Evaluation Metrics

Sai Ganesh Pendela
Maryland Applied Graduate Engineering
University of Maryland
Collge Park, MD, United States
spendela@umd.edu

***Abstract*—This paper addresses the critical issue of flight delays, which have far-reaching implications for both airlines and passengers. Utilizing a comprehensive flight delay dataset of the year 2008, the Decision Tree and Logistic Regression models are built to predict flight delays of at least 15 minutes. The results reveal that the Decision Tree model outperforms Logistic Regression, with a significantly higher Area Under the ROC Curve. The Decision Tree achieved higher AUC than that of the Logistic Regression. The two model's performance is also compared based on different classification metrics.**

***Index Terms*—Flight dataset, Logistic Regression, Decision Tree, Receiver Operating Characteristic (ROC) curve, Area Under the Curve (AUC).**

I. Introduction

In this paper, the Flight dataset [1] is used to train and test two different classifiers, they are, Decision Tree classifier [2], Logistic Regression classifier [3]. Subsequently, the results are compared, the ROC curves are generated and the AUC metrics are compared .

The Flight delay dataset contains 29 columns, they are **'Year', 'Month', 'DayofMonth', 'DayOfWeek', 'DepTime', 'CRSDepTime', 'ArrTime', 'CRSArrTime', 'UniqueCarrier', 'FlightNum', 'TailNum', 'ActualElapsedTime', 'CRSElapsedTime', 'AirTime', 'ArrDelay', 'DepDelay', 'Origin', 'Dest', 'Distance', 'TaxiIn', 'TaxiOut', 'Cancelled', 'CancellationCode', 'Diverted', 'CarrierDelay', 'WeatherDelay', 'NASDelay', 'SecurityDelay', 'LateAircraftDelay'**. Out of all these columns only the columns in below Table I are used to built the Decision Tree and Logistic Regression Model.

Column	Variable Type	Data Type
DayOfWeek	Independent	Int
UniqueCarrier	Independent	String
Origin	Independent	String
Dest	Independent	String
WeatherDelay	Independent	Int
Dep_Time_Bl	Independent	Int
Delay_15	Dependent	Int

TABLE I
FLIGHT DELAY DATASET

The rest of the paper is organized as follows, Section II describes the Methodology to build Decision tree, Logistic Regression, how compare the two models using various classification metrics [5] and how to generate ROC curves and compare the AUC metrics. The Results section provides comparison of the results of the models. In the Conclusion section, possible reasons for the Results obtained are explained.

II. Methodology

In this section, the steps to build Decision tree, Logistic regression on the Flight dataset is explained. Also, how the comparision between two models is done, how the ROC curves are generated and how the AUC metrics are calculated is discussed .

A. Setting up and importing necessary packages

The Machine Learning model is built using **Python 3.8.*** and **Google colab** [6]. The exact versions of them are mentioned in table II

Software	Version
Python	3.8.17
Google colab	Colab Pro

TABLE II
SOFTWARE AND THEIR VERSIONS

Generally any version of Python 3.8.* should work, but the exact version should be preferred to avoid any issues.

Other than the softwares mentioned, All the other packages that are needed are mentioned in the table III.

Module	Version
Numpy [7]	1.24.4
Scikit-Learn [8]	1.3.0
Pandas [9]	2.0.3
Matplotlib [10]	3.7.2

TABLE III
MODULES USED AND THEIR VERSIONS

All the above mentioned packages can be imported using the Python syntax defined.

B. Loading the Flight Delay dataset

The Flight dataset is readily available for download in CSV format from several public sources like Kaggle. Once downloaded, you can import the dataset into a Google colab

notebook as a DataFrame [11] using the Pandas *read_csv()* [12] function. This function accepts the *file path* as an input and returns a DataFrame object. Utilizing a DataFrame to load the data is advantageous because it enables us to access the dataset's columns individually, unlike the conventional row-wise approach commonly used by most software. The loading of the Flight Delay dataset is same as mentioned in the previous paper **Decision Tree Classifier** [14].

C. Data Preprocessing

The Departure Time Between column is not present in the flight delay dataset, so to generate the **Dep_Time_Blk** column, the **"DepTime"** values, which are in the format "hhmm," are first converted into hours, These hour values are then categorized into specific time blocks. The time blocks are defined based on the range of hours.

```
df['Hour'] = df['DepTime'] // 100
# Define a function to categorize hours into hourly blocks
def categorize_hour(hour):
    if 0 <= hour < 6:
        return '00:00-05:59'
    elif 6 <= hour < 12:
        return '06:00-11:59'
    elif 12 <= hour < 18:
        return '12:00-17:59'
    else:
        return '18:00-23:59'

# Apply the categorization function to create the 'DEP_TIME_BLK'
 #column
df['DEP_TIME_BLK'] = df['Hour'].apply(categorize_hour)
```

Fig. 1. Creating the Dep_Time_Blk column.

The dependent variable, the **Delay_15** is also not present in the flight delay dataset, the 'Delay_15' variable is calculated based on the **'ArrDelay**' variable, which represents the arrival delay of a flight in minutes. 'Delay_15' is a binary indicator variable that is set to 1 if the 'ArrDelay' is greater than or equal to 15 minutes, Otherwise, 'Delay_15' is set to 0, indicating that the flight was not delayed by at least 15 min.

Since, the **'Origin', 'Dest', 'UniqueCarrier', Dep_Time_Blk** columns are categorical, Encoding is done on the four columns. Encoding is necessary to convert categorical data into numerical format because some machine learning models cannot deal with categorical data. ***LabelEncoder*** [17] in Python is a tool that assigns a unique numerical label to each category in a categorical variable, making it suitable for model input.

```
# Encode categorical variables
label_encoder = LabelEncoder()
for column in ['Origin', 'Dest', 'UniqueCarrier', 'DEP_TIME_BLK']:
    df[column] = label_encoder.fit_transform(df[column])
```

Fig. 2. Encoding the categorical variables.

There may be some missing values and NA values in the dataset. The missing values should be identified and handled. The ***df[columns_to_check].isnull().sum()*** [18] calculates the number of missing values in each column specified in the specified columns. The ***fillna*** [19] fills missing values in the specified column with a value of 0 using the fillna method. The ***inplace=True*** parameter ensures that the changes are applied directly to the DataFrame.

```
columns_to_check = ['DayOfWeek', 'Origin', 'Dest',
'UniqueCarrier', 'WeatherDelay', 'Delay_15', 'DEP_TIME_BLK']

# Check for missing values in the specified columns
missing_values = df[columns_to_check].isnull().sum()

# Print the missing values count for each column
print(missing_values)
```

```
DayOfWeek              0
Origin                 0
Dest                   0
UniqueCarrier          0
WeatherDelay     1804634
Delay_15               0
DEP_TIME_BLK           0
dtype: int64
```

Fig. 3. Finding and handling the missing values.

D. Defining Target Variables and Data partitioning

The **Delay_15** is the dependent variable and the **DayOfWeek', 'Origin', 'Dest', 'UniqueCarrier', 'WeatherDelay', 'DEP_TIME_BLK'** are Independent variables.

Data partitioning is done to train and test models, prevent overfitting, assess generalization, and facilitate hyperparameter tuning and model selection. To perform data partitioning, the *train_test_split()* [13] function from Scikit-Learn is used. The function takes *data*, *ground truth* and *test_size* as input to return the *X_train*, *X_test*, *y_train* and *y_test*.

Data partitioning results in four variables:

X train (for training data), X test (for testing data), y train (ground truths for training), and y test (ground truths for evaluation).

A *30%* test split is chosen, offering a suitable number of samples for both training and testing, such as *100* training samples and *50* testing samples in the case of the Iris dataset.

The above process is same as mentioned in the previous paper **Decision Tree Classifier** [14].

E. Building models

Scikit-Learn provides various classes and functions to build Machine learning models. Decision tree, Logistic regression models are built using the methodology described in the papers **Decision Tree Classifier** [14], **Understanding**

Logistic Regression [15] respectively.

F. Comparing the models

The *sklearn.metrics* is a module in the scikit-learn (sklearn) library, which is a popular machine learning library in Python. This module contains various functions and classes for evaluating the performance of machine learning models. The ***confusion_matrix()*** [20], ***accuracy_score()*** [20], ***classification_report()*** [21], these functions should be imported to calculate the accuracy and to view the confusion matrix and classification report.

Using the all the functions above the decision tree model, logistic regression model are compared. The code snippet for calculating the metrics of decision tree and logistic regression is same as mentioned in the previous paper **Classification Metrics** [16].

G. Generating ROC curves and comparing AUC metrics

ROC [22](Receiver Operating Characteristic) curves are graphical representations of a binary classification model's performance. They illustrate the trade-off between a model's True Positive Rate (TPR) and False Positive Rate (FPR) at various classification thresholds. To calculate an ROC curve in Python, you can use libraries like scikit-learn.

The ***roc_curve*** [23]function in the scikit-learn is used calculating ROC curves. The ***y_true*** parameter in the function is used to provide the true binary labels (0 or 1) of the samples. The ***y_score*** parameter represents the predicted scores or probabilities that the samples belong to the positive class. The function return three arrays, **fpr** (False Positive Rate), an array of FPR values for different threshold settings, **tpr** (True Positive Rate), an array of TPR values for the corresponding threshold settings and the thresholds, an array of the thresholds used to compute the FPR and TPR values.

After obtainig the FPR and TPR values, the ***auc*** function is used to compue the Area under the ROC curve. This value which represents the performance of your binary classification model.

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Decision Tree ROC curve and AUC
decision_tree_proba = decision_tree.predict_proba(X_test)[:, 1]
fpr_dt, tpr_dt, _ = roc_curve(y_test, decision_tree_proba)
roc_auc_dt = auc(fpr_dt, tpr_dt)

# Logistic Regression ROC curve and AUC
logistic_regression_proba = logistic_regression.predict_proba(X_test)[:, 1]
fpr_lr, tpr_lr, _ = roc_curve(y_test, logistic_regression_proba)
roc_auc_lr = auc(fpr_lr, tpr_lr)
```

Fig. 4. Generating the ROC curves and finding the AUC.

RESULTS

A. Metrics

Both the decision tree and the logistic regression almost have the same accuracy, but have a different F1 and AUC score. The decision tree has higher F1 and AUC score than the logistic regression.

Metric	Decision Tree	Logistic Regression
Accuracy	0.7767541987203634	0.7796491463043727
F1 score	0.27	0.12
AUC score	0.67	0.62

TABLE IV
METRICS OF THE MODEL'S

B. Confusion Matrix and Classification Report

The confusion matrix helps assess the model's accuracy, precision, recall, and F1-score. The classification report provides the precision, recall, f1-score, support, accuracy, average, weighted average for the model. The classification report provides all the above metrics for each class.

The confusion matrix and classification report for the decision tree is shown below.

```
Confusion Matrix:
[[527705  20587]
 [139421  29053]]
Classification Report:
              precision    recall  f1-score   support

       False       0.79      0.96      0.87    548292
        True       0.59      0.17      0.27    168474

    accuracy                           0.78    716766
   macro avg       0.69      0.57      0.57    716766
weighted avg       0.74      0.78      0.73    716766
```

Fig. 5. Confusion matrix and Classification Report for Decision Tree.

The confusion matrix and classification report for the Logistic Regression is shown below.

```
Confusion Matrix:
[[548132    160]
 [157780  10694]]
Classification Report:
              precision    recall  f1-score   support

       False       0.78      1.00      0.87    548292
        True       0.99      0.06      0.12    168474

    accuracy                           0.78    716766
   macro avg       0.88      0.53      0.50    716766
weighted avg       0.83      0.78      0.70    716766
```

Fig. 6. Confusion matrix and Classification Report for Logistic Regression.

C. ROC curve and AUC

The decision tree has higher AUC than the logistic regression and the ROC curve for the decision tree is shown below.
The ROC curve for the Logistic Regression is shown below.

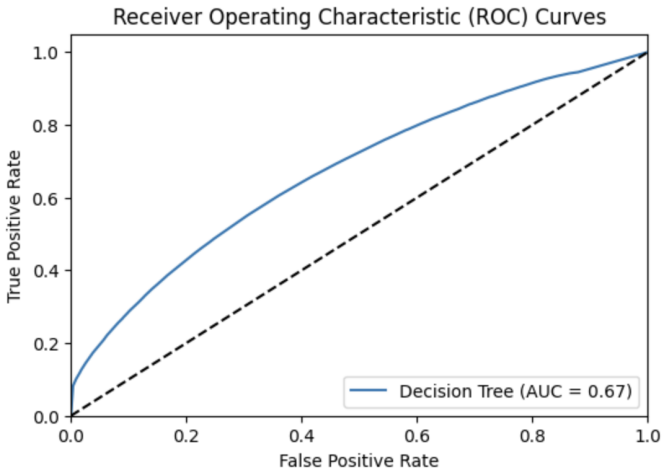

Fig. 7. ROC curve for Decision Tree.

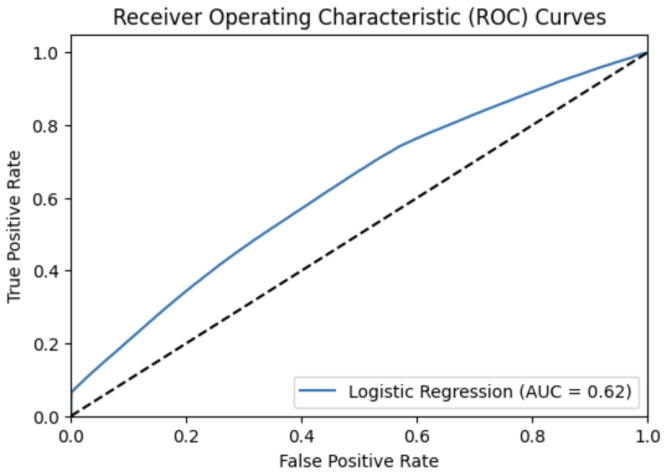

Fig. 8. ROC curve for Logistic Regression.

CONCLUSION

For the flight delay dataset, the constructed Decision Tree and Logistic Regression (LR) exhibited almost the same accuracy, with an identical value of **0.78**, indicating their ability to correctly classify flights. However, when considering the F1 score, a metric that balances precision and recall, the Decision Tree model outperformed LR significantly, achieving an F1 score of **0.27** compared to LR's **0.12**, highlighting Decision Tree's superior ability to correctly identify delayed flights while minimizing false positives. Additionally, the ROC AUC score, reflecting the overall discriminative power of the models, favored the Decision Tree model with a score of **0.67**, whereas LR achieved a score of **0.62**. This suggests that the Decision Tree model possesses a better ability to distinguish between delayed and non-delayed flights. In summary, while both models exhibited comparable accuracy, the Decision Tree model showcased superior performance in correctly identifying flight delays, as indicated by higher F1 score and ROC AUC, making it the preferred choice for this classification task and The Decision Tree model has higher AUC than Logistic Regression due to its ability to capture non-linear relationships and adapt to data distribution, enabling it to discriminate between classes more effectively in the ROC analysis. Additionally, Decision Trees can better handle imbalanced data and overfitting, contributing to their superior AUC performance.

REFERENCES

[1] https://www.kaggle.com/datasets/giovamata/airlinedelaycauses
[2] https://en.wikipedia.org/wiki/Decision_Trees.
[3] https://www.geeksforgeeks.org/understanding-logistic-regression/
[4] https://www.ibm.com/topics/knn
[5] https://neptune.ai/blog/performance-metrics-in-machine-learning-complete-guide
[6] https://colab.google/
[7] https://numpy.org/doc/stable/
[8] https://scikit-learn.org/stable/tutorial/index.html
[9] https://pandas.pydata.org/docs/getting_started/index.html
[10] https://matplotlib.org/stable/tutorials/pyplot.html
[11] https://www.geeksforgeeks.org/python-pandas-dataframe/
[12] https://www.geeksforgeeks.org/python-read-csv-using-pandas-read_csv/.
[13] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
[14] Decision Tree Classifier Paper
[15] Understanding Logistic regression
[16] Classification Metrics
[17] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html
[18] https://note.nkmk.me/en/python-pandas-nan-judge-count/
[19] https://scikitlearn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html
[20] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion _matrix.html
[21] https://scikitlearn.org/stable/modules/generated/sklearn.metrics.classification_report.html
[22] https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5
[23] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html