# Understanding Logistic Regression

Akash Perni, Sai Ganesh Pendela, Samba Sivesh Kurra, K A V Puneeth Sarma
*Maryland Applied Graduate Engineering*
*University of Maryland*
Collge Park, MD, United States
*aperni@umd.edu, spendela@umd.edu, shiv1818@umd.edu, akondamu@umd.edu*

*Abstract*—**When it comes to classification in the fields of machine learning and data analysis, logistic Regression is a crucial tool. This study investigates the development and assessment of Logistic Regression [1] for binary classification on the Wiki Pass/Fail dataset [2] and multi-class classification on the Iris dataset [3]. To better understand the Logistic Regression model, the classifier's performance is compared to that of a Decision tree classifier [4]. The results showed that both the models performed similarly on the dataset suggesting that the data was not large enough. This analysis is essential in developing the ability to compare multiple models in multiple ways and also gain a deeper understanding of Logistic Regression.**

*Index Terms*—**Iris dataset, Logistic Regression, Scikit-learn.**

## I. Introduction

When a classification problem in supervised learning involves data that can be linearly separated, then Logistic Regression is the model to use. Logistic Regression gives good results when data is binary such as True/False problem or any other similar one. This paper focuses on training a Logistic Regression model over the Iris dataset and the Wiki Pass/Fail dataset. The dataset is loaded using the Pandas module [5] into a DataFrame [6] and split into train and test. The train part is utilised during training and the test part is utilised during model evaluation. This is followed by comparing the results obtained from the Logistic Regression model with the results from the Decision tree model.

A brief overview of the columns in the Iris dataset is available in Table I. The objective is to predict the Dependent variable ***Class*** using the Independent variables ***Sepal Length, Sepal Width, Petal Length, Petal Width***.

| **Column** | **Type** |
|---|---|
| Sepal Length | Independent |
| Sepal Width | Independent |
| Petal Length | Independent |
| Petal Width | Independent |
| Class | Dependent |

TABLE I
IRIS DATASET

Table II provides an overview of the Wiki Pass/Fail dataset. The objective is to predict the dependent variable ***Pass*** using the independent variable ***Hours of study***.

| **Column** | **Type** |
|---|---|
| Hours of Study | Independent |
| Pass | dependent |

TABLE II
WIKI PASS/FAIL DATASET

The rest of the paper is organized as follows, Section II describes the Methodology used to build a Logistic Regression model to be trained on the Iris and Wiki Pass/Fail dataset, followed by explaining the results obtained in the Results section. In the Conclusion section, possible reasons for the Results obtained are explained.

## II. Methodology

### A. Prerequisites

Before training the Logistic Regression model, some prerequisites need to be setup or installed.

1) **Python** : Python should be installed on the computer; It can be done by downloading it from the official Python website.

2) **IDLE** : Any Python IDLE like pycharm [7], Google Colab [8], Jupyter Notebook [9] can be used to work with Machine Learning problems.

3) **Packages** : Numpy [10], pandas, Scikit-Learn [11], Matplotlib [12] are needed to work with Logistic Regression.

4) **Data set** : The Iris dataset can be found on multiple sites like the official website [3], the Kaggle website [13] or various other links available on Google. The Wiki Pass/Fail dataset is available in the official Wikipedia entry for Logistic Regression [2].

### B. Importing Packages

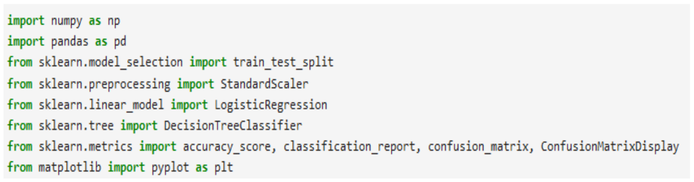
To work with datasets and Logistic Regression, the following packages are required.

2) **Pandas**: Pandas is a freely available Python library that manages and analyzes data effectively. It offers user-friendly tools and functions for handling structured data, like tables or spreadsheets.

3) **sklearn** : Scikit-Learn [11] is an open-source machine learning package that is often used in the Python community and is commonly referred to as ”sklearn” in Python. It offers resources for a range of machine learning projects.

4) **matplotlib** : Matplotlib is useful in plotting complex plots in a very simple manner in Python.

Figure 1 shows all the imports done for the training of the Logistic Regression model.



```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, ConfusionMatrixDisplay
from matplotlib import pyplot as plt
```

Fig. 1. Importing the required packages

## *C. Reading a CSV file*

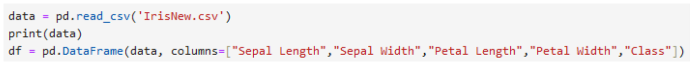
Python supports operations on various types of files. To read a CSV (Comma-Separated Values) file in Python,the CSV module or the pandas library in Python can be used. Here’s a step-by-step process on how to do it.

**Step 1**: Start by importing the panda’s module.

**Step 2**: Use the ***pd.read_csv()*** [14] function to read the CSV file. The **path** to the CSV file should be provided as an **argument**. Various optional parameters to customize the import, such as specifying the **delimiter, encoding, header row,** etc can also be provided.

**Step 3**: After reading the CSV file, the data is stored in a Pandas DataFrame. Then various operations can be performed on this DataFrame, such as displaying its contents, setting the column names etc.

Figure 2 shows the code used to read a CSV using Pandas.



```
data = pd.read_csv('IrisNew.csv')
print(data)
df = pd.DataFrame(data, columns=["Sepal Length","Sepal Width","Petal Length","Petal Width","Class"])
```

Fig. 2. Reading the CSV file using pandas

## *D. Data partitioning into Training and Testing*

Dividing your data into training and testing portions is an essential procedure in machine learning for evaluating how well your model works. In Python, you can achieve this task with the help of libraries such as NumPy or Scikit-Learn. Here’s a method for doing it with Scikit-Learn, which is widely used in Machine Learning Industry.

Here, the first four columns in the Iris data set represent the independent variables and the fifth column is the dependent variable.

The Scikit-Learn function ***train_test_split()*** [15] divides the data into sets for training and testing. The **test_size** parameter specifies the fraction of data to be used for testing (e.g., 0.3 for 30 percent testing data). The **random_state** is an optional parameter used to seed the random number generator for reproducibility.

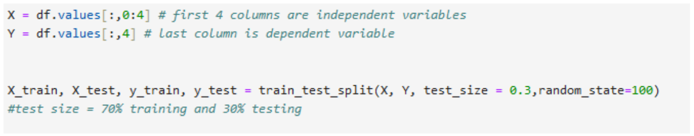
The characteristics of the training dataset are stored in the X_train variable.

The Features from the testing dataset are stored in the X_test variable.

The y_train variable contains the training dataset’s corresponding labels.

The y_test variable has the testing dataset’s related labels.

Figure 3 shows the code to perform splitting using Scikit-Learn.



```
X = df.values[:,0:4] # first 4 columns are independent variables
Y = df.values[:,4] # last column is dependent variable


X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3,random_state=100)
#test size = 70% training and 30% testing
```

Fig. 3. Data partitioning into Training and Testing sets

Once the data is split using these variables, pre-processing can be performed on the data followed by training and evaluation of the machine learning models.

## *E. Building Logistic Regression model on top of the Iris Dataset*

Logistic Regression model can be built on top of the Iris Dataset using the Scikit-Learn module. Scikit-Learn provides the class **LogisticRegression()** [16] that can be used to build a simple Logistic Regression model. The step-by-step procedure to achieve that is mentioned below:

**Step 1**: Start by importing the required libraries, including Scikit-Learn.

**Step 2**: Ensure that you have your dataset prepared with features (X) and corresponding labels (y).

**Step 3**: Initialize the *LogisticRegression()*. You can also specify other hyperparameters to prevent overfitting.

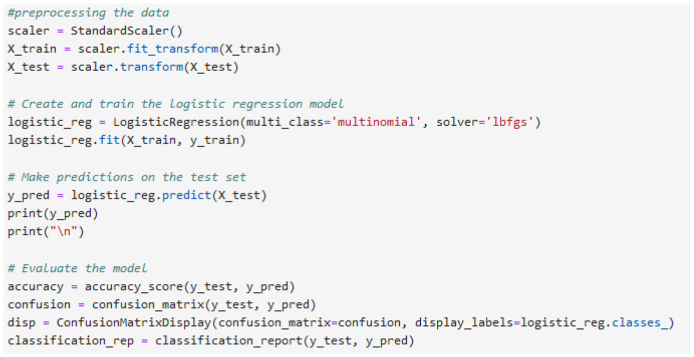
**Step 4**: Train the *LogisticRegression()* on your training data using the *fit()* method.

**Step 5**: After training the classifier, you can use it to make predictions on new, unseen data (e.g., your testing data).

**Step 6**: Use a variety of measures, such as precision, recall, accuracy, or F1-score [17], to assess the effectiveness of your decision tree model. Scikit-Learn methods like ***accuracy_score()*** [18] are also available.

**Step 7** : The probability scores can be calculated using the ***predict_proba()*** [19] method of Scikit-Learn, which takes as input *X_test* in this case. It is also useful in calculating the ranking of the results using the probabilities predicted.

Figure 4 shows the code to build a Logistic Regression model.



```
#preprocessing the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create and train the logistic regression model
logistic_reg = LogisticRegression(multi_class='multinomial', solver='lbfgs')
logistic_reg.fit(X_train, y_train)

# Make predictions on the test set
y_pred = logistic_reg.predict(X_test)
print(y_pred)
print("\n")

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=confusion, display_labels=logistic_reg.classes_)
classification_rep = classification_report(y_test, y_pred)
```

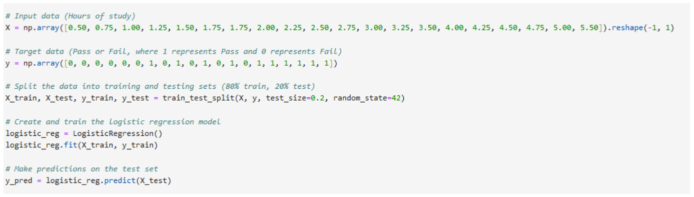
Fig. 4. Logistic Regression using Iris dataset

## *F. Building Logistic Regression model on top of the Wiki Dataset*

To build a Logistic Regression model on Wiki dataset, the same steps in the previous subsection can be used, but with the data from the Wiki dataset. Train-test split is also optional, as there is a separate test data with ground truths available for testing the model performance. But, if there is a necessity to check the model performance before performing testing, then the split can be done to be used as validation data.

Also, ***LogisticRegression()*** by default enables regularization of the weights. This causes the model weights to slightly different resulting in the expected results to be slightly different. This can be fixed by setting the **penalty** parameter to be **None**. This causes the regularization feature to be turned off and the desired results can be expected.

Figure 5 shows the code to build a Logistic Regression classifier on the Wiki dataset.



```
# Input data (Hours of study)
X = np.array([0.50, 0.75, 1.00, 1.25, 1.50, 1.75, 1.75, 2.00, 2.25, 2.50, 2.75, 3.00, 3.25, 3.50, 4.00, 4.25, 4.50, 4.75, 5.00, 5.50]).reshape(-1, 1)

# Target data (Pass or Fail, where 1 represents Pass and 0 represents Fail)
y = np.array([0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1])

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the logistic regression model
logistic_reg = LogisticRegression()
logistic_reg.fit(X_train, y_train)

# Make predictions on the test set
y_pred = logistic_reg.predict(X_test)
```

Fig. 5. Logistic Regression using Wiki dataset

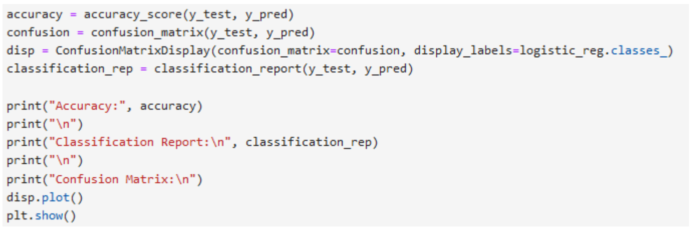
## *G. Generating the results and visualization using Matplotlib*

After doing the necessary training and testing on the data, the results are plotted using the Matplotlib module and the Scikit-Learn module.

The ***confusion_matrix()*** [20] method from the ***sklearn.metrics*** module can be used to generate a confusion matrix of the true predictions to the model predictions, which can then be visualized using the Matplotlib library in Python.

The ***classification_report()*** [21] method from the ***sklearn.metrics*** module is helpful in generating the F1-scores, accuracy, precision and recall for each class of the Iris dataset. The model takes the **true class** and the **predict class** as input to generate the report.

The same scores can be generated in the model trained for the Wiki dataset using the steps described above.



```
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=confusion, display_labels=logistic_reg.classes_)
classification_rep = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)
print("\n")
print("Classification Report:\n", classification_rep)
print("\n")
print("Confusion Matrix:\n")
disp.plot()
plt.show()
```

Fig. 6. Generate results of model and plot the confusion matrix

## RESULTS

After reading the csv file, you can print the sample data in the data frame. A sample data frame was previously visualized in the paper **Decision tree classifier**.

### *A. Results on Wiki dataset*

To ensure that the code is working correctly, the model is tested on the Wiki dataset. The predicted results and the results expected from the Wiki dataset are available in Table III.

| **Hours of study (x)** | **expected probability** | **predicted probability** |
|---|---|---|
| **1** | 0.07 | 0.0709 |
| **2** | 0.26 | 0.2557 |
| **2.7** | 0.50 | 0.4962 |
| **3** | 0.61 | 0.6074 |
| **4** | 0.87 | 0.8744 |
| **5** | 0.97 | 0.9691 |

TABLE III
EXPECTED VS PREDICTED PROBABILITIES FOR LOGISTIC REGRESSION ON WIKI DATASET

Figure 7 shows the same predictions visualized using **Matplotlib** library.

### *B. Results on the Iris dataset*

In case of the Iris dataset, three models are used to evaluate the performance, **Decision tree with gini, Decision tree with entropy** and **Logistic Regression**.

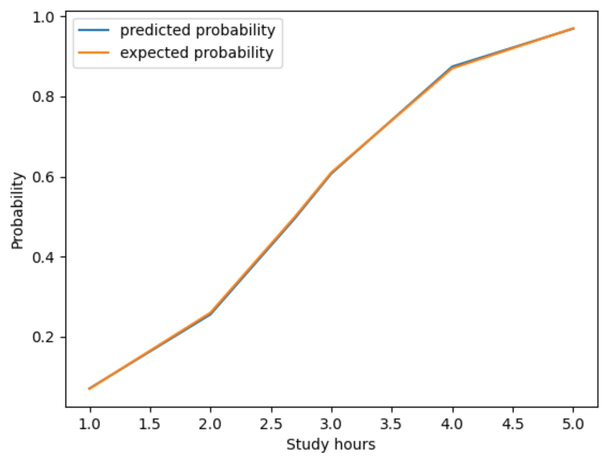


Fig. 7. Plot of Expected vs predicted probabilities

Figure 8 shows the Confusion matrix of the 3 models which shows how accurate each model was for every class in the same test data of Iris dataset.

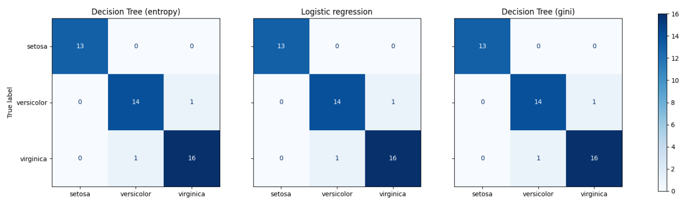


Fig. 8. Confusion matrices for the 3 models on Iris dataset

Figure 9 shows the bar plot providing the F1-scores for each class of the Iris dataset and accuracy for each of the models used.

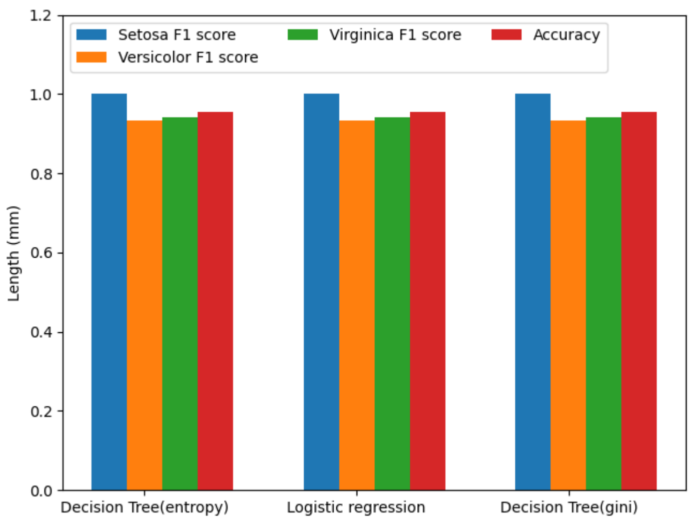
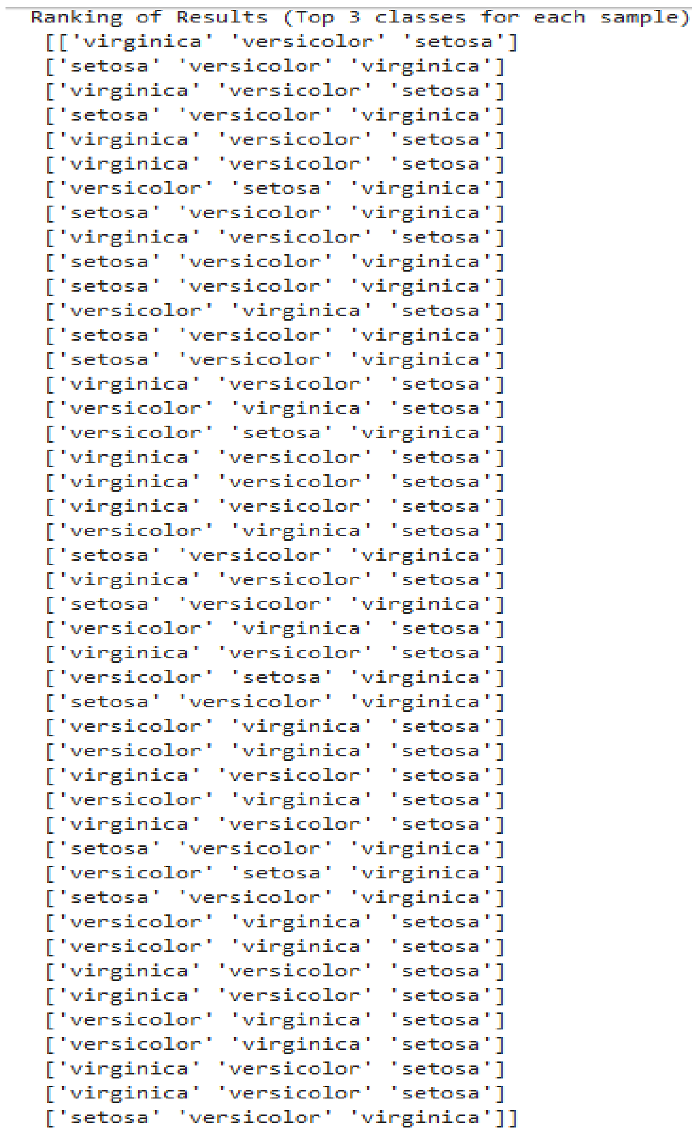


Fig. 9. F1 scores and Accuracy for each species of Iris dataset using different models

## C. Ranking over Iris and unseen data

After the model is trained on the train part of the Iris dataset, it is tested on the test part of the dataset. Figure 10 shows the results of the model after using ***predict_proba*** method of the Scikit-Learn followed by ranking them based on their probabilities. The list contains predicted order of classes based on their probabilities for each test sample.



```
 Ranking of Results (Top 3 classes for each sample)
  [['virginica' 'versicolor' 'setosa']
  ['setosa' 'versicolor' 'virginica']
  ['virginica' 'versicolor' 'setosa']
  ['setosa' 'versicolor' 'virginica']
  ['virginica' 'versicolor' 'setosa']
  ['virginica' 'versicolor' 'setosa']
  ['versicolor' 'setosa' 'virginica']
  ['setosa' 'versicolor' 'virginica']
  ['virginica' 'versicolor' 'setosa']
  ['setosa' 'versicolor' 'virginica']
  ['setosa' 'versicolor' 'virginica']
  ['versicolor' 'virginica' 'setosa']
  ['setosa' 'versicolor' 'virginica']
  ['setosa' 'versicolor' 'virginica']
  ['virginica' 'versicolor' 'setosa']
  ['versicolor' 'virginica' 'setosa']
  ['versicolor' 'setosa' 'virginica']
  ['virginica' 'versicolor' 'setosa']
  ['virginica' 'versicolor' 'setosa']
  ['virginica' 'versicolor' 'setosa']
  ['versicolor' 'virginica' 'setosa']
  ['setosa' 'versicolor' 'virginica']
  ['virginica' 'versicolor' 'setosa']
  ['setosa' 'versicolor' 'virginica']
  ['versicolor' 'virginica' 'setosa']
  ['virginica' 'versicolor' 'setosa']
  ['versicolor' 'setosa' 'virginica']
  ['setosa' 'versicolor' 'virginica']
  ['versicolor' 'virginica' 'setosa']
  ['versicolor' 'virginica' 'setosa']
  ['virginica' 'versicolor' 'setosa']
  ['versicolor' 'virginica' 'setosa']
  ['virginica' 'versicolor' 'setosa']
  ['setosa' 'versicolor' 'virginica']
  ['versicolor' 'setosa' 'virginica']
  ['setosa' 'versicolor' 'virginica']
  ['versicolor' 'virginica' 'setosa']
  ['versicolor' 'virginica' 'setosa']
  ['virginica' 'versicolor' 'setosa']
  ['virginica' 'versicolor' 'setosa']
  ['versicolor' 'virginica' 'setosa']
  ['versicolor' 'virginica' 'setosa']
  ['virginica' 'versicolor' 'setosa']
  ['virginica' 'versicolor' 'setosa']
  ['setosa' 'versicolor' 'virginica']]
```

Fig. 10. Ranking of probabilities on test data of Iris dataset

For the data **[[5.8, 2.8, 5.1, 2.4],[6.0, 2.2, 4.0, 1.0 ]]**, the same model that was trained on the train-test split data is used. This is followed by using the **predict_probaba** function from Scikit-Learn to predict the probabilities and their rankings. The results for the same are as follows:

```
Ranking for New Record 1:
Rank 1: virginica (Probability: 0.9621)
Rank 2: versicolor (Probability: 0.0376)
Rank 3: setosa (Probability: 0.0002)
Ranking for New Record 2:
Rank 1: versicolor (Probability: 0.9507)
Rank 2: virginica (Probability: 0.0418)
Rank 3: setosa (Probability: 0.0074)
```

## Conclusion

In this paper, Logistic Regression is understood, analyzed and explored using the Iris and Wiki dataset. The model is initially evaluated using the Wiki dataset by testing with test values that have their ground truths. The results showed that the predicted probabilities and the ground truths are very close showing that the model's performance is as expected. Further, another model is also developed on the Iris dataset, which was split into train and test. The predicted scores on the test data were compared with those received from the Decision tree classifier which was explored previously. The results showed that all the 3 models Decision tree with gini, Decision tree with entropy and Logistic Regression provided the same results for the test data. The possible reason for this might be due to the size of the dataset. To see a comparable difference between the results, possible methods like data augmentation or choosing a larger dataset etc can be used. Another exploration done in this paper was to understand the working of **predict_proba** method of Scikit-Learn to learn how it works and how to use it. This research is essential not only in understanding more about Logistic Regression but also important in developing the necessary skills to evaluate different models and compare them.

## References

[1] Logistic Regression, https://scikit-learn.org/stable/modules/generated /sklearn.linear_model.LogisticRegression.html

[2] Wiki Pass/Fail dataset, https://en.wikipedia.org/wiki/ Logistic_Regression#Example

[3] Iris dataset, https://archive.ics.uci.edu/dataset/53/iris

[4] Decision tree, https://scikit-learn.org/stable/modules/tree.html

[5] Pandas module, https://pandas.pydata.org/docs/getting_started/index.html

[6] DataFrame, https://www.databricks.com/glossary/what-are-dataframes

[7] PyCharm, https://www.jetbrains.com/pycharm/

[8] Google Colab, https://colab.research.google.com/

[9] Jupyter notebook, https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html

[10] Numpy, https://numpy.org/doc/stable/user/absolute_beginners.html

[11] Scikit-Learn, https://scikit-learn.org/stable/tutorial/index.html

[12] Matplotlib, https://matplotlib.org/stable/tutorials/pyplot.html

[13] Iris dataset on Kaggle, https://www.kaggle.com/datasets/uciml/iris

[14] Pandas read_csv(), https://pandas.pydata.org/docs/reference/api/ pandas.read_csv.html

[15] train-test-split, https://scikit-learn.org/stable/modules/generated/ sklearn.model_selection.train_test_split.html

[16] LogsitcRegression(), https://scikit-learn.org/stable/modules/generated /sklearn.linear_model.LogisticRegression.html

[17] precision-recall-accuracy, https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision _recall_fscore_support.html

[18] accuracy_score(), https://scikit-learn.org/stable/modules/generated /sklearn.metrics.accuracy_score.html

[19] predict_proba(), https://scikit-learn.org/stable/modules/generated /sklearn.linear_model.LogisticRegression.html#sklearn.linear_model. LogisticRegression.predict_proba

[20] confusion_matrix(), https://scikit-learn.org/stable/modules/generated /sklearn.metrics.confusion_matrix.html

[21] classification_report(), https://scikit-learn.org/stable/modules/generated /sklearn.metrics.classification_report.html