|  |  |
| --- | --- |
|  | |
| 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 5

2.3 Feature Engineering 6

3 Model Creation 7

3.1 Split Kaggle Training Data 7

3.2 Choice of Algorithms 7

3.3 Set Up PySpark Pipeline 7

4 Module Evaluation 8

4.1 Key Performance Metrics 8

4.2 Refining the Model 9

4.3 Chosen Model : Why? Explanation of Decision 9

5 Spark : Key Features Used in Assignment 10

5.1 Spark 1 10

5.2 Spark 2 10

6 Kaggle Competition : My Entries 11

6.1 Logistic Regression 11

# 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 a 500K+ 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.

# 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.

<Image – Code Load data>

<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).

<Image – Code Bar Chart>

Section nn 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.

<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.

<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*.

<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.

<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.

<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.

<Image – Code of Pipeline>

# Module Evaluation

## Key Performance Metrics

The performance of 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).

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.

<Image – ROC graph>

### Area Under Curve and Recall Metrics

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

<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.

<Image – Score of model>

### 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) )

<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.

<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.

<Image – Code of HyperParameter + CV>

## Chosen Model : Why? Explanation of Decision

# Spark : Key Features Used in Assignment

## Spark 1

The..

## Spark 2

The…

# Kaggle Competition : My Entries

The primary assessment of 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.

## Logistic Regression

The first LR submission was built on a small sample of the competition training data (~100+ balanced record). The second uses the full Training data to build another LR model. In both cases the full Kaggle Test data was ingested, and fraud scores/predictions generated.

<Image – LR Kaggle submissions>