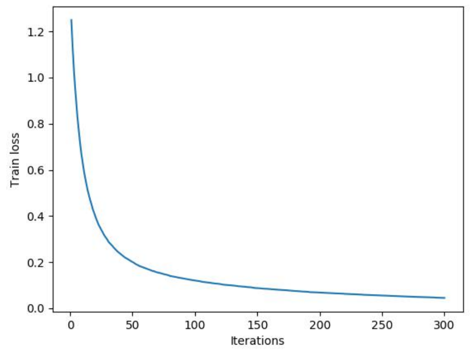
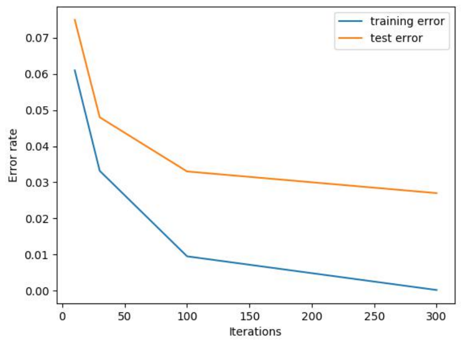
A)

In this assignment, the loss function was chosen as deviant(logistic) loss function. The learning rate has been chosen as 0.1 to get the best performance
The training loss vs iteration number has been plotted as below when k=300:



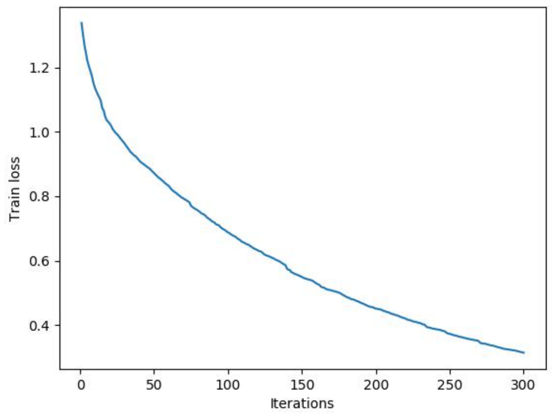
The misclassification error vs the number of iteration has been plotted below:



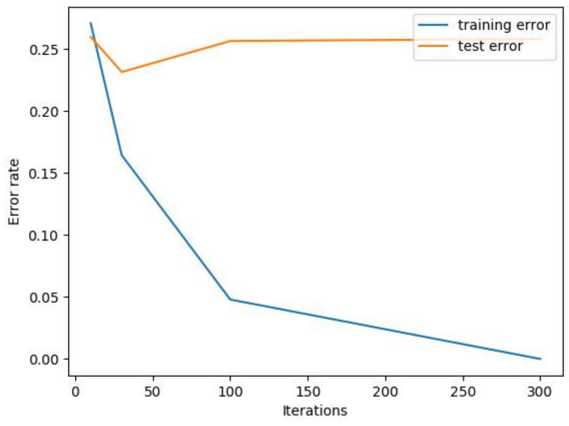
The detailed result has been shown in the table below (together with the result of madelon and arcene dataset):

Iteration number	Train error (gissette)	Test error (gissette)	Train error (madelon)	Test error (madelon)	Train error (arcene)	Test error (arcene)
10	0.061	0.075	0.271	0.26	0	0.28
30	0.033	0.048	0.1645	0.232	0	0.3
100	0.0095	0.033	0.048	0.257	0	0.27
300	0.00017	0.027	0.0	0.258	0	0.29

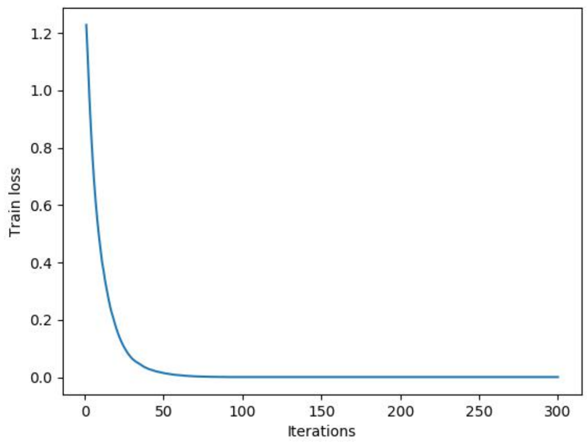
B)
1)The algorithm has been used on madelon data set.
The training loss vs iteration number has been plotted as below when k=300:



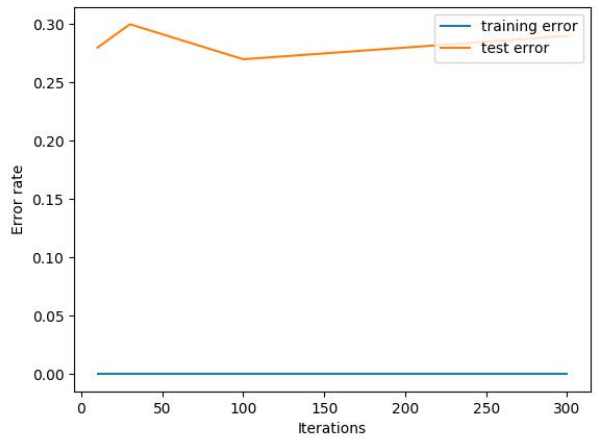
The misclassification error vs the number of iterations has been plotted below:



2)The algorithm has been used on arcene data set.
The training loss vs iteration number has been plotted as below when k=300:



The misclassification error vs the number of iterations has been plotted below:



The code is as below:

```
import numpy as np
import heapq
import pandas as pd
import math
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn import ensemble
from scipy import stats
# reading data from the file
train_data=pd.read_csv('gisette_train.data', sep=' ', header=None).dropna(1)
train_labels=pd.read_csv('gisette_train.labels', sep=' ', header=None)
test_data=pd.read_csv('gisette_valid.data', sep=' ', header=None).dropna(1)
test_labels=pd.read_csv('gisette_valid.labels', sep=' ', header=None)
#normalize the data to mean 0 and std 1
def normalize(train, test):
    mean=np.mean(train, axis=0)
    std= np.std(train, axis=0)
    train=(train-mean)/(std+1e-7)
    test=(test-mean)/(std+1e-7)
    return train, test

train_data_norm, test_data_norm =normalize(train_data, test_data)

#take out the data size


#add one extra column 1s at the beginning of the data
train_data =train_data_norm
test_data = test_data_norm

iteration_list=[10,30,100,300]
train_err_list=[]
test_err_list=[]

for i in iteration_list:
    #training the logistic boosting
    logboost=ensemble.GradientBoostingClassifier(loss='deviance',
learning_rate=0.1, n_estimators=i)
    logboost.fit(train_data, train_labels.values.ravel())

    #predicting
    train_pred=logboost.predict(train_data)
    test_pred=logboost.predict(test_data)

    #evaluate the error
    train_error=1-accuracy_score(train_labels, train_pred)
    test_error=1-accuracy_score(test_labels, test_pred)
    if(i==300):
        plt.plot(np.arange(i) + 1, logboost.train_score_)
        plt.xlabel('Iterations')
        plt.ylabel('Train loss')
        plt.show()
    print('iterations: ',i, 'train error: ', train_error, 'test error: ', 
test_error)
    #recording
    train_err_list.append(train_error)
    test_err_list.append(test_error)


plt.plot(iteration_list, train_err_list,label='training error')
plt.plot(iteration_list, test_err_list,label='test error')
plt.xlabel('Iterations')
plt.ylabel('Error rate')
plt.legend(loc=1)
plt.show()
```