Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Master in Data Science

Big Data: Spark Practical Work

Madrid, January 2025

Contents

1. Introduction

The objective of this project is to implement and apply Big Data concepts using Spark. The focus is on developing practical skills in distributed data processing, analysis, and machine learning while tackling a real-world problem. The task is centered on creating a predictive model to estimate the arrival delays of commercial flights based on features known at the time of takeoff. This project combines theoretical knowledge with hands-on implementation to demonstrate proficiency in Spark's capabilities for large-scale data processing and machine learning.

The dataset utilized for this task is derived from publicly available flight data, specifically domestic flights in the United States. This project emphasizes efficient data handling, feature engineering, model training, and validation.

1.1 Project proposal

The goal of this project is to predict the arrival delay of commercial flights using a machine learning model trained on historical flight data over different years. The project will be carried out in the following stages:

1. **Data Exploration and Preparation**:
   - Select relevant variables from the dataset and exclude forbidden ones.
   - Analyze and preprocess the data, addressing missing values and transforming raw features into usable formats.
   - Use feature engineering where appropriate to enhance model performance.
2. **Model Training and Validation**:
   - Utilize Spark MLlib to train the model, selecting and justifying the most suitable machine learning algorithm.
   - Split the data into training and validation sets to evaluate model performance.
   - Optimize model hyperparameters to achieve the best results.
3. **Model Deployment**:
   - Save the trained model in a standardized format for future use.
   - Develop a Spark application capable of loading the model and applying it to unseen data for prediction and evaluation.
4. **Performance Analysis**:
   - Measure and report the performance of the model on unseen data, including detailed accuracy metrics and insights derived from the results.

1.2 Tools

The tools and technologies used in this project include Apache Spark, leveraging Spark Core for distributed data processing, Spark SQL for structured data querying, and MLlib for machine learning and evaluation tasks. Jupyter Notebook will facilitate exploratory data analysis (EDA), feature engineering, and model training, while Python serves as the primary programming language, supported by libraries like Pandas, NumPy, and Matplotlib for data manipulation and visualization. The dataset comprises public flight data from the US Department of Transportation, available via Dataverse.

2. Exploration and Feature Selection

2.1 Data description

In this section, we present an overview of the dataset used in this project, highlighting its structure, variables, and format. The data consists of flight records that provide detailed information about flight schedules, delays, and other operational metrics. The list of all the variables is presented in the following table.

Table 2.1: Structure of the features

Variable	**Description**	**Type**	**Format**
Year	Year of the flight departure	Numerical	YYYY
Month	Month of the flight departure	Categorical	MM
DayofMonth	Day of Month of the flight departure	Categorical	DD
DayOfWeek	Day of the week of the flight departure	Categorical	1 - 7
DepTime	Actual departure time	Numerical	hhmm
CRSDepTime	Scheduled departure time	Numerical	hhmm
ArrTime	Actual arrival time	Numerical	hhmm
CRSArrTime	Scheduled arrival time	Numerical	hhmm
UniqueCarrier	Unique carrier code	Categorical	Text
FlightNum	Flight number	Numerical	Numeric ID
TailNum	Plane tail number	Categorical	Text
ActualElapsedTime	Actual elapsed time	Numerical	Minutes
CRSElapsedTime	Scheduled flight duration	Numerical	Minutes
AirTime	Time in the air during the flight	Numerical	Minutes
ArrDelay	Arrival delay	Numerical	Minutes
DepDelay	Departure delay	Numerical	Minutes
Origin	Origin IATA airport code	Categorical	IATA Code
Dest	Destination IATA airport code	Categorical	IATA Code
Distance	Distance between origin and destination	Numerical	Miles
TaxiIn	Taxi in time, in minutes	Numerical	Minutes
TaxiOut	Taxi out time, in minutes	Numerical	Minutes
Cancelled	Was the flight cancelled?	Numerical	Binary (0/1)
CancellationCode	(A = carrier, B = weather, C = NAS, D = security)	Categorical	Text (A, B, C, D)
Diverted	1 = yes, 0 = no	Numerical	Binary (0/1)
CarrierDelay	Minutes of sar carrier delay	Numerical	Minutes

WeatherDelay	Minutes of delay due to weather	Numerical	Minutes
NASDelay	Minutes of delay due to the National Aviation System	Numerical	Minutes
SecurityDelay	Minutes of delay due to security reasons	Numerical	Minutes
LateAircraftDelay	Minutes of delay due to the late arrival of flight aircraft	Numerical	Minutes

In addition, the features available for use by plane-data are listed. Several columns have been renamed to avoid any problems with joins to the main dataset.

Table 2.2: Structure of plane-data

Variable	**Description**	**Type**	**Format**
Tailnum	Tail number of aircraft	Categorical	Numeric ID
Type	Type of aircraft (either commercial or private)	Categorical	Text
Manufacturer	Identifies the maker of the plane (Boeing, Airbus)	Categorical	Text
IssueDate	Date aircraft first issued into service	Date	YYYY-MM-DD
PlaneModel	Specific model type of plane (ex: 747-4)	Categorical	Text
Status	Status if aircraft is active or not	Categorical	Text
AircraftType	More specific identification of aircraft (ex: fixed wing)	Categorical	Text
PlaneEngineType	Type of engine used by aircraft (turbo-prop, turbo-fan)	Categorical	Text
PlaneYear	Year of plane’s introduction to service	Numerical	YYYY

2.2 Feature Engineering

Several variables have been considered that can be useful when predicting the Delay of a flight. Several variables have been considered as potentially useful for predicting the Delay of a flight. This is only an exploratory analysis of variables that could be used. However, since one of the models will be a linear regression, these variables will not be used as-is, because multicollinearity issues may arise when new variables are created that are perfectly correlated with the original ones. From the predictive variables, we can compute others that might be beneficial for the analysis. This will create more variables, some of which may not be useful but will be studied later. The first variables that will be extracted are related to temporal information. The new variables considered are:

- **IsWeekend**: A binary variable indicating whether the flight is on a weekend (Saturday or Sunday). This helps to capture patterns related to weekend travel behavior.

- **Season**: A categorical variable representing the season of the year in which the flight takes place, classified as Winter, Spring, Summer, or Fall, based on the month of the flight. This could be useful to account for seasonal travel trends.
- **DayPart**: A categorical variable representing different parts of the day (Morning, Afternoon, Night) based on the departure time. This can help capture travel patterns that vary depending on the time of day.
- **IsHoliday**: A binary variable indicating whether the flight occurs on a holiday. This could be a useful feature when considering higher travel demand during holiday periods.
- **FlightDuration**: The duration of the flight in minutes, calculated as the difference between the departure and arrival times. This can help model flight times and delays more accurately.
- **Delay**: The delay in minutes between the actual departure time and the scheduled departure time. This is an important feature to model the punctuality of flights.

Some of the other variables than could have been created are:

- **FlightSpeed**: This variable represents the speed of the flight, calculated as the distance divided by the flight duration. It is cast to a float16 type to save memory.

Binary Variables for Different Parts of the Day:

- **IsMorningFlight**: Indicates if the flight departs in the morning (before 12:00 PM).
- **IsAfternoonFlight**: Indicates if the flight departs in the afternoon (between 12:00 PM and 6:00 PM).
- **IsEveningFlight**: Indicates if the flight departs in the evening (between 6:00 PM and 9:00 PM).
- **IsNightFlight**: Indicates if the flight departs at night (after 9:00 PM).

These variables help capture patterns related to the time of day when the flight departs.

Binary Variables for Flight Duration:

- **IsShortHaul**: Indicates if the flight duration is less than 180 minutes.
- **IsMediumHaul**: Indicates if the flight duration is between 180 and 360 minutes.
- **IsLongHaul**: Indicates if the flight duration is more than 360 minutes.

None of these variables have been used in the models because, as we are using simple models such as linear regression, there is a problem with multicollinearity. Creating a variable from another variable results in a correlation of 1, and using them together in linear regression introduces severe multicollinearity. This increases the variance and makes the model highly unstable.

2.3 Features Selected

Below is a table of the features selected for predicting **ArrDelay (bolded given it is the target variable)**.

Table 2.3: Structure of the features selected

Variable	**Description**	**Type**	**Format**
Month	Month of the flight departure	Categorical	MM
DayofMonth	Day of Month of the flight departure	Categorical	DD
DayOfWeek	Day of the week of the flight departure	Categorical	1 - 7
DepTime	Actual departure time	Numerical	hhmm
CRSDepTime	Scheduled departure time	Numerical	hhmm
UniqueCarrier	Unique carrier code	Categorical	Text
CRSElapsedTime	Scheduled flight duration	Numerical	Minutes
DepDelay	Departure delay	Numerical	Minutes
Origin	Origin IATA airport code	Categorical	IATA Code
Distance	Distance between origin and destination	Numerical	Miles
TaxiOut	Taxi out time, in minutes	Numerical	Minutes
ArrDelay	**Arrival delay**	**Numerical**	**Minutes**

Table 2: Structure of the features selected

In addition, using the plane-data.csv file, which was provided in the repository several features were also utilized to get specific information on the plane by conducting an inner join with the tail number.

Table 2.4: Structure of the features selected from plane-data

Variable	**Description**	**Type**	**Format**
PlaneModel	Specific model type of plane (ex: 747-4)	Categorical	Text
PlaneYear	Year of plane's introduction to service	Numerical	YYYY

2.4 Features excluded

Below the following features were not utilized

Table 2.5: Structure of the features excluded

Variable	**Description**	**Type**	**Format**
Year	Year of the flight departure	Numerical	YYYY
ArrTime	Actual arrival time	Numerical	hhmm
CRSArrTime	Scheduled arrival time	Numerical	hhmm
FlightNum	Flight number	Numerical	Numeric ID
TailNum	Plane tail number	Categorical	Text
ActualElapsedTime	Actual elapsed time	Numerical	Minutes

CRSElapsedTime	Scheduled flight duration	Numerical	Minutes
AirTime	Time in the air during the flight	Numerical	Minutes
ArrDelay	Arrival delay	Numerical	Minutes
Distance	Distance between origin and destination	Numerical	Miles
ActualElapsedTime	Actual elapsed time	Numerical	Minutes
TaxiIn	Taxi in time, in minutes	Numerical	Minutes
Cancelled	Was the flight cancelled?	Numerical	Binary (0/1)
CancellationCode	(A = carrier, B = weather, C = NAS, D = security)	Categorical	Text (A, B, C, D)
Diverted	1 = yes, 0 = no	Numerical	Binary (0/1)
CarrierDelay	Minutes of sar carrier delay	Numerical	Minutes
WeatherDelay	Minutes of delay due to weather	Numerical	Minutes
NASDelay	Minutes of delay due to the National Aviation System	Numerical	Minutes
SecurityDelay	Minutes of delay due to security reasons	Numerical	Minutes
LateAircraftDelay	Minutes of delay due to the late arrival of flight aircraft	Numerical	Minutes

As specified by the project requirements, the variables *ArrTime*, *ActualElapsedTime*, *AirTime, TaxiIn*, *Diverted*, *CarrierDelay*, *WeatherDelay*, *NASDelay, SecurityDelay* and *LateAircraftDelay* have been excluded due to this information not being available at the time of departure. *CRSArrTime* has also been excluded as its value isn't very clear. Additional variables have been removed such as year, which is a redundant feature given that only the 2007.csv file is being considered for model development. Variables such as *FlightNum* and *TailNum* are more useful as a key/identifier in the dataset and have no real prediction value for *ArrDelay*. Similarly, for exceptional flights, Cancelled and CancellationCode are excluded, because the data preprocessing will not consider these flights (the flights never occurred), and so their prediction value is null. The feature *Dest*, though potentially useful, is partially encoded by the variable distance (showing how far away the destination is). Furthermore, when considering the specific domain of this problem, the departure airport is far more likely to be responsible for a delay due to various factors than the destination airport. For these reasons, destination was excluded.

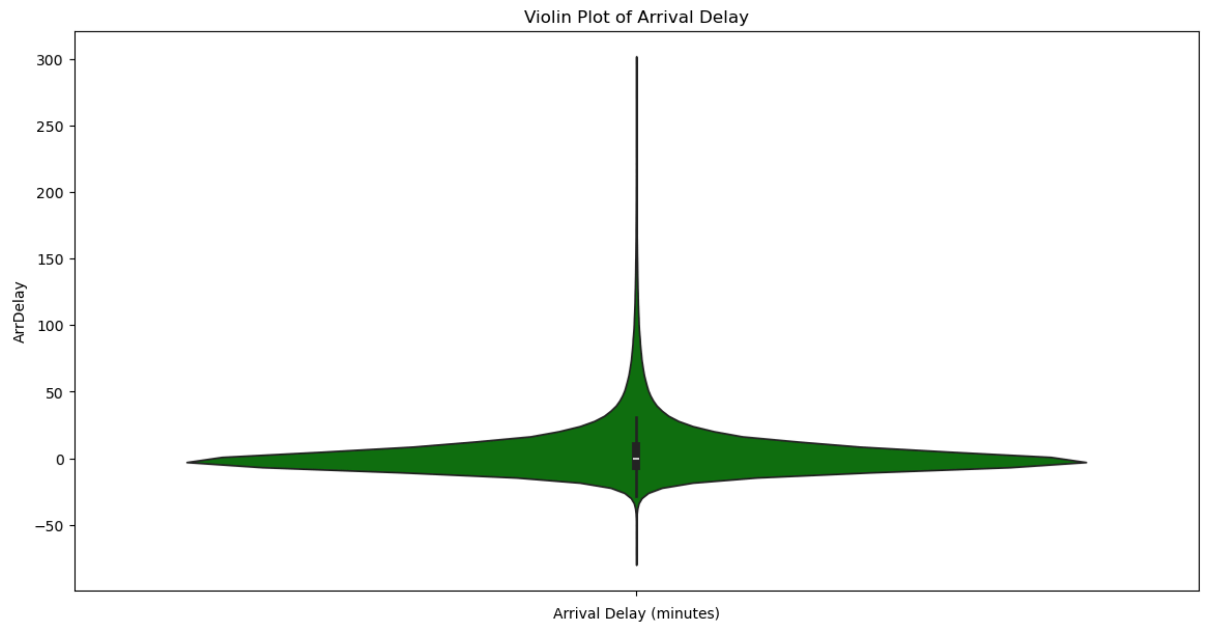
2.5 Preprocessing Steps

Various preprocessing steps were taken to ensure that predictions were made correctly. Firstly, it was useful to transform *DepTime* and *CRSDepTime,* encoded in an integer time format (ex: 800), into standard intervals. This was accomplished by first identifying the hour and minutes of the time, and then converting the time fields into "minutes since midnight". The results are the columns *DepTimeMinutesSinceMidnight* and *CRSDepTimeMinutesSinceMidnight* (ranging from 0-1440). Simple transformations were achieved using the modulo operator "%" with base 100 and dividing the time by 100 and casting it to an integer, and then multiplying it by 60 to get the actual amount of minutes. The two columns then are both added as minutes to get a true value of time. However, given the range of these new columns, it was necessary to also scale these variables so the models didn't place higher importance to these features unnecessarily. For this, the VectorAssembler and StandardScaler methods available in spark were utilized. The result is a scaled set of features that will be used in the model. Furthermore, the variable *Distance* also needed to be scaled, as there were values ranging from several hundred to over several thousand miles. The same steps as outlined above using StandardScaler were utilized. In addition to this, the variables *Origin, UniqueCarrier* and *PlaneModel* are defined as varchar columns. Though one hot encoding is possible, it is more efficient in Spark to utilize the StringIndexer functionality in order to convert each distinct value to an integer. This was taken care of and the columns were transformed into discrete integer variables. These mapping files were also saved for future use by the spark application as well, built off of the entire 2007 dataset. When new rows are analyzed by the spark application, an inner join is conducted on the mapping files with the primary dataset, so by default, any airports, plane models, or carriers which have not been defined will not be considered. The 2007 dataset however provides a robust and exhaustive list of such variables.

The column *PlaneYear* (related to planes) was transformed by subtracting it from the given *Year* of the flight to derive *PlaneAgeYears*. All *Cancelled* and *Diverted* flights were removed from the dataframe, as these are the exception and not the norm for a given flight, and don't serve any value for the problem at hand. Lastly, though null values were rarely identified if at all, any remaining null values were replaced with zero. Finally, it is worth mentioning that any plane which doesn't have a match in the plane-data dataset will not be considered, though the overwhelming majority of rows in the dataset have a match.

2.6 Exploratory Data Analysis

The objective variable ArrDelay has been studied. The distribution of the variable is not normal and has a lot of extreme values. Data ranges from -800 minutes to 1200 minutes. As a Delay of -800 minutes means a flight has arrived 12 hours in advance, this doesn't make any sense. A filter of minimum -120 minutes has been set, and observations with a lower delay are considered to be measure errors. In the other extreme, a flight in theory might be delayed for a long period of time, but these points could interfere a lot with the future models so extreme observations have also been deleted, as these extreme values account for less than 0.5% of observations. The violin plot is used to study the distribution of the remaining data. Although there is still a lot of values that are not inside the box plot limits, it has a much more normal distribution than before, with high variance and centred in 0.



We can also analyze the relationship between the potential regressors and the dependent variable for the regression. First, we used box plots for each variable to understand their distributions. Then, we performed a Kruskal-Wallis test to evaluate the significance of the relationships between the variables and the Delay. We obtain that all variables considered are useful for the models, so we have not eliminated any of them.

3. Models training

This section outlines the process of training various machine learning models to predict flight arrival delays. Key steps such as selected models, train/test data split and hyperparameter tuning, and model evaluation were performed to identify the most accurate and robust model for deployment. Just as a warning the model training and selection process was conducted using the dataset 2007.csv. The decision to use a single dataset was made to manage computational efficiency, as incorporating multiple datasets significantly increased execution time, extending it to several days.

3.1 Models selected

After reviewing the documentation of Pyspark regarding regression methods, it was determined that linear regression, decision tree regression, random forest regression and gradient boosting tree regression would be appropriate to consider as potential candidates.

3.2 Train/Test data split

For each model evaluated, a standard 80/20 train test split was done using RandomSplit along with a random seed.

3.3 Hyperparameter tuning

A modest hyperparameter grid was constructed for each model independently and is broken out below. The ParamGrid method was utilized.

Table 2.6: Hyperparameter Grid by Model Class

Linear Regression	**Decision Tree**	**Random Forest**	**Gradient Boosting Tree Regression**
RegParam: [.01,.1 1]	maxDepth: [3,5,10]	maxDepth: [3,5,10]	maxDepth: [3,5,7]
Elastic Param: [0, .5, 1]	maxBins: [300,400]	maxBins: [300,400]	maxBins: [300,400]

For each model, each grid of parameters was evaluated using k-fold cross validation with 5 folds. The best performing model was then selected for the four different model types. The CrossValidator method along with ParamGrid was utilized.

4. Model validation

Model validation ensures the reliability and generalization of the trained models. This section outlines the techniques, metrics, and results used to evaluate and select the best-performing model.

4.1 Validation techniques

Model validation was performed using 5-fold cross-validation, ensuring that the dataset was split into multiple partitions so that all data points were used for both training and testing at different stages. This approach minimizes the risk of overfitting and provides more robust and reliable evaluation metrics for each model.

In the models.py code, PySpark's CrossValidator object was used to perform cross-validation. A hyperparameter grid was defined for each model, optimizing key parameters such as *maxDepth*, *numTrees*, and *regParam*, depending on the type of model being evaluated.

4.2 Metrics used

The metrics used to evaluate model performance include:

1. **Root Mean Squared Error (RMSE)**:
    - ○ A standard measure of the square root of the mean squared differences between model predictions and actual arrival delay values.
2. **R-Squared (R^2)**:
    - ○ Indicates how well the model's predictions fit the actual data.
    - ○ Range: [0, 1], where values closer to 1 indicate better fit.
3. **Mean Absolute Error (MAE)**:
    - ○ The average of the absolute differences between predicted and actual values, useful for measuring overall accuracy.
4. **Mean Absolute Percentage Error (MAPE)**:
    - ○ Represents the error as a percentage relative to the actual value.

In the *app.py* code, these metrics were computed using PySpark's *RegressionEvaluator* class on the test dataset.

4.3 Validation results

The best-trained models for each model type were evaluated against the test dataset. Results were compared using the standard root mean squared error method (RMSE) and the results for RMSE are displayed below.

Table 2.7: Root Mean Squared Error by Model Class

Metric	**Linear Regression**	**Decision Tree**	**Random Forest**	**Gradient Boosting Tree Regression**
RMSE	10.42	10.3	10.45	**9.70**

- **Best Model**: The **Gradient Boosting Tree (GBT) Regression** model achieved the lowest Root Mean Squared Error (RMSE), indicating it is the most suitable model for predicting arrival delays.
- However, more complex models like **Random Forest** and **Gradient Boosting** also delivered competitive results and could be more effective for larger datasets or scenarios with non-linear relationships.

Regarding R^2 and MAE:

- The **GBT Regression** model demonstrated higher R^2 and lower MAE compared to other models.
- The MAPE values were relatively low across all models, indicating that predictions were within a reasonable range relative to the actual values.

5. Final evaluation

The final evaluation of the project focuses on assessing the performance of the best-selected model, Linear Regression, on unseen test data. This evaluation provides insights into how well the model generalizes and predicts arrival delays under realistic conditions. The evaluation metrics include:

- **Root Mean Squared Error (RMSE)**: The model achieved an RMSE of 10.459, indicating a reasonably accurate prediction of arrival delays on average. This is the lowest among the models tested.
- **R-Squared (R^2)**: The R^2 value reflects that the Linear Regression model captures a significant proportion of the variance in the data.
- **Mean Absolute Error (MAE)**: The MAE metric further confirms that the model predictions deviate minimally from the actual values on average.
- **Mean Absolute Percentage Error (MAPE)**: The MAPE value suggests that the model's predictions are within an acceptable error range relative to the actual arrival delay values.

The evaluation confirmed that while more complex models like Random Forest and Gradient Boosting Trees delivered competitive results, Linear Regression's simplicity and interpretability, coupled with its low computational overhead, make it the best choice for this dataset and problem.

These results demonstrate the model's ability to support airline operations by providing reliable delay predictions, which could help improve decision-making and enhance customer satisfaction.

6. Spark Application

This section covers the explanation of the Spark application created to load some test data and evaluate the performance of the predictions of some models.

6.1 Application design

The Spark application is divided into different blocks following a modular and scalable design. It performs the following key steps:

- **Data Loading**:
  - Accepts the path to the test dataset as an argument and reads the test data, handling any inconsistencies or errors (e.g., missing values).
- **Model Loading**:
  - Loads the pre-trained machine learning model best_model stored during the notebook phase.
- **Data Preparation**:
  - Transforms the test dataset to match the format used during model training, ensuring consistency in feature engineering and preprocessing.
- **Predictions**:
  - Applies the loaded model to the test data to generate predictions for the target variable ArrDelay.
- **Performance Evaluation**:
  - Evaluates the predictions using predefined metrics to assess model accuracy on unseen data.
  - Also saves the predictions and evaluation results in a CSV.

It is worth mentioning that the code was developed and tested in Linux. It is recommended that the code be run in such an environment, as there is less support for Apache software such as spark in operating systems such as Windows.

6.2 Application execution

This section provides a step-by-step guide to executing the Spark application in Linux:

- First make sure that you got Apache Spark installed and properly configured in your system and place the test dataset in an accessible directory.
- Use the following command to execute the application: `spark-submit app.py <path_to_test_data>` Where `<path_to_test_data>` is the path where the test data is stored. Also the app is able to process different test_data for this in the <path_to_test_data> is necessary to indicate the different files you want to use. Example: `spark-submit app.py “data/2008.csv” “data/2006.csv” “data/2007.csv”`
- After the execution the application will produce in the results folder both different csv with the prediction of the result called predictionsX.csv being X the filename and metricsX.txt being X the filename where different evaluation metrics are displayed.

7. Conclusions

In conclusion, this project successfully demonstrated the use of Apache Spark for scalable data processing and machine learning to predict flight arrival delays. By leveraging robust feature engineering and systematic model validation, the Linear Regression model was identified as the best performer, achieving a balance between accuracy and computational efficiency with the lowest RMSE. The developed Spark application further ensures the practical applicability of the solution, enabling seamless predictions on unseen data. This project underscores the potential of Big Data technologies in addressing real-world challenges, paving the way for future enhancements such as incorporating additional features like weather or expanding scalability to handle even larger datasets.