How can we use PySpark to maintain customers?

Contents

1. Introduction

1.1 Spark

A large part of a data scientist's role is to be able to predict future events. This can be done predominantly by using machine learning (ML) techniques. With the advancement of cloud computing, companies can create and store more data, which in turn can be directly used to train these predictive models. Clearly, the more information and features we have, the better the model will be. However, this does come at a cost because as the amount of input data scales, so does the computing power required to run the model. Therefore, standard programming languages like Python is not adequate to handle this.

Here's where Apache Spark comes in. By utilising cluster computing, Spark can handle vast amount of data as the workload is distributed over several computers, avoiding the bottleneck issues experienced by traditional processing software. PySpark is the Python API for Spark, which allows access to this large-scale data processing, facilitated by cluster computing, while using familiar syntax.

Traditionally, programmers in Python would use a library such as scikit-learn to develop a ML model however this is not possible in PySpark. Instead, we use the Spark Machine Learning library which is scalable and provides a uniform set of high-level APIs that help users create and tune practical machine learning pipelines.

1.2 Application

With the processing power of PySpark and the ability to utilise this for building ML models, this is a very powerful tool to allow businesses to better understand the behaviour of its customers. For a subscription-based service, a common issue faced is trying to predict what segment of its customer base is likely to churn (cancel their subscription). If this can be predicted before churning, it would allow for targeted advertisement to entice those customers to stay on and prevent the loss of revenue.

1.3 Sparkify

Sparkify is a fictional company but has similar characteristics to 'Spotify' – a music streaming service which users can either pay a subscription or use the free tier. Every time a user

interacts with the platform, whether this is logging on, skipping song etc, this creates a data point which is stored. Clearly, as a single user of the platform can produce several amounts of data point per day, PySpark is a very useful tool to process this.

Of the metrics monitored, the one relevant to this study is whether that user churned or not. By making this variable our target, a ML model can be built to understand which behaviours are most important in influencing if a customer will churn.

For this purpose of this study, we will only be using a small segment of Sparkify data which can be processed using the local mode of PySpark. This allows analysis to be conducted on a Jupyter Notebook as cluster computing is no longer needed. However, the methodology and syntax are identical and can easily be scaled to process the whole dataset.

2. Sparkify Data

2.1 EDA

As mentioned in **Section 1.3**, each interaction between the user and the platform is recorded as a separate row in the dataset, so firstly we will explore which features are available to us. There's 18 columns present in the subset that we're analysing, including the following –

- 'gender' = Sex of the user.
- 'level' = What subscription level the user is using, free or paid.
- 'song' = Name of the song the user's listening to.
- 'location' = Geographical location of where the user interacted with the platform.
- 'page' = What page of the platform the user's using.
- 'userId' = Unique identifier of the user.

Using the 'userId' column and counting the number of unique variables, we can see that this dataset contains information about 226 users. In total, there's 286500 interactions monitored, giving an average of 1268 interaction per user. We can therefore easily see that if Sparkify had 100,000 users for example, why cluster computing would be required to process this large amount of data.

2.2 Churn

The feature which indicates if a user has churned is ‘page’ as this contains the variable "Cancellation Confirmation". Therefore, by using this as the marker, we can see that of the 226 users present in the dataset, 52 of them churned.

For this to be used as a label in a ML model, it must be converted into a numerical value. Therefore, we will create a new column called ‘Churn’ which is binary, containing the value 1 if ‘page’ = ‘Cancellation Confirmation’ and 0 otherwise.

2.3 Trends

Using this new column, we can separate our dataset into two; one containing users who’ve churned and the other where users didn’t. By doing this, we can explore which features vary significantly between the two groups.

Firstly, a feature of particular interest is ‘level’ as this tells us if Sparkify makes a revenue from the user or not. Instinctively, one could speculate that if a customer regularly uses the platform, then they’d be willing to use the paid level. However, if the user is looking to cut personal costs, then a paid level customer is more likely to churn.

The dataset contains 3.7 times more free level users compared to paid level users. Looking at this deeper, we see that of the 178 customers in the free level, 44 of them churned giving a churn rate of about 25%. Comparing this to the 48 customers in the paid level, 8 of these churned giving a churn rate of about 17%. Thus, although this doesn’t directly explain to use why a free level user is more likely to churn, it does tell us that it’s an important feature and should be included in the model.

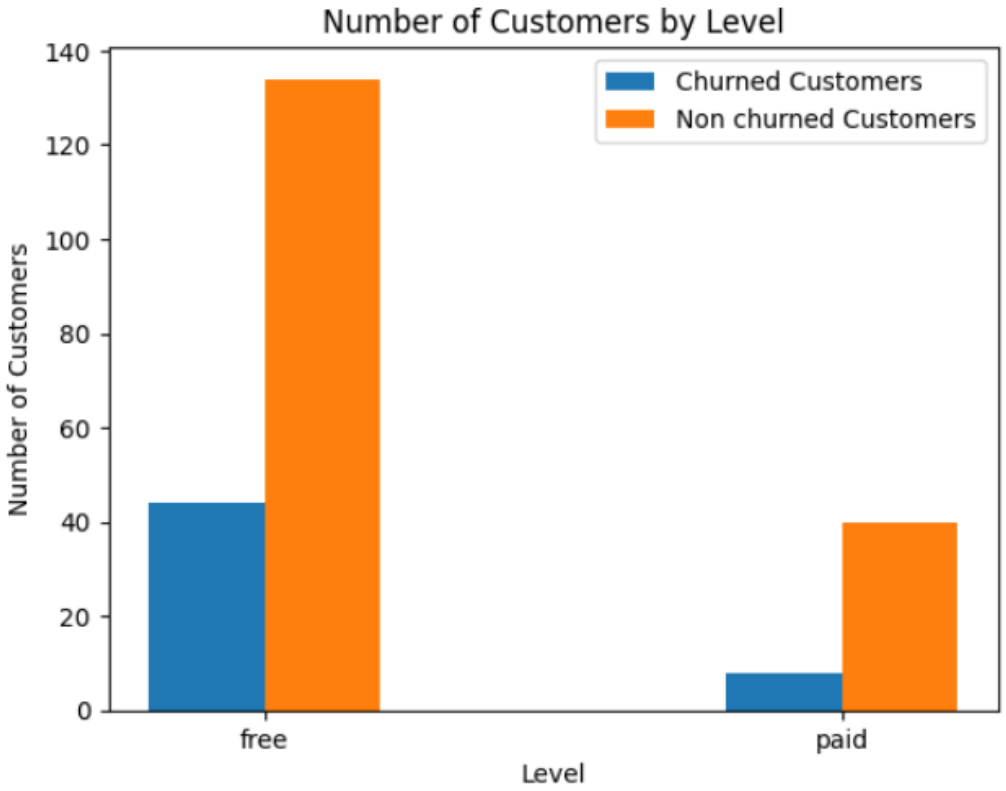

Figure 1 - Amount of customer who churned broken down by level.

Another interesting feature in the dataset is 'gender' as this provides some personal information about the user. Of the 121 male customers in the dataset, 32 of them churned (26% churn rate). 20 out of the 104 female customers churned, giving a churn rate of 19% (one user had a null value). Given the fairly equal split between the two genders, a difference of 7% can be considered significant.

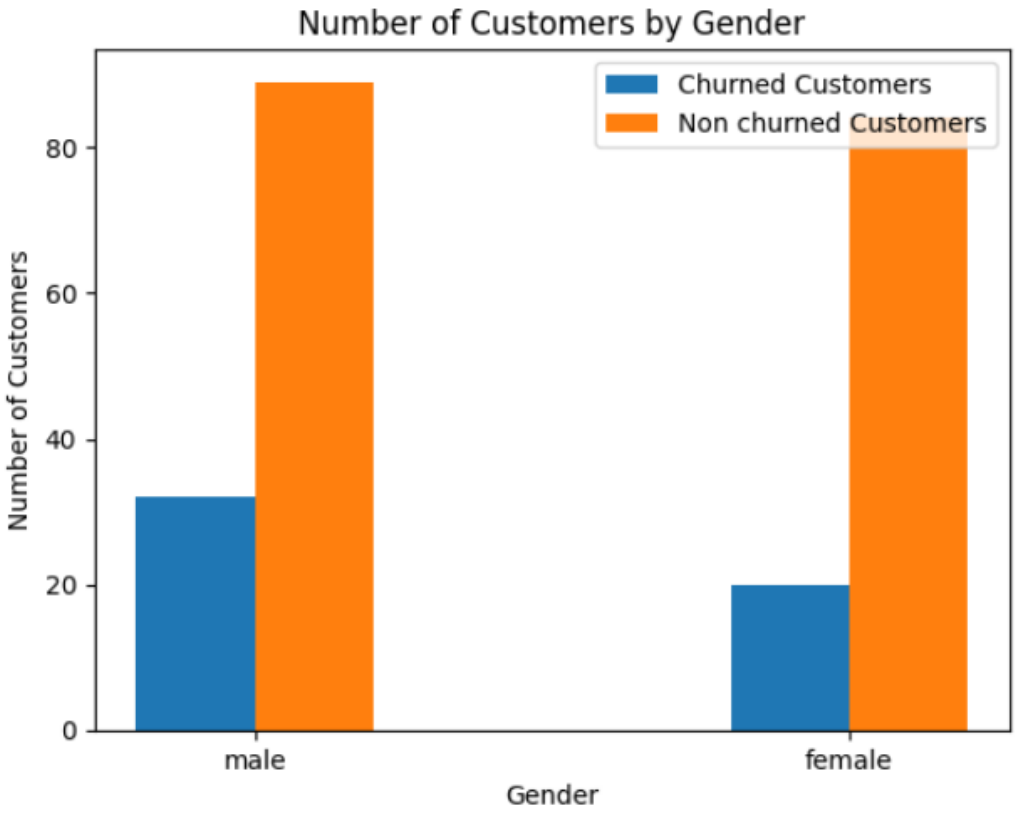

Figure 2 - Amount of customer who churned broken down by gender.

A feature which gives us an insight into the habit of a user is 'song'. If we look at the average amount of unique songs listened to by both sets, we see that users who didn't churn listen to around 1.6 times more songs than those who did churn (1388.7 / 862.8). This result makes sense as we see a higher level of engagement from users who are satisfied with the service.

Another personal feature about the user is location, which may be linked to churning rate. By ranking the locations by the amount of times an interaction took place there, we can see where is the most popular place and if this differs between users who churned or not. Interestingly, Los Angeles was the top location for both datasets however the rest differed highlighting that this may be a feature of interest.

3. Feature Engineering

3.1 Numeric Variables

The feature that we're interested in predicting is the binary variable, churn, therefore we will re-name this column to be 'label'.

To use a ML model, this requires for all the input data to be in a vector format, thus numeric. We've already seen that both the 'gender' and the 'level' column is a key difference between

customers who churn or not. These are simple to convert to numeric, as we assign 1 to female and 0 to males and then 1 to those in the paid level and 0 to those in the free level.

The other two identified features of interest are 'location' and 'song'. Clearly, these can't simply be converted to binary values as it requires text processing. Therefore, we begin by using the Tokenizer function to split the text into a list of words. This is then passed through the Count Vectorizer and IDF function, which determines the importance of each word.

At this stage, we now have the 4 features of interest in a numeric representation and can finally be passed through the Vector Assembler function followed by the Normalizer function.

4. Modelling

4.1 Splitting Data

As we now have our desired features in a vectorized form, the data is ready to be used in a model. For this, we need to split the data into a training set to build the model on, a testing set to test the accuracy and a validation set to confirm the results. A 70, 20, 10 split was decided between the training, testing and validating set respectively.

4.2 Logistic Regression

The first model that was used with the training data was a Logistic Regression (LR). Here, we have a single binary dependent variable (which is 'label') plotted against the independent variables which can be either in a binary or a continuous format. This maps each value a probability between 0 and 1. If we then apply a threshold value of 0.5, we can then classify each row as being more likely to churn or not.

To assess the performance of this LR model on the testing dataset, we used the F1-score. This value is calculated from the recall and precision of the test i.e $F1 = 2TP / (2TP + FP + FN)$ where TP = true positive, FP = false positive and FN = false negative.

A F1-score of 0.94 was achieved using the LR model, which is a very good value. However, it must be noted that the relative size of this dataset is small and we would expect this value to differ on the full dataset. This should be taken into consideration with all the outputs from the models.

4.3 Random Forest

The second model that we'll explore is the Random Forest (RF) classifier. This model uses an ensemble technique which uses multiple decision trees to make a decision. A decision tree is a classification model, where each leaf represents a class label, and each branch represents a decision rule. As this is an ensemble technique, the output will be the results of all the decision trees. The final prediction is decided by majority voting.

We see a dip in performance of the RF model compared to the LR, giving a F1-score of 0.77 using the same testing dataset.

4.4 Decision Tree

The final model that we're using is a Decision Tree (DT). Clearly, this is a simpler version of the RF model, as it only requires a single classification model. Due to the poorer performance of the ensemble equivalent, perhaps a simpler model would give a higher F1-score.

Using the testing dataset, we get an F1-score of 0.88 which is a good score however I believe that this value can be optimised. One such way this model can be tuned is by changing the maxDepth hyperparameter. In a DT model, the data is split into subsets based on a decision rule until no further value is added to the prediction. Therefore, the amount of times the data is split dictates the depth of the tree. By altering this hyperparameter, we dictate to the model it's complexity and could avoid the issue of over-fitting the data to the training set for example.

By fitting the training dataset to a DT model with a maxDepth of 10, 20 and 30, we get an improvement of F1-score as the size of the tree increases. With a maxDepth of 30, we achieve an F1-score of 0.97 which is now our best performing model.

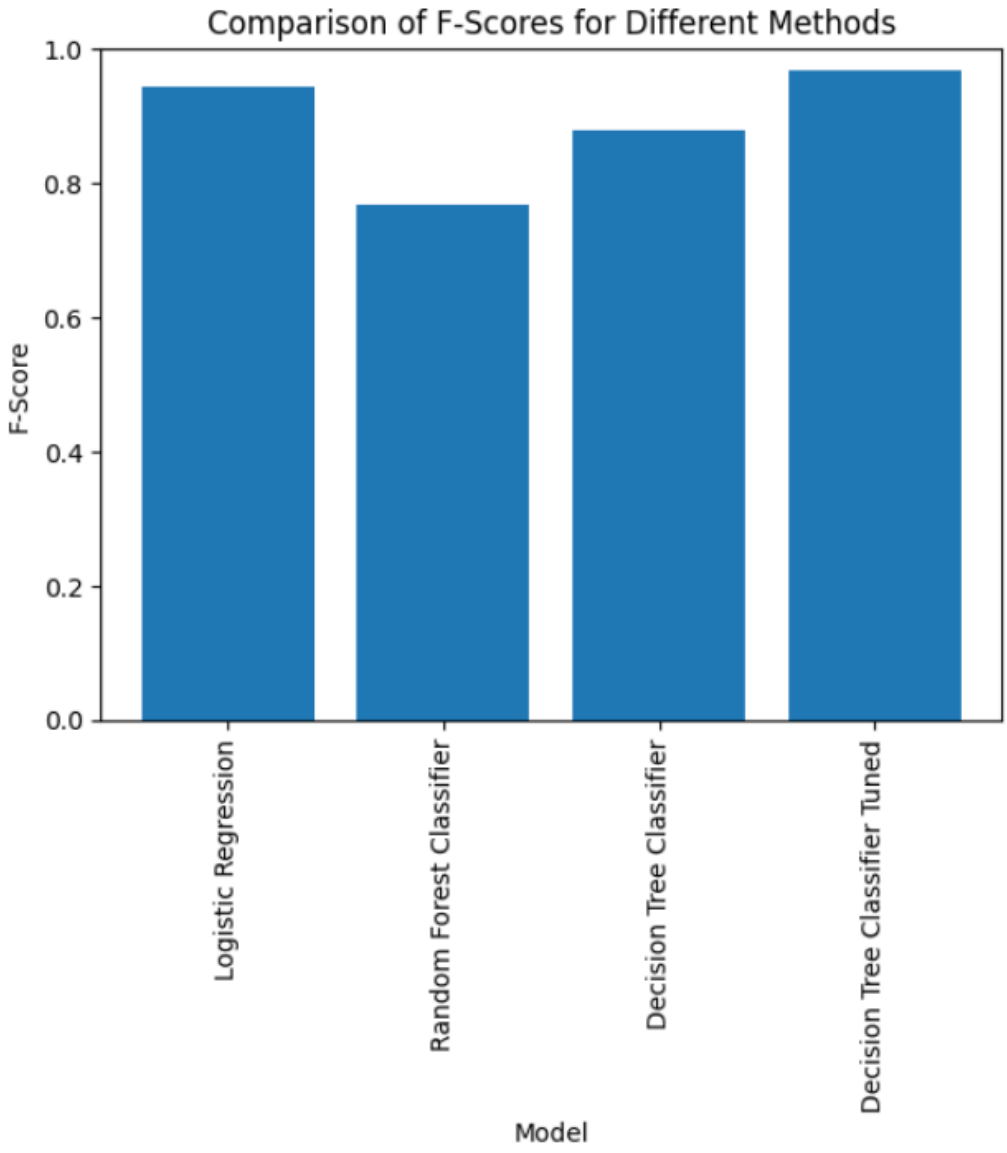

Figure 3 - F1-scores of different models.

4.4 Validating

As this is an unbalanced dataset (only 52 out of the 226 users have churned), F1-score is a good metric to rank the model against each other. Another metric that can be used is accuracy, which takes the number of correct classification and divides it be the total amount of classifications. For the tuned DT classifier, we get an accuracy value of 0.97, which again is a very good score.

Finally, we should test how well the model performs on an unseen dataset, confirming if there's been overfitting on the training dataset or not. By predicting the churned users on the validation dataset, we get an F1-score of 0.97, confirming the performance of the DT model.

5. Conclusions

Using the predictive performance of the Spark Machine Learning library, we can see why PySpark is very useful for businesses when the want to predict which users are prone to leaving their platform. With this knowledge in hand, a company such as Sparkify can target this specific cluster of users with advertisement to try and prevent them from leaving.

The Spark session used to build the model discussed here was done so using the local mode of PySpark, meaning the amount of data read was limited. However, this can easily be upgraded to using a cluster be utilising a cloud platform such as AWS. This would allow for all of Sparkify's user data to be used to train the model, which would give a much more representative F1-score.