DATA MINING – CECS 632-01

# Comparative study on automatic fraud detection using datamining techniques

Sima Shafaei

25 Nov 2019

***Abstract***

*fraud means obtaining money or property through false pretenses. In recent years, Financial Fraud have begun to appear and continue to grow rapidly, which has shocked the confidence of investors and threatened the economics of entire countries. While auditors are the last line of defense to detect Fraud, still lack the experience and expertise threatening deal with the related risks. This study focused on applying different datamining techniques for automated fraud detection. It investigates usefulness of 4 different supervised datamining techniques including linear and non-linear SVM (Support Vector Machine), random forest, logistic and K-nearest neighbor and compares and summarizes their results. This survey covers technical detail about these methods and parameter tuning for them. Our obtained results show that among four selected classifiers random forest is the most accurate classifier, kNN is the fastest model in fitting time and Logistic has the minimum score time.*

# 1 Project

**Comparative study on automatic fraud detection using datamining techniques**

# 2 Problem description

The term fraud means obtaining money or property through false pretenses. In 2016 the Association of Certified Fraud Examiners (ACFE) reported that businesses lost 5% of their annual income through fraud[1]. According to the same report, the total loss for 2,410 cases was worth $ 6.3 billion[2]. Therefore, in today's competitive environment, fraud is one of the business critical problems. Automatic fraud detection is being part of the overall fraud control which can reduce the manual parts of a screening and checking process. This area has become one of the most established industry and government data mining applications[3]. Although it is impossible to be 100 percent certain about the intention behind a transaction or request but fraud detection enables business owners to identify and block suspicious activities and attempts based on available evidences and data using existing data mining and mathematical algorithms. Different research communities used solution based on artificial intelligence, auditing, database, distributed and parallel computing, economics, fuzzy logic, genetic algorithms, machine learning, neural networks, pattern recognition, statistics, visualization and others.[3] to automate fraud detection. In these paper we focused on applying several different datamining techniques including linear and non-linear SVM (Support Vector Machine), Neural Network,

random forest, logistic and K-nearest neighbor. We compared their accuracy on a fraud dataset which contains 434 features and 540484 samples.

In Section 3 we review and discussed several existing supervised fraud detection methods and the reasons for the choice of 6 classification algorithms. Section 4 contains briefly description about tools we used in this project and gives a short explanation of applied algorithms and their parameter in applied tools. Section 5 describes all steps of data cleaning, preprocessing and dimensionality reduction. It explains the tools and methodologies used, and the results obtained. In section 6 we described all steps of a data mining process, and all activities with selected data mining tools. Finally, Section 7 gives the detailed discussion of obtained results.

## 3 Review of publications

As previously mentioned, in this study, financial fraud is classified into four broad categories. Which are Bank Fraud, Insurance fraud, Securities and commodities fraud and Other related financial fraud [4] Each of the six data mining application classes (Classification, visualization, regression, clustering, prediction and outlier detection) has been applied in the area of FFD. These approaches differ in the classes of problems that they are able to solve. The main focus of this paper is on classification approach for supervised fraud detection.

The most frequently used techniques are logistic models, neural networks, the Bayesian belief network, and decision trees, all of which fall into the "classification" category. In [4] 75 FFD works analyzed and categorized based on their method in these paper, logistic models are the most popular, being used in 21.3% (16 of75) of the studies reviewed, followed by neural networks used in 13.3% (10 of 75), and then the Bayesian belief network and decision trees, both used in 6.7% (5 of 75) of studies.

[5] evaluates support vector machines and random forests, together with the well-known logistic regression, as part of an attempt to better detect (and thus control and prosecute) credit card fraud. Their study is based on real-life data of transactions from an international credit card operation. Based on this study random forest can capture higher proportion of fraud in comparison to SVM and Logistic regression and the result obtained by SVM is very similar to logistic regression achievement.

[6] introduces a support vector machine-based fraud warning (SVMFW) model to reduce the risk of FFS. This model integrates sequential forward selection (SFS), support vector machine (SVM), and a classification and regression tree (CART). SFS is employed to overcome information overload problems, and the SVM technique is then used to assess the likelihood of

FFS. Their experiment results show that their proposed model can reduce unnecessary information while satisfactorily detect FFS.

[7] investigates the usefulness of Decision Trees, Neural Networks and Bayesian Belief Networks in the identification of fraudulent financial statements. Based on their study, in terms of performance, the Bayesian Belief Network model achieved the best performance managing. Then the accuracy rates of the Neural Network model and the Decision Tree model came respectively. The Type I error rate was lower for all models. The Bayesian Belief Network revealed dependencies between falsification and the ratios debt to equity, net profit to total assets, sales to total assets, working capital to total assets and Z score. Each of these ratios refers to a different aspect of a firm's financial status, i.e., leverage, profitability, sales performance, solvency and financial distress, respectively. The Decision Tree model primarily associated falsification with financial distress, since it used Z score as a first level splitter.

# 4 Software Tools

This project is totally implemented in python 3. We used numpy, scikit-learn and pandas libraries to implement all data preprocessing steps and data mining algorithms. These Python modules are simple and efficient tools for data analysis, and fortunately they are accessible to everybody, open source and commercially usable. Although, we first start using Weka and then SaaS but we gave up because none of them gave us as much freedom as pandas and sickit-learn to work with and manipulate data. Moreover, Weka is not good for large dataset. It is very slow and can reduce the performance of our work.

Following modules are used in this project for preprocessing implementation:

| module | library | Description |
|---|---|---|
| isna() and fillna() | pandas | Handling missing values |
| get_dummies() | pandas | Handling categorical variables |
| resample() | sklearn.utils | Creating a balance dataset |
| VarianceThreshold | sklearn.feature_selection | For removing constant and quasi constant variables |
| min() & max() | pandas | Data normalization |
| corr() | pandas | Remove highly correlated features |
| SelectPercentile | sklearn.feature_selection | Select feature based on chi2 criterion |

| | | |
|---|---|---|
| SelectKBest<br>chi2 | | |

For classification we applied following modules

| module | library | Description |
|---|---|---|
| train_test_split | sklearn.model_selection | Splitting dataset into train and test partition |
| cross_validate | sklearn.model_selection | Implementing corss-validation for classifiers |
| RandomForestClassifier | sklearn.ensemble | Implementing random forest algorithm |
| SVC | sklearn.svm | Implementing linear and non-linear SVM classifier |
| LogisticRegression | sklearn.linear_model | Implementing logistic classifier |
| KNeighborsClassifier | sklearn.neighbors | Implementing KNN classifier |
| MLPClassifier | sklearn.neural_network | Implementing multi-layer perceptron classifier |

Following table shows a brief description of parameters we will use and tune during the mining process

| module | Parameter description |
|---|---|
| isna() and fillna() | isna does not have any parameter but fillna receive a value to use to fill missing values. We used different values for different types of data. More detail is provided below section 5 |
| get_dummies() | It receives the list of all categorical features to get their dummy indicators, we also set *'prefix'* parameter to original variables name and *'sep'* to '_' so new dummy variable names will have the name such as originalName_categoricalValue |
| resample() | The parameters include the fraction of dataset that we want to make new samples from it, *replace*=False/True which indicates resampling is done with replacement or not and *n_samples* that shows the number |

| | |
|---|---|
| | of sampling. More detail about the value of this parameter is provided in section 5. |
| VarianceThreshold | This module receive *threshold* as a float value that indicates the minimum variance we want to have in each column of dataset.<br>More detail about the value of this parameter is provided in section 5. |
| min() & max() | These two functions don't have any parameters they extract minimum and maximum value in each column of dataframe |
| corr() | This function doesn't get any input value and calculate the correlation matrix of our dataset |
| SelectPercentile | This module receive a criterion and a percentile and select features according to a percentile of the highest scores based on input criterion function.<br>More detail about the value of this parameter is provided in section 5. |
| SelectKBest | This module receive a criterion and an integer number and select features according to the K highest scores based on input criterion function.<br>More detail about the value of this parameter is provided in section 5. |
| chi2 | This function doesn't get any input value and compute chi2 score value for different column in dataset |
| train_test_split | It receives four input parameters. Two arrays which the first one is the matrix of input features and the second one is a vector of classification outputs. The length of matrix and vector should be equal.<br>*test_size* If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the test split. If integer, represents the absolute number of test samples.<br>*random_state* is the seed used by the random number generator. We select '4' as the input number<br>More detail about parameter tuning is provided in section 6 |
| cross_validate | It receives four inputs the first one is the estimator (classification method) then a matrix of input features and a vector of classification outputs. Finally, the *cv* parameter determines the cross-validation splitting strategy we used an integer that represent the number of folds.<br>More detail about parameter tuning is provided in section 6 |

| | |
|---|---|
| RandomForestClassifier | We used two parameters for this function:<br>*n_estimators* : integer, this shows the number of trees in the forest.<br>criterion : string, The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain<br>More detail about parameter tuning is provided in section 6 |
| SVC | We used four parameters for this function:<br><br>*C*: float, Penalty parameter C of the error term.<br>*kernel*: string, Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable.<br>*degree*: int, Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.<br>*gamma*: float, Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.<br>More detail about parameter tuning is provided in section 6 |
| LogisticRegression | We used four parameters for this function<br>*penalty*: str, It must be one of 'l1', 'l2', 'elasticnet' or 'none'<br>*max_iter*: int Maximum number of iterations taken for the solvers to converge.<br>*solver*: str, It must be one of 'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'<br>More detail about parameter tuning is provided in section 6 |
| KNeighborsClassifier | We used four parameters for this function<br>*n_neighbors* : int, Number of neighbors to use by default for kneighbors queries.<br>*weights* : str or callable, weight function used in prediction. Possible values are:<br>- '*uniform'*: uniform weights. All points in each neighborhood are weighted equally.<br>- '*distance'*: weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away. |

| | |
|---|---|
| | - [*callable*]: a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.<br>*algorithm*: It must be one of {'auto', 'ball_tree', 'kd_tree', 'brute'}, which are algorithm used to compute the nearest neighbors:<br>- 'ball_tree' will use BallTree<br>- 'kd_tree' will use KDTree<br>- 'brute' will use a brute-force search.<br>- 'auto' will attempt to decide the most appropriate algorithm based on the values passed to fit method.<br>*leaf_size*: int, optional (default = 30): Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem. |
| accuracy_score<br>f1_score | We used these methods to calculate metrics of accuracy and f1-score |

# 5 Preprocessing

## 5.1 Dataset characteristics

In this project we used a fraud dataset which contains 434 features and 540484 samples. It contains feature with different types including integer, float, Boolean and categorical values. Many attributes contain a significant percentage of missing value. A quick data overview shows that there are 32 categorical value that may cause problem for some predicting models and we have to deal with them. Figure 1 shows the number of features with numerical, categorical and string type. As it is obvious, most of features are numerical still we cannot neglect categorical values.

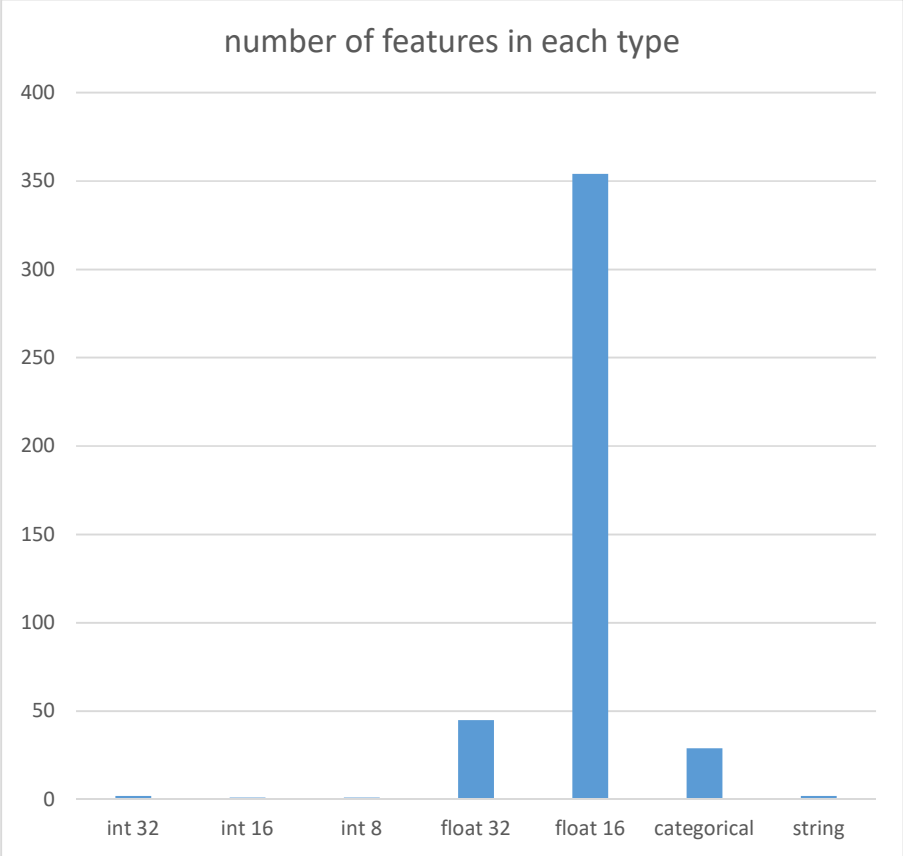


*Figure 1: number of features for with different types including integer, float, categorical and string*

Figure 2 shows the number of features with determined percentage of missing values. There are 133 features with 71% to 80% missing values and 12 features with more than 90 percent missing values. Among all 434 features only 20 features don't contain any missing value.

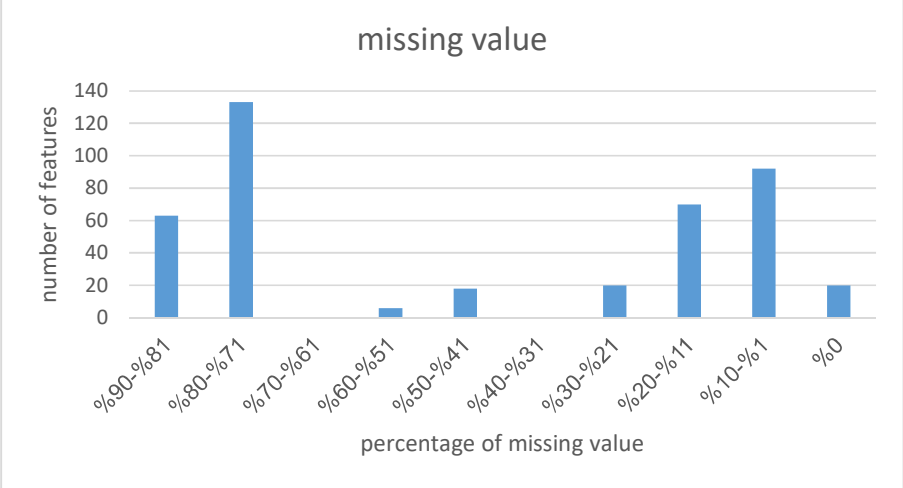


*Figure 2: number of features with specified percentage of missing values*

Moreover, the dataset is not normalized. Some attributes have greater range of numeric values such as D8, D15 and V127. Therefore, normalization is required to be sure that smaller sized values are not neglected. Another problem that can be consider in this dataset is the imbalanced number of samples available for fraud and legal transaction. As it is shown in Figure 3 only 3.5 percent of samples are coming from fraud class and about 96.5 percent of them are legal transactions.

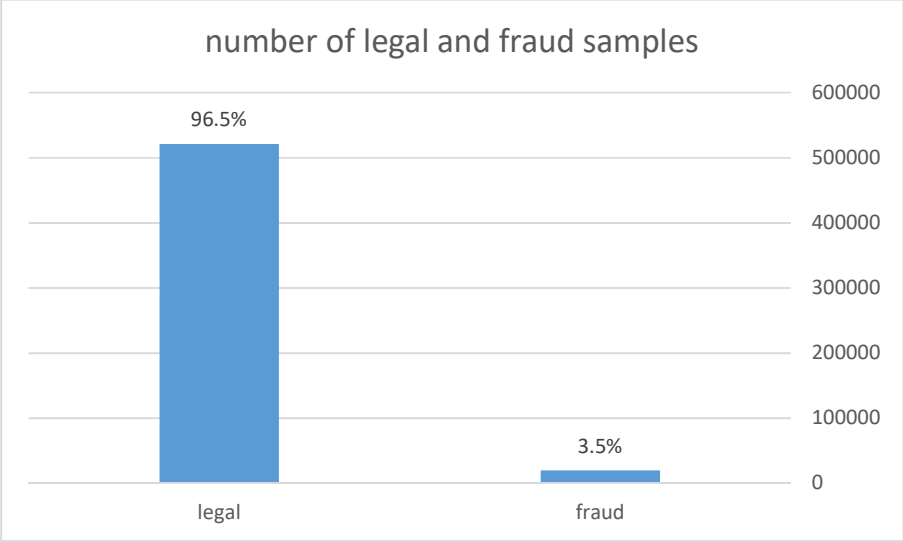


*Figure 3: number of existing legal and fraud samples in dataset*

Finally, large number of features can cause some efficiency issues in learning algorithm. Thus to get better results we need to use a couple of feature selection methods.in this project we used a couple of filter based feature selection. As the name suggest, in this method, you filter and take only the subset of the relevant features. The model is built after selecting the features. We design the following 8 steps for cleaning and preprocessing the dataset.

1. Manual feature selection
2. Handling of missing value
3. Dialing with categorical values
4. Deal with imbalanced dataset
5. Removing constant and quasi constant features
6. Normalization
7. Removing highly correlated features
8. Feature Selection based on chi2 criterion

In the remaining of this section we will discuss each step of this flowchart, and their required parameter tuning.

## 5.2 Manual feature selection

As the first step of our preprocessing we manually removed DeviceInfo. The main reason was that this feature was in the form of string and although it can be considered as a categorical feature, in its current form it cannot be useful for our classification methods. extracting useful information from this feature (even if it is possible) needs using natural language processing methods which is beyond this project.

## 5.3 Handling of missing value

There are several possible ways to deal with missing data:

- Encoding nulls with -1 or -9999: works reasonably well for numerical features that are predominantly positive in value. But we have also negative value in our dataset.
- Case wise deletion of missing values: This method cannot work out for our dataset because number of samples with missing value are large and can seriously skew the predictive model.
- Run predictive models that impute the missing data.

- Replace missing values with mean or median value of feature in which they occur. We used this method to handle missing values in numerical features. In fact, we replaced Nulls in each integer variable with its median value and in each float variables with its mean value.

```
float_features = dataset.select_dtypes(include=['float64','float32','float16'])
for f in float_features:
    if dataset[f].isna().any():
        dataset[f].fillna(dataset[f].mean(),inplace=True)

int_features = dataset.select_dtypes(include=['int64','int32','int16','int8'])
for i in float_features:
    if dataset[i].isna().any():
        dataset[i].fillna(dataset[i].median(),inplace=True)
```

*Figure 4: part of python code that handle missing values in numerical features*

- Label encode NULLs as another level of categorical variables. We applied this method to remove Null values in 32 categorical features. We simply replaced Null values with "undefined" value.

```
categorical_features = dataset.select_dtypes(include=[object])
for c in categorical_features:
    if dataset[c].isna().any():
        dataset[c].fillna('undefined',inplace=True)
```

*Figure 5: part of python code that handle missing values in numerical features*

## 5.4 Converting categorical values

Categorical values should be converted into a form that could be provided to classification algorithms to do a better job in prediction. There are two main solutions for this problem.

- **Label encoder**: It is used to transform non-numerical labels to numerical labels. Numerical labels are always between 0 and ($N_{class}$-1) for example in the dataset for card4 feature we have for categories: {'debit', 'credit', 'charged cart', 'debit or credit'} that can be replace respectively by {0,1,2,3}

  A common challenge for this method is that replacing categories with numbers can make this categorical feature look like or correlated with another existing feature. Moreover, we are imposing ordering to this feature that is not inherently one of its characteristics.
- **Dummy coding**: This is a commonly used method for converting a categorical variable into continuous variable. A dummy variable is a duplicate variable which represents one level of categorical variable. Presence of a level is represented by 1 and absence is represented by 0. For each value one dummy variable will be created. We can use One Hot Encoder (from sklearn) or get_dummies (from pandas) in Python to implement dummy coding. In this project we used get_dummies. To avoid creating redundant variables we removed one of the dummy variables created for each categorical features. For example, for card4 three dummy variable is enough to cover all of its information. After applying this method, number of features in dataset will increase from 434 to 920 attributes**.**

## 5.5 Deal with imbalanced dataset

Most of classification methods do not work very well with imbalanced datasets. There are several techniques that can help us to train a classifier to detect abnormal classes. Among all existing solutions, we applied two of the most common methods:

1) **Use the right evaluation metrics**

   Applying inappropriate evaluation metrics for imbalanced dataset can lead to false results. For example, in our dataset accuracy is not a good metric because if we classify all samples as legal transaction will have an excellent accuracy 96.5%, but obviously this model won't provide a valuable information for us. Alternative evaluation metrics that are useful for this dataset are precision, recall, F1 score, MCC and AUC. We used F1-score to evaluate our model.

2) **Resample the training dataset**

   Two approaches to make a balanced dataset out of an imbalanced one are under-sampling and over-sampling.

   - Under-sampling: In this case we should balance the dataset by reducing the size of the legal class (as our abundant class). To this aim we reduced the size of our legal class to

   - Over-sampling: It tries to retrieve a balanced dataset for future modelling by increasing the size of rare fraud samples. New samples can be generated using repetition, bootstrapping, or SMOTE (Synthetic minority over sampling technique). Because there is no absolute advantage of one resampling method over another we tried repetition as our over-sampling method.

We examined different size for our final balanced dataset including {500000, 200000, 100000, 60000, 40000} samples. Best results were obtained for a dataset containing 60000 samples {30000 samples of legal transactions and 30000 samples of fraud transaction}. To create such dataset, we used under-sampling for legal class and over sampling with repletion for fraud class.

## 5.6 Removing constant and quasi constant features

These are features that are almost constant or completely constant. In fact, they have same values for large subset of dataset. Such features are not useful for making predictions. There is

no rule as to what be the threshold for the variance of quasi constant features. However, as a rule of thumb, we can remove those quasi constant features that have more than 99% similar values. To achieve this aim, we will create a quasi-constant filter with the help of VarianceThreshold function in python. we will pass 0.01, which means that if the variance of the values in a column is less than 0.01, remove that column. In other words, remove feature column where approximately 99% of the values are similar. Using this method, we simply removed 461 attributes from all 920 existing features and reduce the size of them to 459.

## 5.7 Normalization

Data normalization is one of the main strategies that needs to be employed before performing classification methods. The basic purpose of data normalization is to keep check on the attributes having greater range of numeric values (such as D8, D15 and V127 in our dataset), so that the smaller sized values (such as V28,V29,V30) are not neglected. To achieve this aim we used normalization function as mentioned below:

$$X' = a + \frac{(X - X_{min})(b - a)}{(X_{max} - X_{min})}$$

Where X the data value such that $X \in D$, and D is the domain of X. X' represents the data value after normalization. $X_{max}$ and $X_{min}$ are respectively the maximum and minimum data values in D. a and b are the minimum and maximum values in the specified output range. In the proposed approach, all the data values are scaled down in the range [1,0]

## 5.8 Removing highly correlated features

Correlation between the output observations and the input features is very important and such features should be retained. However, if two or more than two features are mutually correlated, they convey redundant information to the model and hence only one of the correlated features should be retained to reduce the number of features.

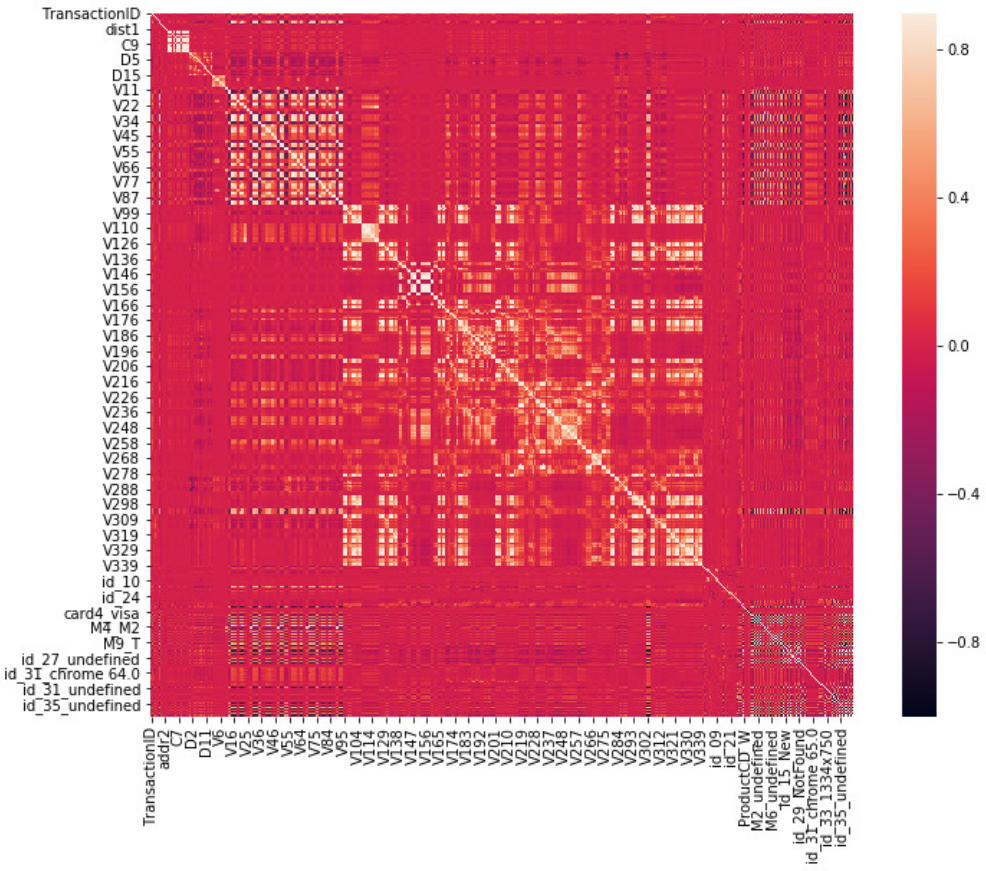


*Figure 6: heat-map of correlation matrix*

Therefore, in next step we performed preliminary feature selection on the basis of correlation matrix. Some basic takeaways from the correlation matrix are as follows:

1. If we look at the heat-map in Figure 6 , all white squares indicate high correlation between the corresponding variables
2. We can find highly correlated features among cx (c1-c9) attributes (the small white square at the left top of the map) to clarify this, we represent the value of correlation matrix for some of these variables in Figure 7.
3. Furthermore, white large square in the middle of heat-map shows that there are several highly correlated features in V*x* (V1.. V339) variables.
4. Another set of variables that show high correlation are some of variables that created by dummy encoder and specially those variables that contains missing values (dummy variables with "_undefined" suffix). Which implies that for some features have NULL value in same samples.

| C1 | C2 | C3 | C4 | C5 | C6 | C7 |
|---|---|---|---|---|---|---|
| 1 | 0.994063 | -0.00244481 | 0.977609 | 0.0847097 | 0.980558 | 0.93781 |
| 0.994063 | 1 | -0.0025919 | 0.969522 | 0.0617955 | 0.968558 | 0.93701 |
| -0.00244481 | -0.0025919 | 1 | -0.00169985 | -0.00397743 | -0.00268697 | -0.0015454 |
| 0.977609 | 0.969522 | -0.00169985 | 1 | -0.0144068 | 0.987371 | 0.889875 |
| 0.0847097 | 0.0617955 | -0.00397743 | -0.0144068 | 1 | 0.12868 | -0.0114992 |
| 0.980558 | 0.968558 | -0.00268697 | 0.987371 | 0.12868 | 1 | 0.874991 |
| 0.93781 | 0.93701 | -0.0015454 | 0.889875 | -0.0114992 | 0.874991 | 1 |
| 0.979802 | 0.974783 | -0.00136288 | 0.955513 | -0.0139701 | 0.94091 | 0.982216 |
| 0.0854117 | 0.0606381 | -0.00541763 | -0.0196234 | 0.926698 | 0.134463 | -0.0156629 |
| 0.973395 | 0.968875 | -0.00144746 | 0.952032 | -0.0126977 | 0.937588 | 0.985107 |
| 0.995315 | 0.987323 | -0.00237932 | 0.986645 | 0.0902103 | 0.991305 | 0.924329 |
| 0.939719 | 0.940473 | -0.00171216 | 0.888191 | -0.0125553 | 0.873127 | 0.998311 |
| 0.867968 | 0.846846 | -0.00308353 | 0.814911 | 0.520211 | 0.883608 | 0.794044 |
| 0.957059 | 0.937706 | -0.00231567 | 0.964447 | 0.223012 | 0.991354 | 0.835583 |
| -0.016385 | -0.0178195 | 0.0331191 | -0.0312734 | 0.131069 | -0.00812196 | -0.0227251 |

*Figure 7: a small part of correlation matrix obtained for attributes*

Finally, we dropped among the set of variables with more than 0.9 positive or negative correlation we kept one of them and removed the other.

## 5.9 Feature selection based on chi2 criterion

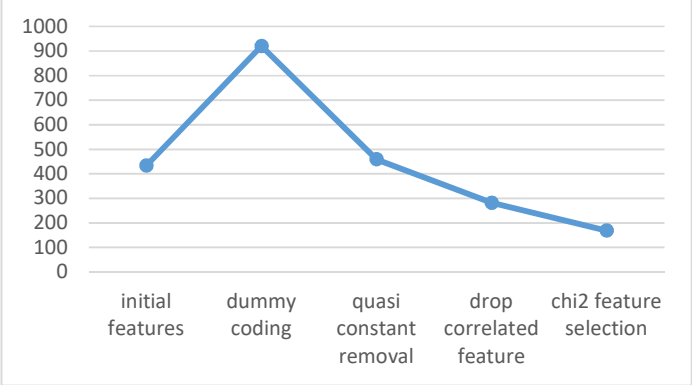


*Figure 8:number of features after each step of preprocessing*

The Chi-square statistic is a non-parametric (distribution free) tool designed to analyze group differences when the dependent variable is measured at a nominal level. Because the Chi-square is robust with respect to the distribution of the data, it is a good choice for feature selection criteria. Specifically, it does not require equality of variances among the study groups

or homoscedasticity in the data. We used SelectPercentile(chi2,60) to select 60 percent of remaining feature based on chi2 score. Figure 8 shows the number of features after applying each step of data preprocessing.

# 6 Classification

## 6.1 Logistic

One of the most common utilized linear statistical models for discriminant analysis is logistic regression. Logistic regression algorithm also uses a linear equation with independent predictors to predict a value. The predicted value can be anywhere between negative infinity to positive infinity. We need the output of the algorithm to be class variable, i.e 0-no, 1-yes. Therefore, we are squashing the output of the linear equation into a range of [0,1]. To squash the predicted value between 0 and 1, we use the sigmoid function.

- **Choosing model parameter**: most important parameter in logistic regression which need to be set are determining penalty function and determining maximum number of iteration in learning process. For penalty function we examined the accuracy obtained by "l1" and "l2". l2 create higher f1 score. Therefore, in next section we report results based on penalty = 'l1'. The value of maximum iteration would be 100, because higher value of iteration won't improve the obtained result s.

## 6.2 SVM

SVM is a supervised machine learning algorithm which can be used for problems. It uses a technique called the kernel trick to transform data and then based on these transformations it finds an optimal boundary between the possible outputs.

- **Choosing appropriate kernels:** choosing kernels according to the problem is also very important task in SVM classification. We should choose the best available kernel that fits the data. In this project, we test one linear and one non-linear kernel for classification. Gaussian Radial Basis Function (RBF) kernel is the most commonly non-linear kernel used in different application. [8] This kernel is based on Gaussian function that can be written as:

$$K(x_i, x_j) = \exp(-\frac{\|x_i - x_j\|^2}{2\sigma^2})$$

Where $x_i$ is the support vector and $x_i$ is the current data value. Another representation of RBF kernel is:

$$K(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right) for\ \gamma > 0$$

The primary objective of this function is making decision boundary that classifies the training set into two classes the mathematical representation of this decision boundary is as follows[8]:

$$f(x) = \boldsymbol{w} \cdot k(\boldsymbol{x}_i, x_j) + b,\ \boldsymbol{w} \in \boldsymbol{x}_i, b \in \Re^m$$

This decision function can be represented as an optimization function which is given by

$$\min \left( \underbrace{\frac{1}{2}||\boldsymbol{w}||^2}_{\text{Max. margin}} + \underbrace{C \sum_{i=1}^{n} \xi_i}_{\text{Min. error}} \right)$$

Where C is the additional regularization parameter used to tradeoff between maximization of margin and minimization of training error.

- **Choosing model parameter**: For RBF kernel, two modeling parameters namely C and $\gamma$ need to be chosen. These parameters have a direct effect on the accuracy of the SVM classifier. Thus selecting the right values of these parameters is an important task. Here we used python implementation of libsvm[9]. LIBSVM is used for optimizing these parameters based on cross validation and grid search. The detail can be found in [10] Moreover, we have to determine the degree of kernel, we examined two different value for degree (2,3) and results had not significant difference. Results in next section are shown based on degree=2

## 6.3 K-nearest neighbor

Another supervised learning algorithm that we want to use in this project is the **k-nearest neighbor classifier** (**KNN**), which is chosen because it is fundamentally different from the other selected learning algorithms.

KNN is a typical example of a **lazy learner**. It is called lazy not because of its apparent simplicity, but because it doesn't learn a discriminative function from the training data but

memorizes the training dataset instead. Some of most important KNN classifiers that are good to know in this project are:

- k-NN performs much better if all of the data have the same scale
- k-NN works well with a small number of input variables (p), but struggles when the number of inputs is very large
- k-NN makes no assumptions about the functional form of the problem being solved

- **Choosing model parameter**: most important parameter in kNN classifier is k (number of neighbors that should be considered in the process of learning). We test different value for K, and k=1 turn out to be the best value. It is interesting to keep in mind that K=1 means that bios is low and variance is high. So the model is not very robust.

### 6.4 Random forest

Random forest is an ensemble-based learning algorithm which is comprised of n collections of de-correlated decision trees. [11] when we do not have much time to preprocess the data (and, or have a mix of categorical and numerical features), the random forest works well. Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction

Main parameters of this classifier can be listed as follow:

- **Criterion**: which can be gini or entropy, we selected gini
- **max_depth**: The maximum depth of the tree. We select Null so nodes are expanded until all leaves are pure or until all leaves contain less than 2 samples
- **bootstrap**: Determines whether bootstrap samples are used when building trees. If it is False, the whole dataset is used to build each tree. We set this variable to True.

## 7 Results

### 7.1 Random Forest Classifier

As shown in Figure 9 and Figure 10 fit time varies from 11.25 to 13 seconds and score time for different fold is between 0.16 to 2 seconds which show that score time is significantly lower than fit time for this classifier.

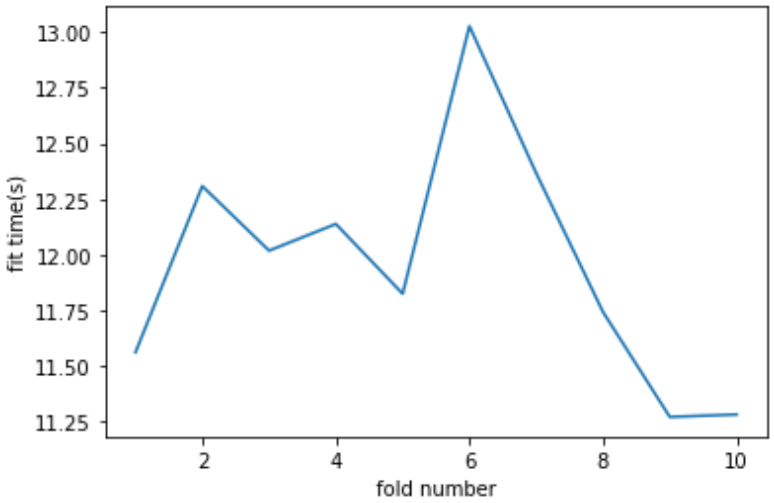


*Figure 9: fit time for 10 fold cross validation in random forest classifier*

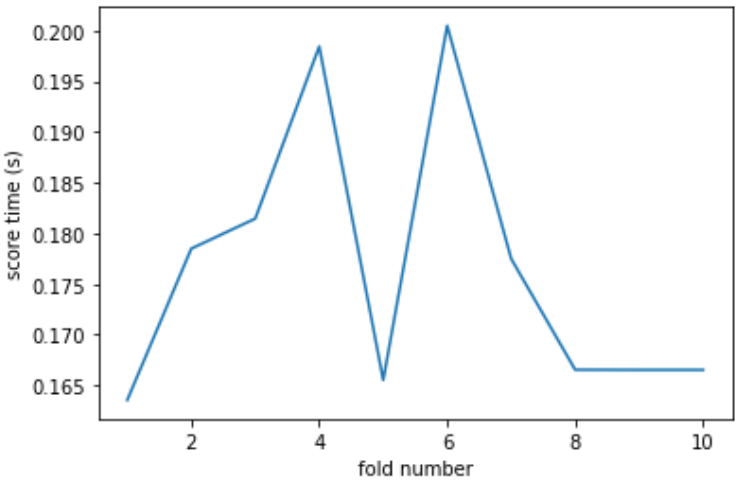


*Figure 10: score time for 10 fold cross validation in random forest classifier*

Figure 11 represent f1-score value of 10 fold in cross validation analysis of random forest. All f1 score values are higher than 0.89 and less than 0.95 (average is 0.91) this is the best result we obtained among all examined classifiers (SVM, kNN and LR).

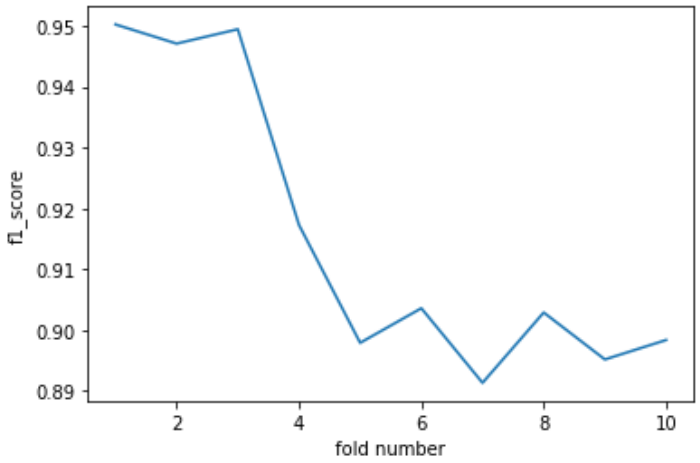


*Figure 11: f1 score for each 10 fold cross validation in random forest classifier*

In random forest model we also can measure the importance of different feature in modeling. Figure 12 shows importance of different 168 attributes. (which are numbered based on their order in the dataset) As it obvious, three features have significantly higher value than others. These features are Card1, C1, Card5. which means that these features play more important role in determining the class of each sample.

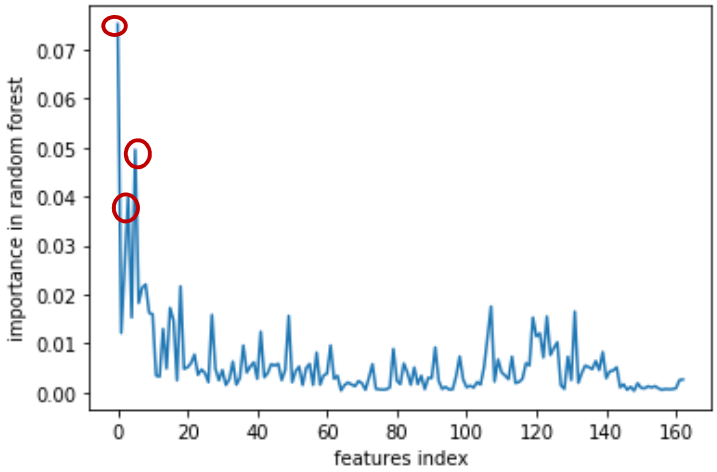


*Figure 12:importance of feature in random forest*

## 7.2 kNN Classifier

Figure 13 and Figure 14 respectively represent the fitting time of the model and scoring time for each fold. As shown below, fitting time is bounded between 1.85 to 3.65 second but scoring time is much higher and goes up from 21 to 31 second. In fact, kNN perform very well when we have many instances and few dimensions. But brute force version of kNN can be very slow when data is large.

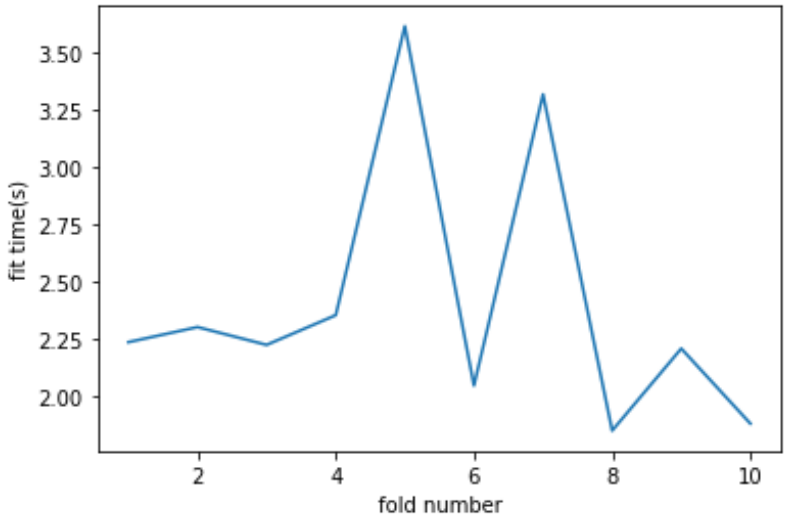


*Figure 13: fit time for 10 fold cross validation in K nearest neighbor classifier (K=1)*

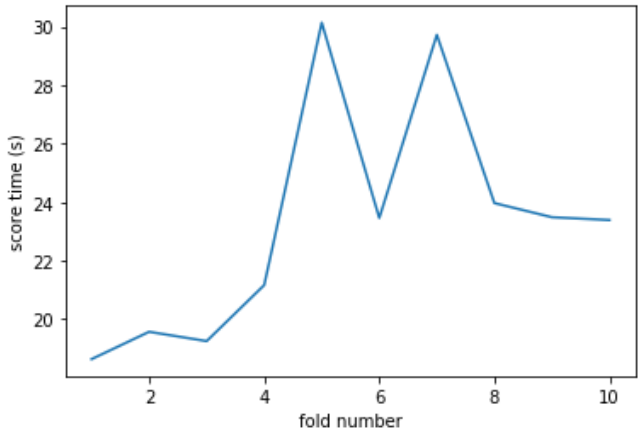


*Figure 14: score time for 10 fold cross validation in K nearest neighbor classifier (K=1)*

Figure 15 shows obtained F1-score for each fold of cross validation. As shown here all results are between 0.84 and 0.92. the average value is 0.87 on balanced dataset.

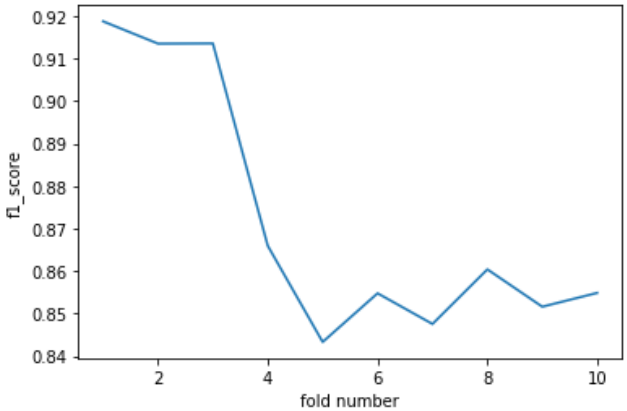


*Figure 15: f1 score  for each 10 fold cross validation in K nearest neighbor classifier (K=1)*

To select best number of neighbors in this algorithm we examined different values for K and obtained average f1_score in 10-fold cross validation. As shown in Figure 16 N=1 get better f1-score on our balanced dataset.

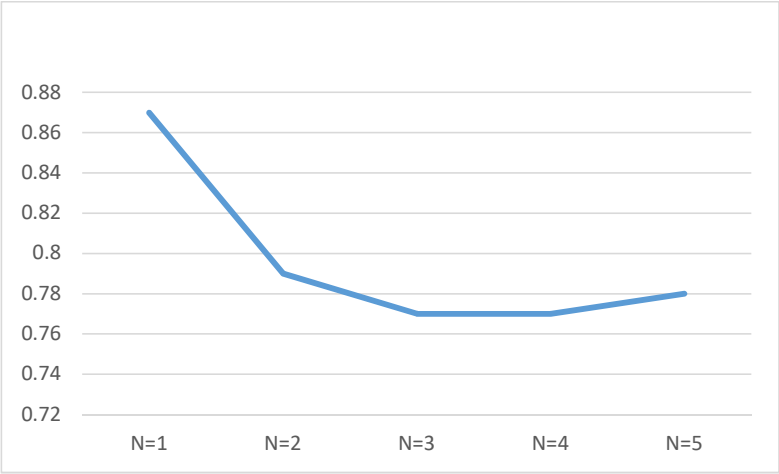


*Figure 16: average of f1 score in KNN classifier for different value of K (K=1 .. K=5)*

## 7.3 Logistic Classifier:

In Logit Classifier fit time is clearly higher than score time. In fact, its fit time take longer than random forest and kNN but its score time is minimum among 4 comparing classifiers. Figure 17 and Figure 18 show respectively the fit time and score time for each fold in cross validation analysis

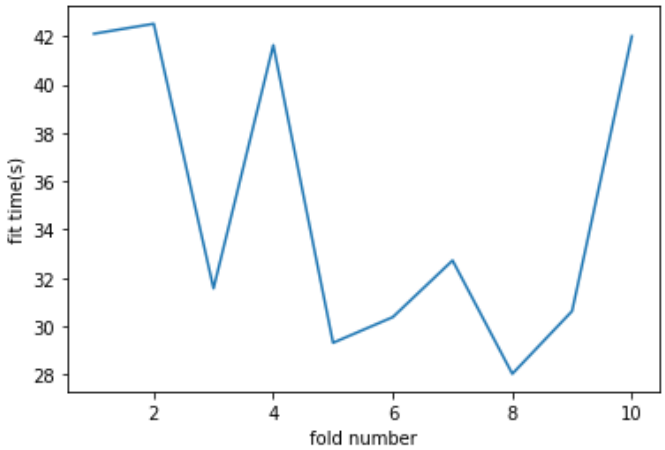


*Figure 17: fit time for 10 fold cross validation in logistic classifier (using l1 as penalty function)*

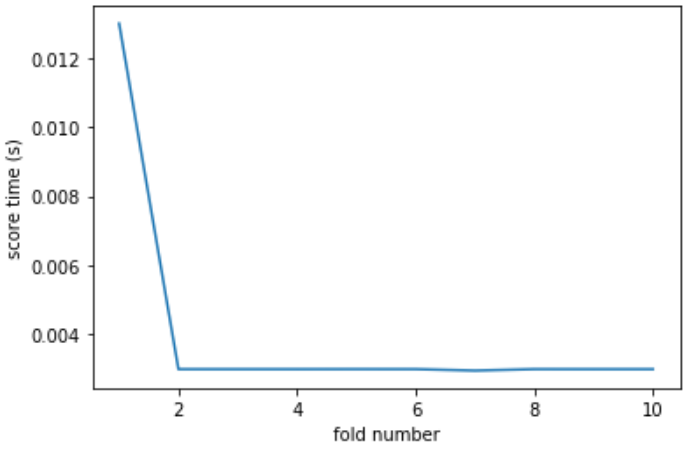


*Figure 18: score time for 10 fold cross validation in logistic classifier (using l1 as penalty function)*

Logistic classifier supports only linear solutions and for non-linear data would not find an accurate model. This is shown in Figure 19 which indicate that f1 score in LR classifier is lower than kNN and random forest classifiers

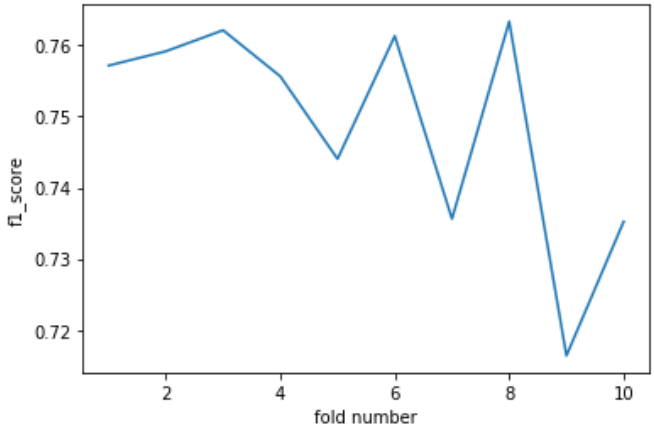


*Figure 19: f1 score for each 10 fold cross validation in logistic classifier (l1 as penalty function)*

## 7.4 SVM Classifier

Unfortunately, SVM has a high computational complexity and its response time for 169 feature is too high to be obtainable. Therefore, we were forced to remove more feature to execute SVM in an acceptable time. We used chi2 and select 30 best features based on this criterion. The results shown below derived from these 30 features. So we should keep in mind that comparing time and f1-score of SVM with three other classifiers is not a fair comparison. Figure 20 and Figure 21 represent fit time and score time for 5 fold cross validation analysis. As we can see, despite decreasing number of features, SVM still has the highest fitting time among other classifiers.

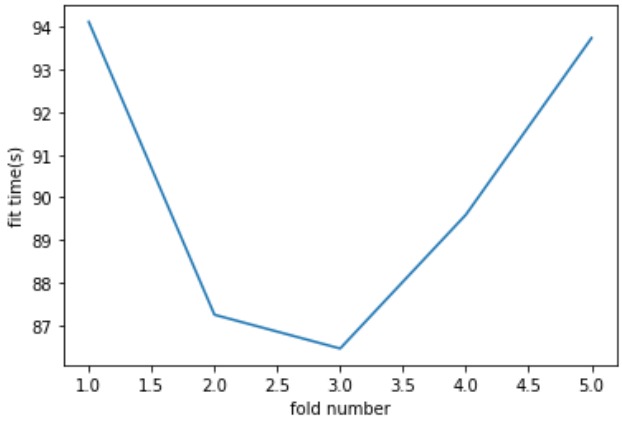


*Figure 20: fit time for 5 fold cross validation in SVM classifier (RBF kernel , degree=2)*

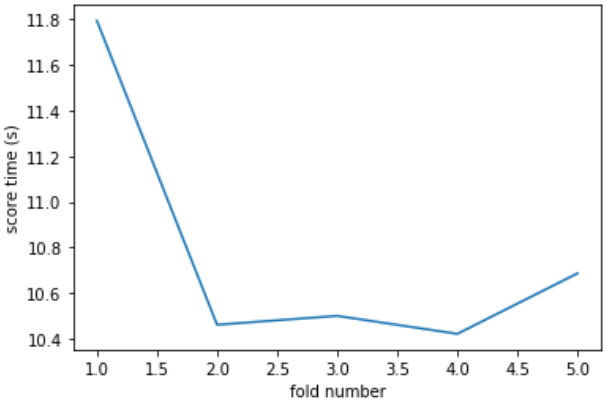


*Figure 21: score time for 5 fold cross validation in SVM classifier (RBF kernel , degree=2)*

Finally, Figure 22 shows the f-score for 5 fold of cross validation in SVM classifier. The average value is 0.69, although it is lower than other classifier's f1 score, we should not forget that this result obtained by fewer features which can effect on precision and recall.

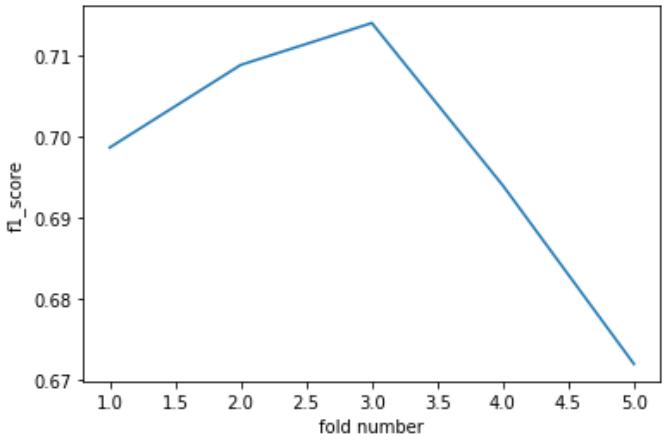


*Figure 22: f-score for 5 fold cross validation in SVM classifier (RBF kernel , degree=2)*

## 7.5 Comparative analysis

Table 1 shows average result obtained for four selected method (kNN, SVM, Random forest and logit). As we explained in previous section, comparing time and f1-score of SVM with three other classifiers is not a fair comparison. Because this model is obtained with fewer number of attributes. However, SVM still has the highest fitting time among other classifiers. In fact, SVM is more reliable but kNN is less computationally intensive than SVM.

KNN is comparatively slower than logistic regression. But kNN support non-linear solution where LR supports only linear solutions. So kNN end up with a higher accuracy and f1-score kNN has the minimum fit time and Logit has the minimum score time. Random forest as expected has the maximum f1-score and accuracy. Therefore for fraud_test we used random forest as our final model

*Table 1: comparing results of four classifier*

| | KNN | Logistic | Random forest | SVM |
|---|---|---|---|---|
| Average of f1 score | 0.87 | 0.75 | **0.91** | 0.69 |
| Average of accuracy | 0.85 | 0.77 | **0.93** | 0.65 |
| Average of fit time | **2.40(s)** | 35.08 | 12 | 90.2 |
| Average of score time | 23.3(s) | **0.004** | 0.17 | 10.77 |

## 7.6 Conclusions and future work

We used SVM, kNN, Logistic and Random forest for classification of fraud and legal transactions. Among all selected classifier random forest turn out to be most accurate, kNN is fastest model in fitting time and Logistic has the minimum score time. There are plenty of published papers that work on modified version of SVM (such as Bayesian Nonlinear Support Vector Machines), neural network and deep learning for fraud detection. Unfortunately, time limitation prevented us from implementing these promising methods. For future work analyzing and comparing Neural network based method such as deep learning, and better version of SVM is recommended. Moreover, the feature selection part has many room for innovation and improvement.

# 8 References


1. Examiners, A.o.C.F. The Staggering Cost of Fraud. 2016; Available from: http://www.acfe.com/rttn2016/docs/Staggering-Cost-ofFraud-infographic.pdf

2. Sorkun, M. C., & Toraman, T. (2017). Fraud Detection on Financial Statements Using Data Mining Techniques. *International Journal of Intelligent Systems and Applications in Engineering*, *5*(3), 132-134.

3. Phua, C., Lee, V., Smith, K., & Gayler, R. (2010). A comprehensive survey of data mining-based fraud detection research. *arXiv preprint arXiv:1009.6119*.

4. Ngai, E. W., Hu, Y., Wong, Y. H., Chen, Y., & Sun, X. (2011). The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature. *Decision support systems*, *50*(3), 559-569.

5. Bhattacharyya, S., Jha, S., Tharakunnel, K., & Westland, J. C. (2011). Data mining for credit card fraud: A comparative study. *Decision Support Systems*, *50*(3), 602-613.

6. Pai, P. F., Hsu, M. F., & Wang, M. C. (2011). A support vector machine-based model for detecting top management fraud. *Knowledge-Based Systems*, *24*(2), 314-321.

7. Kirkos, E., Spathis, C., & Manolopoulos, Y. (2007). Data mining techniques for the detection of fraudulent financial statements. *Expert systems with applications*, *32*(4), 995-1003.

8. Jindal, A., Dua, A., Kaur, K., Singh, M., Kumar, N., & Mishra, S. (2016). Decision tree and SVM-based data analytics for theft detection in smart grid. *IEEE Transactions on Industrial Informatics*, *12*(3), 1005-1016.

9. Chang, C. C., & Lin, C. J. (2011). LIBSVM: A library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, *2*(3), 27.

10. Hsu, C. W., Chang, C. C., & Lin, C. J. (2003). A practical guide to support vector classification.

11. Hastie, T., Tibshirani, R., Friedman, J., & Franklin, J. (2005). The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer*, *27*(2), 83-85.