```
import numpy as np


# this matrix will store the data
labels = np.array (1)

# and this vector will store the labels
points = np.array (1)

# open up the input text file
with open('bc.txt') as f:
     #
     # read in the lines and init the data and labels
     lines = f.readlines ()
     labels = np.zeros (len (lines))
     points = np.zeros ((len (lines), 30))
     counter = 0
     #
     # loop through each of the lines
     for line in lines:
          #
          # get all of the items on the line
          array = [x for x in line.split (',')]
          #
          # get the data point
          for index in range (2, 32):
               points[counter,index - 2] = float (array[index])
          #
          # if cancerous, 1, else -1
          if (array[1] == 'M'):
               labels [counter] = 1
          else:
               labels [counter] = -1
          counter = counter + 1


# evaluates the loss function and returns the loss
#
# x is the data set
# y is the labels
# w is the current set of weights
# c is the weight of the slack variables
#
def f (x, y, w, c):
     loss = 0
     n = len(x)
     var = 1 / (n * c)
     # fill in missing code here!!
     for i in range(n):
        temp = 1 - y[i]*np.dot(w, x[i])
        loss = loss + 1/n * max([0, temp])

     loss = loss + var/2 * np.dot(w,w)

     return loss
```

```
# evaluates and returns the gradient
#
# x is the data set
# y is the labels
# w is the current set of weights
# c is the weight of the slack variables
#
def gradient(x, y, w, c):
     #
     # Note that the gradient has 30 dims because the data has 30 dims
     gradient = np.zeros (30)
     # fill in missing code here!!
     n = len(x)
     var = 1 / (n ∗ c)
     for i in range(n):
        temp = 1 − y[i]∗(w@x[i])
        gradient = gradient + 1/n∗(0 if temp<0 else −y[i]∗x[i])
     gradient = gradient + var∗w


     return gradient



# make predictions using all of the data points in x
# print ‘success’ or ‘failure’ depending on whether the
# prediction is correct
#
# x is the data set
# y is the labels
# w is the current set of weights
#
def predict (x, y, w):
     correct = 0;
     tp = 0
     tn = 0
     for index in range (len (y)):
          if ((np.dot (x[index], w) > 0) and (y[index] > 0)):
               print ('success')
               correct = correct + 1
               tp = tp + 1
          elif ((np.dot (x[index], w) < 0) and (y[index] < 0)):
               print ('success')
               correct = correct + 1
               tn = tn + 1
          else:
               print ('failure')
     pos_tot = np.sum(y>0, axis=0)
     neg_tot = np.sum(y<0, axis=0)
     fp = neg_tot − tn
     fn = pos_tot − tp
     precision = tp / (tp + fp)
     recall = tp / pos_tot
     f1 = 2 ∗ precision ∗ recall / (precision + recall)
     print ('%d out of %d correct.' % (correct, len(y)))
     print ('precision: %f, recall: %f, F1 score: %f' % (precision, recall, f1))
     return f1
```

```
# performs gradient descent optimization, returns the learned set of weights
# uses the bold driver to set the learning rate
#
# x is the data set
# y is the labels
# w is the current set of weights  to start with
# c is the weight of the slack variable
#
def gd_optimize (x, y, w, c):
    rate = 1
    w_last = w + np.full (30, 1.0)
    while (abs(f (x, y, w, c) − f (x, y, w_last, c)) > 10e−4):
         w_last = w
         w = w − rate ∗ gradient (x, y, w, c)
         if f (x, y, w, c) > f (x, y, w_last, c):
              rate = rate ∗ .5
         else:
              rate = rate ∗ 1.1
         print (f (x, y, w, c))
    return w



w = np.zeros (30)

w = gd_optimize (points[0:400], labels[0:400], w, .1)

predict (points[400:], labels[400:], w)
```

```
success
success
success
success
success
success
success
failure
success
success
success
success
success
success
151 out of 169 correct.
precision: 0.714286, recall: 0.897436, F1 score: 0.795455
0.7954545454545455
```

```
#best c
c_arr = [0.1, 0.01, 0.05, 0.5, 0.001]
f1_arr = []
for c_val in c_arr:
  wt = np.zeros (30)
  wt = gd_optimize (points[0:400], labels[0:400], wt, c_val)
  f1 = predict (points[400:], labels[400:], wt)
  f1_arr.append(f1)
```

```
success
success
success
success
success
success
success
success
success
success
failure
success
success
success
success
success
success
151 out of 169 correct.
precision: 0.723404, recall: 0.871795, F1 score: 0.790698
```

```
print('The best c is ', c_arr[np.argmax(f1_arr)])
```

```
The best c is  0.05
```

```
f1_arr
```

```
[0.7954545454545455,
 0.8095238095238095,
 0.8292682926829267,
 0.7906976744186047,
 0.7906976744186047]
```