In [3]:
```
import numpy
import urllib
import scipy.optimize
import random
import ast
```

In [4]:
```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')

from sklearn.metrics import mean_squared_error, accuracy_score
from math import sqrt
from sklearn import svm
from sklearn import preprocessing
```

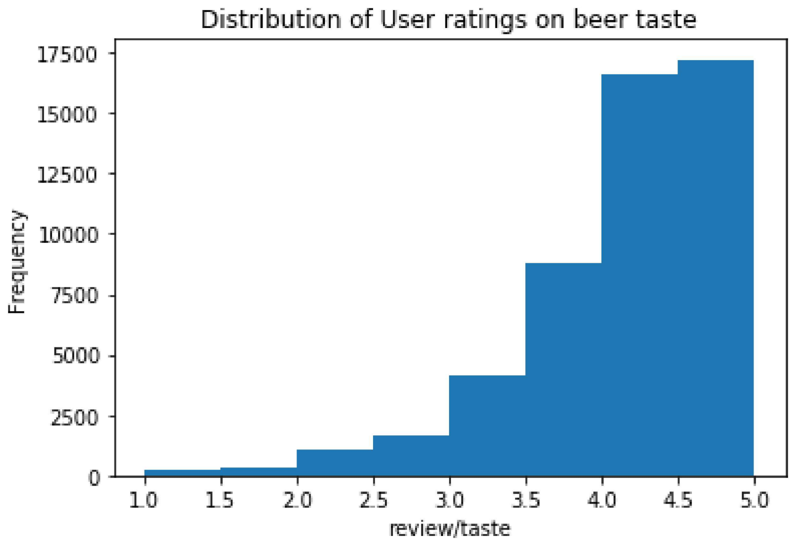
In [5]:
```
def parseData(fname):
  for l in urllib.urlopen(fname):
    yield eval(l)
```

In [6]:
```
def parseDatafromFile(fname):
  for l in open(fname):
    yield ast.literal_eval(l)
```

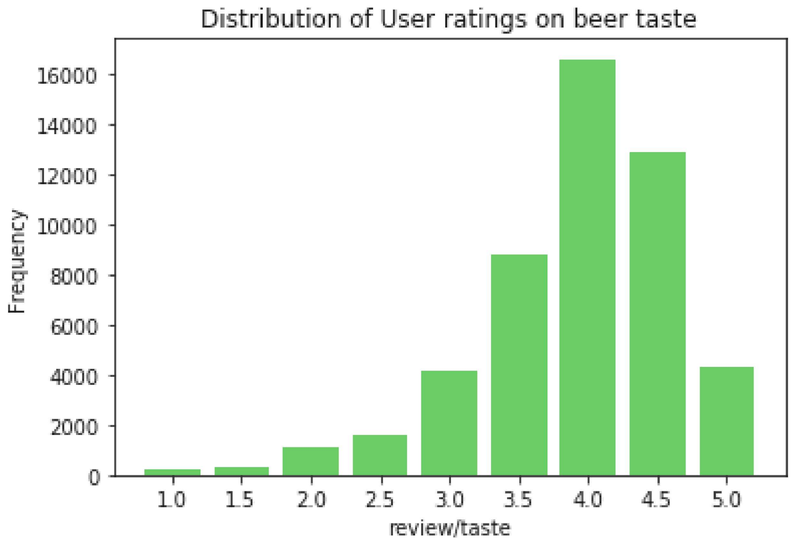
In [7]:
```
data = list(parseDatafromFile("C:/Users/ashak/Desktop/CSE258/beer_50000.json"
))
```

1) Distribution plot : 'review/taste'

```
In [9]: data_hist =[d['review/taste'] for d in data]
        plt.hist(data_hist,bins = 8)
        plt.xlabel('review/taste')
        plt.ylabel('Frequency')
        plt.title('Distribution of User ratings on beer taste');
```



```
In [68]: unique,counts = numpy.unique(numpy.array(data_hist), return_counts=True)
         plt.bar(unique,counts,width=0.4,color='g',align='center');
         plt.xlabel('review/taste')
         plt.ylabel('Frequency')
         plt.xticks(unique)
         plt.title('Distribution of User ratings on beer taste');
```



2) Simple linear predictor

In [10]:
```
def feature(datum):
    if datum['beer/style'] == 'Hefeweizen':
        isHefeweizen = 1;
    else:
        isHefeweizen = 0;
    feat =[1,isHefeweizen,datum['beer/ABV']]
    return feat
```

In [11]:
```
Y = data_hist
X = [feature(d) for d in data]
```

In [12]:
```
theta,residuals,rank,s = numpy.linalg.lstsq(X,Y)
theta
```

Out[12]:
```
array([ 3.11795084, -0.05637406,  0.10877902])
```

Answer

In this problem we are trying to predict the review/taste based on two features

- If the beer reviewed is a hefeweizen style beer
- Its ABV value of the beer reviewed
- The feature vector which consists of these two features and 1 to get the optimum regression coefficients and the intercept respectively. The theta returned by the lstsq contains the parameters for which the squared error was the least.
- The values of the coefficients are listed below
  - Intercept - 3.11795084 - Value of the dep var review/taste when other two features are 0. It gives the offset
  - Coefficient of isHefeweizen - -0.05637406 - is negatively correlated with the target variable. the coefficient is much smaller and the variable value can either be 1. So the rating decreases by 0.05 when the beer is a hefeweizen.
  - Coefficient of the ABV - 0.10877902 - is positively correlated with the target variable. The review increases by the coefficient for every unit increase in the ABV value. The rating in this model seems to increase with the increase in ABV and is a much stronger variable than ishefeweizen as it is more generic.

3) Train and test model by splitting data - Split fraction = 0.5, Shuffle = False

In [13]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.5,  shuf
fle=False)
```

```
In [14]: theta_q3,residuals_q3,rank_q3,s_q3 = numpy.linalg.lstsq(X_train,Y_train)
         print(" MSE is " + str(residuals_q3/len(X_train)))
         theta_q3

          MSE is [0.48396806]

Out[14]: array([ 2.99691466, -0.03573098,  0.11672256])
```

```
In [15]: Y_test_predicted = numpy.dot(X_test,theta_q3)
         mse_q3 = mean_squared_error(Y_test, Y_test_predicted)
         print('MSE on test set is: ' + str(mse_q3))

         MSE on test set is: 0.4237065211985383
```

Answer:

The model trained with a non shuffled split from the data has a

- Training MSE : 0.48396806
- Testing MSE : 0.42370652
- #### Note:
  - It can be obsereved that Training error is larger than the testing error and this is not typical of a good model. This does not give any intuition about how the model could perform for further unknown data. The goodness of fit cannot be stated or inferred from this result.
  - The result is counter intuitive to how machine learning works. The essence of ML is to predict the unknown. If you are better at predicting the unknown than what you have 'learned', My answer is that the data between training and validation must be different in some way.
  - This effect or observation could be result of High bias which can cause an algorithm to miss the relevant/more appropriate relations between features and target outputs. This could mean you either need to reevaluate your data splitting method, adding more data. etc.,
  - In this case since it is a straight forward 50-50 split of the data without shuffling, so the training data is not more representative of the entire dataset. This misinterpretations of relationships between variables and uderfitting could be a more reasonable explanation for why this is happening.

4) Train and test model by splitting data - Split fraction = 0.5, Shuffle = True

```
In [16]: X_trainS, X_testS, Y_trainS, Y_testS  = train_test_split(X, Y, test_size=0.5,
          shuffle=True)
```

```
In [17]: theta_q4,residuals_q4,rank_q4,s_q4 = numpy.linalg.lstsq(X_trainS,Y_trainS)
         print(" MSE is " + str(residuals_q4/len(X_trainS)))
         theta_q4

          MSE is [0.43727544]

Out[17]: array([ 3.11744403, -0.02561108,  0.10906673])
```

In [18]:
```
Y_testS_predicted = numpy.dot(X_testS,theta_q4)
mse_q4 = mean_squared_error(Y_testS, Y_testS_predicted)
print('MSE on test set is: ' + str(mse_q4))
```

```
MSE on test set is: 0.46207215413914166
```

Answer

- The random splitting seems to give an expected output of MSE for training and the test data. We expect our model to perform well on the known data than the unknown data and thats what it does. The data seems to have understood that the isHefeweizen does affect the rating to an extent as its negative effect on the rating has reduced. This is due to random sampling the distribution is more generic as it incorporates the randomness in the data so that it could perform better on the test set.

5) ABV with binary categorical Variable

In [19]:
```
def feature_q5(datum):
    if datum['beer/style'] == 'Hefeweizen':
        isHefeweizen = datum['beer/ABV'];
        isnotHefeweizen = 0
    else:
        isHefeweizen = 0;
        isnotHefeweizen = datum['beer/ABV']
    feat =[1,isHefeweizen,isnotHefeweizen]
    return feat
```

In [20]:
```
x = [feature_q5(d) for d in data]
```

In [21]:
```
X_trainq5, X_testq5, Y_trainq5, Y_testq5  = train_test_split(x, Y, test_size=
0.5,  shuffle= True)
theta_q5,residuals_q5,rank_q5,s_q5 = numpy.linalg.lstsq(X_trainq5,Y_trainq5)
print(" MSE is " + str(residuals_q5/len(X_trainq5)))
theta_q5
```

```
 MSE is [0.44406646]
```

Out[21]:
```
array([3.09450376, 0.0951324 , 0.11153861])
```

In [22]:
```
Y_testq5_predicted = numpy.dot(X_testq5,theta_q5)
mse_q5 = mean_squared_error(Y_testq5, Y_testq5_predicted)
print('MSE on test set is: ' + str(mse_q5))
```

```
MSE on test set is: 0.45537091073532426
```

6) Model comparison and explanation

Answer

- This method of engineering the same features used in the previous problem seems to perform better than the previous method because, the value of the variable isHefeweizen was only 1 or 0 whose coefficient was relatively smaller than the ABV. The feature engineering done in problem 6 gives a higher weightage of ABV values of one type and 0 to all the others and vice versa for the second feature. This way the model could find significant differences in the ABV values between the types and within a particular type itself. The way the features are engineered to bring out the information in the dataset plays a very important role in developing a better models for prediction.

7) Classification

In [23]:
```
def feature_q7(datum):
    feat =[datum['review/taste'], datum['review/appearance'], datum['review/ar
oma'], datum['review/palate'], datum['review/overall']]
    return feat
```

In [24]:
```
X_q7 = [feature_q7(d) for d in data]
X_q7[:10]
```

Out[24]:
```
[[1.5, 2.5, 2.0, 1.5, 1.5],
 [3.0, 3.0, 2.5, 3.0, 3.0],
 [3.0, 3.0, 2.5, 3.0, 3.0],
 [3.0, 3.5, 3.0, 2.5, 3.0],
 [4.5, 4.0, 4.5, 4.0, 4.0],
 [3.5, 3.5, 3.5, 3.0, 3.0],
 [4.0, 3.5, 3.5, 4.0, 3.5],
 [3.5, 3.5, 2.5, 2.0, 3.0],
 [4.0, 3.5, 3.0, 3.5, 4.0],
 [4.0, 5.0, 3.5, 4.0, 4.5]]
```

In [11]:
```
Y_q7 = [1 if d['beer/style'] == 'Hefeweizen' else 0 for d in data]
```

In [26]:
```
X_trainq7, X_testq7, Y_trainq7, Y_testq7  = train_test_split(X_q7, Y_q7, test_
size=0.5,  shuffle= True)
```

In [27]:
```
clf = svm.SVC(C=1000, kernel='linear')
clf.fit(X_trainq7, Y_trainq7)

train_predictions = clf.predict(X_trainq7)
test_predictions = clf.predict(X_testq7)
```

Training Error Q7:

```
In [28]: accuracy_score(Y_trainq7, train_predictions)

Out[28]: 0.9878
```

Testing Error Q7:

```
In [29]: accuracy_score(Y_testq7, test_predictions)

Out[29]: 0.98748
```

Trying to build out a better model:

- I tried out a bunch of ways to build a better model producing high accuracy.
  - Added the ABV feature as I observed in the regression questions that it could be related to the style of the beer.
  - However, the accuracy was not much different and the difference is extremely small to decide which model is better. And most of the times it performed worse than the previous model.

```
In [1]: import pandas as pd
        def feature_q8(datum):
            feat =[datum['review/taste'], datum['review/appearance'], datum['review/ar
        oma'], datum['review/palate'], datum['review/overall'],datum['beer/ABV']]
            return feat
```

```
In [12]: X_q8 = [feature_q8(d) for d in data]
         x = pd.DataFrame(X_q8)
         x['Label'] = Y_q7
         X_trainq8, X_testq8, Y_trainq8, Y_testq8  = train_test_split(X_q8, Y_q7, test_
         size=0.5,  shuffle= True)
```

```
In [17]: %%time
         clf = svm.SVC(C=1000, kernel='sigmoid')
         clf.fit(X_trainq8, Y_trainq8)

         train_predictions8 = clf.predict(X_trainq8)
         print(accuracy_score(Y_trainq8, train_predictions8))
         test_predictions8 = clf.predict(X_testq8)
         print(accuracy_score(Y_testq8, test_predictions8))

         0.97764
         0.97832
         Wall time: 2.18 s
```

```
In [18]: def label_race(row):
             if row == 'Hefeweizen':
                 return 10;
             elif row =='Hefe-Weizen' :
                 return 10;
             else:
                 return 0;
```

In [19]:
```
def feature_q8_1(datum):
    feat =[datum['review/taste'], datum['review/appearance'], datum['review/ar
oma'], datum['review/palate'],datum['review/overall'],datum['beer/ABV'],datum[
'beer/name']]
    return feat
X_q8_1 = [feature_q8_1(d) for d in data]
x = pd.DataFrame(X_q8_1)
x['Label'] = Y_q7
x[6].unique()
x['name_feat'] = x[6].apply (lambda row: label_race(row))
#print(x.loc[x['Label']==1])#.groupby([6]).size().reset_index(name='Count').pl
ot(kind ='bar')
x_feat = x[[0,1,2,3,4,5,'name_feat']]
```

In [21]:
```
X_trainq81, X_testq81, Y_trainq81, Y_testq81  = train_test_split(x_feat, Y_q7,
 test_size=0.5,  shuffle= True)
clf = svm.SVC(C=1000, kernel='linear')
clf.fit(X_trainq81, Y_trainq81)

train_predictions81 = clf.predict(X_trainq81)
print(accuracy_score(Y_trainq81, train_predictions81))
test_predictions81 = clf.predict(X_testq81)
print(accuracy_score(Y_testq81, test_predictions81))
```

```
0.99068
0.99028
```

In [22]:
```
def feature_q8_2(datum):

    feat =[datum['review/taste'], datum['review/appearance'], datum['review/ar
oma'], datum['review/palate'],datum['review/overall'],datum['beer/ABV'],datum[
'beer/name']]
    return feat
X_q8_2 = [feature_q8_2(d) for d in data]
x2 = pd.DataFrame(X_q8_2)
```

In [23]:
```
X_trainq82, X_testq82, Y_trainq82, Y_testq82  = train_test_split(X_q8_2, Y_q7,
 test_size=0.5,  shuffle= True)
```

In [25]:
```
x2t = pd.DataFrame(X_trainq82)
x2t['name_feat'] = x2t[6].apply (lambda row: label_race(row))
x2t_feat = x2t[[0,1,2,3,4,5,'name_feat']]
x2tst = pd.DataFrame(X_testq82)
x2tst['name_feat'] = x2tst[6].apply (lambda row: label_race(row))
x2tst_feat = x2tst[[0,1,2,3,4,5,'name_feat']]
```

Using beer/name:

- Looked for the name Hefeweizen in beer/name and created a new feature with a high value for all beers with name Hefeweizen or hefe-weizen and 0 for the beers reviewed that does not have those two names. The classifier seems to perform a little better than the model built in Q7.
- Training accuracy : 99.04%
- Testing accuracy : 99.056%

In [26]:
```
clf = svm.SVC(C=1000, kernel='linear')
clf.fit(x2t_feat, Y_trainq82)

train_predictions82 = clf.predict(x2t_feat)
print(accuracy_score(Y_trainq82, train_predictions82))
test_predictions82 = clf.predict(x2tst_feat)
print(accuracy_score(Y_testq82, test_predictions82))
```

```
0.9904
0.99056
```

- Also tried a bunch of different methods like scaling(normalising), finding the correlation between the label and the features and trying to give it a higher value them or scaling them.
- However, did not see any improvement in the accuracy of the model.

In [27]:
```
X_scaler = preprocessing.MaxAbsScaler()
X_scaled = X_scaler.fit_transform(X_q8)
X_trainq8s, X_testq8s, Y_trainq8s, Y_testq8s  = train_test_split(X_scaled, Y_q
7, test_size=0.5,  shuffle= True)
```

In [28]:
```
import pandas as pd
df = pd.DataFrame(X_q8)
df['Y'] = Y_q7
df.corr()
```

Out[28]:

	0	1	2	3	4	5	Y
0	1.000000	0.533859	0.712236	0.730433	0.784999	0.352663	-0.044823
1	0.533859	1.000000	0.539011	0.557103	0.500057	0.287801	-0.047023
2	0.712236	0.539011	1.000000	0.611383	0.615361	0.386149	-0.039543
3	0.730433	0.557103	0.611383	1.000000	0.695278	0.350955	-0.036527
4	0.784999	0.500057	0.615361	0.695278	1.000000	0.197399	-0.016567
5	0.352663	0.287801	0.386149	0.350955	0.197399	1.000000	-0.102713
Y	-0.044823	-0.047023	-0.039543	-0.036527	-0.016567	-0.102713	1.000000

In [29]:
```
clf = svm.SVC(C=1000, kernel='linear')
```

In [32]:
```
%%time
clf.fit(X_trainq8s,Y_trainq8s)
train_predictions8s = clf.predict(X_trainq8s)
print(accuracy_score(Y_trainq8s, train_predictions8s))
#test_predictions8s = clf.predict(X_testq8s)
#print(accuracy_score(Y_testq8s, test_predictions8s))
```

```
0.9878
Wall time: 2min 28s
```

In [33]:
```
test_predictions8s = clf.predict(X_testq8s)
print(accuracy_score(Y_testq8s, test_predictions8s))
```

```
0.9874
```

In [66]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

feat_text = [d['review/text'].replace('\t', '') for d in data]
```

In [38]:
```
tfidf = TfidfVectorizer(sublinear_tf=True, min_df=5, norm='l2', encoding='lati
n-1', ngram_range=(1, 2), stop_words='english')
features = tfidf.fit_transform(feat_text)
labels = Y_q7
```

In [41]:
```
features
```

Out[41]:
```
<50000x103335 sparse matrix of type '<class 'numpy.float64'>'
        with 4793043 stored elements in Compressed Sparse Row format>
```

In [65]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.pipeline import Pipeline
import seaborn as sns
```

In [53]:
```
result_cols = ["Classifier", "Accuracy"]
result_frame = pd.DataFrame(columns=result_cols)

x_traintxt,x_testtxt,y_traintxt,y_testtxt = train_test_split(feat_text,Y_q7,te
st_size = 0.5, shuffle = False)
# Any results you write to the current directory are saved as output.

classifiers = [svm.SVC(kernel="linear", C=1000)]


for clf in classifiers:
    name = clf.__class__.__name__
    text_clf = Pipeline([('vect', CountVectorizer()), ('tfidf', TfidfTransform
er()),
                  ('clf', clf),])
    text_clf.fit(x_traintxt, y_traintxt)
    trainpred = text_clf.predict(x_traintxt)
    predicted = text_clf.predict(x_testtxt)
    acc = accuracy_score(y_testtxt,predicted)
    print (name+' accuracy = '+str(acc*100)+'%')
    acc_field = pd.DataFrame([[name, acc*100]], columns=result_cols)
    result_frame = result_frame.append(acc_field)
result_frame

"""sns.set_color_codes("muted")
sns.barplot(x='Accuracy', y='Classifier', data=result_frame, color="r")

plt.xlabel('Accuracy %')
plt.title('Classifier Accuracy')
plt.show()"""
```

```
SVC accuracy = 99.336%
```

Out[53]:
```
'sns.set_color_codes("muted")\nsns.barplot(x=\'Accuracy\', y=\'Classifier\',
data=result_frame, color="r")\n\nplt.xlabel(\'Accuracy %\')\nplt.title(\'Clas
sifier Accuracy\')\nplt.show()'
```

In [64]:
```
result_frame
```

Out[64]:

	Classifier	Accuracy
0	SVC	99.336

In [61]:
```
pred = list(trainpred) + list(predicted)
```

In [62]:
```
df_txt = pd.DataFrame(X_q8)
df_txt['text_pred'] = pred
df_txt.head()
```

Out[62]:

	0	1	2	3	4	5	text_pred
0	1.5	2.5	2.0	1.5	1.5	5.0	1
1	3.0	3.0	2.5	3.0	3.0	6.2	0
2	3.0	3.0	2.5	3.0	3.0	6.5	0
3	3.0	3.5	3.0	2.5	3.0	5.0	0
4	4.5	4.0	4.5	4.0	4.0	7.7	0

Classification using review/text

- I used tfidf to get the text features and classified the text features using a linear SVM Classifier.
- Trained using 50% of the data and tested using 50% data. Used the predicted classes for every review as an feature in addition to the already existing features. This classifier seems to work better than the SVM classifier with just the ratings.
- Train error : 99.704%
- Test error : 99.632%

In [63]:
```
%%time
X_trainq8txt, X_testq8txt, Y_trainq8txt, Y_testq8txt  = train_test_split(df_tx
t, Y_q7, test_size=0.5,  shuffle= True)

clf = svm.SVC(C=1000, kernel='linear')
clf.fit(X_trainq8txt, Y_trainq8txt)

train_predictions8txt = clf.predict(X_trainq8txt)
print(accuracy_score(Y_trainq8txt, train_predictions8txt))
test_predictions8txt = clf.predict(X_testq8txt)
print(accuracy_score(Y_testq8txt, test_predictions8txt))
```

```
0.99704
0.99632
Wall time: 3min 41s
```

In [69]:
```
result_cols = ["Classifier", "Accuracy"]
result_frame = pd.DataFrame(columns=result_cols)

x_traintxt,x_testtxt,y_traintxt,y_testtxt = train_test_split(feat_text,Y_q7,te
st_size = 0.5, shuffle = True)
# Any results you write to the current directory are saved as output.

classifiers = [svm.SVC(kernel="linear", C=1000)]


for clf in classifiers:
    name = clf.__class__.__name__
    text_clf = Pipeline([('vect', CountVectorizer()), ('tfidf', TfidfTransform
er()),('clf', clf),])
    text_clf.fit(x_traintxt, y_traintxt)
    #trainpred = text_clf.predict(x_traintxt)
    predicted = text_clf.predict(feat_text)
    acc = accuracy_score(Y_q7,predicted)
    print (name+' accuracy = '+str(acc*100)+'%')
    acc_field = pd.DataFrame([[name, acc*100]], columns=result_cols)
    result_frame = result_frame.append(acc_field)
result_frame
```

```
SVC accuracy = 0.0%
```

Out[69]:

	Classifier	Accuracy
0	SVC	0.0

In [70]:
```
accuracy_score(Y_q7,predicted)
```

Out[70]:
```
0.99696
```

In [73]:
```
df_txt1 = pd.DataFrame(X_q8)
df_txt1['text_pred'] = predicted
df_txt1.head()
```

Out[73]:

	0	1	2	3	4	5	text_pred
0	1.5	2.5	2.0	1.5	1.5	5.0	1
1	3.0	3.0	2.5	3.0	3.0	6.2	0
2	3.0	3.0	2.5	3.0	3.0	6.5	0
3	3.0	3.5	3.0	2.5	3.0	5.0	0
4	4.5	4.0	4.5	4.0	4.0	7.7	0

In [74]:
```
%%time
X_trainq8txt, X_testq8txt, Y_trainq8txt, Y_testq8txt  = train_test_split(df_tx
t1, Y_q7, test_size=0.5,  shuffle= True)

clf = svm.SVC(C=1000, kernel='linear')
clf.fit(X_trainq8txt, Y_trainq8txt)

train_predictions8txt = clf.predict(X_trainq8txt)
print(accuracy_score(Y_trainq8txt, train_predictions8txt))
test_predictions8txt = clf.predict(X_testq8txt)
print(accuracy_score(Y_testq8txt, test_predictions8txt))
```

```
0.99684
0.99708
Wall time: 5min 17s
```

- This was trained using the text classified with the randomly shuffled dataset. Used the predicted values and appended to the existing features to finally predict the classes.