FraudDetection

May 2, 2018

1 Problem Description : Classification of Fraudulent Transactions

In [1]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from keras.layers import Flatten,Input
from keras import backend as K
from keras.models import Model, load_model
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold
from scipy.spatial import distance
from sklearn.decomposition import PCA
from numpy import linalg as LA
from keras.objectives import categorical_crossentropy
from sklearn.metrics import roc_curve, auc
import math
from scipy.stats import pearsonr
import copy
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import itertools
import csv
from sklearn import metrics
import tensorflow as tf
import tensorflow.contrib.layers as tl
import numpy as np
import pandas as pd
from sklearn import linear_model
from sklearn.ensemble import RandomForestClassifier
import seaborn as sb
%matplotlib inline
```

```
/home/ramchalamkr/.local/lib/python2.7/site-packages/h5py/__init__.py:36: FutureWarning: Conve
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

2 Part 1 : Supervised Learning Approaches

3 Steps

3.1 1. Checking the Data

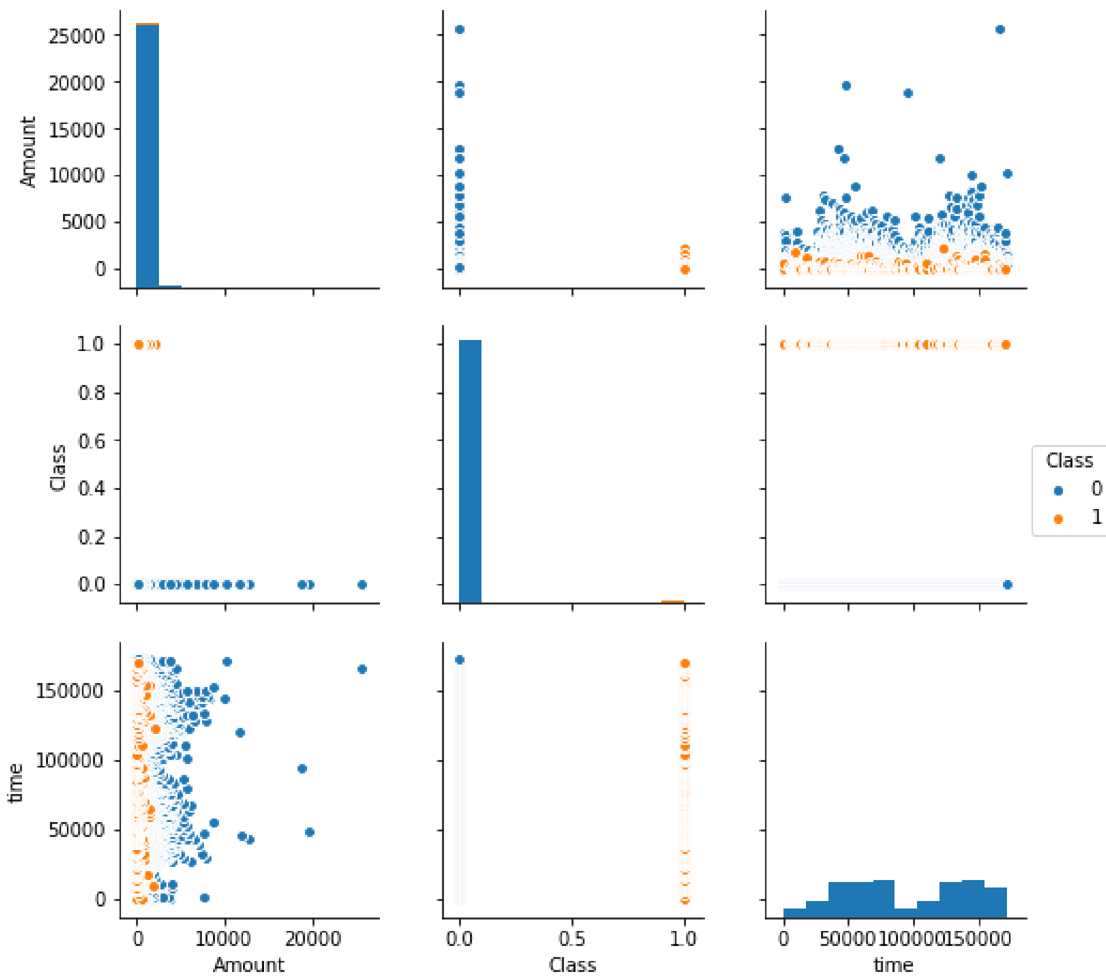
In [5]:
```
X = pd.read_csv('fraud_prep.csv',delimiter=',')
print X.shape
X.head()
```

```
(284807, 31)
```

Out[5]:
```
   Time        V1        V2        V3        V4        V5        V6        V7  \
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         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 [3]:
```
temp = pd.DataFrame({'Amount':X['Amount'],'time':X['Time'], 'Class':X['Class']})
sb.pairplot(temp.dropna(), hue='Class')
```

Out[3]:
```
<seaborn.axisgrid.PairGrid at 0x7fda7509dcd0>
```



3.1.1 Supervised Methods

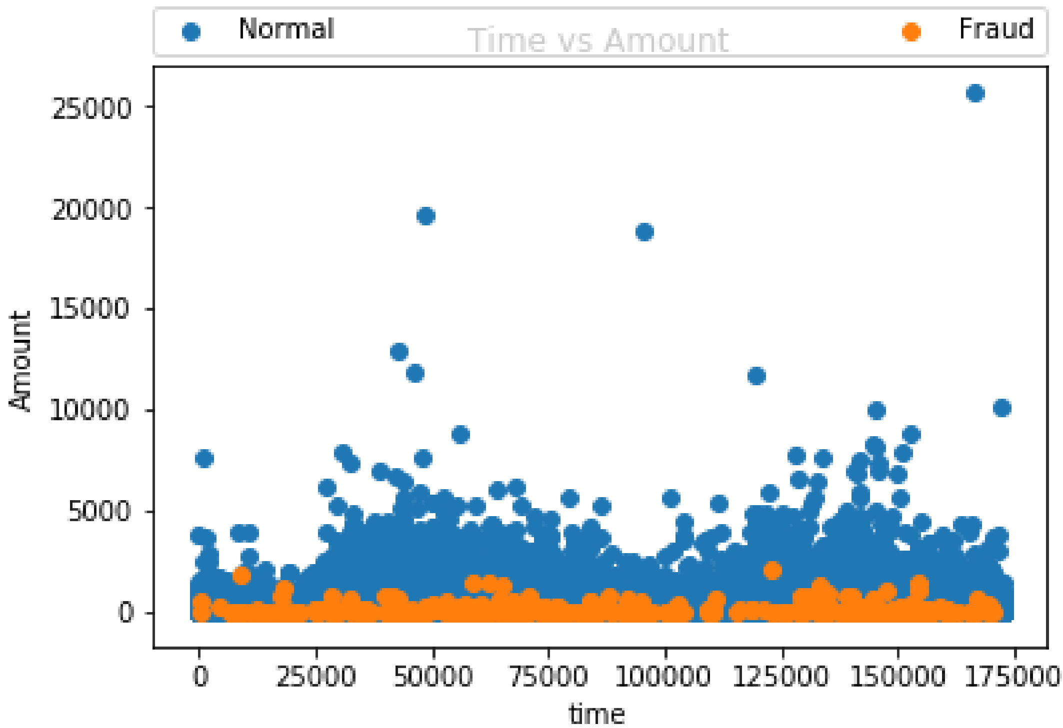
```
In [7]: Y = X['Class']
        del X['Class']
        print X.shape
        print Y.shape
```

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

As can be seen, across time, not much distinction in data, but mostly in terms of amount. So can drop time

```
In [4]: time = pd.DataFrame({'Amount':X['Amount'],'time':X['Time'], 'class':Y})
        Normaltime = time[(time['class'] == 0)]
        FraudTran = time[(time['class'] == 1)]
```

```
plt.scatter(Normaltime['time'], Normaltime['Amount'],label ="Normal")
plt.scatter(FraudTran['time'],FraudTran['Amount'],label ="Fraud")
plt.title("Time vs Amount")
plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,
           ncol=2, mode="expand", borderaxespad=0.)
plt.xlabel("time")
plt.ylabel("Amount")
plt.show()
```



3.2 2. Tidying the Data

```
In [5]: #Removing mean and scaling to unit variance
        X['Amount'] = StandardScaler().fit_transform(X['Amount'].values.reshape(-1, 1))
        #removing the time column
        del X['Time']
```

3.2.1 Logistic Regression (Baseline Model)

```
In [6]: X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.3,random_state=42)
        clf = linear_model.LogisticRegression()
        clf.fit(X_train,Y_train)
        Y_pred = clf.predict(X_test)
        probs = clf.predict_proba(X_test)
```

```
print "Accuracy",metrics.accuracy_score(Y_test, Y_pred)
print "AUC Value", metrics.roc_auc_score(Y_test, probs[:, 1])
print "Confusion Matrix"
print metrics.confusion_matrix(Y_test, Y_pred)
print "Precision Recall"
print metrics.classification_report(Y_test, Y_pred)
```

```
Accuracy 0.9992626663389628
AUC Value 0.9806636963106952
Confusion Matrix
[[85295    12]
 [   51    85]]
Precision Recall
             precision    recall  f1-score   support

          0       1.00      1.00      1.00     85307
          1       0.88      0.62      0.73       136

avg / total       1.00      1.00      1.00     85443
```

```
In [7]: X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.3,random_state=42)
        clf = RandomForestClassifier(max_depth=20)
        clf.fit(X_train,Y_train)
        Y_pred = clf.predict(X_test)
        probs = clf.predict_proba(X_test)
        print "Accuracy",metrics.accuracy_score(Y_test, Y_pred)
        print "AUC Value", metrics.roc_auc_score(Y_test, probs[:, 1])
        print "Confusion Matrix"
        print metrics.confusion_matrix(Y_test, Y_pred)
        print "Precision Recall"
        print metrics.classification_report(Y_test, Y_pred)
```

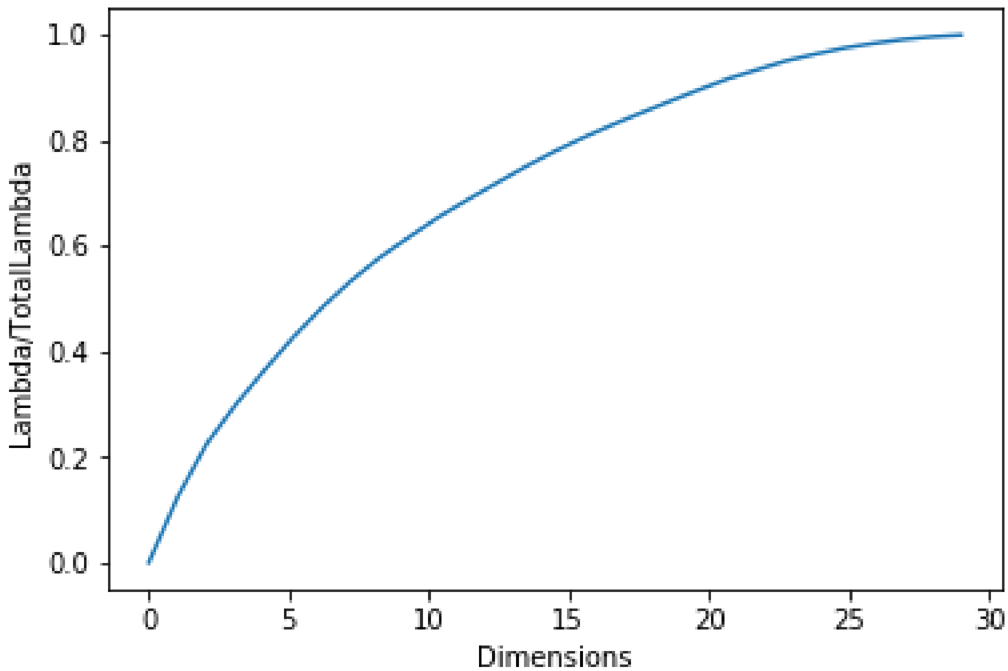
```
Accuracy 0.9995669627705019
AUC Value 0.9526959807449773
Confusion Matrix
[[85299     8]
 [   29   107]]
Precision Recall
             precision    recall  f1-score   support

          0       1.00      1.00      1.00     85307
          1       0.93      0.79      0.85       136

avg / total       1.00      1.00      1.00     85443
```

3.2.2 PCA

In [8]:
```
print X.shape
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.2,random_state=42)

Covariance = np.dot(X_train.T,X_train)

Lambda, e = LA.eigh(Covariance)
Lambda = Lambda.reshape(Lambda.shape[0],1)
Lambda = sorted(Lambda,reverse=True)
TotalLambda = np.sum(Lambda)
LambdaProp = []
for i in range(X_train.shape[1]):
    temp = np.sum(Lambda[0:i])*1.0/TotalLambda
    LambdaProp.append(temp)

Dim = np.linspace(0, X_train.shape[1], X_train.shape[1])

plt.plot(Dim,LambdaProp)
plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,
           ncol=2, mode="expand", borderaxespad=0.)
plt.xlabel('Dimensions')
plt.ylabel('Lambda/TotalLambda')
plt.show()
```

```
(284807, 29)
```

```
/home/ramchalamkr/.local/lib/python2.7/site-packages/matplotlib/axes/_axes.py:545: UserWarning
  warnings.warn("No labelled objects found. "
```



3.2.3 This means there is no point in doing PCA as the total variance is represented by mostly all 28~30 features. It could also mean that the data already contains the best features and all are posibly required to get best results.

3.3 3. Model Development and Performance

3.3.1 Feed Forwad Neural Networks

3.3.2 (Model Architecture followed an extension of a published paper.https://pdfs.semanticscholar.org/0419/c275f05841d87ab9a4c9767a4f997b61a50e.pdf)

3.3.3 using original data

In [41]:
```
Y = np.reshape(Y,[Y.shape[0],1])
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.3,random_state=42)
X_train = np.asarray(X_train)
X_test = np.asarray(X_test)
Y_train = np.asarray(Y_train)
Y_test = np.asarray(Y_test)
K.clear_session()
InputWidth = X_train.shape[1]
model = Sequential()
model.add(Dense(256, input_shape = (InputWidth,), activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.2))
```

```python
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
print model.summary()
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
op = model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=10, batch_si
#output = model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=150, ba
y_pred = model.predict(X_test)
y_pred = [int(item) for sublist in y_pred for item in sublist]
#print y_pred.shape
print "Accuracy",metrics.accuracy_score(Y_test, y_pred)
#print np.mean(y_pred==Y_test)
print metrics.confusion_matrix(Y_test, y_pred)
print metrics.classification_report(Y_test, y_pred)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 256)               7680
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_4 (Dense)              (None, 32)                2080
_________________________________________________________________
dropout_4 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 33
=================================================================
Total params: 50,945
Trainable params: 50,945
Non-trainable params: 0
_________________________________________________________________
None
Train on 199364 samples, validate on 85443 samples
Epoch 1/10
10s - loss: 0.0103 - acc: 0.9979 - val_loss: 0.0037 - val_acc: 0.9994
Epoch 2/10
```

```
11s - loss: 0.0041 - acc: 0.9994 - val_loss: 0.0033 - val_acc: 0.9994
Epoch 3/10
10s - loss: 0.0038 - acc: 0.9994 - val_loss: 0.0030 - val_acc: 0.9994
Epoch 4/10
10s - loss: 0.0035 - acc: 0.9994 - val_loss: 0.0032 - val_acc: 0.9994
Epoch 5/10
11s - loss: 0.0032 - acc: 0.9994 - val_loss: 0.0028 - val_acc: 0.9994
Epoch 6/10
11s - loss: 0.0032 - acc: 0.9993 - val_loss: 0.0028 - val_acc: 0.9994
Epoch 7/10
11s - loss: 0.0030 - acc: 0.9994 - val_loss: 0.0027 - val_acc: 0.9994
Epoch 8/10
14s - loss: 0.0028 - acc: 0.9994 - val_loss: 0.0028 - val_acc: 0.9994
Epoch 9/10
12s - loss: 0.0027 - acc: 0.9994 - val_loss: 0.0028 - val_acc: 0.9994
Epoch 10/10
12s - loss: 0.0028 - acc: 0.9994 - val_loss: 0.0029 - val_acc: 0.9994
0.0
[[85307     0]
 [  128     8]]
             precision    recall  f1-score   support

          0       1.00      1.00      1.00     85307
          1       1.00      0.06      0.11       136

avg / total       1.00      1.00      1.00     85443


/home/ramchalamkr/.local/lib/python2.7/site-packages/ipykernel_launcher.py:27: DeprecationWarni
```

3.3.4 DownSampling for imbalanced class

```
In [47]: #Downsampling the data to handle imbalance class.
         X = pd.read_csv('fraud_prep.csv',delimiter=',')
         fraud = X[X.Class ==1]
         normal = X[X.Class==0]
         #fraud.head()
         frames = [fraud, normal[0:1000]]

         DownSample = pd.concat(frames)
         DownSample.head()
         Y = DownSample['Class']
         del DownSample['Class']
         print DownSample.shape
         print Y.shape
         del DownSample['Time']
```

```
#Removing mean and scaling to unit variance
DownSample['Amount'] = StandardScaler().fit_transform(DownSample['Amount'].values.resh
```

```
(1492, 30)
(1492,)
```

Only keras

```
In [48]: Y = np.reshape(Y,[Y.shape[0],1])
         class_weight = {0: 1.,1: 50.}
         X_train,X_test,Y_train,Y_test = train_test_split(DownSample,Y,test_size = 0.3,random_s
         X_train = np.asarray(X_train)
         X_test = np.asarray(X_test)
         Y_train = np.asarray(Y_train)
         Y_test = np.asarray(Y_test)
         K.clear_session()
         InputWidth = X_train.shape[1]
         model = Sequential()
         model.add(Dense(256, input_shape = (InputWidth,), activation='relu'))
         model.add(Dropout(0.2))
         model.add(Dense(128, activation='relu'))
         model.add(Dropout(0.2))
         model.add(Dense(64, activation='relu'))
         model.add(Dropout(0.2))
         model.add(Dense(32, activation='relu'))
         model.add(Dropout(0.2))
         model.add(Dense(1, activation='sigmoid'))
         print model.summary()
         # Compile model
         model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
         #output = model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=150, b
         output = model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=50, batc
         y_pred = model.predict(X_test)
         y_pred = [int(item) for sublist in y_pred for item in sublist]
         #print y_pred.shape
         print "accuracy", metrics.accuracy_score(Y_test, y_pred)
         print metrics.confusion_matrix(Y_test, y_pred)
         print metrics.classification_report(Y_test, y_pred)
         print "predicting on whole dataset"
         Y = X['Class']
         del X['Class']
         print X.shape
         print Y.shape
         del X['Time']
         #Removing mean and scaling to unit variance
         X['Amount'] = StandardScaler().fit_transform(X['Amount'].values.reshape(-1, 1))
         X1 = np.asarray(X)
```

```python
y_pred = model.predict(X1)
y_pred = [int(item) for sublist in y_pred for item in sublist]
#print y_pred.shape
print "accuracy", metrics.accuracy_score(Y, y_pred)
print metrics.confusion_matrix(Y, y_pred)
print metrics.classification_report(Y, y_pred)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 256)               7680
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_4 (Dense)              (None, 32)                2080
_________________________________________________________________
dropout_4 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 33
=================================================================
Total params: 50,945
Trainable params: 50,945
Non-trainable params: 0
_________________________________________________________________
None
Train on 1044 samples, validate on 448 samples
Epoch 1/50
0s - loss: 5.1743 - acc: 0.4042 - val_loss: 0.8446 - val_acc: 0.3281
Epoch 2/50
0s - loss: 1.4790 - acc: 0.3314 - val_loss: 1.0919 - val_acc: 0.3281
Epoch 3/50
0s - loss: 1.3967 - acc: 0.3305 - val_loss: 0.9615 - val_acc: 0.3281
Epoch 4/50
0s - loss: 1.1933 - acc: 0.3343 - val_loss: 0.7728 - val_acc: 0.3281
Epoch 5/50
0s - loss: 1.0534 - acc: 0.3879 - val_loss: 0.6780 - val_acc: 0.4799
Epoch 6/50
0s - loss: 0.9248 - acc: 0.5651 - val_loss: 0.6338 - val_acc: 0.7522
Epoch 7/50
0s - loss: 0.8536 - acc: 0.7605 - val_loss: 0.5559 - val_acc: 0.8638
```

```
Epoch 8/50
0s - loss: 0.7207 - acc: 0.8439 - val_loss: 0.4839 - val_acc: 0.8951
Epoch 9/50
0s - loss: 0.6813 - acc: 0.9119 - val_loss: 0.4420 - val_acc: 0.8996
Epoch 10/50
0s - loss: 0.5412 - acc: 0.9167 - val_loss: 0.4118 - val_acc: 0.9062
Epoch 11/50
0s - loss: 0.5269 - acc: 0.9234 - val_loss: 0.4168 - val_acc: 0.8996
Epoch 12/50
0s - loss: 0.4526 - acc: 0.9195 - val_loss: 0.3912 - val_acc: 0.8996
Epoch 13/50
0s - loss: 0.6253 - acc: 0.9291 - val_loss: 0.4197 - val_acc: 0.8906
Epoch 14/50
0s - loss: 0.4697 - acc: 0.8841 - val_loss: 0.4543 - val_acc: 0.8839
Epoch 15/50
0s - loss: 0.4478 - acc: 0.9205 - val_loss: 0.3316 - val_acc: 0.9107
Epoch 16/50
0s - loss: 0.3840 - acc: 0.9195 - val_loss: 0.4413 - val_acc: 0.8772
Epoch 17/50
0s - loss: 0.3357 - acc: 0.9176 - val_loss: 0.3639 - val_acc: 0.8951
Epoch 18/50
0s - loss: 0.3070 - acc: 0.9502 - val_loss: 0.3096 - val_acc: 0.9241
Epoch 19/50
0s - loss: 0.2654 - acc: 0.9550 - val_loss: 0.3343 - val_acc: 0.9241
Epoch 20/50
0s - loss: 0.3184 - acc: 0.9511 - val_loss: 0.3510 - val_acc: 0.9152
Epoch 21/50
0s - loss: 0.2361 - acc: 0.9425 - val_loss: 0.3329 - val_acc: 0.9196
Epoch 22/50
0s - loss: 0.1889 - acc: 0.9569 - val_loss: 0.2884 - val_acc: 0.9353
Epoch 23/50
0s - loss: 0.3477 - acc: 0.9502 - val_loss: 0.4055 - val_acc: 0.9085
Epoch 24/50
0s - loss: 0.2495 - acc: 0.9454 - val_loss: 0.3276 - val_acc: 0.9241
Epoch 25/50
0s - loss: 0.1852 - acc: 0.9626 - val_loss: 0.2700 - val_acc: 0.9397
Epoch 26/50
0s - loss: 0.1982 - acc: 0.9674 - val_loss: 0.2760 - val_acc: 0.9353
Epoch 27/50
0s - loss: 0.2091 - acc: 0.9636 - val_loss: 0.3116 - val_acc: 0.9241
Epoch 28/50
0s - loss: 0.3697 - acc: 0.9626 - val_loss: 0.2978 - val_acc: 0.9308
Epoch 29/50
0s - loss: 0.2018 - acc: 0.9521 - val_loss: 0.3580 - val_acc: 0.9196
Epoch 30/50
0s - loss: 0.2253 - acc: 0.9531 - val_loss: 0.3407 - val_acc: 0.9219
Epoch 31/50
0s - loss: 0.1935 - acc: 0.9579 - val_loss: 0.3021 - val_acc: 0.9286
```

```
Epoch 32/50
0s - loss: 0.2171 - acc: 0.9579 - val_loss: 0.3012 - val_acc: 0.9308
Epoch 33/50
0s - loss: 0.1752 - acc: 0.9607 - val_loss: 0.3244 - val_acc: 0.9241
Epoch 34/50
0s - loss: 0.1622 - acc: 0.9607 - val_loss: 0.3094 - val_acc: 0.9308
Epoch 35/50
0s - loss: 0.1339 - acc: 0.9703 - val_loss: 0.2647 - val_acc: 0.9464
Epoch 36/50
0s - loss: 0.1047 - acc: 0.9808 - val_loss: 0.2698 - val_acc: 0.9487
Epoch 37/50
0s - loss: 0.1291 - acc: 0.9703 - val_loss: 0.2763 - val_acc: 0.9487
Epoch 38/50
0s - loss: 0.0981 - acc: 0.9751 - val_loss: 0.2710 - val_acc: 0.9487
Epoch 39/50
0s - loss: 0.1477 - acc: 0.9722 - val_loss: 0.2906 - val_acc: 0.9442
Epoch 40/50
0s - loss: 0.1133 - acc: 0.9674 - val_loss: 0.3074 - val_acc: 0.9375
Epoch 41/50
0s - loss: 0.1317 - acc: 0.9684 - val_loss: 0.2840 - val_acc: 0.9420
Epoch 42/50
0s - loss: 0.1229 - acc: 0.9713 - val_loss: 0.2584 - val_acc: 0.9487
Epoch 43/50
0s - loss: 0.1300 - acc: 0.9741 - val_loss: 0.2892 - val_acc: 0.9397
Epoch 44/50
0s - loss: 0.1509 - acc: 0.9693 - val_loss: 0.3375 - val_acc: 0.9286
Epoch 45/50
0s - loss: 0.1307 - acc: 0.9626 - val_loss: 0.2941 - val_acc: 0.9397
Epoch 46/50
0s - loss: 0.1939 - acc: 0.9665 - val_loss: 0.2985 - val_acc: 0.9286
Epoch 47/50
0s - loss: 0.1113 - acc: 0.9684 - val_loss: 0.2752 - val_acc: 0.9442
Epoch 48/50
0s - loss: 0.1086 - acc: 0.9732 - val_loss: 0.3133 - val_acc: 0.9397
Epoch 49/50
0s - loss: 0.1056 - acc: 0.9665 - val_loss: 0.2998 - val_acc: 0.9464
Epoch 50/50
0s - loss: 0.1192 - acc: 0.9770 - val_loss: 0.3017 - val_acc: 0.9442
accuracy 0.9419642857142857
[[299   2]
 [ 24 123]]
             precision    recall  f1-score   support

          0       0.93      0.99      0.96       301
          1       0.98      0.84      0.90       147

avg / total       0.94      0.94      0.94       448
```

```
predicting on whole dataset
(284807, 30)
(284807,)
accuracy 0.9954916838420405
[[283124   1191]
 [    93    399]]
             precision    recall  f1-score   support

          0       1.00      1.00      1.00    284315
          1       0.25      0.81      0.38       492

avg / total       1.00      1.00      1.00    284807
```

3.3.5 No Downsampling but provide weights for the clases based on the imbalance

```
In [51]: Y = np.reshape(Y,[Y.shape[0],1])
         class_weight = {0: 1.,1: 50.}
         X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.3,random_state=42)
         X_train = np.asarray(X_train)
         X_test = np.asarray(X_test)
         Y_train = np.asarray(Y_train)
         Y_test = np.asarray(Y_test)
         K.clear_session()
         InputWidth = X_train.shape[1]
         model = Sequential()
         model.add(Dense(256, input_shape = (InputWidth,), activation='relu'))
         model.add(Dropout(0.2))
         model.add(Dense(128, activation='relu'))
         model.add(Dropout(0.2))
         model.add(Dense(64, activation='relu'))
         model.add(Dropout(0.2))
         model.add(Dense(32, activation='relu'))
         model.add(Dropout(0.2))
         model.add(Dense(1, activation='sigmoid'))
         print model.summary()
         # Compile model
         model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
         output = model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=10, batc
         y_pred = model.predict(X_test)
         y_pred = [int(item) for sublist in y_pred for item in sublist]
         #print y_pred.shape
         print "Test accuracy", metrics.accuracy_score(Y_test, y_pred)
         print metrics.confusion_matrix(Y_test, y_pred)
         print metrics.classification_report(Y_test, y_pred)

         y_pred = model.predict(X_train)
```

```python
y_pred = [int(item) for sublist in y_pred for item in sublist]
#print y_pred.shape
print "Train accuracy", metrics.accuracy_score(Y_train, y_pred)
print metrics.confusion_matrix(Y_train, y_pred)
print metrics.classification_report(Y_train, y_pred)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 256)               7680
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_4 (Dense)              (None, 32)                2080
_________________________________________________________________
dropout_4 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 33
=================================================================
Total params: 50,945
Trainable params: 50,945
Non-trainable params: 0
_________________________________________________________________
None
Train on 199364 samples, validate on 85443 samples
Epoch 1/10
14s - loss: 0.1443 - acc: 0.9956 - val_loss: 0.0736 - val_acc: 0.9707
Epoch 2/10
12s - loss: 0.0810 - acc: 0.9966 - val_loss: 0.0083 - val_acc: 0.9993
Epoch 3/10
11s - loss: 0.0762 - acc: 0.9974 - val_loss: 0.0159 - val_acc: 0.9987
Epoch 4/10
12s - loss: 0.0864 - acc: 0.9976 - val_loss: 0.0294 - val_acc: 0.9993
Epoch 5/10
11s - loss: 0.0679 - acc: 0.9980 - val_loss: 0.0093 - val_acc: 0.9991
Epoch 6/10
11s - loss: 0.0589 - acc: 0.9969 - val_loss: 0.0169 - val_acc: 0.9979
Epoch 7/10
12s - loss: 0.0608 - acc: 0.9969 - val_loss: 0.0122 - val_acc: 0.9968
Epoch 8/10
```

```
12s - loss: 0.0641 - acc: 0.9968 - val_loss: 0.0264 - val_acc: 0.9972
Epoch 9/10
12s - loss: 0.0584 - acc: 0.9972 - val_loss: 0.0179 - val_acc: 0.9979
Epoch 10/10
13s - loss: 0.0574 - acc: 0.9984 - val_loss: 0.0174 - val_acc: 0.9980
Test accuracy 0.999403110845827
[[85281    26]
 [   25   111]]
             precision    recall  f1-score   support

          0       1.00      1.00      1.00     85307
          1       0.81      0.82      0.81       136

avg / total       1.00      1.00      1.00     85443

Train accuracy 0.9994031018639273
[[198965     43]
 [    76    280]]
             precision    recall  f1-score   support

          0       1.00      1.00      1.00    199008
          1       0.87      0.79      0.82       356

avg / total       1.00      1.00      1.00    199364
```

Tensorflow + keras

```
In [52]: Y = np.reshape(Y,[Y.shape[0],1])
         X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.33,random_state=42)
         tf.logging.set_verbosity(tf.logging.INFO)
         sess = tf.Session()
         K.set_session(sess)

         InputWidth = X_train.shape[1]
         OutputWidth = 1
         #print Input[0:2]
         #print Label[0:2]
         inp = tf.placeholder(tf.float32, shape=(None,InputWidth))
         out = tf.placeholder(tf.float32, shape=(None,OutputWidth))
         print(out.get_shape())
         print(inp.get_shape())
         predictions=[]
         predictionsTrain =[]
         #OutputWidth = len(y_train[0])
         learning_rate = 0.0001
         #inplayer = Input(shape=(InputWidth, ))
```

```
x = Dense(512,activation='relu')(inp)
print(x)
x = Dense(256,activation='relu')(x)
print(x)
x = Dense(128,activation='relu')(x)
print(x)
#x = K.reshape(x,(len(X_train)*InputWidth,5))
#print x
x = Dense(32,activation='relu')(x)
print(x)
x = Dense(16,activation='relu')(x)
print(x)
preds = Dense(OutputWidth, activation='sigmoid')(x)
print(preds)

loss = tf.reduce_mean(categorical_crossentropy(out, preds))
train_step = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(loss)
#acc_value = accuracy(out, preds)
with sess.as_default():
    sess.run(tf.global_variables_initializer())
    for epoch in range(20):
        print("epoch " + str(epoch))
    #train_step.run(feed_dict={inp:X_train,out:y_train,K.learning_phase(): 1})
        for i in range(1000):
            if(i%1000==0):
                print("iteration number"+str(i))
            _, loss_val = sess.run([train_step, loss],
                         feed_dict={inp:X_train[i*180:(i+1)*180],out:Y_train[i*180:
        print "loss", loss_val
            #train_step.run(feed_dict={inp:X_train[i*180:(i+1)*180],out:X_train[i*180
        #temp=acc_value.eval(feed_dict={inp: X_train,out: y_train})
        #print type(temp)
        #print temp
        #save_path = saver.save(sess, "model_512_5neighbours_ExtraDense.ckpt")
        print(out.get_shape())
        print(preds.get_shape())
        #print preds.eval(feed_dict={inp:X_train[0:2],out: y_train[0:2]})
        #p = tf.argmax(preds, axis=1)
    predictions = preds.eval(feed_dict={inp: X_test})
    predictionsTrain = preds.eval(feed_dict={inp: X_train})
    predictions = [int(item) for sublist in predictions for item in sublist]
    predictionsTrain = [int(item) for sublist in predictionsTrain for item in sublist]
    #print predictions.shape
    print Y_test.shape
    print np.mean(predictions==Y_test)
    print "Train accuracy"
    print metrics.confusion_matrix(Y_train, predictionsTrain)
    print metrics.classification_report(Y_train, predictionsTrain)
```

```python
            print "test accuracy"
            print metrics.confusion_matrix(Y_test, predictions)
            print metrics.classification_report(Y_test, predictions)
            #print(len(predictions))
            #print(len(predictionsTrain))
```

```
(?, 1)
(?, 29)
Tensor("dense_6/Relu:0", shape=(?, 512), dtype=float32)
Tensor("dense_7/Relu:0", shape=(?, 256), dtype=float32)
Tensor("dense_8/Relu:0", shape=(?, 128), dtype=float32)
Tensor("dense_9/Relu:0", shape=(?, 32), dtype=float32)
Tensor("dense_10/Relu:0", shape=(?, 16), dtype=float32)
Tensor("dense_11/Sigmoid:0", shape=(?, 1), dtype=float32)
epoch 0
iteration number0
loss 0.0
(?, 1)
(?, 1)
epoch 1
iteration number0
loss 0.0
(?, 1)
(?, 1)
epoch 2
iteration number0
loss 0.0
(?, 1)
(?, 1)
epoch 3
iteration number0
loss 0.0
(?, 1)
(?, 1)
epoch 4
iteration number0
loss 0.0
(?, 1)
(?, 1)
epoch 5
iteration number0
loss 0.0
(?, 1)
(?, 1)
epoch 6
iteration number0
loss 0.0
```

```
(?, 1)
(?, 1)
epoch 7
iteration number0
loss 0.0
(?, 1)
(?, 1)
epoch 8
iteration number0
loss 0.0
(?, 1)
(?, 1)
epoch 9
iteration number0
loss 0.0
(?, 1)
(?, 1)
epoch 10
iteration number0
loss 0.0
(?, 1)
(?, 1)
epoch 11
iteration number0
loss 0.0
(?, 1)
(?, 1)
epoch 12
iteration number0
loss 0.0
(?, 1)
(?, 1)
epoch 13
iteration number0
loss 0.0
(?, 1)
(?, 1)
epoch 14
iteration number0
loss 0.0
(?, 1)
(?, 1)
epoch 15
iteration number0
loss 0.0
(?, 1)
(?, 1)
epoch 16
```

```
iteration number0
loss 0.0
(?, 1)
(?, 1)
epoch 17
iteration number0
loss 0.0
(?, 1)
(?, 1)
epoch 18
iteration number0
loss 0.0
(?, 1)
(?, 1)
epoch 19
iteration number0
loss 0.0
(?, 1)
(?, 1)
(93987, 1)
0.0
Train accuracy
[[190477      0]
 [   343      0]]

/home/ramchalamkr/.local/lib/python2.7/site-packages/ipykernel_launcher.py:64: DeprecationWarn

             precision    recall  f1-score   support

          0       1.00      1.00      1.00    190477
          1       0.00      0.00      0.00       343

avg / total       1.00      1.00      1.00    190820

test accuracy
[[93838     0]
 [  149     0]]
             precision    recall  f1-score   support

          0       1.00      1.00      1.00     93838
          1       0.00      0.00      0.00       149

avg / total       1.00      1.00      1.00     93987
```

4 Part 2 : Unsupervised Learning approach using Autoencoders

5 Steps

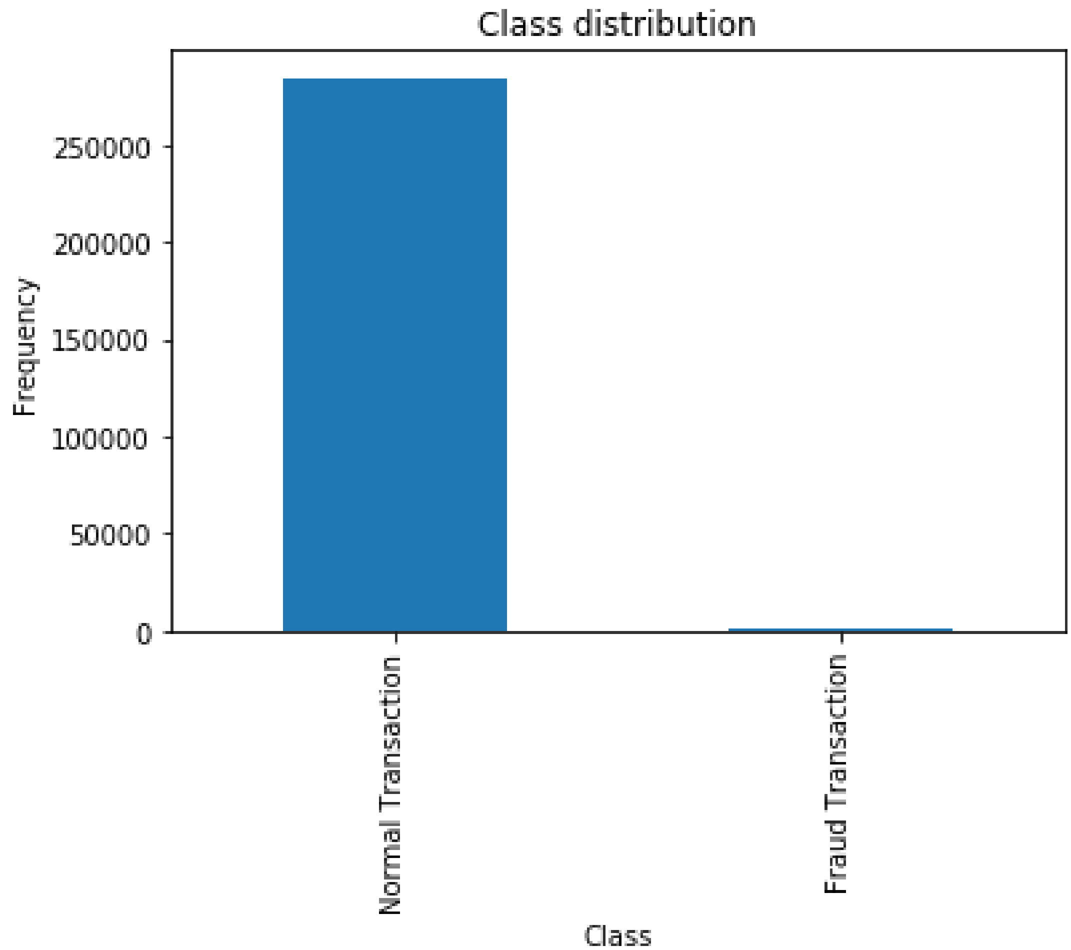
5.1 1. Checking and Tidying Data

```
In [9]: X = pd.read_csv('fraud_prep.csv',delimiter=',')
        print X.shape
        del X['Time']
        #Removing mean and scaling to unit variance
        X['Amount'] = StandardScaler().fit_transform(X['Amount'].values.reshape(-1, 1))
```

```
(284807, 31)
```

```
In [10]: count_classes = pd.value_counts(X['Class'], sort = True)
         print count_classes
         count_classes.plot(kind = 'bar')
         plt.title("Class distribution")
         plt.xticks(range(2), ['Normal Transaction','Fraud Transaction'])
         plt.xlabel("Class")
         plt.ylabel("Frequency")
         plt.show()
```

```
0    284315
1       492
Name: Class, dtype: int64
```




```
In [11]: fraud = X[X.Class ==1]
         normal = X[X.Class ==0]

In [12]: fraud.Amount.describe()

Out[12]: count    492.000000
         mean       0.135382
         std        1.026242
         min       -0.353229
         25%       -0.349231
         50%       -0.316247
         75%        0.070128
         max        8.146182
         Name: Amount, dtype: float64

In [13]: normal.Amount.describe()
```

```
Out[13]: count    284315.000000
         mean         -0.000234
         std           0.999942
         min          -0.353229
         25%          -0.330640
         50%          -0.265271
         75%          -0.045177
         max         102.362243
         Name: Amount, dtype: float64
```

```
In [15]: X_train,X_test = train_test_split(X,test_size = 0.3,random_state=42)
         X_train = X_train[X_train.Class==0]
         del X_train['Class']
         y_test = X_test['Class']
         del X_test['Class']
         X_train = np.asarray(X_train)
         X_test = np.asarray(X_test)
         print X_train.shape
         print X_test.shape
         print y_test.shape
```

```
(199008, 29)
(85443, 29)
(85443,)
```

5.2 2. Model Development

5.2.1 Build Model Pure tensorflow

```
In [ ]: inputs = tf.placeholder(tf.float32, shape=(None, X.shape[1]))
        out = X.shape[1]
        def buildmodel(inp):
            e1 = tl.fully_connected(inp, 32, activation_fn=tf.nn.softplus)
            e2 = tl.fully_connected(e1, 16, activation_fn=tf.nn.softplus)
            e3 = tl.fully_connected(e2, 8, activation_fn=tf.nn.softplus)
            d1 = tl.fully_connected(e3, 8, activation_fn=tf.nn.softplus)
            d2 = tl.fully_connected(d1, 16, activation_fn=tf.nn.softplus)
            d3 = tl.fully_connected(d2, out, activation_fn=tf.nn.softplus)
            return out

        outputs = buildmodel(inputs)
        loss = tf.reduce_mean(tf.square(outputs - inputs))
        train_op = tf.train.AdamOptimizer(learning_rate=0.001).minimize(loss)
        #init = tf.global_variables_initializer()
```

5.2.2 Keras + Tensorflow

```
In [60]: tf.logging.set_verbosity(tf.logging.INFO)
         sess = tf.Session()
```

```python
K.set_session(sess)

InputWidth = X_train.shape[1]
#print Input[0:2]
#print Label[0:2]
inp = tf.placeholder(tf.float32, shape=(None,InputWidth))
out = tf.placeholder(tf.float32, shape=(None,InputWidth))
print(out.get_shape())
print(inp.get_shape())
predictions=[]
predictionsTrain =[]
#OutputWidth = len(y_train[0])
learning_rate = 0.001
#inplayer = Input(shape=(InputWidth, ))
x = Dense(64,activation='relu')(inp)
print(x)
x = Dense(32,activation='relu')(x)
print(x)
x = Dense(16,activation='relu')(x)
print(x)
#x = K.reshape(x,(len(X_train)*InputWidth,5))
#print x
x = Dense(16,activation='relu')(x)
print(x)
x = Dense(32,activation='relu')(x)
print(x)
x = Dense(64,activation='relu')(x)
print(x)
preds = Dense(InputWidth, activation='relu')(x)
print(preds)

loss = tf.reduce_mean(tf.square(preds - inp))
train_step = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(loss)
#acc_value = accuracy(out, preds)
with sess.as_default():
    sess.run(tf.global_variables_initializer())
    for epoch in range(20):
        print("epoch " + str(epoch))
    #train_step.run(feed_dict={inp:X_train,out:y_train,K.learning_phase(): 1})
        for i in range(1000):
            if(i%1000==0):
                print("iteration number"+str(i))
            _, loss_val = sess.run([train_step, loss],
                           feed_dict={inp:X_train[i*180:(i+1)*180],out:X_train[i*180:
        print "loss", loss_val
            #train_step.run(feed_dict={inp:X_train[i*180:(i+1)*180],out:X_train[i*180
        #temp=acc_value.eval(feed_dict={inp: X_train,out: y_train})
        #print type(temp)
```

```python
                #print temp
                #save_path = saver.save(sess, "model_512_5neighbours_ExtraDense.ckpt")
                print(out.get_shape())
                print(preds.get_shape())
                #print preds.eval(feed_dict={inp:X_train[0:2],out: y_train[0:2]})
                #p = tf.argmax(preds, axis=1)
            predictions = preds.eval(feed_dict={inp: X_test})
            print predictions.shape
            mse = np.mean(np.power(X_test - predictions, 2))
            print "test mse",mse
            predictions = preds.eval(feed_dict={inp: X_train})
            mse = np.mean(np.power(X_train - predictions, 2))
            print "train mse",mse
            print(len(predictions))
            print(len(predictionsTrain))
```

```
(?, 29)
(?, 29)
Tensor("dense_18/Relu:0", shape=(?, 64), dtype=float32)
Tensor("dense_19/Relu:0", shape=(?, 32), dtype=float32)
Tensor("dense_20/Relu:0", shape=(?, 16), dtype=float32)
Tensor("dense_21/Relu:0", shape=(?, 16), dtype=float32)
Tensor("dense_22/Relu:0", shape=(?, 32), dtype=float32)
Tensor("dense_23/Relu:0", shape=(?, 29), dtype=float32)
epoch 0
iteration number0
loss 0.43287447
(?, 29)
(?, 29)
epoch 1
iteration number0
loss 0.40982887
(?, 29)
(?, 29)
epoch 2
iteration number0
loss 0.39938778
(?, 29)
(?, 29)
epoch 3
iteration number0
loss 0.3843758
(?, 29)
(?, 29)
epoch 4
iteration number0
loss 0.37673163
```

```
(?, 29)
(?, 29)
epoch 5
iteration number0
loss 0.3734822
(?, 29)
(?, 29)
epoch 6
iteration number0
loss 0.37235656
(?, 29)
(?, 29)
epoch 7
iteration number0
loss 0.37125868
(?, 29)
(?, 29)
epoch 8
iteration number0
loss 0.3702827
(?, 29)
(?, 29)
epoch 9
iteration number0
loss 0.3692525
(?, 29)
(?, 29)
epoch 10
iteration number0
loss 0.36820066
(?, 29)
(?, 29)
epoch 11
iteration number0
loss 0.36791378
(?, 29)
(?, 29)
epoch 12
iteration number0
loss 0.36705187
(?, 29)
(?, 29)
epoch 13
iteration number0
loss 0.3665546
(?, 29)
(?, 29)
epoch 14
```

```
iteration number0
loss 0.36628783
(?, 29)
(?, 29)
epoch 15
iteration number0
loss 0.36620545
(?, 29)
(?, 29)
epoch 16
iteration number0
loss 0.3660761
(?, 29)
(?, 29)
epoch 17
iteration number0
loss 0.36574745
(?, 29)
(?, 29)
epoch 18
iteration number0
loss 0.3655525
(?, 29)
(?, 29)
epoch 19
iteration number0
loss 0.36588037
(?, 29)
(?, 29)
(85443, 29)
test mse 0.6086097037513645
train mse 0.580954868454774
199008
0
```

5.2.3 Pure Keras

In [17]:
```
InputWidth = X_train.shape[1]
K.clear_session()
model = Sequential()
model.add(Dense(64, input_shape = (InputWidth,), activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(InputWidth, activation='relu'))
```

```python
print model.summary()
# Compile model
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
op = model.fit(X_train, X_train, validation_data=(X_test, X_test), epochs=30, batch_si
```

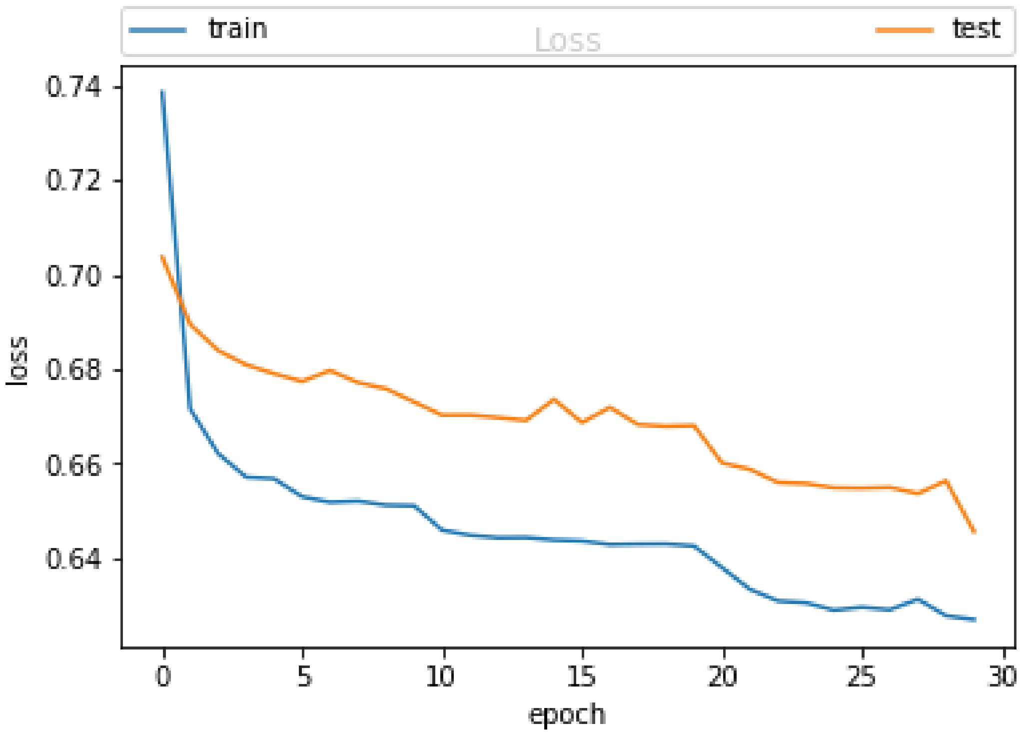
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 64)                1920
_________________________________________________________________
dense_2 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_3 (Dense)              (None, 16)                528
_________________________________________________________________
dense_4 (Dense)              (None, 16)                272
_________________________________________________________________
dense_5 (Dense)              (None, 32)                544
_________________________________________________________________
dense_6 (Dense)              (None, 64)                2112
_________________________________________________________________
dense_7 (Dense)              (None, 29)                1885
=================================================================
Total params: 9,341
Trainable params: 9,341
Non-trainable params: 0
_________________________________________________________________
None
Train on 199008 samples, validate on 85443 samples
Epoch 1/30
10s - loss: 0.7387 - acc: 0.6483 - val_loss: 0.7038 - val_acc: 0.7653
Epoch 2/30
9s - loss: 0.6716 - acc: 0.7754 - val_loss: 0.6895 - val_acc: 0.7953
Epoch 3/30
9s - loss: 0.6622 - acc: 0.7953 - val_loss: 0.6840 - val_acc: 0.8092
Epoch 4/30
10s - loss: 0.6572 - acc: 0.8069 - val_loss: 0.6809 - val_acc: 0.8201
Epoch 5/30
10s - loss: 0.6568 - acc: 0.8062 - val_loss: 0.6791 - val_acc: 0.8167
Epoch 6/30
11s - loss: 0.6530 - acc: 0.8155 - val_loss: 0.6774 - val_acc: 0.8266
Epoch 7/30
9s - loss: 0.6518 - acc: 0.8198 - val_loss: 0.6797 - val_acc: 0.7824
Epoch 8/30
10s - loss: 0.6521 - acc: 0.8185 - val_loss: 0.6772 - val_acc: 0.8182
Epoch 9/30
9s - loss: 0.6512 - acc: 0.8227 - val_loss: 0.6759 - val_acc: 0.8289
Epoch 10/30
9s - loss: 0.6511 - acc: 0.8237 - val_loss: 0.6731 - val_acc: 0.8175
```

```
Epoch 11/30
9s - loss: 0.6459 - acc: 0.8283 - val_loss: 0.6703 - val_acc: 0.8396
Epoch 12/30
9s - loss: 0.6448 - acc: 0.8333 - val_loss: 0.6703 - val_acc: 0.8402
Epoch 13/30
9s - loss: 0.6443 - acc: 0.8350 - val_loss: 0.6698 - val_acc: 0.8429
Epoch 14/30
9s - loss: 0.6444 - acc: 0.8345 - val_loss: 0.6692 - val_acc: 0.8448
Epoch 15/30
9s - loss: 0.6438 - acc: 0.8362 - val_loss: 0.6736 - val_acc: 0.7922
Epoch 16/30
9s - loss: 0.6436 - acc: 0.8359 - val_loss: 0.6687 - val_acc: 0.8427
Epoch 17/30
9s - loss: 0.6429 - acc: 0.8385 - val_loss: 0.6719 - val_acc: 0.8133
Epoch 18/30
9s - loss: 0.6430 - acc: 0.8379 - val_loss: 0.6683 - val_acc: 0.8466
Epoch 19/30
9s - loss: 0.6430 - acc: 0.8384 - val_loss: 0.6679 - val_acc: 0.8425
Epoch 20/30
9s - loss: 0.6426 - acc: 0.8393 - val_loss: 0.6681 - val_acc: 0.8431
Epoch 21/30
9s - loss: 0.6380 - acc: 0.8450 - val_loss: 0.6602 - val_acc: 0.8450
Epoch 22/30
9s - loss: 0.6334 - acc: 0.8485 - val_loss: 0.6588 - val_acc: 0.8507
Epoch 23/30
9s - loss: 0.6309 - acc: 0.8557 - val_loss: 0.6560 - val_acc: 0.8619
Epoch 24/30
9s - loss: 0.6305 - acc: 0.8568 - val_loss: 0.6558 - val_acc: 0.8626
Epoch 25/30
9s - loss: 0.6290 - acc: 0.8599 - val_loss: 0.6549 - val_acc: 0.8584
Epoch 26/30
9s - loss: 0.6296 - acc: 0.8574 - val_loss: 0.6548 - val_acc: 0.8636
Epoch 27/30
9s - loss: 0.6291 - acc: 0.8586 - val_loss: 0.6550 - val_acc: 0.8620
Epoch 28/30
9s - loss: 0.6313 - acc: 0.8530 - val_loss: 0.6537 - val_acc: 0.8674
Epoch 29/30
9s - loss: 0.6278 - acc: 0.8601 - val_loss: 0.6564 - val_acc: 0.8550
Epoch 30/30
9s - loss: 0.6270 - acc: 0.8551 - val_loss: 0.6456 - val_acc: 0.8690
```

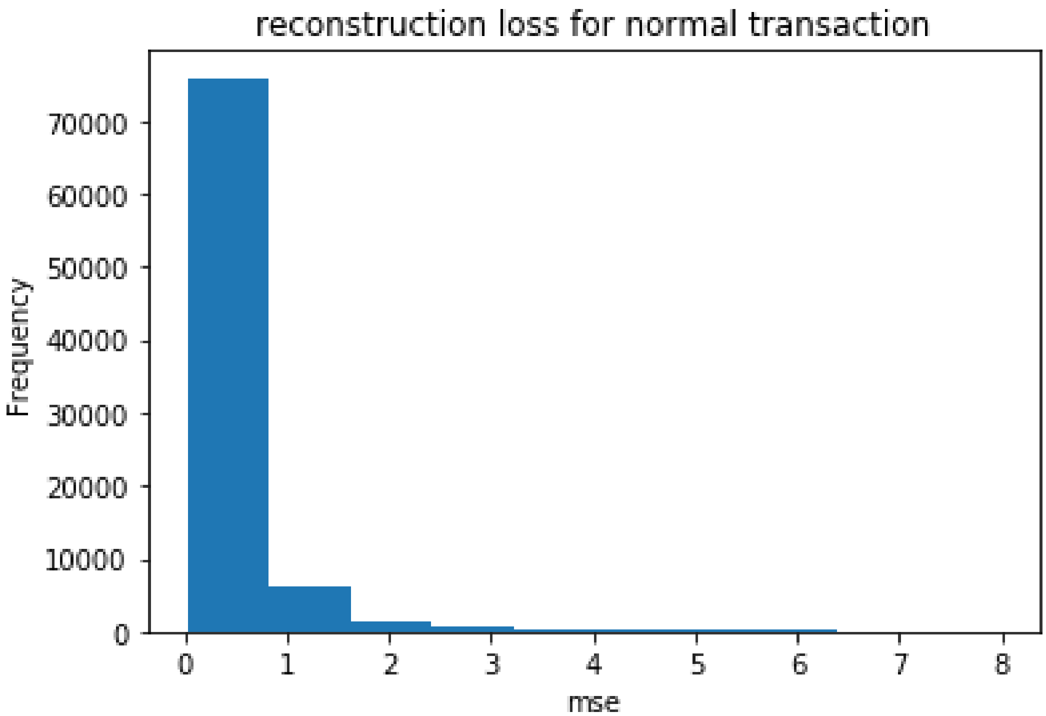
5.3 3. Model Performance

```
In [18]: plt.plot(op['loss'],label ='train')
         plt.plot(op['val_loss'],label ='test')
         plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,
                    ncol=2, mode="expand", borderaxespad=0.)
```

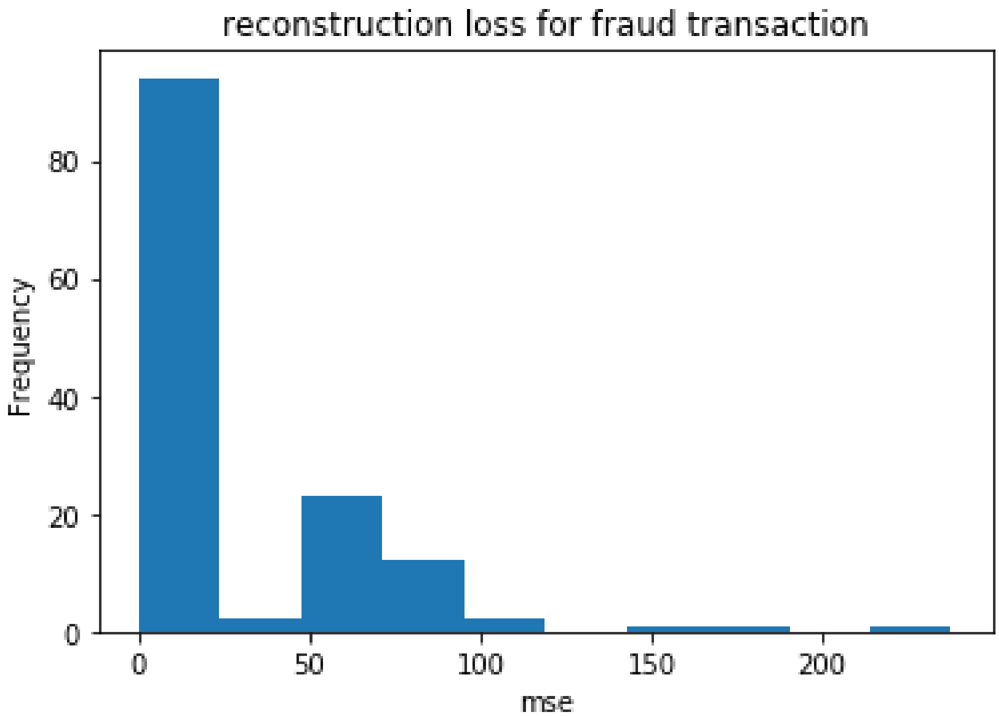
```
plt.title('Loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.show()
```




```
In [19]: y_pred = model.predict(X_test)
         mse = np.mean(np.power(X_test - y_pred, 2), axis=1)

In [23]: reconstructionLoss = pd.DataFrame({'ActualClass':y_test, 'mse':mse})
         Normal = reconstructionLoss[(reconstructionLoss['ActualClass'] == 0) & (reconstructio
         Fraud = reconstructionLoss[(reconstructionLoss['ActualClass'] == 1)]

In [24]: plt.hist(Normal['mse'],bins=10)
         plt.title("reconstruction loss for normal transaction")
         plt.xlabel("mse")
         plt.ylabel("Frequency")
         plt.show()
```



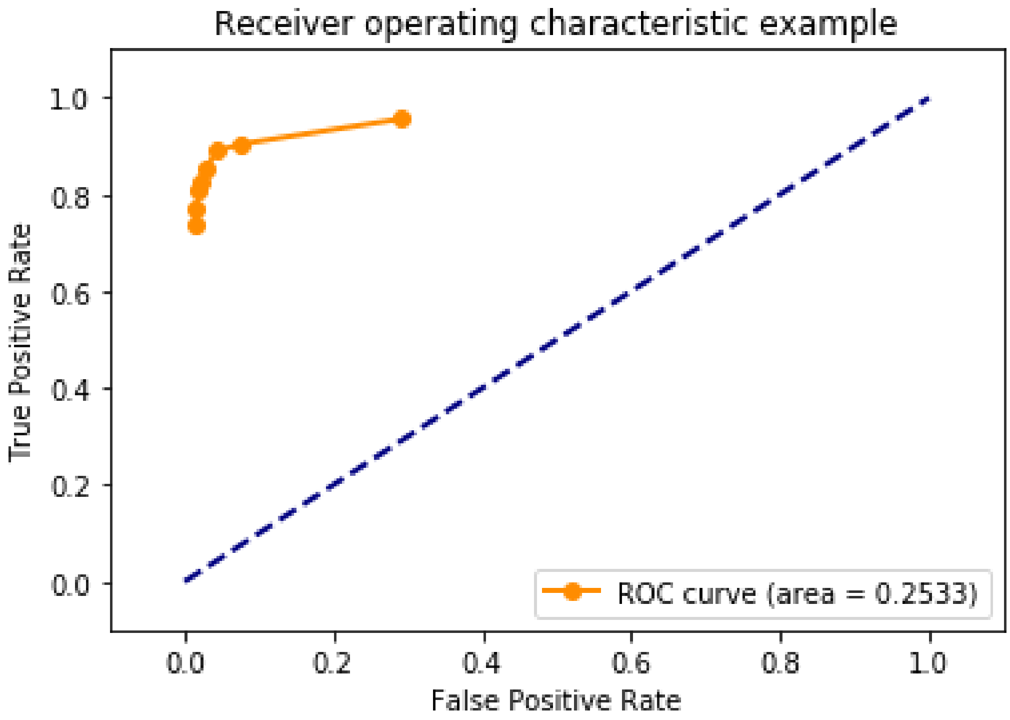
In [25]:
```
plt.hist(Fraud['mse'],bins=10)
plt.title("reconstruction loss for fraud transaction")
plt.xlabel("mse")
plt.ylabel("Frequency")
plt.show()
```




```
In [26]: def TN_TP(ytrue,ypred,l,tpe):
             count = 0
             for i in range(l):
                 if(ytrue[i] == tpe and ypred[i] == tpe):
                     count+=1
             return count

         def FN_FP(ytrue,ypred,l,tpe):
             count = 0
             for i in range(l):
                 if(ytrue[i] == tpe and ypred[i] != tpe):
                     count+=1
             return count

         def calculate_TPR_FPR(ytrue,ypred,P):
             TP = TN_TP(ytrue,ypred,len(P),1)
             TN = TN_TP(ytrue,ypred,len(P),0)
             FP = FN_FP(ytrue,ypred,len(P),0)
             FN = FN_FP(ytrue,ypred,len(P),1)

             TPR = float(TP)/(TP+FN)
             FPR = float(FP)/(FP+TN)

             return TPR,FPR
```

```python
def plot_roc(TPRFinal,FPRFinal):
    roc = auc(FPRFinal,TPRFinal)
    plt.figure()
    lw = 2
    plt.plot(FPRFinal, TPRFinal, color='darkorange',lw=lw,label='ROC curve (area = %0
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([-0.1, 1.1])
    plt.ylim([-0.1, 1.1])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()
```

In [27]:
```python
TPRFinal = []
FPRFinal = []
Treshold = [0.5,1,1.5,2,2.5,3,3.5,4]
for i in Treshold:
    y_pred = [1 if x>i else 0 for x in reconstructionLoss['mse']]
    TPR,FPR = calculate_TPR_FPR(reconstructionLoss['ActualClass'].values,y_pred,y_prec
    TPRFinal.append(TPR)
    FPRFinal.append(FPR)

#print TPRFinal
#print FPRFinal

plot_roc(TPRFinal,FPRFinal)
```



```
In [28]: Treshold = [0.5,1,1.5,2,2.5,3,3.5,4,4.5,5,5.5]
         for i in Treshold:
             y_pred = [1 if x>i else 0 for x in reconstructionLoss['mse']]
             print metrics.confusion_matrix(reconstructionLoss['ActualClass'], y_pred)
             print metrics.classification_report(reconstructionLoss['ActualClass'], y_pred)
```

```
[[60575 24732]
 [    6   130]]
             precision    recall  f1-score   support

          0       1.00      0.71      0.83     85307
          1       0.01      0.96      0.01       136

avg / total       1.00      0.71      0.83     85443

[[78740  6567]
 [   13   123]]
             precision    recall  f1-score   support

          0       1.00      0.92      0.96     85307
          1       0.02      0.90      0.04       136

avg / total       1.00      0.92      0.96     85443

[[81722  3585]
 [   15   121]]
             precision    recall  f1-score   support

          0       1.00      0.96      0.98     85307
          1       0.03      0.89      0.06       136

avg / total       1.00      0.96      0.98     85443

[[82783  2524]
 [   20   116]]
             precision    recall  f1-score   support

          0       1.00      0.97      0.98     85307
          1       0.04      0.85      0.08       136

avg / total       1.00      0.97      0.98     85443

[[83356  1951]
 [   24   112]]
             precision    recall  f1-score   support
```

```
          0       1.00      0.98      0.99     85307
          1       0.05      0.82      0.10       136

avg / total       1.00      0.98      0.99     85443

[[83695  1612]
 [   26   110]]
             precision    recall  f1-score   support

          0       1.00      0.98      0.99     85307
          1       0.06      0.81      0.12       136

avg / total       1.00      0.98      0.99     85443

[[83943  1364]
 [   31   105]]
             precision    recall  f1-score   support

          0       1.00      0.98      0.99     85307
          1       0.07      0.77      0.13       136

avg / total       1.00      0.98      0.99     85443

[[84157  1150]
 [   36   100]]
             precision    recall  f1-score   support

          0       1.00      0.99      0.99     85307
          1       0.08      0.74      0.14       136

avg / total       1.00      0.99      0.99     85443

[[84317   990]
 [   42    94]]
             precision    recall  f1-score   support

          0       1.00      0.99      0.99     85307
          1       0.09      0.69      0.15       136

avg / total       1.00      0.99      0.99     85443

[[84440   867]
 [   44    92]]
             precision    recall  f1-score   support

          0       1.00      0.99      0.99     85307
          1       0.10      0.68      0.17       136
```

```
avg / total       1.00      0.99      0.99     85443

[[84536   771]
 [   47    89]]
             precision    recall  f1-score   support

          0       1.00      0.99      1.00     85307
          1       0.10      0.65      0.18       136

avg / total       1.00      0.99      0.99     85443
```