# FIT5202 Data Processing for Big Data

## Assignment 2: Phase 3

## Developing a binary classification model to predict flight delays for travel between between Australian airports


**Group 8**

**Wasnik Malla 29891191**
**Angus McCall 30691001**
**Peter McEniery 30280958**
**Tod Nestor 10443436**

# Introduction

Flying business class is one of the most luxurious ways to travel: waited on hand and foot, room enough to take part in the complimentary yoga class and seats as soft as the clouds you soar above! A vivid contrast awaits passengers merely fifteen metres and a thin curtain of fabric away in economy: shoulder to shoulder seating, one blanket per row and an unspoken rule of “we’re all in this together. Please… please don’t recline your seat”

The above shows two completely different worlds, but the great equaliser in air travel is flight delays. Be you first class, economy or even pilot, flight delays affect all those travelling and the airlines themselves.

This project will analyse Australian flight delay data to uncover systemic associations between flight delay, airports, destinations and monthly weather cycles in an attempt to create a predictive tool. The dataset (Domestic Airlines - On Time Performance, 2020) is a csv file of historical flight data, spanning January 2004-March 2020. The dataset consists of 74 features related to 6,296,897 flights over this time interval travelling between Australian domestic airports. The monthly aggregated flight delay data spanning January 2004-March 2019 will be used to train the models and the records spanning May 2019-March 2020 will be used to test the models. The large size of this dataset provided significant challenges to those group members running on machines with limited RAM, necessitating paring back the data set by taking a random sample of 20% prior to further processing (in hindsight, the fraction should have been even smaller).

This phase 3 report describes the models and the attendant data pipelines constructed to train a number of different supervised learning models that predict whether or not a particular flight will be late.

# Assumptions

The “No Free Lunch Theorem” (NFL) of optimisation theory states that there is no “one-best” method that is universally applicable to all optimisation problems. In a landmark paper (Wolpert, 1996), used mathematical proof to show that Supervised Machine Learning was a special case of optimisation, and as a consequence, the immediate corollary of the No Free Lunch Theorem applied, i.e. there was no ”one-best” Supervised Machine Learning Algorithm (SMLA), suitable to all classification problems.

# Steps

Due to the NFL, the group was unable to predict *a priori*, which Binary Classification SMLA would be most suitable to predict flight delay. Consequently different Group 8 members investigated different SMLA models: Logistic Regression, Random Forest , Gradient Boosted Tree and Multilayer Perceptron Classifier. Each of these investigated models are assessed with a *weighted precision* performance measurement to effect the model selection process.

It is important to note that *four* Jupyter notebooks accompany this report - one for each of the models described above. The four notebooks share mostly common code, but tailored to the specific needs of each model.

# Justification

The candidate ML classification models are tested against flight data with only two labels (delayed by 15 minutes or more, or on-time), consequently the problem domain fits nicely within that of Binary Classification. The outcome of a binary classification algorithm is readily assessed using a Confusion Matrix, where predictions are classified into True Positives (number of correct identified flight delays), False Positives (number of predicted late departures that were on-time), True Negatives (number of predicted on-time departures that were on-time) and False Negatives (number of predicted on-time departures that departed late).

| | | Predicted Late: 0 | Predicted Late: 1 |
|---|---|---|---|
| Was Late | 0 | TN | FP |
| | 1 | FN | TP |

**Table 1.** The confusion matrix for the binary classification of late departures.

The Spark DataFrame-based ML API is used instead of the older RDD based MLlib API to develop and implement ML models, because this enabled Group 8 to set up a pipeline (pipelines are not available to the MLlib API user). The feature engineering steps for example, indexing and one-hot encoding the categorical features, normalising the norm of feature vectors, or using Principal Component Analysis (PCA) for dimension reduction, as well as the model evaluation steps, were common to all SMLAs considered. Consequently,

the group needed only to change the estimator and transformer along with the attendant parameter grid in the pipeline in order to compare the candidate ML models.

An additional advantage of the Spark 2.x ML API over the older MLlib API is that the former uses DataFrames (a tabular structure with named columns of a specific data-type, so familiar to Python and R users). The older MLlib API is built directly on top of Resilient Distributed Datasets (RDDs) which then requires the development of functional programming pipelines composed of higher order functions.

# Challenges

The flight delay data was only available as monthly aggregates; therefore day of the week, or the time of the day, could not be used to predict flight delay. In Phase 2, the monthly counts of late departures and ontime departures for each route were replicated the appropriate number of times. A label column was also added to the dataframe with an entry of one for a late departure and a label of zero otherwise. Due to the poor time resolution of the dataset it was augmented with monthly aggregates of meteorological data and jet fuel prices (IndexMundi, 2020), and the resultant dataset was then stored as a parquet file (resulting in 6,296,897 flight records), to be read into the models for Phase 3.

The Phase 3 dataset is unbalanced, with more than five times as many on-time departures as delayed departures. Consequently, a random sample (without replacement) of one fifth of the on-time departures was taken from the training data, then concatenated (row-wise, using a union set operation) with the delayed departures. The new training set then had roughly equal numbers of delayed and on-time flights.

It is well known that whenever a model is overtrained on the training dataset (by choosing too many model parameters in order to reduce the bias) it will learn the noise as well as the signal in the data. In this case, the overtrained model becomes highly sensitive to even small changes in the training data, i.e. the overtrained model develops high variance. Our group has used cross validation and principal components analysis to minimise the effects of overtraining.

# Model Evaluation

The testing data (comprising the most recent 12 months) has not been balanced in the same way as the training data, so the Group 8 models attempt to predict a relatively unlikely event. Consequently, a weighted precision measurement (described below) is used to assess model efficacy.

Strangely, the Spark ML DataFrame API does not have an inbuilt function to return a Confusion Matrix (Table 1). Therefore Group 8 wrote a custom function for this purpose.

With reference to Table 1, the weighted precision (*WP*) weights the precision by the fraction of positives present in the test set, as defined below :

- $NPV = \frac{TN}{TN+FN}$ (the Negative Predictive Value)
- $PPV = \frac{TP}{TP+FP}$ (the Positive Predictive Value i.e. precision)
- $NF = \frac{FP+TN}{Total}$ (the fraction of on-time departures in the test set)
- $PF = \frac{TP+FN}{Total}$ (the fraction of late departures in the test set)
- $WP = NF \times NPV + PF \times PPV$ (weighted precision)

# The Machine Learning Models

## Logistic Regression

Let us assume that the training data consists of $M$-flights and the on-time departure of the ith flight is described by $y_i,\ 1 \le i \le M$ ($y_i = 1$ if the ith flight is late, otherwise $y_i = 0$).
The ith flight has $N$ explanatory variables (the feature vector) $x_{ij},\ 1 \le j \le N$.

The Group 8 Logistic Regression model equates the odds of the ith flight departing late $\frac{\pi_i}{1-\pi_i}$ $1 \le i \le M$ to the exponential expression $e^{\alpha+x_{i1}\beta_1+\ldots+x_{iN}\beta_N}$ where the vector of weights $\beta = \{\alpha, \beta_1, \ldots, \beta_n\}$ is to be learned by our ML model by **maximising** the probability $\prod_{i=1}^{n} (\pi_i)^{y_i}(1-\pi_i)^{1-y_i}$ (the joint probability of N independent Bernoulli Random Variables).

By rearranging $\frac{\pi_i}{1-\pi_i} = e^{\alpha+x_{i1}\beta_1+\ldots+x_{iN}\beta_N}$, the probability $Pr(y_i = 1|x_{i1}, x_{i2}, \ldots, x_{iN}) = \pi_i$ that the ith flight will leave late is given by $\pi_i = \frac{1}{1+e^{-(\alpha+x_{i1}\beta_1+\ldots+x_{iN}\beta_N)}}$, so $\beta_j > 0$ increases the chance of the flight leaving late and $\beta_j < 0$ decreases this chance (the value of the features $x_{ij} \ge 0$ in our model).

The maximum log-likelihood of the observations given the model parameters then corresponds to choosing $\beta = \{\alpha, \beta_1, \ldots, \beta_n\}$ to **minimise** the cost function $\frac{1}{M}\sum_{i=1}^{M} f_i(\alpha, \beta_1, \ldots, \beta_n)$ where $f_i(\alpha, \beta_1, \ldots, \beta_n) = ln(1 + e^{-y_i(\alpha+x_{i1}\beta_1+\ldots+x_{iN}\beta_N)})$. Finally, to avoid overfitting the model, a regularisation term (the Euclidean norm of the model parameters) is added, so that the cost function to be minimised is now $\frac{1}{M}\sum_{i=1}^{M} f_i(\alpha, \beta_1, \ldots, \beta_n) + \lambda|\beta|^2$ where $\lambda$ is typically a small positive real number, for example $\lambda = 0.01$ in most models considered here.

Spark Dataframe SQL transformations were applied to label routes (routeID) in the order of the volume of traffic (i.e. routeID=1, had the least traffic, etc). It was assumed that there may be an association between volume of traffic carried by a particular route, and the chance of a delayed flight.

| feature | logistic regression coeff |
|---|---|
| nonWeatherFeatureVectors_Airline_Vec_Qantas | 0.375965 |
| nonWeatherFeatureVectors_Airline_Vec_QantasLink | 0.353675 |
| nonWeatherFeatureVectors_Airline_Vec_Skywest | 0.214355 |
| nonWeatherFeatureVectors_Airline_Vec_Jetstar | 0.0488989 |
| nonWeatherFeatureVectors_Fuel_Price | 0.00102749 |
| nonWeatherFeatureVectors_routeID | 0.000530108 |
| (Intercept) | −0.00974174 |
| nonWeatherFeatureVectors_Airline_Vec_Regional Express | −0.103597 |
| nonWeatherFeatureVectors_Airline_Vec_Virgin Australia – ATR/F100 Operations | −0.10947 |
| continuosWeatherFeatures_0 | −0.112921 |
| nonWeatherFeatureVectors_Airline_Vec_Virgin Australia | −0.154316 |
| nonWeatherFeatureVectors_Airline_Vec_Virgin Australia Regional Airlines | −0.176453 |
| nonWeatherFeatureVectors_Airline_Vec_Tigerair Australia | −0.271731 |
| nonWeatherFeatureVectors_Date_Num | −0.582125 |

**Table 2:** Features included in the logistic regression and their corresponding coefficients.

A distinct advantage of the logistic regression model (compared to the other ML algorithms investigated here) is that it enables the modeller to look back and identify (i.e. infer) those explanatory variables (i.e. features) that are most associated with delays. *Positive* coefficients *increase* the chance of delay, and *negative* coefficients *decrease* the chance of delay.

With reference to Table 2:

- The carrier with the most number of flights has the largest positive association with an increased possibility of flight delay. It is possible that the data should have been standardised to account this bias.
- There is a very weak positive association between routeID and the chances of late departure. As expected, the routes carrying the most traffic are slightly more likely to suffer a delayed departure.
- The large negative coefficient for Date_Num indicates that the possibility of a delayed flight is decreasing over time on average.
- The very weak positive association between fuel price and delayed flights indicates that fuel price is not a powerful predictor of flight delays.
- A linear combination of weather features has been bundled into a single variable continuousWeatherFeatures_0. As this variable increases, the chase of a delayed flight goes down.

```
Area under ROC: 0.5549496002652068
Accuracy: 0.6452245291022275
Weighted Precision: 0.6843316200270014
Weighted Recall: 0.6452245291022275
F1: 0.6619162296415152
Accuracy: 0.6452245291022275
Precision: 0.2682888601543827
Recall: 0.3581254249699252
F1: 0.3067652329749104
=================== Confusion Matrix ==========================
##########| Predicted = 0                        Predicted = 1
----------+-----------------------------------------------------
Actual = 0| 49434                                18674
Actual = 1| 12272                                6847
===============================================================
```

**Figure 1:** 10 Fold Cross Validated Logistic Model with 68.43% weighted precision with a Threshold=0.53.

## Random Forest

A random forest classification is a machine learning method that implements multiple decision trees in order to classify the data.

Firstly, a decision tree within a random forest is a method to classify objects by passing through a series of binary decisions. A very simple tree is shown below.

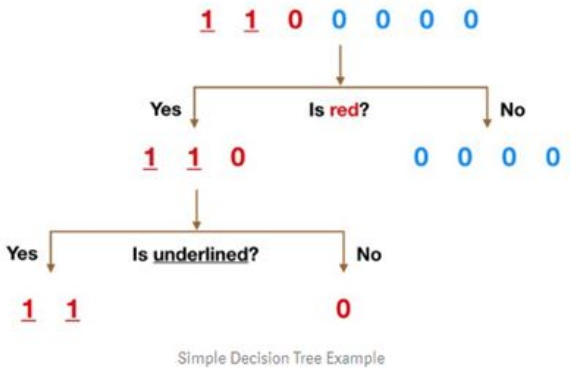


**Figure 2.** A generic decision tree

In Figure 2, the first option divides the data into red = TRUE and blue = FALSE. In the next node, another true/false argument is met, and the data is sorted. The end of the branches on a decision tree is the leaf node. Above these are “pure”; meaning there is exactly one type of data in the node. While the data is clean in the example in Figure 2, it is undesirable for this to happen with training data for fear of overfitting.

A random forest takes the simplicity of this decision tree and implements the tree multiple times to create many decision trees, hence is known as a ‘forest’.

During the training phase, the data will be applied to the trees and the tree will have a range of features to pick from to determine how to split the data most efficiently. However, if we just

did that for every tree for all the same data, the trees would be identical. The process is randomized in two ways.

- The amount of data is the same as applied to all trees, but it is a sample of the same size with replacement. This means it's unlikely that any of the trees will have the same data, especially if when dealing with large amounts of data.
- The range of features available to the trees to use at the decision nodes is different for each tree. That means that even though there may be a clear feature selection that divides up the data efficiently across the dataset, it won't be available to all the trees, so the first node in these trees will be different.

With randomization, there are different results for different trees. Some trees will classify as delayed and some will classify as on time, for the same initial data input. The crux of the random forest however is that the prediction is based on the aggregated results for the individual trees in the model. The prevailing result from the aggregation is used to make a classification.

The following code displays how a Random Forest is implemented in Spark/Scala:

```
val randomForestClassifier = new RandomForestClassifier()
  .setImpurity("gini")
  .setMaxDepth(3)
  .setNumTrees(10)
  .setFeatureSubsetStrategy("auto")
  .setSeed(12345)
```

**Figure 3.** Choosing the random forest model parameters.

The attribute Max Depth indicates how many nodes the tree will travel down before reaching the leaf node.

The attribute NumTrees is the number of trees chosen for the model. Usually training error goes down as NumTree increases, however care must be taken to ensure that the model is not overtrained. Cross Validation has been employed to avoid overtraining.

The Random Forest algorithm uses one of two methods to evaluate which features to use; Gini or Impurity. Gini is preferred in most cases as it's less computationally intensive (Yiu, T, 2019). Both these methods attempt to measure the information gained by comparing features, the one with the highest information gain is the feature chosen for that node.

```
Area under ROC: 0.5866672840594541
Accuracy: 0.4889045316593005
Weighted Precision: 0.7023629917433207
Weighted Recall: 0.4889045316593005
F1: 0.5249495856299868
Accuracy: 0.4889045316593005
Precision: 0.2572132164740557
Recall: 0.68916251039069
F1: 0.3746116916125389
=================== Confusion Matrix ==========================
##########| Predicted = 0                      Predicted = 1
----------+-----------------------------------------------------
Actual = 0| 29102                              38307
Actual = 1| 5983                               13265
===============================================================
```

**Figure 4:** 10 Fold Cross Validated Random Forest Model with 70.27% weighted precision.

## Multilayer Perceptron Classifier

A multilayer perceptron classifier (MLPC) is a type of neural network model. It classifies an output layer of neurons that activate depending on the signals from an input layer of neurons, propagated through an arbitrary number of hidden layer neurons with different weightings. These weightings are learnt by the model with training data and determine which output neurons are activated. A generic MLPC is shown in the figure below (Databricks, 2020):

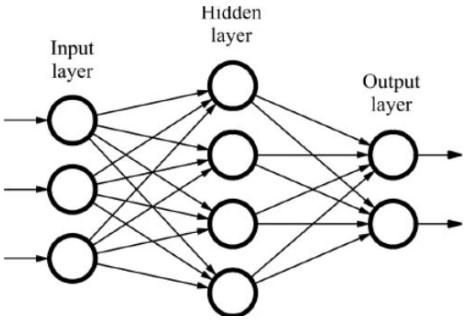


**Figure 5.** A generic multi layer perceptron binary classifier

Principal Component Analysis was used to determine nine linear combinations of flight data features describing the most variation in the dataset, and these were fed into the input layer, consisting of nine neurons. The MLPC intends to classify whether or not a particular flight is late or not, thus an output layer of two neurons is formed.

There is no one correct method of selecting the number of hidden layers and number of hidden neurons. However there are 'rules of thumb' that may be followed (Heaton, 2008):

- There is no theoretical reason to select more than two hidden layers.
- The number of hidden neurons should be between the size of the input layer and the size of the output layer.

For the flight delay MLPC model, two hidden layers of nine neurons each (equal to the number of neurons in the input layer) were selected. The MLPC model was trained using a Spark data pipeline, and then used to make predictions on a test set.

```
Area under ROC: 0.5296895229129935
Accuracy: 0.6655349223946785
Weighted Precision: 0.6706590403863103
Weighted Recall: 0.6655349223946785
F1: 0.6680483691662149
Accuracy: 0.6655349223946785
Precision: 0.25185854932690377
Recall: 0.2627607169059847
F1: 0.2571941523467556
=================== Confusion Matrix ==========================
##########| Predicted = 0                        Predicted = 1
----------+-----------------------------------------------------
Actual = 0| 26308                                7447
Actual = 1| 7034                                 2507
===============================================================
```

**Fig 6:** The MLPC model assessment metrics

With reference to Figure 6, the model MLPC achieved a Weighted Precision of 67.07%.

## Gradient Boosted Trees

Gradient-boosted trees (GBTs) are a popular classification and regression algorithm using ensembles of decision trees (Apache Spark MLlib, 2020). GBT uses a boosting method to combine individual decision trees. Boosting means combining learning algorithms in sequence to achieve a stronger model by using many sequentially connected decision tree algorithms (Yildirim, 2020). Since each tree attempts to minimize the errors from previous trees in the sequence, boosting can improve the accuracy of the model..

Boosting resembles bagging; which is a technique used with Random Forest classification algorithms. However, boosting is fundamentally different (Hastie T, Tibshirani R, Friedman J, 2008).

Furthermore, (Hastie *et al*, 2008) describes the purpose of boosting as to sequentially apply a weak classification algorithm on repeatedly modified versions of the data, thereby producing a sequence of weak classifiers (in an effort to avoid overfitting). The data modifications at each boosting step consist of applying weights (such as $w_1, w_2, ..., w_N$) to each training observation. Initially all of the weights are averaged (i.e. set to $w_i = 1/N$) so that first step simply trains the classifier on the data in the usual manner. For each successive iteration $m = 2, 3, ..., M$ the observation weights are individually modified and the classification algorithm is reapplied to the weighted observations. At step $-m$, those observations $j$ that were misclassified by the classifier induced at the previous step have their weights $w_j$ increased, whereas the weights are decreased for those that were classified correctly. Thus, as iterations proceed, observations that are difficult to classify correctly receive ever-increasing influence. Each successive classifier is thereby forced to

concentrate on those training observations that are missed by previous ones in the sequence. Thus correctly sizing trees for boosting is important and the value of iterations can be fine tuned by trying several different values and choosing the one that produces the lowest risk of overfitting (Hastie *et al*, 2008, p. 337).

The weather features included in Phase 2 are continuous variables, but range over differing scales. Consequently, the different variables are standardized, and passed through Principal Component Analysis to reduce the number of weather variables to 14 linear combinations (the 14 Principal Components describing the most variation in the dataset) prior to including the categorical features *see* Figure 7.

```
import org.apache.spark.ml.feature.StandardScaler

// creating standardarized features
val std_scalar = new StandardScaler()
                 .setInputCol("weather_features")
                 .setOutputCol("standard_features")
                 .setWithStd(true)
                 .setWithMean(false)

// implementing to PCA
val pca = new PCA()
          .setInputCol("standard_features")
          .setOutputCol("continuos_features")
          .setK(14)
```

**Figure 7:** Taking 14 principal components of all the standardised weather features as explanatory variables.

```
// creating gradient boosted tree classifier
val gbt = new GBTClassifier()
          .setLabelCol("label")
          .setFeaturesCol("features")
          .setMaxIter(10)
          .setMaxBins(10)
          .setFeatureSubsetStrategy("auto")
```

**Figure 8:** A code snippet of parameters used in the GBT estimator as implemented in the GBT model

The maximum iterations (Fig. 8) is chosen to be 10, as it produces better performance when evaluated against confusion matrix performance measurements. The number of bins (setMaxBins(10)), this is set to at least 2, or at least the number of categories for any categorical feature. Since the categorical variables “Airline” has a distinct value of 10, the number of bins is set to 10. The attribute MaxDepths is set at the default value of 5. The GBT trained model yields the following performance:

```
Area under ROC: 0.5966888615450411
Accuracy: 0.5003797992818341
Weighted Precision: 0.7118122607647752
Weighted Recall: 0.5003797992818342
F1: 0.5369334524822094
Accuracy: 0.5003797992818341
Precision: 0.26242455122677744
Recall: 0.6987216279676494
F1: 0.3815480176085934
=================== Confusion Matrix ==========================
##########| Predicted = 0                        Predicted = 1
----------+-----------------------------------------------------
Actual = 0| 30086                                37637
Actual = 1| 5774                                 13391
===============================================================
```

**Figure 9:** The Gradient Boosted Tree (GBT) model assessment metrics with a weighted Precision of 71.18%.

# Summary

By the end of the Phase 3 investigation, the original machine learning pipeline had been adapted by the Group 8 members to better suit the purpose of each SMLA. Also, due to hardware limitations, not all models were subjected to cross validation. Consequently, choosing a best estimator for this project is made more challenging. An objective assessment of estimator model performance is shown in Figure 10.


**Figure 10:** A comparison of Logistic Regression (lr), Random Forests (rf), MultiLayer Perceptron (mlp) and Gradient Boosted Trees (gbt) based on the performance measurements Accuracy and Weighted Precision.

In summary:

- Logistic regression was both moderately accurate and precise but also highly interpretable (if inference was the end goal of the analysis).
- The multi-layered perceptron provided a nice balance of accuracy and precision, and was quick to train, but difficult to interpret.
- The two tree based models had the highest weighted precision, but were very memory intensive, and sensitive to their hyper-parameters.
- The prediction of the No Free Lunch Theorem rings true!

# References

Apache Spark MLlib (2020). Retrieved from: https://spark.apache.org/docs/latest/ml-classification-regression.html#gradient-boosted-tree-classifier

Australian Government, (2020, April 23). Domestic Airlines - On Time Performance. Retrieved from: https://data.gov.au/data/dataset/domestic-airline-on-time-performance

Databricks (2020). Neural network. Retrieved from: https://databricks.com/glossary/neural-network

Hastie T, Tibshirani R, Friedman J, The Elements of Statistical Learning: Data Mining, Inference, and Prediction (Second Edition, 2008), chapter 10, pp. 337

Heaton, H (2008). Introduction to Neural Networks for Java.Retrieved from: https://web.archive.org/web/20140721050413/http://www.heatonresearch.com/node/707

IndexMundi (2020, April). Price Index for Jet Fuel (AUD). Retrieved from: https://www.indexmundi.com/commodities/?commodity=jet-fuel&months=240&currency=aud

Wolpert,D. H. (1996) The lack of a prior distinctions between learning algorithms and the existence of a priori distinctions between learning algorithms, Neural Computation 8 , 1341–1390,1391–1421.

Yildirim, S. (2020) Gradient Boosted Decision Trees- Explained. Retrieved from: https://towardsdatascience.com/gradient-boosted-decision-trees-explained-9259bd8205af

Yiu , T., (2019) decision tree image - Retrieved From: https://towardsdatascience.com/understanding-random-forest-58381e0602d2