ECM3420 Coursework Assessment

690019495

November 2021

1 A: Main objective

1.1 Credit Card Fraud Detection

The dataset that is used:
`https://www.kaggle.com/mlg-ulb/creditcardfraud`

Cybersecurity is becoming a crucial part of life, and ensuring that online transactions are safe is important. Credit card fraud is when an unauthorised person uses another person's credit card or account information to make purchases or access funds. By correctly recognizing potentially fraudulent uses of credit cards, this can be avoided. This dataset is a collection of many transactions, where a few are fraudulent.

Credit card fraud detection aims to decide whether or not a transaction is fraudulent on the basis of historical data [2,3]. The decision is notoriously difficult because of changes of a users spending behavior [2,3]. It is known that machine learning techniques offer an effective approach to tackling problems like these [3].

1.2 Problem statement

To tackle this problem there needs to be a system that can track the pattern of multiple transactions, and if any pattern is abnormal then the transaction should be aborted, assuming its fraud. Therefore, machine learning models will be focused on classification of fraudulent and non fraudulent transactions [2,3,4]. These models will then be used to identify whether a new transaction is fraudulent or not. The hypothesis: can the models detect 100% of the fraudulent transactions while minimizing the incorrect fraud classifications.

1.3 Observations

Fraudulent transactions occur only a small amount in real world analytical data. The data consist of 284,807 total observations. In general, the aim is to maximize the recall while capping the false negative rate. Due to the large class of true negatives, the model could classify the wrong transaction type and still maintain a high false negative rate. This is conducive to picking a relatively low threshold, which results in the high recall but extremely low precision.

The dataset consists of numerical values from the 28 'Principal Component Analysis (PCA)' transformed features, namely V1 to V28 [4]. Furthermore, there is no metadata about the original features provided, so pre-analysis could not be done. Since all features are anonymous, the 'Time' and 'Amount' features are the only none transformed data, and therefore will focus the analysis on non-anonymous features.

1.4 Structure

The structure of this analysis and implementation is as follows: Section B will be an in depth description of the dataset and a summary of its attributes. Section C will be a brief summary of data exploration and actions taken for data cleaning and feature engineering. Section D will be the results of training two machine learning models chosen, and leading to section E where the explanation of the final model that is best in terms of accuracy. Section F will be the key finding summary and insights of the data derived from the chosen models. Section G will be a conclusion and further suggestions for the next steps in analysing the dataset.

2 B: Data Analysis

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days:

492 fraudulent out of 284,807 transactions. The dataset is highly unbalanced where the positive class (frauds) account for 0.172% of all transactions.
This skewed dataset could potentially cause problems in the training for the models.

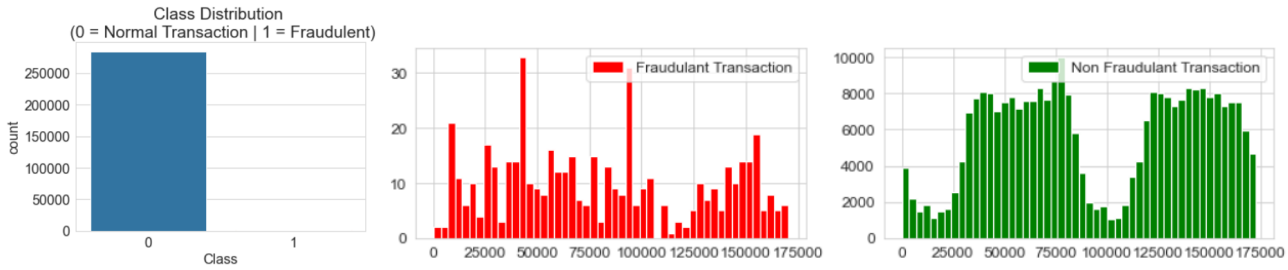


Figure 1: Transactions distributed over time, where there is significantly less fraud in comparison to the amount of transactions

It contains only numerical variable inputs which are the result of a PCA transformation [4]. Due to confidentiality the original features and more background information about the data are not included. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with the PCA are the variables 'Time' and 'Amount'.
'Time'- contains the seconds elapsed between each transaction and the first transaction in the dataset.
'Amount' - transaction Amount; this feature can be used for example-dependant cost-sensitive learning.
'Class' - is the response variable and it takes. Value 1 = Fraud and 0 = Not fraud.

```
print(data.columns)
print("Null:  ", data.isnull().sum().sum())
>>> Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20', 'V21',
'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount','Class'],
dtype='object')
>>> Null:  0
```

There is no missing data from the dataset file, all rows and columns have the relevant data. Most of the transactions are non-fraud. Using this dataframe as the base for the predictive models and analysis will most likely get errors due to the algorithms predicting an overfit of non fraud since it will assume that most transactions are not fraud. This is not ideal, and instead the models should to detect patterns that give signs of fraud.

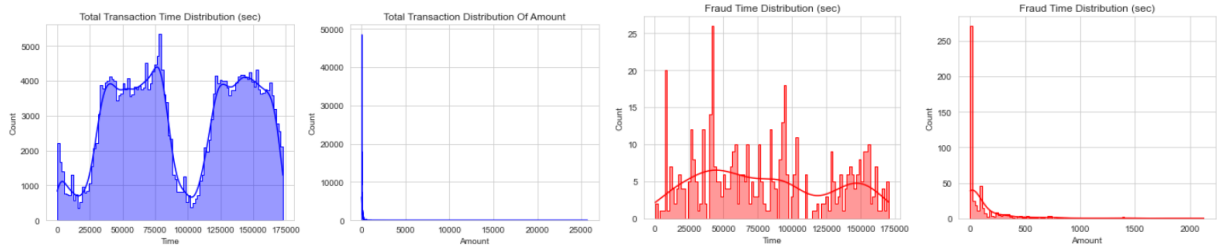


Figure 2: Distribution of Time and Amount

Looking at the distribution of the amount in regards to time: do fraudulent transactions occur more often during a certain time frame?

In figure 2, it is observed that there are two large time frames where the majority of the transactions occurred. The fraudulent transactions occurred at seemingly random time intervals hidden within the other transactions. The time and amount has seemingly no correlation, but everyday people have a certain regularity in the way a payment is made. For example, someone visits a certain shop once a week at around the same time and always spends about €40 to €60. If the same account is used to make a payment sum of more than €60 or at a location that has never been used before, this behavior would be considered irregular. From the dataset, the time in-between payments and the amount will affect the classification.

Metrics such as standard deviation, averages, and high/low values are the most useful to spot irregular behavior. The values that are available are the Time and Amount features that will be used by the models. The slight time/amount variations and outliers for each transaction can be picked up by computer models where humans see no patterns. Classification machine learning models learn from prior data and estimate the probability of a new transaction for whether it is a fraudulent credit card transaction.

Before the classification computer models can be implemented, the extent of the skewed distributions for the amount of transactions and the features needs to be addressed. It is important to implement techniques that can help balance the data.

3 C: Data prepossessing

Data preprocessing is necessary because it helps to gain a better accuracy rather than just using raw data to build a model. The PCA transformation is effected by scale, therefore the features in the data need to be scaled, using Scikit-learn's StandardScaler() [1]. In order to fit to the scaler the data should be reshaped within -1 and 1. All the features are standardize into unit scale (mean = 0 and variance = 1)

```
data['Vamount']=StandardScaler().fit_transform(df['Amount'].values.reshape(-
1,1))
data['Vtime']=StandardScaler().fit_transform(df['Time'].values.reshape(-1,1))
```

Initially, an equal amount of fraudulent and non fraudulent transactions will be used for the model comparison. Further analysis will be implemented by increasing the number of non fraud to compares how the models perform with the larger data.

4 D: Machine learning models

Predictive classification models will be used for this dataset. The models are:

- Logistic Regression
- K Nearest Neighbours

Both proposed techniques will be tested using equal weights of fraud and non fraud transactions. Then the selected best model will be tested further using undersampling techniques. Theses applications of methods for data balancing are widely used in these dataset classification cases. Changing the sampling makes the algorithm more "sensitive" to fraudulent transactions. Undersampling removes the major class. In this case, it is necessary to remove random records from the legitimate class (No fraud), in order to obtain a number of records close to the amount of the minority class (fraud) in order to train the model.

For this test, equal amounts of fraud and non fraud will be used. Firstly, the models will be compared using equal weights. Each model will explore the best possible techniques to improve the results. Each model will use the same training data for fair comparison.

```
frauds = data[data['Class'] == 1]
non_frauds = data[data['Class'] == 0][:500]
# take 500 to roughly match the amount of fraud

new_df = pd.concat([non_frauds, frauds])
# Shuffle dataframe rows
new_df = new_df.sample(frac=1, random_state=42)
```

This reduces the unbalanced class (0) in the data. The data then has to be split into equal training and testing sets. As stated above the training size will be 80% and the test size of 20%. This is chosen to give the models more data for training to increase the accuracy. The models need to be tested for the classification prediction. A false positive is a type 1 error in binary classification in which a test result incorrectly indicates the presence of a condition, while a false negative is a type 2 error where the test result incorrectly indicates the absence of a condition when it is actually present.

4.1 Logistic Regression

Logistic Regression is commonly used to estimate the probabilities that on instance belongs to a particular class. If the probability is more than 50%, then the model predicts that the instance belongs to that class otherwise it does not belong to that class. It is a binary classifier, which works well for this dataset as a transaction is either fraud (1) or not fraud (0).

The results from Figure 3 using the training data show that 2 non fraud transaction was identified as fraud (false positive) and 9 fraudulent transactions were predicted as not fraud (false negative). The accuracy overall was 94% with a total of 11 miss-prediction.

```
Accuracy: 0.9447236180904522
False negative rate (with respect to misclassifications):  0.8181818181818182
False negative rate (with respect to all the data):  0.04522613065326633
False negatives: 9
False positives: 2
Mispredictions: 11
Total test data points: 199

Logistic regression
Scores
Accuracy --> 0.9447236180904522
Precison --> 0.9789473684210527
Recall --> 0.9117647058823529
F1 --> 0.9441624365482234
```

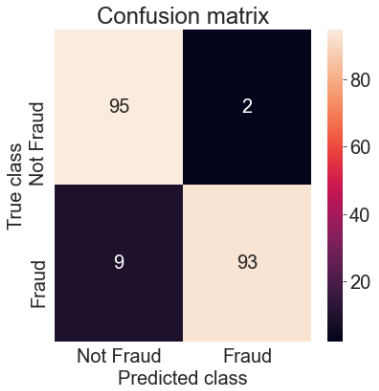


Figure 3: Results of test using Logistic Regression

For 97 non fraudulent transactions in the test, 2% of the transactions that were non fraudulent have been misclassified as fraud. For 102 fraudulent transactions in the test, 8.8% of the transactions that were fraudulent have been misclassified. This is a low number, however too many type 2 errors is not advisable, especially for further tests in a larger volume of transactions.

A False Negative is more dangerous than a False Positive, as not flagging fraudulent transactions causes more harm than accidentally flagging a real transaction. Hence, in order to increase the sensitivity, the threshold will be lowered.

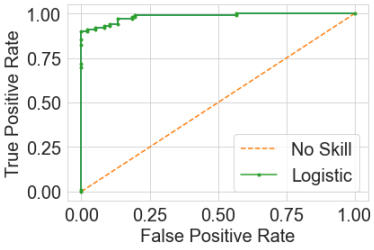


Figure 4: ROC curve

4.1.1 ROC curve

The ROC curve is a diagnostic plot that evaluates a set of probability predictions made by a model on a test dataset.

A set of different thresholds are used to interpret the true positive rate and the false positive rate of the predictions on the positive (minority) class, and the scores are plotted in a line of increasing thresholds to create a curve.

The area under the Curve (the AUC), provides a single number to summarize the performance of a model in terms of its ROC Curve with a value between 0.5 (no-skill) and 1.0 (perfect skill).

```
Accuracy: 0.9095477386934674
False negative rate (with respect to misclassifications):  0.16666666666666666
False negative rate (with respect to all the data):  0.01507537688442211
False negatives: 3
False positives: 15
Mispredictions: 18
Total test data points: 199

Logistic regression
Scores
Accuracy --> 0.9095477386934674
Precison --> 0.868421052631579
Recall --> 0.9705882352941176
F1 --> 0.9166666666666667
```

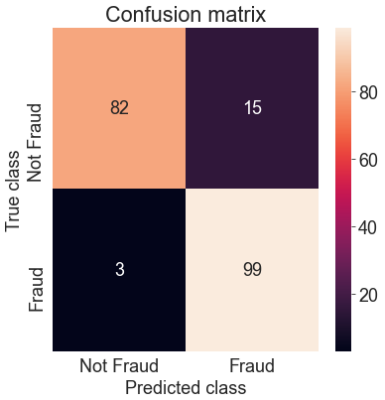


Figure 5: Using a threshold of 0.25 for predicting the test set

```
y_prediction = logreg.predict_proba(X_test)
predictions3 = binarize(y_prediction, threshold = 0.25)[:,1]
```

The test set (X_test) is predicted against the fitted training set where the predicted classes are tested using the values of their true class. From figure 5, the model reduced the amount of fraudulent transactions predicted as not fraud.

The overall accuracy was 91% which is less than the previous prediction score. The number of missed fraudulent transactions decreased from 8.8% $\rightarrow$ 2.9% which demonstrates that using a lower threshold for the

prediction reduces the false negative rate. However the number of false positives greatly increases, from 2% → 15%. In terms of the hypothesis objectives, this is not as big of an issue. Further testing indicates, as the threshold decreases the model assumes that most transactions are fraud; which the number of false positives becomes too incorrect for an accurate classification prediction model.

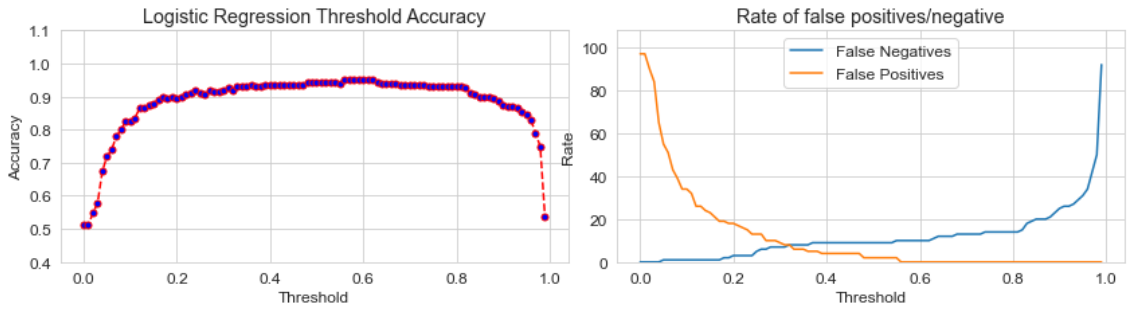


Figure 6: Threshold values from 0 to 1, with the accuracy and the number of false positives and negatives as a result

The threshold scale tips the balance between each transaction type. The lower threshold, the lower the number of missed fraudulent transaction, however there is an increase of classified normal transactions labeled as fraud. This is the opposite as the threshold increases. From Figure 6, the crossover threshold is 0.317 where there are the most optimal amounts of the misclassifications. Therefore, for the defined research hypothesis the threshold should be ≤ 0.317 which results in less type 1 and 2 predictions which overall reduces the amount of missed fraudulent transactions.

4.2 K Nearest Neighbours

K Nearest Neighbours (KNN) is a common supervised machine learning algorithm when the target variable is known, used for classification [2]. It is a lazy algorithm as KNN does not have a training step. All data points will be used only at the time of prediction. With no training step, the prediction step is costly. An eager learner algorithm eagerly learns during the training step. It uses feature similarity to predict the cluster that the new point will fall into, where K is a number used to identify similar neighbors for the new data point.

Step 1: Choose a value for K. Step 2: Find the distance of the new point to each of the training data. Step 3: Find the K nearest neighbors to the new data point. Step 4: Count the number of data points in each category among the K neighbors. New data point will belong to class that has the most neighbors.

The model uses the same data, training set and test set that the Logistic Regression model used for a fair comparison. Initially, the KNN model used 5 neighbours and achieved the following results:

```
knn=KNeighborsClassifier(n_neighbors=5,algorithm="kd_tree",n_jobs=-1)
knn.fit(X_train,y_train.ravel())
```

```
Accuracy: 0.9195979899497487
False negative rate (with respect to misclassifications):  0.6875
False negative rate (with respect to all the data):  0.05527638190954774
False negatives: 11
False positives: 5
Mispredictions: 16
Total test data points: 199

K-Nearest Neighbours
Scores
Accuracy --> 0.9195979899497487
Precison --> 0.9479166666666666
Recall --> 0.8921568627450981
F1 --> 0.9191919191919192
```

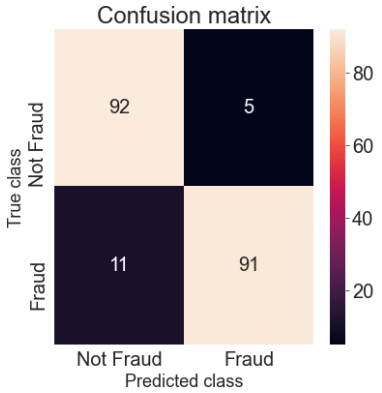


Figure 7: K = 5 for neighbours

The KNN model predicted more incorrect transactions, compared to the Logistic regression model with the KNN model achieving a 91% accuracy. The KNN classified 5 false positives and 11 false negatives. For 97 non fraudulent transactions in the test, 5.1% were misclassified as fraud. For 102 fraudulent transactions in the test, 10.2% were misclassified as non fraudulent.

The results show that the basic KNN model is less accurate than the basic Logistic regression model. The KNN model can be improved by using a greater number variables for the KNN model.

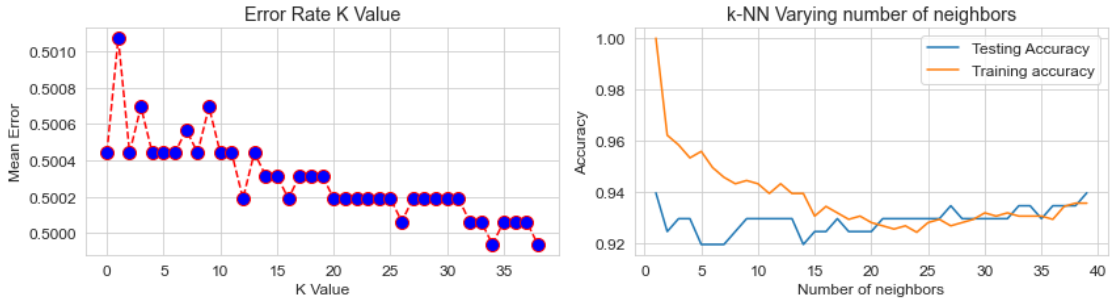


Figure 8: Testing accuracy for k in range = 0 to 40. A k of 38 Proved the best test accuracy

From Figure 8, the number of neighbours increases the accuracy, where 38-39 was the optimal number where the test accuracy was the highest. The mean error value decreased in only small increments, as the amount of transactions is small.

```
Accuracy: 0.9396984924623115
False negative rate (with respect to misclassifications):  0.5833333333333334
False negative rate (with respect to all the data):  0.035175879396984924
False negatives: 7
False positives: 5
Mispredictions: 12
Total test data points: 199

K-Nearest Neighbours
Scores
Accuracy --> 0.9396984924623115
Precison --> 0.95
Recall --> 0.9313725490196079
F1 --> 0.9405940594059405
```

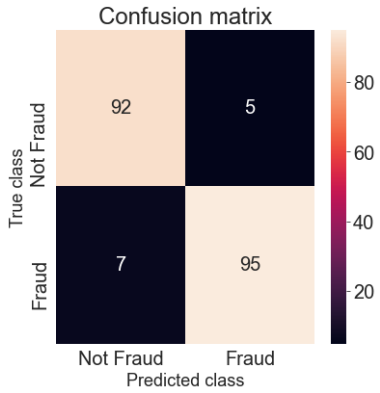


Figure 9: K = 38 for neighbours with optimised classifier

Using k = 38 for the KNN model, the number of false positives has remained the same, where the number of false negatives has reduced from 10.2% → 6%. The accuracy overall has also increased to 94%. The KNN model is as accurate as the basic logistic regression model but has more predicted fraud.

5 E: Recommended model

The primary indicators of fraud are not obvious, for example, an increase in the transaction amount made is possibly a real transaction that a user has made for a one off purchase. The same can be said for a normal transaction amount, which looks normal but is a small fraudulent transaction. These types of transactions are more likely to be misclassified.

Increasing the amount of non fraudulent transactions does not necessarily improve the models.

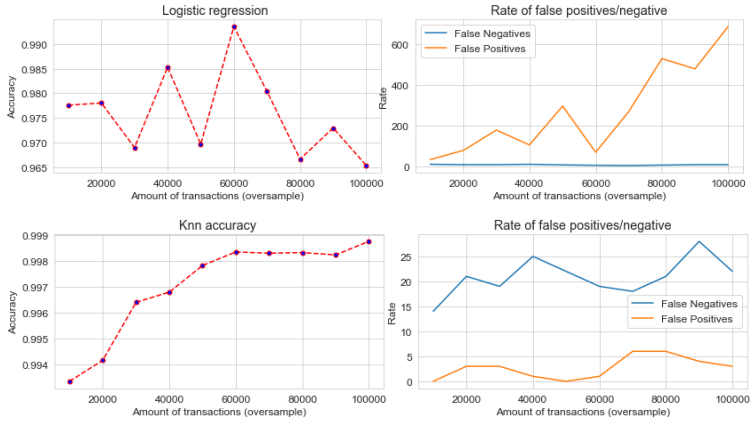


Figure 10: Accuracy and Rate of error as the number of transactions increases

Figure 10 shows that as the number of transactions increases in the test and training data, the accuracy increases for both models. However, the logistic regression model accuracy becomes lower at 100,000 transactions. Discussed above, the data is highly skewed and for high transaction amounts the logistic regression model threshold function reduces the amount of false negatives, however this

has an effect on the misclassified fraud transactions. This results in high incorrect misclassifications of non fraud.

On the other hand, the KNN model has a higher accuracy rate as the number of transactions increases. The number of false positives is low but the skewness means that false negative transactions are misclassified more often. As the number of transactions increases, the machine learning KNN model stood out to be the better out of the two. KNN accuracy was maintained above 99% and the number of misclassifications never exceeded 50. The Logistic regression model's accuracy is inconsistent, however still achieved a high accuracy of 99% at 60,000 transactions but the model proves that it cannot maintain this level of accuracy. As the number increased to 100,000 transactions, the accuracy dropped below 97%. From these results, the KNN model trained on the credit card fraud dataset is suitable as a model to predict fraudulent transactions [3]. It demonstrates consistently the least amount of error for the classification prediction, while still achieving a high accuracy. The

6 G: Further analysis

The main shortcoming of this dataset is the small amount of fraud data. The training data is highly unbalanced, and while using undersampling improves the accuracy, there is still not enough to properly train the models.

Further analysis should include more fraud types. Future areas of research could be in examining meta-classifiers approaches in handling highly imbalanced credit card fraud data [3]. Oversampling could be explored: adding the minority class (fraud) to our training sample, thus increasing the overall proportion of fraud records. This could be done by generating samples from the minority class by either duplicating existing records or artificially generating others.

Understandably, the users details were hidden for confidentiality but this data could be incorporated in a machine learning model to better identify fraud. Location could also be a factor which would benefit the fraud classification; if a transaction was made outside a users home city address then this would be flagged. To improve the model fitness, a high number of fraudulent transactions need to be tested using these machine learning models.

Banks and companies are less concerned with models which do not produce a high prediction score. It is better to be aware and to identify transactions that are fraudulent to reduce the impacts it could have on a customer or a company. Machine learning-based classification for credit card fraud prediction can further help to identify these transactions.

The findings from the machine learning models are suitable for a diverse transaction system, as real transactions will have more information about the transactions made which will give better identification and classification methods for stopping fraud.

References

[1] Scikit-learn: Machine Learning in Python, 2011. Pedregosa, F. Varoquaux, G. Gramfort, A. Michel, V. Thirion, B. Grisel, O. Blondel, M. Prettenhofer, P. Weiss, R. Dubourg, V. Vanderplas, J. Passos, A. Cournapeau, D.Brucher, M. Perrot, M. Duchesnay, E. Journal of Machine Learning Research, volume 12, 2825–2830,

[2] Carcillo, F., Le Borgne, Y.A., Caelen, O., Kessaci, Y., Oblé, F. and Bontempi, G., 2021. Combining unsupervised and supervised learning in credit card fraud detection. Information sciences, 557, pp.317-331.

[3] Awoyemi, J.O., Adetunmbi, A.O. and Oluwadare, S.A., 2017, October. Credit card fraud detection using machine learning techniques: A comparative analysis. In 2017 International Conference on Computing Networking and Informatics (ICCNI) (pp. 1-9). IEEE.

[4] A. Thennakoon, C. Bhagyani, S. Premadasa, S. Mihiranga and N. Kuruwitaarachchi, "Real-time Credit Card Fraud Detection Using Machine Learning," 2019 9th International Conference on Cloud Computing, Data Science Engineering (Confluence), 2019, pp. 488-493, doi: 10.1109/CONFLUENCE.2019.8776942.

Credit Card Fraud Detection

Fraudulent transactions happen every day and it is important that credit card companies are able to recognize these fraudulent credit card transactions.

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days:

492 fraudulent out of 284,807 transactions. The dataset is highly unbalanced where the positive class (frauds) account for 0.172% of all transactions.

https://www.kaggle.com/mlg-ulb/creditcardfraud

It contains only numerical variable inputs which are the result of a PCA transformation. Due to confidentiality the original features and more background information about the data are not included. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with the PCA are the variables 'Time' and 'Amount'.

'Time'- contains the seconds elapsed between each transaction and the first transaction in the dataset. 'Amount' - transaction Amount; this feature can be used for example-dependant cost-sensitive learning.

'Class' - is the response variable and it takes. Value 1 = Fraud and 0 = Not fraud.

In [120…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier # KNN algorithm
from sklearn.model_selection import train_test_split # data split

from sklearn.preprocessing import StandardScaler



%matplotlib inline
sns.set_style("whitegrid")
data = pd.read_csv("creditcard.csv")

scaler=StandardScaler()
df = pd.DataFrame(scaler.fit_transform(data.iloc[:,:-1]), columns=data.columns[:-1])

data.head()
```

Out[120…

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817

5 rows × 31 columns

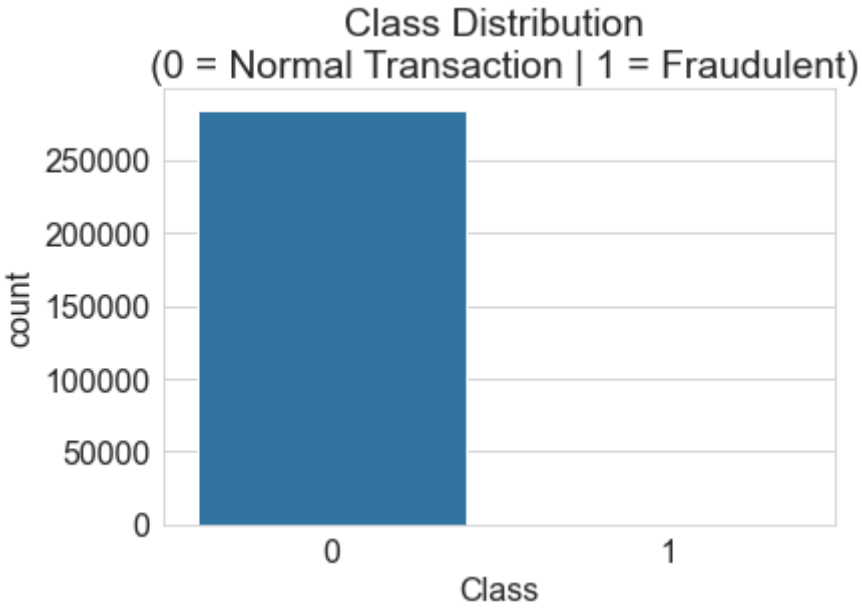
In [2]:
```
#any missing data
print(data.columns)
print("Null: ", data.isnull().sum().sum())
```

```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
Null:  0
```

In [4]:
```
Total_transactions = len(data)
normal = len(data[data.Class == 0])
fraudulent = len(data[data.Class == 1])
fraud_percentage = round(fraudulent/normal*100, 3)
print('Total number of Trnsactions are {}'.format(Total_transactions))
print('Number of Normal Transactions are {}'.format(normal))
print('Number of fraudulent Transactions are {}'.format(fraudulent))
print('Percentage of fraud Transactions is {}'.format(fraud_percentage))
```

```
Total number of Trnsactions are 284807
Number of Normal Transactions are 284315
Number of fraudulent Transactions are 492
Percentage of fraud Transactions is 0.173
```

In [126…
```
plt.figure(figsize=(6,4))
plt.rcParams.update({'font.size': 16})
sns.countplot(x = 'Class', data = data)
plt.title('Class Distribution \n (0 = Normal Transaction | 1 = Fraudulent)');
```

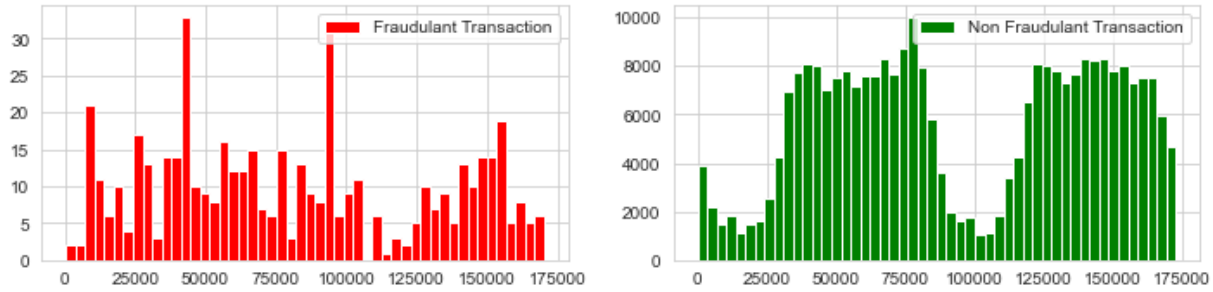


Looking at the distributions the tranaction type is highly skewed. Looking at further distributions of the other features:

In [121…
```
plt.figure(figsize=(12, 6))

plt.subplot(2, 2, 1)
data[data.Class == 1].Time.hist(bins=50, color='red', label="Fraudulant Transaction"
plt.legend()
```

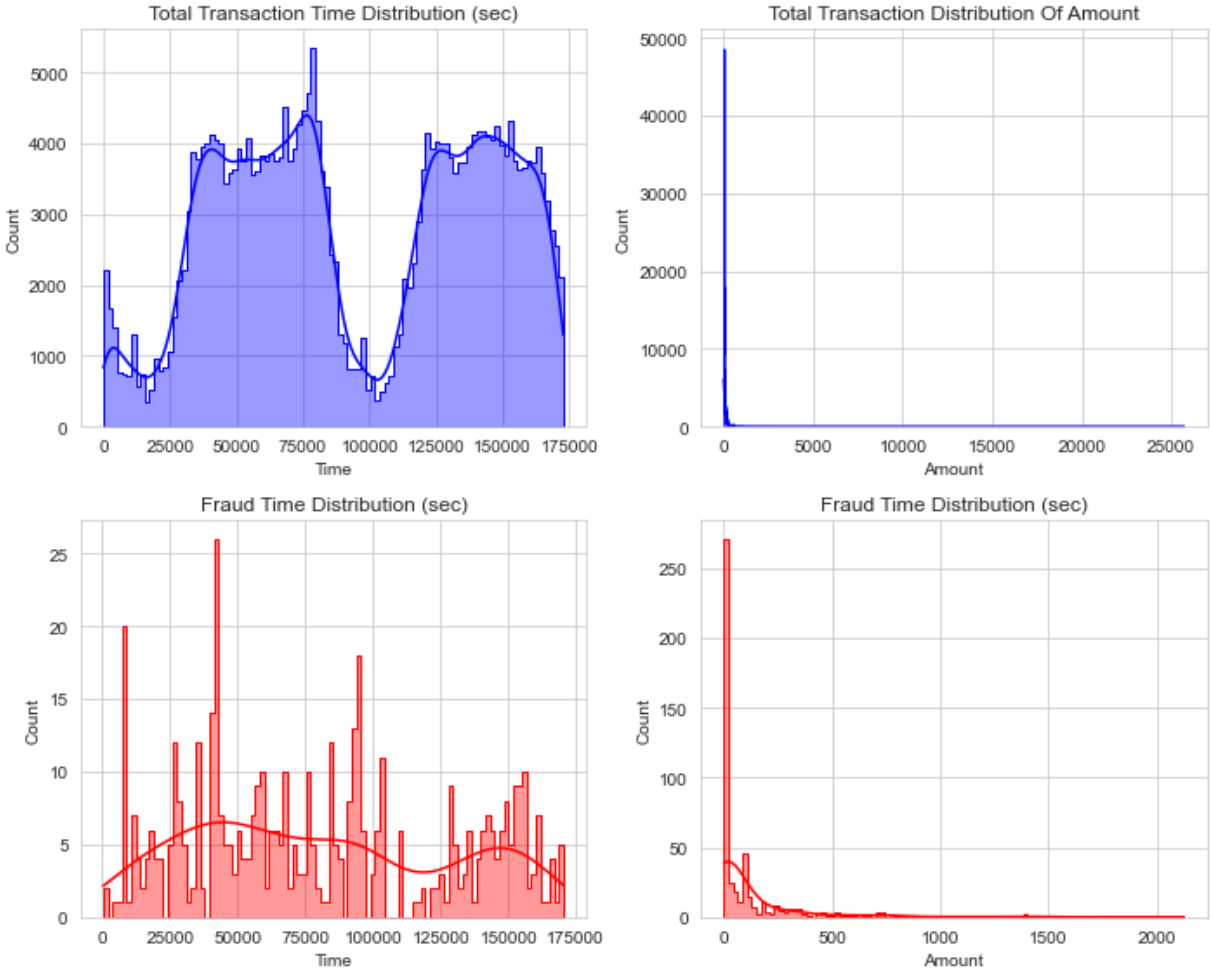
```
plt.subplot(2, 2, 2)
data[data.Class == 0].Time.hist(bins=50, color='green', label="Non Fraudulant Transa
plt.legend()
```

Out[121… <matplotlib.legend.Legend at 0x1e154e69a30>



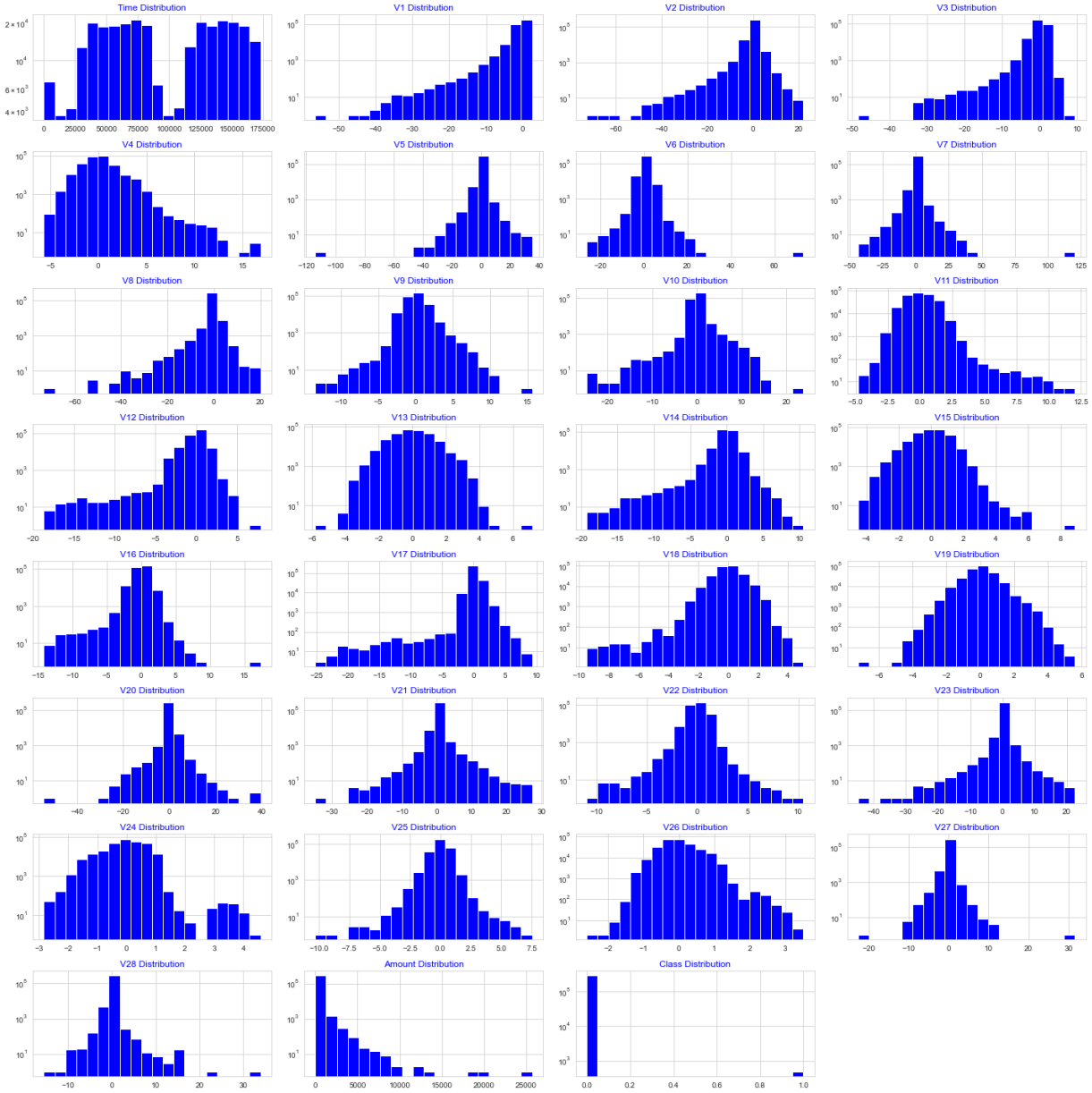
In [7]:

```
plt.figure(figsize=(10,8))

frauds = data[data['Class'] == 1]

plt.subplot(2, 2, 1)
plt.tight_layout()
plt.title('Total Transaction Time Distribution (sec)')
sns.histplot(data['Time'], bins= 100, alpha = 0.4, kde=True, element="step",color='b


# #plot the amount feature

plt.subplot(2, 2, 2)
plt.tight_layout()
plt.title('Total Transaction Distribution Of Amount')
sns.histplot(data['Amount'], alpha = 0.4, kde=True, element="step",color='blue');


plt.subplot(2, 2, 3)
plt.tight_layout()
plt.title('Fraud Time Distribution (sec)')
sns.histplot(frauds['Time'], bins= 100, alpha = 0.4, kde=True, element="step",color=

plt.subplot(2, 2, 4)
plt.tight_layout()
plt.title('Fraud Time Distribution (sec)')
sns.histplot(frauds['Amount'], bins= 100, alpha = 0.4, kde=True, element="step",colo
```



In [8]:

```
def draw_histograms(dataframe, features, rows, cols):
    fig=plt.figure(figsize=(20,20))
    for i, feature in enumerate(features):
        ax=fig.add_subplot(rows,cols,i+1)
        dataframe[feature].hist(bins=20,ax=ax,facecolor='Blue')
        ax.set_title(feature+" Distribution",color='Blue')
        ax.set_yscale('log')
    fig.tight_layout()
    plt.show()
draw_histograms(data,data.columns,8,4)
```



Data Pre-processing

Data preprocessing is neccessary because it helps to gain a better accuracy rather than just using raw data to build a model.

First, standardise the data using using Scikit-learn's StandardScaler(). In order to fit to the scaler the data should be reshaped within the range of -1 and 1.

In [80]:
```
from sklearn.metrics import accuracy_score # evaluation metric
from sklearn.metrics import confusion_matrix,accuracy_score,precision_score,recall_s



data['Vamount'] = StandardScaler().fit_transform(df['Amount'].values.reshape(-1,1))
data['Vtime'] = StandardScaler().fit_transform(df['Time'].values.reshape(-1,1))

data = data.drop(['Time','Amount'], axis = 1)
data.head()
```

Out[80]:

V1	V2	V3	V4	V5	V6	V7	V8	V9

	V1	V2	V3	V4	V5	V6	V7	V8	V9	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	(
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-(
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	(
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-(
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	(

5 rows × 31 columns

Shuffle the data

In [81]:
```
data = data.sample(frac=1)

frauds = data[data['Class'] == 1]
non_frauds = data[data['Class'] == 0][:500] # take 500 to match the amount of fraud

new_df = pd.concat([non_frauds, frauds])
# Shuffle dataframe rows
new_df = new_df.sample(frac=1, random_state=42)

new_df.head()
```

Out[81]:

	V1	V2	V3	V4	V5	V6	V7	V8	
268183	-4.237557	4.405630	-3.519578	-0.984004	-1.155992	-1.110501	-0.966956	2.196317	1.000
215692	-1.748580	1.841401	-0.123831	-0.888903	0.326017	-1.394945	0.776152	0.440814	-0.834(
258403	-5.976119	-7.196980	-5.388316	5.104799	4.676533	-5.566870	-4.291180	0.876531	-1.075-
182976	-0.879042	1.585618	-0.522312	-0.558494	-0.186661	-1.234188	0.355993	0.598737	-0.336-
215306	-0.713734	0.883207	-0.621973	-1.024331	1.000327	0.075694	0.870387	0.173685	-0.025-

5 rows × 31 columns

With the dataset defined, separating the input variables from the target variable, we divided the data into training and test sets, importing the train_test_split function.

The train_test_split function uses a randomizer to separate data into training and test sets. In this case, 80% of the data for training and 20% for tests were defined.

Features are the entire dataset exempt from the Class variable. Lables are the actual class of the dataset rows.

In [82]:
```
from sklearn.preprocessing import normalize

features = new_df.drop(['Class'], axis = 1)
labels = pd.DataFrame(new_df['Class'])


feature_array = features.values
label_array = labels.values
```

```
X_train,X_test,y_train,y_test = train_test_split(feature_array,label_array,test_size


# normalize: Scale input vectors individually to unit norm (vector length).
X_train = normalize(X_train)
X_test = normalize(X_test)
```

KNN

K Nearest Neighbors model. Starting using k = 5, then testing how the number of neighbours affects the test accuracy

In [102…

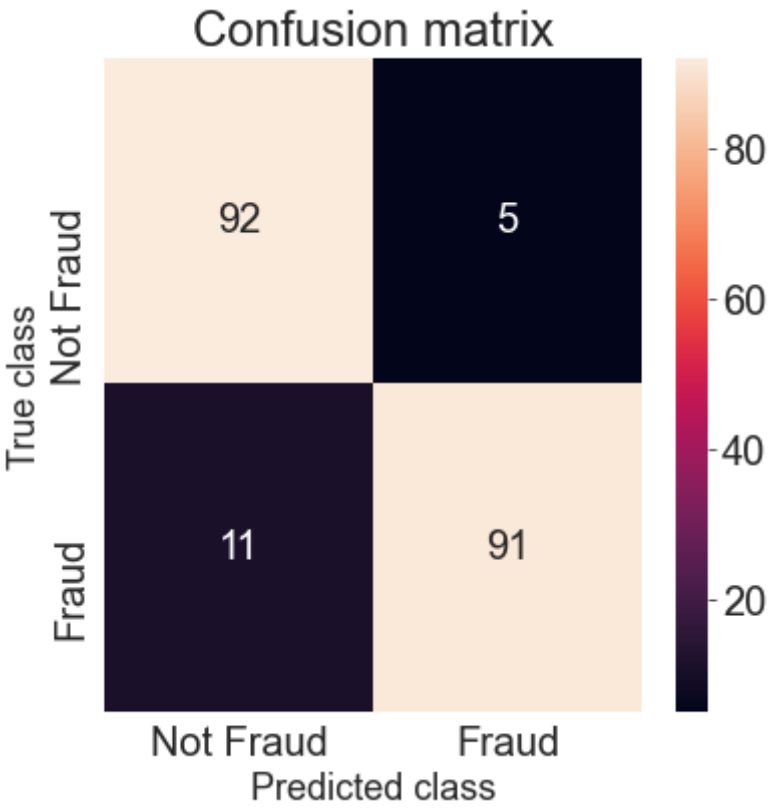
```
knn = KNeighborsClassifier(algorithm='kd_tree',n_jobs=-1, n_neighbors=5)

knn.fit(X_train,y_train.ravel())
predictions1 = knn.predict(X_test)

pred_fraud = np.where(predictions1 == 1)[0]
real_fraud = np.where(y_test == 1)[0]
false_pos = len(np.setdiff1d(pred_fraud, real_fraud))



pred_good = np.where(predictions1 == 0)[0]
real_good = np.where(y_test == 0)[0]
false_neg = len(np.setdiff1d(pred_good, real_good))

false_neg_rate = false_neg/(false_pos+false_neg)

accuracy = (len(X_test) - (false_neg + false_pos)) / len(X_test)
print("Accuracy:", accuracy)
print("False negative rate (with respect to misclassifications): ", false_neg_rate)
print("False negative rate (with respect to all the data): ", false_neg / len(predic
print("False negatives:", false_neg , "\n"
      "False positives:", false_pos , "\n"
      "Mispredictions:", false_neg + false_pos)
print("Total test data points:", len(X_test))

knn_accuracy_score  = accuracy_score(y_test,predictions1)
knn_precison_score  = precision_score(y_test,predictions1)
knn_recall_score    = recall_score(y_test,predictions1)
knn_f1_score        = f1_score(y_test,predictions1)

#printing
print("")
print("K-Nearest Neighbours")
print("Scores")
print("Accuracy -->",knn_accuracy_score)
print("Precison -->",knn_precison_score)
print("Recall -->",knn_recall_score)
print("F1 -->",knn_f1_score)
```

```
Accuracy: 0.9195979899497487
False negative rate (with respect to misclassifications):  0.6875
False negative rate (with respect to all the data):  0.05527638190954774
False negatives: 11
False positives: 5
Mispredictions: 16
Total test data points: 199
```

```
K-Nearest Neighbours
Scores
Accuracy --> 0.9195979899497487
Precison --> 0.9479166666666666
Recall --> 0.8921568627450981
F1 --> 0.9191919191919192
```

In [103…

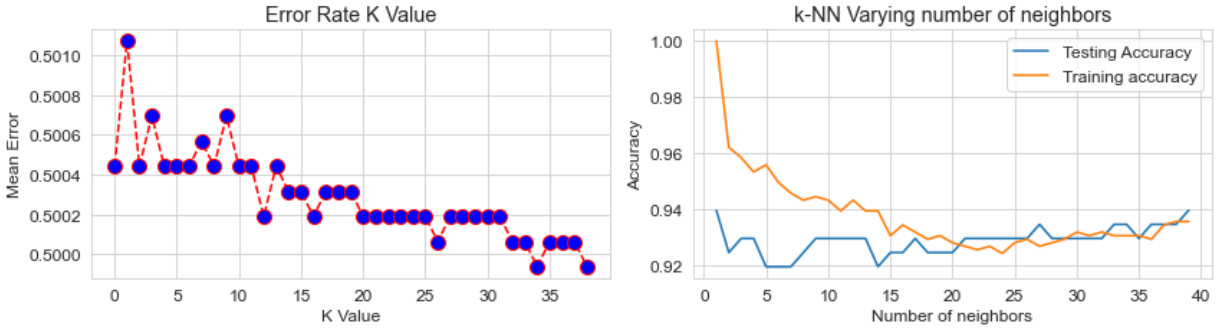
```
label_font = {'size':'18'}  # Adjust to fit
LABELS = ['Not Fraud', 'Fraud']
conf_matrix = confusion_matrix(y_test, predictions1)
plt.figure(figsize=(6, 6))
plt.rcParams.update({'font.size': 20})
sns.heatmap(conf_matrix, xticklabels=LABELS, yticklabels=LABELS, annot=True, fmt="d"
plt.title("Confusion matrix")
plt.ylabel('True class', fontdict=label_font)
plt.xlabel('Predicted class', fontdict=label_font)
plt.show()
```



In [104…

```
neighbours = np.arange(1,40)
train_accuracy = np.empty(len(neighbours))
test_accuracy = np.empty(len(neighbours))

error = []

for i,k in enumerate(neighbours):

    #Setup a knn classifier with k neighbors
    knn = KNeighborsClassifier(algorithm='kd_tree',n_jobs=-1, n_neighbors=k)


    #Fit the model
    knn.fit(X_train,y_train.ravel())


    #Compute accuracy on the training set
    train_accuracy[i] = knn.score(X_train, y_train.ravel())
```

```
    #Compute accuracy on the test set
    test_accuracy[i] = knn.score(X_test, y_test.ravel())

    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != y_test))

    #print(i)

plt.figure(figsize=(12, 6))
plt.rcParams.update({'font.size': 12})
plt.subplot(2,2,1)
plt.plot(range(len(neighbours)), error, color='red', linestyle='dashed', marker='o',
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.tight_layout()
plt.ylabel('Mean Error')


#Generate plot
plt.subplot(2,2,2)

plt.title('k-NN Varying number of neighbors')
plt.plot(neighbours, test_accuracy, label='Testing Accuracy')
plt.plot(neighbours, train_accuracy, label='Training accuracy')
plt.legend()
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.tight_layout()
plt.show()
```



In [116…

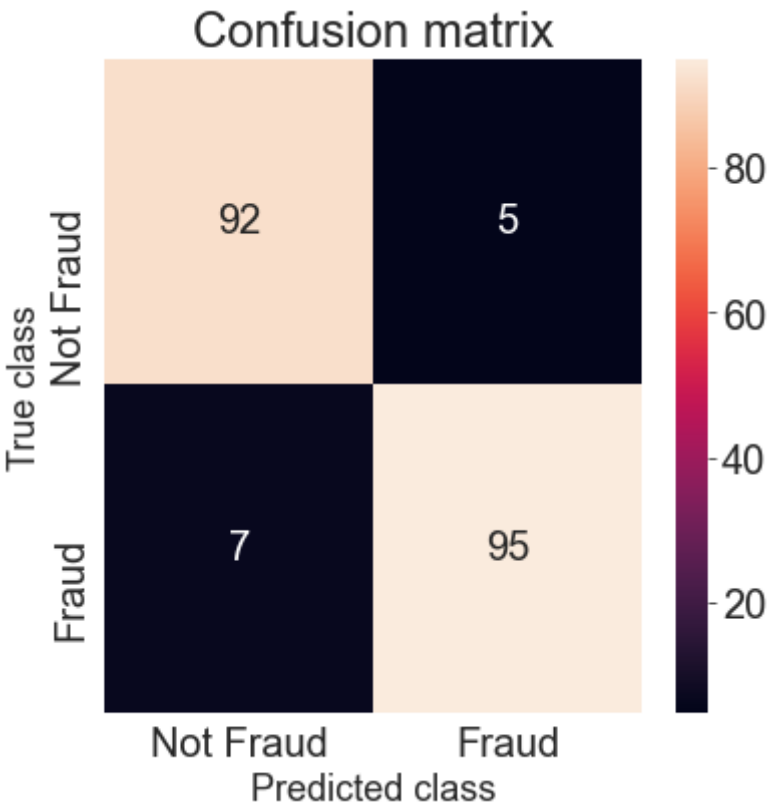
```
idx = np.where(test_accuracy == max(test_accuracy))
x = neighbours[idx]

#knn= KNeighborsClassifier(algorithm='kd_tree', leaf_size=30, n_jobs=-1, n_neighbors
knn = KNeighborsClassifier(algorithm='kd_tree',n_jobs=-1, n_neighbors=x[1])

knn.fit(X_train,y_train.ravel())
predictions1 = knn.predict(X_test)

pred_fraud = np.where(predictions1 == 1)[0]
real_fraud = np.where(y_test == 1)[0]
false_pos = len(np.setdiff1d(pred_fraud, real_fraud))



pred_good = np.where(predictions1 == 0)[0]
real_good = np.where(y_test == 0)[0]
false_neg = len(np.setdiff1d(pred_good, real_good))

false_neg_rate = false_neg/(false_pos+false_neg)

accuracy = (len(X_test) - (false_neg + false_pos)) / len(X_test)
```

```
print("Accuracy:", accuracy)
print("False negative rate (with respect to misclassifications): ", false_neg_rate)
print("False negative rate (with respect to all the data): ", false_neg / len(predic
print("False negatives:", false_neg , "\n"
      "False positives:", false_pos , "\n"
      "Mispredictions:", false_neg + false_pos)
print("Total test data points:", len(X_test))

knn_accuracy_score  = accuracy_score(y_test,predictions1)
knn_precison_score  = precision_score(y_test,predictions1)
knn_recall_score    = recall_score(y_test,predictions1)
knn_f1_score        = f1_score(y_test,predictions1)

#printing
print("")
print("K-Nearest Neighbours")
print("Scores")
print("Accuracy -->",knn_accuracy_score)
print("Precison -->",knn_precison_score)
print("Recall -->",knn_recall_score)
print("F1 -->",knn_f1_score)
```

```
Accuracy: 0.9396984924623115
False negative rate (with respect to misclassifications):  0.5833333333333334
False negative rate (with respect to all the data):  0.035175879396984924
False negatives: 7
False positives: 5
Mispredictions: 12
Total test data points: 199

K-Nearest Neighbours
Scores
Accuracy --> 0.9396984924623115
Precison --> 0.95
Recall --> 0.9313725490196079
F1 --> 0.9405940594059405
```

In [117…

```
label_font = {'size':'18'}  # Adjust to fit
LABELS = ['Not Fraud', 'Fraud']
conf_matrix = confusion_matrix(y_test, predictions1)
plt.figure(figsize=(6, 6))
plt.rcParams.update({'font.size': 20})
sns.heatmap(conf_matrix, xticklabels=LABELS, yticklabels=LABELS, annot=True, fmt="d"
plt.title("Confusion matrix")
plt.ylabel('True class', fontdict=label_font)
plt.xlabel('Predicted class', fontdict=label_font)
plt.show()
```



Logistic regression

Using the same processed dataset for the same testest using Logistc regression

In [108…

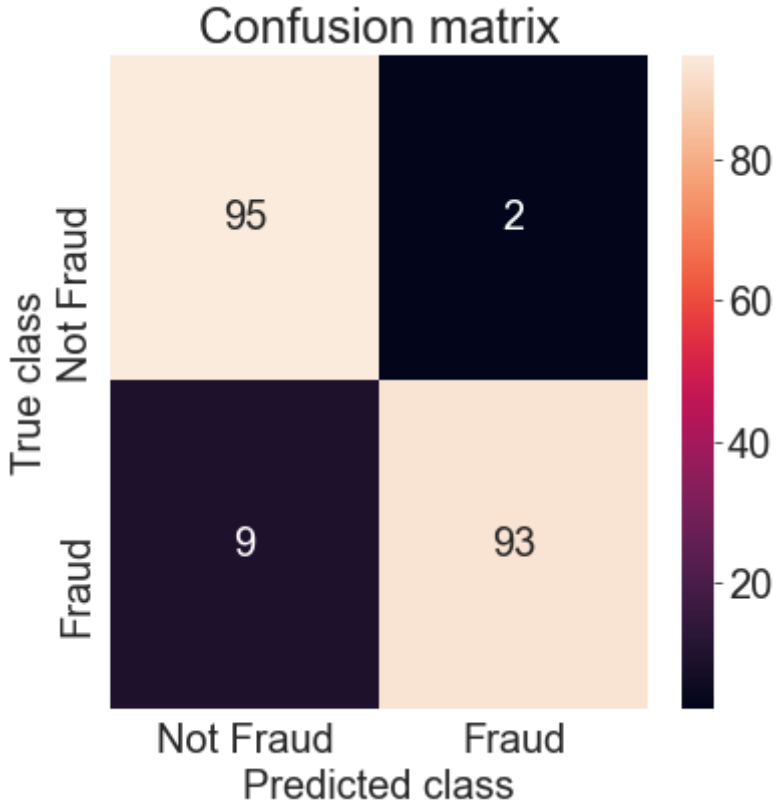
```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(C=1, penalty='l2')
lr.fit(X_train, y_train.ravel())
predictions2 = lr.predict(X_test)

pred_fraud = np.where(predictions2 == 1)[0]
real_fraud = np.where(y_test == 1)[0]
false_pos = len(np.setdiff1d(pred_fraud, real_fraud))


pred_good = np.where(predictions2 == 0)[0]
real_good = np.where(y_test == 0)[0]
false_neg = len(np.setdiff1d(pred_good, real_good))

false_neg_rate = false_neg/(false_pos+false_neg)

accuracy = (len(X_test) - (false_neg + false_pos)) / len(X_test)
print("Accuracy:", accuracy)
print("False negative rate (with respect to misclassifications): ", false_neg_rate)
print("False negative rate (with respect to all the data): ", false_neg / len(predic
print("False negatives:", false_neg , "\n"
      "False positives:", false_pos , "\n"
      "Mispredictions:", false_neg + false_pos)
print("Total test data points:", len(X_test))

lr_accuracy_score  = accuracy_score(y_test,predictions2)
lr_precison_score  = precision_score(y_test,predictions2)
lr_recall_score    = recall_score(y_test,predictions2)
lr_f1_score        = f1_score(y_test,predictions2)
```

```
#printing
print("")
print("Logistic regression")
print("Scores")
print("Accuracy -->",lr_accuracy_score)
print("Precison -->",lr_precison_score)
print("Recall -->",lr_recall_score)
print("F1 -->",lr_f1_score)
```

```
Accuracy: 0.9447236180904522
False negative rate (with respect to misclassifications):  0.8181818181818182
False negative rate (with respect to all the data):  0.04522613065326633
False negatives: 9
False positives: 2
Mispredictions: 11
Total test data points: 199

Logistic regression
Scores
Accuracy --> 0.9447236180904522
Precison --> 0.9789473684210527
Recall --> 0.9117647058823529
F1 --> 0.9441624365482234
```

In [93]:

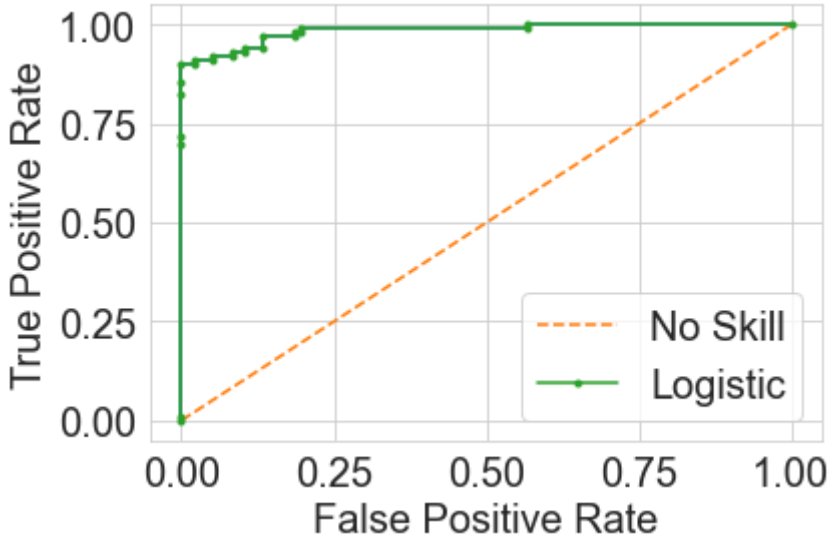
```
label_font = {'size':'20'}  # Adjust to fit
LABELS = ['Not Fraud', 'Fraud']
conf_matrix = confusion_matrix(y_test, predictions2)
plt.figure(figsize=(6, 6))
plt.rcParams.update({'font.size': 20})
sns.heatmap(conf_matrix, xticklabels=LABELS, yticklabels=LABELS, annot=True, fmt="d"
plt.title("Confusion matrix")
plt.ylabel('True class', fontdict=label_font)
plt.xlabel('Predicted class', fontdict=label_font)
plt.show()
```



++++++++++++++++++++

In [94]:

```
from sklearn.preprocessing import binarize
from sklearn.metrics import roc_curve
from sklearn.metrics import classification_report

logreg=LogisticRegression(C=1, penalty='l2')
logreg.fit(X_train,y_train.ravel())
y_pred=logreg.predict(X_test)

y_pred_thresh1=logreg.predict_proba(X_test)

fpr, tpr, thresholds = roc_curve(y_test, y_pred_thresh1[:,1])


plt.plot(fpr,tpr)
plt.plot([0,1], [0,1], linestyle='--', label='No Skill')
plt.plot(fpr, tpr, marker='.', label='Logistic')
# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
# show the plot
plt.show()
```



In [127…

```
from numpy import sqrt

gmeans = sqrt(tpr * (1-fpr))
ix = np.argmax(gmeans)
print('Best Threshold=%f, G-Mean=%.3f' % (thresholds[ix], gmeans[ix]))
```

```
Best Threshold=0.620366, G-Mean=0.950
```

In [112…

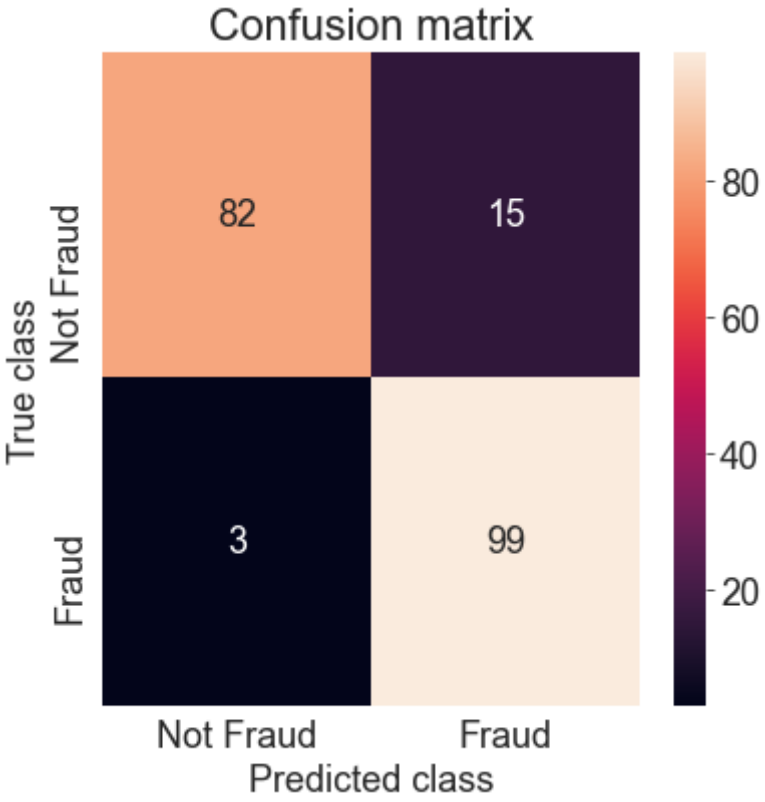
```
y_prediction = logreg.predict_proba(X_test)
predictions3 = binarize(y_prediction, threshold=0.23)[:,1]

pred_fraud = np.where(predictions3 == 1)[0]
real_fraud = np.where(y_test == 1)[0]
false_pos = len(np.setdiff1d(pred_fraud, real_fraud))

pred_good = np.where(predictions3 == 0)[0]
real_good = np.where(y_test == 0)[0]
false_neg = len(np.setdiff1d(pred_good, real_good))

false_neg_rate = false_neg/(false_pos+false_neg)
```

```
accuracy = (len(X_test) - (false_neg + false_pos)) / len(X_test)
print("Accuracy:", accuracy)
print("False negative rate (with respect to misclassifications): ", false_neg_rate)
print("False negative rate (with respect to all the data): ", false_neg / len(predic
print("False negatives:", false_neg , "\n"
      "False positives:", false_pos , "\n"
      "Mispredictions:", false_neg + false_pos)
print("Total test data points:", len(X_test))

lr_accuracy_score  = accuracy_score(y_test,predictions3)
lr_precison_score  = precision_score(y_test,predictions3)
lr_recall_score    = recall_score(y_test,predictions3)
lr_f1_score        = f1_score(y_test,predictions3)

#printing
print("")
print("Logistic regression")
print("Scores")
print("Accuracy -->",lr_accuracy_score)
print("Precison -->",lr_precison_score)
print("Recall -->",lr_recall_score)
print("F1 -->",lr_f1_score)
```

```
Accuracy: 0.9095477386934674
False negative rate (with respect to misclassifications):  0.16666666666666666
False negative rate (with respect to all the data):  0.01507537688442211
False negatives: 3
False positives: 15
Mispredictions: 18
Total test data points: 199

Logistic regression
Scores
Accuracy --> 0.9095477386934674
Precison --> 0.868421052631579
Recall --> 0.9705882352941176
F1 --> 0.9166666666666667
```

In [113…

```
label_font = {'size':'18'}  # Adjust to fit
LABELS = ['Not Fraud', 'Fraud', ]
conf_matrix = confusion_matrix(y_test, predictions3)
plt.figure(figsize=(6, 6))
plt.rcParams.update({'font.size': 18})
sns.heatmap(conf_matrix, xticklabels=LABELS, yticklabels=LABELS, annot=True, fmt="d"
plt.title("Confusion matrix")
plt.ylabel('True class', fontdict=label_font)
plt.xlabel('Predicted class', fontdict=label_font)
plt.show()
```

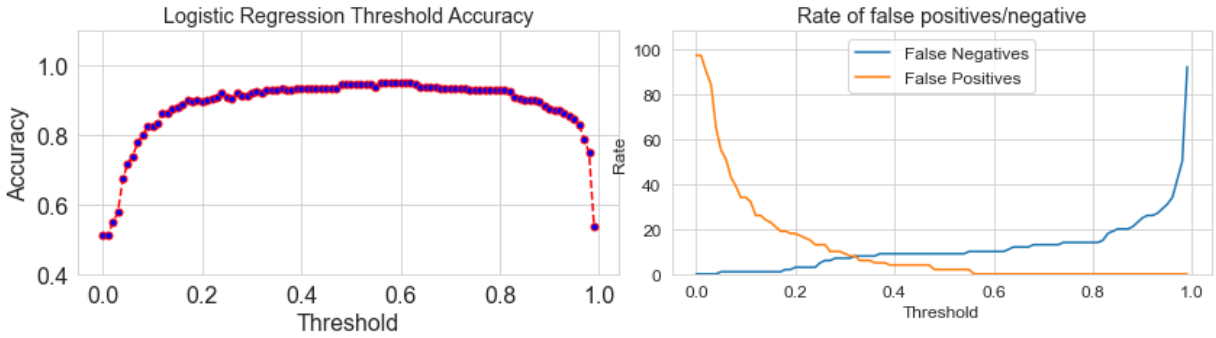


We can test how the model performs with the threshold value from 0 to 1 in 0.01 increments

In [128…

```
from matplotlib.pyplot import figure

x = []
acc = []
false_n = []
false_p = []

for i in np.arange(0, 1, 0.01):
    x.append(i)

    y_prediction = logreg.predict_proba(X_test)
    predictions3 = binarize(y_prediction, threshold=i)[:,1]

    pred_fraud = np.where(predictions3 == 1)[0]
    real_fraud = np.where(y_test == 1)[0]
    false_pos = len(np.setdiff1d(pred_fraud, real_fraud))



    pred_good = np.where(predictions3 == 0)[0]
    real_good = np.where(y_test == 0)[0]
    false_neg = len(np.setdiff1d(pred_good, real_good))



    false_neg_rate = false_neg/(false_pos+false_neg)

    accuracy = (len(X_test) - (false_neg + false_pos)) / len(X_test)

    false_n.append(false_neg)
    false_p.append(false_pos)
    acc.append(accuracy)
```

```
plt.figure(figsize=(12, 6))

plt.subplot(2,2,1)
plt.rcParams.update({'font.size': 12})
plt.ylim(ymax = 1.1, ymin = 0.4)
plt.plot(x, acc, color='red', linestyle='dashed', marker='o', markerfacecolor='blue'
plt.title('Logistic Regression Threshold Accuracy')
plt.xlabel('Threshold')
plt.tight_layout()
plt.ylabel('Accuracy')


plt.subplot(2,2,2)
plt.tight_layout()
plt.ylim(ymax = 108, ymin = 0)
plt.title('Rate of false positives/negative')
plt.plot(x, false_n, label='False Negatives')
plt.plot(x, false_p, label='False Positives')
plt.xlabel('Threshold')
plt.legend()
plt.ylabel('Rate')

plt.show()
```



How do the models perfrom with a higher imbalence of non fraud transactions? So far the weight has been (roughly) equal.

Each model will use 10,000 to 100,000 non fraud transactions.

Logistic regression

In [76]:

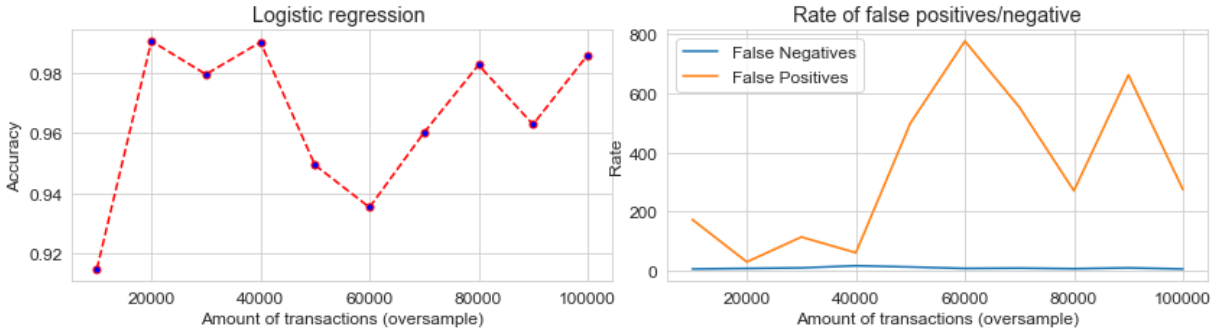
```
from numpy import sqrt


df = pd.read_csv('creditcard.csv')

df['Vamount'] = StandardScaler().fit_transform(df['Amount'].values.reshape(-1,1))
df['Vtime'] = StandardScaler().fit_transform(df['Time'].values.reshape(-1,1))

df = df.drop(['Time','Amount'], axis = 1)

df = df.sample(frac=1)

range_lr = []

acc_lr = []
false_nl = []
false_pl = []
```

```
for trn in np.arange(10000, 110000, 10000):
    range_lr.append(trn)
    frauds = df[df['Class'] == 1]
    non_frauds = df[df['Class'] == 0][:trn]

    new_df = pd.concat([non_frauds, frauds])
    # Shuffle dataframe rows
    new_df = new_df.sample(frac=1, random_state=42)

    features = new_df.drop(['Class'], axis = 1)
    labels = pd.DataFrame(new_df['Class'])

    feature_array = features.values
    label_array = labels.values

    X_train,X_test,y_train,y_test = train_test_split(feature_array,label_array,test_

    # normalize: Scale input vectors individually to unit norm (vector length).
    X_train = normalize(X_train)
    X_test= normalize(X_test)

    logreg=LogisticRegression(C=1, penalty='l2')
    logreg.fit(X_train,y_train.ravel())
    y_pred=logreg.predict(X_test)



    y_pred_thresh2=logreg.predict_proba(X_test)



    fpr, tpr, thresholds = roc_curve(y_test, y_pred_thresh2[:,1])

    gmeans = sqrt(tpr * (1-fpr))
    ix = np.argmax(gmeans)

    y_prediction = logreg.predict_proba(X_test)
    predictions3 = binarize(y_prediction, threshold= thresholds[ix])[:,1]

    pred_fraud = np.where(predictions3 == 1)[0]
    real_fraud = np.where(y_test == 1)[0]
    false_pos = len(np.setdiff1d(pred_fraud, real_fraud))

    pred_good = np.where(predictions3 == 0)[0]
    real_good = np.where(y_test == 0)[0]
    false_neg = len(np.setdiff1d(pred_good, real_good))

    false_neg_rate = false_neg/(false_pos+false_neg)

    accuracy = (len(X_test) - (false_neg + false_pos)) / len(X_test)


    false_nl.append(false_neg)
    false_pl.append(false_pos)
    acc_lr.append(accuracy)
```

In [77]:

```
plt.figure(figsize=(12, 6))

plt.subplot(2,2,1)
plt.rcParams.update({'font.size': 12})
plt.plot(range_lr, acc_lr, color='red', linestyle='dashed', marker='o', markerfaceco
```

```
plt.title('Logistic regression ')
plt.xlabel('Amount of transactions (oversample)')
plt.tight_layout()
plt.ylabel('Accuracy')


plt.subplot(2,2,2)
plt.tight_layout()

plt.title('Rate of false positives/negative')
plt.plot(range_lr, false_nl, label='False Negatives')
plt.plot(range_lr, false_pl, label='False Positives')
plt.xlabel('Amount of transactions (oversample)')
plt.legend()
plt.ylabel('Rate')

plt.show()
```



KNN

In [470…

```
df = pd.read_csv('creditcard.csv')

df['Vamount'] = StandardScaler().fit_transform(df['Amount'].values.reshape(-1,1))
df['Vtime'] = StandardScaler().fit_transform(df['Time'].values.reshape(-1,1))

df = df.drop(['Time','Amount'], axis = 1)

df = df.sample(frac=1)

range_knn = []

acc_knn = []
false_nk = []
false_pk = []

for trn in np.arange(10000, 110000, 10000):
    print(trn)
    range_knn.append(trn)
    frauds = df[df['Class'] == 1]
    non_frauds = df[df['Class'] == 0][:trn]

    new_df = pd.concat([non_frauds, frauds])
    # Shuffle dataframe rows
    new_df = new_df.sample(frac=1, random_state=42)

    features = new_df.drop(['Class'], axis = 1)
    labels = pd.DataFrame(new_df['Class'])

    feature_array = features.values
    label_array = labels.values

    X_train,X_test,y_train,y_test = train_test_split(feature_array,label_array,test_
```

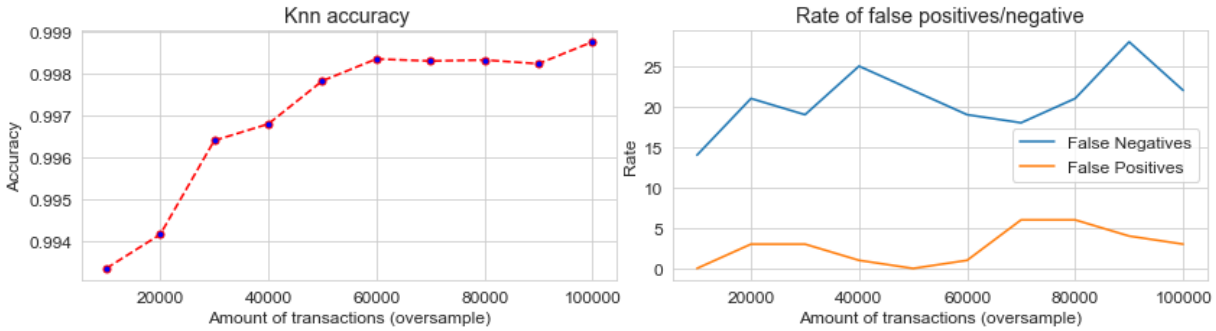
```
    # normalize: Scale input vectors individually to unit norm (vector length).
    X_train = normalize(X_train)
    X_test= normalize(X_test)

    neighbours = np.arange(27,40)
    train_accuracy = np.empty(len(neighbours))
    test_accuracy = np.empty(len(neighbours))

#     for i,k in enumerate(neighbours):
#         #Setup a knn classifier with k neighbors
#         knn=KNeighborsClassifier(n_neighbors=k,algorithm="kd_tree",n_jobs=-1)

#         #Fit the model
#         knn.fit(X_train,y_train.ravel())

#         #Compute accuracy on the training set
#         train_accuracy[i] = knn.score(X_train, y_train.ravel())

#         #Compute accuracy on the test set
#         test_accuracy[i] = knn.score(X_test, y_test.ravel())

    idx = np.where(test_accuracy == max(test_accuracy))
    x = neighbours[idx]

    knn = KNeighborsClassifier(algorithm='kd_tree',n_jobs=-1, n_neighbors=38)

    knn.fit(X_train,y_train.ravel())
    predictions1 = knn.predict(X_test)

    pred_fraud = np.where(predictions1 == 1)[0]
    real_fraud = np.where(y_test == 1)[0]
    false_pos = len(np.setdiff1d(pred_fraud, real_fraud))



    pred_good = np.where(predictions1 == 0)[0]
    real_good = np.where(y_test == 0)[0]
    false_neg = len(np.setdiff1d(pred_good, real_good))

    false_neg_rate = false_neg/(false_pos+false_neg)

    accuracy = (len(X_test) - (false_neg + false_pos)) / len(X_test)

    false_nk.append(false_neg)
    false_pk.append(false_pos)
    acc_knn.append(accuracy)
```

```
10000
20000
30000
40000
50000
60000
70000
80000
90000
```

In [477…

```
plt.figure(figsize=(12, 6))

plt.subplot(2,2,1)
plt.rcParams.update({'font.size': 12})
plt.plot(range_knn, acc_knn, color='red', linestyle='dashed', marker='o', markerface
plt.title('Knn accuracy ')
```

```
plt.xlabel('Amount of transactions (oversample)')
plt.tight_layout()
plt.ylabel('Accuracy')


plt.subplot(2,2,2)
plt.tight_layout()
plt.title('Rate of false positives/negative')
plt.plot(range_knn, false_nk, label='False Negatives')
plt.plot(range_knn, false_pk, label='False Positives')
plt.xlabel('Amount of transactions (oversample)')
plt.legend()
plt.ylabel('Rate')
plt.show()
```



In [ ]: