```
# lab of perceptron on 28 april
```

Perceptron

```
import numpy as np

class Perceptron:
  def __init__(self,learning_rate, n_iters, function):
    self.lr = learning_rate #learning rate
    self.n_iters = n_iters #number of iterations
    self.weight = None #weights
    self.bias = None #bias
    if function == 'step_function':
      self.activation_func = self._unit_step_func       #taking step function as activatio
    elif function =='sigmoid':
      self.activation_func = self._sigmoid_func       #taking sigmoid function as activati
    elif function == 'relu':
      self.activation_func = self._relu_func      #taking relu function as activation func
    elif function == 'tanh':
      self.activation_func = self._tanh_func      #taking tanh function as activation func

  def _unit_step_func(self,x):
    return np.where(x>=0, 1, 0)

  def _sigmoid_func(self,x):
    return 1/(1 + np.exp(-x))

  def _relu_func(self, x):
    return np.maximum(0,x)

  def _tanh_func(self,x):
    return (2/(1+np.exp((-2)*x)))-1

  def fit(self,X,y):
    #samples and features
    n_samples, n_features = X.shape

    #converting values to 1 and 0
    y_ = [1 if i>0 else 0 for i in y]

    #initializing weight and bias
    self.weight = np.zeros(n_features)
    self.bias = 0

    #gradient descenting
    for _ in range(self.n_iters):
      for idx, x_i in enumerate(X):
        lm = np.dot(x_i, self.weight) + self.bias
        y_prediction = self.activation_func(lm)

        #update rule
```

```
            #update rule
            update = self.lr*(y_[idx]- y_prediction)
            self.weight += update*x_i
            self.bias += update


    def predict(self,X):
      y_prediction = np.dot(X,self.weight) + self.bias
      return self.activation_func(y_prediction)


def accuracy(y_true, y_pred):
  return (np.sum(y_pred==y_true))/(len(y_true))
```

Above model creates a perceptron that can take different values for the learning rate, number of iterations and non-linear activation function such as

- Step Function (step_function)
- Signmoid Function (sigmoid)
- RELU Function (relu)
- Tanh Function (tanh)

Credit Card Fraud Detection

```
!wget https://raw.githubusercontent.com/nsethi31/Kaggle-Data-Credit-Card-Fraud-Detection/m
```

```
--2021-05-11 11:37:38--  https://raw.githubusercontent.com/nsethi31/Kaggle-Data-Cred:
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, :
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|
HTTP request sent, awaiting response... 200 OK
Length: 102634230 (98M) [text/plain]
Saving to: ‘creditcard.csv’

creditcard.csv      100%[===================>]  97.88M   122MB/s    in 0.8s

2021-05-11 11:37:40 (122 MB/s) - ‘creditcard.csv’ saved [102634230/102634230]
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
```

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

```
df.head()
```

	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	0.09
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.08
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.24
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.37
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.27

```
df.columns
```

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

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
```

```
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

```
df.isnull().sum()
```

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

There are no missing values in the dataset

```
df.Class.value_counts()
```

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

The given dataset is heavily imbalanced as seen from the counts, so undersampling method is used.

After doing so, there will be same number of 0's and 1's in the dataset.

Base Line Model

```
df1 = df.copy()
```

```
X = df1.drop("Class",axis =1)
y = df1.Class
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.3, random_state=42)
```

```
p = Perceptron(learning_rate=0.01, n_iters = 1000)
```

```
X_train = np.array(X_train)
y_train = np.array(y_train)
```

```
p.fit(X_train,y_train)
```

```
y_pred = p.predict(X_test)
#print("learning rate: " + str(i) + " no. of iterations: " + str(j))
print(accuracy(y_test, y_pred))
```

```
0.9982093325374811
```

The baseline model gives an accuracy of 99.82%, which is invalid since the dataset is imbalanced.

Models will be created using some undersampling methods

Undersampling

```
X = df.drop("Class",axis=1)
y = df.Class
```

```
from imblearn.under_sampling import NearMiss
nm = NearMiss(random_state=42)
X_res,y_res=nm.fit_sample(X,y)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning
  warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning
  warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning
  warnings.warn(msg, category=FutureWarning)
```

```
X_res.shape , y_res.shape
```

```
((984, 30), (984,))
```

```
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size = 0.3, random_
```

```
#List of values of hyperparameters used while training the model

learning_rate_list = [0.01,0.001,0.0001]
n_iters_list = [10,100,1000]
```

Step Function

```
accuracy_step_list=[]
learning_rate_list1 = []
n_iters_list1 = []
for i in learning_rate_list:
  for j in n_iters_list:
    p = Perceptron(i,j,function='step_function')
    p.fit(X_train,y_train)
    y_pred = p.predict(X_test)
    print("learning rate: " + str(i) + " no. of iterations: " + str(j))
    print(accuracy(y_test, y_pred)*100,'%')
    accuracy_step_list.append(accuracy(y_test, y_pred))
    learning_rate_list1.append(i)
    n_iters_list1.append(j)
```

```
learning rate: 0.01 no. of iterations: 10
51.35135135135135 %
learning rate: 0.01 no. of iterations: 100
53.71621621621622 %
learning rate: 0.01 no. of iterations: 1000
60.810810810810814 %
learning rate: 0.001 no. of iterations: 10
51.35135135135135 %
learning rate: 0.001 no. of iterations: 100
53.71621621621622 %
learning rate: 0.001 no. of iterations: 1000
60.810810810810814 %
learning rate: 0.0001 no. of iterations: 10
51.35135135135135 %
learning rate: 0.0001 no. of iterations: 100
53.71621621621622 %
learning rate: 0.0001 no. of iterations: 1000
60.810810810810814 %
```

```
def max_accuracy(accuracy_list, learning_rate_accuracy, no_iters_accuracy):
  max_accuracy = max(accuracy_list)
  i = accuracy_list.index(max_accuracy)
  lr = learning_rate_accuracy[i]
```

```
    n1 = no_iters_accuracy[i]
    print("Non-linear function: ",p.activation_func)
    print("Maximum accuracy: ",(max_accuracy)*100,"%\nLearning rate: ",lr,"\nnumber of itera
```

```
max_accuracy(accuracy_step_list,learning_rate_list1,n_iters_list1)
```

```
Non-linear function:  <bound method Perceptron._unit_step_func of <__main__.Perceptro
Maximum accuracy:  60.810810810810814 %
Learning rate:  0.01
number of iterations:  1000
```

Sigmoid Function

```
learning_rate_list2 = []
n_iters_list2 = []
accuracy_sigmoid_list=[]
for i in learning_rate_list:
  for j in n_iters_list:
    p = Perceptron(i,j,function='sigmoid')
    p.fit(X_train,y_train)
    y_pred = p.predict(X_test)
    print("learning rate: " + str(i) + " no. of iterations: " + str(j))
    print(accuracy(y_test, y_pred))
    accuracy_sigmoid_list.append(accuracy(y_test, y_pred))
    learning_rate_list2.append(i)
    n_iters_list2.append(j)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:22: RuntimeWarning: over
learning rate: 0.01 no. of iterations: 10
0.5135135135135135
learning rate: 0.01 no. of iterations: 100
0.7567567567567568
learning rate: 0.01 no. of iterations: 1000
0.668918918918919
learning rate: 0.001 no. of iterations: 10
0.5135135135135135
learning rate: 0.001 no. of iterations: 100
0.75
learning rate: 0.001 no. of iterations: 1000
0.5337837837837838
learning rate: 0.0001 no. of iterations: 10
0.5135135135135135
learning rate: 0.0001 no. of iterations: 100
0.5371621621621622
learning rate: 0.0001 no. of iterations: 1000
0.5337837837837838
```

```
max_accuracy(accuracy_sigmoid_list,learning_rate_list2,n_iters_list2)
```

```
Non-linear function:  <bound method Perceptron._sigmoid_func of <__main__.Perceptron
Maximum accuracy:  75.67567567567568 %
Learning rate:  0.01
number of iterations:  100
```

Relu Function

```
learning_rate_list3 = []
n_iters_list3 = []
accuracy_relu_list=[]
for i in learning_rate_list:
  for j in n_iters_list:
    p = Perceptron(learning_rate = i,n_iters=j,function='relu')
    p.fit(X_train,y_train)
    y_pred = p.predict(X_test)
    print("learning rate: " + str(i) + " no. of iterations: " + str(j))
    print(accuracy(y_test, y_pred))
    accuracy_relu_list.append(accuracy(y_test, y_pred))
    learning_rate_list3.append(i)
    n_iters_list3.append(j)
```

```
learning rate: 0.01 no. of iterations: 10
0.5067567567567568
learning rate: 0.01 no. of iterations: 100
0.5067567567567568
learning rate: 0.01 no. of iterations: 1000
0.5067567567567568
learning rate: 0.001 no. of iterations: 10
0.5067567567567568
learning rate: 0.001 no. of iterations: 100
0.5067567567567568
learning rate: 0.001 no. of iterations: 1000
0.5067567567567568
learning rate: 0.0001 no. of iterations: 10
0.5067567567567568
learning rate: 0.0001 no. of iterations: 100
0.5067567567567568
learning rate: 0.0001 no. of iterations: 1000
0.5067567567567568
```

```
max_accuracy(accuracy_relu_list,learning_rate_list3,n_iters_list3)
```

```
Non-linear function:  <bound method Perceptron._relu_func of <__main__.Perceptron obj
Maximum accuracy:  50.67567567567568 %
Learning rate:  0.01
number of iterations:  10
```

Tanh Function

```
learning_rate_list4 = []
n_iters_list4 = []
accuracy_tanh_list=[]
for i in learning_rate_list:
  for j in n_iters_list:
    p = Perceptron(learning_rate = i,n_iters=j,function='tanh')
    p.fit(X_train,y_train)
    y_pred = p.predict(X_test)
    print("learning rate: " + str(i) + " no. of iterations: " + str(j))
    print(accuracy(y_test, y_pred))
    accuracy_tanh_list.append(accuracy(y_test, y_pred))
    learning_rate_list4.append(i)
    n_iters_list4.append(j)
```

```
learning rate: 0.01 no. of iterations: 10
0.0
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: RuntimeWarning: over
learning rate: 0.01 no. of iterations: 100
0.006756756756756757
learning rate: 0.01 no. of iterations: 1000
0.02702702702702703
learning rate: 0.001 no. of iterations: 10
0.0
learning rate: 0.001 no. of iterations: 100
0.006756756756756757
learning rate: 0.001 no. of iterations: 1000
0.02702702702702703
learning rate: 0.0001 no. of iterations: 10
0.0
learning rate: 0.0001 no. of iterations: 100
0.006756756756756757
learning rate: 0.0001 no. of iterations: 1000
0.02702702702702703
```

```
max_accuracy(accuracy_tanh_list,learning_rate_list4,n_iters_list4)
```

```
Non-linear function:  <bound method Perceptron._tanh_func of <__main__.Perceptron obj
Maximum accuracy:  2.7027027027027026 %
Learning rate:  0.01
number of iterations:  1000
```

✓ 0s completed at 5:07 PM