Flight Delay Prediction

Bui Tuan Kiet
20521493@gm.uit.edu.vn

Tran Dinh Khoi
20521482@gm.uit.edu.vn

Ton Anh Truc
20520944@gm.uit.edu.vn

Vu Quoc Thai Binh
20521119@gm.uit.edu.vn

***Abstract*—Air traffic growth has led to increased economic development, making it crucial to address flight delays, which have negative impacts on airlines, passengers, and the environment. Predicting flight delays accurately is vital for enhancing customer satisfaction and maximizing revenue. In this study, various machine learning algorithms including Decision Tree (DT), Random Forest (RF), XGBoost (XGB), and K-nearest neighbors (KNN) are employed for prediction. Furthermore, the train test split method is employed to bolster the models' performance and reliability.**

I. INTRODUCTION

The rapid growth of air traffic in recent times has significantly contributed to economic development. However, the issue of flight delays has emerged as a crucial concern for both airlines and passengers. Flight delays have negative impacts on various stakeholders, including commuters, airline industries, airport authorities, and the environment. It is essential to address this issue to minimize economic losses and environmental harm caused by increased fuel consumption and gas emissions. Predicting flight delays has become a necessity across a wide range of airline networks.

Accurate estimation of flight delays is critical for airlines as it can enhance customer satisfaction and improve revenue generation. In this study, our team focuses on modeling and predicting flight delays by extracting relevant features and characteristics. We propose a model that employs four distinct Machine Learning algorithms and tuning their hyperparameters to identify the best prediction method. The objective is to enhance the overall efficiency of airline operations and mitigate the adverse impacts of flight delays.

Within this specific context, the following representations are assigned to the corresponding Machine Learning algorithms: Truc represents the KNN, Kiet represents the Decision Tree, Binh represents the Random Forest and Khoi represents the XGBoost. This report has been written by Khoi and Kiet.

II. ALGORITHMS

A. *K-nearest neighbors*

The K-Nearest Neighbors (KNN) algorithm is a non-parametric machine learning algorithm that is used for classification and regression problems. It is a simple and effective algorithm that is based on the idea of similarity between data points.

In KNN, the number of neighbors (K) is specified and the algorithm identifies the K nearest data points to the query point. The class or output value of the query point is then determined by the majority class or average output value of its K-nearest neighbors.

The distance metric used for computing the similarity between data points can be Euclidean, Manhattan, Minkowski, etc. The choice of distance metric depends on the type of data and the problem at hand. Example of Euclidean:

$$\text{distance}(q, x_i) = \sqrt{\sum_{j=1}^{n} (q_j - x_{ij})^2}$$

where q_j represents the jth feature of the query point q and x_{ij} represents the jth feature of the data point x_i. This formula calculates the Euclidean distance between the query point and each data point in the dataset, which is used to find the k nearest neighbors in the KNN algorithm.

KNN can be used for both classification and regression problems. In classification problems, the output variable is categorical and KNN assigns the class to the query point based on the majority class of its K-nearest neighbors. In regression problems, the output variable is continuous and KNN assigns the output value to the query point based on the average output value of its K-nearest neighbors.

KNN's Parameters

KNeighborsClassifier(n_neighbors = 5, *, weights = 'uniform', algorithm = 'auto', leaf_size = 30, p=2, metric = 'minkowski', metric_params = None, n_jobs = None)

- **n_neighbors**: This parameter corresponds to the value of k in KNN. It determines the number of neighbors to consider for classification.
- **weights**: This parameter specifies the weighting scheme for the neighbors. The default is 'uniform', where all neighbors are equally weighted. Alternatively, you can choose 'distance', where closer neighbors have more influence on the prediction.
- **metric**: This parameter determines the distance metric to be used. The default is 'minkowski' with a parameter p=2, which corresponds to the Euclidean distance. Other options include 'manhattan' for Manhattan distance and 'cosine' for cosine similarity.

Advantages of KNN

- Simple and easy to implement.
- Can be used for both classification and regression problems.

- Does not make any assumptions about the underlying data distribution.
- Can work well with small datasets.
- Can be easily updated with new data.

Disadvantages of KNN

- Computationally expensive, especially with large datasets.
- Requires a distance metric to compute similarity between data points.
- Sensitive to the choice of k and the distance metric used.
- Can be affected by the presence of irrelevant features in the data.
- Can be biased towards the class with more instances in the training data.

B. Decision Tree

Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, the decision tree algorithm can be used for solving regression and classification problems too. The goal of using a Decision Tree is to create a training model that can use to predict the class or value of the target variable by learning simple decision rules inferred from prior data(training data).

Decision Tree works in the following stages:

1) Select the best attribute using Attribute Selection Measures (ASM) to split the records.
2) Make that attribute a decision node and divide the dataset into smaller subsets.
3) Start tree building by repeating this process recursively for each child until one of the conditions matches:
    - All the tuples belong to the same attribute value.
    - There are no more remaining attributes.
    - There are no more instances.

Important Terminology related to Decision Trees

- **Root Node:** It represents the entire population or sample and this further gets divided into two or more homogeneous sets.
- **Splitting:** It is a process of dividing a node into two or more sub-nodes.
- **Decision Node:** When a sub-node splits into further sub-nodes, then it is called the decision node.
- **Leaf / Terminal Node:** Nodes do not split is called Leaf or Terminal node.
- **Pruning:** When we remove sub-nodes of a decision node, this process is called pruning. You can say the opposite process of splitting.
- **Branch / Sub-Tree:** A subsection of the entire tree is called branch or sub-tree.
- **Parent and Child Node:** A node, which is divided into sub-nodes is called a parent node of sub-nodes whereas sub-nodes are the child of a parent node.

Decision Trees' Parameters

DecisionTreeClassifier(*, criterion = 'gini', splitter = 'best', max_depth = one, min_samples_split = 2, min_samples_leaf = 1, max_features = None, max_leaf_nodes = None)

- **criterion**: Determines the method to measure the quality of a split. Two popular options are "gini" and "entropy".
- **splitter**: Determines the strategy to choose the feature to split on at each node. Two common options are "best" (select the best feature) and "random" (select a random feature).
- **max_depth**: Limits the maximum depth of the decision tree. If not specified, the tree will grow until it cannot make further splits or reaches the minimum samples per leaf.
- **min_samples_split**: The minimum number of samples required to perform a split. If a node has fewer samples than this value, it will not be split.
- **min_samples_leaf**: The minimum number of samples required to be at a leaf node. If a leaf has fewer samples than this value, a split will not be performed for that leaf.
- **max_features**: The maximum number of features to consider when looking for the best split.
- **max_leaf_nodes**: Limits the maximum number of leaf nodes in the decision tree. If not specified, the tree will grow until it cannot make further splits or reaches the minimum samples per leaf.

Advantages of Decision Tree

- Easy to understand and interpret.
- Can handle both categorical and numerical data.
- Can handle missing values and noisy data.
- Can be used for both classification and regression problems.
- Can help identify important features in the data.
- Can be combined with other Decision Tree to form ensemble methods that can further improve performance.

Disadvantages of Decision Tree

- Can be prone to overfitting, especially with complex trees.
- Can be sensitive to small changes in the data, which can result in different tree structures.
- Can be biased towards features with more levels or more categories.
- Can be biased towards features with more instances in the training data.
- Can be computationally expensive, especially with large datasets.
- Can produce unstable trees that can be difficult to interpret.

C. Random Forest

Random Forest is a supervised machine learning algorithm—used for both classification and regression problems.

A Random Forest is a combination of multiple Decision Trees. Two types of randomnesses are built into the trees. First, each tree is constructed using a random subset of the original data. Second, during the construction of each tree, only a subset of features is randomly chosen at each node to determine the optimal split.

Random Forest works in the following stages:

1) Random subsets of the original data (with replacement) are created, called bootstrap samples. This is done to create multiple versions of the training data.
2) A decision tree is built on each bootstrap sample. However, during the building process, only a random subset of the features is considered for each split. This is done to introduce further randomness and reduce correlation between the trees.
3) Once all the decision trees are built, predictions are made by aggregating the predictions of all the trees. For classification problems, the mode (most common) class is taken as the final prediction. For regression problems, the mean prediction of all the trees is taken as the final prediction.
4) Finally, the importance of each feature is calculated by measuring the decrease in accuracy when that feature is removed from the model. This information can be used to select the most important features for the model.

Random Forest's parameters

RandomForestClassifier(n_estimators = 100, *, criterion = 'gini', max_depth = None, min_samples_split = 2, min_samples_leaf = 1, max_features = 'sqrt'. bootstrap = True)

- **n_estimators**: Number of trees in the forest.
- **max_features**: Maximum number of features considered for splitting a node.
- **max_depth**: Maximum number of levels in each decision tree.
- **min_samples_split**: Minimum number of data points placed in a node before the node is split.
- **min_samples_leaf**: Minimum number of data points allowed in a leaf node.
- **bootstrap**: Method for sampling data points (with or without replacement).

Advantages of Random Forest

- Reduces overfitting compared to Decision Tree.
- Can handle high dimensional data.
- Can handle nonlinear relationships between features.
- Can be used for both classification and regression problems.
- Provides a measure of feature importance.

Disadvantages of Random Forest

- Can be computationally expensive, especially with large datasets.
- Can be difficult to interpret compared to decision trees.
- Can be sensitive to the quality of the data.
- May not perform as well as other algorithms on certain datasets.

D. XGBoost

XGBoost, or Extreme Gradient Boosting, is a powerful machine learning algorithm that uses Decision Trees as base learners to predict a target variable. It is an ensemble method that combines multiple Decision Trees to improve the accuracy of predictions.

Like other boosting algorithms, XGBoost works by iteratively adding new Decision Trees to the model, with each new tree attempting to correct the mistakes of the previous trees. The final prediction is made by aggregating the predictions of all the trees in the model.

Features of XGBoost

- Parallelization of tree construction using all of your CPU cores during training. Collecting statistics for each column can be parallelized, giving us a parallel algorithm for split finding.
- Cache-aware Access: XGBoost has been designed to make optimal use of hardware. This is done by allocating internal buffers in each thread, where the gradient statistics can be stored.
- Blocks for Out-of-core Computation for very large datasets that don't fit into memory.
- Distributed Computing for training very large models using a cluster of machines.
- Column Block for Parallel Learning. The most time-consuming part of tree learning is to get the data into sorted order. In order to reduce the cost of sorting, the data is stored in the column blocks in sorted order in compressed format.

XGBoost works in the following stages:

1) Initialize the model with a single decision tree.
2) Calculate the errors of the initial model on the training data.
3) For each new iteration, add a new decision tree to the model to correct the errors of the previous trees.
4) Update the weights of the training instances to give more weight to the instances that were misclassified in the previous iterations.
5) Repeat steps 2-4 until the desired number of trees is reached or the model reaches its optimal performance.

XGBoost's parameters

XGBClassifier(learning_rate = 0.1, n_estimators = 100, max_depth = 3)

- **learning_rate**: Controls the learning rate or step size shrinkage. Smaller values make the model more robust but require more boosting iterations. Typical values range from 0.01 to 0.3.
- **n_estimators**: The number of boosting rounds or trees to build. This parameter specifies the number of iterations at which boosting is stopped. Higher values can lead to overfitting, so it should be tuned carefully.
- **max_depth**: The maximum depth of each tree. It controls the complexity of the individual trees and can help prevent overfitting. Typical values range from 3 to 10, but it depends on the dataset.
- **subsample**: The subsample ratio of the training instances. It controls the fraction of data to be used for each tree. Smaller values make the model more conservative and reduce overfitting. Typical values range from 0.5 to 1.

Advantages of XGBoost

- Can handle missing values and noisy data.

- Has built-in regularization to prevent overfitting.
- Can handle both classification and regression problems.
- Often performs well in machine learning competitions.
- Provides a measure of feature importance.

Disadvantages of XGBoost

- Can be computationally expensive, especially with large datasets.
- Can be difficult to interpret compared to simpler models like linear regression.
- Requires careful tuning of hyperparameters to achieve optimal performance.

III. METHODS

A. Train-Test Split

The Train-Test Split is a technique for evaluating the performance of a machine learning algorithm. It can be used for classification or regression problems and can be used for any supervised learning algorithm.

The procedure involves taking a dataset and dividing it into two subsets. The first subset is used to fit the model and is referred to as the training dataset. The second subset is not used to train the model; instead, the input element of the dataset is provided to the model, then predictions are made and compared to the expected values. This second dataset is referred to as the test dataset.

- **Train Dataset:** Used to fit the machine learning model.
- **Test Dataset:** Used to evaluate the fit machine learning model.

The objective is to estimate the performance of the machine learning model on new data: data not used to train the model. This is how we expect to use the model in practice. Namely, to fit it on available data with known inputs and outputs, then make predictions on new examples in the future where we do not have the expected output or target values. The train-test procedure is appropriate when there is a sufficiently large dataset available.

B. F1-Score

The F1-Score is a commonly used metric in classification tasks, particularly when dealing with imbalanced datasets. It combines precision and recall into a single value, providing a balanced measure of a model's performance. To understand the F1 score, we first need to define precision and recall:

- **Precision:** It measures the proportion of correctly predicted positive instances out of the total instances predicted as positive. It indicates how well the model identifies true positives while minimizing false positives. Precision is calculated as:
$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$
- **Recall:** It measures the proportion of correctly predicted positive instances out of the total actual positive instances. It indicates how well the model captures all the positive instances without missing any. Recall is calculated as:
$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives}$$

Precision and Recall are the two building blocks of the F1 score. The goal of the F1 score is to combine the precision and recall metrics into a single metric. At the same time, the F1 score has been designed to work well on imbalanced data. It is calculated using the following formula:

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

C. Min Max Scaler

MinMaxScaler is a popular data normalization technique used in machine learning to scale numeric features to a specific range. It transforms the data so that it falls within a predefined interval, typically between 0 and 1. MinMaxScaler is particularly useful when the features have different scales and ranges, as it brings them all to a common scale, making them more comparable and suitable for many machine learning algorithms.

The theory behind MinMaxScaler is based on the linear transformation of data using minimum and maximum values. The formula for MinMaxScaler is as follows:

$$X_{scaled} = \frac{X - X_{min}}{X_{max} \times X_{min}}$$

In this formula, X represents the original feature values, X_{min} is the minimum value of the feature, and X_{max} is the maximum value of the feature. The numerator subtracts the minimum value from each data point, and the denominator calculates the range of the feature. Dividing the numerator by the denominator scales the feature values to the desired range.

MinMaxScaler applies the above formula to each feature independently, ensuring that all features are transformed to the range of 0 to 1. The transformed values maintain the relative relationships and proportions among the original data points, preserving the structure of the dataset.

MinMaxScaler is widely used in various machine learning tasks, such as regression, clustering, and neural networks. It not only standardizes the range of features but also helps prevent certain features from dominating the learning process due to their larger scales. By normalizing the data, MinMaxScaler facilitates better convergence and performance of machine learning models.

D. ADASYN

Adaptive Synthetic Sampling (ADASYN) is a popular technique used to address the issue of imbalanced classification in machine learning. In imbalanced datasets, the number of samples in different classes is significantly uneven, making it challenging for models to accurately predict the minority class. ADASYN aims to balance the dataset by generating synthetic samples specifically for the minority class.

The theory behind ADASYN involves assigning weights to individual minority class samples, indicating their level of difficulty in learning. The higher the weight, the more challenging it is for the model to correctly classify that particular sample. ADASYN calculates the weight of each minority class sample using the following formula:

$$w_i = \frac{n_{majority}}{n_{neighbors}(i) \times n_{minority}}$$

Here, w_i represents the weight assigned to the $i - th$ minority class sample, $n_{majority}$ is the number of majority class samples, is the number of majority class samples, $n_{neighbors}(i)$ is the number of minority class samples among the k nearest neighbors of the i-th sample, and $n_{minority}$ is the total number of minority class samples.

Once the weights are assigned, ADASYN generates synthetic samples by interpolating features between a randomly selected minority class sample and its nearest neighbors. The synthetic samples are created iteratively, with more emphasis on generating samples in regions where they are most needed, as determined by the assigned weights.

The iterative process continues until the desired balance ratio between the minority and majority classes is achieved. ADASYN offers a data-driven approach to address imbalanced classification problems, effectively improving the performance of machine learning models in scenarios where imbalanced datasets are prevalent.

E. Bayesian Optimization

Bayesian optimization is a technique used for optimizing the performance of machine learning models. It works by iteratively selecting the best hyperparameters for the model based on the results of previous iterations. The core idea behind Bayesian optimization is to use a probabilistic model to estimate the performance of the model with different hyperparameters. The formula for Bayesian optimization is as follows:

$$x_{t+1} = \arg\max_{x \in \mathcal{X}} \mathbb{E}_{y \sim p(y|x_{1:t}, y_{1:t})}[y]$$

where $x_{1:t}$ and $y_{1:t}$ represent the hyperparameters and corresponding function values evaluated in previous iterations, $\mathcal{X}$ is the hyperparameter search space, and $p(y|x_{1:t}, y_{1:t})$ is the posterior distribution over the function values given the observed hyperparameters and function values. The acquisition function is used to balance the exploration-exploitation trade-off and to determine the next set of hyperparameters to evaluate.

The process of Bayesian optimization is as follows:

1) Define the search space for hyperparameters.
2) Choose an acquisition function to determine the next set of hyperparameters to evaluate.
3) Evaluate the model with the selected hyperparameters and update the probabilistic model.
4) Repeat steps 2 and 3 until the desired performance is achieved or a maximum number of iterations is reached.

Bayesian Optimization has been shown to be effective in a variety of machine learning tasks, including hyperparameter tuning, neural architecture search, and feature selection. It is particularly useful when the evaluation of the model is expensive, as it allows for efficient exploration of the hyperparameter space.

In conclusion, Bayesian Optimization is a powerful technique for optimizing machine learning models. It allows for efficient exploration of the hyperparameter space and has been shown to be effective in a variety of tasks.

IV. EXPERIMENTS

The provided dataset, named ”The Flight Delay and Cancellation,” compiles information from various air carriers, encompassing departure delays, arrival delays, scheduled departures, and other related details. The dataset sources its information from the Bureau of Transportation Statistics, which falls under the purview of the Department of Transportation (DOT).

Dataset:https://www.kaggle.com/datasets/usdot/flight-delays?datasetId=810sortBy=voteCountselect=airlines.csv

A. Overview of the dataset

Each entry of the *flights.csv* corresponds to a flight and we see that more than 5'800'000 flights have been recorded in 2015. These flights are described according to 31 variables.

flights.csv has 31 features (columns) and 5819079 rows

Description of the attributes involved in the dataset

- **YEAR, MONTH, DAY, DAY_OF_WEEK:** dates of the flight.
- **AIRLINE:** An identification number assigned by US DOT to identify a unique airline.
- **ORIGIN_AIRPORT** and **DESTINATION_AIRPORT:** code attributed by IATA to identify the airports.
- **SCHEDULED_DEPARTURE** and **SCHEDULED_ARRIVAL :** scheduled times of take-off and landing.
- **DEPARTURE_TIME** and **ARRIVAL_TIME:** real times at which take-off and landing took place.
- **DEPARTURE_DELAY** and **ARRIVAL_DELAY:** difference (in minutes) between planned and real times.
- **DISTANCE:** distance (in miles).

An additional file of this dataset, the *airports.csv* (322 rows × 7 columns) file gives a more exhaustive description of the airports and the *airlines.csv* (13 rows x 2 columns) file contains airline name and company abbreviation code.

B. Preprocessing

To expedite the process, we exclusively utilized data from the month of January. This decision is made due to the substantial size of the original dataset, which consisted of 5,819,079 rows and 31 columns. By narrowing our focus, we were able to reduce the dataset to a more manageable size of 469,968 rows and 31 columns.

1) Convert datatime
   - Convert **YEAR, MONTH, DAY** into **DATE**

```
0    2015-01-01
1    2015-01-01
2    2015-01-01
...        ...
Name: DATE, Length: 469968
```

- Convert time into **hour:min:second** format

SCHEDULED_ARRIVAL	DEPARTURE_TIME
04:30:00	23:54:00
07:50:00	00:02:00
08:06:00	00:18:00

2) Drop unnecessary columns
   - Features such as **TAXI_OUT**, **TAXI_IN**, **WHEELS_ON**, **WHEELS_OFF**, and others are found to have no significant contribution to the prediction task and are subsequently dropped from the dataset.
   - Drop **YEAR**, **MONTH**, **DAY**, **DAY_OF_WEEK**, and **DATE** columns because they have been previously converted into the `hour:min:second` format.
   - Drop features that are irrelevant for predictions and contain numerous null values.

variable	missing values
ARRIVAL_DELAY	12955
ELAPSED_TIME	12955
ARRIVAL_TIME	12271
DEPARTURE_TIME	11657
DEPARTURE_DELAY	11657

3) Accounting for destinations model
   - Flights with different destinations were previously grouped together as soon as they departed simultaneously. In addition, we have developed a model that takes into account both the departure and arrival times.
   - To prepare for classification, we create decision features **DELAY** based on **DEPARTURE_DELAY**. If **DEPARTURE_DELAY** $<$ 5 mins, the label is 0 (on time). The remaining values are labeled as 1 (delay).

4) Data augmentation

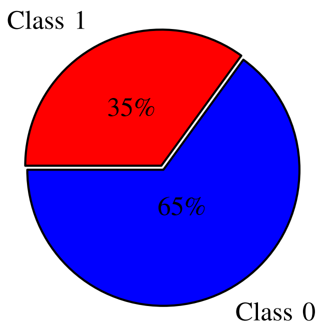


Fig. 1. Distribution of classes

Figure 1 illustrates the prevalence of class 0 over class 1 in the dataset, indicating a significant imbalance in class distribution. Therefore, to ensure the accuracy of our model is not skewed by imbalances between class 1 and class 0, we implemented the ADASYN (Adaptive Synthetic) technique to augment our dataset. By employing ADASYN, we aim to create a more balanced and representative dataset that enables our model to make accurate predictions across both classes.

5) Train test split

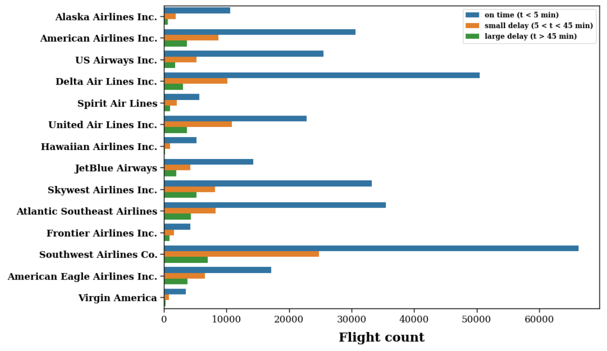
Given that the data has a high dimensionality, we will utilize the train-test split approach to divide it into two parts: a training set and a testing set. The split will be performed with a ratio of 70% for the training set and 30% for the testing set. This will enable us to evaluate our model's performance on unseen data.

6) Normalize

In order to preprocess the data and ensure uniformity, a two-step transformation is applied.

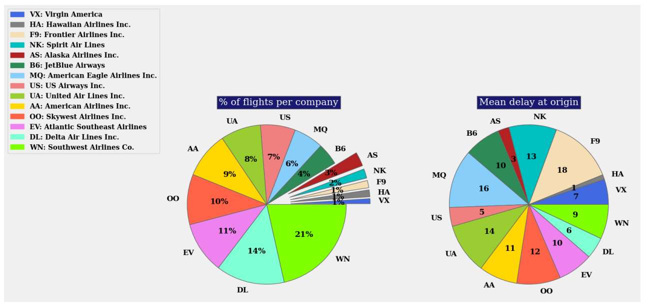
   - Firstly, the **LabelEncoder()** is employed to encode columns containing string values, exemplified by **ORIGIN_AIRPORT**, into numerical labels. This encoding process assigns unique numeric representations to each distinct string value within the column.
   - Secondly, the **MinMaxScaler()** is utilized to scale the remaining columns that comprise numeric values. By rescaling the numeric data to a common range, typically between 0 and 1, the data points are brought to a consistent scale, enabling fair comparisons between the different features. This combined transformation, involving label encoding and scaling, aims to facilitate the subsequent analysis or modeling tasks that may require standardized and compatible input data.

C. *Visualization*



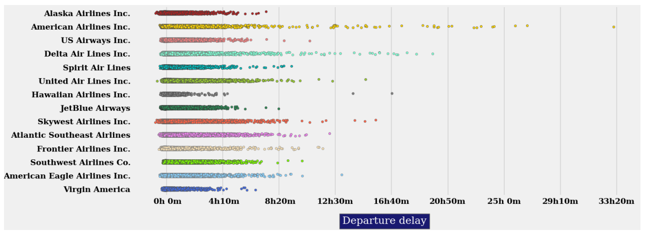
Graph 1: The graph shows the correlation of delay time between distinct airlines in January.

Comment: According to this graph, it is obvious that almost airlines companies have a significantly higher on-time flights than delay flights. Furthermore, Southwest Airlines has the highest number of on-time flights while the Virgin America hits the lowest point.



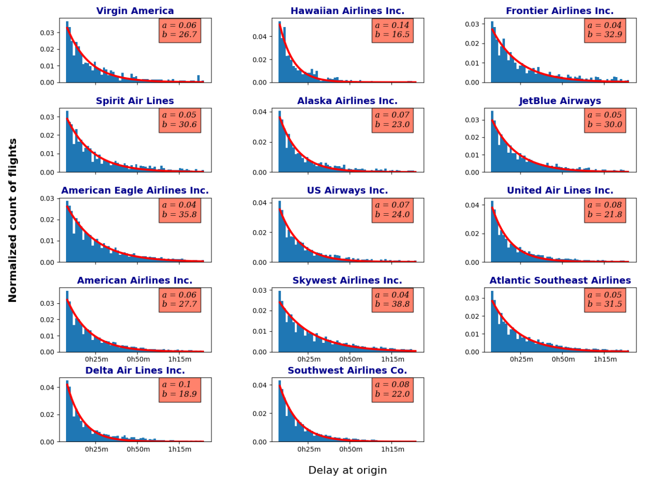
Graph 2: The pie chart gives the percentage of flights per airline and the mean delay time of each company in January.

Comment: Overall, the Southwest Airlines accounts for approximately one-fifth of the total. Moreover, the average figure for mean delay times oscillates from 6 to 11 minutes while Hawaiian Airlines and Alaska Airlines reported extremely low mean delays.



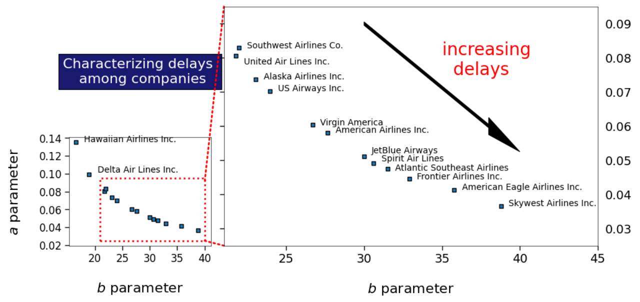
Graph 3: The graph shows the departure delay time of each airline in January.

Comment: The delay time is mostly under 4 hours. However, we see that occasionally, we can face really large delays that can reach a few tens of hours.



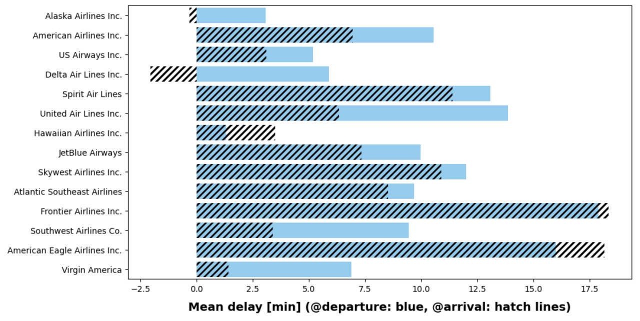
Graph 4: The graph shows the normalised distribution of delay in January.

Comment: This figure displays the normalized distribution of delays that we modeled using an exponential distribution, specifically $f(x) = a \cdot e^{-\frac{x}{b}}$. By analyzing the values of either 'a' or 'b', we can establish a ranking of the companies. Airlines with low values of 'b' will have a higher proportion of significant delays, while airlines with high values of 'a' will be known for their punctuality. A clearer explanation of this relationship is provided in the subsequent graph.



Graph 5: The graph shows characterizing delays among companies in January.

Comment: The left panel of this figure provides an overview of the coefficients 'a' and 'b' for the 14 airlines, revealing that Hawaiian Airlines and Delta Airlines hold the top two positions. The right panel focuses on 12 additional airlines. It is evident that SouthWest Airlines, representing approximately 20% of the total number of flights, is well-ranked and occupies the third position. According to this ranking, SkyWest Airlines is identified as the worst carrier.



Graph 6: The graph shows the difference between departure delay and arrival delay in January.

Comment: On this figure, we can see that delays at arrival are generally lower than at departure. This indicates that airlines adjust their flight speed in order to reduce the delays at arrival.

D. Analyze experimental results

TABLE I
EXPERIMENTAL RESULTS

Algorithm	F1-Score	
	Normal	Optimize
KNN	0.71	0.72
DT	0.59	0.68
RF	0.64	0.71
XGB	**0.73**	**0.75**

The results are presented in Table I, showcasing the F1-scores for each algorithm. The algorithms evaluated include K-Nearest Neighbors (KNN), Decision Tree (DT), Random Forest (RF), and XGBoost (XGB).

In the normal state, the algorithm with the highest F1-score is XGBoost (0.73), followed closely by KNN (0.71), RF (0.64), and DT (0.59). These scores provide an initial indication of the algorithms' performance without any optimization.

After applying optimization techniques to fine-tune the algorithms, noticeable improvements in the F1-scores were observed. XGBoost achieved the highest F1-score (0.75) in the optimized state, demonstrating the effectiveness of the optimization process. KNN also experienced a slight improvement in its F1-score, reaching 0.72. The optimized RF algorithm achieved an F1-score of 0.71, while DT demonstrated a substantial improvement, reaching an F1-score of 0.68.

These results highlight the importance of algorithm optimization in enhancing model performance. By carefully fine-tuning the algorithms, we were able to achieve higher F1-scores, indicating improved precision and recall.

It is important to note that these F1-scores provide an evaluation of the algorithms' performance specifically in the context of the dataset and evaluation metrics used. Further analysis and experimentation may be required to determine the most suitable algorithm for a specific task or dataset.

Overall, the evaluation of the F1-scores in both the normal and optimized states provides valuable insights into the performance of the algorithms and underscores the significance of optimization in enhancing their effectiveness.

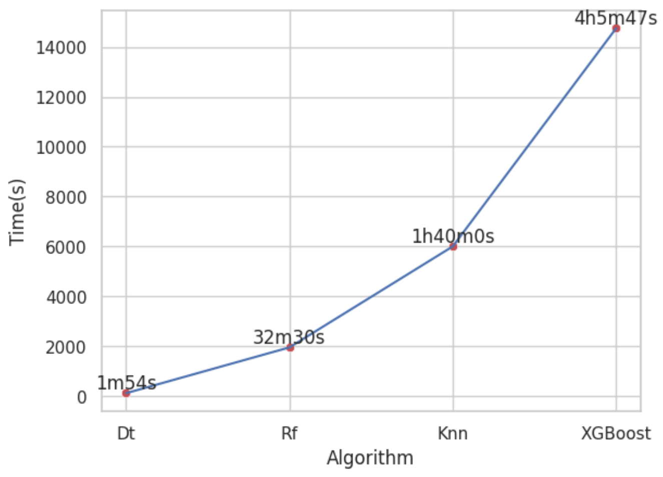


Fig. 2. Optimized time of each algorithm

Figure 2 illustrates the execution time of each algorithm obtained through the utilization of Bayesian Optimizer. The findings indicate that XGBoost necessitates a substantial tuning time, but it yields the highest performance. On the other hand, Decision Tree demonstrates a shorter tuning time while showcasing a notable improvement in accuracy.

Contrary to other algorithms, Decision Tree demonstrates a shorter tuning time of just 1 minute. Despite this, it manages to achieve a substantial improvement in accuracy, approximately 10%. Random Forest, being an ensemble method, involves multiple decision trees and thus requires a longer tuning time of 32 minutes and 30 seconds. However, it is known for its high efficiency and often yields excellent results.

KNN, on the other hand, demands more computational resources and time due to its distance-based computations. With a tuning time of 1 hour and 40 minutes, KNN may be relatively slower in terms of tuning.

XGBoost, while taking the longest time to tune at 4 hours, delivers the highest performance among the mentioned algorithms. The additional time spent on tuning is justified by the improved accuracy and predictive power of XGBoost.

It is crucial to consider the trade-off between tuning time and performance when selecting a machine learning algorithm. While XGBoost may require a longer tuning time, it provides the highest accuracy. On the other hand, Decision Tree exhibits a short tuning time with a notable improvement in performance.

V. CONCLUSION

Through our extensive project, we have successfully created a highly sophisticated machine learning model specifically designed to predict flight departure delays. Among the various models we evaluated, the XGBoost algorithm emerged as the best performer, showcasing remarkable F1-Score.

Our model achieved an impressive F1-Score rate of 75% in accurately identifying flight departure delays. This indicates its remarkable ability to forecast potential delays and provide valuable insights for proactive planning and decision-making. By harnessing the power of machine learning, we have unlocked a valuable resource that can significantly improve the efficiency and reliability of air travel.

In our in-depth analysis, we uncovered a compelling relationship between flight delays and the departure airports. Notably, we observed a strong correlation between the busyness and significance of an airport and the likelihood of experiencing delays. Major airports with high passenger volumes and extensive air traffic tend to encounter a greater number of delays compared to smaller, less congested airports. This finding provides critical insights into the factors influencing flight punctuality and offers valuable guidance for optimizing airport operations.

The implications of this analysis are profound. By identifying the airports most prone to delays, we can focus our attention and resources on implementing targeted strategies to improve their efficiency and minimize disruptions. This includes optimizing scheduling practices, streamlining ground operations, and enhancing overall infrastructure to ensure smoother departures and arrivals.

By reducing departure delays, we can enhance the overall travel experience for passengers, improve customer satisfaction, and bolster the reputation of both airlines and airports alike. Moreover, a well-managed and punctual flight system contributes to economic growth, as it facilitates seamless business transactions, encourages tourism, and fosters global connectivity.

In summary, our project introduces an advanced machine learning model for accurately predicting flight departure delays. By analyzing the correlation between delays and departure airports, we can strategically enhance airport efficiency and improve the overall travel experience.