Sihao_Gu_homework1

October 15, 2018

0.1 CSE 258, Fall 2018: Homework 1

```
In [1]: ##### Task - Regression
        import numpy
        import urllib.request
        import scipy.optimize
        import random
        from sklearn.metrics import mean_squared_error

In [2]: def parseDataFromFile(fname):
          for l in open(fname):
            yield eval(l)

In [3]: data = list(parseDataFromFile("beer_50000.json"))
```

0.2 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 (1 mark).

```
In [4]: review_taste = []
        for i in range(0, len(data)):
            review_taste.append(data[i]['review/taste'])

In [5]: unique = numpy.unique(review_taste)

In [6]: unique

Out[6]: array([1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. ])

In [7]: review_taste1 = review_taste.count(1.0)
        review_taste2 = review_taste.count(1.5)
        review_taste3 = review_taste.count(2.0)
        review_taste4 = review_taste.count(2.5)
        review_taste5 = review_taste.count(3.0)
        review_taste6 = review_taste.count(3.5)
        review_taste7 = review_taste.count(4.0)
        review_taste8 = review_taste.count(4.5)
        review_taste9 = review_taste.count(5.0)
```

```
In [8]: print(review_taste1) #count of review/taste=1.0
        print(review_taste2) #count of review/taste=1.5
        print(review_taste3) #count of review/taste=2.0
        print(review_taste4) #count of review/taste=2.5
        print(review_taste5) #count of review/taste=3.0
        print(review_taste6) #count of review/taste=3.5
        print(review_taste7) #count of review/taste=4.0
        print(review_taste8) #count of review/taste=4.5
        print(review_taste9) #count of review/taste=5.0
```

```
211
343
1099
1624
4137
8797
16575
12883
4331
```

0.3 2. Train a simple predictor to predict a beer's 'taste' score using two features: review/taste ′ = theta0 + theta1 * [beer is a Hefeweizen] + theta2 * beer/ABV. Report the values of theta0, theta1, and theta2. Briefly describe your interpretation of these values, i.e., what do theta0, theta1, and theta2 represent (1 mark)?

```
In [9]: def feature(datum):
          feat = [1]
          if datum['beer/style'] == "Hefeweizen":
              feat.append(1)
          else:
              feat.append(0)
          feat.append(datum['beer/ABV'])
          return feat

        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=None)
```

```
In [10]: theta
```

```
Out[10]: array([ 3.11795084, -0.05637406,  0.10877902])
```

1 theta 0 is intercept, theta 1 and theta 2 are coeffecients of whether beer is Hefeweizen and beer's ABV.

1.1 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)?

```
In [11]: train = data[:25000]
         test = data[25000:]
```

```
In [12]: def feature(datum):
           feat = [1]
           if datum['beer/style'] == "Hefeweizen":
               feat.append(1)
           else:
               feat.append(0)
           feat.append(datum['beer/ABV'])
           return feat

         X = [feature(d) for d in train]
         y = [d['review/taste'] for d in train]
         theta,residuals,rank,s = numpy.linalg.lstsq(X, y, rcond=None)
```

```
In [13]: theta

Out[13]: array([ 2.99691466, -0.03573098,  0.11672256])
```

```
In [14]: value_train = theta * X
```

```
In [15]: y_pred_train = []
         for i in value_train:
             y_pred_train.append(i[0] + i[1] + i[2])
```

```
In [16]: # MSE on train set
         mean_squared_error(y, y_pred_train)

Out[16]: 0.4839680560134243
```

```
In [17]: y_test = [d['review/taste'] for d in test]
```

```
In [18]: X_test = [feature(d) for d in test]
         value_test =theta * X_test
```

```
In [19]: y_pred_test = []
         for i in value_test:
             y_pred_test.append(i[0] + i[1] + i[2])
```

```
In [20]: # MSE on test set
         mean_squared_error(y_test, y_pred_test)

Out[20]: 0.4237065211985418
```

1.2 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 (1 mark).

```
In [21]: from sklearn.utils import shuffle
```

```
In [22]: 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 [23]: data_shuffle = shuffle(data, random_state = 5)
```

```
In [24]: X = [feature(d) for d in data_shuffle]
         y = [d['review/taste'] for d in data_shuffle]
```

```
In [25]: X_train  = X[:25000]
         X_test = X[25000:]
         y_train = y[:25000]
         y_test = y[25000:]
```

```
In [26]: theta,residuals,rank,s = numpy.linalg.lstsq(X_train, y_train, rcond=None)
```

```
In [27]: theta
```

```
Out[27]: array([ 3.09739998, -0.05149452,  0.11127861])
```

```
In [28]: value_train_4 = theta * X_train
```

```
In [29]: y_pred_train_4 = []
         for i in value_train_4:
             y_pred_train_4.append(i[0] + i[1] + i[2])
```

```
In [30]: # MSE on train set
         mean_squared_error(y_train, y_pred_train_4)
```

```
Out[30]: 0.45246733676565154
```

```
In [31]: value_test_5 = theta * X_test
```

```
In [32]: y_pred_test_5 = []
         for i in value_test_5:
             y_pred_test_5.append(i[0] + i[1] + i[2])
```

```
In [33]: # MSE on test set
         mean_squared_error(y_test, y_pred_test_5)
```

```
Out[33]: 0.4469235257636903
```

1.3 5. Modify your experiment from Question 4 to use the features: review/taste ′ = theta0 + theta1 * [ABV if beer is a Hefeweizen] + theta2 * [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 (1 mark).

```
In [34]: 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 [35]: data_shuffle = shuffle(data, random_state = 5)
```

```
In [36]: X = [feature(d) for d in data_shuffle]
         y = [d['review/taste'] for d in data_shuffle]
```

```
In [37]: X_train  = X[:25000]
         X_test = X[25000:]
         y_train = y[:25000]
         y_test = y[25000:]
```

```
In [38]: theta,residuals,rank,s = numpy.linalg.lstsq(X_train, y_train, rcond=None)
```

```
In [39]: theta
```

```
Out[39]: array([3.09776369, 0.09988903, 0.11124353])
```

```
In [40]: value_train_6 = theta * X_train
```

```
In [41]: y_pred_train_6 = []
         for i in value_train_6:
             y_pred_train_6.append(i[0] + i[1] + i[2])
```

```
In [42]: # MSE on train set
         mean_squared_error(y_train, y_pred_train_6)
```

```
Out[42]: 0.45245562268694434
```

```
In [43]: value_test_7 = theta * X_test
```

```
In [44]: y_pred_test_7 = []
         for i in value_test_7:
             y_pred_test_7.append(i[0] + i[1] + i[2])
```

```
In [45]: # MSE on test set
         mean_squared_error(y_test, y_pred_test_7)
```

```
Out[45]: 0.4469246426210711
```

1.4 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 (1 mark).

2 Answer: Although Question 5 uses the same two features as the model from Questions 2-4 and has the same dimensionlity, the result of feature is different. For example, the first beer in the dataset in Questions 2-4 model is [1, 1, 5.0]. However, in Question 5 model, the feature of first beer in the dataset is [1, 5.0, 0]. Model in Questions 2-4 consider whether beer is Hefeweizen in the second position and put beer/ABV no matter which kinds of beer/style in the third position. Model in Questions 5 consider whether beer is Hefeweizne first. If yes,put corresponding beer/ABV in the second position and 0 in third position. If not, put 0 in the second position and corresponding beer/ABV in the third position.

```
In [46]: #####Task - Classfication
         import numpy
         import urllib
         import scipy.optimize
         import random
         from sklearn import svm
```

2.1 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']. 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? (1 mark)

```
In [47]: def select(datum):
             feat = []
             feat.append(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 [48]: data_shuffle = shuffle(data, random_state = 5)
```

```
In [49]: X = [select(d) for d in data_shuffle]
         y = [d['beer/style'] == 'Hefeweizen' for d in data_shuffle]
```

```
In [50]: X_train  = X[:25000]
         X_test = X[25000:]
         y_train = y[:25000]
         y_test = y[25000:]
```

```
In [51]: 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)
```

```
In [52]: #accuracy of the predictor on train set
         correct_train = train_predictions == y_train
```

```
In [53]: train_Acc = sum(correct_train)/ len(correct_train)
         train_Acc
```

```
Out[53]: 0.98768
```

```
In [54]: #accuracy of the predictor on test set
         correct_test = test_predictions == y_test
```

```
In [55]: test_Acc = sum(correct_test)/ len(correct_test)
         test_Acc
```

```
Out[55]: 0.9876
```

2.2 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 (1 mark).

```
In [56]: def select(datum):
           feat = []
           feat.append(datum['review/taste'])
           feat.append(datum['review/appearance'])
           feat.append(datum['review/aroma'])
           feat.append(datum['review/palate'])
           feat.append(datum['review/overall'])
           feat.append('weizen' in datum['beer/name'])
           return feat
```

3 I add whether 'weizen' in 'beer/name' in the feature vector and use the same five features in question 7.

```
In [57]: data_shuffle = shuffle(data, random_state = 5)
```

```
In [58]: X = [select(d) for d in data_shuffle]
         y = [d['beer/style'] == 'Hefeweizen' for d in data_shuffle]
```

```
In [59]: X_train  = X[:25000]
         X_test = X[25000:]
         y_train = y[:25000]
         y_test = y[25000:]

In [60]: 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)

In [61]: #accuracy of the predictor on train set
         correct_train = train_predictions == y_train

In [62]: train_Acc = sum(correct_train)/ len(correct_train)
         train_Acc

Out[62]: 0.99172

In [63]: #accuracy of the predictor on test set
         correct_test = test_predictions == y_test

In [64]: test_Acc = sum(correct_test)/ len(correct_test)
         test_Acc

Out[64]: 0.99188

In [ ]:
```