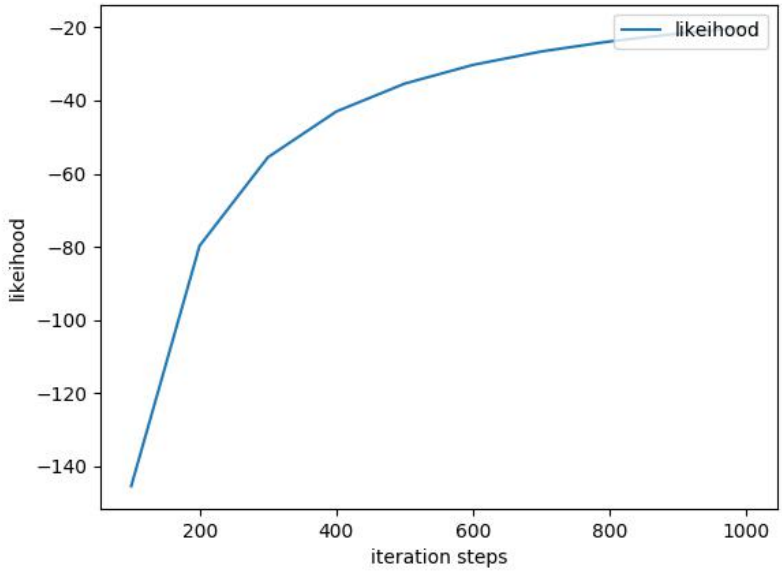
A) Using logistic regression to analyze gisette data set.

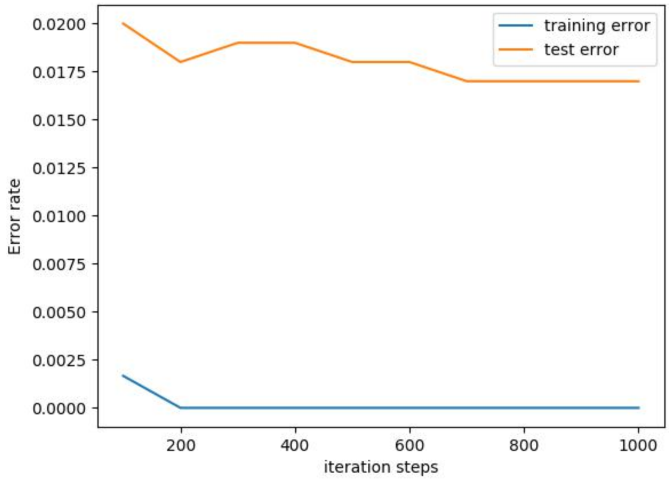
The step size has been chosen to be 0.1. The following figure shows the log-likelihood vs iteration:



The test error and training error are shown as following:

iteration number	train error	test error
100	0.00166666666667	0.02
200	0	0.018
300	0	0.019
400	0	0.019
500	0	0.018
600	0	0.018
700	0	0.017
800	0	0.017
90	0	0.017
1000	0	0.017

Which has been plotted in the graph:



The code is given as following:

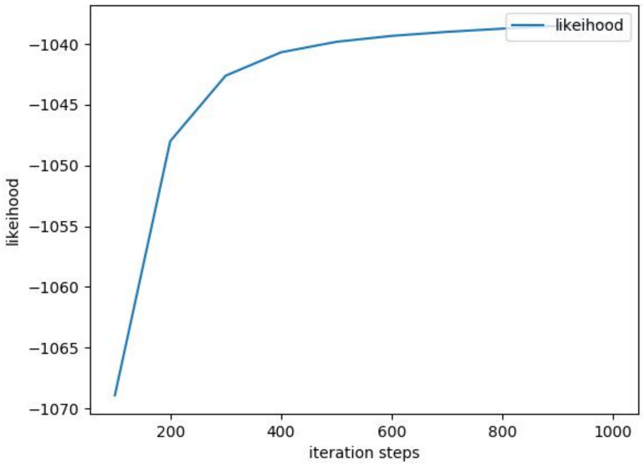
```
import numpy as np
import pandas as pd
import math
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn import preprocessing

# 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
train_data_norm = preprocessing.scale(train_data)
test_data_norm = preprocessing.scale(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))

#functioin for the partial derivative
def partial_derivative(train_data_, train_label_, w):
    N = train_data_.shape[0]
    cofficiency = -train_data_ * train_label_
    exp_number = -train_label_ * np.dot(train_data_, np.transpose(w))
    derivative = np.sum(cofficiency * np.exp(exp_number) / ( 1 + np.exp(exp_number)), axis=0)
    return np.expand_dims(derivative, axis=0)

def gradient_descent(train_data_, train_label_, w, steps, _lambda, param):
    N = train_data_.shape[0]
    for _ in range(steps):
        w_temp = w - param * _lambda * w - param/N * partial_derivative(train_data_,
train_label_, w)
        w = w_temp
    return w

def log_likehood(train_data_, train_label_, w):
    likeihood = -np.sum(np.log(1 + np.exp(-train_label_ * np.dot(train_data_,
np.transpose(w)))))
    return likeihood

def linear_regression_predit(w, test_data_):
    N = test_data_.shape[0]
    results = []
    for i in range(N):
        expvalue = math.exp(np.inner(test_data_[i, :],w))
        p = expvalue / ( 1 + expvalue)
```

```
        if p > 0.5:
            results.append(1)
        else:
            results.append(-1)
    return results

_lambda = 0.001#0.001
param = 0.1
likehood_list=[]
likehood_xlabel=[]
train_err_list = []
test_err_list=[]
w = np.zeros(NN + 1)
w=np.expand_dims(w, axis=0)
iteration_list = [100,  100, 100, 100, 100, 100, 100, 100, 100, 100]
steps_ = 0

for i in iteration_list: #range(300, 1001):
    steps_ = steps_ + i
    w = gradient_descent(train_data, train_labels, w, i, _lambda, param)
    likehood = log_likehood(train_data, train_labels, w)
    likehood_list.append(likehood)
    likehood_xlabel.append(steps_)
    #predicting
    train_pred=np.asarray(linear_regression_predit(w, train_data))
    # for i in range(1000):
    #     print(train_pred[i])
    #     print(train_labels[i])
    test_pred=np.asarray(linear_regression_predit(w, test_data))
    # print test_data.shape
    # print test_pred.shape
    # print test_labels.shape
    #evaluate the error
    train_error=1-accuracy_score(train_labels, train_pred)
    test_error=1-accuracy_score(test_labels, test_pred)
    print('iteration number: ',steps_, 'train error: ', train_error, 'test error: ',
test_error, 'likelyhood: ',likehood)
    #recording
    train_err_list.append(train_error)
    test_err_list.append(test_error)

plt.plot(likehood_xlabel, train_err_list,label='training error')
plt.plot(likehood_xlabel, test_err_list,label='test error')
#plt.plot(likehood_xlabel, likehood_list,label='likeihood')
plt.xlabel('iteration steps')
plt.ylabel('Error rate')
plt.legend(loc=1)
plt.show()
plt.plot(likehood_xlabel, likehood_list,label='likeihood')
plt.xlabel('iteration steps')
plt.ylabel('likeihood')
plt.legend(loc=1)
plt.show()
```

B)

1) Re-apply the method to madelon data set:

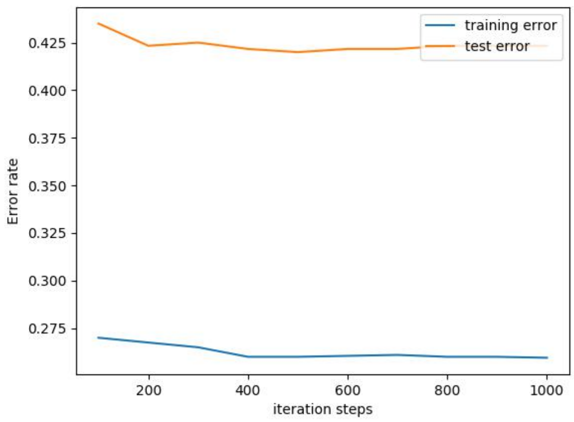
We get the log-likelihood vs iteration:



The test error and training error are shown as following:

iteration number	train error	test error
100	0.27	0.435
200	0.2675	0.423333333333
300	0.265	0.425
400	0.26	0.421666666667
500	0.26	0.42
600	0.2605	0.421666666667
700	0.261	0.421666666667
800	0.26	0.423333333333
90	0.26	0.423333333333
1000	0.2595	0.423333333333

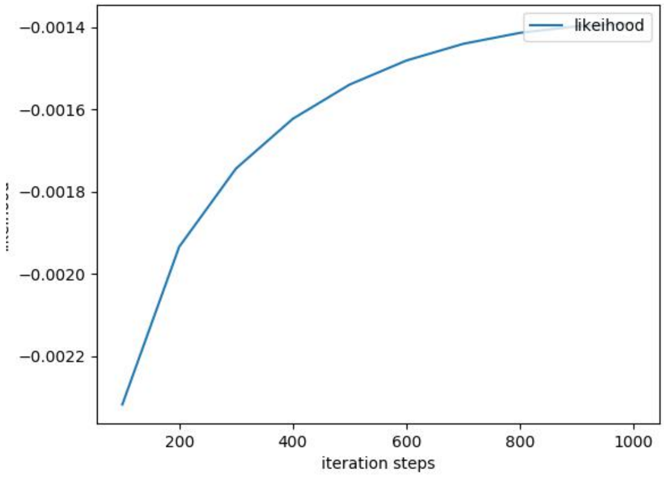
Which has been plotted below:



2)

We then reapply the method to arcene data set.

We get the log-likelihood vs iteration:



The test error and training error are shown as following:

iteration number	train error	test error
100	0	0
200	0	0
300	0	0
400	0	0
500	0	0
600	0	0
700	0	0
800	0	0
90	0	0
1000	0	0

Which has been plotted below: