Statistical learning: Second assignment

Ali Zamani(96123035)

December 14, 2018

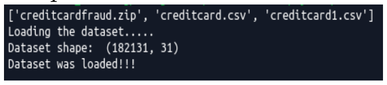
0.1 Visualize dataset

In this project I use creditcard dataset The dataset contains transactions made by credit cards in September 2013 by European cardholders over a two day period. There are 492 frauds out of a total 284,807 examples. Thus, the dataset is highly unbalanced, with the positive class (frauds) accounting for only 0.172% of all transactions. You can imagine that any such dataset would be highly unbalanced, as expected fraud or anomalous cases would only make up for a small percentage of the total transactions. Let's have look at our dataset.
I used seaborn and matplotlib to visualize dataset.

0.1.1 Import packages and dataset

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 14 13:06:00 2018

@author: ali(zamanilai1995@gmail.com)
"""
#%%Imort packages
import numpy as np # linear algebra
import seaborn as sns
sns.set(style='whitegrid')
import pandas as pd # data processing, CSV file I/O (e.g. pd.
    read_csv)
import matplotlib.pyplot as plt
#%%Check datasret
import os
print(os.listdir("../dataSet"))
#%%Read the data
print('Loading the dataset.....')
credit_card = pd.read_csv('../dataSet/creditcard.csv')
print('Dataset shape: ',credit_card.shape)
print('Dataset was loaded!!!')
```

Output:



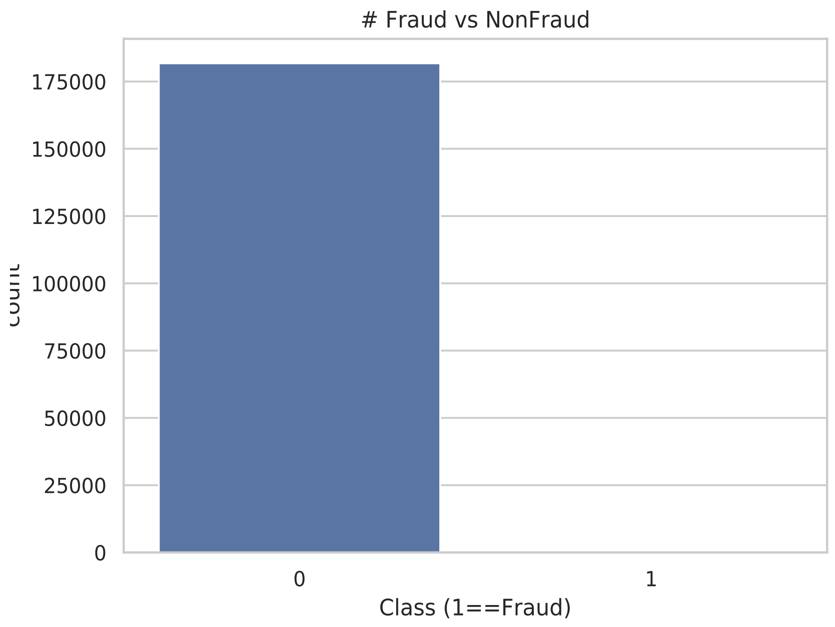
```
['creditcardfraud.zip', 'creditcard.csv', 'creditcard1.csv']
Loading the dataset.....
Dataset shape:  (182131, 31)
Dataset was loaded!!!
```

0.1.2 Balance of Data Visualization

Let's get a visual confirmation of the unbalanced data in this fraud dataset.

```
#%%Plot fraud vs nonfraud
f, ax = plt.subplots(figsize=(7, 5))
sns.countplot(x='Class', data=credit_card)
_ = plt.title('# Fraud vs NonFraud')
_ = plt.xlabel('Class (1==Fraud)')
plt.savefig("fraudvsnonfraud"+".pdf")
plt.show()
```

Output:

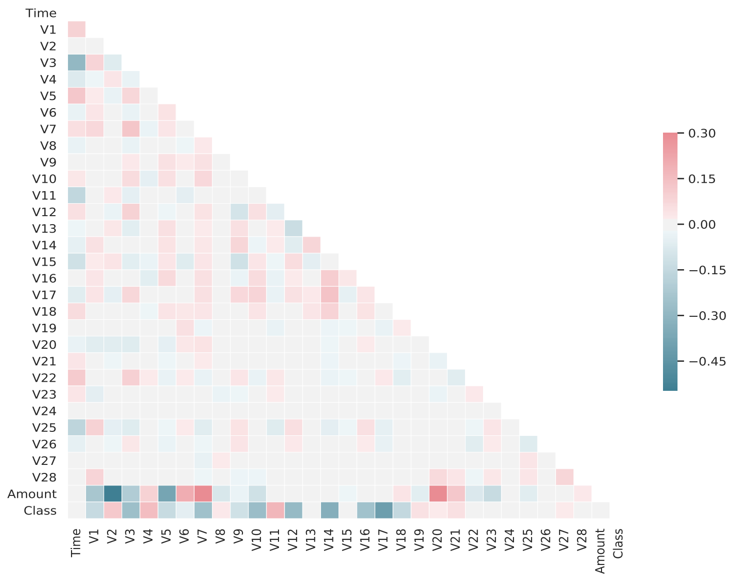


As you can see, the non-fraud cases strongly outweigh the fraud cases.

0.1.3 Heatmap

```
#%%Heatmap
corr=credit_card.corr()
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
cmap = sns.diverging_palette(220, 10, as_cmap=True)
# Draw the heatmap with the mask and correct aspect ratio
f, ax = plt.subplots(figsize=(11, 9))
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
plt.savefig("heatmap"+".pdf")
plt.show()
```

Output:



0.1.4 Fraud and non-fraud data describe

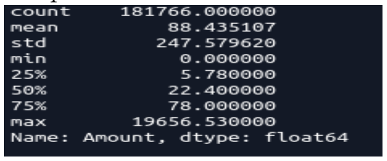
We will cut up the dataset into two data frames, one for non-fraud transactions and the other for fraud.

```
#%%Fraud and non-fraud data describe
non_fraud = credit_card[credit_card.Class == 0]
fraud = credit_card[credit_card.Class == 1]
```

Let's look at some summary statistics and see if there are obvious differences between fraud and non-fraud transactions.

```
non_fraud.Amount.describe()
```

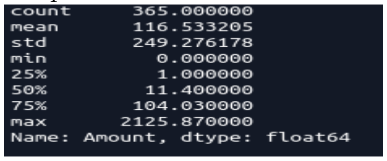
Output:



```
count     181766.000000
mean          88.435107
std          247.579620
min            0.000000
25%            5.780000
50%           22.400000
75%           78.000000
max        19656.530000
Name: Amount, dtype: float64
```

```
fraud.Amount.describe()
```

Output:



```
count      365.000000
mean       116.533205
std        249.276178
min          0.000000
25%          1.000000
50%         11.400000
75%        104.030000
max       2125.870000
Name: Amount, dtype: float64
```

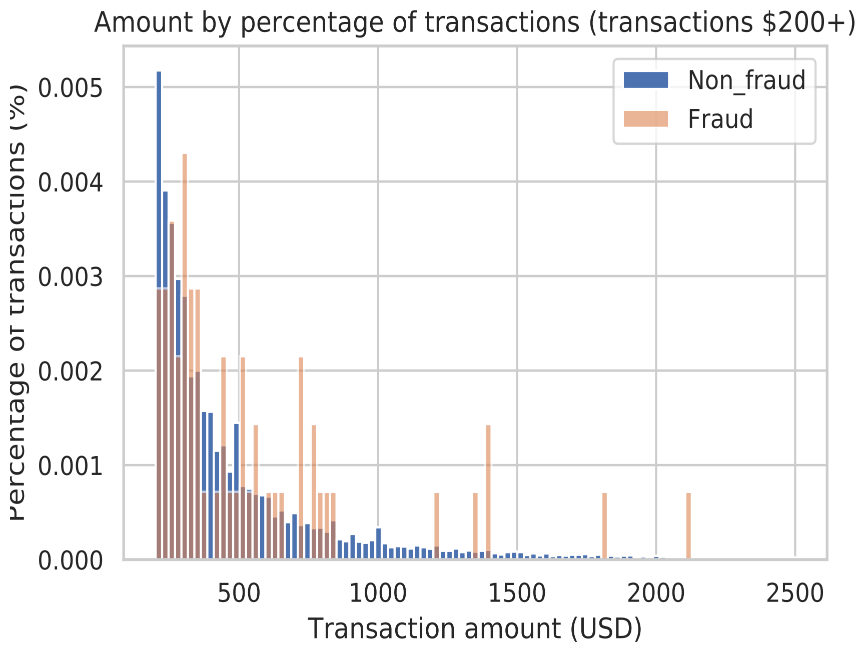
Although the mean is a little higher in the fraud transactions, it is certainly within a standard deviation and so is unlikely to be easy to discriminate in a highly precise manner between the classes with pure statistical methods. I could run statistical tests (e.g. t-test) to support the claim that the two samples likely come from populations with similar means and deviations. However, such statistical methods are not the focus of this article on autoencoders.

0.1.5 Visual Exploration of the Transaction Amount Data

We are going to get more familiar with the data and try some basic visuals. In anomaly detection datasets it is common to have the areas of interest "washed out" by abundant data. The most common method is to simply 'slice and dice' the data in a couple different ways until something interesting is found. Although this practice is common, it is not a scientifically sound way to explore data. There are always non-meaningful quirks to real data, so just looking until you "find something interesting" is likely going to result in you finding false positives. In other words, you find a random pattern in the current data set that will never be seen again. As a famous economist wrote, "If you torture the data long enough, it will confess."
In this dataset, I expect a lot of low-value transactions that will be generally uninteresting (buying cups of coffee, lunches, etc). This abundant data is likely to wash out the rest of the data, so I decided to look at the data in a number different $100 and $1,000 intervals. Since it would be tedious to show reader these graphs, I will only show the final graph that only visualizes the transactions above $200.

```
#%%plot of high value transactions
bins = np.linspace(200, 2500, 100)
plt.hist(non_fraud.Amount, bins, alpha=1, normed=True, label='
    Non_fraud')
plt.hist(fraud.Amount, bins, alpha=0.6, normed=True, label='Fraud')
plt.legend(loc='upper right')
plt.title("Amount by percentage of transactions (transactions \$200
    +)")
plt.xlabel("Transaction amount (USD)")
plt.ylabel("Percentage of transactions (%)");
plt.savefig("Amountbypercentageoftransactions"+".pdf")
plt.show()
```

Output:



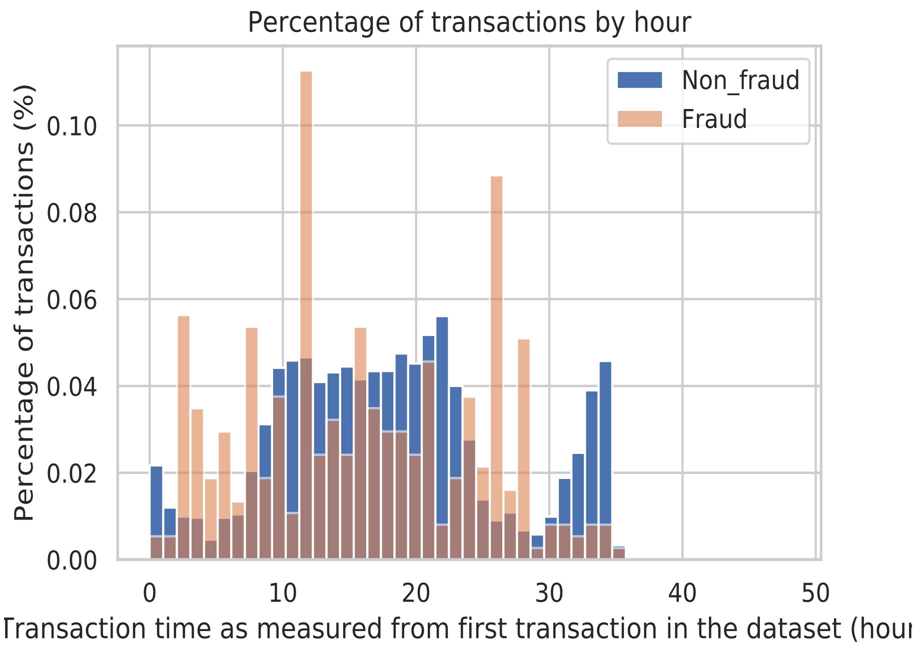
Since the fraud cases are relatively few in number compared to bin size, we see the data looks predictably more variable. In the long tail, especially, we are likely observing only a single fraud transaction. It would be hard to differentiate fraud from non-fraud transactions by transaction amount alone.

0.1.6 Visual Exploration of the Data by Hour

With a few exceptions, the transaction amount does not look very informative. Let's look at the time of day next.

```
#%%Visual Exploration of the Data by Hour
bins = np.linspace(0, 48, 48) #48 hours
plt.hist((non_fraud.Time/(60*60)), bins, alpha=1, normed=True,
    label='Non_fraud')
plt.hist((fraud.Time/(60*60)), bins, alpha=0.6, normed=True, label=
    'Fraud')
plt.legend(loc='upper right')
plt.title("Percentage of transactions by hour")
plt.xlabel("Transaction time as measured from first transaction in
    the dataset (hours)")
plt.ylabel("Percentage of transactions (%)");
plt.savefig("VisualExplorationoftheDatabyHour"+".pdf")
plt.show()
```

Output:



Hour ”zero” corresponds to the hour the first transaction happened and not necessarily 12-1am. Given the heavy decrease in non-fraud transactions from hours 1 to 8 and again roughly at hours 24 to 32, I am assuming those time correspond to nighttime for this dataset. If this is true, fraud tends to occur at higher rates during the night. Statistical tests could be used to give evidence for this fact, but are not in the scope of this article. Again, however, the potential time offset between normal and fraud transactions is not enough to make a simple, precise classifier. Next, we will explore the potential interaction between transaction amount and hour to see if any patterns emerge.

0.1.7 Visual Exploration of Transaction Amount vs. Hour

```
#%%Visual Exploration of Transaction Amount vs. Hour
plt.scatter((non_fraud.Time/(60*60)), non_fraud.Amount, alpha=0.6,
    label='non-fraud')
plt.scatter((fraud.Time/(60*60)), fraud.Amount, alpha=0.9, label='
    Fraud')
plt.title("Amount of transaction by hour")
plt.xlabel("Transaction time as measured from first transaction in
    the dataset (hours)")
plt.ylabel('Amount (USD)')
plt.legend(loc='upper right')
plt.savefig("VisualExplorationofTransactionAmountvsHour"+".pdf")
plt.show()
```

Output:
Again, this is not enough to make a good classifier. For example, it would be

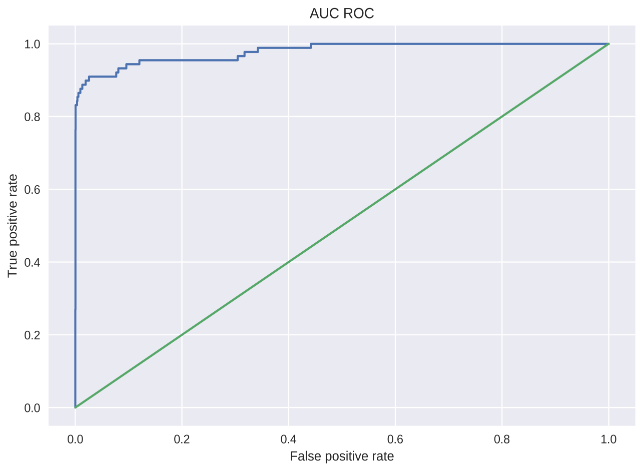
hard to draw a line that cleanly separates fraud and normal transactions. For the experienced Data Scientists in the readership, I am excluding more advanced techniques such as the kernel trick.

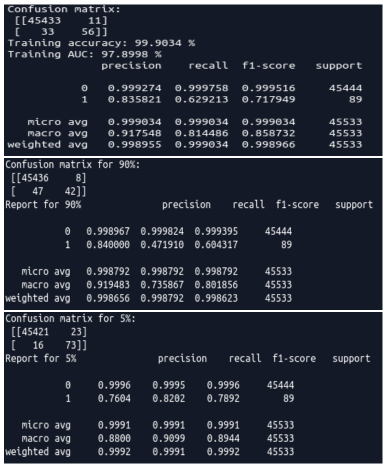
0.2 Logistic model with sklearn

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Dec  6 13:10:34 2018

@author: ali
"""
#%%
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.metrics import roc_curve, roc_auc_score,
    classification_report, accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import os
print(os.listdir("../dataSet"))
credit_card = pd.read_csv('../dataSet/creditcard.csv')
#%%
X = credit_card.drop(columns='Class', axis=1)
y = credit_card.Class.values
#%%
np.random.seed(42)
X_train, X_test, y_train, y_test = train_test_split(X, y)
#%%
scaler = StandardScaler()
lr = LogisticRegression()
model1 = Pipeline([('standardize', scaler),
                    ('log_reg', lr)])
model1.fit(X_train, y_train)
y_test_hat = model1.predict(X_test)
y_test_hat_probs = model1.predict_proba(X_test)[:,1]
test_accuracy = accuracy_score(y_test, y_test_hat)*100
test_auc_roc = roc_auc_score(y_test, y_test_hat_probs)*100
print('Confusion matrix:\n', confusion_matrix(y_test, y_test_hat))
print('Training accuracy: %.4f %%' % test_accuracy)
print('Training AUC: %.4f %%' % test_auc_roc)
print(classification_report(y_test, y_test_hat, digits=6))
fpr, tpr, thresholds = roc_curve(y_test, y_test_hat_probs,
    drop_intermediate=True)
f, ax = plt.subplots(figsize=(9, 6))
_ = plt.plot(fpr, tpr, [0,1], [0, 1])
_ = plt.title('AUC ROC')
_ = plt.xlabel('False positive rate')
_ = plt.ylabel('True positive rate')
plt.style.use('seaborn')

```

```python
plt.savefig('auc_roc.png', dpi=600)
y_hat_90 = (y_test_hat_probs > 0.90 )*1
print('Confusion matrix for 90%:\n', confusion_matrix(y_test,
    y_hat_90))
print('Report for 90%',classification_report(y_test, y_hat_90,
    digits=6))
y_hat_10 = (y_test_hat_probs > 0.05)*1
print('Confusion matrix for 5%:\n', confusion_matrix(y_test,
    y_hat_10))
print('Report for 5%',classification_report(y_test, y_hat_10,
    digits=4))
```

Results:





```
Confusion matrix:
 [[45433    11]
 [   33    56]]
Training accuracy: 99.9034 %
Training AUC: 97.8998 %
              precision    recall  f1-score   support

           0   0.999274  0.999758  0.999516     45444
           1   0.835821  0.629213  0.717949        89

   micro avg   0.999034  0.999034  0.999034     45533
   macro avg   0.917548  0.814486  0.858732     45533
weighted avg   0.998955  0.999034  0.998966     45533
```

```
Confusion matrix for 90%:
 [[45436     8]
 [   47    42]]
Report for 90%               precision    recall  f1-score   support

           0   0.998967  0.999824  0.999395     45444
           1   0.840000  0.471910  0.604317        89

   micro avg   0.998792  0.998792  0.998792     45533
   macro avg   0.919483  0.735867  0.801856     45533
weighted avg   0.998656  0.998792  0.998623     45533
```

```
Confusion matrix for 5%:
 [[45421    23]
 [   16    73]]
Report for 5%               precision    recall  f1-score   support

           0     0.9996    0.9995    0.9996     45444
           1     0.7604    0.8202    0.7892        89

   micro avg     0.9991    0.9991    0.9991     45533
   macro avg     0.8800    0.9099    0.8944     45533
weighted avg     0.9992    0.9991    0.9992     45533
```

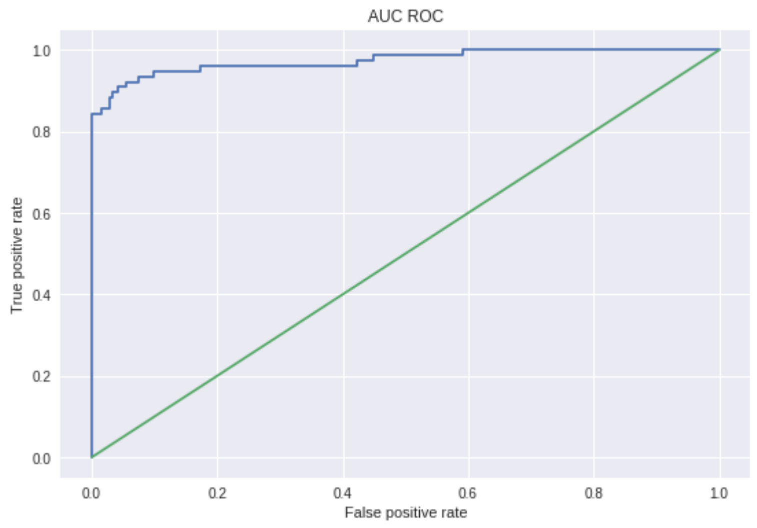
0.3 Logistic model with tensorflow

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
```

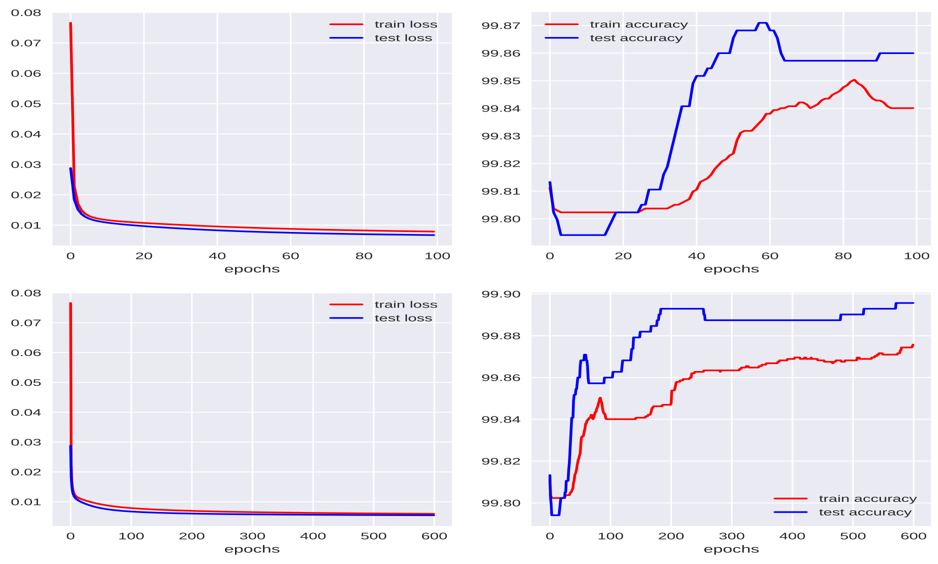
```
"""
Created on Fri Dec  7 05:47:51 2018

@author: ali
"""
#%%Parameters
train_set_num=.8
seed=5
#% Define the learning rate  batch_size etc.
learning_rate = 0.0003
batch_size = 1000
epoch_num = 600
#%%Imort packages
import numpy as np # linear algebra
import seaborn as sns
sns.set(style='whitegrid')
import pandas as pd # data processing, CSV file I/O (e.g. pd.
    read_csv)
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.metrics import roc_curve, roc_auc_score,
    classification_report, accuracy_score, confusion_matrix
#%% Define the normalized function
def min_max_normalized(data):
    col_max = np.max(data, axis=0)
    col_min = np.min(data, axis=0)
    col_mean = np.mean(data, axis=0)
    return np.divide(data - col_mean, col_max - col_min)
#%%Check datasret
import os
print(os.listdir("../dataSet"))
#%%Read the data
print('Loading the dataset.....')
credit_card = pd.read_csv('../dataSet/creditcard.csv')
print('Dataset shape: ',credit_card.shape)
print('Dataset was loaded!!!')
#%%
# set replace=False, Avoid double sampling
X = credit_card.drop(columns='Class', axis=1).values.reshape(-1,30)
y = credit_card.Class.values.reshape(-1,1)
train_index = np.random.choice(len(X), round(len(X) * train_set_num
    ),
                              replace=False)
test_index = np.array(list(set(range(len(X))) - set(train_index)))
train_X = X[train_index]
train_y = y[train_index]
test_X = X[test_index]
test_y = y[test_index]
#%% Normalized processing
train_X = min_max_normalized(train_X)
test_X = min_max_normalized(test_X)
#%%Build the model framework
# Begin building the model framework
# Declare the variables that need to be learned and initialization
# There are 30 features here, A's dimension is (30, 1)
w = tf.Variable(tf.random_normal(shape=[30, 1]))
b = tf.Variable(tf.random_normal(shape=[1, 1]))
```

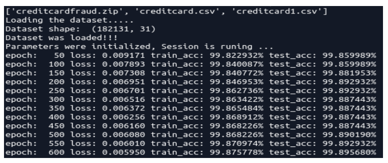
```
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
# Define placeholders
x = tf.placeholder(dtype=tf.float32, shape=[None, 30], name="x")
y = tf.placeholder(dtype=tf.float32, shape=[None, 1], name="y")
# Define logistic Regression
logit = tf.matmul(x, w) + b
y_predicted = 1.0 / (1.0 + tf.exp(-logit))
# Declare loss function
loss = -1 * tf.reduce_sum(y * tf.log(y_predicted) +
                          (1 - y) * tf.log(1 - y_predicted))
# Define optimizer: GradientDescent
optimizer = tf.train.GradientDescentOptimizer(
        learning_rate=learning_rate).minimize(loss)
# Define the accuracy
# The default threshold is 0.5, rounded off directly
prediction = tf.round(tf.sigmoid(logit))
# Bool into float32 type
correct = tf.cast(tf.equal(prediction, y), dtype=tf.float32)
# Average
accuracy = tf.reduce_mean(correct)
# End of the definition of the model framework
#label=[tf.count_nonzero(y) ,tf.subtract(tf.size(y),tf.
    count_nonzero(y))]
#confusion_matrix_tf = tf.confusion_matrix(labels=[10 ,100],
#                                          predictions=[2 ,108])
#FN=tf.metrics.false_negatives(labels=y, predictions=tf.round(
    y_predicted))
confiution=np.zeros(shape=[2,2])
#%%
print("Parameters were initialized, Session is runing ...")
train_error_list = []
train_acc_list = []
test_acc_list = []
test_error_list=[]
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for epoch in range(epoch_num):
        train_loss = 0
        for idx in range(len(train_X)//batch_size):
            input_list = {x: train_X[idx*batch_size:(idx+1)*
    batch_size],
                          y: train_y[idx*batch_size:(idx+1)*
    batch_size]}
            _, train_loss1 = sess.run([optimizer, loss], feed_dict=
    input_list)
            train_loss += train_loss1
        train_error_list.append(train_loss/len(train_X))
        train_acc_list.append(sess.run(
                accuracy, feed_dict={x: train_X, y:
                    train_y})*100)
        test_acc_list.append(sess.run(accuracy
                                      , feed_dict={x: test_X,
                                                   y:  test_y})
    *100)
        test_error_list.append(sess.run(loss,
                                       feed_dict={x: test_X,
                                                  y:test_y})/len(test_y))
        if (epoch + 1) % 50 == 0:
            print('epoch: {:4d} loss: {:5f} train_acc: {:5f}% test_acc: {:5f}%'
                  .format(epoch + 1, train_loss/len(train_X),
                          train_acc_list[epoch], test_acc_list[epoch]))
    w_value, b_value = sess.run([w, b])
    for i in range(len(train_X)):
        logit1 = np.matmul(train_X[i], w_value) + b_value
        if (np.round(1.0 / (1.0 + np.exp(-logit1)))):
#        if(sess.run(tf.round(y_predicted), feed_dict={x:train_X[i]})):
            if train_y[i]:
                confiution[1,1]+=1
            else:
                confiution[0,1]+=1
        else:
            if train_y[i]:
                confiution[1,0]+=1
            else:
                confiution[0,0]+=1
    print ('Confution matrix:',confiution)
#%%
train_y_hat=np.round(1.0/(1.0 + np.exp(-(np.matmul(train_X,w_value)+b_value))))
test_y_hat=np.round(1.0/(1.0 + np.exp(-(np.matmul(test_X,w_value)+b_value))))
test_accuracy = accuracy_score(test_y,test_y_hat)*100
test_auc_roc = roc_auc_score(test_y, test_y_hat)*100
print('Confusion matrix for train data:\n', confusion_matrix(test_y,
                                                              test_y_hat))
print('Confusion matrix for test data:\n', confusion_matrix(train_y,
                                                             train_y_hat))
print('Training accuracy: ',test_accuracy)
print('Training AUC: ', test_auc_roc)
print(classification_report(test_y, test_y_hat, digits=6))
fpr, tpr, thresholds = roc_curve(test_y, 1.0/(1.0 + np.exp(-(np.matmul(test_X,w_value)+b_value)))
                                 , drop_intermediate=True)
#select_tereshold=np.zeros_like(thresholds)
#recall=tpr/(tpr+fpr)
#precision=
#select_tereshold=2*(tpr*fpr)/(tpr+fpr)
#select_tereshold.append
f, ax = plt.subplots(figsize=(9, 6))
_ = plt.plot(fpr, tpr, [0,1], [0, 1])
_ = plt.title('AUC ROC')
_ = plt.xlabel('False positive rate')
_ = plt.ylabel('True positive rate')
plt.style.use('seaborn')
plt.savefig('auc_roc.png', dpi=600)
test_y_hat_10 = (1.0/(1.0 + np.exp(-(np.matmul(test_X,w_value)+
    b_value))) > 0.05 )*1
train_y_hat_10 = (1.0/(1.0 + np.exp(-(np.matmul(train_X,w_value)+
    b_value))) > 0.05 )*1
test_accuracy_10 = accuracy_score(test_y,test_y_hat_10)*100
test_auc_roc_10 = roc_auc_score(test_y, test_y_hat_10)*100
print('Confusion matrix for train data (0.05):\n', confusion_matrix
    (test_y,

    test_y_hat_10))
print('Confusion matrix for test data(0.05):\n', confusion_matrix(
    train_y,

    train_y_hat_10))
print('Training accuracy (0.05): ',test_accuracy_10)
print('Training AUC (0.05): ', test_auc_roc_10)
print(classification_report(test_y, test_y_hat_10, digits=6))
#%%
fig, ax = plt.subplots(2, 2, figsize=(10, 10))
fig.suptitle("test accuracy = " + str(test_acc_list[epoch]))
for a in ax.reshape(-1,1):
    a[0].set_xlabel("epochs")
ax[0][0].plot(train_error_list[:100], color='red', label='train
    loss')
ax[0][0].plot(test_error_list[:100], color='blue', label='test loss
    ')
ax[0][0].legend()
ax[1][0].plot(train_error_list, color='red', label='train loss')
ax[1][0].plot(test_error_list, color='blue', label='test loss')
ax[1][0].legend()
ax[0][1].plot(train_acc_list[:100], color='red', label='train
    accuracy')
ax[0][1].plot(test_acc_list[:100], color='blue', label='test
    accuracy')
ax[0][1].legend()
ax[1][1].plot(train_acc_list, color='red', label='train accuracy')
ax[1][1].plot(test_acc_list, color='blue', label='test accuracy')
ax[1][1].legend()
plt.savefig("trainandtest"+".pdf")
#End main program
```

Results:

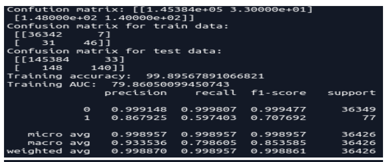


test accuracy = 99.89567995071411

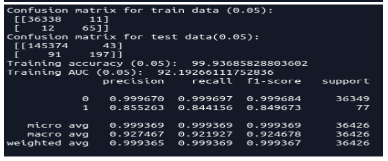




```
['creditcardfraud.zip', 'creditcard.csv', 'creditcard1.csv']
Loading the dataset.....
Dataset shape:  (182131, 31)
Dataset was loaded!!!
Parameters were initialized, Session is runing ...
epoch:   50 loss: 0.009171 train_acc: 99.822932% test_acc: 99.859989%
epoch:  100 loss: 0.007893 train_acc: 99.840087% test_acc: 99.859989%
epoch:  150 loss: 0.007308 train_acc: 99.840772% test_acc: 99.881953%
epoch:  200 loss: 0.006951 train_acc: 99.846953% test_acc: 99.892932%
epoch:  250 loss: 0.006701 train_acc: 99.862736% test_acc: 99.892932%
epoch:  300 loss: 0.006516 train_acc: 99.863422% test_acc: 99.887443%
epoch:  350 loss: 0.006372 train_acc: 99.865484% test_acc: 99.887443%
epoch:  400 loss: 0.006256 train_acc: 99.868912% test_acc: 99.887443%
epoch:  450 loss: 0.006160 train_acc: 99.868226% test_acc: 99.887443%
epoch:  500 loss: 0.006080 train_acc: 99.868226% test_acc: 99.890190%
epoch:  550 loss: 0.006010 train_acc: 99.870974% test_acc: 99.892932%
epoch:  600 loss: 0.005950 train_acc: 99.875778% test_acc: 99.895680%
```



```
Confution matrix: [[1.45384e+05 3.30000e+01]
 [1.48000e+02 1.40000e+02]]
Confusion matrix for train data:
 [[36342     7]
 [   31    46]]
Confusion matrix for test data:
 [[145384     33]
 [   148    140]]
Training accuracy:  99.89567891066821
Training AUC:  79.86050099450743
              precision    recall  f1-score   support

           0   0.999148  0.999807  0.999477     36349
           1   0.867925  0.597403  0.707692        77

   micro avg   0.998957  0.998957  0.998957     36426
   macro avg   0.933536  0.798605  0.853585     36426
weighted avg   0.998870  0.998957  0.998861     36426
```



```
Confusion matrix for train data (0.05):
 [[36338    11]
 [   12    65]]
Confusion matrix for test data(0.05):
 [[145374     43]
 [    91    197]]
Training accuracy (0.05):  99.93685828803602
Training AUC (0.05):  92.19266111752836
              precision    recall  f1-score   support

           0   0.999670  0.999697  0.999684     36349
           1   0.855263  0.844156  0.849673        77

   micro avg   0.999369  0.999369  0.999369     36426
   macro avg   0.927467  0.921927  0.924678     36426
weighted avg   0.999365  0.999369  0.999367     36426
```

0.4 compare sklearn and tensorflow results for logistic model

0.5 SVM with sklearn

0.6 SVM with tensorflow

0.7 Compare sklearn and tensorflow results for SVM