4) **Implementation of 2 class Logistic Regression in Python**

Dataset used: UCI ML Repository, Breast Cancer Wisconsin

Weight Vector estimated from the training set for the 30 real valued attributes

[-18.81744849 2.20771913 -14.68141516 3.43156113 -11.48154784
0.4142682 21.9230974 28.39086015 -4.97384441 -17.96908208
17.29605201 -1.97261204 12.77345807 15.14686682 -5.58377147
-9.36253881 -9.43569909 -5.58015815 -6.10853381 -6.89736002
2.80425196 4.59676559 3.59617615 15.09156915 3.94418386
0.86594514 -3.96872162 4.67076254 7.74522569 7.93982303]

Accuracy on the test instances: 94%

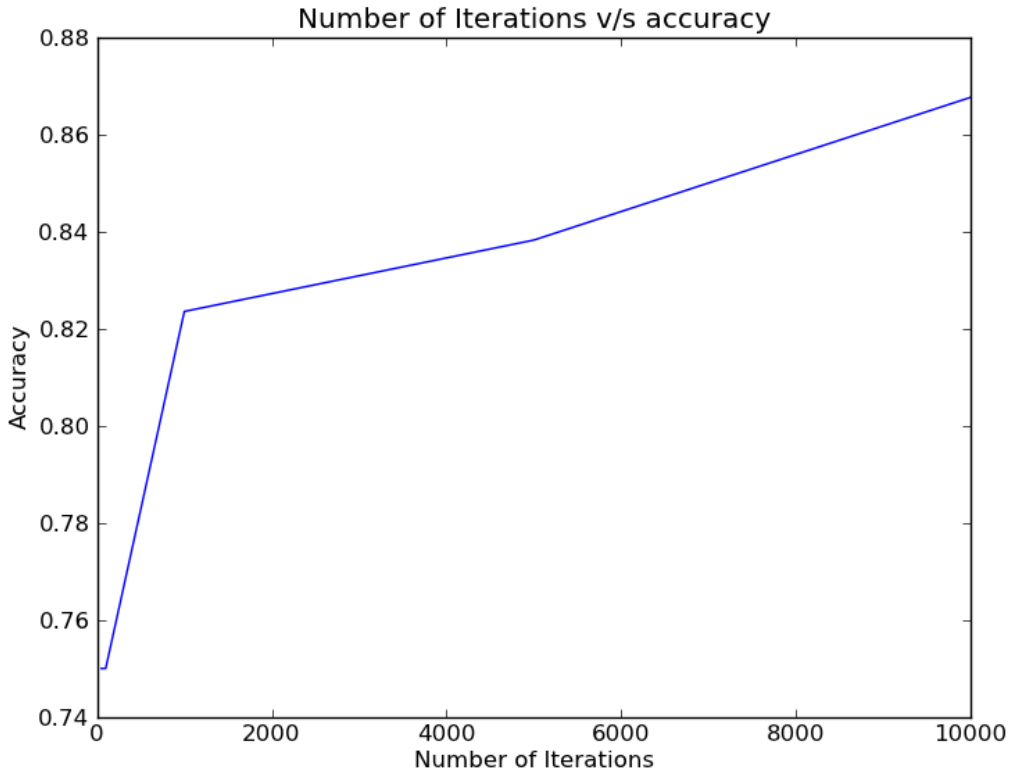
```
"""
@Author: Primal Pappachan
@email: primal1@umbc.edu
2 class Logistic Regression
Dataset used: Breast Cancer Wisconsin
http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagn
ostic)
Dataset Information:
      Number of instances: 569
      Number of attributes: 32 (ID, diagnosis, 30 real-valued input
features)
File used: bc.csv - Contains Dataset with ID removed and diagnosis =
{0, 1}
Accuracy: 90%(averaged)
Required Python Libraries:
      pandas - for preprocessing data using data frames
      numpy - for arrays
      pylab - for plotting graphs
"""

import pandas as pd
import numpy as np
import pylab as pl

def sigmoid(x):
      return 1.0 / (1.0 + np.exp(-x))
```

```
def dot_product(x, w):
     return sum(a * b for a, b in zip(x, w))

def shuffle(df):
     #Randomizes the dataframe
     #Reference:
http://stackoverflow.com/questions/13395725/efficient-way-of-doing-pe
rmutations-with-pandas-over-a-large-dataframe
     ind = df.index
     sampler = np.random.permutation(df.shape[0])
     new_vals = df.take(sampler).values
     df = pd.DataFrame(new_vals, index=ind)
     return df

def train(data, theta, vlen, tlen, limit=50):
     w = np.zeros(vlen) #Weight Vector
     count = 0
     while(count<limit):
          gvector = np.zeros(vlen)
          for row in data.values[:tlen]:
               x = row[:vlen]
               label = row[vlen]
               prob = sigmoid(dot_product(x, w))
               error = label - prob
               for j in xrange(vlen):
                    gvector[j] = gvector[j] + error * x[j]
          w = w + theta * gvector
          count = count + 1
     return w

def test(data, w, vlen, tlen):
     p_ones = c_ones = c_zeros = 0
     for row in data.values[tlen:]:
          #pdb.set_trace()
          x = row[:vlen]
          if row[vlen] == 1:
               c_ones = c_ones + 1
               if dot_product(x, w) >= 0:
                    p_ones = p_ones + 1
          else: c_zeros = c_zeros + 1
     return p_ones, c_ones
```

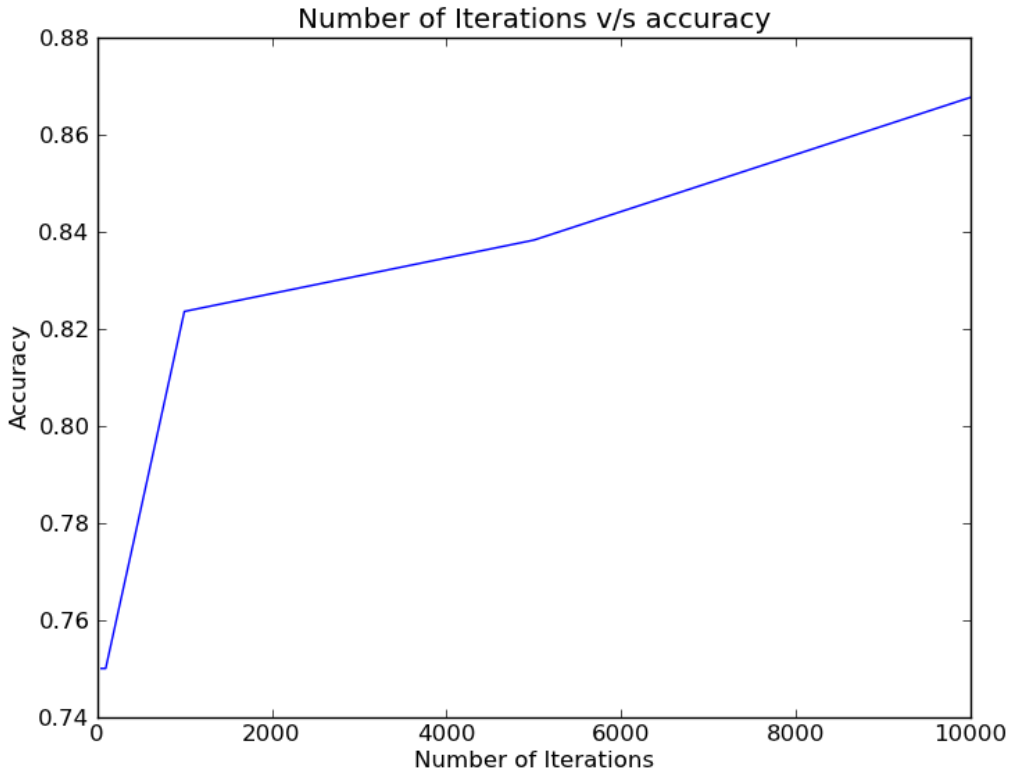
```
def main(theta=0.001):
      df = pd.read_csv('bc.csv', na_values=['?'])
      attrs = []
      for i in xrange(len(df.values[1])-1):
            attrs.append('a_'+str(i-1))
      attrs.append("label")

      df.columns = attrs   #Initializing the attribute names

      #Normalizing the data frame
      data = (df - df.min()) / (df.max() - df.min())
      #Randomizing the values
      data = shuffle(data)

      #Preprocessing of data done

      vlen = df.shape[1] - 1 #Number of attributes
      N = df.shape[0] #Number of training samples
      tlen = int(N * 2/3.0)
      accuracy = []
      limits = [50, 100, 1000, 5000, 10000]
      for limit in limits:
            w = train(data, theta, vlen, tlen,limit)
            p, c = test(data, w, vlen, tlen)
            accuracy.append(p/float(c))

      pl.plot(limits, accuracy)
      pl.title("Number of Iterations v/s accuracy")
      pl.ylabel("Accuracy")
      pl.xlabel("Number of Iterations")
      pl.show()

if __name__ == "__main__":
      main()
```

Accuracy vs Number of Iterations graph

[50, 100, 1000, 5000, 10000]
[0.75, 0.75, 0.8235294117647058, 0.8382352941176471, 0.8676470588235294]



[50, 100, 1000, 5000, 10000]
[0.86, 0.88, 0.91234235234, 0.93235423523, 0.9423423532523, 0.9423423532523]


Number of Iterations v/s accuracy
Accuracy
Number of Iterations
0.88
0.86
0.84
0.82
0.80
0.78
0.76
0.74
0
2000
4000
6000
8000
10000