1.
There are total 95 different kind of beers. The names, frequency and average stars are as follows:

```
[('Hefeweizen', 618, 3.635113268608414),
 ('English Strong Ale', 164, 3.7560975609756095),
 ('Foreign / Export Stout', 55, 3.2545454545454544),
 ('German Pilsener', 586, 3.667235494880546),
 ('American Double / Imperial IPA', 3886, 4.033324755532681),
 ('Herbed / Spiced Beer', 73, 3.4452054794520546),
 ('Oatmeal Stout', 102, 3.7745098039215685),
 ('American Pale Lager', 123, 3.2154471544715446),
 ('Rauchbier', 1938, 4.067853457172343),
 ('American Pale Ale (APA)', 2288, 3.649694055944056),
 ('American Porter', 2230, 4.081838565022421),
 ('Belgian Strong Dark Ale', 146, 3.6952054794520546),
 ('Russian Imperial Stout', 2695, 4.300371057513915),
 ('American Amber / Red Ale', 665, 3.513533834586466),
 ('American Strong Ale', 166, 3.569277108433735),
 ('MÃ¤rzen / Oktoberfest', 557, 3.5933572710951527),
 ('American Adjunct Lager', 242, 2.9483471074380163),
 ('American Blonde Ale', 357, 3.2549019607843137),
 ('American IPA', 4113, 4.00085096036956),
 ('Fruit / Vegetable Beer', 1355, 3.607749077490775),
 ('English Bitter', 267, 3.5374531835205993),
 ('English Porter', 367, 3.70708446866485),
 ('Irish Dry Stout', 101, 3.623762376237624),
 ('American Barleywine', 825, 4.064242424242424),
 ('American Double / Imperial Stout', 5964, 4.479963112005366),
 ('Doppelbock', 873, 3.9828178694158076),
 ('American Stout', 591, 3.8197969543147208),
 ('Maibock / Helles Bock', 225, 3.7466666666666666),
 ('Dortmunder / Export Lager', 31, 3.4193548387096775),
 ('Euro Strong Lager', 329, 2.8480243161094223),
 ('Low Alcohol Beer', 7, 2.7142857142857144),
 ('Light Lager', 503, 2.39662027833002),
 ('Euro Pale Lager', 701, 2.962910128388017),
 ('Bock', 148, 3.189189189189189),
 ('English India Pale Ale (IPA)', 175, 3.4714285714285715),
 ('Altbier', 165, 3.403030303030303),
 ('KÃ¶lsch', 94, 3.6968085106382977),
 ('Pumpkin Ale', 560, 3.7875),
 ('Rye Beer', 1798, 4.213570634037819),
```

('American Pale Wheat Ale', 154, 3.3344155844155843),
('Milk / Sweet Stout', 69, 3.782608695652174),
('Schwarzbier', 53, 3.6226415094339623),
('Munich Dunkel Lager', 141, 3.780141843971631),
('Vienna Lager', 33, 3.5303030303030303),
('American Amber / Red Lager', 42, 3.6904761904761907),
('Scottish Ale', 78, 3.7628205128205128),
('Witbier', 162, 3.5277777777777777),
('Saison / Farmhouse Ale', 141, 3.702127659574468),
('American Black Ale', 138, 3.8731884057971016),
('English Brown Ale', 495, 3.728282828282828),
('English Barleywine', 133, 4.360902255639098),
('Extra Special / Strong Bitter (ESB)', 667, 3.685157421289355),
('California Common / Steam Beer', 11, 3.3181818181818183),
('Euro Dark Lager', 144, 3.7048611111111111),
('Scotch Ale / Wee Heavy', 2776, 4.083393371757925),
('English Pale Ale', 1324, 3.483761329305136),
('Belgian Strong Pale Ale', 632, 4.056170886075949),
('Belgian Pale Ale', 144, 3.7395833333333335),
('Tripel', 257, 3.7840466926070038),
('Flanders Oud Bruin', 13, 3.923076923076923),
('American Brown Ale', 314, 3.7436305732484074),
('Smoked Beer', 61, 3.19672131147541),
('Dunkelweizen', 61, 3.4918032786885247),
('Dubbel', 165, 3.7363636363636363),
('Keller Bier / Zwickel Bier', 23, 3.869565217391304),
('Winter Warmer', 259, 3.6216216216216215),
('BiÃ¨re de Garde', 7, 3.9285714285714284),
('Belgian Dark Ale', 175, 3.34),
('Irish Red Ale', 83, 2.9819277108433737),
('Chile Beer', 11, 3.9545454545454546),
('English Stout', 136, 3.599264705882353),
('Czech Pilsener', 1501, 3.609593604263824),
('Belgian IPA', 128, 3.94921875),
('Black & Tan', 122, 3.942622950819672),
('Cream Ale', 69, 3.028985507246377),
('English Dark Mild Ale', 21, 3.7857142857142856),
('American Wild Ale', 98, 4.188775510204081),
('Weizenbock', 13, 3.3846153846153846),
('American Double / Imperial Pilsner', 14, 3.8214285714285716),
('Scottish Gruit / Ancient Herbed Ale', 65, 3.9076923076923076),
('Wheatwine', 455, 4.186813186813187),

('American Dark Wheat Ale', 14, 3.6785714285714284),
('American Malt Liquor', 90, 2.2555555555555555),
('Munich Helles Lager', 650, 3.959230769230769),
('Kristalweizen', 7, 2.7857142857142856),
('English Pale Mild Ale', 21, 3.5952380952380953),
('Baltic Porter', 514, 4.213035019455253),
('Old Ale', 1052, 4.096007604562738),
('Quadrupel (Quad)', 119, 3.596638655462185),
('Braggot', 26, 3.8076923076923075),
('Lambic - Fruit', 6, 3.75),
('Lambic - Unblended', 10, 3.3),
('Eisbock', 8, 3.75),
('Flanders Red Ale', 2, 3.25),
('Berliner Weissbier', 10, 3.55)]

2.
review/taste = 3.91520474208 + 0.0856462182857 *[beer is an American IPA](1 if beer is American IPA,0 otherwise)
theta0 = 3.91520474208 theta1 = 0.0856462182857

Theta0 represents the average star of review/taste of beers which don't belong to American IPA. Theta1 represents the extra star added on the theta0 if the beer belongs to American IPA.

3.
MSE of training data = 0.558107286559
MSE of test data = 0.468410050967

4.
thetas for extended model are :
```
[ 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]

MSE of training data = 0.36784027709
MSE of test data = 0.43366951042

5.
accuracy of predictor on the training data(C=1000): 0.91296
accuracy of test data(C=1000) : 0.92112

6.
I choose two features to decide whether the beer is "American IPA" or not. If in the "review\text", there exist the word "IPA" or "PA" then predict the type of beer is "American IPA". Another feature is the ABV of the beer.

```
def feature(datum):
   feat=[]
   feat.append(datum['beer/ABV'])
   if "IPA" or "PA" in datum['review/text']:
      feat.append(1)
   else:
      feat.append(0)
   return feat
```

accuracy of predictor on the training data(C=1000): 0.91328
accuracy of test data(C=1000) : 0.92148

7.
C represents the penalty for misclassification. If we set larger constant C, we can get more accuracy on the training data.

C	Training accuracy	Test accuracy
0.1	0.9136	0.92188
10	0.9136	0.92188
1000	0.91328	0.92188
100000	0.88464	0.91868

8.
likelihood after convergence is -6690.83159755
accuracy of the test set of the resulting model is 0.9136.

Source code:

```
import numpy as np
import urllib.request
import scipy.optimize
import random
from collections import Counter

def parseData(fname):
   for l in urllib.request.urlopen(fname):
      yield eval(l)
print ("Reading data...")
data=list(parseData("http://jmcauley.ucsd.edu/cse190/data/beer/beer_50000.json"))
print ("done!")

def feature(datum):
   feat=[1]
   return feat
```

#problem1

```
   def ave_taste(i):
      t=[]
      for d in data:
         if d['beer/style']==i:
            t.append(d['review/taste'])
      average_for_i=sum(t)/len(t)
      average_taste_all.append(average_for_i)
   beer_review=[d['beer/style'] for d in data]
   beer_type=Counter(beer_review).keys()
   beer_type_list=list(beer_type)
   beer_freq=Counter(beer_review).values()
   average_taste_all=[]
   for i in beer_type_list:
      ave_taste(i)
   beer_final=list(zip(beer_type,beer_freq,average_taste_all))
   print("There are ",len(beer_type),"different kinds of beers. ")
   print("The specific type and its corresponding number of reviews and average value of 
'review/taste' are showed below: ")
   print(beer_final)
```

#problem2

```
def feature(datum):
   feat=[1]
```

```
    if datum['beer/style']=='American IPA':
        feat.append(1)
    else:
        feat.append(0)
    return feat
    X=[feature(d) for d in data]
    y=[d['review/taste'] for d in data]
    theta,residuals,rank,s=np.linalg.lstsq(X,y)
    print("review/taste = ",theta[0],"+",theta[1],"*[beer is an American IPA](1 if beer is
American IPA,0 otherwise)")
    print("theta0 =",theta[0]," theta1 = ",theta[1])
```

#problem3

```
    from sklearn.metrics import mean_squared_error
    data_train=data[:int(len(data)/2)]
    data_test=data[int(len(data)/2):]
    X=[feature(d) for d in data_train]
    y=[d['review/taste'] for d in data_train]
    theta,residuals,rank,s=np.linalg.lstsq(X,y)


    test_true=[d['review/taste'] for d in data_test]
    X1=[feature(d) for d in data_test]
    X1=np.matrix(X1)
    y=np.matrix([theta[0],theta[1]]).T
    test_predict=(X1*y).T
    test_predict=test_predict.tolist()[0]
    MSE_test=mean_squared_error(test_true,test_predict)


    print("For training set : ")
    print("review/taste = ",theta[0],"+",theta[1],"*[beer is an American IPA](1 if beer is
American IPA,0 otherwise) ")
    print("theta0 =",theta[0]," theta1 = ",theta[1])
    print("MSE of training data = ",residuals[0]/len(data_train))
    print("MSE of test data = ",MSE_test)
```

#problem4

```
    def feature(d):
        feat=[]
        feat.append(1)
        if d['beer/style'] not in extend_type:
```

```
        for i in range(0,74):
            feat.append(0)
    else:
        name=d['beer/style']
        index=extend_type.index(name)
        for i in range(0,index):
            feat.append(0)
        feat.append(1)
        for i in range(0,73-index):
            feat.append(0)
    return feat
#get the type of beer which has no less than 50 reviews
from sklearn.metrics import mean_squared_error
extend_type=[]
for i in range(len(beer_final)):
    if beer_final[i][1]>=50:
        extend_type.append(beer_final[i][0])
data_train=data[:int(len(data)/2)]
data_test=data[int(len(data)/2):]
#regression
X=[feature(d) for d in data_train]
y=[d['review/taste'] for d in data_train]
theta,residuals,rank,s=np.linalg.lstsq(X,y)

test_true=[d['review/taste'] for d in data_test]
X1=[feature(d) for d in data_test]
X1=np.matrix(X1)
y=np.matrix(theta).T
test_predict=(X1*y).T
test_predict=test_predict.tolist()[0]
MSE_test=mean_squared_error(test_true,test_predict)

print("thetas for extended model are : ")
print(theta)

print("MSE of training data = ",residuals[0]/len(data_train))
print("MSE of test data = ",MSE_test)
```

#problem 5

```
def feature(datum):
    feat=[]
```

```
        feat.append(datum['beer/ABV'])
        feat.append(datum['review/taste'])
        return feat

    X=[feature(d) for d in data]
    y=['American IPA' in d['beer/style'] for d in data]
    X_train=X[:int(len(X)/2)]
    X_test=X[int(len(X)/2):]
    y_train=y[:int(len(y)/2)]
    y_test=y[int(len(y)/2):]

    clf=svm.SVC(C=1000,kernel='linear')
    clf.fit(X_train,y_train)
    train_prediction=clf.predict(X_train)
    test_prediction=clf.predict(X_test)
    match_train=[(x==y) for x,y in zip(y_train,train_prediction)]
    accuracy_train=sum(match_train)/len(match_train)
    match_test=[(x==y) for x,y in zip(y_test,test_prediction)]
    accuracy_test=sum(match_test)/len(match_test)
    print("accuracy of predictor on the training data(C=1000): ",accuracy_train)
    print("accuracy of test data(C=1000) :",accuracy_test)
```

#problem 6

```
def feature(datum):
    feat=[]
    feat.append(datum['beer/ABV'])
    if "IPA" or "PA" in datum['review/text']:
        feat.append(1)
    else:
        feat.append(0)
    return feat

    X=[feature(d) for d in data]
    y=['American IPA' in d['beer/style'] for d in data]
    X_train=X[:int(len(X)/2)]
    X_test=X[int(len(X)/2):]
    y_train=y[:int(len(y)/2)]
    y_test=y[int(len(y)/2):]
    clf=svm.SVC(C=1000,kernel='linear')
    clf.fit(X_train,y_train)
    train_prediction=clf.predict(X_train)
```

```
    test_prediction=clf.predict(X_test)
    match_train=[(x==y) for x,y in zip(y_train,train_prediction)]
    accuracy_train=sum(match_train)/len(match_train)
    match_test=[(x==y) for x,y in zip(y_test,test_prediction)]
    accuracy_test=sum(match_test)/len(match_test)
    print("accuracy of predictor on the training data(C=1000): ",accuracy_train)
    print("accuracy of test data(C=1000) :",accuracy_test)
```

#problem 7

```
clf=svm.SVC(C=0.1,kernel='linear')
clf.fit(X_train,y_train)
train_prediction=clf.predict(X_train)
test_prediction=clf.predict(X_test)
match_train=[(x==y) for x,y in zip(y_train,train_prediction)]
accuracy_train=sum(match_train)/len(match_train)
match_test=[(x==y) for x,y in zip(y_test,test_prediction)]
accuracy_test=sum(match_test)/len(match_test)
print("accuracy of predictor on the training data(C=0.1): ",accuracy_train)
print("accuracy of test data(C=0.1) :",accuracy_test)

clf=svm.SVC(C=10,kernel='linear')
clf.fit(X_train,y_train)
train_prediction=clf.predict(X_train)
test_prediction=clf.predict(X_test)
match_train=[(x==y) for x,y in zip(y_train,train_prediction)]
accuracy_train=sum(match_train)/len(match_train)
match_test=[(x==y) for x,y in zip(y_test,test_prediction)]
accuracy_test=sum(match_test)/len(match_test)
print("accuracy of predictor on the training data(C=10): ",accuracy_train)
print("accuracy of test data(C=10) :",accuracy_test)

clf=svm.SVC(C=100000,kernel='linear')
clf.fit(X_train,y_train)
train_prediction=clf.predict(X_train)
test_prediction=clf.predict(X_test)
match_train=[(x==y) for x,y in zip(y_train,train_prediction)]
accuracy_train=sum(match_train)/len(match_train)
match_test=[(x==y) for x,y in zip(y_test,test_prediction)]
accuracy_test=sum(match_test)/len(match_test)
print("accuracy of predictor on the training data(C=100000): ",accuracy_train)
print("accuracy of test data(C=100000) :",accuracy_test)
```

#problem 8

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

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

print ("Reading data...")
data=list(parseData("http://jmcauley.ucsd.edu/cse190/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

   def fprime(theta, X, y, lam):
      dl = [0.0]*len(theta)
      sig=[]
      for i in range(len(X)):
         logit=inner(X[i],theta)
         sig.append(1-sigmoid(logit))
      X=numpy.matrix(X).T
      sig=numpy.matrix(sig).T
```

```
    y_opposite=[i*-1 for i in y]
    ones=[1]*len(y)
    y_flip=[sum(n) for n in zip(*[ones,y_opposite])]
    y_flip=numpy.matrix(y_flip).T
    theta=numpy.matrix(theta).T
    final=X*(sig-y_flip)-2*lam*theta
    dl=final.flatten().tolist()[0]
    pass

    return numpy.array([-x for x in dl])

  def feature(datum):
    feat=[]
    feat.append(1)
    feat.append(datum['beer/ABV'])
    feat.append(datum['review/taste'])
    return feat
  X=[feature(d) for d in data]
  y=['American IPA' in d['beer/style'] for d in data]
  X_train = X[:int(len(X)/2)]
  X_test = X[int(len(X)/2):]
  y_train=y[:int(len(y)/2)]
  y_test=y[int(len(y)/2):]

  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)

  X_test=numpy.matrix(X_test)
  theta=numpy.matrix(theta).T

  predict_test=X_test*theta
  predict_test_answer=[1 if predict_test[i]>0 else 0 for i in range (len(predict_test))]
  match_test=[(x==y) for x,y in zip(y_train,predict_test_answer)]
  accuracy_test=sum(match_test)/len(match_test)
  print ("Accuracy for test data = ",accuracy_test)
```