector Machines." Medium.
  https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72
- Analytics Vidhya. (2023). "Understanding Support Vector
  Machine(SVM) algorithm from examples (along with code)." Analytics
  Vidhya.
  https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/