A comparative study of big data solutions for Fraud detection

1st Michael Berlian
School of Computer Science
University of Nottingham
Nottingham, England
psymb17@nottingham.ac.uk

2nd Arnav Sankhe
Dept. of physics and astronomy
University of Nottingham
Nottingham, England
ppxas5@nottingham.ac.uk

3rd Jakub Marcoll
University of Nottingham

4th Georgios Theocharidis
University of Nottingham

***Abstract*—Hundreds of transactions happened every second. Some of them are fraud. Each year a huge loss is incurred by fraudulent transactions. Fraud contributes to millions of losses every year, therefore its an important job to prevent it from happening. A machine learning approach to predict if a transaction is a fraud or not is one of the solutions. However, as it is known that millions of transactions happen every single day, machine learning training needs to be optimized to increase efficiency. One of the approaches to optimized this is using the big data approach. In this study we go over the details of big data approaches to prevent fraud transactions. Moreover, a big data approach to pre-processing the data is also implemented. The implementation in this study will be utilizing spark. Two machine learning algorithms, Random Forest and Multi-Layer Perceptron, are employed for the solution. In this study, we will compare the approaches of model building and data pre-processing. For this study, we use kaggle competition dataset containing details of both fraud and not fraud transactions. The result of our study shows a successful implementation of fraud identification and comparisons of these approaches.**

***Index Terms*—Big Data, Fraud detection, Random Forest, Multi-Layer Perceptron, comparative study.**

I. Introduction

In 2020, a total of £783.8 million were lost due to fraud transactions. Furthermore, almost 150.000 incidents of scam with a total loss of £479 million happened in 2020 [1]. Fraud detection is one of the most important topics in the banking, finance, and insurance industry. Sometimes the systems can make errors that could be embarrassing for customers, such as when their credit card is declined, to tackle this problem there are a variety of machine learning algorithms that can be used to classify transactions as fraudulent or not, of which we are going to look into two algorithms Random forest and Multi-Layer perceptron.

In this comparative study of big data solutions for fraud detection, our aim is to have a detailed comparison between Global and Local approaches of machine learning. This study also aims to compare the outputs of the models based on the effects of clusterings on the selected features in the dataset. The data that was used to address this problem was the Kaggle competition dataset, the link to which is: https://www.kaggle.com/c/ieee-fraud-detection. The data pre-processing was a very challenging task as the dataset had more than 300 features which we were able to narrow down to 23 features using pre-processing details to which will be discussed in the further section. In addition to this, an under-sampling method is necessary to balance the imbalance dataset consisting of 569,887 majority class and 20,663 minority class. In this comparative study we looked at two training methods, In the global training method, we built a pipeline to fit the models and a grid search was used for hyperparameters tuning. Whereas in the local method we have used RDDs to parallelise the work and are using an ensemble method to predict the outputs for both the models.

This paper serves as a critical review of machine learning algorithms for big data solutions, where we compare the model outputs for Random forest and Multi-Layer Perceptron for both global and local approaches. In the methodology section, we take a detailed look into data preprocessing, model approaches as well as modelling algorithms like Random Forest and Mult-Layer perceptron and their hyperparameter tuning. The evaluation section consists of model details and evaluation metrics that we used to evaluate the model. Models are compared in the results and discussion section, here we talk about the obtained result and model performances. Finally in conclusion we conclude our findings.

II. Methodology

In this section, we'll have a detailed look at the specific methods chosen and used in this study.

A. Data Pre-processing

Dataset pre-processing is a very crucial step if you want good results from the model [2]. But for our case, the models trained on the clustered dataset performed worst that the models trained on the non clustered dataset built on just the feature selection to have the same features, the details of which we will be discussing in the further section.

1) Data Balancing : The first problem that we addressed is the unbalanced dataset problem with the majority class having 569,877 samples and the minority class with just 20,663 samples. This means that the two classes are not equal and thus the model will be biased to select the most populous class since that would translate into a higher score. To tackle this problem we use under-sampling to balance the dataset [3], as we had plenty of data for the majority class hence we decided to go with under sampling. For undersampling we used sampleBy() function provided by pyspark library, where we use 0.036 percent of the majority class and 1.0 of the minority class which led us to have 18950 and 20663 samples for the given classes respectively.

2) Data Cleaning: Even after dataset balancing, there are still a lot of problems in the dataset that needs to be addressed. As the dataset had a lot of categorical data, changing it to numerical took priority before the data clustering and feature selection process. To convert categorical data to numerical data we used StringIndexer() provided by pyspark library [4]. The dataset also had many features with lots of missing values, we could have dropped all the features with missing values but almost all features in the dataset had missing values, Hence we decided to fill the missing values with the mode of that feature. To do this we use imputer() function provided by the pyspark library. The reason why we had to convert categorical data to numerical first was because imputer works only with numerical type data [5].

3) Data Clustering: As there were lots of features we thought we could cluster some of the known features together to reduce the complexity and simplify the features and make the dataset easier to visualize. We combined all the M's, C's and D's in one list respectively. Then we use Bucketizer provided by pyspark ml library, it basically maps a column of continuous features to a column of feature buckets [6] and the intervals were decided by looking at the data.

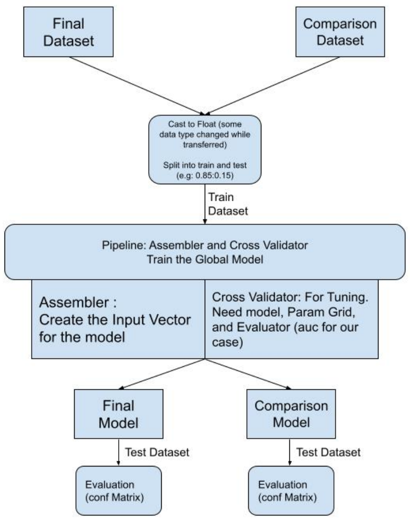
4) Feature selection: After data pre-processing, we use ChiSqSelector to select the top 5 features in the pre-processed dataset. Chi-Squared feature selection, which chooses categorical features to use for categorical label prediction [7]. But chiSqSelector needs a vectorized input of all features, so we used a vector assembler to transform feature data into a vector output which is used as an input to ChiSqSelector [8].

B. Model Approach

Two big data model solutions were proposed: global and local. The goal of the global is to create a single model using the whole dataset. The local aims to divide the dataset into a few partitions to create multiple models. Local and global solutions split the data into training and testing datasets with 85 percent to 15 percent respectively. Both models to evaluate and compare and comparison of the pre-processed model with the raw one, True Positives, False Positives, True Negatives and False Negatives were calculated. The values were visualised with a confusion matrix and used to obtain the F1 score, precision, and recall.

1) Global Approach: Both pre-processed and raw datasets were pysparksql data frames, for which pipelines were created. The pipelines consisted of a vector assembler and cross-validator. To build and train machine learning models, in this case, Random Forest and Multi-Layer Perceptron the data is required to be into a single vector. Therefore, the Vector assembler was used and generated the vector which contained all the feature columns, while the "is fraud" column was dropped. CrossValidator and ParamGrid was applied to look across all given combinations of the parameters to find a model[9]. The parameters for the function to choose from were built upon two-parameter grids. For the Random Forest, it consisted of the depth of each tree in the forest, the maximum number of leaves and impurity, while for the Multi-Layer Perceptron it consisted of the number of layers and nodes, solver method, maximum number of iterations. The hyperparameters for the models were chosen with the cross-validation method based on the Area Under the PR metrics.

Fig. 1. Flow diagram for global approach



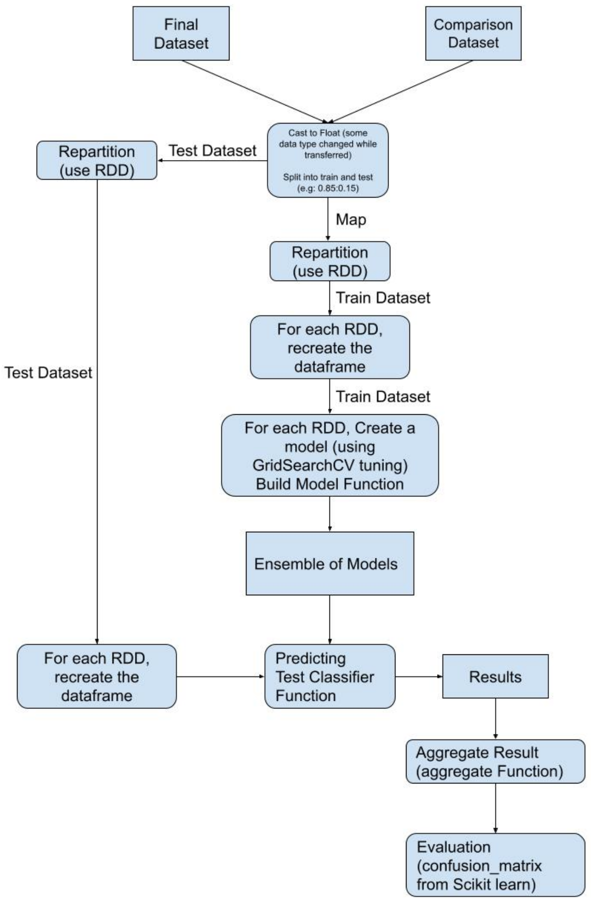
Final Dataset
Comparison Dataset
Cast to Float (some data type changed while transferred)
Split into train and test (e.g: 0.85:0.15)
Train Dataset
Pipeline: Assembler and Cross Validator
Train the Global Model
Assembler : Create the Input Vector for the model
Cross Validator: For Tuning. Need model, Param Grid, and Evaluator (auc for our case)
Final Model
Comparison Model
Test Dataset
Test Dataset
Evaluation (conf Matrix)
Evaluation (conf Matrix)

2) Local Approach: Both datasets were data frames and needed to be repartitioned using RDD. For each RDD the pysparksql data frame was recreated. Two parameters' grids were built. One for Random Forest took into account estimator, criterion(impurity) and a maximum depth of each tree in the forest while the second one, used for Multi-Layer Perceptron included a number of hidden layers and nodes, solver and maximum iterations. Each parameter was given a few possible values. GridSearchCV was applied to look across all given combinations of the hyperparameters to find the best models for each partition based on the F1 score. The obtained models were subjected to ensemble modelling, where the ensemble model aggregates the prediction of each model and predict the outcome using a majority vote.

C. Modelling Algorithm

1) Random Forest : The random forest classifier is made up of a number of tree classifiers, each of which is constructed using a random vector sampled separately from the input vector, and each tree casts a unit vote for the most popular class in order to classify an input vector. To grow a tree, the random forest classifier utilised in this study uses a grid search method to select the hyperparameters for training purposes

Fig. 2. Flow diagram for local approach



Final Dataset
Comparison Dataset
Cast to Float (some data type changed while transferred)
Split into train and test (e.g: 0.85:0.15)
Test Dataset
Repartition (use RDD)
Map
Repartition (use RDD)
Train Dataset
For each RDD, recreate the dataframe
Train Dataset
For each RDD, Create a model (using GridSearchCV tuning) Build Model Function
Test Dataset
Ensemble of Models
For each RDD, recreate the dataframe
Predicting Test Classifier Function
Results
Aggregate Result (aggregate Function)
Evaluation (confusion_matrix from Scikit learn)

[10]. The hyperparameters for the random forest algorithm in the global approach were tuned by using ParamGridBuilder function [11] and with grid search in the local approach, and the evaluation metric is area Under PR.

The hyperparameters that were selected for training are: Global Approach:

HyperParameters	Clustered Data	Non Clustered data
MaxDepth	20	20
NumTrees	20	20
Impurity	'entropy'	'entropy'

TABLE I

Local Approach:

HyperParameters	Clustered Data	Non Clustered data
MaxDepth	5	5,10
N estimators	10,15	10,15
Criterion	'gini'	'gini'

TABLE II

2) Multi-layer Perceptron: A multilayer perceptron classifier is a feedforward artificial neural network-based classifier. Multiple layers of nodes make up MLPC. Each layer in the network is fully connected to the next layer. The input data is represented by nodes in the input layer. All other nodes use a linear combination of the inputs with the node's weights and bias, as well as an activation function, to map inputs to outputs [12]. The hyperparameters for the Multi-layer Perceptron in the global approach were tuned by using ParamGridBuilder function and with grid search in the local approach, and the evaluation metric is F1 score. The selected hyperparameters are.

Global Approach:

HyperParameters	Clustered Data	Non Clustered data
hidden layer	[22,100,2]	[22,100,2]
solver	lbfgs	lbfgs
max iter	100	100

TABLE III

Local Approach:

HyperParameters	Clustered Data	Non Clustered data
hidden layer	150	100,150
solver	sgd	sgd
max iter	200	300

TABLE IV

III. Evaluation

To evaluate the results, several environments and comparisons are made. Such as different dataset, model, aggregation, and approach.

A. Dataset

To understand, if the preprocessing methodology in this experiment is impactful, a comparison dataset was used. However, several processing that was essential still be applied into the comparison dataset. The essential data processing steps are data balancing, altering string columns into numerical columns, and filling empty data. Feature selection is also applied to have same number of columns and comparable result. The difference in these datasets is clustering part. A significantly different process between dataset could be done in the future to have a better understanding in data preprocessing. Furthermore, a unique unseen dataset from other source can also be used to evaluate the models' performance in real world

B. Model

The second evaluation that done in this project is between models. In this project, three algorithms of model will be used. They are Random Forest, Multi-Layer Perceptron, and a combination of both. This was performed to understand which model algorithm perform better for fraud detection problem, and to see if a combination of model forms a better model.

A comparison between approach is also done between two approaches in big data. Local and Global approach. In this comparison two metric will be used, the performance and running time.

Lastly, two different way of aggregating ensemble model prediction is done. The first is to use majority vote. The second method is a sensitive method, treating prediction as a majority class if any of the model predict it as a member of majority class. However, since there are only two algorithms used in this project, this comparison can only be performed in local approach on combination of model. Because, on the other approach, only two model was used and both methods will have the same result. This problem could be tackled by implementing more algorithm model in the future work.

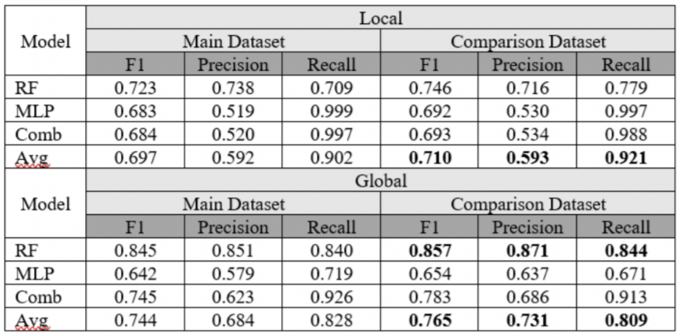
C. Evaluation Metric

To evaluate the performance of the models several metrics are used. In [13] mention that the most relevant metrics for imbalanced dataset is F-measure, recall and precision. Therefore, those metrics are going to be calculated in this project. Other metric that used in this project is confusion matrix. Confusion matrix will be used to overview the result if there is any anomaly within the result.

Precision is a metric to measure how many of the majority class prediction is correct. Meanwhile, recall is a metric to measure how many of the majority class is successfully predicted by the model. F-measure is the harmony between precision and recall. Based on the definition, the focus in this project, is to get the highest F-measure and Recall possible.

IV. Results and Discussion

Fig. 3.



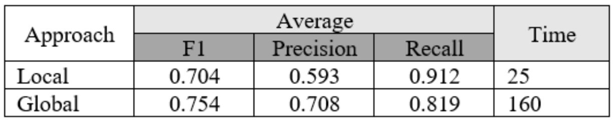
Model	Local					
	Main Dataset			Comparison Dataset		
	F1	Precision	Recall	F1	Precision	Recall
RF	0.723	0.738	0.709	0.746	0.716	0.779
MLP	0.683	0.519	0.999	0.692	0.530	0.997
Comb	0.684	0.520	0.997	0.693	0.534	0.988
Avg	0.697	0.592	0.902	**0.710**	**0.593**	**0.921**

Model	Global					
	Main Dataset			Comparison Dataset		
	F1	Precision	Recall	F1	Precision	Recall
RF	0.845	0.851	0.840	**0.857**	**0.871**	**0.844**
MLP	0.642	0.579	0.719	0.654	0.637	0.671
Comb	0.745	0.623	0.926	0.783	0.686	0.913
Avg	0.744	0.684	0.828	**0.765**	**0.731**	**0.809**

From the table above, it is seen that processed dataset have worse result than the comparison dataset that was not processed on both local and global approach. This was not as expected. It might be caused by the clustering part of processing that was not rigidly done. This mean that the data pre-processing approach taken in this project was not successful. In the table, recall score of MLP and combined model at local approach are very high. This is due to a bad prediction of models that give mostly positive results. This will be discussed further later in the confusion matrix discussion

From the above table, global approach has a better overall performance than local approach. Global approach obtains 0.754 score on F-measure while local approach obtains 0.704 score on F-measure. However, local approach gets a higher recall but a very low precision and F1 score as a trade-off.

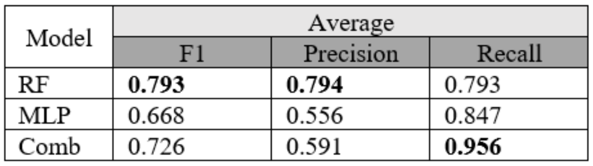
Fig. 4.



Approach	Average			Time
	F1	Precision	Recall	
Local	0.704	0.593	0.912	25
Global	0.754	0.708	0.819	160

This happen because two of the models in local approach are predicting more toward positive class. This can be seen in later part on confusion matrix tables. On the other hand, even though local approach receives a lower score, the training time used by local approach are five time less than global approach need. This time difference and result difference might also be contributed by implementing different libraries for each approach. This because the limitation of each library in implementing the approach.

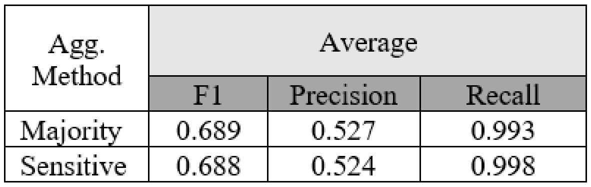
Fig. 5.



Model	Average		
	F1	Precision	Recall
RF	**0.793**	**0.794**	0.793
MLP	0.668	0.556	0.847
Comb	0.726	0.591	**0.956**

From the table above, can be seen that RF model has the highest F1 and precision score with 0.793 and 0.794 respectively. While ensemble of models has the highest recall with 0.956 point. However, ensemble model has a low precision score with only 0.591. this is because the ensemble models inherit the low precision from MLP. Implementing other multiple model that have high recall and precision might be able to tackle the low precision problem. It can be concluded that from this project, the best models are random forest models.

Fig. 6.

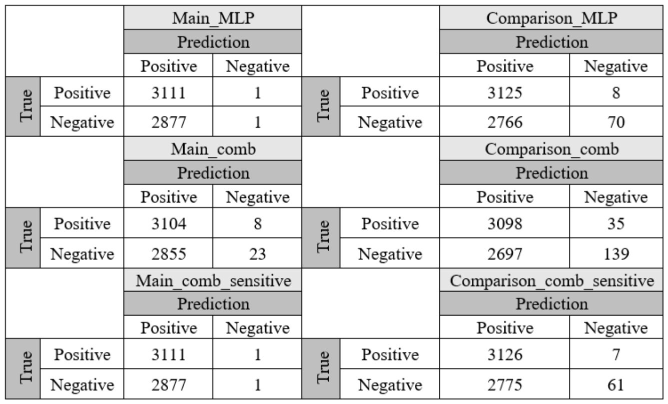


Agg. Method	Average		
	F1	Precision	Recall
Majority	0.689	0.527	0.993
Sensitive	0.688	0.524	0.998

The result of aggregation method of local approach comparison is that sensitive method gives a higher recall score. But there is some trade off on the F1 and precision score. This was expected as the sensitive method are leaning toward giving positive prediction. However, this result is based on a poor MLP models. Therefore, a conclusion whether which method is better could not be made. To tackle this problem, a better

MLP model should be trained or implementation of other machine learning algorithm to reduce the poor performance effect given by MLP models.

Fig. 7.

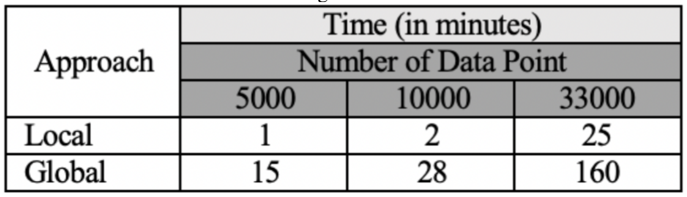


		Main_MLP				Comparison_MLP	
		Prediction				Prediction	
		Positive	Negative			Positive	Negative
True	Positive	3111	1	True	Positive	3125	8
	Negative	2877	1		Negative	2766	70
		Main_comb				Comparison_comb	
		Prediction				Prediction	
		Positive	Negative			Positive	Negative
True	Positive	3104	8	True	Positive	3098	35
	Negative	2855	23		Negative	2697	139
		Main_comb_sensitive				Comparison_comb_sensitive	
		Prediction				Prediction	
		Positive	Negative			Positive	Negative
True	Positive	3111	1	True	Positive	3126	7
	Negative	2877	1		Negative	2775	61

Lastly, can be seen from the confusion matrix. The local approach of MLP models is giving more positive prediction than negative prediction. This performance also translates into the bad performance on both ensemble models either the majority vote performance or sensitive method performance. However, can also be seen that model that use comparison dataset give more negative prediction than main dataset. This also support the first claim that the pre-processing dataset is better.

From the table can be seen the scalability of each approach. Both local and global training time grow twice between 10000 and 5000 data point. However, the time grow significantly higher when training 30000. The time needed to train the models increase 12 times and almost 6 times for each local and global approach significantly. From this, can be seen that the time needed to train the models do not grow linearly by data points. It can be assumed that when training more than 60000 data point the time it will need will be more than 6 times than when training 30000 data point. Therefore, can be concluded even though the global approach has better performance, the scalability of global approach is worse, seeing that it already needs 160 minutes to train 33000 data points.

Fig. 8.



Approach	Time (in minutes)		
	Number of Data Point		
	5000	10000	33000
Local	1	2	25
Global	15	28	160

V. Conclusion

Based on the result can be conclude that pre-processed dataset does not give a better result. However, to be considered the comparison dataset also cleaned and processed for the essential part. Meaning that data pre-processing can be used to create a big data solution for fraud detection. But the clustering part of pre-processing does not give benefit on data pre-processing for machine learning. Can be seen that a successful model is created using the global approach using random forest and trained using a processed comparison dataset. The model having 0.857, 0.871, 0.844 score on F1, precision and recall respectively. From this experiment also known that Global model is a better approach than local but having a long training time as a trade-off. Lastly, no conclusion can be made on which aggregation method is better for fraud detection problem due to bad models.

For future works, there are several areas that can be improved. Firstly, evaluations dataset, to get a real performance of the model another dataset that never been seen can be searched. Second, more algorithm implementation, this project having a problem of bad ensemble model result. This problem can be tackled by implementing more of other machine learning algorithm. Next, a better model could be produced by having a using more parameter combinations on tuning phase. Lastly, a more rigid data pre-processing and analysis could be done to have a dataset that can train a better model.

References

[1] ”FRAUD - THE FACTS 2021”, Ukfinance.org.uk, 2021. [Online]. Available: https://www.ukfinance.org.uk/system/files/Fraud

[2] D. Tanasa and B. Trousse, “Advanced data preprocessing for intersites Web usage mining,” IEEE Intelligent System, vol. 19, no. 2, pp. 59-65, 2004.

[3] J. Brownlee, ”Undersampling Algorithms for Imbalanced Classification”, Machine Learning Mastery, 2022. [Online]. Available: https://machinelearningmastery.com/undersampling-algorithms-for-imbalanced-classification/.

[4] ”StringIndexer — PySpark 3.2.1 documentation”, Spark.apache.org, 2022. [Online]. Available: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.StringIndexer.html.

[5] ”Imputer — PySpark 3.2.1 documentation”, Spark.apache.org, 2022. [Online]. Available: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.Imputer.html.

[6] ”Bucketizer — PySpark 3.2.1 documentation”, Spark.apache.org, 2022. [Online]. Available: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.Bucketizer.html.

[7] ”ChiSqSelector — PySpark 3.2.1 documentation”, Spark.apache.org, 2022. [Online]. Available: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.ChiSqSelector.html.

[8] ”VectorAssembler — PySpark 3.1.1 documentation”, Spark.apache.org, 2022. [Online]. Available: https://spark.apache.org/docs/3.1.1/api/python/reference/api/pyspark.ml.feature.VectorAssembler.html.

[9] ”CrossValidator — PySpark 3.2.1 documentation”, Spark.apache.org, 2022. [Online]. Available: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.tuning.CrossValidator.html.

[10] ”sklearn.model_selection.GridSearchCV”, scikit-learn, 2022. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.

[11] ”ParamGridBuilder — PySpark 3.2.1 documentation”, Spark.apache.org, 2022. Available: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.tuning.ParamGridBuilder.html.

[12] Brownlee, ”Crash Course On Multi-Layer Perceptron Neural Networks”, Machine Learning Mastery, 2022. [Online]. Available: https://machinelearningmastery.com/neural-networks-crash-course/.

[13] C. Weng and J. Poon, ”A New Evaluation Measure for Imbalanced Datasets”, Proceedings of the 7th Australasian Data Mining Conference, vol. 87, pp. 27-32, 2008.