Big Data Report

Dániel Laki, Yacine Merioua, Federico Paschetta

January 2024

As reported in the project statement, our model has been developed using, as example dataset, the first csv file, *1987.csv*. Some choices have been made due to this dataset characteristics, but all main choices are automatically adapted to different datasets or combination of them too.

1 Data Cleaning

1.1 Forbidden Variables

During first step of data cleaning, forbidden variables have been removed from the dataframe as said in project statement. Those variables are:

- ArrTime
- ActualElapsedTime
- AirTime
- TaxiIn
- Diverted
- CarrierDelay
- WeatherDelay
- NASDelay
- SecurityDelay
- LateAircraftDelay

1.2 Null Variables

Then, after computing percentage of NA values present in dataframe for each variable, other variables have been dropped too, as they had a frequency of null way above the threshold (here we considered 25% as limit, but it can vary). Their column, in fact, was only made of NA, so it was necessary to delete them. Those are:

- TailNum
- TaxiOut
- CancellationCode

1.3 0s in Target Variable

As part of data cleaning we needed to make sure that NA actually had the meaning of "Not Available" data and not 0 for our target variable. In order to assure this we checked if any 0 was present in the ArrDelay column and, as more than 60.000 rows had that value, we got out of that doubt.

1.4 Rows with NA values

After removing fully null columns, the vast majority of NA values were removed, but some values were left, sparse in some of the dataframe columns. Considered the huge amount of data (more than 1.5 million of rows), we thought that leaving out all rows containing NA values (just a bit more than 20.000) was not a big loss. At this point the dataframe was rid of nulls.

1.5 Duplicates

Next step we did has been removing all duplicates in the dataframe, filling space and not giving additional information.

1.6 Unique Variables

At last, we needed to check if there were present variables with only one value in the whole dataframe considered. In order to detect them we used Spark aggregate function *countDistinct*. With this tool following variables have been detected as having only one value:

- Year
- Cancelled

Those variables, in particular, are dropped because of dataset characteristics, as they would remain in the model if the whole dataset was considered. Those features are not hard-coded so, if a dataset including many different years is loaded, variable Year would be included automatically.

2 Data Processing

2.1 Variable Grouping

Soon after data cleaning, a critical aspect is grouping variables in categorical and numerical (or qualitative and quantitative). Immediately turns out that the

only categorical variables are the one having letters, as in other cases numbers do not represent strings but quantitative values. We classify as categorical variables UniqueCarrier, Origin and Dest, while all other variables are considered numerical.

2.2 Exploratory Data Analysis

In order to detect eventual problems of multicollinearity and insights from data, it has been considered useful to have a simple exploratory data analysis. We provided also plots and maps, but they're commented in the code because they can be visible in a Python notebook, but not in a simple command line spark execution.

2.2.1 Inspecting Trends

First type of analysis concerned trends, in particular yearly or monthly. In order to perform this simple analysis, at first, the grouping variable (year or month) is selected based on which attribute is still in the dataframe (if observations are only related to one year we take Month, otherwise we take Year) and average aggregation function is performed on grouped dataset. Once we have the correct data, we plot them in a simple linechart.

2.2.2 Inspecting Correlation

Another important analysis step is detecting high correlations between variables. Once correlation between two variables is very close to 1, this means that those variables share quite the same predictive power, so it's enough to have one of them. We thought 0.9 is a good threshold to consider high a certain correlation. At this point we created an heatmap with correlations between variables and then, automatically, the script recognizes variables with correlation higher than 0.9. This step is the only one in the whole project that the developer/scientist should modify based on the datasets he put as input. This happens because maybe 0.8 correlation can be considered too high or because using different datasets with different data can cause different variables to have high correlations. In our plan we decided to remove two variables, *CRSElapsedTime* and *CRSDepTime* because they appeared to have respectively correlations of 0.99 with column *Distance* and 0.98 with *DepTime*.

2.3 Variables Transformations

2.3.1 Cast to Integers

At first, as all variables are imported as string, numerical variables need a transformation and all columns previously mentioned, except categorical ones (UniqueCarrier, Origin, Dest), are casted to integer, in order to be used for computations.

2.3.2 Normalizing Time Variables

Looking at additional dataset variable explanation file, it's said that all columns containing some type of time are formatted as hhmm, having the first (eventually) two characters for hours, and second two for minutes. It's clear that this format can't be treated as an integer due to differences between base 100 and base 60 structure. For this reason we take values from the following columns, just casted as integers:

- DepTime
- CRSDepTime
- CRSArrTime

Function transforms each combination of hours and minutes to correct number of minutes. Minutes as measure unit make normalization possible.

2.3.3 One Hot Encoding

At this point it is useful to transform categorical variables in a way that the model can process. That way can be obtained by applying a One Hot Encoding to those categorical variables. This is obtained using PySpark StringIndexer and then OneHotEncoder.

2.4 Variables Created

Between other datasets given with our main one, there was *airports.csv*, which gives geographic coordinates of airports with its IATA code. As our dataframe contains in columns *Origin* and *Dest* IATA codes of airports where the flight took of or landed on, we thought that joining those two datasets could add some information to the model. To do that we join the two datasets twice, once joining them with the condition (Origin==IATA) and the second time with condition (Dest==IATA). After removing extra columns the dataframe finds itself having two more columns, as it has *OriginLat*, *OriginLong*, *DestLat*, *DestLong*, while dropping *Origin* and *Dest*. At this point those new four become numerical features, leaving only UniqueCarrier as categorical one.

But after handling a bit with those new variables, we notice that the whole process becomes so unmanageable due to its slowness, so at this point we have two different models, one considered better but hard to handle and the other a bit worse, but usable. The first will have only *UniqueCarrier* as categorical variable (OneHotEncoded) and *OriginLat*, *OriginLong*, *DestLat*, *DestLong* as numerical values. The second one, instead, will have *UniqueCarrier*, *Origin* and *Dest* columns as categorical variables, encoded with OneHotEncoding.

3 Modeling

3.1 Variables Excluded

In our execution based on *1987.csv* data, during the different process steps, we excluded from the model the following variables:

- ArrTime
- ActualElapsedTime
- AirTime
- TaxiIn
- Diverted
- CarrierDelay
- WeatherDelay
- NASDelay
- SecurityDelay
- LateAircraftDelay
- TailNum
- TaxiOut
- CancellationCode
- Year
- Cancelled

3.2 Variables Included

At this point, after the drop of variables previously seen, 14 variables are kept, 13 explanatory variables, plus the response one. Those are:

- Month
- DayofMonth
- DayOfWeek
- DepTime
- CRSDepTime
- CRSArrTime

- UniqueCarrier
- FlightNum
- CRSElapsedTime
- ArrDelay
- DepDelay
- Distance
- *Origin
- *Dest
- *OriginLat
- *OriginLong
- *DestLat
- *DestLong

As we already said, we used this configuration with *1987.csv* file as starting dataset because, for example, with *2008.csv* variable *TailNum* would not be dropped, as it doesn't have only null values. Last six vars have a (*) near their name because first two are present as categorical vars in the fastest version, while the other four as numerical vars in the better one.

3.3 Machine Learning Techniques

As the problem needs to find a quantitative value (the arrival delay), regression techniques are the ones able to solve it. We used two different algorithms which are LinearRegression and RandomForestRegression, with Spark LinearRegression and RandomForestRegressor functions. For the second one we used a standard tree factor of 100.

3.4 Model Validation

At first, we decided to use train-test split (with 80/20 proportion) in order to fit the model and to use it to predict our test data arrival delay. Then we thought of implementing a crossvalidation technique that surely would have given better results, but we left it commented in the code as it never gave our final result, even after 3 hours of execution.

4 Evaluation

To evaluate in the best way possible the model, we decided to use two metrics, RMSE and R-squared, as both shed light on different aspects of model. So we thought it was the best to have them, in order to give both the error and the percentage of the variability in data explained by the model. We calculated them with RegressionEvaluator function in Spark, with parameter metric set to rmse or r2. Then we decided to add other metrics like MAE (Mean Absolute Error), MSE (Mean Squared Error) and Var (Explained Variance), all just changing parameter 'metric' in RegressionEvaluator object.

5 Results

As said before we have two different models available, our main one is *project_num.py*, having coordinates of origin and destination airports (we will refer to it as "numerical"), while the one with origin and destination variables handled as categorical variables (we can refer to it as "categorical") is available at *project.py*. Similarly to cross-validation, also RandomForestRegression gave us problems, writing continuously as log:

```
WARN MemoryStore: Not enough space to cache rdd_819 in memory!
(computed 108.3 MiB so far)
```

RandomForestRegression code is available in both models but we think that maybe higher performance machines are needed to handle it. LinearRegression model, on the other hand, worked really fast with train-test split in our "categorical" model, and got 15.27 of RMSE and 65.03% of $R\hat{2}$. LinearRegression fitting and validation for "numerical" model with train-test split, was really slow, employing 1.5 hours but obtaining a little better performance than other model. It got 14.62 of RMSE and 67.68% of $R\hat{2}$. Then we tried to use the model with another csv file, in particular we tried with *2008.csv*. With this dataframe results were promisingly better: the "categorical" model got 10.05 of RMSE and 93.23% of $R\hat{2}$, while "numerical" one, due to laptop limited resources, never finished its execution and, after two hours of waiting, we considered it as a failing execution.

6 Model Instructions

6.1 Folder Structure

Provided folder *Big_Data_Project* has the structure illustrated below.

```
.
|-- data
|   |-- 1987.csv
|   |-- 2008.csv
```

```
|   |-- airports.csv
|-- authors.txt
|-- requirements.txt
|-- report.pdf
|-- src
    |-- main
        |-- python
            |-- project_num.py
            |-- project.py
```

6.2 Data

Data files need to be inserted in the corresponding *data* folder, where in the picture there are *1987.csv* and *2008.csv*. It is mandatory to have at least one data file inside the folder (the one which is passed as parameter in terminal command) and, for "numerical" version, to have *airports.csv* in the same folder.

6.3 Execution

At first, inside the file *requirements.txt* are present all libraries needed to run the script. In order to execute the script it's needed to have every file in the right place and a running Python Environment. Then, opened the terminal in the folder Big_Data_Project, the execution is triggered by the following command

```
spark-submit --master local[*] src/main/python/project_num.py 1987.csv
```

With this command it is executed the 'numerical' model, with the data file *1987.csv*. The faster 'categorical' model can be executed just removing the substring *_num* between *project* and *.py*. It can be executed with any data file instead of *1987.csv* or can even be added to it.

```
spark-submit --master local[*] src/main/python/project_num.py 1987.csv 2008.csv
```

This command will execute the script with both dataframe merged.