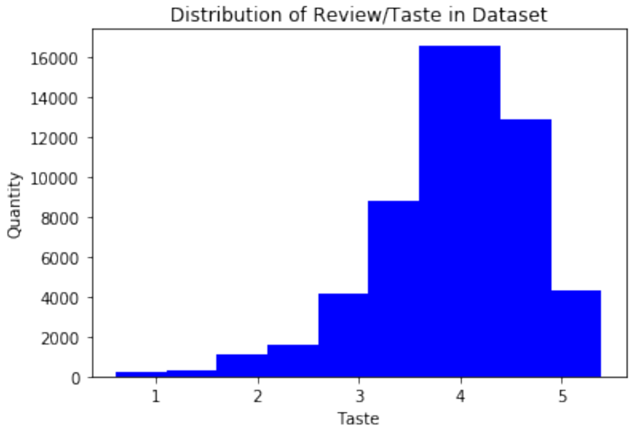
CSE 258 Assignment 1 Linbin Yang A53277054

1. I got the final distribution of review/taste as follows and draw graph to visualize it: {1.0: 211, 1.5: 343, 2.0: 1099, 2.5: 1624, 3.0: 4137, 3.5: 8797, 4.0: 16575, 4.5: 12883, 5.0: 4331}



2. I got: Theta 0: 3.11795084. Theta 1: -0.05637406 Theta 2: 0.10877902

After finishing training process and we got the final model, here theta 0 represents the initial taste score for each beer (ABV not given). Theta 2 represents how ABV of each beer influence taste score. Theta 1 measures how Hefeweizen beer performs differently from other kinds of beer on this beer_50000 dataset.

3. After splitting the dataset, I got the MSE as follows:
   MSE on train data: 0.483968
   MSE on test data: 0.423707

4. After shuffling the dataset and repeating the training process in question 3, I got the MSE as follows:
   MSE on train data: 0.448804
   MSE on test data: 0.450521

5. Modifying the feature of the model mentioned in question 4 and I got the following MSE values:
   MSE on train data: 0.448795
   MSE on test data: 0.450518

6. Although we use the same features, that is, ABV and whether the beer is Hefeweizen or not. But the meaning of theta varies and data distribution is also different.
   6.1. For model of Q3 and Q4, the difference is whether we shuffle the data or not. In order to make the model we trained on training data convincing and generalized, we must make sure that the data distributions on train data and test data are the same. So after shuffling the data, we got one different but more convincing model of Q4 compared with model of Q3.

6.2. For model of Q4 and Q5, the meaning of thetas is different. We have already mentioned the meaning of thetas for model of Q4. For thetas in model of Q5, theta2 and theta3 measures the differences in contribution of ABV to taste score between Hefeweizen beer and other beers. We use the same features, the final models we got are actually different. This is why they perform differently. (the difference is tiny on 50000 dataset)

7. I got: Acc of train data: 0.987440, Acc of test data is 0.987840

8. For this question I tried the following three ways:

8.1. Scaling all features to [0,1]
8.2. Analysis the review/text data on Hefeweizen beer and other beer. I got that words "Banana" and "Wheat" appears in the review/text of Hefeweizen more often. So I create one new feature, if the review/text of each sample in data has either of the two words, append 1 to feature set, append 0 if not.
8.3. Try different kernel function: I use RBF function as kernel.

After using 8.1 and 8.2, I find there is no improvement on performance of model.
After using 8.3, I find the Acc on training data increase to 99%, but the Acc on test set decreased.

(I also attached my analysis code for data processing in 8.2)

This is my code for texts and words analysis using NLTK in 8.2
We remove stops words and those words with frequency that is smaller than 5 in the review/text.

```
from nltk.book import *
from nltk.corpus import stopwords
import operator

stop_words = set(stopwords.words('english'))
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 ("We are done")

def extract_data_Hefeweizen(data):
    f = open("No_Hefeweizen.txt","w")
    for elem in data:
        if (elem['beer/style'] != 'Hefeweizen'):
            f.write(elem['review/text']+"\r\n")
    f.close()

def analysis_words(fpath):
    filter_words = {}
    fdist = FreqDist(gutenberg.words(fpath))
    for key,value in fdist.items():
        if key not in stop_words and value > 5:
            # we filter the stop words and those words of which freq <= 5
            filter_words[key] = value
    sorted_filter_words = sorted(filter_words.items(), key=operator.itemgetter(1))
    for key, value in sorted_filter_words:
        print (key, value)
```

In [102]:
```
# all library we need for this task
import numpy as np
import urllib.request
import scipy.optimize
import random
import matplotlib.pyplot as plt
```

In [103]:
```
#load data from website
def parseData(fname):
    for l in urllib.request.urlopen(fname):
        yield eval(l)
```

In [104]:
```
#store data to local variable
print ("Reading data.....")
data = list(parseData("http://jmcauley.ucsd.edu/cse255/data/beer/be
er_50000.json"))
print ("We are done")
```

```
Reading data.....
We are done
```

In [105]:
```
#init one dict for storing review/taste
TasteValue = {}
taste = [d['review/taste'] for d in data]
```

In [106]:
```
for elem in taste:
    if elem not in TasteValue.keys():
        TasteValue[elem] = 1
    else:
        TasteValue[elem] = TasteValue[elem] + 1
```

In [107]:
```
# here we get the distribution of review/taste
print (TasteValue)
```

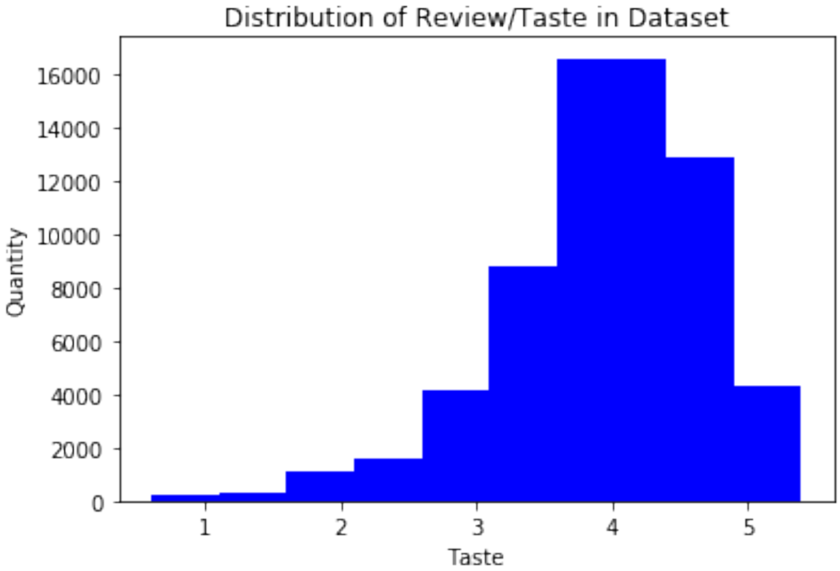
```
{1.5: 343, 3.0: 4137, 4.5: 12883, 3.5: 8797, 4.0: 16575, 2.0: 1099
, 5.0: 4331, 2.5: 1624, 1.0: 211}
```

In [108]:
```
SortedTasteValue = {key:TasteValue[key] for key in sorted(TasteValu
e.keys())}
```

In [109]:
```
print (SortedTasteValue)
```

```
{1.0: 211, 1.5: 343, 2.0: 1099, 2.5: 1624, 3.0: 4137, 3.5: 8797, 4
.0: 16575, 4.5: 12883, 5.0: 4331}
```

In [110]:
```
x = [elem for elem in SortedTasteValue.keys()]
y = [elem for elem in SortedTasteValue.values()]
plt.bar(x,y,color='blue')
plt.title('Distribution of Review/Taste in Dataset')
plt.xlabel('Taste')
plt.ylabel('Quantity')
plt.show()
```



In [111]:
```
# we need to construct the input matrix and output matrix
# unit[1] = 1 denotes the beer is Hefeweizen
def construct(data, input_x, output_y):
    unit_x = [1]
    # init theta 0
    for elem in data:
        if elem['beer/style'] == 'Hefeweizen':
            unit_x.append(1)
        else:
            unit_x.append(0)
        unit_x.append(elem['beer/ABV'])
        input_x.append(unit_x)
        output_y.append(elem['review/taste'])
        unit_x = [1]
```

In [112]:
```
input_x = []
output_y = []
construct(data, input_x, output_y)
```

In [113]:
```
theta,residuals,rank,s = np.linalg.lstsq(input_x, output_y, rcond=N
one)
```

In [114]:
```
print (theta)
```

```
[ 3.11795084 -0.05637406  0.10877902]
```

In [115]:
```
#split the data into two equal farctions
train_data = data[:int(len(data)/2)]
test_data = data[-int(len(data)/2):]
```

In [116]:
```
# train the model on train_data only
input_train = []
output_train = []
construct(train_data, input_train, output_train)
theta_train, residuals, rank, s = np.linalg.lstsq(input_train, outp
ut_train, rcond=None)
```

In [117]:
```
print (theta_train)
```

```
[ 2.99691466 -0.03573098  0.11672256]
```

In [118]:
```
# construct data for test dataset
input_test = []
output_test = []
construct(test_data, input_test, output_test)
```

In [119]:
```
# we have already got the model, now we need to calculate the MSE o
n training set
MSE_train = ((np.dot(np.array(input_train), np.array(theta_train).T
) - np.array(output_train))**2).mean()
MSE_test =  ((np.dot(np.array(input_test), np.array(theta_train).T)
- np.array(output_test))**2).mean()
```

In [87]:
```
# print (MSE_train-np.array(output_train))
```

```
[ 2.04479649  0.72059454  0.75561131 ... -0.23271644 -0.73271644
 -0.73271644]
```

In [120]:
```
# Here we output the MSE value for train data and test data
print ("MSE on train data: %f"%(MSE_train))
print ("MSE on test data: %f"%(MSE_test))
```

```
MSE on train data: 0.483968
MSE on test data: 0.423707
```

In [121]:
```
# We need to shuffle the data
random.shuffle(data)
```

In [122]:
```
# Then train model just as we did before
train_data_shuffled = data[:int(len(data)/2)]
test_data_shuffled = data[-int(len(data)/2):]
```

In [123]:
```
# input train feature
input_train_shuffled = []
output_train_shuffled = []
construct(train_data_shuffled, input_train_shuffled, output_train_s
huffled)
# input test feature
input_test_shuffled = []
output_test_shuffled = []
construct(test_data_shuffled, input_test_shuffled, output_test_shuf
fled)
```

In [124]:
```
# train model
theta_train_shuffled,residuals,rank,s = np.linalg.lstsq(input_train
_shuffled, output_train_shuffled, rcond=None)
```

In [126]:
```
print (theta_train_shuffled)
```

```
[ 3.11336703 -0.0449949   0.10947105]
```

In [127]:
```
# Here we again calculate MSE on train set and test set
MSE_train_shuffled = ((np.dot(np.array(input_train_shuffled), np.ar
ray(theta_train_shuffled).T) - np.array(output_train_shuffled))**2)
.mean()
MSE_test_shuffled =  ((np.dot(np.array(input_test_shuffled), np.arr
ay(theta_train_shuffled).T) - np.array(output_test_shuffled))**2).m
ean()
```

In [128]:
```
# Here we output the MSE value for train shuffled data and test shu
ffled data
print ("MSE on train shuffled data: %f"%(MSE_train_shuffled))
print ("MSE on test shuffled data: %f"%(MSE_test_shuffled))
```

```
MSE on train shuffled data: 0.448804
MSE on test shuffled data: 0.450521
```

In [129]:
```
def construct_newfeature(data, input_x, output_y):
    unit_x = [1]
    # init theta 0
    for elem in data:
        if elem['beer/style'] == 'Hefeweizen':
            unit_x.append(elem['beer/ABV'])
            unit_x.append(0)
        else:
            unit_x.append(0)
            unit_x.append(elem['beer/ABV'])
        input_x.append(unit_x)
        output_y.append(elem['review/taste'])
        unit_x = [1]
```

In [130]:
```
# reconstruct features using new method
# input train feature
new_input_train_shuffled = []
new_output_train_shuffled = []
construct_newfeature(train_data_shuffled, new_input_train_shuffled, new_output_train_shuffled)
# input test feature
new_input_test_shuffled = []
new_output_test_shuffled = []
construct_newfeature(test_data_shuffled, new_input_test_shuffled, new_output_test_shuffled)
```

In [131]:
```
# train new model
new_theta_train_shuffled,residuals,rank,s = np.linalg.lstsq(new_input_train_shuffled, new_output_train_shuffled, rcond=None)
```

In [132]:
```
# print the theta under this case
print (new_theta_train_shuffled)
```

```
[3.11370652 0.09945572 0.10943807]
```

In [133]:
```
# Here we again calculate MSE on train set and test set base on new feature
new_MSE_train_shuffled = ((np.dot(np.array(new_input_train_shuffled), np.array(new_theta_train_shuffled).T) - np.array(new_output_train_shuffled))**2).mean()
new_MSE_test_shuffled =  ((np.dot(np.array(new_input_test_shuffled), np.array(new_theta_train_shuffled).T) - np.array(new_output_test_shuffled))**2).mean()
```

In [134]:
```
# Here we output the MSE value for train shuffled data and test shuffled data
print ("MSE on train shuffled data: %f"%(new_MSE_train_shuffled))
print ("MSE on test shuffled data: %f"%(new_MSE_test_shuffled))
```

```
MSE on train shuffled data: 0.448795
MSE on test shuffled data: 0.450518
```

In [1]:
```
import urllib.request
import scipy.optimize
import random
import math
from sklearn import svm
```

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

In [6]:
```
print ("Reading data......")
data = list(parseData("http://jmcauley.ucsd.edu/cse255/data/beer/be
er_50000.json"))
print ("We are done")
```

```
Reading data......
We are done
```

In [5]:
```
random.shuffle(data)
train_data_shuffled = data[:int(len(data)/2)]
test_data_shuffled = data[-int(len(data)/2):]
```

In [7]:
```
def Construct_Feature(data, x_input, y_output):
    unit_x = []
    for elem in data:
        unit_x.append(elem['review/taste'])
        unit_x.append(elem['review/appearance'])
        unit_x.append(elem['review/aroma'])
        unit_x.append(elem['review/palate'])
        unit_x.append(elem['review/overall'])
        x_input.append(unit_x)
        if elem['beer/style'] == 'Hefeweizen':
            y_output.append(1)
        else:
            y_output.append(0)
        unit_x = []
```

In [8]:
```
input_x_train = []
output_y_train = []
intput_x_test = []
output_y_test = []
Construct_Feature(train_data_shuffled, input_x_train, output_y_trai
n)
Construct_Feature(test_data_shuffled, intput_x_test, output_y_test)
```

In [9]:
```
# train data using SVM model
clf = svm.SVC(C=1000, kernel='linear')
clf.fit(input_x_train, output_y_train)
```

Out[9]:
```
SVC(C=1000, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='l
inear',
  max_iter=-1, probability=False, random_state=None, shrinking=Tru
e,
  tol=0.001, verbose=False)
```

In [10]:
```
# make prediction
train_predictions = clf.predict(input_x_train)
test_predictions = clf.predict(intput_x_test)
```

In [11]:
```
# calculate accuracy
result_train = [train_predictions[i] == output_y_train[i] for i in
range(len(train_predictions))]
result_test = [test_predictions[j] == output_y_test[j] for j in ran
ge(len(test_predictions))]
acc_train = sum(result_train)/len(train_predictions)
acc_test = sum(result_test)/len(test_predictions)
```

In [12]:
```
# print acc
print ("The acc of train data is %f"%(acc_train))
print("The acc of test data is %f"%(acc_test))
```

```
The acc of train data is 0.988360
The acc of test data is 0.986920
```

In [13]:
```
# Ways to improve the SVM model
def Construct_New_Feature(data, x_input, y_output):
    unit_x = []
    for elem in data:
        unit_x.append(elem['review/taste'])
        unit_x.append(elem['review/appearance'])
        unit_x.append(elem['review/aroma'])
        unit_x.append(elem['review/palate'])
        unit_x.append(elem['review/overall'])
        if (("banana" in elem['review/text'].split(" ") and "wheat"
in elem['review/text'].split(" ")) or "banana" in elem['review/text
'].split(" ") or "wheat" in elem['review/text'].split(" ")):
            unit_x.append(1)
        else:
            unit_x.append(0)
        x_input.append(unit_x)
        if elem['beer/style'] == 'Hefeweizen':
            y_output.append(1)
        else:
            y_output.append(0)
        unit_x = []
```

In [14]:
```
input_x_train_new = []
output_y_train_new = []
intput_x_test_new = []
output_y_test_new = []
Construct_New_Feature(train_data_shuffled, input_x_train_new, outpu
t_y_train_new)
Construct_New_Feature(test_data_shuffled, intput_x_test_new, output
_y_test_new)
```

In [15]:
```
# clf_new = svm.SVC(C=1500, kernel='linear')
clf_new = svm.SVC(C=1000, kernel='rbf', gamma=1.0, decision_functio
n_shape='ovr')
clf_new.fit(input_x_train_new, output_y_train_new)
```

Out[15]:
```
SVC(C=1000, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=1.0, kernel='rbf'
,
  max_iter=-1, probability=False, random_state=None, shrinking=Tru
e,
  tol=0.001, verbose=False)
```

In [16]:
```
new_train_predictions = clf_new.predict(input_x_train_new)
new_test_predictions = clf_new.predict(intput_x_test_new)
```

In [17]:
```
result_train_new = [new_train_predictions[i] == output_y_train_new[
i] for i in range(len(new_train_predictions))]
result_test_new = [new_test_predictions[j] == output_y_test_new[j]
for j in range(len(new_test_predictions))]
acc_train_new = sum(result_train_new)/len(new_train_predictions)
acc_test_new = sum(result_test_new)/len(new_test_predictions)
```

In [18]:
```
# print acc
print ("The acc of train data is %f"%(acc_train_new))
print("The acc of test data is %f"%(acc_test_new))
```

```
The acc of train data is 0.991720
The acc of test data is 0.983360
```