**Big data course:**

Spark Application

|  |  |
| --- | --- |
| **Surname** | **Name** |
| Nevado | Julio |
| Nikolskaya | Anna |
| Zabarte | Gorka |

1. Introduction
2. Data preprocessing

Data preprocessing involve the following tasks:

* Data Exploration
* Data Cleaning
* Data Transformation
  1. Data Exploration
  2. Data Cleaning

Before start modeling the data available cleaning it is necessary. First of all, forbidden variables were removed. Besides forbidden variables, FlightNum was also removed since it is a unique identifyer for each flight and due to its specificity, no information could be gained.

Regarding the available columns, some of them had an inconvenient format. Hour variables (DepTime, CRSDepTime and CRSArrTime) had format “1234”, two first numbers being hour and two last, minutes. Each of the previous columns were splitted into hour and minutes, creating 6 new columns, namely:

* DepTimeHour
* DepTimeMinute
* CRSDepTimeHour
* CRSDepTimeMinute
* CRSArrTimeHour
* CRSArrTimeMinute

This is done using withColumn method and using pyspark sql functions (regex\_replace). For each column, the regular expression is the following:

* ([\\d{1,2})(\\d{2}](file://d%7b1,2%7d)(//d%7b2%7d))
  + Meaning: Two groups separated by brackets. \d stands for decimal number. {1,2} means from one to two digits. {2} stands for strictly two digits.
  + For hour columns, we selected first term: $1
  + For minute columns, we selected second term: $2

Therefore, the final set of variables is the following:

* DayOfWeek: nominal
* CRSELapsedTime: numerical
* DepDelay: numerical
* Distance: numerical
* TaxiOut: numerical
* DepHour: numerical discrete
* DepMinute: numerical
* CRSDepHour: numerical discrete
* CRSDepMinute: numerical
* CRSArrHour: numerical discrete
* CRSArrMinute: numerical
* Month: nominal
* DayOfMonth: numerical
* ArrDelay: numerical
* UniqueCarrier: nominal
* TailNum: nominal
* Origin: nominal
* Dest: nominal

Next, missing values are addressed. Since we want to predict the arrival delay, instances with missing values on this variable are useless. In addition, these values are due to cancelled flights so, after removing these instances only, cancelled variable only contain 1s (cancelled flight) so Cancelled and Cancellation Code variables are useless since no flights are cancelled in the new dataset. After this, only TailNum attribute contains missing values for two instances, which are also dropped.

Finally, the obtained dataset is divided into train (70%) and test (30%).

* 1. Data Transformation

The resultant dataframe contains several nominal attributes (DayOfWeek, Month, UniqueCarrier, TailNum, Origin, Dest). These attributes are going to be converted to a numerical type. This is done by using preprocessing tools provided by Mllib in pyspark.

First, StringIndexer allow us to convert string attributes to indexes. Then, OneHotEncoder converts these indexes into vectors.

Finally Univariate Feature Subset Selection is performed, again using the tools provided by Mllib in pyspark. This is done since some regression models will be used. It is known that multicollinearity can clearly affect how we understand relationships between explanatory and response variables.

First, VectorAssembler allow us to create a new column (a vector column) containing all values for each instance and each attribute. This new column will be the input to UnivariateFeatureSelector. A selection threshold of 0.05 is used and, since all categorical attributes have already been converted to numeric types, both input and output types are set to numeric.

This is done using a pipeline called the “preprocessing\_pipeline”. This pipeline is defined by a set of the previously mentioned stages in the following order:

1. StringIndexer: we get indexes for categorical features
2. OneHotEncoder: convert each index into a vector of 0s with only one non-zero value, corresponding with the index value. The least frequent value is an exception to this since it is codified as a zero vector.
3. VectorAssembler: converts the data columns of each instance into sparse vectors.
4. UnivariateFeatureSubsetSelection: performs an univariate feature selection over the vector created by VectorAssembler and outputs a new column of vector type containing the selected features.

This pipeline es fit over the training set. Once the pipeline is fit, our train and test sets are transformed. Once we get our preprocessed train and test sets we are ready for data modelling. However, since both generated sets are going to be used in three different models, we are going to persist them in order to avoid repeated operations over dataframes.

1. Data modelling
   1. Decision Tree Regressor

The first model used in the present paper is a Decision Tree regressor. The input features are contained in the column “selectedFeatures” which, as said, is the result of UFSS. The output will be stored in a new created column, “prediction”.

* 1. Linear Regression

Linear regression model was chosen as one of the modesl for the project because it allows to understand the strength of relationships between the variables. Using R-squared metric it can give clarity on how much total variability in the data is explained by the model. It can as simplify the understanding on which predictors in the model are statistically significant .

After the loading and processing of data steps described above, the vector with following features was created.

|  |
| --- |
| DayOfWeek, CRSElapsedTime, DepDelay, Distance, TaxiOut, DepTime\_index, CRSDepTime\_index, CRSArrTime\_index, Origin\_inde, Dest\_index |

To train the model the regressor takes in the input of features (the vector decribed above) and the name of the colun that represents the feature to be predicted, in this case '*ArrDelay*'. The training is performed on train\_data which includes 7/10 of data.

Following, the model was applied to the test dataframe. The prediction calculated can be found in "Valdating model" section.

1. Validation

The error metric used to assess the performance of our models is RMSE, since this metric penalizes extremely large errors which is what we want. In fact, the impact of predicting the exact number of delayed minutes for a flight makes no sense. If the delay is 15 minutes and we predicted 13, we have an outstanding prediction. We do not want to exactly predict the number of delayed minutes but the range in which this number of minutes is going to be. We can allow an error of 3 or 4 minutes, but not predicting a 20 minute delay when in fact the delay is 10. In other words, we opt for penalizing large errors more than small ones since not every error is going to have the same impact.

* 1. Decision Tree Regressor

As it is known, the main parameter in a Decision Tree is its depth. Correctly validating this property is crucial, since a very low value would lead to an excessively general model which underfits the data. An extremely high value would yield very low error rates, but our model would be overfitting data, unable to generalize to new data.

In order to obtain the best value for this parameter, cross validation will be used. A grid of parameters is created to test several “maxDepth” values, selecting the model which offers a better performance. The values tested are 5, 7, 11, 13 and 15.

The best model was achieved for a maximum depth of XXXX.

* 1. Linear Regression

The metrics metricsof the model found after performing the training are presented in the screenshot below.

Text

Description automatically generated

* **R-squared** measures the strength of the relationship between the input vector of features (*’features*’) and the dependent variable(*’ArrDelay’*). In other words it answers the question: "How much of the data does this model explain? ". The value of 0.939473 means that 93% of the data represent the variance of the dependent variable (*’ArrDelay’*).
* **Root Mean Square Error (RMSE)** is the square root of the average of the squared difference of the predicted and actual value. It is used as ameasure of accuracy, to compare prediction errors of different models. The model with the lowest RMSE is the best one.
* **Coefficience** decribes relationship between the dependent variable (x in our case *vector*) and independent (y in our case ‘*ArrDelay*’) variable. Results presented below show that the attributes with the most significant influence (0ver 0.9) on *ArrDelay* are *DepDelay* are *TaxiOut*. The strength of relationship between *CRSElapsedTime* and our independent varible is 0.2.

Text

Description automatically generated

Prediction metrics compare the predicted value by the model with the actual value of ArrDelay of the testing set.

Table

Description automatically generated

CrossValidation allows us to compare different machine learning models. Rather the spliting the data into train and test sets crossvalidation separates data into the f*olds* or "blocks" and uses one at the time,and summarizes the results obtained in the end.

For the performing cross validation for linear Regression the amount of 5 folds was chosen. The metrics of the best model found after performing the validation are presented in the screenshot below.

Graphical user interface, text, application

Description automatically generated

The metrics from CrossValidation demonstrate metrics closed to ones from the Linear regression with train/test set . The the same explanation apply in this case.

**Coefficience** as explained above, decribes relationship between the dependent variable. Results presented below show the significance compared with Linear regration with train/testresults, the attributes with the most significant influence (0ver 0.9) on *ArrDelay* are *DepDelay* are *TaxiOut.* The strength of relationship between *CRSElapsedTime* and our independent varible is -0.2, this time negative.

Text

Description automatically generated