REPORT

Domain Background

Anomaly detection is the process of identifying unexpected items or events in a datasets. It is a process that in machine learning is usually applied to unlabelled data to classify points that differ from the norm. Its applications include domains such as network intrusion detection, fraud detection as well as life science and medicine.

Grubbs in 1969 defined it as “An outlying observation, or *outlier*, is one that appears to deviate markedly from other members of the sample in which it occurs”. The way to detect anomalies is to look for data-points whose features re different from the nom, and are rare in a dataset compared to normal instances.

The main types of anomaly detection -

Intrusion detection - most well known application. Anomaly detection is used to monitor network traffic and server applications for potential intrusion attempts and to identify exploits. Intrusion detection systems use fast anomaly detection algorithms because they deal with huge amounts of data that they have to process at real time.

Fraud detection: log data is analysed to identify misuse of the systems or suspicious activities. Its very important in financial transactions to spot fraudulent accounting.

Data Leakage Prevention (DLP) - in practice, DLP is fraud detection with a focus pm near-real time analysis on the logs from databases, file servers and other information sources.

Medical sciences - Anomaly detection is very powerful and promising in medical applications because it can allows inputs like images and electrocardiography (ECG) signals to be processes and analysed for critical, possibly life-threatening illnesses. In this application, anomaly detection algorithms rely on complex image processing methods for preprocessing. In life sciences, anomaly detection might also be utilised to find pathologies and mutants.

Now, supervised learning methods might be at most times very difficult to use to detect credit card fraud. Firstly, labelled datasets for credit card fraud is rare, and fraud itself is very unique amongst the total number of transactions taking place over a time period. This means that fraud might need to be over-sampled to get a big enough sample size. The popular models to choose are Logistic regression, Bayesian models and Random Forests.

Problem statement

- Goal: to detect credit-card fraud from labelled anonymised data.
- We are going to train one supervised detection model, and one anomaly detection model on the credit-card transaction data. We want to see how they compare with each other, and a benchmark model.
- Such models will be able to predict if future credit-card transaction is fraudulent or not.

Datasets and inputs

Dataset: https://www.kaggle.com/dalpozz/creditcardfraud*:* public data of anonymised credit card fraud. The data-set contains credit card transactions made during a period of two days, with 492 frauds out of total 284,807 transactions made. It shows the time the transactions were made, the dollar-amount of the transactions and the labels (class, 0/1

or normal/fraudulent) for the transactions. It has been pre-processed by PCA transformation to protect the anonymity of the account holders.

Evaluation metics

Receiver Operating Characteristics is the True Positive rate versus False positive rate. We will use this to evaluate the sensitivity/recall, specificity, and F_2 score for the models. True positives would represent the datapoint that are manipulated cases classified correctly as anomalies. False positives would represent the number of non-manipulated samples that are incorrectly classified as anomalies, while True Negatives would represent datapoint that are correctly classified to be not anomalous while False negatives would be the number of manipulated samples that are incorrectly classified as negative.

ROC analysis measure three aspects of the learning algorithm: precision(P) and recall(R) and specificity (SPC).

$$P = \frac{TP}{TP+FP} \qquad R = \frac{TP}{TP+FN} \qquad SPC = \frac{TN}{TN+FP}$$

Recall would measure the how well the model correctly classifies manipulated samples as anomalies. Specificity measures how successful the model is in classifying non-manipulated samples as negative. With these measurements, we want to find the F_2 score. We choose the F_2 score over F_1 score because F_2 would put a larger (i.e. a factor of two) emphasise (weight) on recall over precision because beta is larger.

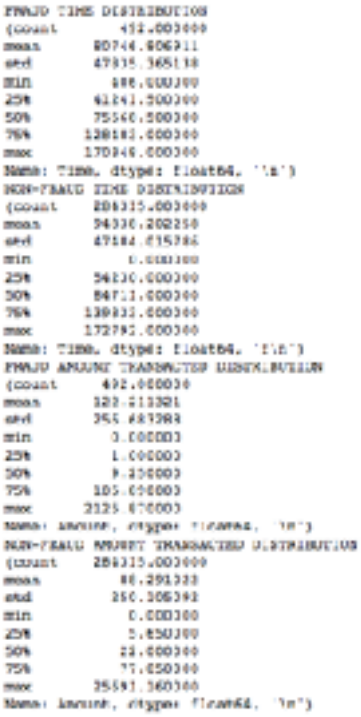
\

REPORT

Data exploration

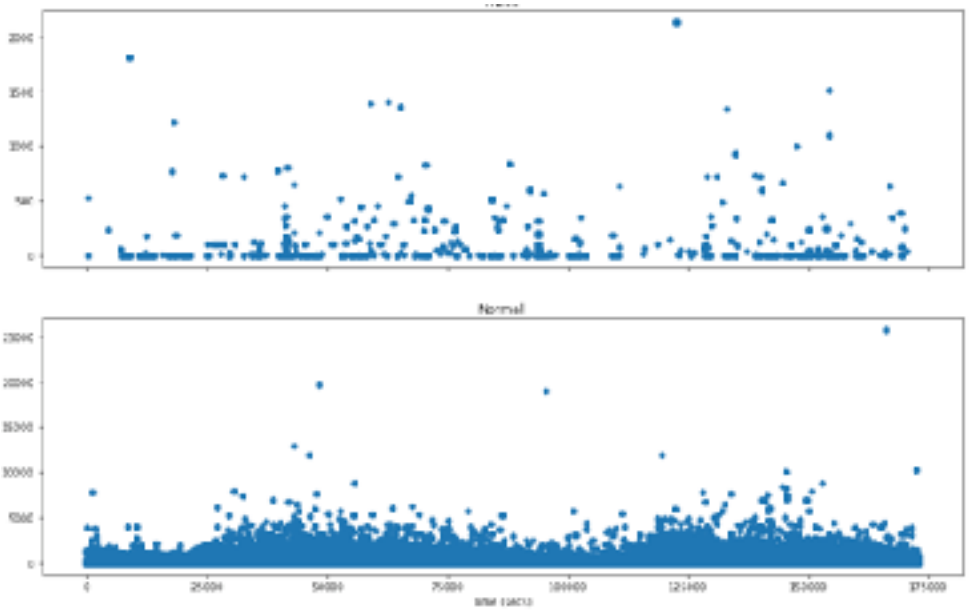
	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.27
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.63
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.77
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.00
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.79
5	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-0.568671	...	-0.208254	-0.55
6	4.0	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213	0.464960	...	-0.167716	-0.27
7	7.0	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.807864	0.615375	...	1.943465	-1.01
8	7.0	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.851084	-0.392048	...	-0.073425	-0.26
9	9.0	-0.338262	1.119593	1.044367	-0.222187	0.499361	-0.246761	0.651583	0.069539	-0.736727	...	-0.246914	-0.63

Looking at raw data, we can see that it has 10 rows × 31 columns. The V_features represent the anonymised data that has been PCA transformed. They seem to be standardised between the values (-1,1).



The above visualisation and statistical description of the data shows that it is totally unbalanced (492 to 284315). This means that we have to be careful when computing a accuracy score because even if the model predicts all of the anomalies as normal (i.e. 492 false positives), the ROC analysis would still output high precision and recall scores. So, we should use resampling (specifically under-sampling) to bring the balance to a 50/50 ratio. Now, lets continue to explore the data and investigate any relationships between the features.

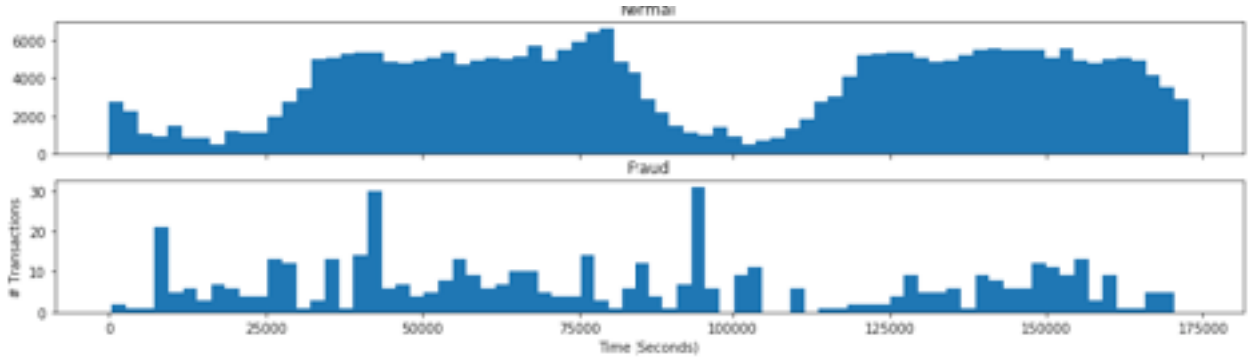
Lets look if there a relationship between time and amount for the two classes. We will plot a scatter plot to do this:



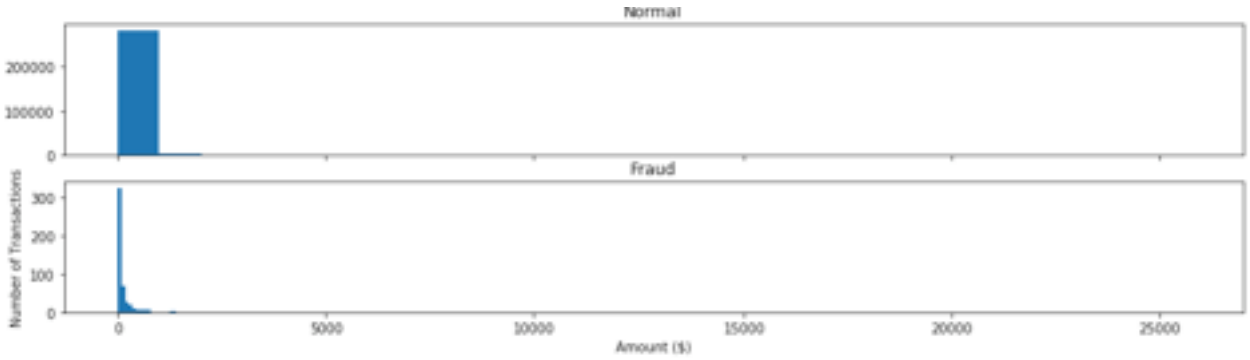
No relationship is apparent. This means that we wont be putting importance in the relationship between the features Time and Amount when building the model.

Now lets look at the frequency of two classes of transaction separately over time: The plot above shows

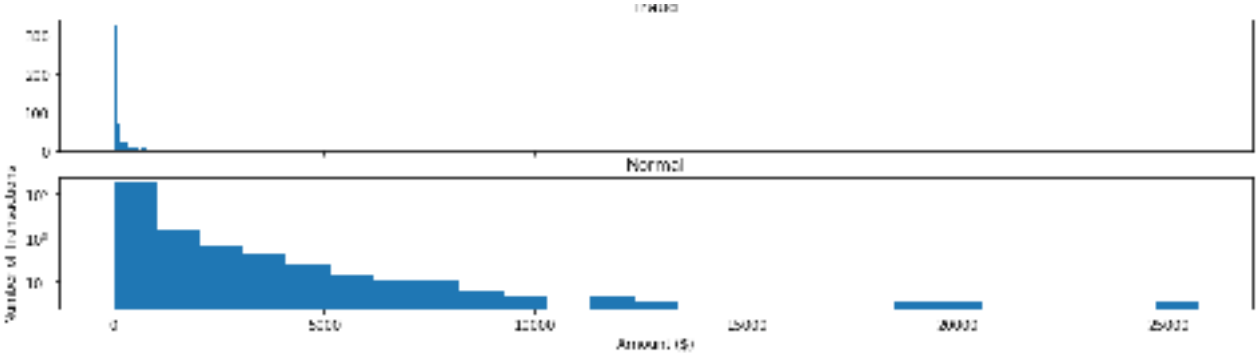
REPORT



there isn't a relationship between time and the frequency of fraudulent/non-fraudulent transactions, except for that normal transactions seems to be cyclical (lows could represent night/ sleeping time or off-peak hours vice versa). So the feature time seems not to be very important for our data analysis.

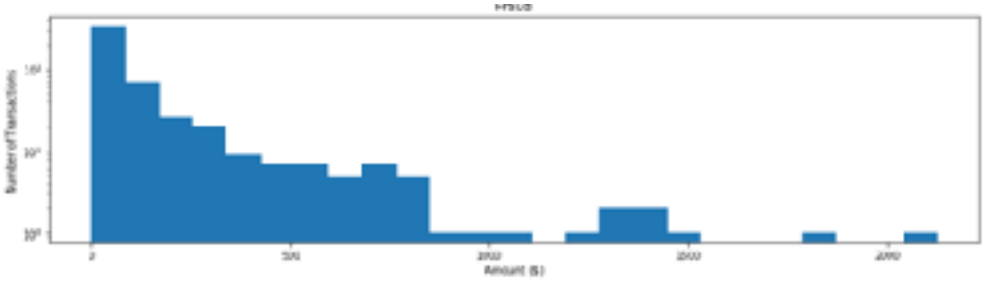


Now, here is a simple histogram plot of dollar-amount of transactions and their frequency. We can see that the y-scale is too large for the small amounts to be clearly visible in the visualisations for the normal transactions, so lets change the scale to logarithmic one from a linear one.



We can now clearly see that there is a relationship between the amount in $ of the transactions and the frequency of non-fraudulent transactions. Although the highest frequency for transactions are both at low amounts for normal and fraudulent transactions, fraudulent transactions seem to have no high-amount transactions while normal transactions do. So this seems to be the distinction amount wise.

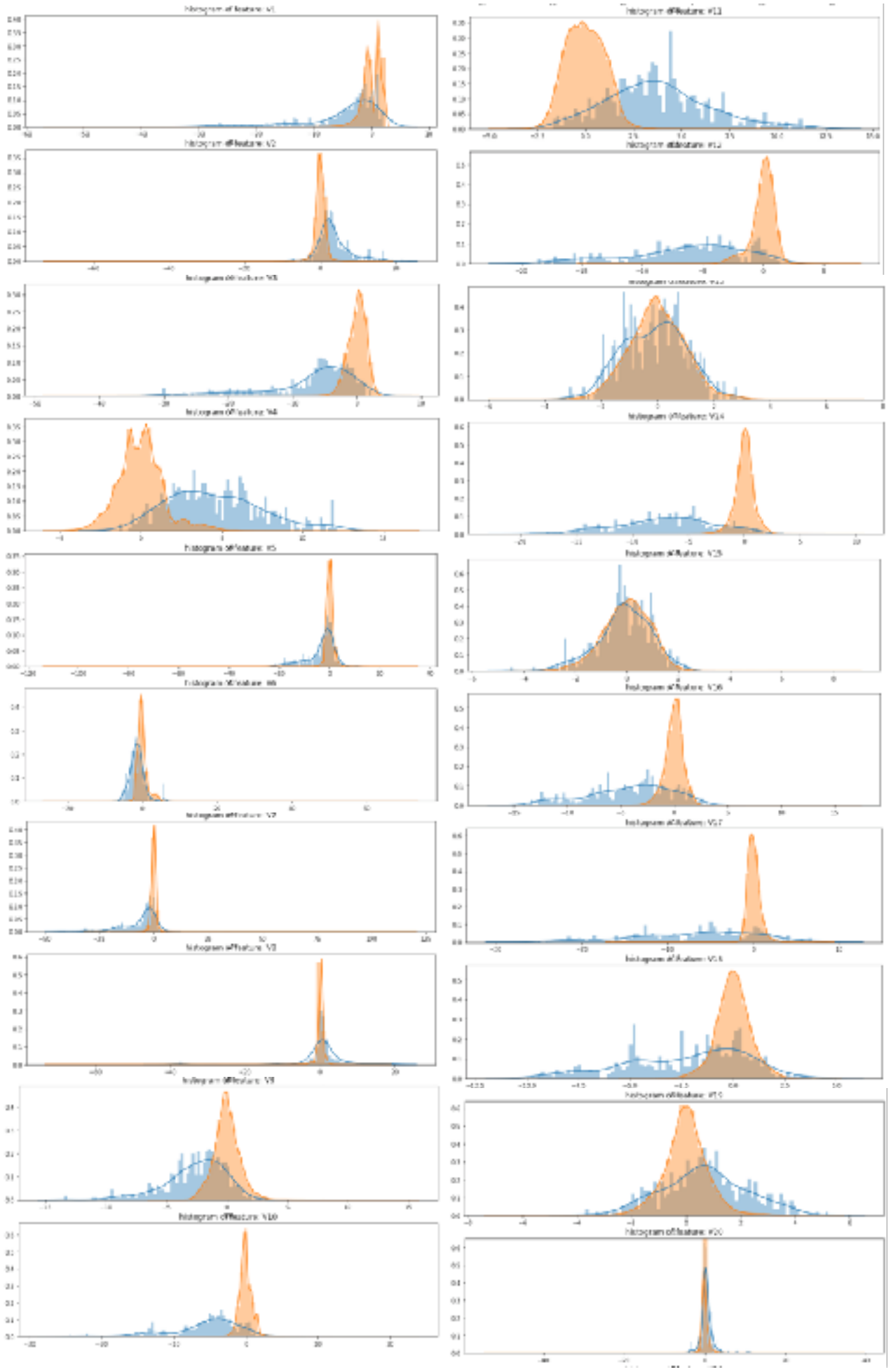
Lets have a closer look at dollar-amount of the fraudulent transactions:



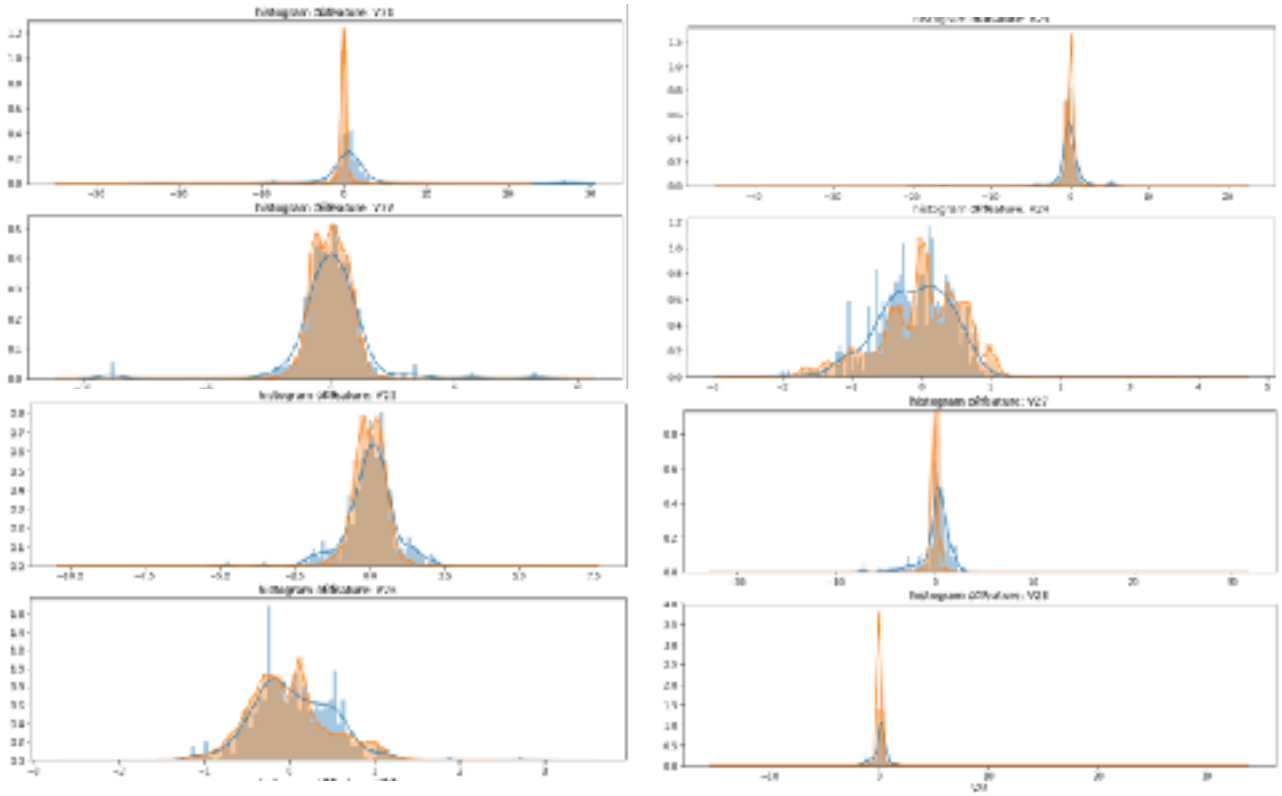
So it is very clear that although there are fraudulent transactions that are big amounts, the bulk of fraudulent transaction are always small amounts. This is a property of the amount feature that might be helpful for us in the future

REPORT

Now lets plot the distributions of the anonymised features. Not all of them might be important for our model, and it is important to remove redundant/unnecessary data out of the sample dataset.



REPORT



By looking at the peaks and the skewness of the distributions for fraud data, we see what features are similar to each other for fraudulent and normal data. This is because the features that are similar would not help us distinguish between fraudulent and normal data when we build our model, so it would be better to take them out as discussed above. Orange plot represents normal transactions and the blue plots represent fraudulent transactions

From these graphs, we can see that we should drop the features V8, V13, V15, V20, V22, V23, V24, V25, V26.

Lets now compute some statistical values for the the columns, grouped by their class,
Mean:

	Time	V1	V2	V3	V4	V5	V6	V7	V9	V10
Class										
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637	0.004467	0.009824
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	-2.581123	-5.676883

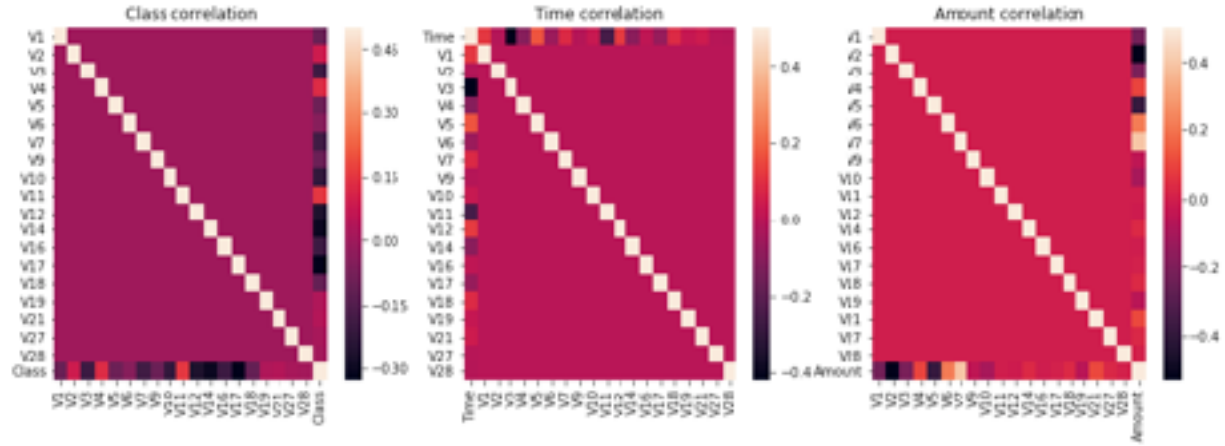
Median:

	Time	V1	V2	V3	V4	V5	V6	V7	V9	V10	...	V1
Class												
0	84711.0	0.020023	0.064070	0.182158	-0.022405	-0.053457	-0.273123	0.041138	-0.049964	-0.091872	...	0.1
1	75568.5	-2.342497	2.717869	-5.075257	4.177147	-1.522962	-1.424616	-3.034402	-2.208768	-4.578825	...	-5.

These don't reveal any differences between the classes except define thresholds of values where one class is more probable than the other. We can use this data to transform into binary values (0,1). Our logistic regressor might fit the binary data better than the PCA transformed features we have now. This is something we will investigate when we build the model.

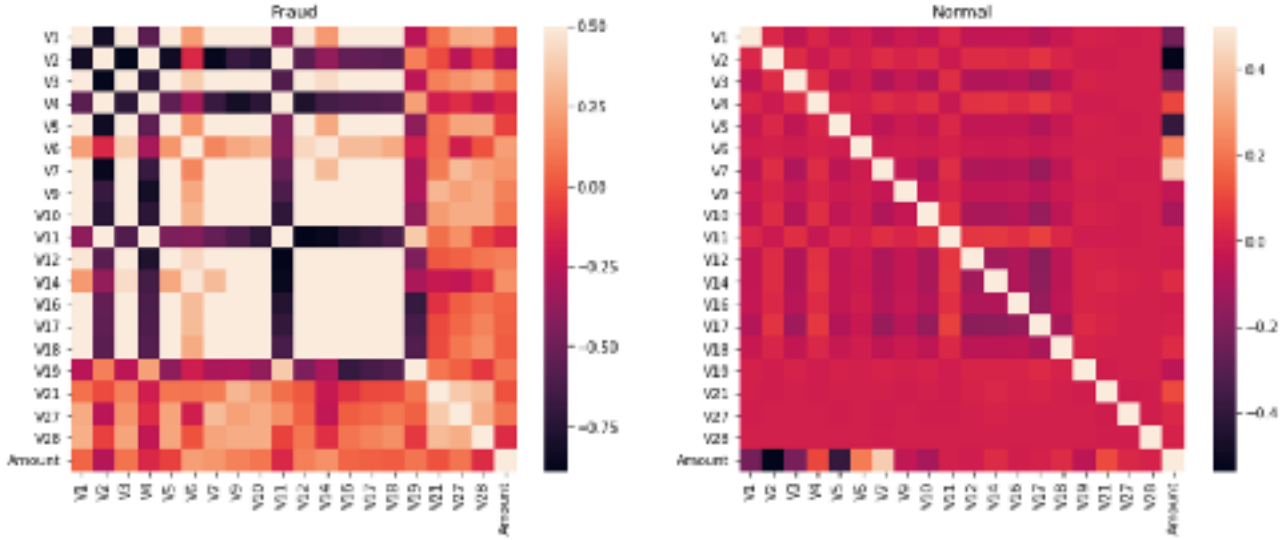
Now lets look at the correlation between features. This is again to investigate which of the features might be unnecessary, so we can drop them and make our model leaner.

REPORT

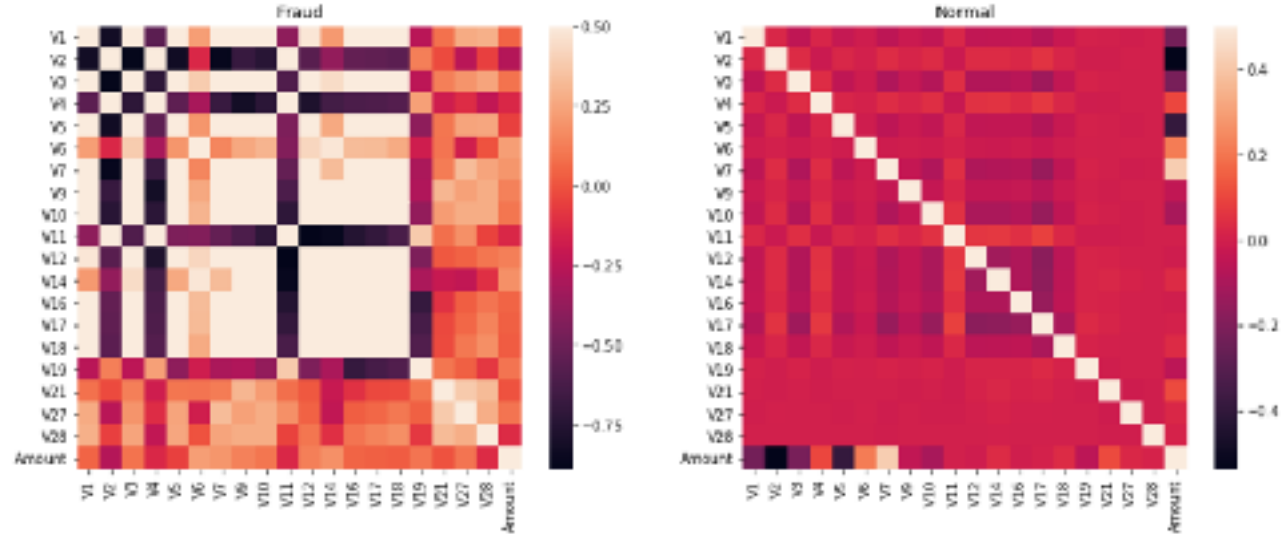


These correlation graphs shows us that amount has a class correlates mostly with the values of the features V1-V18. There is also correlation (although weak) between time and the class until V17. For Amount, there is strong correlation for features and time upto V10 to classify normal data. So it seems that we are going to need Amount more than Time to classify the fraudulent data.

So lets investigate the correlation for amount and the anonymised features:



For Amount, the correlation is only strong for features and V1-V19.



The visualisations above show that for time too, the correlation is strongest features V1:V19 for fraud data. So it look like V1:V19 are the features to investigate. So we are going drop V21, V27, V28.

REPORT

Feature extraction and transformation

Alright lets split the data we are left with into a normal and fraud sample.

```
import random

fraud_sample = CCD3[CCD3.Class==1]
normal_data = CCD3[CCD3.Class==0]

normal_data_indices = normal_data.index.tolist()
normal_data_indices = list(map(lambda x:
random.choice(normal_data_indices), range(len(fraud_sample))))

normal_sample = normal_data.loc[normal_data_indices]

sample = pd.concat([fraud_sample, normal_sample])
```

After splitting the dataset into fraud and normal samples, we make a sample with normal and fraud data split into 50:50 ratio (with all the fraud samples in it). This is to correct the imbalance in the data. This is the result

	Time	V1	V2	V3	V4	V5	V6	V7	V9	V10	V11	V12	V14	
541	406.0	-2.312227	1.951992	-1.609851	3.997906	-0.522188	-1.426545	-2.537387	-2.770089	-2.772272	3.202033	-2.899907	-4.289254	-1
623	472.0	-3.043541	-3.157307	1.088463	2.288644	1.359805	-1.064823	0.325574	-0.270953	-0.838587	-0.414575	-0.503141	-1.692029	0.6
4920	4462.0	-2.303350	1.759247	-0.359745	2.330243	-0.821628	-0.075788	0.562320	-0.238253	-1.525412	2.032912	-6.560124	-1.470102	-2
6108	6986.0	-4.397974	1.358367	-2.592844	2.679787	-1.128131	-1.706536	-3.496197	-0.247768	-4.801637	4.895844	-10.912819	-6.771097	-7
6329	7519.0	1.234235	3.019740	-4.304597	4.732795	3.624201	-1.357746	1.713445	-1.282858	-2.447469	2.101344	-4.609628	-6.079337	2.5

The index are random at the start as I randomly chose 492 (the amount of fraud data sets) items from the normal class, and then added together all of the fraud class to it, creating a new DataFrame 'sample'.

Then, I made a new DataFrame and mapped the values for the V-features with a lambda function that standardised it to binary form, using the means calculated above as a guideline. I didn't add the feature 'Time' to 'sampleT' because we saw that it was as an important for distinguishing between the classes as 'Amount'. This is how I did it:

```
#T stands for transformed
sampleT = pd.DataFrame()

sampleT["V1"] = sample.V1.map(lambda x: 1 if x < -3 else 0)
sampleT["V2"] = sample.V2.map(lambda x: 1 if x > 3 else 0)
sampleT["V3"] = sample.V3.map(lambda x: 1 if x < -3.5 else 0)
sampleT["V4"] = sample.V4.map(lambda x: 1 if x > 2 else 0)
sampleT["V5"] = sample.V5.map(lambda x: 1 if x < -4 else 0)
sampleT["V6"] = sample.V6.map(lambda x: 1 if x < -2.5 else 0)
sampleT["V7"] = sample.V7.map(lambda x: 1 if x < -3 else 0)
sample["V9"] = sample.V9.map(lambda x: 1 if x < -2 else 0)
sampleT["V10"] = sample.V10.map(lambda x: 1 if x < -2.5 else 0)
sampleT["V11"] = sample.V11.map(lambda x: 1 if x > 2 else 0)
sampleT["V12"] = sample.V12.map(lambda x: 1 if x < -2 else 0)
sampleT["V14"] = sample.V14.map(lambda x: 1 if x < -2.5 else 0)
sampleT["V16"] = sample.V16.map(lambda x: 1 if x < --2.5 else 0)
sampleT["V17"] = sample.V17.map(lambda x: 1 if x < -1.5 else 0)
sampleT["V18"] = sample.V18.map(lambda x: 1 if x < -2 else 0)
sampleT["V19"] = sample.V19.map(lambda x: 1 if x > 0.5 else 0)
sampleT["Amount"] = sample.Amount
sampleT["Class"] = sample.Class
```

REPORT

```
#time is not added as we saw it not be relevant before

sampleT.head(10)
```

This was the result:

	V1	V2	V3	V4	V5	V6	V7	V10	V11	V12	V14	V16	V17	V18	V19	Amount	Class
146170	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	7.99	0
15736	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	99.99	1
267984	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	20.14	0
63758	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	35.75	0
250021	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	4.00	0
53901	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	117.70	0
123301	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	1.00	1
15204	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	99.99	1
204079	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	108.51	1
198552	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0.01	0

Building models

Before a trained a model on my data, I needed to split into test and training sets, using the train_test_split method from sklearn:

```
from sklearn.model_selection import train_test_split

X = sample.drop(["Class"],1)
y = sample.Class

X_train, X_test, y_train, y_test = train_test_split(X, y,
train_size = .3, test_size = .7, random_state=42)
```

Model 1: logistic regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import fbeta_score

def test_LR(C_val):
    clf1 = LogisticRegression(C = C_val)
    clf1.fit(X_train, y_train)

    y_pred = clf1.predict(X_test)

    recall = recall_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    F2 = fbeta_score(y_test, y_pred,beta = 2)

    print "recall: ",recall
    print "precision: ",precision
    print "F2: ",F2
```

REPORT

```
    print ""
```

I imported the relevant modules, then defined a function that fit a logistic regression classifier to the sample. The function's parameter was C-value, which was the parameter of the model that I decided to tune. I stored the predictions in the variable y_pred (which is of the type pandas.DataSeries), and then calculate the scores using the library functions from sklearn.metrics. These were the result:

We can see the best result was for C-value 0.5.

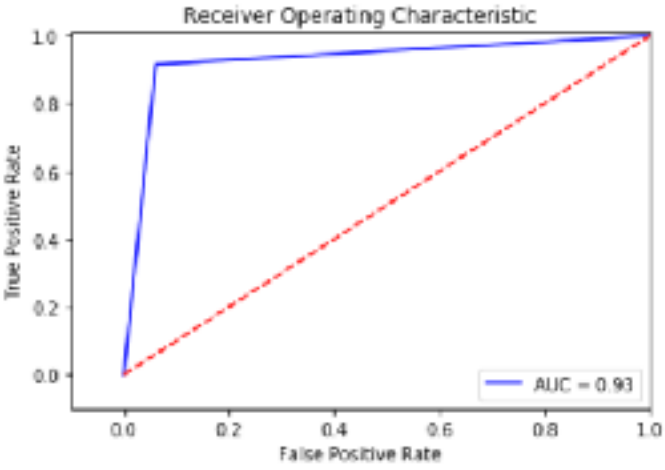
```
for c in [0.001,0.005,.01,0.05,.1,.5,1]:
    print "C value: " + str(c)
    print"_____"
    test_LR(c)
```

```
C value: 0.001

_____
recall:  0.845481049563
precision:  0.932475884244
F2:  0.86155674391

C value: 0.005

_____
recall:  0.883381924198
precision:  0.955835962145
F2:  0.896980461812
C value: 0.05

_____
recall:  0.906705539359
precision:  0.951070336391
F2:  0.91524426133

C value: 0.1

_____
recall:  0.909620991254
precision:  0.948328267477
F2:  0.917107583774

C value: 0.5

_____
recall:  0.918367346939
precision:  0.95166163142
F2:  0.924838520258

C value: 1

_____
recall:  0.918367346939
precision:  0.95166163142
F2:  0.924838520258
```

ROC curves show a trade-off between sensitivity(recall) and specificity(accuracy). From our proposal, we know these were calculated as:

$$R = \frac{TP}{TP+FN} \quad SPC = \frac{TN}{TN+FP}$$

```
from sklearn.metrics import
roc_curve
from sklearn.metrics import auc

clf1 = LogisticRegression(C = 0.5)
clf1.fit(X_train, y_train)

y_pred = clf1.predict(X_test)

fpr, tpr, thresholds =
roc_curve(y_test, y_pred)
roc_auc = auc(fpr,tpr)
```



Plotting the ROC curve, we get an area of curve of 0.93. This is very close to 1, which means that the accuracy of the model its very high. So this model with the parameter C as 0.5 gives a very near perfect test.

REPORT

Model 2: one-class SVM

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
#CCD3['AmountS'] = scaler.fit_transform(CCD3.iloc[:,
16].values.reshape(-1,1))

normal_data = CCD3[CCD3.Class==0].head(10000)
fraud_data = CCD3[CCD3.Class==1]

X = normal_data.drop(["Class", "Time", "Amount"],1)
y = normal_data["Class"]

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = .
3, test_size = .7, random_state=42)

X_test = X_test.append(fraud_data.drop(["Class", "Time", "Amount"],1))

y_test = y_test.append(fraud_data["Class"])

display(X_test.head(5))
display(y_test.head(5))
```

V19	AmountS
22534	0.027682
72775	-0.826465
39367	-0.662582
83232	-2.804819
85474	0.842066

We import the standard scaler model and create a new column in the dataset called AmountS which would be standardised so that each value has a bigger importance to the model. We drop the features Class, Time and Amount(redundant feature) as the one-class SVM takes a very long time to train and so taking out Time would streamline the model without affecting the result of the model as we found it earlier Time is not that integral in classifying if a data is fraudulent or not. The testing and training sets are split so that This is how the AmountS column came out:

Test 1
nu = .95, kernel = 'rbf', gamme = .0001

Now we build the

Test 5
nu = .05, kernel = 'poly', gamma = .0001

One-Class SVM model:

```
from sklearn.svm import OneClassSVM
from itertools import izip

def test_OCSVC(n,k,g):
    clf2 = OneClassSVM(nu = n, kernel = k, gamma = g)
    clf2.fit(X_train)

    y_pred = clf2.predict(X_test)

    binary_mapper = lambda x: 0 if x==-1 else 1
    vfunc = np.vectorize(binary_mapper)
    Y_pred = vfunc(y_pred)

    X_test_result = X_test
    X_test_result["y_test"] = y_test
    X_test_result["Y_pred"] = Y_pred


    tp, fp, fn, tn = 0.0, 0.0, 0.0, 0.0

    for i, row in X_test_result.iterrows():
```

```
        yt = X_test_result.get_value(i,"y_test")
        yp = X_test_result.get_value(i,"Y_pred")

        if yt == 0:
            if yt == yp:
                tp = tp+ 1
            else:
                fn = fn+ 1

        else:
            if yt == yp:
                tn = tn + 1
            else:
                fp = fp+ 1


    print "True pos: %s, true neg: %s, false pos: %s and false neg: %s" %
(tp, tn, fp, fn)
    R = tp / (tp + fn)
    P = tp / (tp + fp)
    F2 = 2*R*P/(P+R)

    print "recall: ", R
    print "precision: ", P
    print "F2: ",F2
    print ""
```

We define the function test_OCSV that fits the X_train set with the parameter passed in with the function. We predict the classes for the test set.We map the result of the prediction (as it is -1 for anomalous data and 1 for normal data) into binary form to be able to compare it to the actual classes Them, the function creates a new DataFrame called X_test_results that is a copy of the testing sample, and then add the y_predictions and the actual result.

We then compare the predictions to the actual classes, and compute the number of true positives, true negatives, false positives and false negatives. With these, the function compute the recall, precision and F2 scores. Here are the results for the different tests I ran:

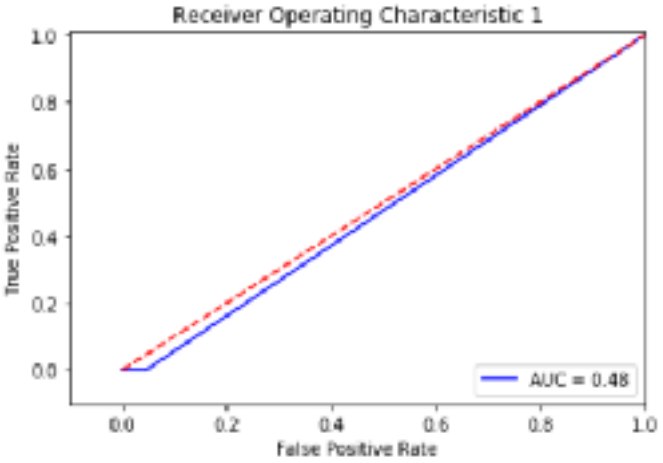
```
test_OCSVC(n = .95, k = 'rbf', g = .0001)
True pos: 6678.0, true neg: 0.0, false pos: 492.0 and false neg: 322.0
recall:  0.954
precision:  0.931380753138
F2:  0.942554693013

test_OCSVC(n = .15, k = 'linear', g = .1)
True pos: 1482.0, true neg: 34.0, false pos: 458.0 and false neg: 5518.0
recall:  0.211714285714
precision:  0.763917525773
F2:  0.331543624161

test_OCSVC(n = .05, k = 'linear', g = .0001)
True pos: 3851.0, true neg: 107.0, false pos: 385.0 and false neg: 3149.0
recall:  0.550142857143
precision:  0.909112370161
F2:  0.685475258099

test_OCSVC(n = .95, k = 'poly', g = .0001)
True pos: 4628.0, true neg: 332.0, false pos: 160.0 and false neg: 2372.0
recall:  0.661142857143
precision:  0.966583124478
```

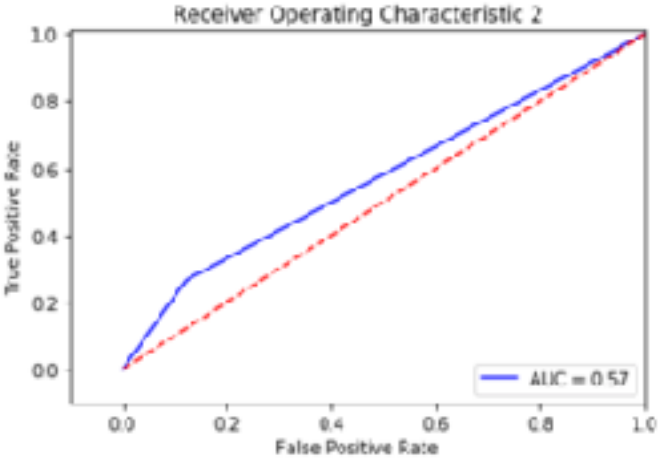
```
REPORT

F2:  0.785205293519
```

```
test_OCSVC(n = .05, k = 'poly', g = .0001)
True pos: 6141.0, true neg: 131.0, false pos: 361.0 and false neg: 859.0
recall:  0.877285714286
precision:  0.944478621962
F2:  0.90964301585
```

The results show that the model's predictions are not very accurate. Although for the first test we get very high precision, recall and F2 scores, we see that the number of false positives was 492, which meant that all of the data that was fraudulent got classified as normal by the model. It is ver important that the model is able to classify as many fraudulent transactions as such as possible. So the first test is of the model underperforms significantly, and this will be apparent when we plot a ROC curve for it.

I experimented with some more iterations of the parameters and the above shows the model was not able to perform to the standards we need, no matter the type of kernel. So overall, one-class SVC is not a good method to use for this sample. For further evidence, I plotted the ROC curve for the two best performing tests. We can see that the AUC calculated area underneath the ROC curve are (very) low for both tests as they are close 0.5 (which represents a random classifier). So, the first test of the model is worse than a random classifier. Test 5 is slightly better.





Benchmark models

Here is a project on Kaggle on the same dataset: https://www.kaggle.com/kamathhrishi/detecting-credit-card-frauds. It didn't use the same supervised learning method, but it got these results:

Support Vector Machine

```
clf_1=svm.SVC(kernel='linear')
clf_1.fit(x_train,y_train)
print(classification_report(y_test,clf_1.predict(x_test)))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	24931
1	0.64	0.67	0.65	69

For the SVC classifier, on average its recall and precision was lower than my supervised learning model.

Random Forest Classifier

```
clf = RandomForestClassifier(max_depth=2, random_state=0)
clf.fit(x_train, y_train)
print(classification_report(y_test,clf.predict(x_test)))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     24911
           1       0.73      0.78      0.76        89

   avg / total       1.00      1.00      1.00     25000
```

And the same applied for the random forest classifier

This is probably due to my parameter tuning and feature extraction, which this project didn't undertake. The parameter tuning let me improve my model to achieve better scores.

Now, this project did build a one-class SVM model. These were the results:

```
print("Training: One Class SVM (RBF) : ",(Train_Accuracy(train_AD)),"%")
print("Test: One Class SVM (RBF) : ",(Test_Accuracy(test_AD)),"%")
```

```
Training: One Class SVM (RBF) :  32.90606005897575 %
Test: One Class SVM (RBF) :  100.0 %
```

```
print("Training: One Class SVM (Linear) : ",(Train_Accuracy(train_AD_L)),"%")
print("Test: One Class SVM (Linear) : ",(Test_Accuracy(test_AD_L)),"%")
```

```
Training: One Class SVM (Linear) :  89.9981946199675 %
Test: One Class SVM (Linear) :  2.027027027027027 %
```

For the RBF, we can see that although it cannot identify normal data well (training data), it does well with classifying fraudulent data (test). So this one-class sac model with 'rbf' kernel does better at detecting fraud than that of mine. However, the linear kernel, although effective in identifying normal data, is much worse in identifying fraudulent data than the iterations of my OCSVM with linear kernel.

So comparing my models to the benchmarks, we can see that my supervised learning model perform well, while the anomaly detection model varied in its performance in different aspects compared to the benchmark model. We can see that one-class svm is not really suited for this dataset. It might have worked better if there were fewer features, and if I had used more of the complete data sets (more transactions). But this would have increased the running time, so it would be a very slow model. So the best strategy to use with labelled data, as implied by results, would be to use a supervised learning method when there is labelled data. This goes against my earlier hypothesis that an anomaly detection method might be more effective.

REPORT

Conclusion

Process:

1. Imported the data from a csv file into pandas DataFrame and called it CCD (stands for **C**redit **C**ard **D**ata)
2. Performed simple statistical analysis on the data
3. Started data analysis with plotting various histograms by isolating the features amount and time. Looked for relationships between these features and the class of the transaction.
4. Looked at the distributions of the scaled features (V1-V29) to see which were irrelevant to the class of a transaction
5. Dropped the irrelevant features and stored the remaining data in a new DataFrame called CCD2
6. Calculate the mean and median of the data so they can be useful when finding my thresholds when binarising the data
7. Now look for correlation, do analysis and drop features that do no help in identifying the class of a transactions
8. Undersampled the data so the ratio of fraud to normal transaction is 50:50
9. Binarised the data
10. Build the logistic regression model. But did not use binary data after researching that it might be not that effective for logistic regression, but for models based on methods like neural networks instead.
11. Tested alternate values for C (parameter for logistic regression)
12. Plot ROC curve
13. Create training sets and testing sets for one-class svm
14. Created a testing function for OCSVM, which accepted the three parameters that were being tunes
15. Ran 5 tests for the model, displayed ROC analysis scores
16. Plotted the ROC curve for two best performing tests.

The interesting part of the project was the data analysis.This is because some of the relationships I discovered were surprising and I was surprised to not find some. For example, for frequency of fraud data across time, I expected the volume of fraud transactions to occur during day time when there was many transactions taking place so that the fraudulent transactions is masked among the huge volume of transactions.

The difficult aspect of the project was keeping track of all the changes in data each time I transformed the features or deleted some columns. It was very important to make sure I knew exactly what values and what type of data I was dealing and to that I had to name variables so that they were easy to remember and make sure the code was structured neatly.

My final solution for this dataset would be to use a supervised learning method as explained before. But I think there are betterI think I might be able to improve my model by using a neural network (a multilayer feed-forward network to be precise) instead of a one-class SVM. Neural networks (work faster than SVM, so I could have trained my model in a reasonable time over the whole sample instead of training a part like I had to do with one-class SVM, hence getting a better output.

There were models that I researched that might have worked better, but I did not know how to implement. There was a method called local outlier factor (LOF). This is a outlier detection method. I did not know how to implement this model, and wasn't sure if would be suitable for the data I had. But, as it was a clustering model, I would have tested on the data if I knew how to implement LOF, to see if the data easily grouped into normal transactions and outliers (fraud). So although I might not have made LOF my final model, it would have been very helpful to see how simple clustering works on the data.