Air Traffic scaling, delay- analysis and prediction using PySpark on Hadoop Cluster

Scaling, Simulation, Delay analysis and Delay prediction of 2019 US domestic air flights

Mohammed Zoher Guniem
University of Stavanger, Norway
m.guniem@stud.uis.no

Megh Raj Upreti
University of Stavanger, Norway
mr.upreti@stud.uis.no

ABSTRACT

As more commercial airplanes take off to the sky, data generated from air traffic is rapidly increasing in both size and complexity. According to the International Air Transport Association (IATA), Air travel industry is anticipated to annually grow by 3.5% in the next 2 decades (IATA, 2018). This growth raises big questions on how to gather, store, analyze and use the data generated from those flying cities in the sky. In this research, we start by using flight route information to produce a scalable and realistic simulation model for air traffic, and in the second phase we analyze the delay distribution and then evaluate some machine learning algorithms so we can have a better scientific foundation on which machine learning algorithm can produce better and more accurate prediction on flight delays.

KEYWORDS

Hadoop, Spark, PySpark, Power BI, Python, Aviation, Air-traffic, Scaling, Simulation, delay, Prediction, Machine learning, Random Forest, Logistic Regression, Decision Trees.

1 INTRODUCTION

Over the last few decades, air transport is increasing in popularity because of its speed and comfort which eventually increases the traffic in the airspace. With the great increase in air traffic comes a large increase in the demand for capacity in airports and airspace. However, the capacity in the aviation industry is often a tight constraint and it cannot keep increasing at the rate necessary to match the rising demand.

During peak hours, the demand for resources in both airports and airspace is at its highest. Some of the most important resources are:

- Good trained human resources.
- Take-off and landing slots.
- Spacious airspace.
- Available Gates at airports.
- Available taxiways and runways.
- Many other factors.

Airspace congestion and flight delays are two of the most important bottlenecks factors that limit the available resources and causes multiple unhealthy side effects on both the operation of air industry and thus the growth of the economy.

It is therefore crucial to have a good trained human resource to manage the airspace and ground operations to avoid any leak of capacity. In this paper we shall investigate the possibilities of using big data technologies to solve such challenges and limitations. Some of these cutting-edge technologies are:

- Apache Hadoop Clustering, which uses the MapReduce programming model and a network distribution to help solving problems related to big data and its demanding computation (Wikipedia, 2020).
- Apache Spark framework is a cluster-computing framework that offers an interface to program entire clusters with implicit data parallelism and fault-tolerance. In this project we use the powerful python language to work with

spark by its popular library PySpark (Wikipedia, 2020).

- We will also be using some other python libraries and software's like "basemap", "matplotlib" and Power BI to further help and assist our research (matplotlib.org, 2020).

2 BACKGROUND AND MOTIVATION

2.1 Air Traffic Scaling and Simulation

Today's modern aircrafts are equipped with numerous sensors which measures the performance and different states of each part and system of the aircraft. From the very basic of flying parameters like speed, altitude and location to more detailed data like temperature and pressure from the airplane's engines and cabin. Data from each airplane is often sent to datacenters on the ground for use in maintenance and troubleshooting. Most airlines also store this same data about each aircraft for later use in analysis and simulation.

Through the age of aviation, Simulation has had great benefits in training staff to handle the routines, procedures and challenges of the aviation industry. All trainee pilots and airspace controllers must spend a big amount of their training in simulation environments. Such simulations enable them to be prepared for daily handling of air traffic that never stops around the clock. Unfortunately, making a realistic simulation model is a challenge for all software engineers. Simulation software's can either generate air traffic randomly or by using data from realistic flights. And because the simulation must be as realistic as possible, the first option is hard to consider. This makes any simulation software very dependent on data from real life.

The simulation algorithms must have a normal uncomplicated flow and low running time, which means that data must be easily scalable with low overhead. In this research, we take advantage of the known capabilities of Apache Hadoop and Spark to design a scaling algorithm that uses data from earlier US domestic flights to provide realistic data for general use in air traffic simulation and analysis.

2.2 Delay analysis

Another problem the aviation industry faces is in the delays caused on arrival and departure, these delays cost both the airlines, and passengers billions of dollars every year. A report about the effects of flight delays on economy was given by the United States Joint Economic Committee (JEC) (Schumer & Maloney, May 2008, p. 3) and it has estimated that delayed domestic passenger flights costed the U.S. economy close to $41 billion in 2007 alone. The table in figure 1 below shows the distribution of delay costs between airlines, passengers and other related businesses. Understanding and predicting such delays should help all involved parts to be prepared and thus minimizing the effect of such delays on the economy.

Airline Operating Cost	Value of Passenger Time	Spillover Costs to the Economy	Total
$19.1 Billion	$12.0 Billion	$9.6 Billion	$40.7 Billion

Table 1. Flight delays cost passengers, airlines, and the US economy billions. (https://www.jec.senate.gov/public/_cache/files/47e8d8a7-661d-4e6b-ae72-0f1831dd1207/yourflighthasbeendelayed0.pdf). Page 3

Since Microsoft launched its first public version of Power BI in 2015, the interests have grown around it because of it is user-friendly and flexibility in performing data analysis. Although Power BI has a relatively big collections of different visualization tools, loading a big dataset into Power BI will produce a very heavy and slow visualization model with degraded user experience.

This is where we can take advantage of the computational power of a Hadoop Cluster and PySpark methods to easily extract target information about delays from big datasets and use it to build fast and flexible power BI models.

2.3 Delay Prediction

To take the research in flight delays to the next level, we shall later apply machine learning to predict and expect such delays before they occur based on what we already know. Hadoop PySpark has some great built-in libraries that helps runs the most popular and efficient machine learning algorithms on our datasets, and based on related work we should then test the accuracy of some popular machine learning algorithms on our dataset like Random Forest, Decision Trees and Logistic Regression.

An accurate machine learning algorithm can be helpful when working with big datasets and trying to understand the hidden relations between attributes. But as we already know a machine learning algorithm can be highly accurate with one dataset but very much less accurate with another. Therefore, we should run tests where we compare different machine learning algorithms based on earlier related work and calculate their accuracy before moving on and selecting one machine algorithm to work with in the future.

We should also keep in mind that even though a prediction algorithm works with high accuracy today, this does not mean the same algorithm will necessary give high accuracy prediction in the future, this is why it is important to keep a continuous evaluation of the performance of any available machine learning algorithm on our an updated dataset.

2.4 Dataset

For this research, we had the need of a dataset that is realistic, includes detailed information about flights and covers the geographical size of nearly a continent. Therefore, the choice has fallen on the United States of America. Since the airspace over the mainland USA is known to be heavily loaded with commercial airplanes around the clock, most of them are on domestic routes.

We are using an open source dataset which includes all the domestic flights on both mainland USA and overseas territories from the year of 2019. The dataset is downloaded from Bureau of Transportation Statistics. It is contained in 12 CSV files, one file per month. Its size can vary according to research need, because it is possible to only select the needed data parameters on each flight. In our case the initial size of the dataset is about 1.88 GB.

The most important advantages of this dataset are:

- Provides detailed information about the flight route from taxing into the runway until final arrival at the terminal, like actual and scheduled clock of taxi out, wheels off, wheels on, and taxi in.
- Reasons for flight delays are given in 5 clear and reasonable categories. Which gives us a clear picture on the actual causes of each delayed flight.

But the dataset is not perfect and has some issues that was discovered and dealt with under this research. Some of these issues are:

- Clock time like wheels off and wheels on is given in local time, which makes it essential to convert these into UTC times.
- Delay categories are numeric and needs to be converted into suitable interval categories for prediction.
- No airport coordinates included in the main dataset. Which makes it necessary to use some support dataset to provide this information.

The support dataset comes from the same bureau and enables us to gain more information about the geographic coordinates of the departure and arrival airports, along with UTC time variation. This support dataset is of size 1.17 MB for our needs, but it comes in a separate CSV file and includes data from almost all airports inside and outside the USA, which is very useful in case of a global extension of our research.

2.5 Tools & Setup

The following highlights some of the tools and technologies used in this research:

- Apache Hadoop, version 3.1.1

Using a distributed cluster consisting of one master node and 3 slave nodes.

- Apache Spark, version 2.4.5

Installed on top of the Hadoop cluster and utilizes its resources.

- PySpark

A powerful library that enables us to use its effective functions to write a consistent and god structured code using python. The PySpark library takes care of translating the python code into Scala which is the native programming language of Spark.

- Power BI

A popular visualization tool introduced by Microsoft. it has the main advantage in analyzing data in a user-friendly, and reactive way. It also has multiple visualization tools and models that makes the analysis model more flexible to changes in the future.

- mpl_toolkits.basemap

An open-source python library that enables us to retrieve the flight path using the method of great circle calculation. The library also helps with the actual simulation on the client side by plotting the results in a geographic map.

- matplotlib

A popular open source library that is helpful for making simulation and statistical plots.

- Another helpful open-source library

datetime, argparse, sys, imageio, json, time, os, shutil, and cv2.

2.6 Related Work

There are several works in the literature that focus on air-traffic and airport delays. Airline operations are highly complicated processes that are intended to regulate expenses, constraints, and interdependent resources, such as the crew, aircrafts, airports, and maintenance facilities. Myriads of research have been carried out on aviation planning problems such as delays, but only a few have been performed on the characteristics of flight scaling, airspace simulation, airline delays and the use of machine learning to predict flight delays. Following are some useful researches that we used as an inspiration and foundation for our research:

- Agent-based Modelling and Simulation of Air Transport Technology in 2013 by Grether and Nagel.

Researchers used scaling and simulation software for flight data. They normalized the data, performed pre-processing and simulated the results between inbound and outbound flights. They also converted local times to UTC (Coordinated Universal Time) for consistency.

- Predicting Flight Delays in 2012 by Dieterich Lawson and William Castillo.

This research used a dataset of flights including several years, with 135 million flights. However, the research was limited to weather data only and obtaining 40% recall. They used Machine learning algorithms like SVM, Random Forest and naive Bayes.

- Application of Machine Learning Algorithms to predict flight arrival delays in 2017 by Nathalie Kuhn and Navaneeth Jamadagni.

The contributors to this project used machine Learning algorithms like decision trees, Neural Network and Logistic regression algorithms and concluded that the departure delays are the main factor in flight delays. They also performed some statistical analysis like average delays of flights and basic summary statistics.

- Predicting flight delays and cancellations using weather as a feature in 2016 by Movva, N. and Menon, S.

The report of this research has compared algorithms like Random forest, XGboost (Extreme Gradient), Linear regression, and SVM and finally concluded with AUC (Area Under Curve) 0.81 highest among all algorithms.

3 AIR TRAFFIC SIMULATION

3.1 Design

The main goal of this part of the project is to provide simulation applications with easily scalable data that has the following properties:

- The geographic position of an aircraft at each minute in a route.
- The route information of each aircraft.

The data should also be easily scalable between a time interval in the day, and according to which geographic area the simulation view is based on.

Based on the requirements above we decided to split the solution into 3 main steps which are explained in figure 1 below.

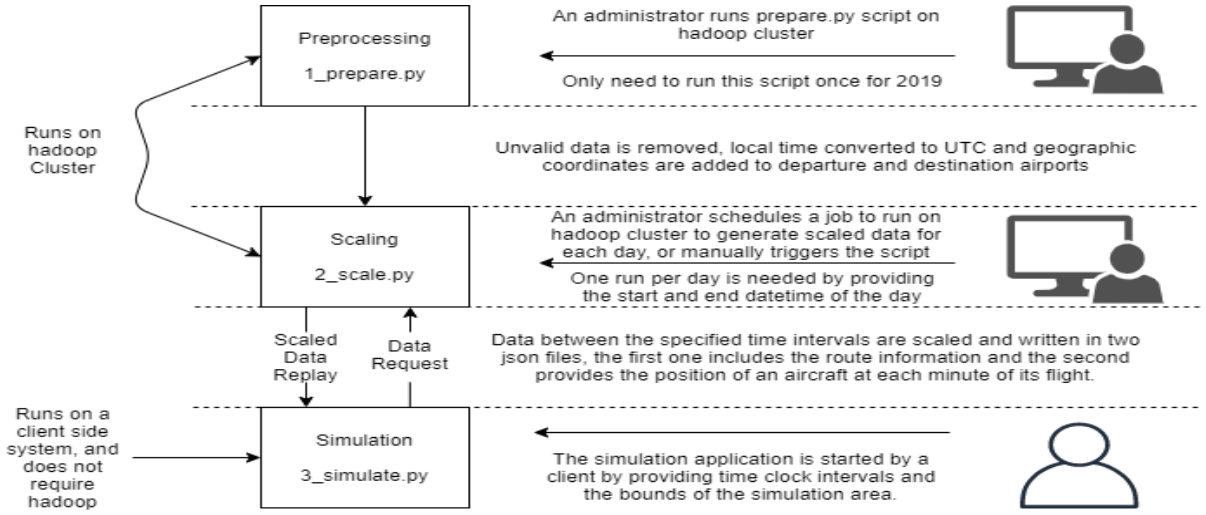

Figure 1. The overall design architecture of scaling and simulation for air-traffic.

3.2 Analysis

The 3-step divided design in figure 1 should improve the running time and flow of the simulation application, this is because much of the preparing and scaling steps are performed on the distributed cluster of Hadoop which is expected to be much powerful than the client computer.

For instance, the preprocessing step takes care of cleaning data, converting to UTC time and adding the needed coordinates for scaling. The scaling on its side filters out those rows that does not fit into the user specified time frame and generates the full path of a flight in geographic points. This leaves the client with the only need to check if a flight is inside or outside the bounds of the simulation area at a certain time (a minute in Unix timestamp).

On further examination, we found out that the preparing script takes about 13 minutes to finish for the entire dataset, and the scaling script takes only 5 minutes for a one-day time frame. These running times duration can off course change based on multiple factors like background processes in the operating system and other Hadoop jobs executed on the same cluster, but we could assume having a dedicated cluster to service the preparing and scaling scripts. However, the running time of the most critical script which is "2_scale.py" is not expected to exceed 30 minutes per day which is considered a good result when considering the massive size and processing need of our initial dataset.

3.3 Optimization

To optimize our solution and make it run faster, we decided to isolate the preprocessing in the

"1_prepare.py" script, this relaxes the scaling script from needing to perform a local time conversion to UTC, adding coordinates and other data validation steps each time the scaling "2_scale.py" script is running.

The scaling script "2_script.py" can then focus on finding out which flights belong to the given time frame and calculating the route path based on great circle calculations. In addition, this script is made more robust by enabling the administrator to choose any different time interval according to his need and can run as a scheduled job for each day or week, etc. in the future.

Replacing the scaled data into json formatted files, makes it easier for the client application to retrieve and process data which in turn improves the running time of the client application. The following figures shows the structure of the output data of the scaling step stored in 2 json formatted files.

Figure 2. The structure of the route_information.json file, for demonstration purposes the file is truncated to only show one example item.

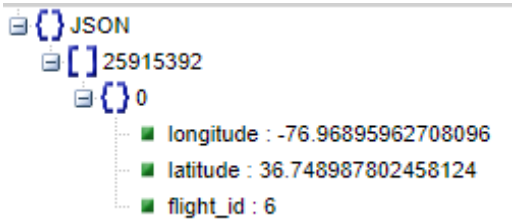

Figure 3. The structure of the position_information.json file which includes the position of an aircraft at each time, notice that the primary key in this file is the minute given in Unix minute timestamp. For demonstration purposes the file is truncated to only show one example item.

For further details on the final experimental phase on the application level see section 3.5 on Experimental evaluation of air traffic scaling and simulation, but first let's dive into the details of our implementation.

3.4 Implementation

To match the design that was previously mentioned in section 3.1, this implementation has been divided into 3 steps, these steps are explained below:

- Preparing the dataset

As mentioned earlier it is essential to deal with the weaknesses and deficiencies in the dataset before moving on with designing a scaling algorithm. To help us overcome this we design a preprocessing script that runs on the Apache Hadoop cluster using PySpark. The script does the following on the entire dataset:

- Some minority number of the flights in the dataset does not have any airtime, wheels off and/or wheels on time due to cancelation or some other reasons. It is then essential to remove these rows before proceeding on.

- Because flights can often connect two regions located in different time zones, It is difficult to use local time when considering the position of an aircraft in respect to time, it is then better to convert all times from local time to Coordinated Universal Time (UTC).

- Our original dataset does contain the sequential ID of the departure and arrival airports, but it does not contain their geographic coordinates that are needed to calculate the

flight path later in the scaling step. This is solved by joining our dataset with a support dataset that provides geographic coordinates of each airport.

The preparing step is meant to run only once on the entire dataset with no command line input arguments. The script is to be found under the following path and name in the project code folder
"/1_scaling_and_simulation/1_prepare.py".

➔ Scaling the dataset

The output of the preparing algorithm is written on the Hadoop file system in csv format and will be used by the next step which scales the dataset to produce a useful route information and location at each time in json format which can be used by a simulation application. The following steps explains the general structure of the scaling algorithm:

- A start and end datetime, along with a folder name where the output should be stored is provided by the user using command line arguments.

- After reading the prepared data, PySpark functions are used to filter the dataset in between the specified time frame, by keeping flight rows that have wheels off time before the end datetime and wheels on time after the start datetime. Figure 4 below explains this stage.

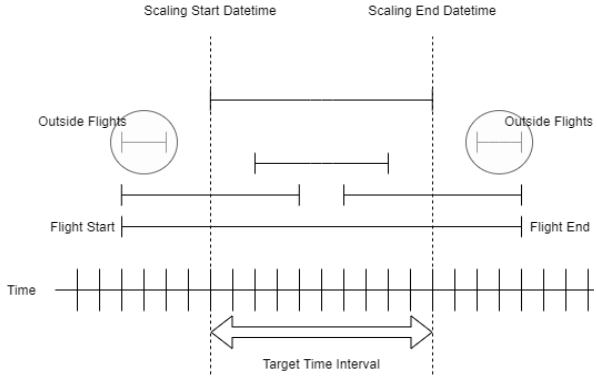

Figure 4. All possible flight durations relative to the specified time frame are shown in this figure, circled flights are filtered out under the scaling step as they were in the sky before or after the specidifed time frame in scaling.

- To be able to provide a location of the aircraft at each minute of the flight we rely on great circle calculations. The spherical shape of the earth makes it looks like pilots want to take a circle shape route from source to destination which seems longer on a traditional 2-dimensional map, but it is indeed the shortest path between source and destination. In addition to this pilots has to take into account the wind directions along the path and they do also fly according to established non-physical traffic checkpoints on the map, but often these checkpoints are placed in a path that is very much similar to a great circle.

To calculate the great circle of each flight, we take advantage of the python open-source library "mpl_toolkits.basemap" which takes in the coordinates of the source and destination as an argument along with the number of points to be generated along the path, by letting the airtime in minute represents the number of points to be generated by the method "gcpoints" we can make sure that the average speed is taking into consideration. Average speed lays very much near the cruising speed of a flight which is the speed that the flight is having most of the time during its duration.

All of the scaling steps are performed using PySpark function, except the great circle calculation which we experienced to be faster if done by iterating over the remaining scaled routes using the "rdd.collect()" PySpark function.

The script is designed to produce 2 json files, one that contains the information of each different flight and the other that provide the geographic position of each flight between the given time interval.

Although it is possible to change the time intervals, it is recommended to run the script for one day at a time, either in advance or a week before. The output can then be served to the client simulation application by using ftp or by giving access to a NoSQL database like mongoDB. The json structure makes it easy to establish a NoSQL database for this purpose, but for our project we feel it is enough to try to save some time and use the scp command to be able to use and test the scaled data on the client side.

The scaling script is to be found under the following path and name in the project code folder.
".../1_scaling_and_simulation/2_scale.py".

➔ Simulation

In this step, we decided to develop a small simulation application that uses the scaled data from the scaling step above, the results of this step should determine the reliability and efficiency of our implementation. Further details on the experimental setup and result is described in the section of experimental evaluation for air traffic simulation 3.5

The simulation application is to be found under the following path and name in the project code folder ".../1_scaling_and_simulation/3_simulate.py".

And following is an example of its output images

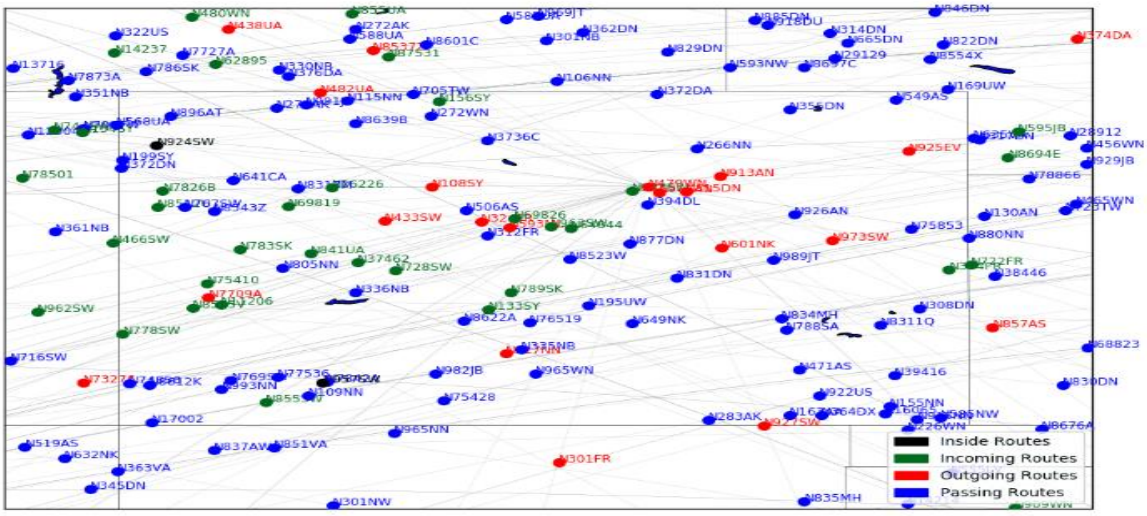

Figure 5. A screenshot taken from simulation over the state of Colorado at the 10th of April 2019. The aircrafts are labeled by their tail number, and given the color of red for outbound, green for inbound, blue for passing over and black for inside area flights

3.5 Experimental Evaluation of Air-traffic Scaling and Simulation

A. Experimental setup

We have developed a simulation application for general use purposes that uses the output of the Hadoop scaling step and takes in the following command line argument from a user:

- North-, south-, east- and west bounds.
- Tag of the aircraft whether it is the tail number of the aircraft or route between airports, cities or states.
- Start and end clock of the simulation.
- The name of the folder where the output should be stored.
- Whether to keep or delete snap images.
- The gif duration of each minute interval in the output.

We randomly select a day, run the scaling script on Hadoop for this day and then use the scaled results to test this simulation application. Then measure its running time by dividing the day into 8 intervals each interval containing 3 distinct hours of that day. Also, for this task we select the entire airspace of the state of Colorado as a simulation area because of its strategic location in the middle of the United State, and its busy airspace. We will also be running the

client simulation on a normal PC with RAM equals to 32 Gb which is much lower than must machines that are dedicated to simulation can offer in modern days. The simulations are also executed concurrently on the same machine and operating system.

B. Results

The running time results of our experimental setup is giving in the table below

Time interval in the 10th of April	***Simulation running time to produce output, hh:mm:ss***
00:00 – 03:00	0:06:55.823885
03:01 – 06:00	0:06:19.043982
06:01 – 09:00	0:04:18.139255
09:01 – 12:00	0:04:18.947238
12:01 – 15:00	0:05:17.992771
15:01 – 18:00	0:06:41.680611
18:01 – 21:00	0:06:21.062542
21:01 – 23:59	0:05:57.069641
Total Running time	0:46:6
Average per simulated minute in running seconds	

Table 2, The running times from the simulation scenario.

The sum of running times in the table above is around 46 minutes and 6 seconds for all 8 simulations. If we divide this sum on the number of seconds in a day, we get the amount of average running time spent on each minute of simulation.

(46*60)+6 / 24*60*60 ~ 0,08 mean running time per minute of simulation.

As we see from the calculations above, it takes a client machine around 0,08 seconds to retrieve simulation data from one minute of airspace traffic above the State of Colorado. Which is considered a promising result as 0,08 seconds is barely noticeable by humans under simulation.

4 FLIGHT DELAY ANALYSIS

4.1 Design

There exist 5 different types of delays in our dataset, they are categorized according to their main operation area where the delays were caused, it is also possible for a flight to have multiple delays at the same time. These delays are specified in the following table:

Category of delay	Explanation
Carrier delay	related to the maintenance and condition of an aircraft
Weather delay	caused by weather conditions
NAS delay	related to the national airspace system
Security delay	caused by any security breach
Late aircraft delay	arrival delay at an airport due to the late arrival of the same aircraft at a previous airport

Table 3. Types of Delay. (https://aspmhelp.faa.gov/index.php/Types_of_Delay).

The aim of this part of our research is to help improving the speed and efficiency of a power BI model that visualizes the distribution of different delays across multiple other features from the same dataset.

To help achieve a better power BI model, we have designed a PySpark script that aggregate and groups the rows in our datasets in terms of a given feature column like an airport, city or state. The script then calculates some basic statistics like the sum, mean, minimum and maximum of each delay grouped by a given feature category.

The script is designed in a way that a user can use a json config file which is found under the path "2_delay_statistics\analyze_config.json" to be able to enter instructions to what feature category(s) the script should use to produce the different delay statistics.

The output is then written to one or more csv files on both the local file system and the Hadoop file system for distributed use. This output will serve as a csv data feed by loading each generated csv to a power BI model, for

more information about the experiments and tests of the power bi model, please see section 4.5

4.2 Analysis

The main generator script is to be found under the following path and name "2_delay_statistics\analyze.py" and it loops through the array of "analyze_targets" in the previously mentioned json configuration file. During every loop similar operation are performed on different features using PySpark functions.

Thus, the running time of the script is

$F(n) = O(n * P)$,

where n is the size of "analyze_targets", and P represents the unknown cost of running built-in PySpark functions.

But in case the target feature is built from more than just one single feature as it is possible, the running time of the script will be

$F(n) = O(n * M * P)$,

where M is the size of the array of "target_components". But "M" can also change in length between different merged configuration according to the need of a user on how many features he/she wishes to merge. Therefore, we can write the final running time of our script as

$f(n) = O(N1 * M1 * P1) + O(N2 * M2 * P2) + \ldots + O(N1(n-1) * M(n-1) * P(n-1))$,

where

$n = N1 + N2 + N3 + \ldots$

$M = M1 + M2 + M3 + \ldots$

$P = P1 + P2 + P3 + \ldots$

As we see from the time analysis, the running time is highly dependent on the user input from the configuration file. Which makes it necessary to setup an experiment and take a close look at actual performance under production.

4.3 Optimization

To make the PySpark script more dynamic and easier to use, we have decided to use a configuration file that enables the user to specify different parameter settings. An example that shows the structure of the json configuration "analyze_config.json" file is given in figure 6.

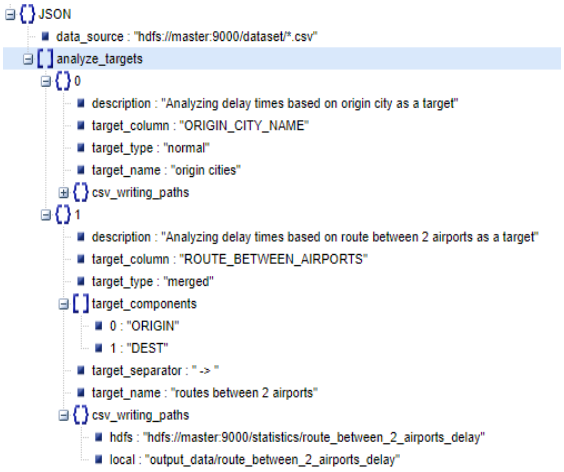

Figure 6. A configuration example of the analyze.py script.

A user can also use the opportunity of merging some group of feature data columns as one target feature, an example would be if a user wants to analyze based on the route from destination to arrival airport, this is shown in the seconds element of the configuration in figure 6 above. And makes the script even more helpful in analyzing different correlations between data features and their impact on delays.

4.4 Implementation

Based on the design constraints and possible optimization, which was discussed earlier, the pseudo-code for the analyze script is given below.

- Read "analyze_config.json"

- Read dataset

- For N in length(«analyze_targets»):

    - read configuration parameters

    - if target_type == "merged":

        - merged_features = target_components[0]

        - for M in "target_components":

            - merged features = merge(merged_features, M)

        - target_column = merged_features

    - else:

- target_column = “target_column”

- aggregate and group delays according to merged_features.

- compute sum, mean, minimum and maximum for each feature or feature merged group.

- Write to csv and Hadoop file system.

Under development we did notice that most of the flights has been without any delays and therefore, we know that calculating the minimum, mode and median of the delays will most probably equals 0, and results in slowing down the analysis script. Therefore, we decided to eliminate the calculations of the mode and median and just calculating the minimum of each delay. Later, and under analysis a minimum value that is higher than 0 can alarm us about a high delay as minimum value on any selected filtering effect of any category feature(s).

We also had to perform some data cleaning to replace all delays that were not registered with a value of 0, this is because the dataset provider wanted to save size and therefore had no value what so ever when there was no delays and Spark automatically assumes that the value there is null and not 0, calculating statistics with a value of null using PySpark functions gives us the wrong calculations, an example is given below:

Data = [3, 3, null]

Average = (3 + 3) / 2 = 3 // wrong average

Data = [3, 3, 0] // replacing null with 0

Average = (3 + 3 + 0) / 3 = 2 // correct average

4.5 Experimental Evaluation of Delay Analysis

A. Experimental Setup

The experimental setup consists of 2 stages, the first is regarding the PySpark script that runs on Hadoop cluster and produce the csv feed files to by used by the Power BI model, and the second stage is to use the csv file to produce a working Power BI model.

We start off by running the “analyze.py” on Hadoop cluster with 11 configured analyze targets. 8 of these targets are single features about origin, destination, month and aircraft and 3 others are merged to represent the route between state, city and airports.

B. Results

The table below shows the running time of each configured section in “analyze_targets”

Feature	Type	#of merged features	Running time in seconds
Tail_number	Single	1	43
Destination state	Single	1	30
Destination city	Single	1	29
Destination airport	Single	1	29
Origin state	Single	1	31
Origin city	Single	1	30
Origin airport	Single	1	28
State route	Merged	2	33
City route	Merged	2	34
Airport route	Merged	2	32
month	Single	1	28

Table 4. Experimental running times of different configurations in analyze_config.py.

We notice here that the highest running time was registered not in merging as we would expect but it is under categorizing by the tail number of different aircrafts. The reason for this is that the total sum of generated distinct categories would be highest by using the tail number, which proves that the expensive part is computing statistics for each category using PySpark functions and not by merging feature columns together.

Following in table 5 is some approximately measured basic statistics in seconds from the observed running times.

Statistic Variable	Estimated Value
Mean or average	32
Mode	28, 29, and 30
Median	30
Standard deviation	4
25% percentile	29
75% percentile	33
Minimum	28
Maximum	43

Table 5. Calculated basic statistics about the observed running times on the script analyze.py.

We notice that it is possible to locate 3 different mode values and they all together represent more than 50% of the measured running times. Which means that there exist a 50% probability that the running time will be one of these 3 mode values.

The standard deviation tells us that most of the running times are within the range of 4 seconds from the mean value of 32 seconds.

Further on, we locate the 25-percentile to be at 29 seconds and the 75-percentile to be at 33 seconds, the differentiation between these 2 percentiles is equal to the standard deviation of 4 seconds. Which give us a good confidence to conclude that statistically there exist a high probability that the running time of each target component will fall within 29 and 33 seconds, especially when observing that the 3 observed modes are all within or close to the range from 29 to 33 seconds.

We now have analyzed the running time of our script, but what about the main goal of this script, and it was to make power BI run faster and more user-friendly with minimal delay when processed by the visualization tools. There is no better way to test this than building a Power BI model that uses all the csv outputs from the "analyze.py" script that has been running on Hadoop cluster. After building a model using all the output data and many of the available visualization tools, we can conclude that the power BI model is working in much more effective way than just loading the whole dataset into the model, the model has also become light weight with a total size of 1,83 MB and include detailed information about different types of delay according to different feature categories. In addition to this the model is easily scalable giving the user the opportunity to focus on isolated incidents, features and extracting useful information about delays.

The model is available in the project files at the "2_delay_statistics\delay_analysis_power_bi_model.pbix"

After installing Power BI, feel free to open the model and test its capabilities, with no actual need to rerun the "analyze.py" script on Hadoop cluster.

5 FLIGHT DELAY PREDICTION

5.1 Design

The main aim of this part of the project is to predict whether a flight gets delayed or not. We have chosen three classification algorithms namely Decision Trees, Random Forest, and logistic regression. The general flow of all the algorithms is presented in figure 7. All the Machine Learning algorithms are run separately to make the flow of code smooth, efficient and clear.

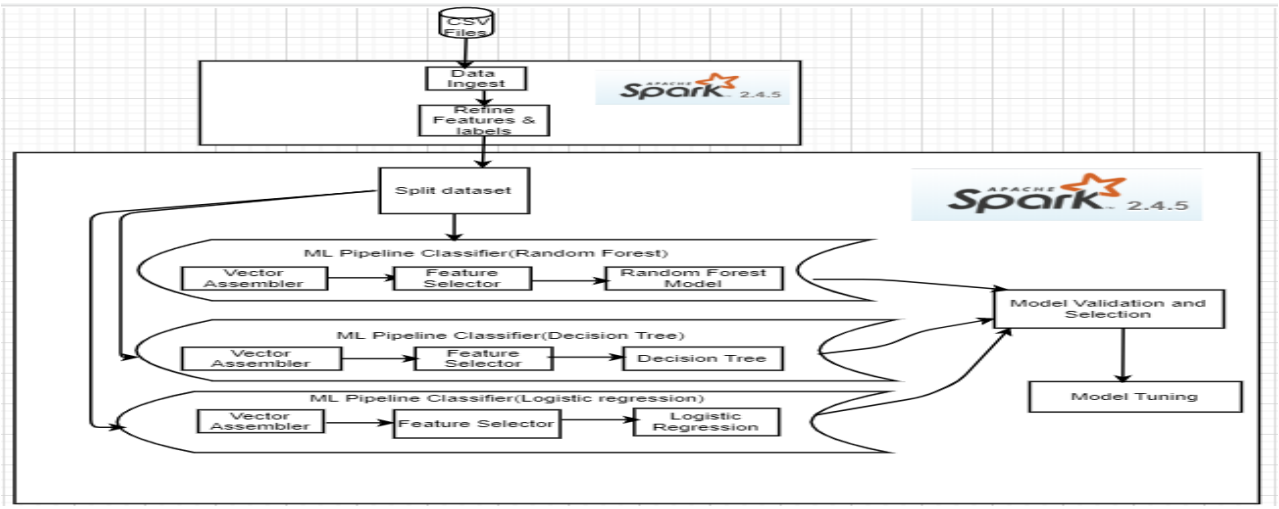

Figure 7. The general flow of machine learning implementations to predict flight delays

5.2 Analysis

The algorithms are run separated unlike many trends to run all at one time. This decreases the running time and prevent running out of memory, data from all months of 2019 are loaded into a data frame using PySpark.

All the algorithms take a long time to show the desired output because of the Hadoop cluster getting slow and code need to run multiple times through pipelines while training, prediction and evaluating the model. Usually It takes 25-30 minutes to run the single algorithm through the pipeline. Running the same dataset locally causes "out of memory" errors and if it works, it still very slow.

Pipeline has been introduced after splitting the dataset so that it combines all transformers and estimators in an orderly manner to specify ML workflow.

5.3 Optimization

To optimize our solution and make it run faster, we decided to run separate scripts for each of the machine learning algorithms.

The decision tree script "descision_tree.py" focuses on using the algorithm of its own and evaluate the model performance. In addition, this script is made more robust by enabling transformers and estimators in a pipeline for faster processing.

The predicted results of all three algorithms are saved in parquet files for using those predicted values and raw prediction values when needed in future work.

5.4 Implementation

To match the design that was mentioned in 5.1, this implementation has been divided into 3 parts such as data ingest, splitting the dataset and using pipelines for modeling and evaluating the data.

The data we have kept in Hadoop distributed file system is loaded into spark framework. We used Spark MLlib rich transformations like OneHotEncoderEstimator, VectorAssembler.

OneHotEncoderEstimator maps categorical features to a binary vector. It can transform multiple columns and for each input column output vector column is generated and used in VectorAssembler.

VectorAssembler combines all columns to a single vector to train ML algorithms. This single vector is put in pipeline for the training and testing data.[11]

Pipelines are sequence of stages such as transformers and estimators. Transform method is used for transformer stages and called on dataframe and fit method is used for estimator method which becomes Pipeline model. The advantage of using ML pipelines is hyperparameter optimization.[11]

- Supervised Learning Classifier:

Supervised Learning comes into effect when the model is getting trained in the labelled dataset. In our dataset, we made a label as 0 and 1, and we use several machine learning algorithms to predict the labels.

We have selected supervised learning models such as Random Forest, Decision Trees and Logistic regression. Since, the aim of the project is to predict whether the flight will be delayed or not, which is a classification problem.

- Random Forest (RF):

It combines many decision trees resulting in the risk of overfitting. Spark MLlib supports and handles categorical and numerical variables in a good way. The algorithm works randomly in training dataset, in such a way that each decision tree is different. Due to this randomness, there are chances of model bias slightly. From each decision tree, random forest get class vote and then it is converted into majority vote by taking average of all class voted obtained from each decision tree[12]. The code of this algorithm is "random_forest.py".

- Decision Tree (DT):

DT makes it easy to use, interpret, and handle categorical features of data. This algorithm does not have to scale and normalize the data. Therefore, decision tree requires less effort for data pre-processing during data preparation. The drawback of this model is that they are unstable, small change in data leads to change in structure of the optimal decision tree.

The code of this algorithm is "descision_tree.py".

- Logistic Regression (LR):

Logistic regression is a classification model suitable to predict categorical responses. It predicts the probability of outcomes. Using family parameter while selecting binomial or multinomial logistic regression otherwise the spark will automatically find correct variant and classify the parameter.

5.5 Experimental Evaluation of Delay Prediction models

A. Experimental setup

We have used Machine learning algorithms which will do the following tasks:

- Load the data in spark and make a label of 1s and 0s.
- Use of transformers and estimators to convert data into vectors.
- Randomly split the dataset and introduce a pipeline for ML flows.
- Store the predicted and raw prediction values in parquet files for all the algorithms.
- Calculate evaluation metrics of all three machine learning algorithms.

B. Results

Classifiers are typically evaluated by Confusion matrix, and AUC. All the results are tabulated below. Random Forest got higher accuracy and AUC than the other two models. Based on TP, TN, FP, FN, the models got the result for Recall, Precision, Accuracy, and F1-score.

Confusion Matrix Classification Report

Metrics	Logistic Regression	Decision Trees	Random Forest
True Positive	403542	403106	403546
True Negative	0	45	0
False Positive	13911	13866	13907
False Negative	0	436	0
Precision	0.967	0.967	0.967
Recall	1	0.999	1
Accuracy	0.967	0.966	0.968
F1-Score	0.983	0.983	0.983
AUC	0.727	0.735	0.80

Figure 8, The figure shows the confusion matrix of the 3 models based on the experimental setup specified in 5.5.A.

Precision: Precision helps us to understand how many correctly predicted cases turned to be positive.

Recall: It helps us to understand how many actual positive cases were correctly predicted by our model

Accuracy: It is the proportion of predictions and total number of cases examined.

F1-score: It is the harmonic mean of precision and recall.

AUC: AUC stands for Area Under Curve. An excellent model has AUC near to 1 which implies it has good measure of separability and poor model has AUC near to 0 which implies it has worst measure of separability. This metric demonstrates how much model is capable of distinguishing between classes.

6 CONSLUSION

We started our project by scaling the flight data in way that enables us to run fast and efficient simulations by providing simulation programs with json data, which speed up the process of simulation due to its useful NoSQL structure.

Further on, we established a Hadoop PySpark script that has a mechanism to distribute different flight delay accordioning to some other user chosen feature(s) in the same dataset, This produced a light weight csv file that

contains important information about the distribution of delays, so Power BI models can easily visualize the results of "analyze.py" with minimal processing overhead and high user experience.

At the end, we decided to select 3 machine learning algorithms based on previous results from related work, and then test their performance so we can identify the most efficient algorithm to use in flight delay prediction. And based on the confusion matrix shown in figure 8, we concluded that Random Forest with an AUC measure of 0.80 is the most efficient in this case for the purpose of predicting flight delays in US domestic routes.

The 3 parts of this project contribute equally in solving major problems that the aviation industry experience at each day. From training pilots and air-traffic controllers with the most realistic picture of air traffic, to the power to be able to analyze and predict delays so they can be avoided in the future.

8 FURTHER WORK

There still a big potential for improving and further building of the work in this project, some suggestions are:

- Scaling and Simulation

Integrating a NoSQL database to be storing the scaled data from the "scale.py" algorithm. In addition, it is also possible to investigate building a simulation software directly on a Hadoop cluster and not by using scaled data from this Hadoop cluster.

- Delay analysis

Although our Power BI model is working in an efficient way, it still having minor issues with unclean data, like delays in over sea us territories which are not supported by the power BI map system. Other improvements can also be made on the method the user can use to program "analyze.py" script in case of using "analyze_config.py" to make it even more fault tolerant and user-friendly.

- Flight Delay Prediction

When it comes to flight delay predication, it is fully possible to start predicting flight delays on the fly based on flight data that is known before the flight takes place. We then recommend using the Random Forest library of PySpark to enable user to predict delays. In addition, we would like to explore if we can include external variables such as weather data to improve data analysis and prediction accuracy of the models.

In future work, we would like to compare the ML results under this project with unsupervised ML models like clustering, and dimensionality reduction.

REFERENCES

[1] IATA. (2018, 24 October). IATA Forecast Predicts 8.2 billion Air Travelers in 2037. Obtained from https://www.iata.org/en/pressroom/pr/2018-10-24-02/

[2] Wikipedia, (2020, 26 March). Apache Hadoop. Obtained fromhttps://en.wikipedia.org/wiki/Apache_Hadoop

[3] Wikipedia. (2020, 20 April). Apache Spark. Obtained fromhttps://en.wikipedia.org/wiki/Apache_Spark

[4] Matplotlib Basemap Toolkit documentation. (2020). Matplot and basemap documentation. Obtained from https://matplotlib.org/basemap/

[5] Schumer C. E. & Maloney C. B., (May 2008). Flight delays cost passengers, airlines, and the US economy billions. United States Joint Economic Committee (JEC). Obtained from https://www.jec.senate.gov/public/_cache/files/47e8d8a7-661d-4e6b-ae72-0f1831dd1207/yourflighthasbeendelayed0.pdf

[6] Grether D., Furbas S. & Nagel K., (2013). Agent-based Modelling and Simulation of Air Transport Technology. Procedia Computer Science.Available athttps://www.sciencedirect.com/science/article/pii/S1877050913007175

[7] Lawson D. & Castillo W., (2012). Predicting Flight Delays. Stanford Education. Obtained fromhttp://cs229.stanford.edu/proj2012/CastilloLawson-PredictingFlightDelays.pdf

[8] Kuhn N. & Jamadagni N., (2017). Application of Machine Learning Algorithms to predict flight arrival delays. Stanford Education.http://cs229.stanford.edu/proj2017/final-reports/5243248.pdf

[9] Movva N. & Menon, S., (2016). Predicting flight delays and cancellations using weather as a feature. Stanford Education. Obtained from http://cs229.stanford.edu/proj2016/report/MenonMovva-PredictingFlightDelays-report.pdf

[10] Federal Aviation Administration, (7 March 2019). Types of Delay. Obtained from https://aspmhelp.faa.gov/index.php/Types_of_Delay

[11] MLlib: Main Guide - Spark 2.4.5 Documentation. (2020).Retrieved April 22 from https://spark.apache.org/docs/latest/ml-guide.html

[12] Koehrsen, W. (2020). An Implementation and Explanation of the Random Forest in Python. Retrieved 23 April 2020, from https://towardsdatascience.com/an-implementation-and-explanation-of-the-random-forest-in-python-77bf308a9b76