Building a Big Data Machine Learning PySpark Application for Flight Delay Prediction

18BCE079 (Ishaan Buch)
18BCE104 (Maher Thakkar)

We will demonstrate how to create a Spark application that allows you to create a machine learning model for a real-world problem using real-world data in this report: Commercial flight arrival delays can be predicted.

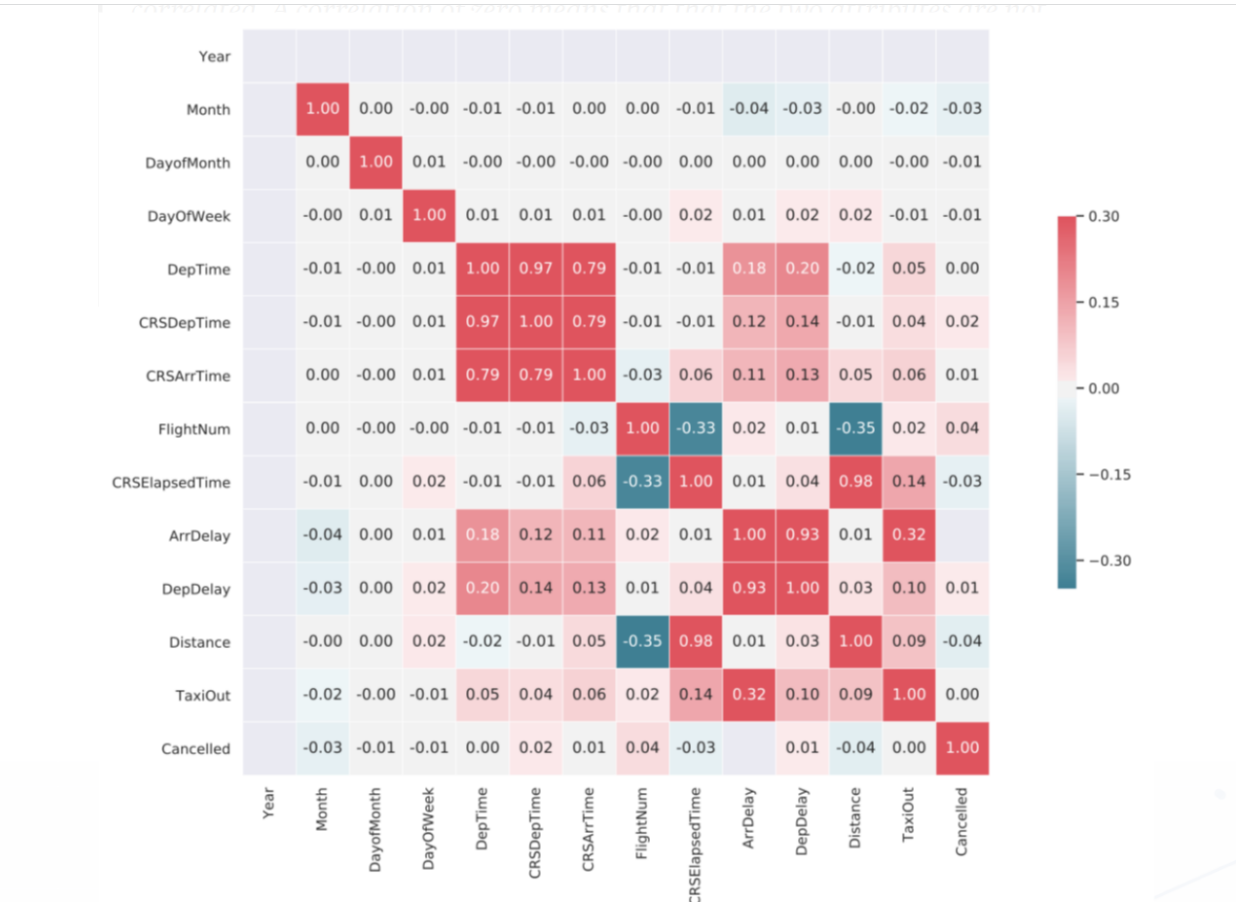
Several machine learning approaches were applied, together with a variety of input combinations, to discover the optimal model. Linear Regression, Random Forest Trees, and Gradient-Boosted Trees are the machine learning algorithms investigated. Scala will be used to write all of the code.

Problem
We want to be able to anticipate the arrival delay of a flight based on past data using only information available before the flight takes off. We will only utilise regression techniques because the goal variable (ArrDelay - arrival delay) is a numerical value; we will explain this more below.

Dataset
The US Department of Transportation provided the data, which can be downloaded here. It contains about 23 years of data, and analysing such a large volume of data necessitates the use of big data tools like Spark. Since September 2010, the Department of Transportation has made a list of publicly available data sets available on this website and at http://catalog.data.gov. This new Data Inventory page was launched in November 2013. The creation of this new page satisfies the Open Data Policy's standards as well as the DOT's promise in its Open Government Plan. You'll see that their new inventory is a tad lower than what we have on data.gov. They fixed numerous broken connections for our State Traffic Safety Information datasets when creating our November 2013 public data listing, removing 480 broken links and replacing them with three significantly larger and more valuable datasets.

Exploratory Data Analysis

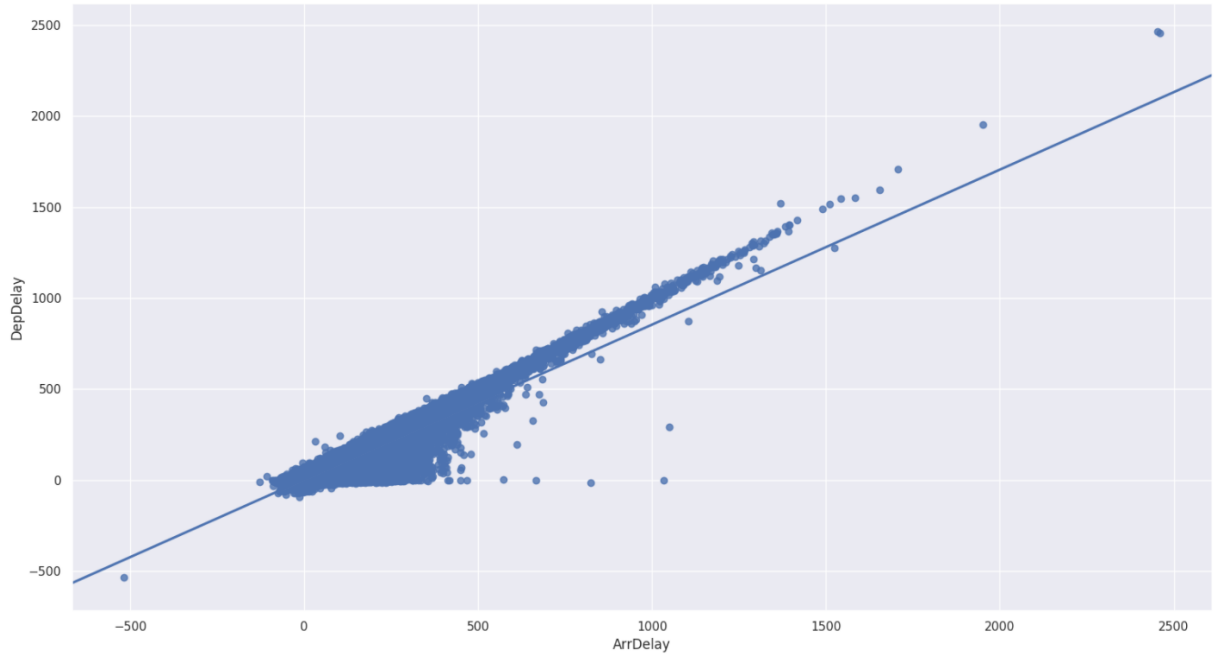
The most crucial aspect of any data science endeavour is a thorough examination of the problem and the data available. Because the majority of the variables are numerical, we'll start by creating a correlation matrix of the numerical variables to see whether there's any association between them.



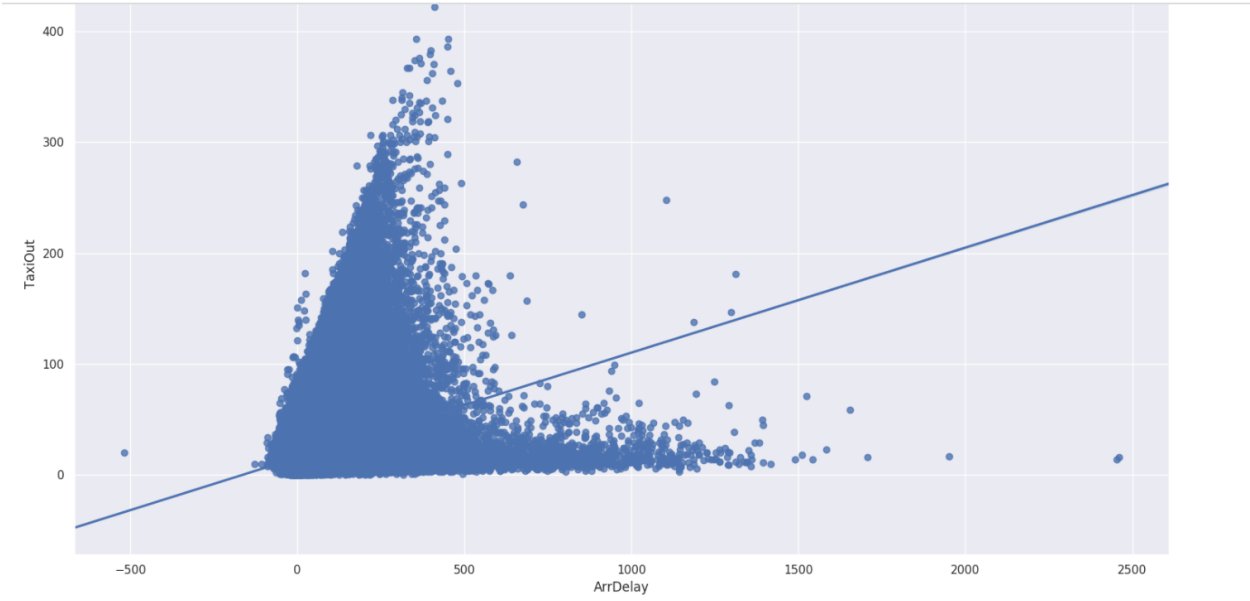
Correlation Explained

The strength of the link between two attributes is described by a correlation. The more positively or negatively the variables are connected, the closer the value is to 1 or -1. A correlation of zero indicates that the two characteristics are unrelated. A coefficient of +1 shows that the two traits are perfectly correlated, indicating that any change in one property is accompanied by an equal and opposite change in the other attribute. A coefficient of -1 indicates that any change in one characteristic is accompanied by a change in the other attribute in the opposite direction.

The DepDelay (departure delay) is substantially connected with the arrival delay, hence the departure delay may be the most suggestive aspect of a flight delay, according to the correlation matrix. We may also notice a weak, but present, correlation between the TaxiOut and the arrival delay if we look a little deeper. As a result, we'll go deeper into these two variables.



Scatterplot between departure delay and arrival delay from the 2008 dataset.



Scatterplot between taxi-out duration and arrival delay from the 2008 dataset.

These two graphs help us better grasp the correlation value that was previously shown.

Exploration: What are the primary causes for flight delays

```
In [8]: cause_delay = sqlContext.sql("SELECT sum(WeatherDelay) Weather,sum(NASDelay) NAS,sum(SecurityDelay) Security,sum(Lat
                                      FROM airlineDF ")

In [9]: df_cause_delay = cause_delay.toPandas()

In [10]: df_cause_delay.head()
```

Out[10]:

	Weather	NAS	Security	lateAircraft	Carrier
0	5739649	28200746	176906	38004942	28808434

We are able to identify the cause of all the flight delays.

Exploration: Which Airports have the Most Delays?

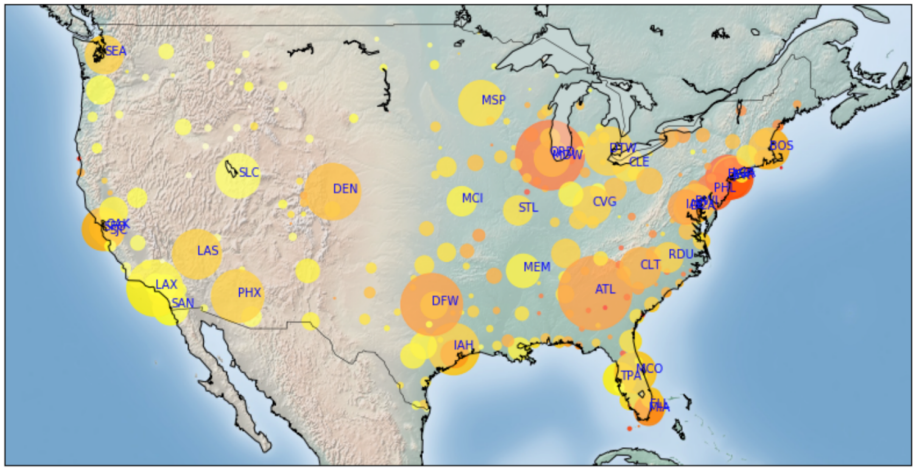
```
In [7]: groupedDelay = sqlContext.sql("SELECT Origin, count(*) conFlight,avg(DepDelay) delay \
                                       FROM airlineDF \
                                       GROUP BY Origin")

        df_origin = groupedDelay.toPandas()

In [9]: df_origin.sort('delay',ascending=0).head()
```

```
/Users/Sally/anaconda/lib/python2.7/site-packages/ipykernel/__main__.py:1: FutureWarning: sort(columns=....) is dep
recated, use sort_values(by=.....)
  if __name__ == '__main__':
```

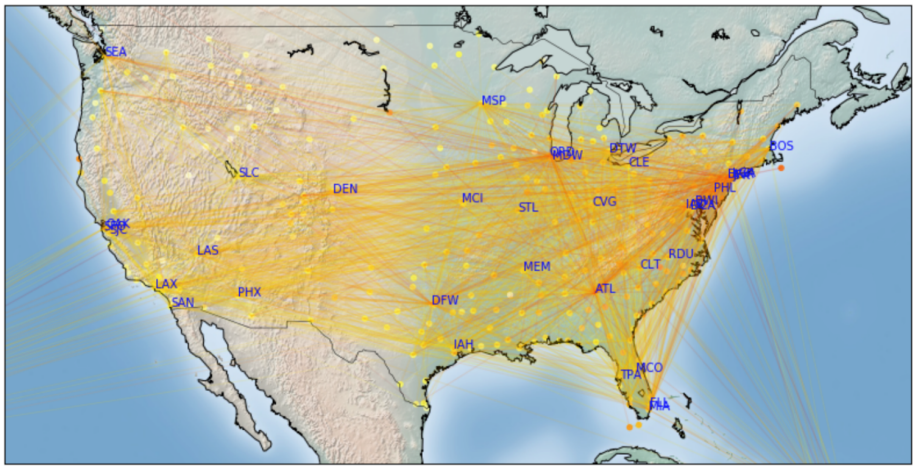
Out[9]:

	Origin	conFlight	delay
272	ACK	277	51.346570
279	PIR	4	45.500000
79	SOP	185	35.859459
81	HHH	944	23.855932
175	MCN	936	23.842949

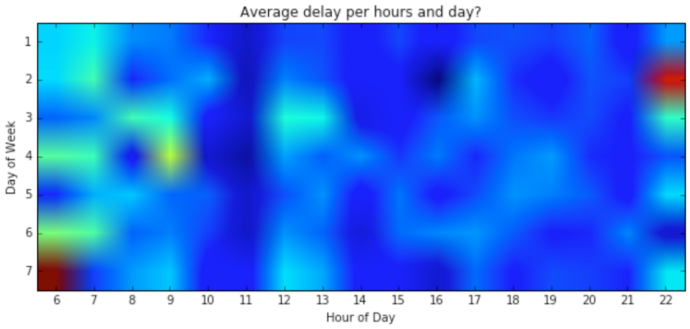
This helps us list the airports which have the most delays.



Each marker is an airport.
Size of markers: Airport Traffic (larger means higher number of flights in year)
Color of markers: Average Flight Delay (Redder means longer delays)



Each line represents a route from the Origin to Destination airport.
The redder the line, the higher the probability of delay.



A clear pattern here: flights tend to be delayed in these situations:

Later in the day: possibly because delays tend to pile up as the day progresses and the problem tends to compound later in the day.

Mornings in first day of week possibly because of more business meetings

Implementation

Loading the data

Using the spark SQLContext, we first read the data from the CSV file. Simply provide a location like /mydata/*.csv to import several CSV files. The data will be loaded into a DataFrame, which is a distributed collection of data with named columns. If we run the spark programme in a cluster, the data will be distributed randomly around the cluster.

In [2]:
```
textFile = sc.textFile('/Users/ishaanbuch/Downloads/2007.csv')
```

In [3]:
```
#remove the header of file
textFileRDD = textFile.map(lambda x: x.split(','))
header = textFileRDD.first()

textRDD = textFileRDD.filter(lambda r: r != header)
```

Then, by using the.drop() function to remove those variables that are only known after the flight has taken off, as well as variables that are overly correlated and may be redundant to the model, the model may become overfit. We also eliminate variables that are unrelated to the arrival delay, such as cancelled flights.

Processing the data

The data processing is greatly dependent on the implementation of the machine learning algorithm as well as the experiment being conducted. The key difference will be in which variables are removed and whether nominal (categorical) values, such as the origin and destination airports, are used.

Further processing was required in order to employ categorical variables in regression techniques. A StringIndexer and OneHotEncoder were used to transform these variables to numerical values. The links provide more information on these two procedures and why they were used.

add a new column to our data frame, DepDelayed, a binary variable:

True, for flights that have > 15 minutes of delay False, for flights that have <= 15 minutes of delay

In [6]:
```
airline_df = airline_df.withColumn('DepDelayed', airline_df['DepDelay']>15)
```

In [7]:
```
# define hour function to obtain hour of day
def hour_ex(x):
    h = int(str(int(x)).zfill(4)[:2])
    return h

# register as a UDF
f = udf(hour_ex, IntegerType())

#CRSDepTime: scheduled departure time (local, hhmm)
airline_df = airline_df.withColumn('hour', f(airline_df.CRSDepTime))
airline_df.registerTempTable("airlineDF")
```

Training the data and obtaining a model

We discovered features with a high linear association, such as departure delay and arrival delay, as shown in the first scatterplot. As a result, Linear Regression will be the first method we examine. This is a quick and easy algorithm that produces good results in certain situations.

We'll set up a pipeline to handle all of the data transformations. This enables us to define all of the stages and have them executed only when the function is called. To train the model, fit() is invoked. The pipeline includes the following steps: translating categorical values (if applicable) to numerical values, combining all of the data into a features vector, and running the machine learning algorithm to generate a model.

The repository contains the code for running the Logistic Regression machine learning algorithm, and these approaches work seamlessly in this pipeline.

Before the data can be used, it must first be separated into training and testing sets, with a 70 percent split used to evaluate the model afterwards. Models should never be tested on the same data that they were trained on, according to the golden rule of model evaluation.

Optimizations

To split the training data into training and validating data, we use the TrainValidationSplit function. Validating data is used to fine-tune the algorithm's parameters. The machine learning method will be trained on the parameters range defined by a ParamGridBuilder. The final model will be built using the best performing set of parameters.

Evaluating the resulting model

The model's performance was assessed using the accuracy metric. The final model accuracy is 85.13%.

Conclusion

During the testing, the Logistic Regression algorithm performs very well. This could be owing to the features' linear relationship with the target variable.

The report in the repository contains the results and additional analysis.
Spark is a very powerful tool, and the code and application shown here can run on a cluster without modification, allowing for the testing of large datasets with more complex algorithms.