|  |  |
| --- | --- |
|  | |
| Programming for Big Data –Assignment 2  SPARK | |
|  | |
| Ciaran Finnegan – TU060 (Part Time) – Second Year  PROG9813: 2022/2023  MSc in Computer Science (Data Science)  Student No : D21124026  23/4/2023 |  |
|  |  |

Table of Contents

1 Problem Definition – Kaggle Competition Objective 3

2 Data Pre-Processing 4

2.1 Data Exploration 4

2.2 Data Preparation 6

2.3 Feature Engineering 7

3 Model Creation 8

3.1 Split Kaggle Training Data 8

3.2 Choice of Algorithms 8

3.3 Set Up PySpark Pipeline 9

4 Module Evaluation 10

4.1 Key Performance Metrics - GBT 10

4.2 Refining the Model 13

4.3 Chosen Model : Why? Explanation of Decision 14

5 Spark : Key Features Used in Assignment 15

5.1 Advantages of Spark – Kaggle Fraud Competition 15

6 Kaggle Competition : My Entries 16

6.1 Submission Results 16

6.2 Kaggle Submission Summary 16

# Problem Definition – Kaggle Competition Objective

This report collates all the delivery requirements for Assignment 2 in the Programming for Big Data module (PROG9813: 2022-23).

The primary content of the report covers;

* The rationale behind the selection of the IEEE-CIS Fraud Detection competition on Kaggle.
* The steps taken to analyse and prepare the data for modelling.
* The build and evaluation of the models, which then generated the Kaggle submission scores.

My professional background is in the area of financial crime detection, with a growing emphasis on Machine Learning techniques for fraud detection.

The IEEE-CIS Fraud Detection competition ([IEEE-CIS Fraud Detection | Kaggle](https://www.kaggle.com/competitions/ieee-fraud-detection/overview)) was an interesting challenge that presented competitors with approximately 600K+ dataset of historical credit/debit card transactions, of which a small percentage were retrospectively marked as fraudulent.

In common with most competitions of this type the user is provided with an unlabelled test set and must generate predictions based on the ML models they create. Competition submissions are evaluation based on the metric of area under the ROC curve. An ROC (Receiver Operating Characteristic) curve is presumably used because of the focus on recall, or the performance in detecting the minority class. In Financial Services the emphasis is generally on detection models that are more effective at finding fraud (True Positives), possibly even at the expense of overall accuracy or an (annoyingly) higher level of False Positives. The cost of actual Fraud is always greater than the overhead of investigating transactions that turn out to be benign.

*Note:- For simplicity the training of the model for this Kaggle competition was based only on the ‘core’ transaction data, and did not include the supplementary ‘identify’ dataset. The Kaggle submission scores achieved were deemed acceptable for this assignment.*

# Data Pre-Processing

## Data Exploration

The Kaggle competition provides the labelled ‘Training’ transaction data to generate the models, which are used to core the separate unlabelled ‘Test’ data.

This competition problem is about generating a binary classification score/flag for each ‘Test’ transaction to indicate the presence of fraud (or not).

Both datasets have been loaded into a Databricks CE table store in advance.

### Load ‘Training’ Data

The first step of data exploration is obviously to load the data into the PySpark notebook. A quick size of the data follows.

Text

Description automatically generated

<Image – Code Load data>

Graphical user interface, text, application, Word, website

Description automatically generated

<Image – Code to show size>

The PySpark notebook attached to this submission displays the top rows of the loaded dataset, but that is not reproduced here.

### Check Balance of Fraud v Non-Fraud

Datasets for transaction fraud are notoriously unbalanced. Fraud is a serious problem but still only presents a tiny fraction of the volume of electronic commerce that takes place each day. Very large majority classes, non-Fraud records in this case, can create a bias in many models and impact performance in detecting the minority class (fraud).

How is the balance of ‘non-Fraud’ versus ‘Fraud’ in the Kaggle IEEE data? As expected, non-fraudulent records vastly outnumber labelled fraud transactions. (Bar chart generation code is only partially shown).



A picture containing chart

Description automatically generated

<Image – Code + Bar Chart>

Section 2.2.1 of this report will show how we address this problem.

### Missing Data

The Kaggle data has a small number of categorical (text) features. Many of these contain null values.

Most of the numerical data columns have been generated post transaction by an IEEE partner organisation (Vesta) but are no consistently populated, with many null values.

Not all the code is reproduced here but the images below indicate the scale of null values in the Kaggle Training dataset.

Graphical user interface, text, application, email

Description automatically generated

<Image – Code to calculate Null values?

Many Machine Learning algorithms do not perform well with null data, so these will be addressed – as described in Section nn of this document.

## Data Preparation

### Re-Balance Dataset

All records labelled as fraudulent are kept and the non-fraudulent set is down sampled to the same size to create and approximate 50/50 balance.

Graphical user interface, text, application

Description automatically generated

Chart

Description automatically generated

<Image – Code and Balanced Bar Chart>

Although this process reduces the overall size of the dataset for modelling, it should greatly eliminate model bias towards non-fraudulent detection and improve the rate of True Positive detection on the Kaggle competition ‘Test’ data.

### Handle Missing Data

There is not enough context in either the Kaggle website, or the dataset itself, to allow for any meaningful impute of missing data and the generation of synthetic values into the *nulls*. For simplicity, the null categorical data is replaced with a blank text entry, and the null numeric cells are replaced with a value of *zero*.

Graphical user interface, text, application, email

Description automatically generated

<image – Code>

Verification code to confirm the successful updates is included in the notebook with this submission, but not replicated here.

## Feature Engineering

### Categorical Data

Many Machine Learning algorithms expect all dataset inputs in numerical form. The Kaggle competition set contains fourteen categorical values, such as a text indicator for ‘debit’ or ‘credit’ card transactions, and series of columns with ‘True/False’ to track matches on names, addresses, etc.



<Image – Code with list of Categorical features>

PySpark provided a function to implement ‘One-Hot Encoding’ on categorical data. This converts the categorical data to multiple numerical features, based on the data attributes.

In the simplest example in this data set the ‘***M1***’ feature with two values ‘*T*’ and ‘*F*’ is essentially converted to two new features (consider them ***M1\_T*** and ***M1\_F***) with binary 0 and 1 values to replace ‘*T*’ and ‘*F*’.

# Model Creation

## Split Kaggle Training Data

In line with the standard Machine Learning workflow process for supervised (‘labelled’) data, the Kaggle Training data is split into a further subset of training and test data to allow us build and evaluate our fraud classification model.

Text

Description automatically generated

<Image – Split Test/Train Data>

## Choice of Algorithms

As this Kaggle competition is a relatively understandable binary classification problem the first choice of algorithm with which to build a fraud detection model is Logistic Regression (LR).

Random Forest (RF) is the next algorithm choice because it is an ensemble learning method that combines the output of multiple decision trees, which helps avoids overfitting and improves generalisation in areas such as fraud detection.

The Gradient Boosted Trees (GBT) is another ensemble learning method provided by PySpark. Many of the most successful Kaggle submission for this competition employed gradient boosted algorithms and GBT appears to be the closest equivalent available (***GBTClassifier***) in the PySpark ML library.

To avoid duplicating PySpark notebooks, the user is allowed to select the algorithm of choice. This allows multiple ‘clones’ of the same PySpark notebook be executed in parallel. *User input code is not replicated in this report but is available for review in the attached PySpark notebook.*

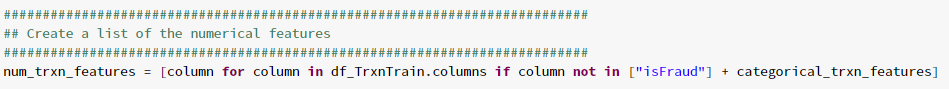
Graphical user interface, text, application, email

Description automatically generated

<Image – Code with LR set up>

## Set Up PySpark Pipeline

PySpark allows use to sequence a series of tasks in the preparation of our fraud detection model.







<Image – Code of Vector Assembly>

The dataset has been balanced and missing data addressed in advance of the above steps, but the encoding and indexing of categorical features is sequenced into the ‘Pipeline’ that creates the model.

The pipeline then builds the model with the selected algorithm and generates a set of test predictions.

Graphical user interface, text, application, email

Description automatically generated

<Image – Code of Pipeline>

# Module Evaluation

## Key Performance Metrics - GBT

The performance of the generated model is assessed against the hold-over subset of test data from the Kaggle Training dataset (not to be confused with the Kaggle Test set of unlabelled data against which is built the final competition predictions).

For simplicity, the code and diagrams below only show the output from the evaluation of the model generated by the GBT algorithm, which proved to be the most performant. A comparison of all model metrics is produced in Section 4.3 of this report.

### ROC Graph

The PySpark notebook presents an ROC curve graphic to present a visual representation of performance. The dotted line represents ‘random guessing’ and the greater the area under the curve, the better the performance of the model.

A function was written to display a more graphically distinctive view of the ROC graph.

Graphical user interface, text

Description automatically generated

Chart

Description automatically generated

<Image – ROC graph>

### Area Under Curve and Recall Metrics

Two PySpark evaluator parameters are used for the *BinaryClassificationEvaluator* function; *areaUnderROC* and *areaUnderPR*.

Text

Description automatically generated

<Image – Code for *BinaryClassificationEvaluator* function>

Although the Kaggle competition uses an area under ROC metric, the AUPRC (Area Under Precision-Recall Curve) metric is also useful in this context. The AUPRC emphasises the performance of the model on the minority (fraud) class, which is a key objective.

Graphical user interface, text, application

Description automatically generated

<Image – Model Scores>

### Recall

To compliment the above metric a function has been implemented to calculate the Recall performance of the model. ( Recall = True Positives / (True Positives + False Negatives) )

Graphical user interface, text, application

Description automatically generated

<Image – Recall Function and Output>

Additional SQL was implemented to present a form of pseudo–Confusion Matrix to highlight the number of fraud records identified and missed.

Table

Description automatically generated

<Image – SQL Table>

## Refining the Model

### Cross Validation and HyperParameter Tuning

Running multiple cycles of training/test data model generation with different selections of data in each subset, will generally improve performance. This process contributes towards an avoidance of overfitting on a particular set of training data or generating an unwanted bias. The following code was written to implement these ‘Cross Validation’ cycles, with a given selection of options.

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

<Image – Code of HyperParameter + CV>

### The Failure of CV and Tuning

In the Databricks CE environment used for this assignment Clusters would routinely time out when running CV and tuning on the full Kaggle dataset (after train/test split).

Through trial and error, it was determined that reducing the training data size to between 8.5% - 10% of previous size would allow Databricks CV processing to complete (after approximately 90 minutes).

Although CV and tuning processes generally improve model performance in ML workflows, the trade off in sample size reduction for this assignment meant that performance metrics were worse than those for the initial ‘basic’ model for GBT and LR.

Aside from the Random Forest built model, Databricks CE could not process enough data before CV/tuning time out to deliver a better fraud detection model.

## Chosen Model : Why? Explanation of Decision

A table of metrics for the different models generated for this Kaggle fraud detection competition is as follows;

<table>

The ‘basic’ GBT model (no CV/tuning applied) performed best across the four metrics, in comparison to the other models. These metrics correlate to the score generated by the Kaggle submission evaluation process itself, as described in Section 6 of this report.

# Spark : Key Features Used in Assignment

This project has been written using PySpark in a Databricks CE environment.

## Advantages of Spark – Kaggle Fraud Competition

The advantages of using such a set up for this Kaggle Machine Learning fraud detection problem can be summarised as follows;

* The Kaggle dataset has a large Training set of nearly 600K+ records. Although this is cut down to approximately 40K records after balancing the binary classes, this is still a significant volume of data. Databricks CE provides an environment for distributed Machine Learning processing and faster model building and evaluation on larger datasets. Within this Databricks environment, PySpark (the Python library for Apache Spark) is designed to handle high volume datasets. Despite the fact that Databricks CE only allows a single cluster at a time, the processing can utilise access to 15+GB and two cores to speed up the ML process for this Kaggle competition.
* PySpark has data manipulation capabilities that are optimised for preparing large datasets for the Machine Learning workflow. In this Kaggle competition there are a number of categorical features that require indexing and encoding prepossessing. PySpark’s DataFrame API provides an optimised set of operations to prepare the categorical features in the Kaggle transaction dataset and set up a sequence of jobs to generate a data vector to feed into the model generation process.
* PySpark included the MLib library that, along with the data transformation functions described above, provide distributed algorithms designed to work on large datasets. The PySpark Logistic Regression algorithm provides the initial Kaggle predictions submitted to this fraud detection competition.
* Databricks CE itself provides a web-based interactive notebook interface to build the Kaggle competition environment and allows the PySpark code to side easily beside Markdown text and visualisations.

If this Kaggle fraud detection competition provided a Training dataset of <10K records, then ‘regular’ Python libraries such as Scikit-Learn or TensorFlow might be considered more appropriate for this binary classification problem.

# Kaggle Competition : My Entries

The key assessment of model performance is provided by the Kaggle submission process itself. A submission is generated on the competition Test data in the PySpark notebook, and an output ***csv*** file is extracted. This ***csv*** fie is then scored once it is submitted to the Kaggle portal.

## Submission Results

The screenshot below shows the results of multiple submissions made to this Kaggle fraud detection competition, using production output generated by the assignment PySpark notebook.

Graphical user interface, text, email

Description automatically generated

## Kaggle Submission Summary

* As expected, the predictions generated on the Kaggle competition Test Data using the ‘basic’ GBT model generated the highest Private/Public score.
* Cross Validation and tuning on the GBT and LR models was ineffective in improving performance because of the need to drastically reduce sample size.
* Logistic Regression was the least performant model, but with Random Forest being only marginally better.
* The best score achieved by this assignment is significantly below the Kaggle competition winner, but still reasonably respectable.