**Whether a given credit card transaction will be fraudulent or not?**

**DSE I2100 Applied Machine Learning and Data Mining Final Project**

**Marjan Rezvani  
Ayub ali Sarker  
 Maryam Akrami**

**Spring 2020**

**Abstract**

One of the most important responsibilities that a bank or financial institution has is to protect the integrity of the institution by working hard to protect the financial assets that it holds. Bank fraud can be defined as an unethical and/or criminal act by an individual or organization to illegally attempt to possess or receive money from a bank or financial institution.

It is anticipated that card frauds would amount to around $30 billion worldwide by 2020. So, how banks can improve security by detecting and obstructing frauds?

Machine learning is perfect for detecting frauds! Its algorithms learn to tell fraudulent transactions from legitimate operations.

In this project, we apply multiple ML techniques to the problem using card transaction data to identify fraudulent transactions (i.e. Fraud Detection)

We show that our proposed approaches are able to detect fraud transactions with high accuracy and reasonably low number of false positives.

**Introduction**

Banking Fraud has been an ever-growing issue with huge consequences to banks and customers, in terms of financial losses, trust and credibility.

An effective fraud detection system should be able to detect fraudulent transactions with high accuracy and efficiency.

A major challenge in applying ML to fraud detection is presence of highly imbalanced data sets. In many available datasets, majority of transactions are genuine with an extremely small percentage of fraudulent ones. Designing an accurate and efficient fraud detection system that is low on false positives but detects fraudulent activity effectively is a significant challenge.

In our project, we apply multiple binary classification approaches such as Logistic Regression, KNN, Random Forest and Voting Classifier to solve this problem on a labeled dataset that consists of payment transactions. our machine learning models collect information, analyzes the data gathered and extracts the required features.

Our goal is to build binary classifiers which are able to separate fraud transactions from non-fraud transactions. We compare the effectiveness of these approaches in detecting fraud transactions.

**Background**

Several ML and non-ML based approaches have been applied to the problem of fraud detection which gave us some ideas to do our project.

For instance, The paper <http://cs229.stanford.edu/proj2018/report/261.pdf> reviews and compares such multiple state of the techniques, datasets and evaluation criteria applied to this problem. it applies multiple binary classification approaches - Logistic regression, Linear SVM and SVM with RBF kernel on a labeled dataset. The paper <https://ieeexplore.ieee.org/abstract/document/1297040> proposes a rule based technique applied to fraud detection problem.

The paper <http://www.ecmlpkdd2018.org/wp-content/uploads/2018/09/567.pdf>

discusses the problem of imbalanced data that result in a very high number of false positives and proposes techniques to alleviate this problem.

**Data**

In this project we have used a dataset [<https://github.com/msarker000/ml-group-project/blob/master/data/transactions.txt.zip>] contains a year credit card transaction made in 2016. There are 641914 instances in the dataset. This dataset is highly unbalance with a low percentage of fraudulent transactions within several records of non-fraud transactions, which is shown in figure 1. Out of the 641914 transactions in the dataset, 10892 were fraudulent which means the True frauds account for 1.7% of all transactions. The feature named ‘isFraud’ is used to classify the transaction whether it is a fraud or not, which takes value 1 in case of fraud and 0 otherwise. There are also features that describe customers’ behavior are added, besides being fraud and not fraud. Some of numerical attributes are like available money, credit limit and categorical attributes like merchant name and transaction type. We also have few attributes such as echoBuffer, merchantCity, merchantState, merchantZip, posOnPremises, recurringAuthInd which totally have missing values that we dropped these columns. Also, the time of transactions does’t seem to really matters. So, we use the following features to train our models.

1) Transaction type

2) Transaction amount

3) Customer Id

4) Account number

5) Merchant name

6) Merchant category code

7) PosEntryMode

8) CurrentBalance

9) EnteredCVV

10) CardPresent

A picture containing large, orange, light, white

Description automatically generated

Figure 1

**Methods**

How did you take your data and set up the problem? Describe things like

normalization, feature selection, the models you chose. In this section, you

may have EDA and graphs showing exploration of hyper-parameters. Note:

Use graphs to illustrate interesting relationships that are important to your

final analyses. DO NOT just show a bunch of graphs because you can. You

should label and discuss every graph you include. There is no required number

to include. The graphs should help us understand your analysis process and

illuminate key features of the data.

**Evaluation**

Here you are going to show your different models performance. It is particularly

useful to show multiple metrics and things like ROC curves (for binary classifiers).

Make sure it is clear not just what the score is but for which instances in the data

one has the largest errors (in a regression), or just sample examples miss-classified.

Make an attempt to interpret the parameters of the model to understand what

was useful about the input data. Method comparison and sensitivity analyses are

absolutely CRUCIAL to good scientific work. To that end, you MUST compare

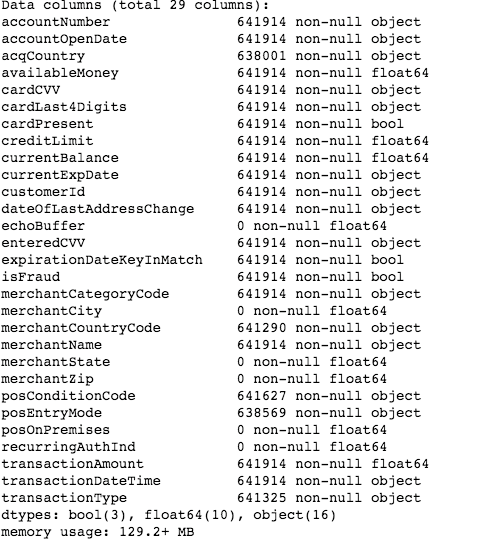
at least 2 different methods from class in answering your scientific questions. It

is important to report what you tried but do so SUCCINCTLY.



**Data Preparation for Model Development**

Data preparation is important part of Machine learning. We prepared the dataset before fit into machine learning model. We have 641914 instances and 29 features in the dataset. Here is the snapshot of our data.



#### **Handling missing values**

#### We have few options:

* + totally drop those attributes from data.
  + Drop those records (remove rows where these attributes are missing)
  + Set the missing to some values. For numerical attributes, we can set them to the mean/median, and for categorical attributes we can set them to the most frequent category.

We have few attributes which totally have missing values. These attributes are

* + echoBuffer,
  + merchantCity,
  + merchantState,
  + merchantZip,
  + posOnPremises,
  + recurringAuthInd

we dropped these features form the dataset.

* **Handle date and datetime features**

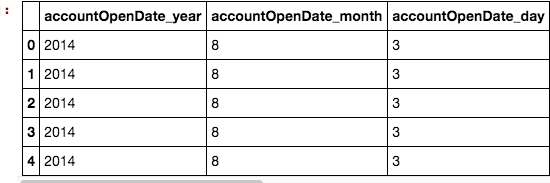
Different systems store dates in different formats: 11.12.2019, 2016-02-12, Sep 24, 2003 etc. But for building models on dates data, we need to somehow convert it to a numeric format. There are 3 most common methods to transform date to numeric format:

* + Unix timestamp
  + KSP date format
  + Divide into several features, construct new feature

For this dataset, the third method is used. Divide into several features. It is perfectly preserved intervals in easy intuition. The datetime features we have in our dataset are

* + accountOpenDate,
  + transactionDateTime,
  + currentExpDate ,
  + dateOfLastAddressChange

we divide these features into several new features. For example, accountOpenDate is converted into accountOpenDate\_year, accountOpenDate\_month, accountOpenDate\_day. Below is a snaphot of this conversion of accountOpenDate.



* **Categorial features**

Handling categorial feature is also very important part of preprocessing before feed data into machine learning model. There are two option we can do

* + Label Encoder
  + One-Hot encoding

We have 10 categorial features and those are

* + acqCountry
  + CardPresent
  + expirationDateKeyInMatch,
  + IsFraud
  + MerchantCategoryCode
  + MerchantCountryCode
  + MerchantName
  + PosConditionCode
  + PosEntryMode
  + TransactionType

We use Label Encoder to convert all the categorial features to numeric.

* **Resample to balance dataset**

We have 622,954 instances in ‘Not Fraud’ and 10, 892 instances in ‘Fraud’ class. If we plot these two classes by bar chart we can clear see how much the ‘Fraud’ class is relatively less count.



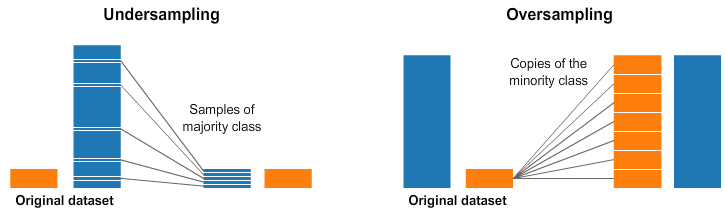
By looking into this picture, we can clearly see out data is unbalanced. One of the major issues that users fall into when dealing with unbalanced datasets relates to the metrics used to evaluate their model. Using simpler metrics like accuracy\_score can be misleading. In a dataset with highly unbalanced classes, if the classifier always "predicts" the most common class without performing any analysis of the features, it will still have a high accuracy rate.

We did an experiment with this unbalanced dataset. We feed this dataset with all features into XGBClassifier and we got accuracy score 98.25%. We also feed again with only one feature and got accuracy score 98.25%. Which is a metric trap.

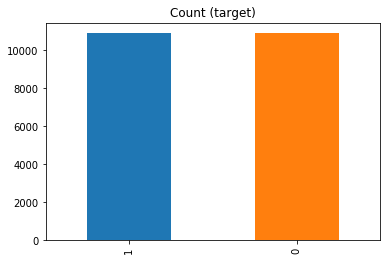
So, our dataset needs to balance before feed into any machine model.

* + **Resampling**

A widely adopted technique for dealing with highly unbalanced datasets is called resampling. It consists of removing samples from the majority class (under-sampling) and / or adding more examples from the minority class (over-sampling).



In our case, we under sampled the large class (Not fraud) into small class (Fraud). After doing this we have now 50% large class and 50 % small class and our data is balanced.

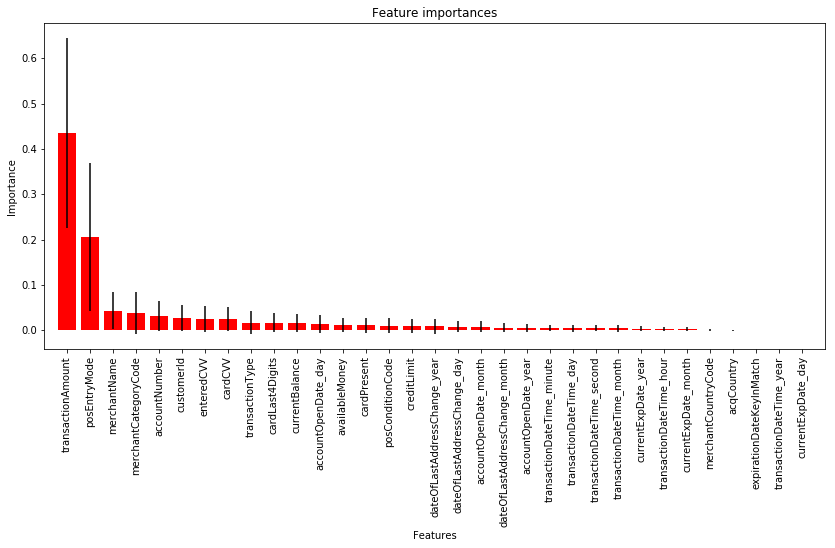


**Feature Selection**

Feature selection is important step in machine learning. By Feature selection we find important features those are most important in explaining the target variable. There are options

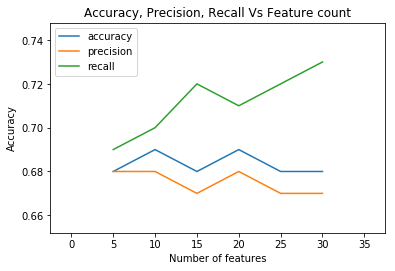
* RandomForestClassifier
* Recursive feature elimination (REF)
* Sequential backward selection (SBS)

In our dataset we used RandomForestClassifier to see feature’s importance. Here the result we got by feed our dataset.



We see here transaction amount has higher importance, then posEntryMode, and merchantName and so on. Some feature’s important are almost zero like acqCountry and merchantCountryCode.

To find top important sets of features we feed RandomForestClassifier to six different sets of top important features **5, 10, 15, 15, 20, 25, 30.**  and we plot the accuracy, precission and recal.



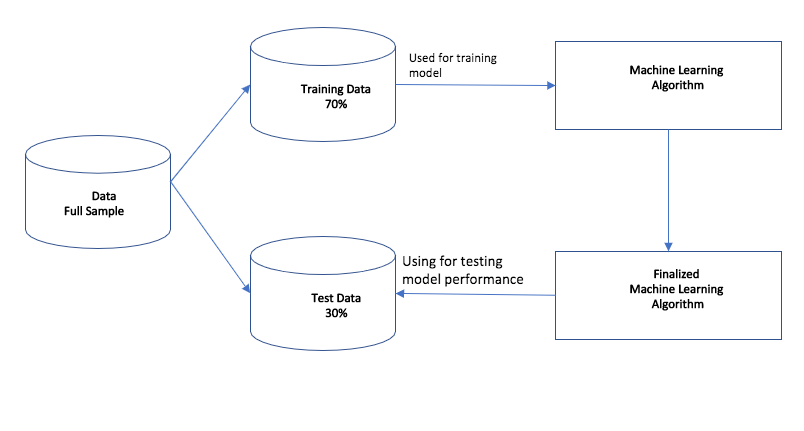
So, we see that accuracy, precision and recall for each 6 sets are almost same. but the set with 20 most important features has highest accuracy.

So, the followings are the high importance 20 features

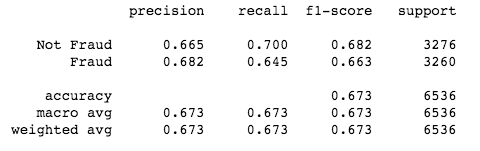
* transactionAmount (0.461284)
* posEntryMode (0.238895)
* merchantName (0.052225)
* merchantCategoryCode (0.042508)
* accountNumber (0.024675)
* customerId (0.022490)
* transactionType (0.021297)
* enteredCVV (0.018811)
* cardCVV (0.017843)
* cardPresent (0.016107)
* currentBalance (0.012183)
* accountOpenDate\_day (0.011294)
* cardLast4Digits (0.010590)
* dateOfLastAddressChange\_year (0.007816)
* dateOfLastAddressChange\_day (0.007122)
* creditLimit (0.006090)
* accountOpenDate\_month (0.005907)
* posConditionCode (0.005774)
* accountOpenDate\_year (0.003972)
* availableMoney (0.003922)

**Model**

In this Project (***Fraud Detection***), we are dealing with a ***classification*** problem. We investigated some ***supervised classification algorithms*** such as ***Logistic Regression***, ***KNN, Decision* Tree Classifier*, Adaboost Voting Classifier*** to solve this problem. We Standardized our preprocessed clean data and then split into train test split by the ration of 0 .30 and feed into model.



* **Logistic Regression**
  + **Add your stuff here**
* **KNN**
  + **Add your stuff here**
* **DecisionTreeClassifier**
  + We feed the decision tree classifier(max\_depth=5) without turning and we got the classification report like this



We see that **66.5%** precision for ‘Not Fraud’ class and **68.2%** for ‘Fraud’ class and average accuracy score is **67.3%**

* + We did 10-fold cross validation and we got accuracy score **67.2%**
  + **Parameter Tuning**

In DecessionTreeClassifier we can tune

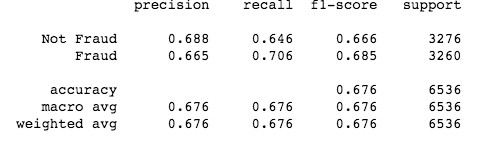
* + - Decision tree is max depth (the depth of the tree)
    - max feature (the feature used to classify)
    - criterion
    - splitter

GridSearchCV Search explores a range of parameters and finds the best combination of parameters. Then repeat the process several times until the best parameters are discovered.

After running GridSearchCV, we got best score **67.1%** and best params

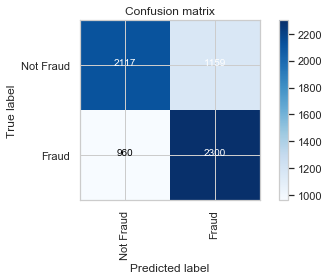
* + - criterion': 'entropy'
    - 'max\_depth': 6
    - 'max\_features': 20
    - 'splitter': 'best'

Here is the classification report we got after feeding our dataset to best model



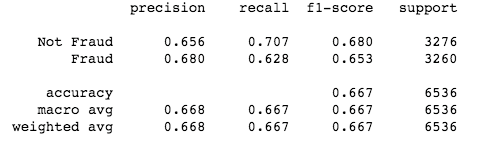
We see that no signification improvement by best model. Here we got accuracy score **67.6%**

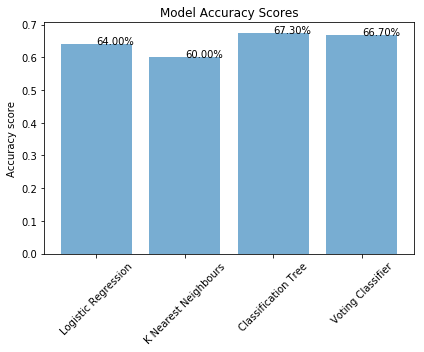
Here is the confusion matrix we got by best model



We see here we got TruePossitive is 2,117 and TrueNegative 2,300. And FlasePositive is 1,159 and false negative is 960.

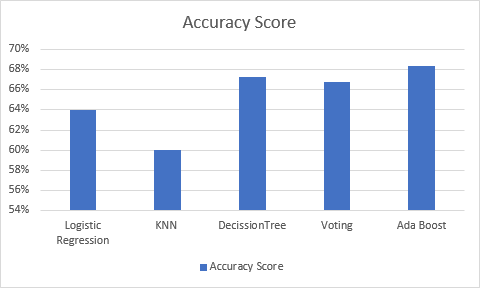
* **Voting Classifier**
  + A model combines multiple different models into a single model, which is (ideally) stronger than any of the individual models alone.
  + In our voting classing we combined three classifiers. LogisticRegression(64%), KNN(60%) and DecissionTreeClassifier(67.3%) and then feed our data to combined classifier and we got classification report like this.



* + We see here **65.6%** precision for “Not Fraud” class and **68%** precision for “Fraud” class and accuracy score is **66.8%.**
  + Not too much improvement by voting classifier as we expected.
  + Here is the comparative accuracy for classifiers
  + 
* **Adaboost Classifier**
  + **Add your stuff here**

**Result Comparison**

Here is the comparative result from all the model we feed in out project.



**We see that Decision Tree and Ada boost have higher accuracy compare to other classifiers and ada boost classifer has the highest score 68%.**

**Conclusion**

How well did it work? Characterize how robust you think the results are (did

you have enough data?) Try for interpretation of what the model found (what

variables were useful, what was not)? Try to avoid describing what you would

do if you had more time. If you have to make a statement about “future work”

limit it to one short statement.

**Attribution**

Using the number and size of github commits by author (bar graph), and the

git hub visualizations of when the commits occured. Using these measures each

person should self-report how many code-hours of thier work are visible in the

repo with 2-3 sentences listing their contribution. Do not report any code hours

that cannot be traces to commits. If you spend hours on a 2-line change of code,

or side-reading you did, you cannot report. If you do searches or research for the

project that does not result in code, you must create notes in a markdown file

(eg. in the project wiki) and the notes should be comeserate with the amount of

work reported. Notes cannot be simply copy pasted from elsewhere (obviously).

**Bibiliiography**

References should appear at the end of the report/notebook. Again, no specific

format is required but be consistant.

* <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>
* <https://medium.com/@haydar_ai/learning-data-science-day-22-cross-validation-and-parameter-tuning-b14bcbc6b012>
* <https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html>

**Appendex**

If there are minor results and graphs that you think should be included, put

them at the end. Do not include anything without an explanation. No random

graphs just for padding!! However, lets say you did a 50 state analysis of poverty

and demographics and your report focused on the 5 most interesting states, for

completeness you could include all in an appendex. Be sure though to provide

some (very short) discussion with each figure/code/result.