DATA MINING & TEXT ANALYSIS

Summative Assessment

3398 words

EXECUTIVE SUMMARY

In this assignment, I used the *fraudTrain.csv* dataset to find solutions for Question 1 and Question 4.

First, I explored the dataset, studied about all the attributes and carried out descriptive statistical analyses to gain a good understanding about the dataset. For Question 1 regarding building three different personal profiles that may help improve fraud detection, frequencies of transactions done in the last 7 and 30 days were generated (denoted as *count_7_days and count_30_days)*, to have a good understanding of customers' spending behaviour. From that, feature selection was implemented to find out which attributes significantly affect the class attribute "*is_fraud*". Three different sets of attributes were chosen to build machine learning (ML) models for credit card fraud prediction. The results of feature selection show the importance of *count_7_days* and *count_30_days* and other attributes such as amount, transaction date and time and categories on fraud detection. Considering the imbalanced and large size of the dataset, under-sampling with ratio 50-50 was carried out. The models were then trained and tested on the balanced dataset using different classifiers with 10-fold cross-validation, then evaluated models using confusion matrix and its derivatives. The obtained results are quite promising for all three sets of attributes, having achieved high accuracy, recall and precision (see Figure 2). This demonstrated that we can help improve fraud detection by building models based on customers' spending behaviour, transaction amount, transaction time and other information such as categories and customers' age.

For Question 4, which is to find out a solution to help protect high-value targets based on transaction amount, time of day of transactions and other attributes, I first explored the relationship between transaction amounts and transaction time. Then based on the result of feature selection in Question 1, other attributes from a dataset were chosen to train and test ML models. The models were then evaluated based on evaluation metrices. The results show that Decision Tree algorithm provided the best model with 97.12 % accuracy and 0.97 recall, which means we can help identify and protect high-value targets based on transactions amount and time, and customers' age and spending categories.

As future directions, ML models to predict and prevent fraud detection based on location of transaction will be built, given existing evidences about the relationship between fraud transaction and its locations [1]. Advanced ML algorithms such as deep learning will also be used to improve accuracy of fraud detection models.

TASK 1: DISCUSSSION OF TECHNIQUES USED IN TWO SOLUTIONS

For task 1, I chose to answer Question 1 and Question 4, using the *fraudTrain.csv* dataset.

Pre-processing

To begin with, data exploration was carried out for an overall picture of the nature and structure of the dataset. This is important e.g. to check the data types of attributes as they affect the choice of methods for analysis, including basic statistics and algorithms to identify relationships between attributes [2].

The dataset contains simulated credit card transactions which are either genuine or fraud. It includes credit cards of 983 customers doing transactions with 693 merchants. The dataset has 1296675 entries and 23 attributes, in which 11 attributes were numeric and 12 attributes were object. The dataset had no missing value, which simplifies the cleaning process. It is highly imbalanced with the class attribute "*is_fraud*" containing 7506 fraud transactions and 1289169 non-fraud transactions (i.e., only 0.58% of the total transactions are fraud). As imbalanced data can lead to poor performance of predictive models[10] (predictions have a high accuracy score even without detecting a fraud transaction), resampling was carried out to balance the dataset. Considering the huge size of the dataset, under-sampling algorithm was used to generate a smaller dataset with 50:50 distribution of *fraud*/*non_fraud* transactions (this new dataset is called "**sample_1**"). Such an approach has proven successful for generating ML algorithms with high performance [3][4][5]. For further details, see Figures A1, A2, A4 in Appendix 1.

To answer Question 1, frequencies of transactions made in the last 7 and 30 days were generated, corresponding to new features *count_7_days* and *count_30_days*. The hypothesis is that understanding these customers' spending behaviours might help detect fraud transactions more efficiently (Figure A3, Appendix 1).

Feature selection was then performed on the balanced dataset, using Weka, to find out which attributes are highly correlated with the class attribute "*is_fraud*". Feature selection can help improve ML models performance by reducing data dimensionality and eliminating redundant attributes [6](pg. 58-61). The obtained results would allow identifying different sets of attributes that are linked to high levels of fraudulent actions.

As the first step of feature selection, the dataset "sample_1" is cleaned from special characters that prevent it from running on Weka (Figure A5, Appendix 2). The result is saved as a *csv* file, which is then imported to Weka and then converted to *arff* file. This arff file was then used in Weka for feature selection.

Feature selection was performed by several single-attributes evaluator methods with ranking which sort attributes into rank order of their evaluation [7](p.564), including *GainRatioAttributeEval* and *InfoGainAttributeEval*. Their results are quite similar, see Figure 1.

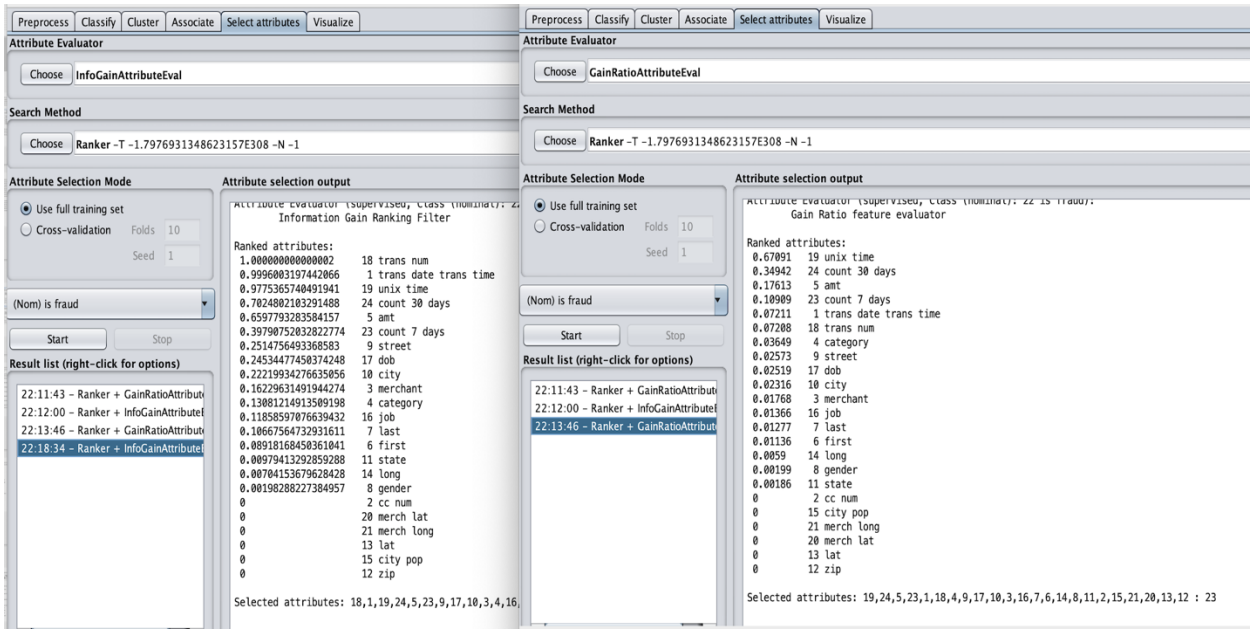

Figure 1. Attribute selection with InfoGainAttributeEval (left) and GainRatioAttributeEval (right)

Namely, they show that some attributes have more influence on the dependent attribute '*is fraud*' than others. As such, I decided to choose three following sets of attributes to build the profile of fraudsters:

Set 1: amount, count 7 days, count 30 days, is fraud
Set 2: amount, count 30 days, DOB, is fraud
Set 3: trans date trans time , DOB, category, is fraud

In Python, all the relevant attributes above were extracted into a new dataframe (denoted by "**sample_2"**). Then, since some classification algorithms being used later require categorical attributes [6](pg. 63), transformation such discretization was carried out, as follows:

- Transaction hour from{Citation} 'trans date trans time' was extracted to create new attribute 'hour', which is categorized into six groups: 0-3 , 4-7, 8-11, 12-15, 16-19, 20-23 (Figure A6, Appendix 2)
- Customers' age was obtained from 'dob'. It is organised into eight age groups (Figure A7, Appendix 2)

As a result, the following new feature sets were obtained for ML models training

- **Set 1**: *amount*, *count_7_days*, *count_30_days*, *is_fraud*.
- **Set 2**: *amount*, *count_30_days*, *age*, *is_fraud*.
- **Set 3**: *hour*, *age*, *category*, *is_fraud*.

Model training and testing:

The next step is to build ML models for fraud detection based on these feature sets. Since our aim is to predict whether a transaction is *fraud* or *non_fraud* based on values of other attributes, we use classification algorithms. Some popular classification algorithms,

namely, Decision Tree (DT), Random Forest (RF), Logistic Regression (LR), Support Vector Machine (SVM), Naïve Bayer (NB) and K-nearest neighbour (IBk), were used since they were proven successful for fraud detection in many previous research [8][9][10][11][12].

DT is based on "divide and conquer" method to generate a prediction rule. It works by recursively breaking down a problem into two or more sub-problems until it is simple enough to be solved directly. It is fast and can produces accurate outputs. RF contains a collection of decision trees on different subsets of a given dataset and takes average to improve the predictive accuracy on that dataset. LR is a regression model where the dependent variable is categorical, and it can handle continuous, binary and categorical data. NB classifier is a probabilistic method that predicts the class of future instances based on information from training instances. It is a speedy algorithm while retaining an accurate predictive ability. IBk classifies new cases based on a similarity measure. Each new instance is compared with existing ones using a distance metric, and the closest existing instance is used to assign the class to the new one. SVM classifies all training instances by separating them into correct classes through a hyperplane [6](chapters 3,4).

To train and test the models, the dataset was partitioned into training and testing sets using 10-fold cross-validation since it was proven useful for improving performance of ML models, achieving the best estimate of error and mitigating bias caused by hold-out approach. Indeed, the dataset was divided into ten equal parts. Nine parts were used to train the model and the remaining part was used to test the model. This process was repeated for ten times with the testing segment being alternated among the ten equal parts. Therefore, each part would be used for testing once and for training nine times [7],p.167-168].

All ML models were generated in Weka using the *Classify* tool. Note that before generating models, the dependent attribute '*is fraud*' was converted from numeric to nominal using Weka filter *numericToNominal*.

Model evaluation:

The models were evaluated and compared based on confusion matrix and its derivatives, such as accuracy, precision, recall, F-Measure, ROC, in order to find the best model. Parameters of confusion matrix include [13]

1. True Positive (**TP**): fraud transaction correctly predicted by the model.
2. True Negative (**TN**): non-fraud transaction correctly predicted by model.
3. False Positive (**FP**): non-fraud transaction wrongly predicted as fraud by model.
4. False Negative (**FN**): fraud transaction wrongly predicted as non-fraud by model.

The derivatives of these matrices are

1. **Accuracy** = (TP+TN)/(TP+TN+FP+FN)
2. **Precision** = TP/(TP+FP)
3. **Recall** = TP/(TP+FN)
4. **F-Score** = 2* (Precision * Recall)/(Recall+ Precision)
5. **ROC**: The TP rate against the FP rate

Evaluation of algorithms against the above parameters would generate a true picture of how they perform. For all the parameters above, **1** corresponds to the best outcome, and **0** to the worst one [7](p.179-194). Since our aim is to improve fraud detection, the false negative rate (FN) needs to be as small as possible. Thus, the most important metric to focus on when evaluating models is Recall.

The results are presented in Figure 2 below

Q1. Set 1: Amount, count_7_days, count_30_days, is_fraud

	Accuracy	TP rate	FP Rate	Precision	Recall	F-Measure	ROC area
Logistic Regression	97.05%	0.97	0.03	0.971	0.97	0.97	0.985
SMO(support vector machine)	96.50%	0.965	0.035	0.966	0.965	0.965	0.965
Naïve Bayer	95.18%	0.952	0.048	0.952	0.952	0.952	0.983
J-48 (Decision tree)	**98.02%**	**0.98**	**0.02**	**0.98**	**0.98**	**0.98**	**0.99**
Random Forest	97.52%	0.975	0.025	0.975	0.975	0.975	0.993
Ibk (K-nearest neighbour)	97.07%	0.971	0.029	0.971	0.971	0.971	0.97

Q1. Set 2: amount, count 30 days, age, is fraud

	Accuracy	TP rate	FP Rate	Precision	Recall	F-Measure	ROC area
Logistic Regression	96.58%	0.966	0.034	0.967	0.966	0.966	0.985
SMO(support vector machine)	95.98%	0.96	0.04	0.961	0.96	0.96	0.96
Naïve Bayer	96.09%	0.961	0.039	0.961	0.961	0.961	0.987
J-48 (Decision tree)	**97.84%**	**0.978**	**0.022**	**0.979**	**0.978**	**0.978**	**0.984**
Random Forest	97.69%	0.977	0.023	0.977	0.977	0.977	0.996
Ibk (K-nearest neighbour)	96.94%	0.969	0.031	0.969	0.969	0.969	0.969

Q1. Set 3: hour, age, category, is fraud

	Accuracy	TP rate	FP Rate	Precision	Recall	F-Measure	ROC area
Logistic Regression	78.76%	0.788	0.212	0.788	0.788	0.788	0.87
SMO(support vector machine)	78.23%	0.782	0.218	0.783	0.782	0.782	0.782
Naïve Bayer	77.80%	0.778	0.222	0.778	0.778	0.778	0.859
J-48 (Decision tree)	81.78%	0.818	0.182	0.818	0.818	0.818	0.888
Random Forest	81.86%	0.819	0.181	0.819	0.819	0.819	0.901
Ibk (K-nearest neighbour)	**82.03%**	**0.82**	**0.18**	**0.82**	**0.82**	**0.82**	**0.901**

Figure 2. Performance of ML algorithms

As we can see, the obtained results on the balanced dataset are promising. The best algorithm for Set 1 is DT (Recall = 0.98, Accuracy = 98.02%), for Set 2 is also DT (Recall = 0.98, Accuracy = 97.84%) (*figure 3*), for Set 3 is K-nearest neighbour (Recall = 0.82, Accuracy = 82.03%) (*figure 4*).

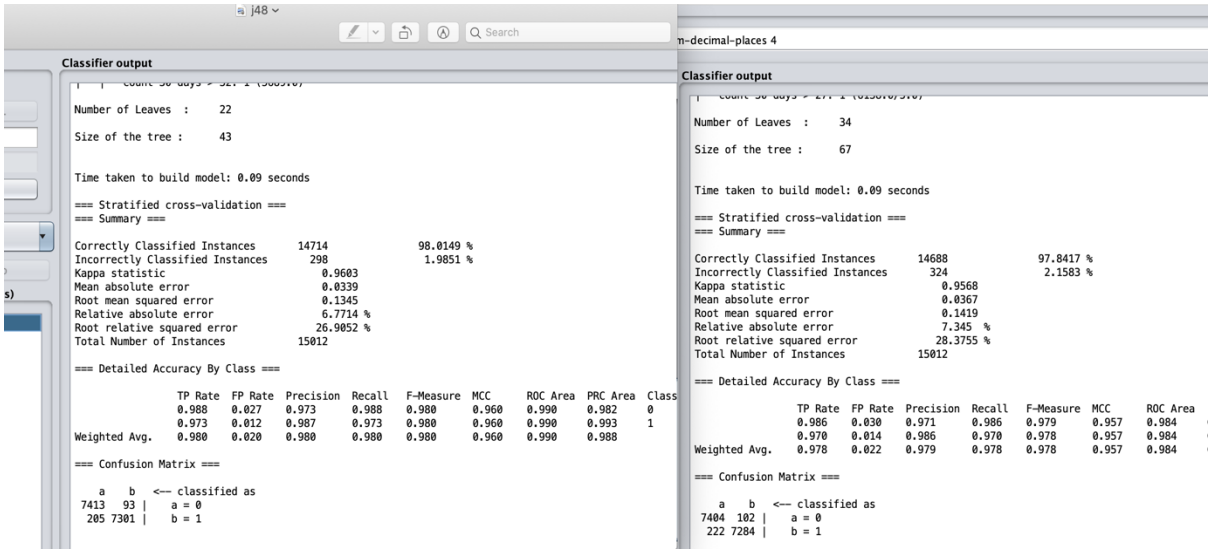

Figure 3. DT classifier for set 1 (left) and set 2 (right)

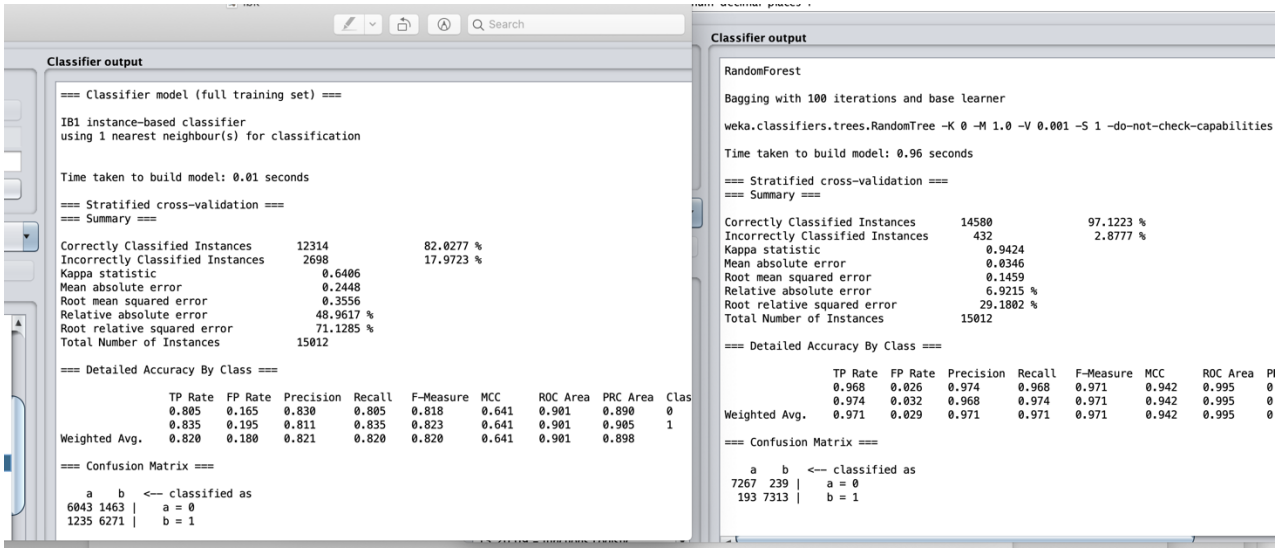

Figure 4. IBk classifier for set 3 (left) and RF classifier for set 4 (right)

Question 4:

I first examined the relationship between transaction amounts (‘*amount’*) and time of day of the transactions (generated attribute ‘*hour’*). To do that, the number of total transactions per hour (group hour) and that of fraud transactions per hour (group hour) were calculated (Figures A8 and A9 in Appendix 3). The results are presented in Figure 5 below.

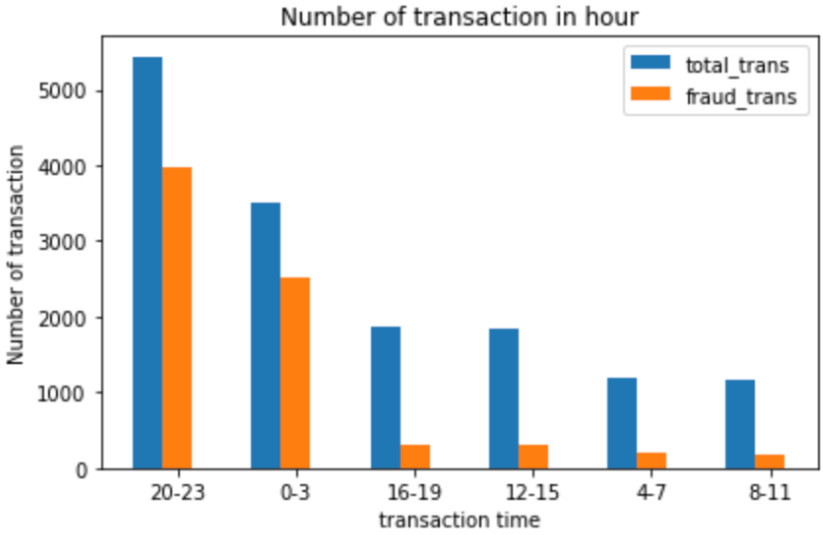

Figure 5. Number of total transactions and fraud transaction in hour.

We observe that, between 20-23 o'clock and between 0-3 o'clock, the percentages of fraud transactions are very high compared to total transactions. That is, fraud transactions happen predominantly from 20 to 3 o'clock.

Now, as the result of the feature selection (performed in Question 1), except for the transaction amount and time, attributes that have a significant influence on the dependent attribute '*is fraud*' are "*category*" and "*age*". Denoting **Set 4** the set of the selected features, i.e. *amount*, *hour*, *category* and *age*. Based on this set, ML models were trained and tested, where the evaluation results are presented in Figure 6 below.

Q4. Set 4: amount, hour, age, category, is fraud

	Accuracy	TP rate	FP Rate	Precision	Recall	F-Measure	ROC area
Logistic Regression	86.47%	0.865	0.135	0.865	0.865	0.865	0.949
SMO(support vector machine)	85.84%	0.858	0.142	0.862	0.858	0.858	0.858
Naïve Bayer	83.70%	0.837	0.163	0.852	0.837	0.835	0.923
J-48 (Decision tree)	**97.12%**	**0.971**	**0.029**	**0.971**	**0.971**	**0.971**	**0.991**
Random Forest	**97.12%**	**0.971**	**0.029**	**0.971**	**0.971**	**0.971**	**0.995**
Ibk (K-nearest neighbour)	96.54%	0.965	0.035	0.965	0.965	0.965	0.966

Figure 6. Evaluation metrics of ML models for Set 4

The results show that DT and RF gave the best models with 97.12% accuracy and 0.97 recall (*figure 4*) . It is noteworthy that DT is much faster than RF.

TASK 2: EVALUATION OF THE TOOLS/LANGUAGES

For the assignment, two tools, **Python** and **Weka**, were used. Python was used for data exploration, data cleaning and data pre-processing while Weka was used for model training and testing, for the following reasons.

Python is one of the most used programming languages for data science because of its simplicity, yet possessing a powerful set of libraries and tools. Python codes are intuitive and straightforward. Python provides essential libraries for data science, such as NumPy, Pandas,

Matplotlib and SciKit-Learn [14]. Importantly, these libraries are continuously enhanced and updated over time.
In the assignment, we need to read a *csv* file with tabular data, then perform data exploration, resampling and transformation. All these can be conveniently done in Python using its Pandas library. Indeed, Pandas is a powerful library for data cleaning and manipulation, with high-level data structures, and is designed to work with tabular data in a fast, easy, and expressive way. Its functions for data import, export, indexing, and data manipulation are simple and effective compared to other popular programming languages like Java. For example, a simple line of code is required for importing a csv file: *"df = pd. read_csv ("file_name")* while in Java, it is much longer [25](pg.527-530).
We also need to perform several descriptive statistical analyses on the dataset, which was convenient using Pandas built-in functions such as info(), describe(), mean(), median(), count(), quantile() and std(). Data was resampled, filtered, combined and manipulated by other Pandas functions like sample(), concat(), groupby(), split(), loc() and replace(). These built-in methods are extremely useful and effective in data wrangling and data transformation process. To make the analysis more intuitive, Python's Matplotlib library was used for plotting and visualization. It was also used as part of the exploratory process, to identify outliers in attribute "*amount*" in our case.
Another reason for choosing Python is because of its powerful web interface, *Jupyter Notebook*, which allows a convenient integration of code, outputs and visualization in a single document. This feature is of great usefulness for developing and presenting a data science project [14].

After data pre-processing, **Weka** was used for building ML models. Weka is a freely available tool for ML. It is written in Java and operates on almost every device. It offers many ML algorithms and data mining techniques, such as classifications, regressions, clustering, association rules, as well as many supervised and un-supervised pre-processing techniques for data transformation. It also has convenient tools for attribute selection and data visualization [15]. Weka has powerful tools and a graphical user interface that helps beginners to perform advanced ML techniques without writing codes.
For my assignment, the *numericToNominal* filter was used to transform class attribute "*is_fraud*" to nominal before applying classification algorithms. I also used attributes evaluator methods for feature selection to find out which attributes have strong influence on the class attribute. After that, I used *Classify* tool to perform different classification algorithms (DT, RF, LP, SVM, NB, IBk) to train and test ML models. It can be easily done under section Explorer/ Classify.
Although Weka is a great tool for ML, it has some important drawbacks. It cannot handle large datasets and is computational expensive. Before importing in Weka, data needs to be cleaned and prepared well, since there is no option for cleaning or replacing missing values in Weka, to the best of my knowledge.

For the abovementioned drawbacks of Weka, if this assignment was to be done again, I would implement ML models using the Scikit-Learn library in Python since it provides options for deploying state-of-the-art ML algorithms and more importantly, it can handle big datasets efficiently [14].

TASK 3: DISCUSSION OF THE CURRENT LITERATURE

As credit cards become more popular in everyday transactions, the number of frauds associated with them also increases significantly. A number of credit card fraud detection (**CCFD**) methods have been developed, mostly based on ML algorithms. They are divided into supervised and unsupervised ones. The former are based on previous records of fraudulent and legitimate transactions to classify new transactions, while the latter are often based on clustering to detect potential fraudulent transactions [10]. Below the main challenges in CCFD and the corresponding solutions are described, following by a brief literature of ML algorithms for tackling CCFD.

Several challenges in CCFD, and solutions to address them, can be identified. *First*, there is a significant lack of real-life data due to data privacy and security. Many works in CCFD literature used the same dataset (of transactions for a 2-day duration in September 2013 [11][16][17][10][18]). Data of credit card users and user's behaviour is not easy to access due to customer privacy. To overcome this, all personal information can be modified to avoid personal information leak.

Second, most existing CCFD datasets are extremely imbalanced, with a very small percentage of fraudulent cases compared to non-fraud ones [16][8]. Imbalanced data can lead to poor predictive performance of ML models, since ML algorithms often bias toward the majority class [6]. A number of tactics were used to combat such imbalanced datasets, including under-sampling and over-sampling methods (i.e. reducing the majority class or raising the minority one, respectively). Moreover, a hybrid of under-sampling and over-sampling techniques was used to achieve two sets of distribution (10:90 and 34:64) for analysis [8]. Other popular methods include Synthetic Minority Oversampling Techniques (SMOTE) [5][16][19], condensed nearest neighbour (CNN) and random under-sampling (RUS) [16]. In short, all these resampling techniques are necessary when dealing with imbalanced data since they can enhance ML models performance significantly.

Finally, another major challenge in addressing CCFD is that the profiles of legitimate and fraudulent transactions are usually very similar, with fraudulent patterns constantly evolving over time. Fraudsters learn to mimic spending behaviour of normal cardholders while credit card fraud transactions are mainly exposed by an unusual phenomenon, making CCFD very challenging [3][16][17]. This leads to a decrease of true positive (i.e. models dismissing many fraudulent transactions) and a large number of false positive (legitimate transactions being flagged as fraudulent ones) in fraud detection ML models [8]. Several solutions have been proposed, mainly based on analysis of cardholders' spending behaviours, including transactions statistics, regional statistics, merchant-type statistics, time-based amount statistics and time-based number of transactions statistics [9]. Apparently, the results have not been sufficiently satisfactory given the large number of fraudulent transactions that happened in the past few years [24].

Reviewing literature of CCFD, four main groups of ML algorithms were identified*:*

- **Classification**: LR models [11][5], the NB algorithms[5][9], and DT [9] KNN [10], RF [5][18] are popular algorithms for cc fraud detection. In [11] LR, RD, SVM and a combination of certain classifiers was used, which led to high recall of over 91% on a European dataset. According to [5] and [18], RF algorithm gives the best results (best accuracy, best recall

and precision) compared to other popular classifiers. Note that in all projects, datasets need to be balanced before apply any ML algorithms to archive good results. KNN algorithm also performed well as reported in [8][10], where authors tested and compared it with other classical algorithms.

- **Unsupervised/Clustering**: Outlier detection and Clustering algorithms were used in [10][20][19][21]. Clustering algorithms divide objects into different groups based on features, and those objects that are far from any group are labelled as outliers. In [19], researchers first used clustering to divide cardholders into three groups (high, medium, low) based on transaction amount before applying other algorithms on these groups to find card holder behavioural patterns. In [10], outlier detection techniques are proven useful in minimizing false positive rates and increasing fraud detection rate. However, according to [12], while outlier detection algorithms are highly suitable for distinguishing fraudulent and legitimate data, they have seen limited used due to the difficulty of detecting outliers. Thus, they deserve more investigation and research.

- **Deep learning** [17][22][23]: different deep learning models were used to detect fraud transactions. In [17], different deep learning models were tested, and the best one is BiLSTM-MaxPooling-BiGRU-MaxPooling. In [17], researchers used Convolution Neuron Network to build a model that reach 99,62% accuracy.

- **Meta-learning** [3][4]. The meta-classifiers combine multiple algorithms to detect credit card fraud. It has been shown that each ML algorithms have their own set of assumptions, and by appropriately combining multiple algorithms they can complement each other's strengths and weaknesses. According to [3], the best combination of classifiers was found to be the k-Nearest Neighbour, DT, and NB algorithms.

In summary, it can be seen that some specific successes in addressing CCFD have been achieved over the years. However, we are still far from conquering all the challenges, as presented above. It is noteworthy that despite the significant body of research (as reviewed above) and a huge effort and investment from financial industry for tackling frauds and scams, the overall losses due to credit card frauds have been enormous. Indeed, according to *statista.com* [24], the value of total annual fraud losses in the UK in 2018, 2019 and 2020 are, respectively, 671.4 , 620.6 and 574.2 million GBP.

APPENDICES

Appendix 1: Exploring data

```
In [31]: df = pd.read_csv('fraudTrain_org.csv')
         pd.set_option("display.max_columns", None) # display all columns

In [57]: df.head(1)

Out[57]:
```

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	street	city	state	zip	lat	long
0	0	2019-01-01 00:00:18	2703186189652095	fraud_Rippin, Kub and Mann	misc_net	4.97	Jennifer	Banks	F	561 Perry Cove	Moravian Falls	NC	28654	36.0788	-81.1781

```
In [33]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1296675 entries, 0 to 1296674
Data columns (total 23 columns):
 #   Column                 Non-Null Count    Dtype
---  ------                 --------------    -----
 0   Unnamed: 0             1296675 non-null  int64
 1   trans_date_trans_time  1296675 non-null  object
 2   cc_num                 1296675 non-null  int64
 3   merchant               1296675 non-null  object
 4   category               1296675 non-null  object
 5   amt                    1296675 non-null  float64
 6   first                  1296675 non-null  object
 7   last                   1296675 non-null  object
 8   gender                 1296675 non-null  object
```

Figure A1. Exploring data

```
click to scroll output; double click to hide
In [5]: df.isna().sum() # check for missing data
...

In [6]: print(len(df['merchant'].unique()))
        print(len(df['cc_num'].unique()))

        693
        983

In [8]: # Count values in 'is_fraud' attribute: 0 = non_fraud transaction, 1 = fraud transaction.
        df['is_fraud'].value_counts()

Out[8]: 0    1289169
        1       7506
        Name: is_fraud, dtype: int64

In [9]: # Calculate percentage of fraud transactions
        print('Frauds', round(df['is_fraud'].value_counts()[1]/len(df) * 100,2), '% of the dataset')

        Frauds 0.58 % of the dataset
```

Figure A2. Exploring data - 2

Generate new features 'count_7_days' and 'count_30_days' corresponding to frequencies of transaction done in the last 7 and 30 days.

```
In [20]: #to generate frequencies of transactions done in last 7/30days, I used pandas rolling function
         df['trans_date_trans_time'] = pd.to_datetime(df['trans_date_trans_time'])

         # Extract frequencies of transactions in last 7/30 days
         def last7DaysTransactionCount(x):
             temp = pd.Series(x.index, index = x.trans_date_trans_time, name='count_7_days').sort_index()
             count_7_days = temp.rolling('7d').count() - 1
             count_7_days.index = temp.values
             x['count_7_days'] = count_7_days.reindex(x.index)
             return x
         def last30DaysTransactionCount(x):
             temp = pd.Series(x.index, index = x.trans_date_trans_time, name='count_30_days').sort_index()
             count_30_days = temp.rolling('30d').count() - 1
             count_30_days.index = temp.values
             x['count_30_days'] = count_30_days.reindex(x.index)
             return x

In [21]: df_new = df.groupby('cc_num').apply(last7DaysTransactionCount)

In [22]: df_new = df_new.groupby('cc_num').apply(last30DaysTransactionCount)

In [23]: df_new.head(3)

Out[23]:
```

	lat	long	city_pop	job	dob	trans_num	unix_time	merch_lat	merch_long	is_fraud	count_7_days	count_30_days
	36.0788	-81.1781	3495	Psychologist, counselling	1988-03-09	0b242abb623afc578575680df30655b9	1325376018	36.011293	-82.048315	0	0.0	0.0

Figure A3. Generate new feature "count_7_days" and "count_30_days"

UNDERSAMPLING DATASET

In [46]:
```
# Undersampling algorithm to generate smaller dataset with 50:50  distribution of fraud/non_fraud transactions

fraud_df = df_new.loc[df_new['is_fraud'] == 1]
non_fraud_df = df_new.loc[df_new['is_fraud'] == 0][:7506]  # since there are 7506 fraud transactions

normal_distributed_df = pd.concat([fraud_df, non_fraud_df])

# Shuffle dataframe rows
sample_1 = normal_distributed_df.sample(frac=1, random_state=42)

sample_1.head(1)
```

Out[46]:

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	street	city	state	zip	lat	l
1239072	1239072	2020-06-01 01:01:21	565399283797	fraud_Koepp-Witting	grocery_pos	312.81	Anthony	Allen	M	6993 Carr Lodge Apt. 311	Jordan Valley	OR	97910	42.8801	-117.

In [25]:
```
sample_1.drop('Unnamed: 0', axis=1, inplace=True)
```

In [26]:
```
sample_1.head(1)
```

Out[26]:

	trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	street	city	state	zip	lat	long	city_pc
1239072	2020-06-01 01:01:21	565399283797	fraud_Koepp-Witting	grocery_pos	312.81	Anthony	Allen	M	6993 Carr Lodge	Jordan Valley	OR	97910	42.8801	-117.281	64

Figure A4. Under-sampling dataset

Appendix 2. Question 1:

In [27]:
```
# save sample_1 df to a csv file
sample_1.to_csv("Final_sample_1.csv", index=False)
```

In [28]:
```
# Clean "sample_1" from special characters that prevented it to be run in Weka.

import re
new_str = ''
string = open('Final_sample_1.csv').read()
new_str = re.sub('[^a-zA-Z0-9\n\,\-\:\"\. ]', ' ', string) #Replace all special characters in the list with ' '
open('Final_sample_1.csv', 'w').write(new_str.lower())
```

Out[28]: 4054705

In [31]:
```
df1= pd.read_csv('Final_sample_1.csv')
```

In [32]:
```
df1.head(1)
```

...

In [34]:
```
# After attribute selection, drop not-interested attributes from dataframe
# and keep only attributes interested (amount, trans date trans time, DOB, count 7 days, count 30 days, category, is fr
sample_2 = df1.drop(['cc_num','merchant','first','last','street','city','state','zip', 'lat','long','city pop','job','t
```

In [35]:
```
sample_2.head(2)
```

Out[35]:

	trans date trans time	category	amt	dob	is fraud	count 7 days	count 30 days
0	2020-06-01 01:01:21	grocery pos	312.81	1993-11-24	1	5.0	23.0
1	2019-09-12 12:49:41	misc net	909.75	1972-07-01	1	24.0	119.0

Figure A5. Cleaning balanced dataset from special characters that prevent it to be run in Weka

TRANSFORMATION ATTRIBUTES

In [38]:
```
# Calculate age of users based on dob
sample_2['dob'] = sample_2.dob.str.slice(0,4)
#group year by tens
def group_by_10s(mylist):
    result = []
    decade = -1
    x = 0
    for i in (mylist):
        j = int(i)
        x = 2022 - j
        if x >= 80:
            result.append('80')
        elif x < 80 and x >= 70:
            result.append('70')
        elif x < 70 and x >= 60:
            result.append('60')
        elif x < 60 and x >= 50:
            result.append('50')
        elif x < 50 and x >= 40:
            result.append('40')
        elif x < 40 and x >= 30:
            result.append('30')
        elif x < 30 and x >= 20:
            result.append('20')
        else:
            result.append('18')
    return result

sample_2['age'] = (group_by_10s(sample_2['dob']))
```

Figure A6. Attribute transformation -1

In [41]:
```
# Transformation on 'trans date trans time'
# Turn activity date to date/time object in python:
sample_2['trans date trans time'] = pd.to_datetime(sample_2['trans date trans time'])

# extract hour into new column 'hour'
sample_2['hour'] = sample_2['trans date trans time'].dt.hour

# Group hour by 4
sample_2["hour"] = (sample_2["hour"].replace([0,1,2,3], "0-3"))
sample_2["hour"] = (sample_2["hour"].replace([4,5,6,7], "4-7"))
sample_2["hour"] = (sample_2["hour"].replace([8,9,10,11], "8-11"))
sample_2["hour"] = (sample_2["hour"].replace([12,13,14,15], "12-15"))
sample_2["hour"] = (sample_2["hour"].replace([16,17,18,19], "16-19"))
sample_2["hour"] = (sample_2["hour"].replace([20,21,22,23], "20-23"))
```

In [42]:
```
sample_2.head(2)
```

...

In [43]:
```
sample_3 = sample_2.drop(['trans date trans time','dob'], axis=1)
```

In [47]:
```
sample_3.head(2)
```

Out[47]:

	category	amt	is fraud	count 7 days	count 30 days	age	hour
0	grocery pos	312.81	1	5.0	23.0	20	0-3
1	misc net	909.75	1	24.0	119.0	50	12-15

In [45]:
```
# save sample_3 df to a csv file
sample_3.to_csv("Final_sample_3.csv", index=False)
```

Figure A7. Attribute transformation - 2

Appendix 3. Question 4:

```
In [48]: # count numbers of transactions each 4 hours
         sample_3['hour'].value_counts()

         ...

In [51]: # put obtained values in a list (list_1)
         list_1 = sample_3['hour'].value_counts().to_list()
         print(list_1)

         [5433, 3493, 1877, 1847, 1198, 1164]

In [52]: # count numbers of fraud-transactions each 4 hours
         sample_3_fraud = sample_3[sample_3['is fraud'] == 1]
         sample_3_fraud['hour'].value_counts()

         ...

In [53]: # put obtained values in a list (list_2)
         list_2 = sample_3_fraud['hour'].value_counts().to_list()
         print(list_2)

         [3971, 2527, 316, 312, 202, 178]
```

Figure A8. Exploring relationship between transaction amount and transaction time

```
In [54]: # BAR CHART - plt.bar(x,y) Number of total transaction and fraud transaction each 4 hours

         hour_x = ['20-23', '0-3','16-19', '12-15', '4-7', '8-11']
         x_indexes = np.arange(len(hour_x)) # arrange dif. bars
         width = 0.25 #(width of the bar and distances between bars)
         total_trans = list_1
         plt.bar(x_indexes - width, total_trans, width = width, label='total_trans')
         fraud_trans = list_2
         plt.bar(x_indexes, fraud_trans, width = width, label='fraud_trans')

         plt.xlabel('transaction time')
         plt.ylabel('Number of transaction')
         plt.title('Number of transaction in hour')
         plt.xticks(ticks = x_indexes, labels = hour_x) # make tick and label right value on x-ax (instead of index)
         plt.legend() # tell which line is which (add label to graph)
         plt.tight_layout()
         plt.show()
```

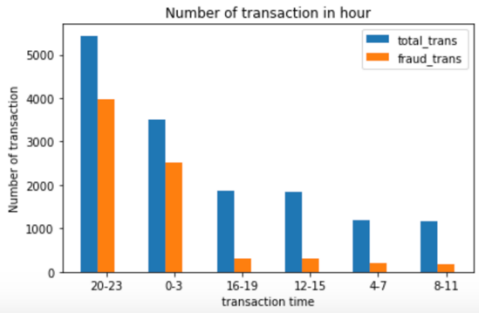

Figure A9. Visualizing relationship between transaction amount and transaction time

References

[1] A. Gulati, P. Dubey, C. MdFuzail, J. Norman, and R. Mangayarkarasi, 'Credit card fraud detection using neural network and geolocation', 2017, vol. 263, no. 4, p. 042039.

[2] J. D. Kelleher and B. Tierney, *Data science*. MIT Press, 2018.

[3] S. Stolfo, D. W. Fan, W. Lee, A. Prodromidis, and P. Chan, 'Credit card fraud detection using meta-learning: Issues and initial results', 1997, pp. 83–90.

[4] J. K.-F. Pun, 'Improving credit card fraud detection using a meta-learning strategy', 2011.

[5] D. Varmedja, M. Karanovic, S. Sladojevic, M. Arsenovic, and A. Anderla, 'Credit card fraud detection-machine learning methods', 2019, pp. 1–5.

[6] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to data mining*. Pearson Education India, 2016.
[7] I. H. Witten, E. Frank, M. A. Hall, and C. Pal, *Data mining: Practical Machine Learning Tools and Techniques*, 4th ed. Morgan Kaufmann, 2016.
[8] J. O. Awoyemi, A. O. Adetunmbi, and S. A. Oluwadare, 'Credit card fraud detection using machine learning techniques: A comparative analysis', 2017, pp. 1–9.
[9] A. C. Bahnsen, A. Stojanovic, D. Aouada, and B. Ottersten, 'Cost sensitive credit card fraud detection using Bayes minimum risk', 2013, vol. 1, pp. 333–338.
[10] N. Malini and M. Pushpa, 'Analysis on credit card fraud identification techniques based on KNN and outlier detection', 2017, pp. 255–258.
[11] A. Mishra and C. Ghorpade, 'Credit card fraud detection on the skewed data using various classification and ensemble techniques', 2018, pp. 1–5.
[12] E. W. Ngai, Y. Hu, Y. H. Wong, Y. Chen, and X. Sun, 'The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature', *Decis. Support Syst.*, vol. 50, no. 3, pp. 559–569, 2011.
[13] J. Kamiri and G. Mariga, 'Research Methods in Machine Learning: A Content Analysis', *Int. J. Comput. Inf. Technol. 2279-0764*, vol. 10, no. 2, 2021.
[14] W. McKinney, *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*. O'Reilly Media, Inc., 2012.
[15] B. Saleh, A. Saedi, A. Al-Aqbi, and L. Salman, 'Analysis of Weka Data Mining Techniques for Heart Disease Prediction System', *Int. J. Med. Rev.*, vol. 7, no. 1, pp. 15–24, 2020.
[16] A. Thennakoon, C. Bhagyani, S. Premadasa, S. Mihiranga, and N. Kuruwitaarachchi, 'Real-time credit card fraud detection using machine learning', 2019, pp. 488–493.
[17] A. M. Babu and A. Pratap, 'Credit Card Fraud Detection Using Deep Learning', 2020, pp. 32–36.
[18] S. Dhankhad, E. Mohammed, and B. Far, 'Supervised machine learning algorithms for credit card fraudulent transaction detection: a comparative study', 2018, pp. 122–125.
[19] V. N. Dornadula and S. Geetha, 'Credit card fraud detection using machine learning algorithms', *Procedia Comput. Sci.*, vol. 165, pp. 631–641, 2019.
[20] R. J. Bolton and D. J. Hand, 'Unsupervised profiling methods for fraud detection', *Credit Scoring Credit Control VII*, pp. 235–255, 2001.
[21] U. Porwal and S. Mukund, 'Credit card fraud detection in e-commerce', 2019, pp. 280–287.
[22] Z. Kazemi and H. Zarrabi, 'Using deep networks for fraud detection in the credit card transactions', 2017, pp. 0630–0633.
[23] H. Najadat, O. Altiti, A. A. Aqouleh, and M. Younes, 'Credit card fraud detection based on machine and deep learning', 2020, pp. 204–208.
[24] https://www.statista.com/statistics/286231/united-kingdom-uk-value-of-fraud-losses-on-uk-issued-cards/
[25] Charatan Q, Kans A. *Java in two semesters*. McGraw-Hill; 2006.