Tasks – Regression (week 1)

1. How many reviews are there for each style of beer in the dataset (‘beer/style’)? What is the average value of ‘review/taste’ for reviews from each style? (1 mark)

Answer:

a) The number of reviews for each style of beer in the dataset is shown as follows:
{'Hefeweizen': 618, 'English Strong Ale': 164, 'Foreign / Export Stout': 55, 'German Pilsener': 586, 'American Double / Imperial IPA': 3886, 'Herbed / Spiced Beer': 73, 'Oatmeal Stout': 102, 'American Pale Lager': 123, 'Rauchbier': 1938, 'American Pale Ale (APA)': 2288, 'American Porter': 2230, 'Belgian Strong Dark Ale': 146, 'Russian Imperial Stout': 2695, 'American Amber / Red Ale': 665, 'American Strong Ale': 166, 'MÃ¤rzen / Oktoberfest': 557, 'American Adjunct Lager': 242, 'American Blonde Ale': 357, 'American IPA': 4113, 'Fruit / Vegetable Beer': 1355, 'English Bitter': 267, 'English Porter': 367, 'Irish Dry Stout': 101, 'American Barleywine': 825, 'American Double / Imperial Stout': 5964, 'Doppelbock': 873, 'American Stout': 591, 'Maibock / Helles Bock': 225, 'Dortmunder / Export Lager': 31, 'Euro Strong Lager': 329, 'Low Alcohol Beer': 7, 'Light Lager': 503, 'Euro Pale Lager': 701, 'Bock': 148, 'English India Pale Ale (IPA)': 175, 'Altbier': 165, 'KÃ¶lsch': 94, 'Pumpkin Ale': 560, 'Rye Beer': 1798, 'American Pale Wheat Ale': 154, 'Milk / Sweet Stout': 69, 'Schwarzbier': 53, 'Munich Dunkel Lager': 141, 'Vienna Lager': 33, 'American Amber / Red Lager': 42, 'Scottish Ale': 78, 'Witbier': 162, 'Saison / Farmhouse Ale': 141, 'American Black Ale': 138, 'English Brown Ale': 495, 'English Barleywine': 133, 'Extra Special / Strong Bitter (ESB)': 667, 'California Common / Steam Beer': 11, 'Euro Dark Lager': 144, 'Scotch Ale / Wee Heavy': 2776, 'English Pale Ale': 1324, 'Belgian Strong Pale Ale': 632, 'Belgian Pale Ale': 144, 'Tripel': 257, 'Flanders Oud Bruin': 13, 'American Brown Ale': 314, 'Smoked Beer': 61, 'Dunkelweizen': 61, 'Dubbel': 165, 'Keller Bier / Zwickel Bier': 23, 'Winter Warmer': 259, 'BiÃ¨re de Garde': 7, 'Belgian Dark Ale': 175, 'Irish Red Ale': 83, 'Chile Beer': 11, 'English Stout': 136, 'Czech Pilsener': 1501, 'Belgian IPA': 128, 'Black & Tan': 122, 'Cream Ale': 69, 'English Dark Mild Ale': 21, 'American Wild Ale': 98, 'Weizenbock': 13, 'American Double / Imperial Pilsner': 14, 'Scottish Gruit / Ancient Herbed Ale': 65, 'Wheatwine': 455, 'American Dark Wheat Ale': 14, 'American Malt Liquor': 90, 'Munich Helles Lager': 650, 'Kristalweizen': 7, 'English Pale Mild Ale': 21, 'Baltic Porter': 514, 'Old Ale': 1052, 'Quadrupel (Quad)': 119, 'Braggot': 26, 'Lambic - Fruit': 6, 'Lambic - Unblended': 10, 'Eisbock': 8, 'Flanders Red Ale': 2, 'Berliner Weissbier': 10}

b) The average value of reviews for each style of beer is shown below:
{'Hefeweizen': 3.635113268608414, 'English Strong Ale': 3.7560975609756095, 'Foreign / Export Stout': 3.2545454545454544, 'German Pilsener': 3.667235494880546, 'American Double / Imperial IPA': 4.033324755532681, 'Herbed / Spiced Beer': 3.4452054794520546, 'Oatmeal Stout': 3.7745098039215685, 'American Pale Lager': 3.2154471544715446, 'Rauchbier': 4.067853457172343, 'American Pale Ale (APA)': 3.649694055944056, 'American Porter': 4.081838565022421, 'Belgian Strong Dark Ale': 3.6952054794520546, 'Russian Imperial Stout': 4.300371057513915, 'American

Amber / Red Ale': 3.513533834586466, 'American Strong Ale': 3.569277108433735, 'MÃ¤rzen / Oktoberfest': 3.5933572710951527, 'American Adjunct Lager': 2.9483471074380163, 'American Blonde Ale': 3.2549019607843137, 'American IPA': 4.00085096036956, 'Fruit / Vegetable Beer': 3.607749077490775, 'English Bitter': 3.5374531835205993, 'English Porter': 3.70708446866485, 'Irish Dry Stout': 3.623762376237624, 'American Barleywine': 4.064242424242424, 'American Double / Imperial Stout': 4.479963112005366, 'Doppelbock': 3.9828178694158076, 'American Stout': 3.8197969543147208, 'Maibock / Helles Bock': 3.7466666666666666, 'Dortmunder / Export Lager': 3.4193548387096775, 'Euro Strong Lager': 2.8480243161094223, 'Low Alcohol Beer': 2.7142857142857144, 'Light Lager': 2.39662027833002, 'Euro Pale Lager': 2.962910128388017, 'Bock': 3.189189189189189, 'English India Pale Ale (IPA)': 3.4714285714285715, 'Altbier': 3.403030303030303, 'KÃ¶lsch': 3.6968085106382977, 'Pumpkin Ale': 3.7875, 'Rye Beer': 4.213570634037819, 'American Pale Wheat Ale': 3.3344155844155843, 'Milk / Sweet Stout': 3.782608695652174, 'Schwarzbier': 3.6226415094339623, 'Munich Dunkel Lager': 3.780141843971631, 'Vienna Lager': 3.5303030303030303, 'American Amber / Red Lager': 3.6904761904761907, 'Scottish Ale': 3.7628205128205128, 'Witbier': 3.5277777777777777, 'Saison / Farmhouse Ale': 3.702127659574468, 'American Black Ale': 3.8731884057971016, 'English Brown Ale': 3.728282828282828, 'English Barleywine': 4.360902255639098, 'Extra Special / Strong Bitter (ESB)': 3.685157421289355, 'California Common / Steam Beer': 3.3181818181818183, 'Euro Dark Lager': 3.704861111111111, 'Scotch Ale / Wee Heavy': 4.083393371757925, 'English Pale Ale': 3.483761329305136, 'Belgian Strong Pale Ale': 4.056170886075949, 'Belgian Pale Ale': 3.7395833333333335, 'Tripel': 3.7840466926070038, 'Flanders Oud Bruin': 3.923076923076923, 'American Brown Ale': 3.7436305732484074, 'Smoked Beer': 3.19672131147541, 'Dunkelweizen': 3.4918032786885247, 'Dubbel': 3.7363636363636363, 'Keller Bier / Zwickel Bier': 3.869565217391304, 'Winter Warmer': 3.6216216216216215, 'BiÃ¨re de Garde': 3.9285714285714284, 'Belgian Dark Ale': 3.34, 'Irish Red Ale': 2.9819277108433737, 'Chile Beer': 3.9545454545454546, 'English Stout': 3.599264705882353, 'Czech Pilsener': 3.609593604263824, 'Belgian IPA': 3.94921875, 'Black & Tan': 3.942622950819672, 'Cream Ale': 3.028985507246377, 'English Dark Mild Ale': 3.7857142857142856, 'American Wild Ale': 4.188775510204081, 'Weizenbock': 3.3846153846153846, 'American Double / Imperial Pilsner': 3.8214285714285716, 'Scottish Gruit / Ancient Herbed Ale': 3.9076923076923076, 'Wheatwine': 4.186813186813187, 'American Dark Wheat Ale': 3.6785714285714284, 'American Malt Liquor': 2.2555555555555555, 'Munich Helles Lager': 3.959230769230769, 'Kristalweizen': 2.7857142857142856, 'English Pale Mild Ale': 3.5952380952380953, 'Baltic Porter': 4.213035019455253, 'Old Ale': 4.096007604562738, 'Quadrupel (Quad)': 3.596638655462185, 'Braggot': 3.8076923076923075, 'Lambic - Fruit': 3.75, 'Lambic - Unblended': 3.3, 'Eisbock': 3.75, 'Flanders Red Ale': 3.25, 'Berliner Weissbier': 3.55}

c) The code for this problem is the following:

```
import numpy
import urllib
from urllib import request

def parseData(fname):
   for l in urllib.request.urlopen(fname):
      yield eval(l)

data = list(parseData("http://jmcauley.ucsd.edu/cse255/data/beer/beer_50000.json"))

m = {} #dictionary to store numbers of reviews for each style of beer
average = {} #dictionary to store average values of reviews
for d in data:
      if d['beer/style'] in m:
            m[d['beer/style']] += 1
            average[d['beer/style']] += d['review/taste']
      else:
            m[d['beer/style']] = 1
            average[d['beer/style']] = d['review/taste']

for style in average:
      average[style] = average[style]/m[style]

print (m)
print (average)
```

2. Train a simple predictor with a single binary feature indicating whether a beer is an 'American IPA':
   'review/taste' $\cong \theta_0 + \theta_1 \times$ [beer is an American IPA]
   Report the values of θ_0 and θ_1. Briey describe your interpretation of these values, i.e., what do θ_0 and θ_1 represent (1 mark)?

Answer:

a) The values of θ_0 and θ_1 are:

$$\theta_0 = 3.91520474$$
$$\theta_1 = 0.08564622$$

Since 'review/taste' $\cong \theta_0 + \theta_1 \times$ [beer is an American IPA],
Then we can rewrite this equation as follows:

$$'\text{review/taste}' = \begin{cases} \theta_0 & beer\ is\ \ NOT\ an\ American\ IPA \\ \theta_0 + \theta_1 & beer\ is\ an\ American\ IPA \end{cases}$$

Hence, we can interpret θ_0 and θ_1 as:
θ_0 represents the average review of taste for all beers except 'American IPA'; θ_1 represents the difference between a review/taste for an American IPA beer and the one for other beers. If $\theta_1 > 0$, then 'American IPA' has higher rate on taste than other beers; if $\theta_1 < 0$, then 'American IPA' has lower rate on taste than other beers.

b) The code for this problem is as follows:

```
import numpy
import urllib
from urllib import request

def parseData(fname):
   for l in urllib.request.urlopen(fname):
      yield eval(l)

print ("Reading data...")
data = list(parseData("http://jmcauley.ucsd.edu/cse255/data/beer/beer_50000.json"))
print ("done")
X = []
for d in data:
      if d['beer/style'] == 'American IPA':
            X.append([1, 1])
      else:
            X.append([1, 0])

y = [d['review/taste'] for d in data]
theta,residuals,rank,s = numpy.linalg.lstsq(X, y)
print (theta)
```

3. Split the data into two equal fractions – the first half for training, the second half for testing (based on the order they appear in the file). Train the same model as above on the training set only. What is the model's MSE on the training and on the test set (1 mark)?

Answer:

a) The model's MSE on the training set is 0.558107286559; the model's MSE on the testing set is 0.468410050967.

b) The code for this problem is as shown below:

```
import numpy
import urllib
import scipy.optimize
import random
from urllib import request

def parseData(fname):
  for l in urllib.request.urlopen(fname):
    yield eval(l)

print ("Reading data...")
data = list(parseData("http://jmcauley.ucsd.edu/cse255/data/beer/beer_50000.json"))
print ("done")

X = []
for d in data:
    if d['beer/style'] == 'American IPA':
        X.append([1, 1])
    else:
        X.append([1, 0])

y = [d['review/taste'] for d in data]
X1 = X[:int(len(X)/2)]
X2 = X[int(len(X)/2):]
y1 = y[:int(len(y)/2)]
y2 = y[int(len(y)/2):]
theta,residuals,rank,s = numpy.linalg.lstsq(X1, y1)
print ('MSE of training part is: ', residuals[0]/len(y1))

X2 = numpy.matrix(X2)
y2 = numpy.matrix(y2)
A = numpy.dot(theta, X2.T)
mse = numpy.square(A - y2).mean()
print ('MSE of testing part is: ', mse)
```

4. Extend the model above so that it incorporates binary features for every style of beer with ≥ 50 reviews (or every style except one, depending on your representation). Report the values of θ that you obtain, and the model's MSE on the training and on the test set (1 mark).

Answer:

a) There are 74 beers which have more than 50 reviews. They are (the number coming after the 'beer/style' is their index number in theta vector):

{'Hefeweizen': 1, 'English Strong Ale': 2, 'Foreign / Export Stout': 3, 'German Pilsener': 4, 'American Double / Imperial IPA': 5, 'Herbed / Spiced Beer': 6, 'Oatmeal Stout': 7, 'American Pale Lager': 8, 'Rauchbier': 9, 'American Pale Ale (APA)': 10, 'American Porter': 11, 'Belgian Strong Dark Ale': 12, 'Russian Imperial Stout': 13, 'American Amber / Red Ale': 14, 'American Strong Ale': 15, 'MÃ¤rzen / Oktoberfest': 16, 'American Adjunct Lager': 17, 'American Blonde Ale': 18, 'American IPA': 19, 'Fruit / Vegetable Beer': 20, 'English Bitter': 21, 'English Porter': 22, 'Irish Dry Stout': 23, 'American Barleywine': 24, 'American Double / Imperial Stout': 25, 'Doppelbock': 26, 'American Stout': 27, 'Maibock / Helles Bock': 28, 'Euro Strong Lager': 29, 'Light Lager': 30, 'Euro Pale Lager': 31, 'Bock': 32, 'English India Pale Ale (IPA)': 33, 'Altbier': 34, 'KÃ¶lsch': 35, 'Pumpkin Ale': 36, 'Rye Beer': 37, 'American Pale Wheat Ale': 38, 'Milk / Sweet Stout': 39, 'Schwarzbier': 40, 'Munich Dunkel Lager': 41, 'Scottish Ale': 42, 'Witbier': 43, 'Saison / Farmhouse Ale': 44, 'American Black Ale': 45, 'English Brown Ale': 46, 'English Barleywine': 47, 'Extra Special / Strong Bitter (ESB)': 48, 'Euro Dark Lager': 49, 'Scotch Ale / Wee Heavy': 50, 'English Pale Ale': 51, 'Belgian Strong Pale Ale': 52, 'Belgian Pale Ale': 53, 'Tripel': 54, 'American Brown Ale': 55, 'Smoked Beer': 56, 'Dunkelweizen': 57, 'Dubbel': 58, 'Winter Warmer': 59, 'Belgian Dark Ale': 60, 'Irish Red Ale': 61, 'English Stout': 62, 'Czech Pilsener': 63, 'Belgian IPA': 64, 'Black & Tan': 65, 'Cream Ale': 66, 'American Wild Ale': 67, 'Scottish Gruit / Ancient Herbed Ale': 68, 'Wheatwine': 69, 'American Malt Liquor': 70, 'Munich Helles Lager': 71, 'Baltic Porter': 72, 'Old Ale': 73, 'Quadrupel (Quad)': 74}

b) Below are all of theta values represented as a theta vector:

[3.60681818 -0.0058248 0.12193999 -0.35681818 -0.62765152 0.33596789
-0.38622995 0.1527972 -0.58598485 0.44318182 0.02739474 0.3305986
0.12651515 0.69564175 -0.11634199 -0.1798951 -0.22045455 -0.73802386
-0.45410882 0.35359848 0.11320382 -0.02450111 0.09815076 0.22651515
0.45638407 0.8416167 -0.41285266 0.20312334 -0.10681818 -0.83277972
-1.23390152 -0.92019846 -0.74318182 -0.13106061 -0.18884943 0.10049889
0.26709486 0.3763751 -0.17824675 0.71136364 0.12395105 -0.50681818
0.26818182 -0.09003966 0.20984848 0.19621212 0.03420746 0.76540404
0.14466111 0.11433566 0.48544372 -0.31931818 0.46842454 0.16385851
0.05895722 0.26761104 -0.4664673 0.25681818 0.13786267 0.01699134
-0.27061129 -0.63806818 -0.10223103 -0.2937747 0.3449362 0.33580477
-0.65227273 0.58195733 0.38356643 0.01818182 -1.00681818 -0.20681818
0.65508658 0.60472028 0.39318182]

c) MSE of training part is: 0.36784028
MSE of testing part is: 0.43366951042

d) Code for this problem is as follows:

```
import numpy
import urllib
import scipy.optimize
import random
from urllib import request

def parseData(fname):
   for l in urllib.request.urlopen(fname):
      yield eval(l)

print ("Reading data...")
data = list(parseData("http://jmcauley.ucsd.edu/cse255/data/beer/beer_50000.json"))
print ("done")

def problem1_4():
      m = {} #dictionary to store numbers of reviews for each style of beer
      for d in data:
            if d['beer/style'] in m:
                  m[d['beer/style']] += 1
            else:
                  m[d['beer/style']] = 1

      n = {} #store >=50 reviews beers
      excluded = set([]) #store those beers which get excluded
      for key,value in m.items():
            if value >= 50:
                  n[key] = len(n)+1
            else:
                  excluded.add(key)

      print ('There are ', len(n), 'beers which have more than 50 reviews. They are:')
      print (n)

      X=[]
      Y=[]
      for d in data:
            if d['beer/style'] not in excluded:
                  cur = [0] * (len(n)+1)
                  cur[0] = 1
                  cur[n[d['beer/style']]] = 1
                  X.append(cur)
```

```
                Y.append(d['review/taste'])
            else:
                cur = [0] * (len(n)+1)
                cur[0] = 1
                X.append(cur)
                Y.append(d['review/taste'])

        X1 = X[:int(len(X)/2)]
        X2 = X[int(len(X)/2):]
        Y1 = Y[:int(len(Y)/2)]
        Y2 = Y[int(len(Y)/2):]
        len1 = len(X1)
        len2 = len(X2)
        theta,residuals,rank,s = numpy.linalg.lstsq(X1, Y1)
        print (theta)
        print ('MSE of training part is: ', residuals/len1)

        X2 = numpy.matrix(X2)
        Y2 = numpy.matrix(Y2)
        A = numpy.dot(theta, X2.T)
        mse = numpy.square(A - Y2).mean()
        print ('MSE of testing part is: ', mse)

problem1_4()
```

Tasks – Classification (week 2)

5. First, let's train a predictor that estimates whether a beer is an `American IPA' using two features:

['beer/ABV'; 'review/taste'].

Train your predictor using an SVM classifier (see the code provided in class) - remember to train on the first half and test on the second half. Use a regularization constant of C = 1000 as in the code stub. What is the accuracy (percentage of correct classifications) of the predictor on the train and test data? (1 mark)

Answer:

a) The accuracy on the train data is 0.91296
The accuracy on the test data is 0.92112

b) The code for this problem is as follows:

```
import numpy
import urllib
import scipy.optimize
import random
from urllib import request
from sklearn import svm

def parseData(fname):
   for l in urllib.request.urlopen(fname):
      yield eval(l)

print ("Reading data...")
data = list(parseData("http://jmcauley.ucsd.edu/cse255/data/beer/beer_50000.json"))
print ("done")

X = [[b['beer/ABV'],b['review/taste']] for b in data]
y = ["American IPA" in b['beer/style'] for b in data]

length = len(X)
X_train = X[:int(length/2)]
y_train = y[:int(length/2)]
X_test = X[int(length/2):]
y_test = y[int(length/2):]

# Create a support vector classifier object, with regularization parameter C = 1000
clf = svm.SVC(C=1000, kernel='linear')
clf.fit(X_train, y_train)

train_predictions = clf.predict(X_train)
test_predictions = clf.predict(X_test)
```

```
match_train = [(x==y) for x,y in zip(train_predictions, y_train)]
match_test    = [(x==y) for x,y in zip(test_predictions, y_test)]

print ("The accuracy on the train data is", sum(match_train)*1.0/len(match_train))
print ("The accuracy on the test data is", sum(match_test)*1.0/len(match_test))
```

6. Considering the 'American IPA' style, can you come up with a more accurate predictor (e.g. using features from the text, or otherwise)? Write down the feature vector you design, and report its train/test accuracy (1 mark).

Answer:

a) Choose features as follows:

['review/appearance', 'review/palate']

And its train accuracy is 0.9136;

Its test accuracy is 0.92188

b) The code for this problem is as follows:

```
import numpy
import urllib
from urllib import request
from sklearn import svm

def parseData(fname):
   for l in urllib.request.urlopen(fname):
      yield eval(l)

print ("Reading data...")
data =
 list(parseData("http://jmcauley.ucsd.edu/cse255/data/beer/beer_50000.json"))
print ("done")

X = [[b['review/appearance'],b['review/palate']] for b in data]
y = ["American IPA" in b['beer/style'] for b in data]

length = len(X)
X_train = X[:int(length/2)]
y_train = y[:int(length/2)]
X_test = X[int(length/2):]
y_test = y[int(length/2):]

# Create a support vector classifier object, with regularization parameter C =
1000
print ("Training...")
clf = svm.SVC(C=1000, kernel='linear')
clf.fit(X_train, y_train)
print ("Training is done.")

train_predictions = clf.predict(X_train)
test_predictions = clf.predict(X_test)

match_train = [(x==y) for x,y in zip(train_predictions, y_train)]
```

```
    match_test    = [(x==y) for x,y in zip(test_predictions, y_test)]

    print ("The accuracy on the train data is", sum(match_train)*1.0/len(match_train))
print ("The accuracy on the test data is", sum(match_test)*1.0/len(match_test))
```

7. What effect does the regularization constant C have on the training/test performance? Report the train/test accuracy of your predictor from the previous question for $C \in (0.1, 10, 1000, 100000)$.

Answer:

a) Constant C has a great effect on the training/test performance. The bigger C is, the more time is needed to train the model.

b) Below are train/test accuracy data for different Cs:

When C = 0.1:
The accuracy on the train data is 0.9136
The accuracy on the test data is 0.92188
When C = 10:
The accuracy on the train data is 0.9136
The accuracy on the test data is 0.92188
When C = 1000:
The accuracy on the train data is 0.9136
The accuracy on the test data is 0.92188
When C = 100000:
The accuracy on the train data is 0.9136
The accuracy on the test data is 0.92188

8. (Hard) Finally, let's fit a model (for the problem from Q5) using logistic regression. A code stub has been provided to perform logistic regression using the above model on http://jmcauley.ucsd.edu/cse258/code/homework1.py Code for the log-likelihood has been provided in the code stub (f) but code for the derivative is incomplete (fprime)
   Complete the code stub for the derivative (fprime) and provide your solution. What is the log-likelihood of after convergence, and what is the accuracy (on the test set) of the resulting model (1 mark)?

Answer:

a) Fprime function is as follows:

```
def fprime(theta, X, y, lam):
    dl = [0.0]*len(theta)
    for k in range(len(theta)):
        res = 0
        for i in range(len(X)):
            logit = inner(X[i], theta)
            res += X[i][k]*(1-sigmoid(logit))
            if not y[i]:
                res -=X[i][k]
        res -= 2*lam*theta[k]
        dl[k] = res
    # Negate the return value since we're doing gradient *ascent*
return numpy.array([-x for x in dl])
```

b) Final log likelihood = -6786.94770036

c) Accuracy on the test set of the resulting model is 0.9218

d) The overall code for this problem is the following:

```
import numpy
import urllib
import scipy.optimize
import random
from math import exp
from math import log
from urllib import request

 def parseData(fname):
    for l in urllib.urlopen(fname):
      yield eval(l)


print ("Reading data...")
data =
```

```
 list(parseData("http://jmcauley.ucsd.edu/cse255/data/beer/beer_50000.json"))
print ("done")

def inner(x,y):
   return sum([x[i]*y[i] for i in range(len(x))])

def sigmoid(x):
   return 1.0 / (1 + exp(-x))

# NEGATIVE Log-likelihood
def f(theta, X, y, lam):
   loglikelihood = 0
   for i in range(len(X)):
      logit = inner(X[i], theta)
      loglikelihood -= log(1 + exp(-logit))
      if not y[i]:
         loglikelihood -= logit
   for k in range(len(theta)):
      loglikelihood -= lam * theta[k]*theta[k]
   #print ("ll =", loglikelihood)
   return -loglikelihood

# NEGATIVE Derivative of log-likelihood
def fprime(theta, X, y, lam):
   dl = [0.0]*len(theta)
   for k in range(len(theta)):
         res = 0
         for i in range(len(X)):
               logit = inner(X[i], theta)
               res += X[i][k]*(1-sigmoid(logit))
               if not y[i]:
                     res -=X[i][k]
         res -= 2*lam*theta[k]
         dl[k] = res
   # Negate the return value since we're doing gradient *ascent*
   return numpy.array([-x for x in dl])


X = [[b['beer/ABV'],b['review/taste']] for b in data]
y = ["American IPA" in b['beer/style'] for b in data]

X_train = X[:int(len(X)/2)]
y_train = y[:int(len(X)/2)]
X_test = X[int(len(X)/2):]
```

```
y_test = y[int(len(X)/2):]

# If we wanted to split with a validation set:
#X_valid = X[len(X)/2:3*len(X)/4]
#X_test = X[3*len(X)/4:]

# Use a library function to run gradient descent (or you can implement yourself!)
theta,l,info = scipy.optimize.fmin_l_bfgs_b(f, [0]*len(X[0]), fprime, args = (X_train, y_train, 1.0))
print ("Final log likelihood =", -l)
print (theta)

y_predicted = [0] * len(X_test)
for i in range(len(X_test)):
    if inner(theta, X_test[i]) > 0:
        y_predicted[i] = 1

match_test  = [(x==y) for x,y in zip(y_predicted, y_test)]
print ("Accuracy = ", sum(match_test)*1.0/len(match_test)) # Compute the accuracy
```