PHENIKAA UNIVERSITY
FACULTY OF COMPUTER SCIENCE

Big Data Integration and Analysis

Final Report

Flight delay prediction using PySpark on AWS Cloud

Giảng viên hướng dẫn: Pham Tien Lam, PhD
Pham Kim Thanh, MSc

Sinh viên: Nguyen Duc Hai 21010560
Nguyen Duc Duy Anh 21011488

HA NOI CITY, JULY 2024

Contents

List of Figures

List of Tables

Listings

1 Introduction

1.1 Abstract

Air travel is one of the cornerstones of global transportation, facilitating the movement of people and goods across vast distances. However, flight delays pose a major issue: disrupting schedules, causing passengers' inconvenience and incurring substantial economic costs for airlines and other affiliated sides. Efficient flight delay prediction can greatly enhance operational efficiency, improve customer satisfaction and optimize resource allocation.

This project aims to explore and exploit the power of big data processing and machine learning frameworks to develop a speedy and highly precise model for forecasting flight delays. The foundation of our used techniques is PySpark, the Python API of Apache Spark[4]; which provides an ideal framework for handling large datasets and performing complex computations required for this task.

The authors hope that this project shall contribute to helping airlines and passengers make informed decisions, leading to improved efficiency and reduced disruptions in air travel. [1]

1.2 Prevailing methods

Flight delay prediction has been a subject of extensive research, and several approaches have been employed to tackle this problem.

- **Statistical methods:** economic models like ARIMA (Autoregressive Integrated Moving Average) model are used for forecasting delays based on historical delay data. These models are particularly useful for capturing seasonality.
- **Feature engineering:** incorporating weather condition data, information on air traffic density / congestion at airports can significantly improve the accuracy of predictions and provides context for forecasting processes.
- **Big data frameworks:** As flight data is usually massive in size, it requires specific frameworks. Apache Hadoop and Spark are widely used for building and deploying large-scale data analytics.

[1] https://github.com/SkyjackerFWO/Flight-Delay-with-PySpark

- **AI/ML models:** AI-fueled prediction usually boasts higher efficiency and speed. The integration of AI/ML models into flight data forecasting has shown great promise.

1.3 Dataset overview[3]

The dataset we opted to use was collected as part of the AWS Academy's Machine Learning Foundations[1] course. This dataset contains scheduled and actual departure and arrival times reported by certified US air carriers that account for at least 1 percent of domestic scheduled passenger revenues. The data was collected by the U.S. Office of Airline Information, Bureau of Transportation Statistics (BTS). The dataset contains date, time, origin, destination, airline, distance, and delay status of flights for flights between 2013 and 2018.

1.4 Question Formulation

Delayed flights are defined by the Federal Aviation Administration of the United States (FAA) as 15 minutes later than its scheduled time. The aim of our study is to predict departure / arrival delay as a binary yes/no feature, or whether the flights depart / arrive 15 minutes or more later than the scheduled time (referred to as the `is_delay` variable in the dataset).

1.5 Evaluation Metrics

To evaluate the performance of a model, there are several different evaluation metrics that can be utilized, such as

$$Precision = \frac{tp}{tp + fp} \tag{1.1a}$$

$$Recall = \frac{tp}{tp + fn} \tag{1.1b}$$

$$Accuracy = \frac{tp + tn}{tp + fp + tn + fn} \tag{1.1c}$$

For our case, we chose to place an emphasis on improving recall (calculated as the percentage of correct predictions for the positive class out of all positive predictions made by the model). Our hypothesis is that the cost resulting from not identifying a potentially delayed flight is higher for the passenger as well as the airline, than the cost associated with failing to predict a delayed flight.

Consequently, we would like to take a balanced approach that aligns with our analysis of the problem. The metric we chose assess the model's performance, therefore, is the F-beta measure.

$$F_\beta = \frac{(1+\beta^2)(precision \cdot recall)}{\beta^2 \cdot (precision \cdot recall)} \tag{1.2}$$

We chose the β value to 2.0, which gives more weight to recall than precision in the calculation of the overall score.

1.6 Data framework

For the deep analysis and forecasting of flight delay, Apache Spark[4] is regarded as one of the best options. But what it is, actually ?

Spark is a unified analytics engine for large-scale data processing, providing high-level APIs in a various programming languages such as Java, Python and R. It is an optimized engine that supports general computation graphs for data analysis, and also support a rich set of higher-level tools for data processing including Spark SQL for SQL and DataFrames, pandas APIs on Spark for pandas-required workloads, MLib for machine learning, etc.

Other Python packages include: numpy for basic data processing, matplotlib and plotly for high-level data visualization.

2 AWS Workspace setup

2.1 Motivation

Amazon Elastic MapReduce (EMR) is a cloud service offered by Amazon Web Services (AWS)[1] that makes it easy to run large data applications on Hadoop, Spark, Hive, Presto, HBase, and more. In our case, we want to use EMR to create and manage Spark clusters without installing and configuring software. In addition, we want to make the most of the course's knowledge by using cloud systems to perform tasks related to big data (specifically, cloud systems come from AWS). AWS S3 is also the service used in the project. S3 makes storing large data easier. Through S3 URI, we can also easily read data through the Pandas library.

2.2 Setup

2.2.1 Sandbox Environment

AWS Canvas provides us with a sandbox environment within the Data Engineering course. The Sandbox environment will not charge us for AWS resource usage, but will instead limit the resources used and the usage time (3 hours). Therefore, this setup is only for testing purposes, and the final results will be processed after we have completed the test run of the code locally. Starting the Sandbox environment is similar to the process followed for regular lab exercises. However, there is an important distinction: the vokey lab key provided will not be functional for SSH access. Therefore, it is necessary to generate a new Keypair for the primary instance.

2.2.2 AWS EMR

To create an AWS EMR cluster, follow these steps:

1. Access the AWS Management Console.
   Connect to the AWS Management Console by choosing the **AWS** (green light turn) link above the terminal window. In the search bar at the top of the screen, type "EMR" and select Amazon Elastic MapReduce.

2. Choose a cluster type. On the Amazon Elastic MapReduce page, click Create Cluster and select the type of cluster want to create (we use EC2 Instance)

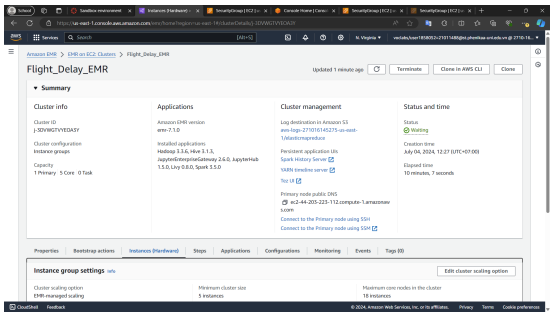
3. Configure your cluster. Provide the following information for your cluster:

   - Cluster name: A name for your cluster.
   - Master instance type: The instance type for the cluster's master node.
   - Worker instance type: The instance type for the cluster's worker nodes.
   - Number of workers: The number of worker nodes for the cluster.
   - Storage: The storage type and size for the cluster.
   - Networking: The network configuration for the cluster.
   - Applications: The applications that you want to install on the cluster.
   - Security: The security configuration for the cluster.

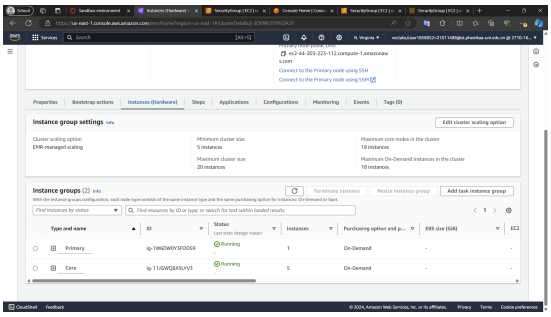
4. Review and create cluster.

5. Wait for your cluster to be created. The cluster creation process may take a few minutes.

6. Connect to your cluster. Once the cluster is created, connect to it using SSH or the EMR web UI.



(a) EMR config 1



(b) EMR config 2

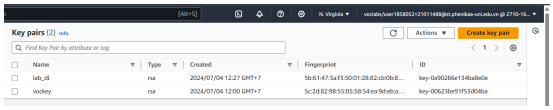
Figure 2.1: AWS EMR setup

2.2.3 SSH with Visual Studio Code

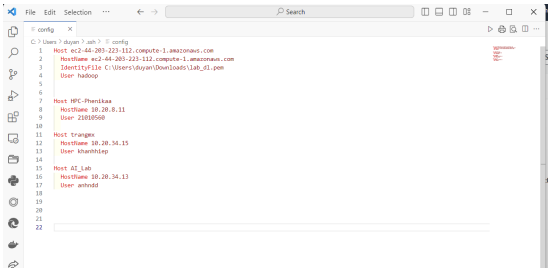
Connecting via SSH on Visual Studio Code[2] can be found here. The SSH command is as follows:

```
ssh -i "~/labsuser.pem" hadoop@ec2-3-237-175-100.compute-1.amazonaws.com
```

Alternatively, we can directly modify the ".ssh/config" file as shown in the image.



(a) Create keypair in EC2 Primary for SSH



(b) Modify the ".ssh/config" file

Figure 2.2: Setup SSH for remote in Visual Studio Code

Once connected, you can use it in the same way as you would on your local machine. Additionally, since the group uses Jupyter notebooks for work, we need to install a few extensions to make it easier to use.

2.2.4 SparkSession setup

SparkSession is a central class in Apache Spark, playing a pivotal role in interacting with Spark and processing data. It offers a comprehensive set of features for initializing SparkContext, accessing DataFrame and SQL, executing Spark tasks, and configuring the Spark environment. To run pyspark on Jupiter notebook, we need to set up as follows

```
spark = (
    SparkSession.builder
        .appName("Flight_Delay")
        .config("spark.dynamicAllocation.enabled", True)
        .config("spark.dynamicAllocation.minExecutors", 5)
        .config("spark.dynamicAllocation.maxExecutors", 20)
        .config("spark.executor.cores", 1)
        .config("spark.executor.instances", 18)
        .config("spark.driver.memory", "10g")
        .config("spark.executor.memory", "4g")
        .config("spark.sql.execution.arrow.pyspark.enabled", True)
        .config("spark.memory.offHeap.enabled", "true")
        .config("spark.memory.offHeap.size", "10g")
        .getOrCreate()
)
```

1. **Spark Session Creation:** This line creates a SparkSession object, the entry point for Spark applications.The .appName("Flight Delay") sets the application name for identification purposes

2. **Dynamic Allocation:** enables dynamic allocation of executors, allowing Spark to adjust the number of executors based on workload.

3. **Minimum Executors:** sets the minimum number of executors to allocate initially (5 in this case).

4. **Maximum Executors:** sets the upper limit for the number of executors that can be allocated (20 in this case).

5. **Executor Cores:** assigns 1 core to each executor in the cluster.

6. **Executor Instances:** creates a cluster with 18 executor instances, each with 1 core.

7. **Driver Memory:** allocates 10 GB of memory to the Spark driver.
8. **Executor Memory:** assigns 4 GB of memory to each executor for processing tasks.
9. **Apache Arrow:** enables Apache Arrow data serialization for potential performance improvements.
10. **Off-Heap Memory:**
    - ‘.config("spark.memory.offHeap.enabled", "true")‘ allows Spark to use off-heap memory in addition to on-heap memory.
    - ‘.config("spark.memory.offHeap.size", "10g")‘ sets the size of the off-heap memory pool to 10 GB.
11. **SparkSession Retrieval:** ‘.getOrCreate())‘ retrieves an existing SparkSession if it exists, or creates a new one if not.

3 Exploratory Data Analysis (EDA)

3.1 Preliminary data information

The dataset consists of several different features of flights between 2013-2018. We shall use a reduced version of this dataset, which contains only:

- Main airports: ATL (Atlanta), ORD (O’Hare), DFW (Dallas-Fort Worth), DEN (Denver), CLT (Charlotte Douglas), LAX (Los Angeles), IAH (George Bush), PHX (Phoenix Sky Harbor), SFO (San Francisco).
- Top five airlines: UA (United Airlines), OO (SkyWest Airlines), WN (Southwest Airlines), AA (American Airlines), DL (Delta Airlines).
- 20 features and 1635590 entries.
- The `Diverted` and `Cancelled` features are 0-valued (all of the entries are 0s).

The feature dictionary for the dataset is given in Fig 3.1.

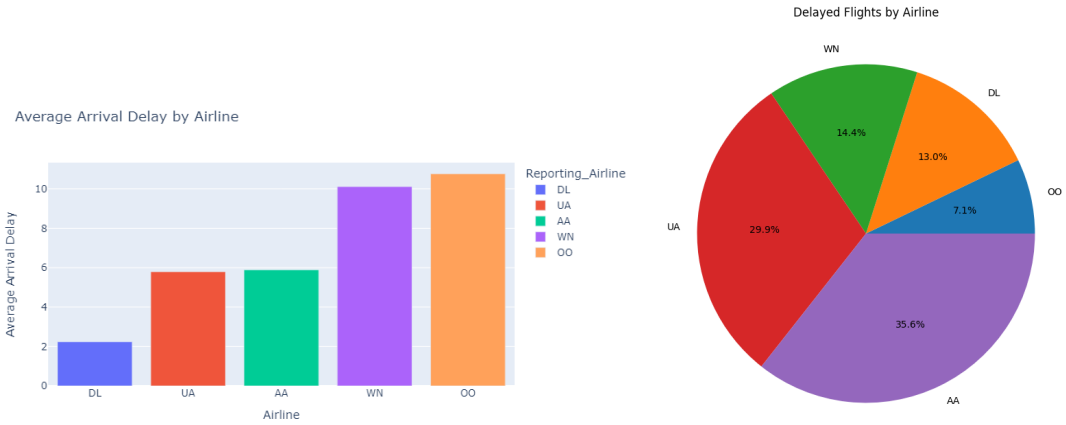
Table 3.1: Dataset feature dictionary

Name	Content	Type
is_delay	whether the flight is delayed or not	64-bit float
Year	year	32-bit integer
Quarter	quarter of the year	32-bit integer
Month	month	32-bit integer
DayOfMonth	day of corresponding month	32-bit integer
DayOfWeek	day of corresponding week	32-bit integer
FlightDate	the exact date of the flight	string
Reporting_Airline	airline conducting the flight	string
Origin	departure airport	string
OriginState	state in which the departure airport is located	string
Dest	destination airport	string
DestState	state in which the destination airport is located	string
CRSDepTime	CRS Departure Time (local time: hhmm)	32-bit integer
Cancelled	whether the flight is cancelled or not	64-bit float
Diverted	whether the flight is diverted or not	64-bit float
Distance	travelling distance	64-bit float
DistanceGroup	Distance Intervals, every 250 Miles, for Flight Segment	32-bit integer
ArrDelay	Difference in minutes between scheduled and actual arrival time	64-bit float
ArrDelayMinutes	similar to ArrDelay, but is equal to 0 if the flight is ahead of schedule	64-bit float
AirTime	Duration of the flight	64-bit float

3.2 In-depth Analysis

First, we may want to look at how airlines and airports correlate to flights delay. We can observe from Fig 3.1a that SkyWest's service has been delayed the most, while Delta proves to be the best choice in terms of limiting delay time.

Meanwhile, the delay time reported by airports shows sign of similarity as in Fig 3.2a, with only a few exceptions such as San Francisco as destination airport : nearly 20 minutes on average, 3 minutes higher than the second one - Los Angeles. Both of them are located within California, indicating high air traffic volume and congestion in this state. O'Hare (Chicago) as origin airport is another example (approximately 17 minutes). It is understandable, as Chicago is one of the most populous cities in the USA. On the other hand; Charlotte, Phoenix and Atlanta enjoy relatively low delay time on average.



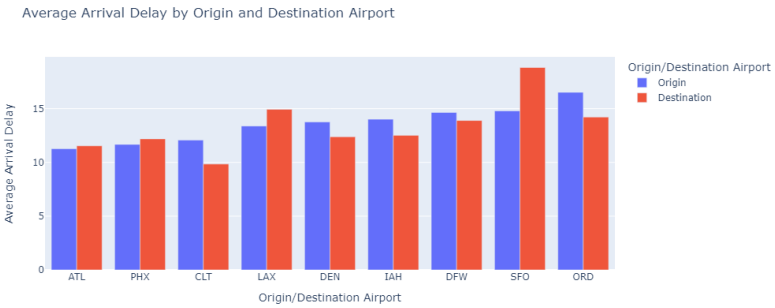
(a) Average arrival delay by airlines

(b) Delayed flights by airlines

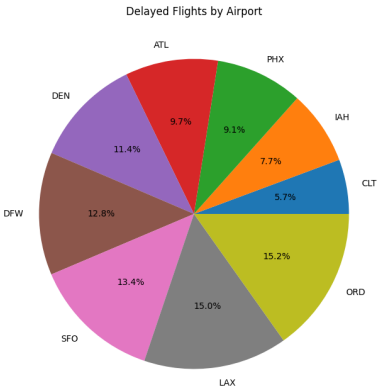
Figure 3.1: Flight delay statistics regarding airlines

Fig 3.3a illustrates the tendency to be delayed longer of flights with shorter distance. As the travel distance increases, delay time plummets from a maximum of more than 10 minutes (distance group 2) to only more than 1 minute (distance group 9). This may mean that longer flights are prioritized, but not necessarily the situation.

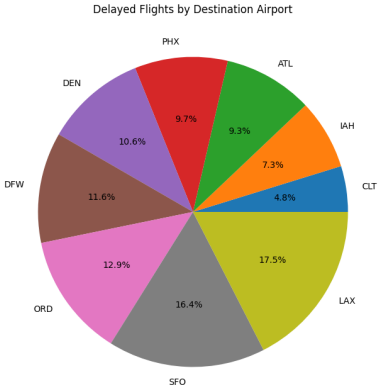
Fig 3.3b shows the rise of need for air travel in the mid-year months, leading to worse traffic and longer delay time. The outstandingly high delay time in December (roughly 15 minutes on average) is primarily due to Christmas taking place.



(a) Average arrival delay by airports

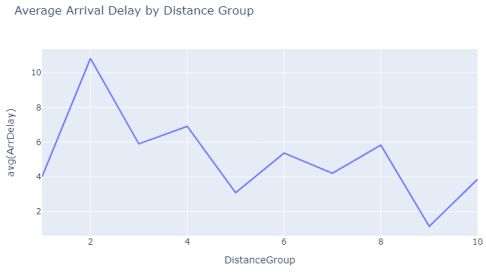


(b) Delayed flights by origin airport

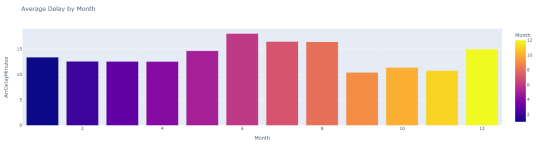


(c) Delayed flights by destination airport

Figure 3.2: Flight delay statistics regarding airports



(a) Average arrival delay by distance group



(b) average arrival delay by month

Figure 3.3: Flight delay statistics regarding distance and months

4 Working with PySpark ML

4.1 Data preprocessing

First, we remove the `Diverted` and `Cancelled` features, as they are zero-valued vectors and therefore do not contribute much to our analysis. We also encode the string-valued variables to numerical form using indexing method. We then examine the correlation between numerical variables in the dataset and decide whether further feature engineering is needed.

It can easily be seen that the features are relatively independent, except for some designed-to-be-related sets such as {`Distance`, `DistanceGroup`} or {`Quarter`, `Month`}. However, the clear relation between our target variable and `ArrDelay` and `ArrDelayMinutes` can be detrimental to the prediction process, so we need to remove them from the dataset. After all, we would like know whether a flight could be late or not, not WAS late or not.

To enhance efficiency, we use normalization for high-valued variables in the dataset, namely `Year`, `CRSDepTime`, `Distance` and `AirTime`. Our choice is standardization, which scales values to unit variance and zero mean. Standardization of a feature $\mathcal{X}$ is conducted with the following formula:

$$\mathcal{X} = \frac{\mathcal{X} - \mathcal{X}_{mean}}{\mathcal{X}_{std}} \tag{4.1}$$

with $\mathcal{X}_{mean}$ and $\mathcal{X}_{std}$ being the original mean and standard deviation of training samples within $\mathcal{X}$, respectively.

4.2 Deploying ML models

After the dataset has been rendered fit for ML training, we deploy the ML models and utilize hyperparameter tuning to find out the which algorithm has the best performance and the tuning to achieve it. For our experiments, we choose 3 distinct ML models from different categories:

- **Random Forest (RF):** an ensemble learning method primarily used for classification and regression tasks. It builds multiple decision trees during training and merges their results to improve accuracy and prevent overfitting. Each tree is constructed using a random subset of the data and features, a process known as bagging, which increases the model's robustness by reducing variance.

- **Factorization Machine (FM):** is designed to handle sparse and high-dimensional data, making them particularly useful for recommendation systems and predictive

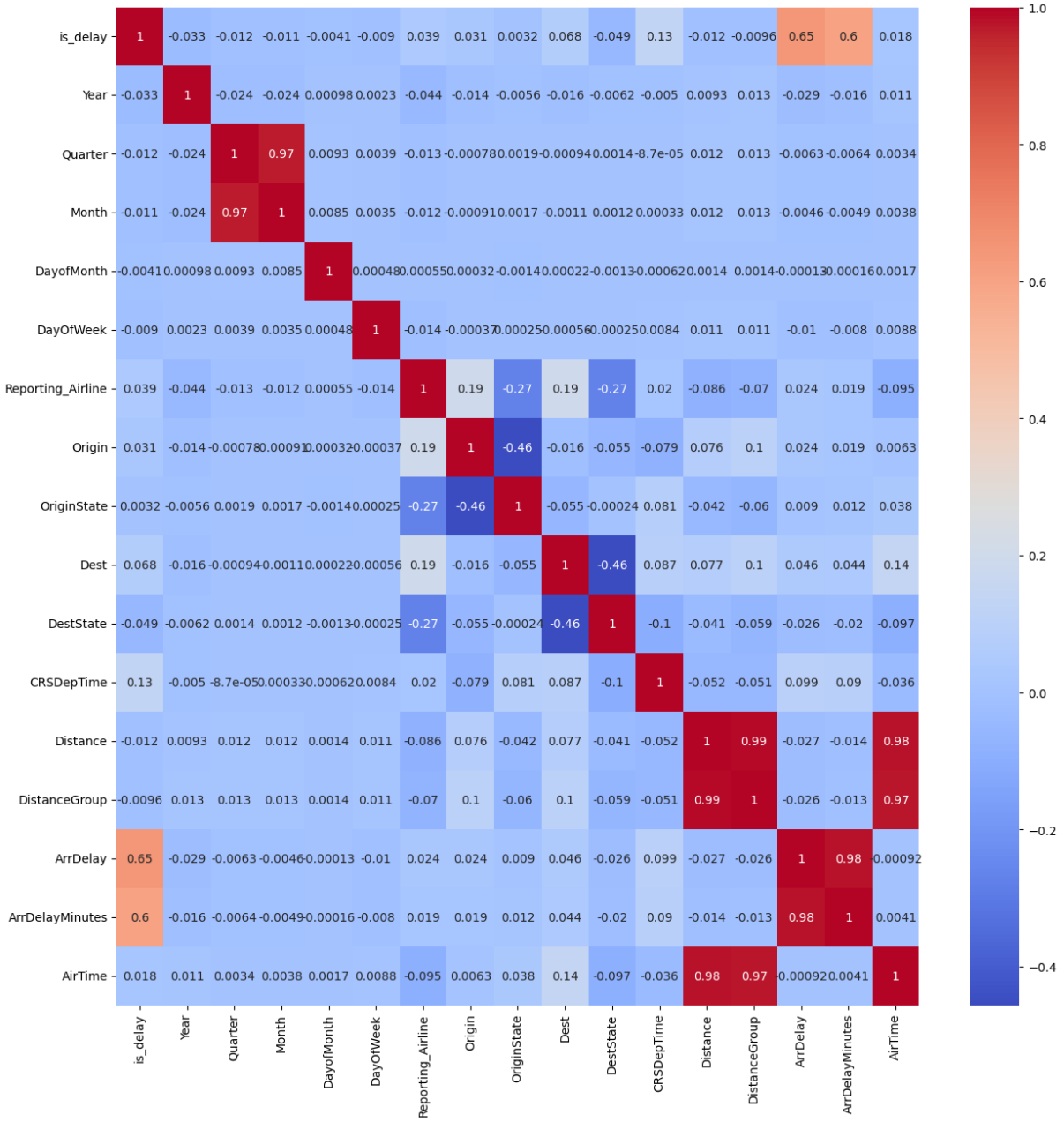


Figure 4.1: Correlation matrix

Table 4.1: Trial settings and measures of ML algorithms

Algorithms	Hyperparameter settings	Best setting	Best accuracy	F-beta score
Random Forest	`numTrees` $\in \{10, 20, 35\}$ `MaxDepth` $\in \{5, 10, 15\}$ `minInfoGain` $\in \{0.0, 0.1, 0.2\}$	`numTrees` $= 10$ `MaxDepth` $= 15$ `minInfoGain` $= 0.0$	0.7993	0.7662
Logistic Regression	`RegParam` $\in \{0.0, 0.1, 0.2\}$ `elasticNetParam` $\in \{0.0, 0.1, 0.2\}$ `fitIntercept` $\in \{True, False\}$	`RegParam` $= 0.0$ `elasticNetParam` $= 0.0$ `fitIntercept` $= True$	0.7930	0.7550
Factorization Machine	`RegParam` $\in \{0.1, 0.01, 0.001\}$ `stepSize` $\in \{0.1, 0.01, 0.001\}$ `maxIter` $\in \{10, 20, 30\}$	`RegParam` $= 0.1$ `stepSize` $= 0.1$ `maxIter` $= 10$	0.7903	0.7505

analytics. FMs model interactions between variables by factorizing the feature matrix into latent vectors, which capture the underlying patterns and interactions.

- **Logistic Regression (LR):** models the probability that a given input belongs to a particular class by applying the logistic function to a linear combination of the input features. The output is a probability score between 0 and 1, which can be thresholded to make a discrete class prediction.

We utilize K-fold (3 in our case) cross validation for the measurements.

Table 4.1 illustrates the rather similar performances of 3 models, with RF being the most notable in terms of F-beta score (0.7662 compared LR's 0.7550 and FM's 0.7505). However, 0.01% difference shows that "simple" ML algorithms, as a whole, do not converge particularly well on this dataset. For better forecast capability, more complex and advanced ML algorithms such as Neural Networks (NNs) are necessary.

5 Conclusion & Future works

In this project, we provide an insight into the flight delay prediction problem, discuss prominent solutions and develop an Apache Spark framework for forecasting. We perform an EDA on our chosen flight delay dataset and experiment with several ML algorithms and their respective hyperparameter settings to find out the best one for it.

Future works shall include developing an user interface to allow more interaction with the model, integrating higher level ML structure and finally, creating a near-realtime flight delay forecasting application.

Table 6.1: Workload assignment

Member	Assigned workload	Contribution
Nguyễn Đức Hải	- Finding the dataset, performing EDA and (partly) ML tuning - Writing the report	50%
Nguyễn Đức Duy Anh	Overseeing the project, prepare AWS Workspace, (partly) ML tuning and correcting codeworks - Writing the report	50%

6 Assignments & Plan

We divided the overall project workload as in Table 6.1.

References

[1] Inc. Amazon Web Services. Amazon web services, 2024.

[2] Microsoft Corporation. Visual studio code, 2024.

[3] Kaggle Datasets. Us domestic flights delay prediction (2013 - 2018), 2018.

[4] Apache Software Foundation. Apache spark, 2024.