hw1_keliu

October 14, 2018

0.1 Homework1 [written in python 3.7]

0.1.1 1. What is the distribution of ratings in the dataset (for 'review/taste')? That is, how many 1-star, 2-star, 3-star (etc.) reviews are there? You may write out the values or include a simple plot.

In [1]:
```
import numpy  #matrix algebra
import urllib  #read data from web
import scipy.optimize
import random
from sklearn import svm  # library for classification
```

In [2]:
```
def parseData(fname):
  for l in urllib.request.urlopen(fname):
    yield eval(l)
```

In [3]:
```
print ("Reading data...")
data = list(parseData("http://jmcauley.ucsd.edu/cse190/data/beer/beer_50000.json"))
print ("done")
```

```
Reading data...
done
```

In [4]:
```
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

taste_stars={}
# define a dictionary whose key is a rating and value is the number of certain rating
for d in data:
  if d['review/taste'] not in taste_stars:
    taste_stars[d['review/taste']] = 1
  else:
    taste_stars[d['review/taste']] +=1
taste_stars
#plt.hist(taste_stars.values(),len(taste_stars));
```

Out[4]:
```
{1.5: 343,
 3.0: 4137,
```

```
4.5: 12883,
3.5: 8797,
4.0: 16575,
2.0: 1099,
5.0: 4331,
2.5: 1624,
1.0: 211}
```

0.1.2 2. Train a simple predictor to predict a beer's 'taste' score using two features:

$review/taste = {}_0 + {}_1[beer\ is\ a\ Hefeweizen] + {}_2 beer/ABV$

0.1.3 Report the values of 0, 1, and 2. Briefly describe your interpretation of these values, i.e., what do 0, 1, and 2 represent ?

```
In [5]: def feature(datum):
          feat = [1]
          if datum['beer/style'] == 'Hefeweizen':
            feat.append(1)
          else:
            feat.append(0)
          feat.append(datum['beer/ABV'])
          return feat
```

```
In [6]: X= [feature(d) for d in data]
        y= [d['review/taste'] for d in data]
        theta,residuals,rank,s = numpy.linalg.lstsq(X, y,rcond=-1)
        theta
```

```
Out[6]: array([ 3.11795084, -0.05637406,  0.10877902])
```

we get ${}_0 = 3.11795084$,${}_1 = -0.05637406$ and ${}_2 = 0.10877902$. ${}_0$ shows when a beer is not a Hefeiwenzen and beer/ABV=0, the predicted review of taste is 3.12. When beer is a Hefeiweizen, the predicted review of taste is ${}_1$, that is, 0.056 lower in taste reviews. And since ${}_2$ is 0.108, the review of taste increases with the ascending ABV.

0.1.4 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 ?

```
In [8]: train_X=[feature(d) for d in data[:len(data)//2]]  # the first half for training
        train_y=[d['review/taste'] for d in data[:len(data)//2]]
        theta,residuals,rank,s = numpy.linalg.lstsq(train_X, train_y,rcond=-1)
        theta
```

```
Out[8]: array([ 2.99691466, -0.03573098,  0.11672256])
```

```
In [9]: def f(theta, X, y):
          theta = numpy.matrix(theta).T
```

```
    X = numpy.matrix(X)
    y = numpy.matrix(y).T
    diff = X*theta - y
    diffSq = diff.T*diff
    mse = diffSq / len(X)    # MSE
#  print ("MSE =", mse.flatten().tolist())
    return mse.flatten().tolist()[0]
```

```
In [10]: f(theta, train_X, train_y)    # MSE for training set
```

```
Out[10]: [0.48396805601342413]
```

```
In [11]: # the second half for testing the quality of parameters
         test_X=[feature(d) for d in data[len(data)//2:]]
         test_y=[d['review/taste'] for d in data[len(data)//2:]]
         f(theta, test_X, test_y)
```

```
Out[11]: [0.4237065211985415]
```

The model' MSE on the traing set is 0.483968 and on the test set is 0.4237065

0.1.5 4.Using the first half for training and the second half for testing may lead to unexpected results (e.g. the training error could be higher than the test error). Repeat the above experiment by using a random 50% split of the data (i.e., half for training, half for testing, after first shuffling the data). Report the MSE on the train and test set, and suggest one possible reason why the result may be different from the previous experiment.

```
In [19]: import random
         random.shuffle(data)   # shuffle the data
         train_X=[feature(d) for d in data[:len(data)//2]]    # the first half for training
         train_y=[d['review/taste'] for d in data[:len(data)//2]]
         theta,residuals,rank,s = numpy.linalg.lstsq(train_X, train_y,rcond=-1)
         f(theta, train_X, train_y)    # MSE for training set
```

```
Out[19]: [0.4463668472935334]
```

```
In [20]: test_X=[feature(d) for d in data[len(data)//2:]]
         test_y=[d['review/taste'] for d in data[len(data)//2:]]
         f(theta, test_X, test_y)
```

```
Out[20]: [0.45299269855573393]
```

In the case, the training error is lower than the test error. The unexpected results may come out if the model is under-fitting or if the first half of data have higher noise.

0.1.6 5. Modify your experiment from Question 4 to use the features

$review/taste = {}_0 + {}_1[ABV\ if\ beer\ is\ a\ Hefeweizen] + {}_2[ABV\ if\ beer\ is\ not\ a\ Hefeweizen]$

e.g. the first beer in the dataset would have feature [1, 5.0, 0] since the beer is a Hefeweizen. Report the training and testing MSE of this method.

```
In [21]: def feature(datum):
           feat = [1]
           if datum['beer/style'] == 'Hefeweizen':
             feat.append(datum['beer/ABV'])
             feat.append(0)
           else:
             feat.append(0)
             feat.append(datum['beer/ABV'])
           return feat

In [22]: train_X1=[feature(d) for d in data[:len(data)//2]]
         train_y1=[d['review/taste'] for d in data[:len(data)//2]]
         theta,residuals,rank,s = numpy.linalg.lstsq(train_X1, train_y1,rcond=-1)
         print('theta=',theta)
         f(theta, train_X1, train_y1)    # MSE for training set

theta= [3.1303229  0.09348837 0.10755352]


Out[22]: [0.4463703048302323]

In [23]: test_X1=[feature(d) for d in data[len(data)//2:]]
         test_y1=[d['review/taste'] for d in data[len(data)//2:]]
         f(theta, test_X1, test_y1)

Out[23]: [0.45297443751569955]
```

0.1.7 6. The model from Question 5 uses the same two features as the model from Questions 2-4 and has the same dimensionality. Comment on why the two models might perform differently

The model in Q5 shows the influence of Hefeweizen's ABV to the review and other beers' ABV to the review. The model in Q2-4 shows the ABV's influence to review and the influence to review if the beer is Hefeweizen.

0.1.8 7. First, let's train a predictor that estimates whether a beer is a 'Hefeweizen' using five features describing its rating:

$[review/taste, review/appearance, review/aroma, review/palate, review/overall]$.

0.1.9 Train your predictor using an SVM classifier (see the code provided in class). Use a random split of the data as we did in Question 4. 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?

```
In [24]: def feature(datum):
             feat = [datum['review/taste']]
             feat.append(datum['review/appearance'])
             feat.append(datum['review/aroma'])
```

```
        feat.append(datum['review/palate'])
        feat.append(datum['review/overall'])
        return feat
```

```
In [25]: X_train=[feature(d) for d in data[:len(data)//2]]
         y_train=[d['beer/style']=='Hefeweizen' for d in data[:len(data)//2]]
```

```
In [26]: # Create a support vector classifier object, with regularization parameter C = 1000
         clf = svm.SVC(C=1000, kernel='linear')
         clf.fit(X_train, y_train)
```

```
Out[26]: SVC(C=1000, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)
```

```
In [27]: X_test=[feature(d) for d in data[len(data)//2:]]
         y_test=[d['beer/style']=='Hefeweizen' for d in data[len(data)//2:]]
```

```
In [28]: train_predictions = clf.predict(X_train)
         test_predictions = clf.predict(X_test)
```

```
In [29]: correct_train= train_predictions==y_train
         trainAcc=sum(correct_train)/len(correct_train)
         trainAcc
```

```
Out[29]: 0.98752
```

```
In [30]: correct_test= test_predictions==y_test
         testAcc=sum(correct_test)/len(correct_test)
         testAcc
```

```
Out[30]: 0.98776
```

0.1.10 8. Considering same prediction problem as above, 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

```
In [31]: def feature(datum):
             feat = [datum['review/taste']]
             feat.append(datum['review/appearance'])
             feat.append(datum['review/aroma'])
             feat.append(datum['review/palate'])
             feat.append(datum['review/overall'])
             if 'Hefeweizen' in datum['review/text']:
                 feat.append(1)
             else:
                 feat.append(0)
             return feat
```

```
In [32]: X_train1=[feature(d) for d in data[:len(data)//2]]
         y_train1=[d['beer/style']=='Hefeweizen' for d in data[:len(data)//2]]
         # Create a support vector classifier object, with regularization parameter C = 1000
         clf = svm.SVC(C=1000, kernel='linear')
         clf.fit(X_train1, y_train1)

Out[32]: SVC(C=1000, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

In [33]: X_test1=[feature(d) for d in data[len(data)//2:]]
         y_test1=[d['beer/style']=='Hefeweizen' for d in data[len(data)//2:]]
         train1_predictions = clf.predict(X_train1)
         test1_predictions = clf.predict(X_test1)

In [34]: correct_train1= train1_predictions==y_train1
         trainAcc1=sum(correct_train1)/len(correct_train1)
         trainAcc1

Out[34]: 0.98852

In [35]: correct_test1= test1_predictions==y_test1
         testAcc1=sum(correct_test1)/len(correct_test1)
         testAcc1

Out[35]: 0.98852
```