403. 1 $\nabla g_2(\tilde{w}) = -\sum_{p=1}^{P} \sigma(-y_p \tilde{x}_p^T \tilde{w})\, y_p \tilde{x}_p$

Softmax cost function

$$g_2(\tilde{w}) = \sum_{p=1}^{P} \log(1+e^{-y_p \tilde{x}_p^T \tilde{w}})$$

Taking gradient

$$\nabla g_2(\tilde{w}) = \nabla \sum_{p=1}^{P} \log(1+e^{-\tilde{y}_p x_p^T \tilde{w}})$$

$$= \sum_{p=1}^{P} \frac{1}{1+e^{-\tilde{y}_p x_p^T \tilde{w}}}\, e^{-y_p \tilde{x}_p^T \tilde{w}} \; -y_p \tilde{x}_p^T$$

$$= \sum_{p=1}^{P} \cancel{-y_p \tilde{x}_p^T} \; \frac{-1}{(1+e^{y_p \tilde{x}_p^T \tilde{w}})}\; y_p \tilde{x}_p^T$$

$\therefore \frac{1}{1+e^{t}} = \sigma(-$

$$= \sum_{p=1}^{P} -\cancel{y_p \tilde{x}_p^T}\; \sigma(-y_p x_p^T \tilde{w})\; y_p \tilde{x}_p^T$$

5.1 $g_2(b,w) = \sum_{p=1}^{P} \log\left(1 + e^{-y_p(b+x_p^T w)}\right)$

$$g_2(cb, cw) = \sum_{p=1}^{P} \log\left(1 + e^{-y_p(cb + x_p^T cw)}\right)$$

$$= \sum_{p=1}^{P} \log\left(1 + e^{-y_p c(b + x_p^T w)}\right)$$

$$e^{-y_p c(b + x_p^T w)} < e^{-y_p(b + x_p^T w)}$$

$$1 + e^{-y_p c(b + x_p^T w)} < 1 + e^{-y_p(b + x_p^T w)}$$

$$\sum_{p=1}^{P} \log\left(1 + e^{-y_p c(b + x_p^T w)}\right) < \sum_{p=1}^{P} \log\left(1 + e^{-y_p(b + x_p^T w)}\right)$$

$$b + x_p^T w = 0$$

multiply by c

$$cb + x_p^T wc = 0$$

$$c(b + x_p^T w) = 0$$

$$b + x_p^T w = 0$$

5.2. If minimizing the softmax cost over a linearly seperable dataset causes parameters to grow too large, this effect is similar to having large 'c'

$e^{-y_p(b+x_p^T w)c}$ → This value becomes too small

Thus, making cost function smaller and thus it is difficult to calculate gradient descent as the step size is too small, this takes large iterations to converge

12.

cost function for logistic regression

$$h(b,w) = -\sum_{p=1}^{P} \bar{y}_p \log \sigma(b+x_p^T w) + (1-\bar{y}_p) \log(1-\sigma(b+x_p^T w))$$

Substituting $y_p = 1$

$$= -\sum_{p=1}^{P} \cancel{y_p} \log \sigma(b+x_p^T w)$$

$$= -\sum_{p=1}^{P} \log \frac{1}{1+e^{-(b+x_p^T w)}} \qquad \sigma(t) = \frac{1}{1+e^{-t}}$$

$$= -\sum_{p=1}^{P} \log 1 - \log(1+e^{-(b+x_p^T w)})$$

$$= \sum_{p=1}^{P} \log(1+e^{-(b+x_p^T w)})$$

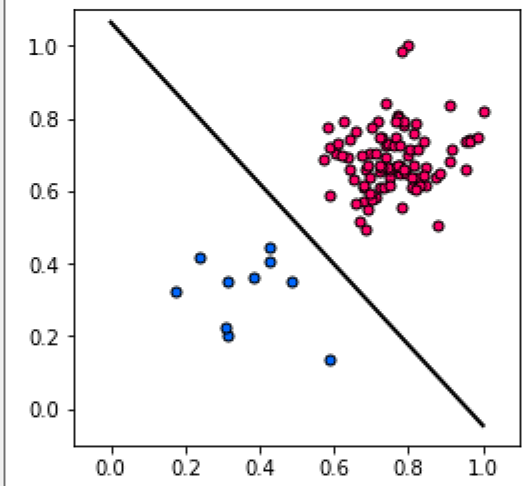
$$\Downarrow$$

$g(b,w)$ = soft max cost function

Scanned by CamScanner

```
#4.3 c
# This file is associated with the book
# "Machine Learning Refined", Cambridge University Press, 2016.
# by Jeremy Watt, Reza Borhani, and Aggelos Katsaggelos.

import numpy as np
import matplotlib.pyplot as plt
import csv

# sigmoid for softmax/logistic regression minimization


# import training data
def load_data(csvname):
   # load in data
   reader = csv.reader(open(csvname, "r"), delimiter=",")
   d = list(reader)

   # import data and reshape appropriately
   data = np.array(d).astype("float")
   X = data[:,0:2]
   y = data[:,2]
   y.shape = (len(y),1)

   # pad data with ones for more compact gradient computation
   o = np.ones((np.shape(X)[0],1))
   X = np.concatenate((o,X),axis = 1)
   X = X.T

   return X,y
def sigmoid(z):
   y = 1/(1+np.exp(-z))
   return y
# YOUR CODE GOES HERE - create a gradient descent function for softmax cost/logistic regression
def softmax_grad(X,y):
   w = np.random.randn(3,1)
   grad = 1
   iter = 1
   max_its = 100
   alpha= 10**-2
   while np.linalg.norm(grad) >= 10**(-2) and iter < max_its:
      r= - sigmoid(-y*np.dot(X.T,w))*y
      grad = np.dot(X,r)
      w -=alpha*grad
```

```
    return w


# plots everything
def plot_all(X,y,w):

    # custom colors for plotting points
    red = [1,0,0.4]
    blue = [0,0.4,1]

    # scatter plot points
    fig = plt.figure(figsize = (4,4))
    ind = np.argwhere(y==1)
    ind = [s[0] for s in ind]
    plt.scatter(X[1,ind],X[2,ind],color = red,edgecolor = 'k',s = 25)
    ind = np.argwhere(y==-1)
    ind = [s[0] for s in ind]
    plt.scatter(X[1,ind],X[2,ind],color = blue,edgecolor = 'k',s = 25)
    plt.grid('off')
    # plot separator
    s = np.linspace(0,1,100)

    plt.plot(s,(-w[0]-w[1]*s)/w[2],color = 'k',linewidth = 2)

    # clean up plot
    plt.xlim([-0.1,1.1])
    plt.ylim([-0.1,1.1])
    plt.show()
# load in data
X,y = load_data('imbalanced_2class.csv')

# run gradient descent
w = softmax_grad(X,y)
print(w)
# plot points and separator
plot_all(X,y,w)
```



In [3]:

4. 9

```
# This file is associated with the book
# "Machine Learning Refined", Cambridge University Press, 2016.
# by Jeremy Watt, Reza Borhani, and Aggelos Katsaggelos.

import numpy as np
import matplotlib.pyplot as plt
import csv
import pandas as pd
import math
# import training data
def load_data(csvname):
    # load in data
    data = np.asarray(pd.read_csv(csvname))

    # import data and reshape appropriately
    X = data[:,0:-1]
    y = data[:,-1]
    y.shape = (len(y),1)

    # pad data with ones for more compact gradient computation
    o = np.ones((np.shape(X)[0],1))
    X = np.concatenate((o,X),axis = 1)
    X = X.T
```

```
    return X,y

### TODO: YOUR CODE GOES HERE ###
# run newton's method
def squared_margin_newtons_method(X,y,w):
    # begin newton's method loop
    max_its=20
    misclass_history=[]
    #w = 0.00* np.random.randn(9,1)
    w=0.01*np.ones((9,1))
    hess=np.zeros((9,9))
    for k in range(max_its):
        r= (1-y*np.dot(X.T,w))*y
        grad = -2* np.dot(X,r)
        hess=2*np.dot(X,X.T)
        w = w -  grad*np.linalg.pinv(hess)
        misclass_history.append(count_misclasses(X,y,w))


    return misclass_history
def count_misclasses(X,y,w):
    y_pred = np.sign(-y*(np.dot(X.T,w)))
    y_pred1=np.array(y_pred)
    i=[]
    num_misclassed=0

    for i in range(0,len(y_pred1)):
        for j in range(0,8):
            if y_pred[i][j] > 0 :
                num_misclassed=num_misclassed+1
    return num_misclassed

def sigmoid(z):
    y = 1/(1+my_exp(-z))
    return y

def my_exp(u):
    s = np.argwhere(u > 100)
    t = np.argwhere(u < -100)
    u[s] = 0
    u[t] = 0
    u = np.exp(u)
    u[t] = 1
    return u
```

```
# run newton's method
def softmax_newtons_method(X,y,w):
    # begin newton's method loop
    max_its = 20
    misclass_history = []
    w = 0.01*np.random.randn(9,1)
    for k in range(max_its):
        r= -y* sigmoid(-y*np.dot(X.T,w))
        grad = np.dot(X,r)
        hess=np.zeros((9,9))

        for i in range(0,len(X.T)):
            XX=X[:,i]
            r1= y[i] * sigmoid(-np.dot(XX,w))
            r2= 1 - sigmoid(-y[i]*np.dot(XX,w))
            hess1 = r1 * r2
            XX.shape = (9,1)
            d= np.dot(XX,XX.T)
            hess= d* hess1

        w= w -  grad*np.linalg.pinv(hess)

        misclass_history.append(count_misclasses(X,y,w))
    return misclass_history


###### run functions above #######
# load data
X,y = load_data('breast_cancer_data.csv')
# run newtons method to minimize squared margin or SVM cost
w = np.zeros((np.shape(X)[0],1))
squared_margin_history = squared_margin_newtons_method(X,y,w)
#print(squared_margin_history)
# run newtons method to minimize logistic regression or softmax cost
w = np.zeros((np.shape(X)[0],1))
softmax_cost_history = softmax_newtons_method(X,y,w)
# plot results
plt.plot(squared_margin_history)
plt.plot(softmax_cost_history)

# clean up plot
#plt.ylim([min(min(squared_margin_history),min(softmax_cost_history)) -
1,max(max(squared_margin_history),max(softmax_cost_history)) + 1])
plt.xlabel('iterations ')
```

```
plt.ylabel(' misclassifications')
plt.legend([,'softmax', 'square margin'])
plt.show()
```