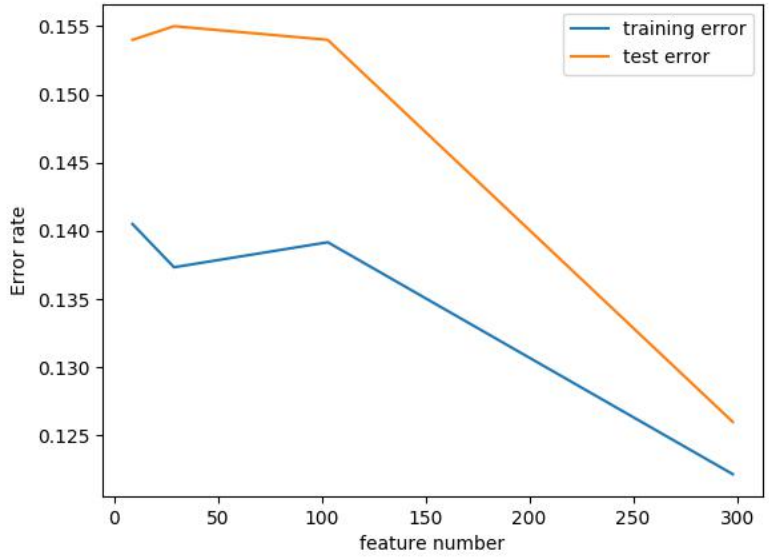
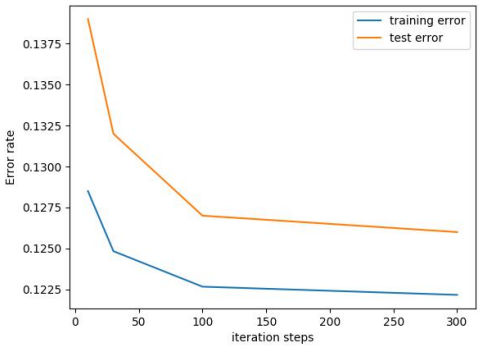
A)
The TISP algorithm has been implemented. The parameter eta was selected to be 0 and the step size was set as 0.01 to make the iteration stable.
The misclassification error vs the number of selected features has been plotted below:



For each experiment, the descending of error rate has been observed as the number of iterations increase, as shown in the figure below:

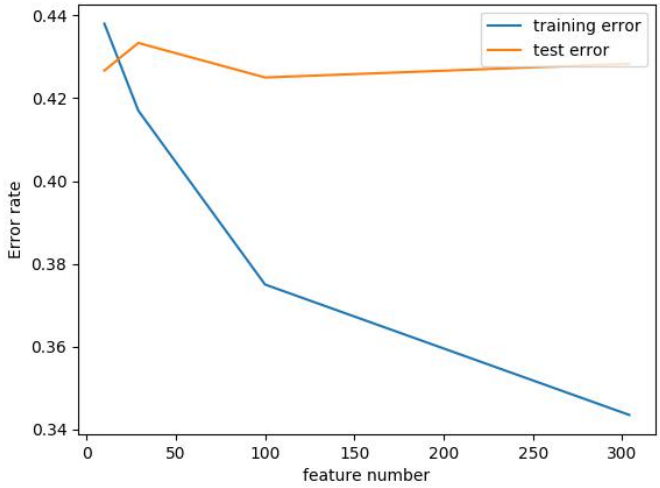


Which indicates that the method has been successfully implemented

The detailed result has been shown in the table below:

Parameter number	Train error	Test error	Lambda value
9	0.1405	0.154	0.0054
29	0.1373	0.155	0.005
103	0.1391	0.154	0.0036
298	0.1222	0.126	0.00215

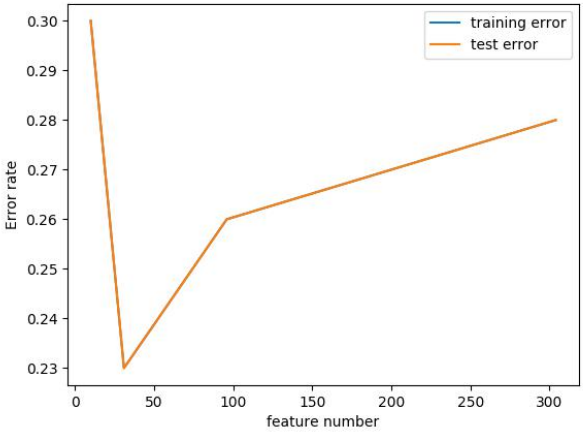
B)
1)
The algorithm has been used on madelon data set. The misclassification error vs the number of selected features has been plotted below:



The detailed result has been shown in the table below:

Parameter number	Train error	Test error	Lambda value
10	0.438	0.427	0.0115
29	0.417	0.433	0.005
100	0.375	0.425	0.0034
304	0.345	0.428	0.0015

2)
The algorithm has been used on arcene data set. The misclassification error vs the number of selected features has been plotted below:



The detailed result has been shown in the table below:

Parameter number	Train error	Test error	Lambda value
10	0.3	0.3	0.0043
31	0.23	0.23	0.0039
96	0.26	0.26	0.00359
304	0.28	0.28	0.00311

The code is as below:

```
import numpy as np
import pandas as pd
import math
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn import preprocessing
from scipy import stats
# reading data from the file
train_data=pd.read_csv('gisette_train.data', sep=' ', header=None).dropna(1).as_matrix()
train_labels=pd.read_csv('gisette_train.labels', sep=' ', header=None).as_matrix()
test_data=pd.read_csv('gisette_valid.data', sep=' ', header=None).dropna(1).as_matrix()
test_labels=pd.read_csv('gisette_valid.labels', sep=' ', header=None).as_matrix()
#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
N =  train_data_norm.shape[0] #row size
NN = train_data_norm.shape[1] #column size
TN = test_data_norm.shape[0]


#add one extra column 1s at the beginning of the data
train_data = np.hstack((np.ones((N, 1)), train_data_norm))
test_data = np.hstack((np.ones((TN, 1)), test_data_norm))



def penalty_theta(x, _lambda, _yita):
    xx = stats.threshold(x, threshmin = _lambda)
    xxx = stats.threshold(x, threshmax = - _lambda)
    x = xx + xxx
    return np.count_nonzero(x), x / ( 1 + _yita)

def iteration_steps(train_data_, train_label_, w, steps, _lambda, param,step_size):
    n=0
    N = train_data_.shape[0]
    for _ in range(steps):
        w_temp = w + np.transpose(np.transpose(train_data_).dot(train_label_- 1 / ( 1 +
np.exp(-train_data_.dot(np.transpose(w))))))/N*step_size
        n,w_temp2 = penalty_theta(w_temp, _lambda, param)
        w = w_temp2
    return n,w

def linear_regression_predit(w, test_data_):
    results = []
    expvalue = np.exp(test_data_.dot(np.transpose(w)))
    p = expvalue / ( 1 + expvalue)
    for i in p:
        if i > 0.5:
            results.append(1)
        else:
            results.append(-1)
    return results

# 1, 0.2
#_lambda = 0.0054
param = 0
step_size=0.01
likehood_list=[]
xlabel=[]
train_err_list = []
test_err_list=[]
iteration=100
lambda_list=[0.0054, 0.005,0.0036,0.00215]
para_num=0

for i in lambda_list:
```

```
    w = np.zeros(NN + 1)
    w = np.expand_dims(w, axis=0)
    para_num,w = iteration_steps(train_data, train_labels, w, iteration, i, param,
step_size)
    xlabel.append(para_num)
    #predicting
    train_pred=np.asarray(linear_regression_predit(w, train_data))
    test_pred=np.asarray(linear_regression_predit(w, test_data))
    train_error=1-accuracy_score(train_labels, train_pred)
    test error=1-accuracy score(test labels, test pred)
    print('parameter number: ',para num, 'train error: ', train error, 'test error: ',
test_error, 'lambda value: ', i)
    #recording
    train_err_list.append(train_error)
    test_err_list.append(test_error)


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