Credit Fraud Detection

May 11, 2017

1 CREDIT CARD FRAUD DETECTION

This following notebook will help us analyze the Credit Card Fraud Detection Classes and the following models will be used to test the accuracy of fraudulent transactions.

1. Random Forest Classifier
2. Decision Tree Classifier (CART)
3. XG Boost Algorithm

```
In [26]: #Importing Libraries
         import pandas as pd
         import matplotlib.pyplot as plt
         import numpy as np
         from sklearn.model_selection import train_test_split
         from sklearn.ensemble import RandomForestClassifier
         from IPython.display import display #import display for DataFrame usage
         from sklearn.metrics import confusion_matrix
         import itertools
         import collections
         from sklearn.preprocessing import normalize
         from sklearn import tree
         import seaborn as sns
         from sklearn.preprocessing import StandardScaler
         from sklearn.metrics import precision_recall_curve, auc, confusion_matrix,
         from sklearn.model_selection import train_test_split
         from sklearn.linear_model import LogisticRegression
         from sklearn.ensemble import ExtraTreesClassifier
         from sklearn.ensemble import RandomForestClassifier
         import xgboost as xgb
         from subprocess import check_output

         %matplotlib inline
```

2 Data Input

```
In [27]: data = pd.read_csv("E:/School/Sem 2/Knowledge Discovery in Databases/Final
         data.head()
```

Out[27]:
```
    Time        V1        V2        V3        V4        V5        V6        
0    0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239
1    0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078
2    1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791
3    1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237
4    2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592

        V8        V9  ...       V21       V22       V23       V24  \
0  0.098698  0.363787  ...  -0.018307  0.277838 -0.110474  0.066928
1  0.085102 -0.255425  ...  -0.225775 -0.638672  0.101288 -0.339846
2  0.247676 -1.514654  ...   0.247998  0.771679  0.909412 -0.689281
3  0.377436 -1.387024  ...  -0.108300  0.005274 -0.190321 -1.175575
4 -0.270533  0.817739  ...  -0.009431  0.798278 -0.137458  0.141267

        V25       V26       V27       V28  Amount  Class
0  0.128539 -0.189115  0.133558 -0.021053  149.62      0
1  0.167170  0.125895 -0.008983  0.014724    2.69      0
2 -0.327642 -0.139097 -0.055353 -0.059752  378.66      0
3  0.647376 -0.221929  0.062723  0.061458  123.50      0
4 -0.206010  0.502292  0.219422  0.215153   69.99      0

[5 rows x 31 columns]
```

3 Assesment of the Target Class

In [28]:
```python
count_classes = pd.value_counts(data['Class'], sort = True).sort_index()
count_classes.plot(kind = 'bar')
plt.title("Fraud class histogram")
plt.xlabel("Class")
plt.ylabel("Frequency")
```

Out[28]:
```
<matplotlib.text.Text at 0x15f2d08c518>
```

- First Pass: Random Forest with all columns

```
In [29]: data_class_outcomes = data['Class']
         #preserving only necessary columns
         data.drop(['Class'], axis = 1, inplace = True)
```

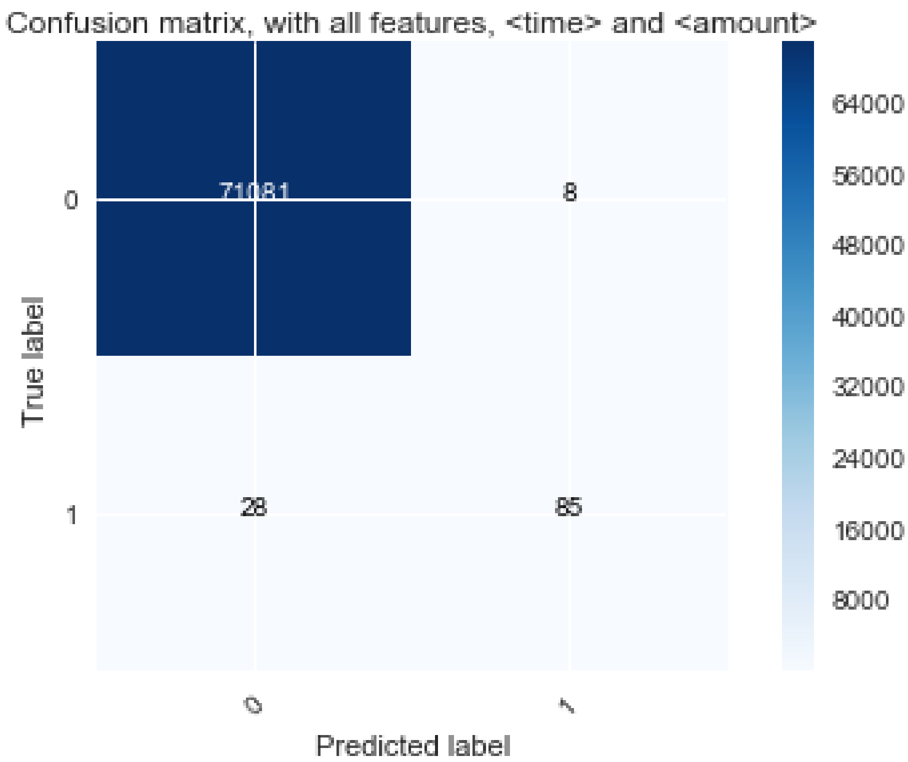
```
In [30]: #import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(data,data_class_outcom
         print("Training and testing split was successful.")
```

```
Training and testing split was successful.
```

```
In [31]: #Classifier = RFC
         def implement_rfc(X_train,y_train,X_test):
             """
             This function fits and transforms data using
             Random Forest Classifier technique and
             returns the y_pred value
             """
             clf_B = RandomForestClassifier(n_estimators=98)
             clf_B.fit(X_train, y_train)
             y_pred = clf_B.predict(X_test)
             return y_pred

         y_pred = implement_rfc(X_train,y_train,X_test)
```

```
In [32]: def calculate_confusion_matrix(y_test, y_pred):
             return confusion_matrix(y_test, y_pred)

         result_confusion_matrix = calculate_confusion_matrix(y_test, y_pred)

         def plot_confusion_matrix(cm, classes,
                                   normalize=False,
                                   title='Confusion matrix',
                                   cmap=plt.cm.Blues):
             """
             This function prints and plots the confusion matrix.
             Normalization can be applied by setting `normalize=True`.
             """
             plt.imshow(cm, interpolation='nearest', cmap=cmap)
             plt.title(title)
             plt.colorbar()
             tick_marks = np.arange(len(classes))
             plt.xticks(tick_marks, classes, rotation=45)
             plt.yticks(tick_marks, classes)

             if normalize:
                 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
                 print("Normalized confusion matrix")
             else:
                 print('Confusion matrix, without normalization')

             print(cm)

             thresh = cm.max() / 2.
             for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
                 plt.text(j, i, cm[i, j],
                          horizontalalignment="center",
                          color="white" if cm[i, j] > thresh else "black")

             plt.tight_layout()
             plt.ylabel('True label')
             plt.xlabel('Predicted label')

         class_names = [0,1]
         plot_confusion_matrix(result_confusion_matrix, classes=class_names,title='
```

```
Confusion matrix, without normalization
[[71081     8]
 [   28    85]]
```




```
In [33]: def calculate_add_scores(confusion_matrix,Classifier="RFC"):
             TP = confusion_matrix[0][0]
             FP = confusion_matrix[0][1]
             FN = confusion_matrix[1][0]
             TN = confusion_matrix[1][1]
             accuracy = (TP+TN)/(TP+FP+FN+TN)
             precision = (TP/TP+FP)
             recall = (TP/TP+FN)
             values = [{'Classifier':Classifier,'Accuracy':accuracy,'Precision':pre
                       'Recall':recall}]
             dataframe = pd.DataFrame(values,columns=values[0].keys())
             return dataframe

         df = calculate_add_scores(result_confusion_matrix)
         print(df)
```

```
   Precision Classifier  Recall  Accuracy
0        9.0        RFC    29.0  0.999494
```

- Second Pass: Random Forest on dropping column 'TIME'

```
In [34]: data_time_outcomes = data['Time']
         #preserving only necessary columns, dropping 'Time'
         data.drop(['Time'], axis = 1, inplace = True)
```

```
In [35]: data.describe()
```

```
Out[35]:                  V1            V2            V3            V4            V
         count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+0
         mean   3.919560e-15  5.688174e-16 -8.769071e-15  2.782312e-15 -1.552563e-1
         std    1.958696e+00  1.651309e+00  1.516255e+00  1.415869e+00  1.380247e+0
         min   -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00 -1.137433e+0
         25%   -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01 -6.915971e-0
         50%    1.810880e-02  6.548556e-02  1.798463e-01 -1.984653e-02 -5.433583e-0
         75%    1.315642e+00  8.037239e-01  1.027196e+00  7.433413e-01  6.119264e-0
         max    2.454930e+00  2.205773e+01  9.382558e+00  1.687534e+01  3.480167e+0

                          V6            V7            V8            V9           V1
         count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+0
         mean   2.010663e-15 -1.694249e-15 -1.927028e-16 -3.137024e-15  1.768627e-1
         std    1.332271e+00  1.237094e+00  1.194353e+00  1.098632e+00  1.088850e+0
         min   -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+01 -2.458826e+0
         25%   -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-01 -5.354257e-0
         50%   -2.741871e-01  4.010308e-02  2.235804e-02 -5.142873e-02 -9.291738e-0
         75%    3.985649e-01  5.704361e-01  3.273459e-01  5.971390e-01  4.539234e-0
         max    7.330163e+01  1.205895e+02  2.000721e+01  1.559499e+01  2.374514e+0

                    ...               V20           V21           V22            V
         count      ...      2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+
         mean       ...      5.085503e-16  1.537294e-16  7.959909e-16  5.367590e-
         std        ...      7.709250e-01  7.345240e-01  7.257016e-01  6.244603e-
         min        ...     -5.449772e+01 -3.483038e+01 -1.093314e+01 -4.480774e+
         25%        ...     -2.117214e-01 -2.283949e-01 -5.423504e-01 -1.618463e-
         50%        ...     -6.248109e-02 -2.945017e-02  6.781943e-03 -1.119293e-
         75%        ...      1.330408e-01  1.863772e-01  5.285536e-01  1.476421e-
         max        ...      3.942090e+01  2.720284e+01  1.050309e+01  2.252841e+

                         V24           V25           V26           V27           V2
         count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+0
         mean   4.458112e-15  1.453003e-15  1.699104e-15 -3.660161e-16 -1.206049e-1
         std    6.056471e-01  5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-0
         min   -2.836627e+00 -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+0
         25%   -3.545861e-01 -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-0
         50%    4.097606e-02  1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-0
         75%    4.395266e-01  3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-0
         max    4.584549e+00  7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+0

                       Amount
         count  284807.000000
```

```
mean        88.349619
std        250.120109
min          0.000000
25%          5.600000
50%         22.000000
75%         77.165000
max      25691.160000

[8 rows x 29 columns]
```

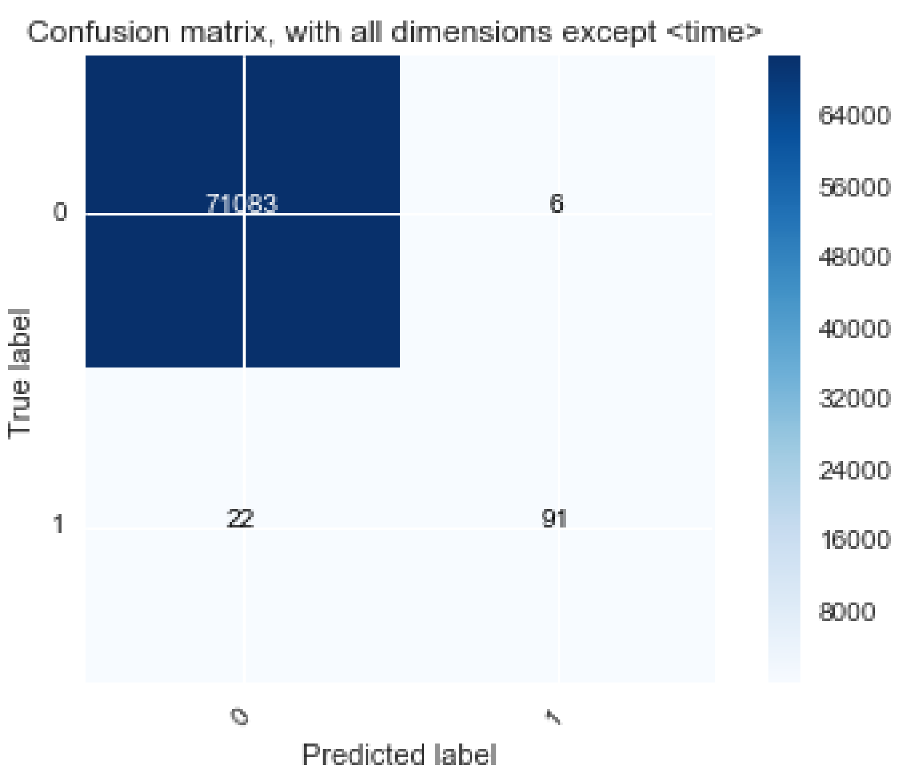
In [36]:
```
#import train_test split
X_train, X_test, y_train, y_test = train_test_split(data,data_class_outcom
print("Training and testing split was successful.")
```

```
Training and testing split was successful.
```

In [37]:
```
y_pred = implement_rfc(X_train,y_train,X_test)
```

In [38]:
```
confusion_matrix_1 = calculate_confusion_matrix(y_test,y_pred)
class_names = [0,1]
plot_confusion_matrix(confusion_matrix_1, normalize=False, classes=class_n
                      title='Confusion matrix, with all dimensions except
```

```
Confusion matrix, without normalization
[[71083     6]
 [   22    91]]
```



```
In [39]: df1 = calculate_add_scores(confusion_matrix_1)
         frames = [df,df1]
         df = pd.concat(frames)
         print(df)
```

```
   Precision Classifier  Recall  Accuracy
0        9.0        RFC    29.0  0.999494
0        7.0        RFC    23.0  0.999607
```

- Pass 3: Random Forest on dropping both 'Time' & 'Amount', preserving only features

```
In [40]: data_amount_outcomes = data['Amount']
         data.drop(['Amount'], axis = 1, inplace = True)
```

```
In [41]: display(data.describe())
```

```
                 V1            V2            V3            V4            V5  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   3.919560e-15  5.688174e-16 -8.769071e-15  2.782312e-15 -1.552563e-15
std    1.958696e+00  1.651309e+00  1.516255e+00  1.415869e+00  1.380247e+00
min   -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00 -1.137433e+02
25%   -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01 -6.915971e-01
50%    1.810880e-02  6.548556e-02  1.798463e-01 -1.984653e-02 -5.433583e-02
75%    1.315642e+00  8.037239e-01  1.027196e+00  7.433413e-01  6.119264e-01
max    2.454930e+00  2.205773e+01  9.382558e+00  1.687534e+01  3.480167e+01

                 V6            V7            V8            V9           V10  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   2.010663e-15 -1.694249e-15 -1.927028e-16 -3.137024e-15  1.768627e-15
std    1.332271e+00  1.237094e+00  1.194353e+00  1.098632e+00  1.088850e+00
min   -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+01 -2.458826e+01
25%   -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-01 -5.354257e-01
50%   -2.741871e-01  4.010308e-02  2.235804e-02 -5.142873e-02 -9.291738e-02
75%    3.985649e-01  5.704361e-01  3.273459e-01  5.971390e-01  4.539234e-01
max    7.330163e+01  1.205895e+02  2.000721e+01  1.559499e+01  2.374514e+01

           ...                V19           V20           V21           V22  \
count      ...       2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean       ...       9.049732e-16  5.085503e-16  1.537294e-16  7.959909e-16
std        ...       8.140405e-01  7.709250e-01  7.345240e-01  7.257016e-01
min        ...      -7.213527e+00 -5.449772e+01 -3.483038e+01 -1.093314e+01
25%        ...      -4.562989e-01 -2.117214e-01 -2.283949e-01 -5.423504e-01
50%        ...       3.734823e-03 -6.248109e-02 -2.945017e-02  6.781943e-03
75%        ...       4.589494e-01  1.330408e-01  1.863772e-01  5.285536e-01
max        ...       5.591971e+00  3.942090e+01  2.720284e+01  1.050309e+01
```

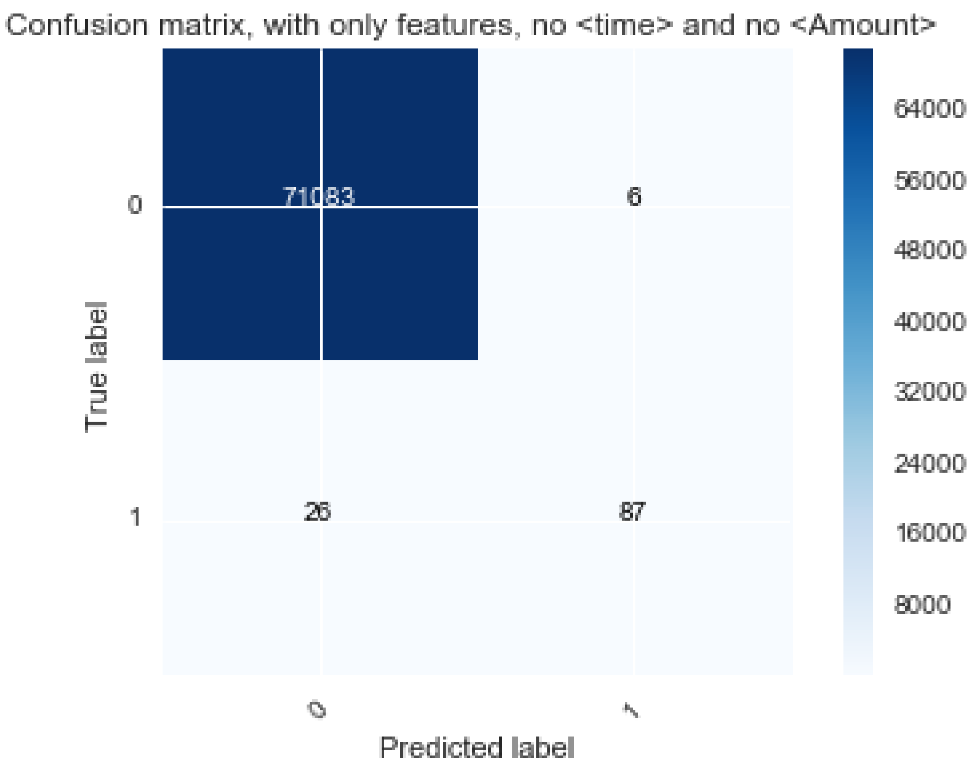
```
                V23           V24           V25           V26           V27  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   5.367590e-16  4.458112e-15  1.453003e-15  1.699104e-15 -3.660161e-16
std    6.244603e-01  6.056471e-01  5.212781e-01  4.822270e-01  4.036325e-01
min   -4.480774e+01 -2.836627e+00 -1.029540e+01 -2.604551e+00 -2.256568e+01
25%   -1.618463e-01 -3.545861e-01 -3.171451e-01 -3.269839e-01 -7.083953e-02
50%   -1.119293e-02  4.097606e-02  1.659350e-02 -5.213911e-02  1.342146e-03
75%    1.476421e-01  4.395266e-01  3.507156e-01  2.409522e-01  9.104512e-02
max    2.252841e+01  4.584549e+00  7.519589e+00  3.517346e+00  3.161220e+01

                V28
count  2.848070e+05
mean  -1.206049e-16
std    3.300833e-01
min   -1.543008e+01
25%   -5.295979e-02
50%    1.124383e-02
75%    7.827995e-02
max    3.384781e+01

[8 rows x 28 columns]
```

```
In [42]: #import train_test split
         X_train, X_test, y_train, y_test = train_test_split(data,data_class_outcom
         print("Training and testing split was successful.")
```

```
Training and testing split was successful.
```

```
In [43]: y_pred = implement_rfc(X_train,y_train,X_test)
         confusion_matrix_2 = calculate_confusion_matrix(y_test,y_pred)
         class_names = [0,1]
         plot_confusion_matrix(confusion_matrix_2, normalize=False, classes=class_n
                               title='Confusion matrix, with only features, no <tim
```

```
Confusion matrix, without normalization
[[71083     6]
 [   26    87]]
```




```
In [44]: df2 = calculate_add_scores(confusion_matrix_2)
         frames = [df,df2]
         df = pd.concat(frames)
         print(df)
```

```
   Precision Classifier  Recall  Accuracy
0        9.0        RFC    29.0  0.999494
0        7.0        RFC    23.0  0.999607
0        7.0        RFC    27.0  0.999551
```

- Now the data is normalized to check accuracy after Data Handling

```
In [45]: normalize_array = normalize(data_amount_outcomes.values.reshape(1,-1))
```

- Pass 4: Random Forest with all features, no 'Time' and Normalized 'Amount'

```
In [46]: #Concatenate data using Numpy
         new_data = np.concatenate((data, normalize_array.T), axis=1)
```

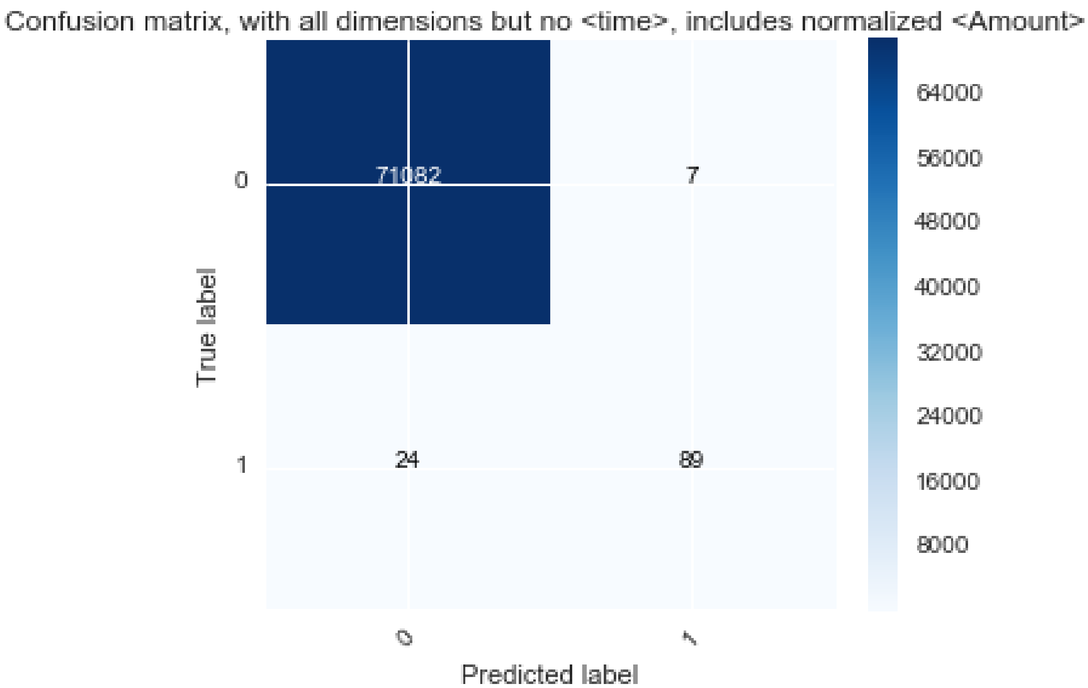
- Training set = 75%, Test set = 25%

```
In [47]: #import train_test split
         X_train, X_test, y_train, y_test = train_test_split(data,data_class_outcom
         print("Training and testing split was successful.")
```

```
Training and testing split was successful.
```

```
In [48]: clf = RandomForestClassifier(n_estimators=98)
         clf.fit(X_train, y_train)
         y_pred = clf.predict(X_test)
         confusion_matrix_3 = calculate_confusion_matrix(y_test,y_pred)
         class_names = [0,1]
         plot_confusion_matrix(confusion_matrix_3, normalize=False, classes=class_r
                               title='Confusion matrix, with all dimensions but no
```

```
Confusion matrix, without normalization
[[71082     7]
 [   24    89]]
```



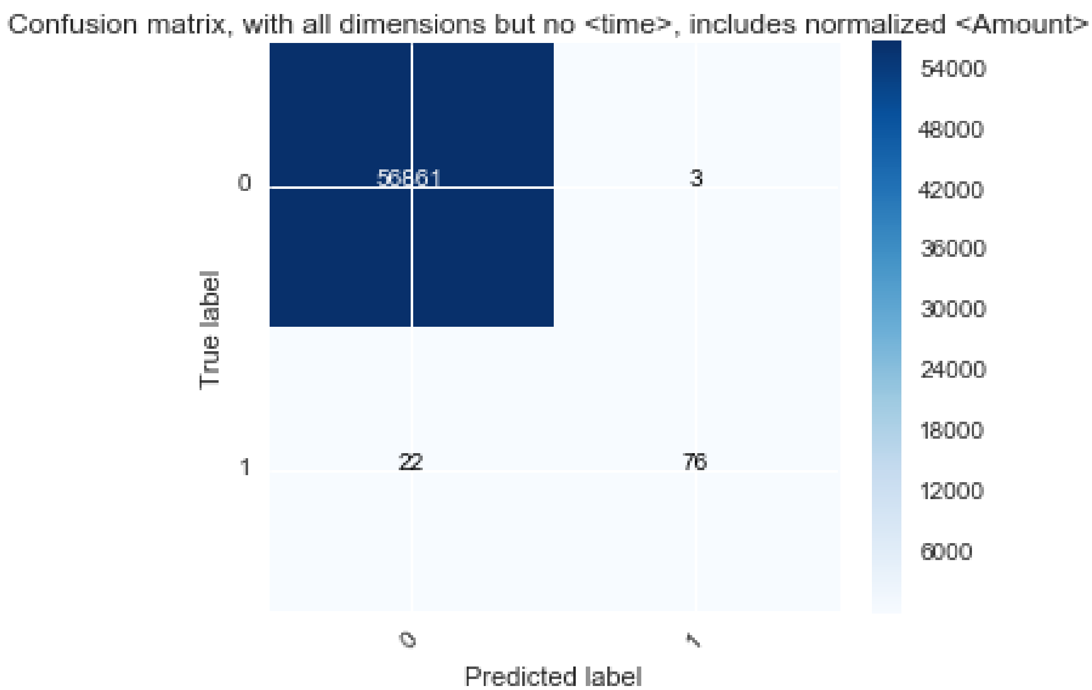

```
In [49]: df3 = calculate_add_scores(confusion_matrix_3)
         frames = [df,df3]
         df = pd.concat(frames)
         print(df)
```

```
   Precision Classifier  Recall  Accuracy
0        9.0        RFC    29.0  0.999494
0        7.0        RFC    23.0  0.999607
0        7.0        RFC    27.0  0.999551
0        8.0        RFC    25.0  0.999565
```

- Pass 5: Random Forest with all features, no 'Time' and Normalized 'Amount'
- Training set = 80%, Test set = 20%

```
In [96]: #try 2 with different parameters
         X_train, X_test, y_train, y_test = train_test_split(new_data,data_class_ou
         print("Training and testing split was successful.")
         clf = RandomForestClassifier(n_estimators=98)
         clf.fit(X_train, y_train)
         y_pred = clf.predict(X_test)
         confusion_matrix_4 = calculate_confusion_matrix(y_test,y_pred)
         class_names = [0,1]
         plot_confusion_matrix(confusion_matrix_4, normalize=False, classes=class_n
                               title='Confusion matrix, with all dimensions but no
```

```
Training and testing split was successful.
Confusion matrix, without normalization
[[56861     3]
 [   22    76]]
```




```
In [95]: importance = clf.feature_importances_
         print(importance)
```

```
[ 0.01374616  0.011479    0.01635103  0.02860632  0.00992177  0.01555839
  0.02423286  0.01085213  0.02879264  0.09852281  0.06454834  0.14500474
```

```
  0.01094227  0.10862927  0.01179022  0.07318633  0.17274959  0.02214618
  0.01312284  0.01131745  0.01777319  0.00941532  0.0076758   0.01199363
  0.01003748  0.018512    0.01183161  0.00929684  0.01196379]
```

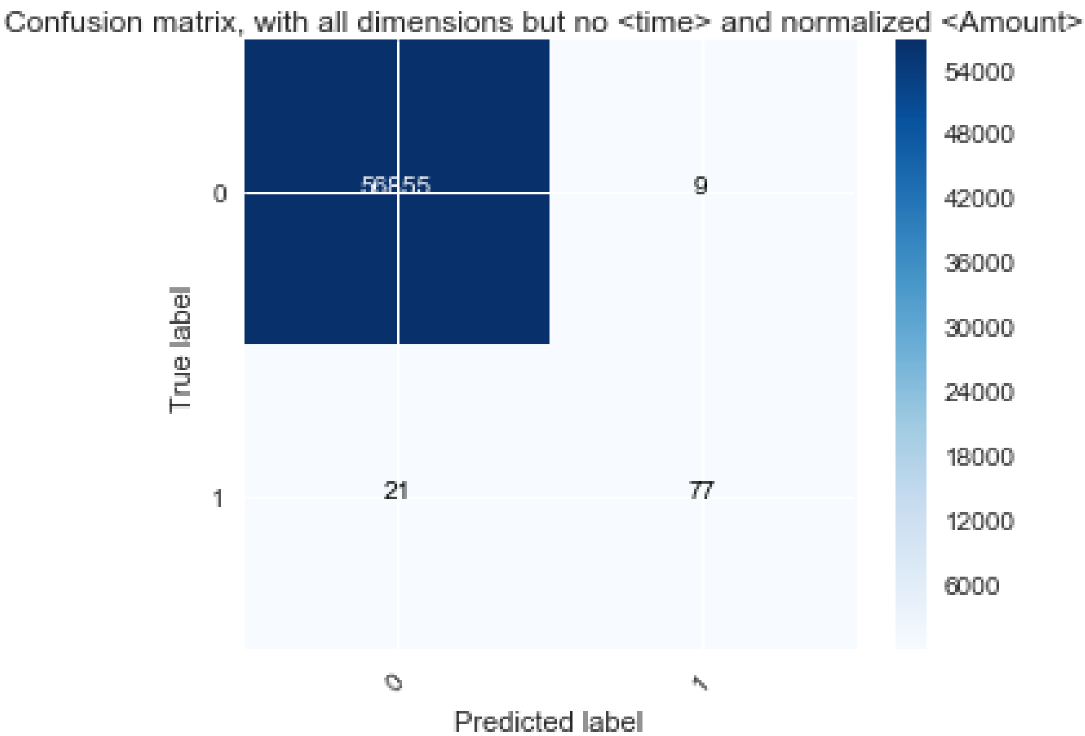
```
In [51]: df4 = calculate_add_scores(confusion_matrix_4)
         frames = [df,df4]
         df = pd.concat(frames)
         print(df)
```

```
   Precision Classifier  Recall  Accuracy
0        9.0        RFC    29.0  0.999494
0        7.0        RFC    23.0  0.999607
0        7.0        RFC    27.0  0.999551
0        8.0        RFC    25.0  0.999565
0        3.0        RFC    23.0  0.999579
```

- Decision Tree Classifier with Max_Depth = 6

```
In [52]: X_train, X_test, y_train, y_test = train_test_split(new_data,data_class_ou
         print("Training and testing split was successful.")
         clf = tree.DecisionTreeClassifier(random_state=42,max_depth=6)
         clf.fit(X_train, y_train)
         y_pred = clf.predict(X_test)
         confusion_matrix_4 = calculate_confusion_matrix(y_test,y_pred)
         class_names = [0,1]
         plot_confusion_matrix(confusion_matrix_4, normalize=False, classes=class_r
                               title='Confusion matrix, with all dimensions but no
```

```
Training and testing split was successful.
Confusion matrix, without normalization
[[56855     9]
 [   21    77]]
```



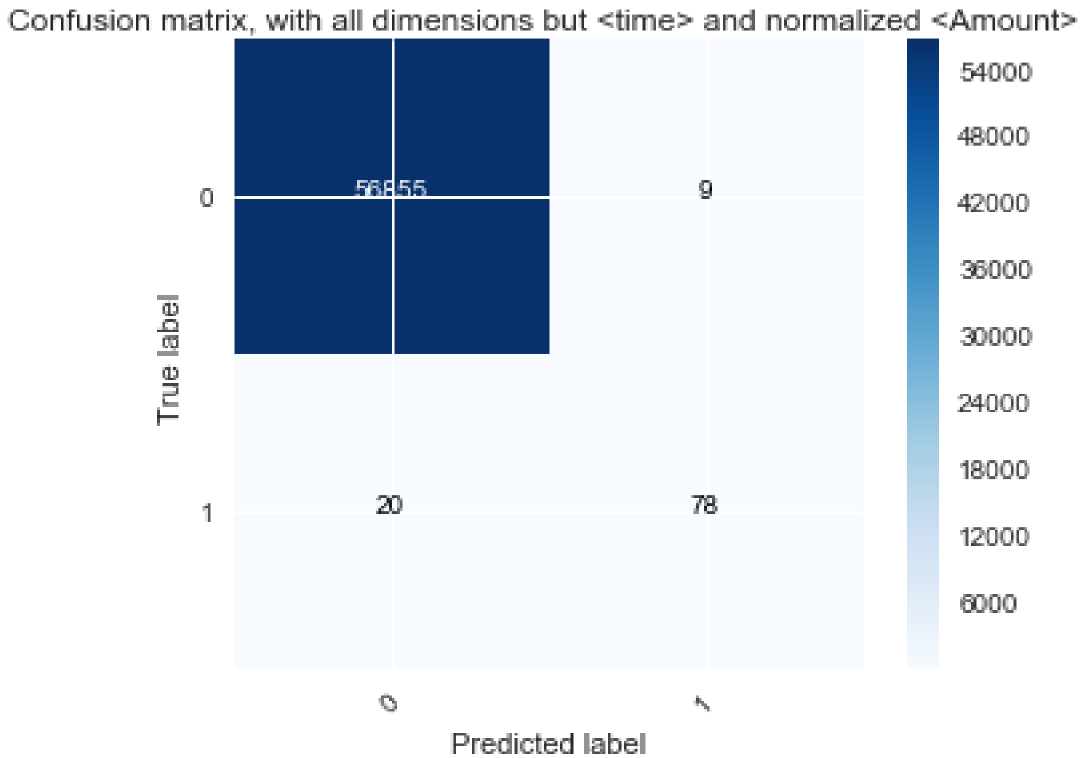

```
In [53]: df5 = calculate_add_scores(confusion_matrix_4,Classifier="DTC-1")
         frames = [df,df5]
         df = pd.concat(frames)
         print(df)
```

```
   Precision Classifier  Recall  Accuracy
0        9.0        RFC    29.0  0.999494
0        7.0        RFC    23.0  0.999607
0        7.0        RFC    27.0  0.999551
0        8.0        RFC    25.0  0.999565
0        3.0        RFC    23.0  0.999579
0       10.0      DTC-1    22.0  0.999473
```

- Decision Tree Classifier with Max Depth = 7

```
In [54]: X_train, X_test, y_train, y_test = train_test_split(new_data,data_class_ou
         print("Training and testing split was successful.")
         clf = tree.DecisionTreeClassifier(random_state=42,max_depth=7)
         clf.fit(X_train, y_train)
         y_pred = clf.predict(X_test)
         confusion_matrix_5 = calculate_confusion_matrix(y_test,y_pred)
         class_names = [0,1]
         plot_confusion_matrix(confusion_matrix_5, normalize=False, classes=class_n
                               title='Confusion matrix, with all dimensions but <ti
```

```
Training and testing split was successful.
Confusion matrix, without normalization
[[56855     9]
 [   20    78]]
```




```
In [87]: dotfile = open("D:/clf.dot", 'w')
         tree.export_graphviz(clf, out_file = dotfile, feature_names = data.columns
         dotfile.close()
```

```
In [84]: from sklearn.externals.six import StringIO
         import pydot
         dot_data = StringIO()
         tree.export_graphviz(clf, out_file=dot_data,
         feature_names=data.columns,
         filled=True, rounded=True,
         special_characters=True)
         graph = pydot.graph_from_dot_data(dot_data.getvalue())
         print(graph)
```

```
[<pydot.Dot object at 0x0000015F43DFA748>]
```

```
In [55]: df6 = calculate_add_scores(confusion_matrix_5,Classifier="DTC-2")
         frames = [df,df6]
         df = pd.concat(frames)
         print(df)
```

```
   Precision Classifier  Recall  Accuracy
0        9.0        RFC    29.0  0.999494
0        7.0        RFC    23.0  0.999607
0        7.0        RFC    27.0  0.999551
0        8.0        RFC    25.0  0.999565
0        3.0        RFC    23.0  0.999579
0       10.0      DTC-1    22.0  0.999473
0       10.0      DTC-2    21.0  0.999491
```

4 Analysis

Above result explanation with dimesnions

- RFC = Random Forest Classifier
- DTC = Decision Tree Classifier

1) First Pass - RFC including all dimensions in data set with test_size =0.25
2) Second Pass - RFC including all dimensions but time in data set with test_size =0.25
3) Third Pass - RFC including all dimensions but (time,amount) in data set with test_size =0.25
4) Fourth Pass - RFC including all dimensions but time, includes normalized amount with test_size =0.25
5) Fifth Pass - RFC including all dimensions but time, includes normalized amount with test_size =0.2
6) Sixth Pass - DTC with max_depth=6, including all dimensions but time, includes normalized amount with test_size =0.2
7) Seventh Pass - DTC with max_depth=7, including all dimensions but time, includes normalized amount with test_size =0.2

5 The best accuracy is obtained in the Random Forest Classifier

- The optimal results are obtained in the fifth and sixth pass due to Precision and Recall

6 XGBoost (3rd classifier)

Data Input

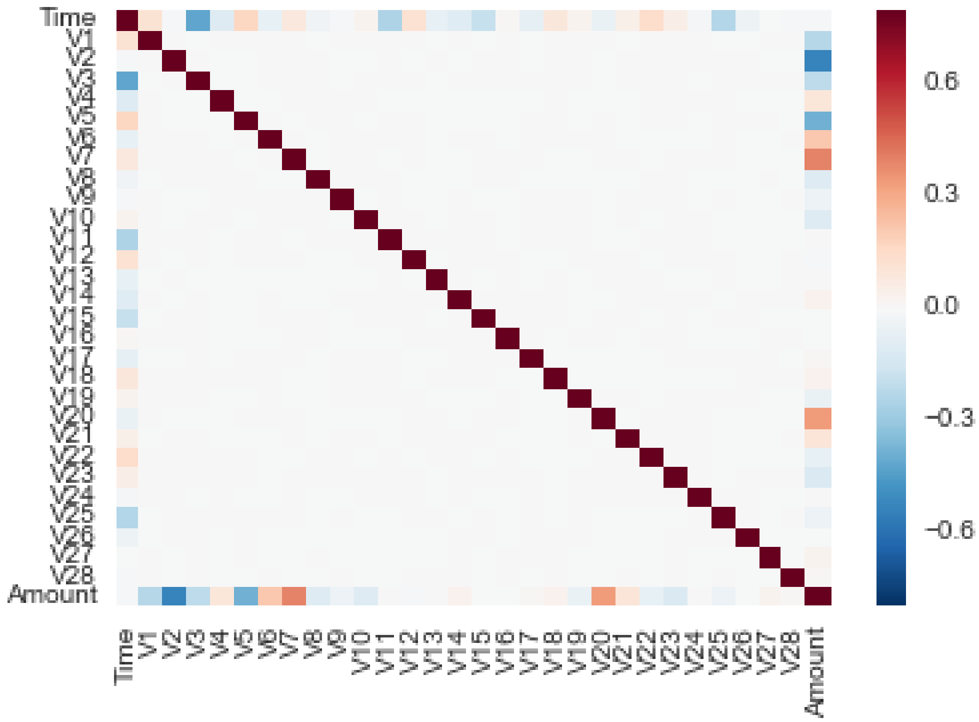
```
In [15]: dataset = pd.read_csv("E:/School/Sem 2/Knowledge Discovery in Databases/Fi
         dataset.head()

Out[15]:    Time        V1        V2        V3        V4        V5        V6
         0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239
         1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078
         2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791
         3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237
         4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592

                  V8        V9  ...        V21       V22       V23       V24  \
         0  0.098698  0.363787  ...  -0.018307  0.277838 -0.110474  0.066928
         1  0.085102 -0.255425  ...  -0.225775 -0.638672  0.101288 -0.339846
         2  0.247676 -1.514654  ...   0.247998  0.771679  0.909412 -0.689281
         3  0.377436 -1.387024  ...  -0.108300  0.005274 -0.190321 -1.175575
         4 -0.270533  0.817739  ...  -0.009431  0.798278 -0.137458  0.141267

                 V25       V26       V27       V28  Amount  Class
         0  0.128539 -0.189115  0.133558 -0.021053  149.62      0
         1  0.167170  0.125895 -0.008983  0.014724    2.69      0
         2 -0.327642 -0.139097 -0.055353 -0.059752  378.66      0
         3  0.647376 -0.221929  0.062723  0.061458  123.50      0
         4 -0.206010  0.502292  0.219422  0.215153   69.99      0

         [5 rows x 31 columns]

In [16]: dataset.describe()

Out[16]:                 Time            V1            V2            V3
         count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e
         mean    94813.859575  3.919560e-15  5.688174e-16 -8.769071e-15  2.782312e-
         std     47488.145955  1.958696e+00  1.651309e+00  1.516255e+00  1.415869e
         min         0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e
         25%     54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-
         50%     84692.000000  1.810880e-02  6.548556e-02  1.798463e-01 -1.984653e-
         75%    139320.500000  1.315642e+00  8.037239e-01  1.027196e+00  7.433413e-
         max    172792.000000  2.454930e+00  2.205773e+01  9.382558e+00  1.687534e

                          V5            V6            V7            V8
         count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+0
         mean  -1.552563e-15  2.010663e-15 -1.694249e-15 -1.927028e-16 -3.137024e-1
         std    1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00  1.098632e+0
         min   -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+0
         25%   -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-0
         50%   -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -5.142873e-0
```

```
75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01  5.971390e-0
max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01  1.559499e+0

            ...             V21           V22           V23            V
count       ...     2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+
mean        ...     1.537294e-16  7.959909e-16  5.367590e-16  4.458112e-
std         ...     7.345240e-01  7.257016e-01  6.244603e-01  6.056471e-
min         ...    -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+
25%         ...    -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-
50%         ...    -2.945017e-02  6.781943e-03 -1.119293e-02  4.097606e-
75%         ...     1.863772e-01  5.285536e-01  1.476421e-01  4.395266e-
max         ...     2.720284e+01  1.050309e+01  2.252841e+01  4.584549e+

                V25           V26           V27           V28          Amou
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  284807.0000
mean   1.453003e-15  1.699104e-15 -3.660161e-16 -1.206049e-16      88.3496
std    5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-01     250.1201
min   -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01       0.0000
25%   -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02       5.6000
50%    1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-02      22.0000
75%    3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-02      77.1650
max    7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+01   25691.1600

               Class
count  284807.000000
mean        0.001727
std         0.041527
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000

[8 rows x 31 columns]
```

In [17]:
```python
print(len(dataset[dataset.Class == 1]))
features = dataset.iloc[:, :-1]
print(features.shape)
label = dataset.iloc[:, -1].values
print(label.shape)

# heatmap for correlation, verifying that pca is already done
corrMat = features.corr()
sns.heatmap(corrMat, vmax=0.8)
```

```
492
(284807, 30)
(284807,)
```

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x15f2c6bda20>



Feature Engineering & Scaling

In [18]:
```
fraudInd = np.asarray(np.where(label == 1))
noFraudInd = np.where(label == 0)
features = features.values

# data standarization (zero-mean, unit variance) ~ truncation to [-1, 1]
scaler = StandardScaler()
scaler.fit(features)
features = scaler.transform(features)
```

In [20]:
```
TestPortion = 0.2
RND_STATE = 1

x_tr, x_test, y_tr, y_test = train_test_split(features, label, test_size =

xgb_model = xgb.XGBClassifier(n_estimators=100)
xgb_model.fit(x_tr, y_tr, verbose = 1)

y_pred = xgb_model.predict(x_test)
precision, recall, thresholds = precision_recall_curve(y_test, y_pred)
area = auc(recall, precision)
```

```python
        print('------------ Results for XGBClassifier ---------------')
        print('cm:', confusion_matrix(y_test,y_pred))
        #print('cr:', classification_report(y_test,y_pred))
        #print('recall_score:', recall_score(y_test,y_pred))
        print('roc_auc_score:',roc_auc_score(y_test,y_pred))
        print("Area Under P-R Curve: ",area)
```

```
------------ Results for XGBClassifier ---------------
cm: [[56869     6]
 [   21    66]]
roc_auc_score: 0.879257597575
Area Under P-R Curve:  0.837828011576
```