**The binary classification**

**from** pylab **import** \*  
**import** matplotlib.lines **as** mlines  
**import** matplotlib.pyplot **as** plt  
  
*# load the data***def** load\_data():  
 data = matrix(genfromtxt(**'spambase\_data.csv'**, delimiter=**','**))  
 X = asarray(data[:,0:-1])  
 X = X.T  
 y = asarray(data[:,-1])  
 y.shape = (len(y),1)  
 **return** (X,y)  
  
**def** rescale(X, unchanged\_len):  
 unchanged\_features = X[:unchanged\_len,:]  
 changed\_features = log(X[unchanged\_len:,:])  
 scaled\_X = concatenate((unchanged\_features, changed\_features))  
 **return** scaled\_X  
  
**def** normalize(X):  
 **return** X.dot((diag(1/norm(X,2,axis=0))))  
  
**def** cal\_misclass(X,y,w):  
 mis = ((1-w.T.dot(X).dot(diag(y.T[0])))>0).dot(ones((len(X[0]),1)))  
 **return** mis[0,0]  
  
**def** squared\_margin\_gradient(X,y):  
 *# use compact notation and initialize* X = concatenate((ones((1,len(X[0]))),X))  
 *# w = zeros((len(X),1))* w = randn(len(X),1)  
 t = 2  
 alpha = t/(pow(norm(X,2),2)\*2)  
 *# start newton's method loop* k = 1  
 max\_its = 1000  
 grad = 1  
 mis = []  
 X\_diagy = X.dot(diag(y.T[0]))  
 **while** norm(grad) > 10\*\*(-5) **and** k <= max\_its:  
 *# print "calculating ...", k, "/", max\_its  
 # compute gradient* grad = -2\*X\_diagy.dot(maximum(0,1-w.T.dot(X\_diagy).T))  
 *# take next step* w = w - alpha\*grad  
 *# record the number of misclassifications* mis.append(cal\_misclass(X, y, w))  
 k += 1  
 **return** (w, mis)  
  
**def** drawPlots(squared\_mis1, squared\_mis2, squared\_mis3):  
 plt.plot(range(1,len(squared\_mis1)+1), squared\_mis1)  
 plt.plot(range(1,len(squared\_mis2)+1), squared\_mis2)  
 plt.plot(range(1,len(squared\_mis3)+1), squared\_mis3)  
 plt.xlabel(**'iteration'**)  
 plt.ylabel(**'number of misclassifications'**)  
 plt.show()  
  
*### main loop ###***def** main():  
 *# load data* BoW\_len = 48  
 CharFre\_len = 6  
 Spam\_len = 3  
 X,y = load\_data()  
 X = rescale(X, BoW\_len+CharFre\_len+1)  
 X = normalize(X)  
 X1 = copy(X[:BoW\_len,:])  
 X2 = copy(X[:BoW\_len+CharFre\_len,:])  
 X3 = copy(X)  
  
 *# run gradient descent* print (**"calculating for BoW features ..."**)  
 squared\_w1, squared\_mis1 = squared\_margin\_gradient(X1, y)  
 print (**"calculating for BoW+charfreqs features ..."**)  
 squared\_w2, squared\_mis2 = squared\_margin\_gradient(X2, y)  
 print (**"calculating for BoW+charfreqs+spam features ..."**)  
 squared\_w3, squared\_mis3 = squared\_margin\_gradient(X3, y)  
 blue\_line = mlines.Line2D([], [], color=**'blue'**,  
 markersize=15, label=**'BoW'**)  
 green\_line = mlines.Line2D([], [], color=**'green'**,  
 markersize=15, label=**'BoW + char(frequency)'**)  
 red\_line = mlines.Line2D([], [], color=**'red'**,  
 markersize=15, label=**'BoW + char(frequency)+spam\_features'**)  
 plt.legend(handles=[blue\_line,green\_line,red\_line])  
  
 drawPlots(squared\_mis1, squared\_mis2, squared\_mis3)  
  
main()