Importing necessary Libraries

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.special import expit
import seaborn as sns
import numpy as np
from scipy import stats
```

Importing datasets

In [...
```
Churn=pd.read_csv("WA_Fn-UseC_-Telco-Customer-
Churn.csv")
CreditCard = pd.read_csv("creditcard.csv")
Adult=pd.read_csv("adult.csv")
```

Slicing the CreditCarddataset

In [3]:
```
CreditCard=CreditCard.sample(frac=0.50)
```

Using HotLabelEncoder

In ...
```
def Labelizer(df):
    from sklearn.preprocessing import LabelEncoder

Columns_list=list(df.select_dtypes(include='object').co
lumns)
    le = LabelEncoder()
    for x in df[Columns_list]:
        df[x] = le.fit_transform(df[x])
    return df
```

In [5]:
```
Train_Churn = Labelizer(Churn)
```

```
Train_adult = Labelizer(Adult)
```

LogisticRegression

In ...
```
class LogisticRegressionTanh:
      def __init__(self,lr=0.001,n_iters=1):
            self.lr =lr
            self.n_iters = n_iters
            self.weights = None
            self.bias = None

       #Tan hyperbolic Function
      def Tanh(self,x):
         tanh = (np.exp(x)-np.exp(-
x))/np.exp(x)+np.exp(-x)
         return tanh

     #Method for calculating the gradients


      def fit(self,x,y):
         m,n = x.shape
         self.weights =np.zeros(n)
         self.bias = 0
         for _ in range(self.n_iters):
             linear_model = np.dot(x,self.weights) +
self.bias
             y_pred = self.Tanh(linear_model)
             dw = (1/n)*np.dot(x.T,(y_pred-y))
             db = (1/n)*np.sum(y_pred-y)

             #Updating the weights
             self.weights -= self.lr*dw
             self.bias -= self.lr*db

      #Method to predict the class label.
      def predict(self, x):
         linear_model = np.dot(x,self.weights) +
self.bias
```

```
        y_pred = self.Tanh(linear_model)
        Y_pred_final = [1 if i >0.5 else 0 for i in
 y_pred]
        return np.array(Y_pred_final)
```

AdaBoost

In...
```
def adaboost_clf(Y_train, X_train, Y_test, X_test, M,
clf):
    n_train, n_test = len(X_train), len(X_test)
    # Initialize weights
    w = np.ones(n_train) / n_train
    pred_train, pred_test = [np.zeros(n_train),
np.zeros(n_test)]

    for i in range(M):
        # Fit a classifier with the specific weights
        model=clf()
        model.fit(X_train, Y_train)
        pred_train_i = model.predict(X_train)
        pred_test_i = model.predict(X_test)
        miss = [int(x) for x in (pred_train_i !=
Y_train)]
        # Equivalent with 1/-1 to update weights
        miss2 = [x if x==1 else -1 for x in miss]
        # Error
        err_m = np.dot(w,miss) / sum(w)
        # Alpha
        alpha_m = 0.5 * np.log( (1 - err_m) /
float(err_m))
        # New weights
        w = np.multiply(w, np.exp([float(x) * alpha_m
for x in miss2]))
        # Add to prediction
        pred_train = [sum(x) for x in zip(pred_train,[x
* alpha_m for x in pred_train_i])]
        pred_test = [sum(x) for x in zip(pred_test,[x *
alpha_m for x in pred_test_i])]
```

```
    pred_train, pred_test = np.sign(pred_train),
np.sign(pred_test)
    pred_train, pred_test = np.sign(pred_train),
np.sign(pred_test)
    # Return error rate in train and test set
    return pred_train, pred_test
```

Data Preprocessing

In
```
def DataPreprocessing(df,Target,n_components=6):
    """ This function will help to do A lot of Data
preprocessing  like Imbalance
    the classes of target column , Data Normalization,
Dimension Reduction and data
    spliting into training and testing. There are a 3
datesets so would be hard to preprocesse
    them separately but this function will help to do
preprocessing in single step
    """
    """This function will take 4 parameters
dataframe,Target COlumn ,Over, y_change and n_components
    """
    # Parameters Description:
    # df: Dataframe (data points)
    # Over : This is by default is false , If you want to
oversample sample Imbalance classe then you have to set it
True ,oTherwise it will be consider it as undersampling
    # n_components : Number of components  for Dimension
reduction by default it is 6.
    X = df.drop(Target,axis=1)
    y = df[Target]
    from imblearn.over_sampling import RandomOverSampler
    ros = RandomOverSampler()
    X_Resampled,y_Resampled=ros.fit_resample(X,y)
    from sklearn.preprocessing import StandardScaler
    Scaler=StandardScaler()
    X_Scaled=Scaler.fit_transform(X_Resampled)
    from sklearn.decomposition import PCA
    pca = PCA(n_components=n_components)
    X_decomposed = pca.fit_transform(X_Scaled)
```

```
    from sklearn.model_selection import train_test_split
    X_train,X_test,y_train,y_test =
train_test_split(X_decomposed,y_Resampled,test_size=0.20,r
andom_state=42)
    return X_train,X_test,y_train,y_test
```

Confusion Matirx

In [9]:
```
def  Confusion_matrix(y_actual, y_predicted):
    tp = 0
    tn = 0
    fp = 0
    fn = 0
    for i in range(len(y_actual)):
        if y_actual.iloc[i] > 0:
            if y_actual.iloc[i] == y_predicted[i]:
                tp = tp + 1
            else:
                fn = fn + 1
        if y_actual.iloc[i] < 1:
            if y_actual.iloc[i] == y_predicted[i]:
                tn = tn + 1
            else:
                fp = fp + 1
    cm = [[tn, fp], [fn, tp]]
    return tp,tn,fp,fn,cm
```

Model Results Saving Function

In [10]:
```
def accuracy(y_true,y_pred):
    accuracy = np.sum(y_true==y_pred)/len(y_true)*100
    return accuracy
```

In [...
```
def Model_Results(y_true,Testing_pred,Title1):
    Accuracy_test =(
np.sum(y_true==Testing_pred)/len(y_true))*100
```

```
tp,tn,fp,fn,cm=Confusion_matrix(y_true,Testing_pred)

    f1 = (tp /(tp+(1/2)*(fp+fn)))*100
    Recall = (tp/(tp+fn))*100
    TP_Rate = (tp/(tp+fn))*100
    TN_Rate = (tn/(tn+fp))*100
    FalseDisRate = (fp/(fp+tp+0.0000001))*100



    Frame=pd.DataFrame({"Accuracy":Accuracy_test,
                        "True Positive Rate":TP_Rate,
                        "True Negative Rate":TN_Rate,
                        "False Discovery
Rate":FalseDisRate,
                        "F1 Score":f1,
                        "Recall":Recall},index=[0]).T

    return Frame.rename({0:Title1},axis=1)
```

Doing data prerprocessing on Churn and splitting the data

In ...
```
X_train_ch,X_test_ch,y_train_ch,y_test_ch=DataPreproces
sing(Train_Churn,'Churn')
```

Building model adaboost with LogisticRegression

I...
```
Prediction_train_ch , Pred_Testing_ch =
adaboost_clf(y_train_ch, X_train_ch, y_test_ch,
X_test_ch,1000,LogisticRegressionTanh)
```

Saving the Churn prediction Results

In ...
```
Testing_Churn=Model_Results(y_test_ch,Pred_Testing_ch,
"Testing")
Training_Churn=Model_Results(y_train_ch,Prediction_tra
in_ch,"Training")
Churn_Results=pd.concat([Training_Churn,Testing_Churn]
,axis=1)
Churn_Results.to_csv("Churn_Results.csv")
Churn_Results
```

Out[33]:

	Training	Testing
Accuracy	72.879923	73.768116
True Positive Rate	66.254545	67.302193
True Negative Rate	79.460631	80.411361
False Discovery Rate	23.786949	22.075055
F1 Score	70.885748	72.225064
Recall	66.254545	67.302193

Number of Boosting Rounds On Churn

I...
```
Accuracy_Training =[]
Accuracy_Testing = []

for i in [5,10,15,20]:
    Prediction_train ,Predion_test =
adaboost_clf(y_train_ch, X_train_ch, y_test_ch,
X_test_ch,i,LogisticRegressionTanh)
    A_Test = accuracy(y_test_ch,Predion_test)
    A_Train = accuracy(y_train_ch,Prediction_train)
```

```
        Accuracy_Training.append(A_Train)
        Accuracy_Testing.append(A_Test)
```

In [16]:
```
Rounds = [5,10,15,20]
df = pd.DataFrame()
df["Number of Boosting Rounds"] = Rounds
df["Training"] = Accuracy_Training
df["Testing"]  = Accuracy_Testing
df.to_csv("Number of Boosting Rounds on Churn.csv")
df
```

Out[16]:

	Number of Boosting Rounds	Training	Testing
0	5	72.879923	73.768116
1	10	72.879923	73.768116
2	15	72.879923	73.768116
3	20	72.879923	73.768116

Data Preprocessing on Adult Dataset

In ...
```
X_train_a,X_test_a,y_train_a,y_test_a=DataPreprocessing
(Train_adult,'income')
```

Building the model adaboost with logisticRegression on Adult

I...
```
Prediction_train_a, Pred_Testing_a =
adaboost_clf(y_train_a, X_train_a, y_test_a,
X_test_a,100,LogisticRegressionTanh)
```

Saving the Adult prediction

Results

In [...
```
Testing_Adult=Model_Results(y_test_a,Pred_Testing_a,"T
esting")
Training_Adult=Model_Results(y_train_a,Prediction_trai
n_a,"Training")
Adult_Results=pd.concat([Training_Adult,Testing_Adult]
,axis=1)
Adult_Results.to_csv("Adult_Results.csv")
Adult_Results
```

Out[32]:

	Training	Testing
Accuracy	64.527710	64.269822
True Positive Rate	35.028049	34.725319
True Negative Rate	94.060207	93.683148
False Discovery Rate	14.484886	15.449161
F1 Score	49.698838	49.231211
Recall	35.028049	34.725319

Number of Boosting Rounds on Adult

I...
```
Accuracy_Training =[]
Accuracy_Testing = []

for i in [5,10,15,20]:
    Prediction_train ,Predion_test =
adaboost_clf(y_train_a, X_train_a, y_test_a,
X_test_a,i,LogisticRegressionTanh)
    A_Test = accuracy(y_test_a,Predion_test)
    A_Train = accuracy(y_train_a,Prediction_train)
    Accuracy_Training.append(A_Train)
    Accuracy_Testing.append(A_Test)
```

In [24]:
```
Rounds = [5,10,15,20]
df = pd.DataFrame()
df["Number of Boosting Rounds"] = Rounds
df["Training"] = Accuracy_Training
df["Testing"]  = Accuracy_Testing
df.to_csv("Number of Boosting Rounds on Adult.csv")
df
```

Out[24]:

	Number of Boosting Rounds	Training	Testing
0	5	64.52771	64.269822
1	10	64.52771	64.269822
2	15	64.52771	64.269822
3	20	64.52771	64.269822

Data Preprocessing on CreditCard and Spliting the data set

In ...
```
X_train_Cr,X_test_Cr,y_train_Cr,y_test_Cr=DataPreprocesing(CreditCard,'Class')
```

Building adaboost model with logisticRegression on CreditCard

I...
```
Prediction_train_Cr, Pred_Testing_Cr = adaboost_clf(y_train_Cr, X_train_Cr, y_test_Cr, X_test_Cr,100,LogisticRegressionTanh)
```

Saving the CreditCard PredictionResults

In ...
```
Testing_Credit=Model_Results(y_test_Cr,Pred_Testing_Cr
,"Testing")
Training_Credit=Model_Results(y_train_Cr,Prediction_tr
ain_Cr,"Training")
Credit_Results=pd.concat([Training_Credit,Testing_Cred
it],axis=1)
Credit_Results.to_csv("Credit_Results.csv")
Credit_Results
```

Out[31]:

	Training	Testing
Accuracy	80.719216	80.762467
True Positive Rate	61.456492	61.537109
True Negative Rate	99.983294	99.982418
False Discovery Rate	0.027174	0.028571
F1 Score	76.119744	76.181145
Recall	61.456492	61.537109

Number of Boosting Rounds on CreditCard

I...
```
Accuracy_Training =[]
Accuracy_Testing = []
for i in [5,10,15,20]:
    Prediction_train ,Predion_test =
adaboost_clf(y_train_Cr, X_train_Cr, y_test_Cr,
X_test_Cr,i,LogisticRegressionTanh)
    A_Test = accuracy(y_test_Cr,Predion_test)
    A_Train = accuracy(y_train_Cr,Prediction_train)
```

```
    Accuracy_Training.append(A_Train)
    Accuracy_Testing.append(A_Test)
```

In [30]:
```
Rounds = [5,10,15,20]
df = pd.DataFrame()
df["Number of Boosting Rounds"] = Rounds
df["Training"] = Accuracy_Training
df["Testing"]  = Accuracy_Testing
df.to_csv("CreditCard Number of Boosting.csv")
df
```

Out[30]:

	Number of Boosting Rounds	Training	Testing
0	5	80.719216	80.762467
1	10	80.719216	80.762467
2	15	80.719216	80.762467
3	20	80.719216	80.762467