Predict Rating

December 26, 2020

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import string
from collections import defaultdict
import gzip
import json
from nltk.corpus import stopwords
import math
from sklearn import linear_model
from scipy.sparse import csr_matrix
```

[2]:
```
def readTXT(path):
    file = []
    d = defaultdict(type(""))
    with gzip.open(path,'rb') as f:
        for l in f:
            l = l.decode("latin-1")
            file.append(l)

    return file
```

[2]:
```
def readJSON(path):
    file = []
    null = None
    for l in gzip.open(path, 'rt'):
        d=eval(l)
        file.append(d)
    return file
```

1 PERCENTAGE MODEL

[3]:
```
df = readTXT("beeradvocate.txt.gz")
```

[4]:
```
def parse(df):
    file = []
    d = {}
    count = 0
    for l in df:
        l = l.strip()
        if(count == 100000): break
        i = l.split(":")

        if len(i) == 1:
            count += 1
            file.append(d)
            d = {}
            continue
        d[i[0]] = i[1]
    return file
```

[5]:
```
df = parse(df)
```

[6]:
```
df = pd.DataFrame(df)
```

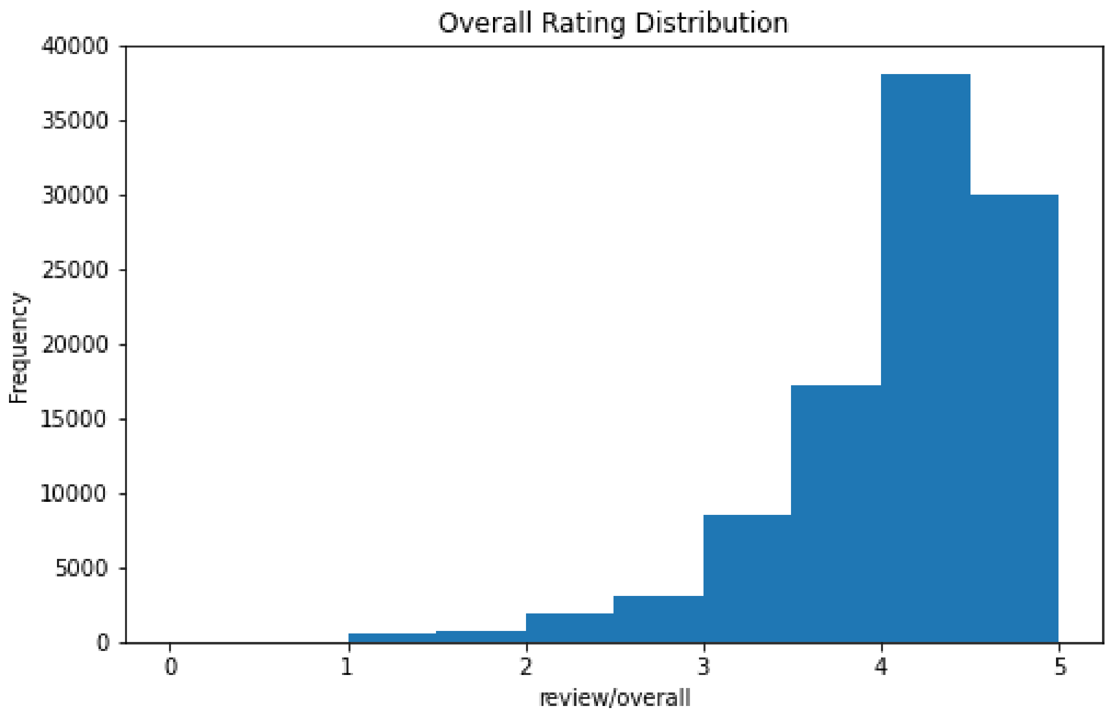
[7]:
```
df
```

[7]:
```
                          beer/name beer/beerId beer/brewerId beer/ABV  \
0                      Sausa Weizen       47986         10325     5.00
1                          Red Moon       48213         10325     6.20
2            Black Horse Black Beer       48215         10325     6.50
3                        Sausa Pils       47969         10325     5.00
4                     Cauldron DIPA       64883          1075     7.70
…                               …           …             …        …
99995                 La Madragore       30968          2958     8.00
99996               CuvÃ©e Du 9Ã¨me       39651          2958     8.00
99997               CuvÃ©e Du 9Ã¨me       39651          2958     8.00
99998                  La Dragonne       30972          2958     7.50
99999                  La Dragonne       30972          2958     7.50

                              beer/style review/appearance review/aroma  \
0                             Hefeweizen               2.5            2
1                     English Strong Ale                 3          2.5
2                 Foreign / Export Stout                 3          2.5
3                        German Pilsener               3.5            3
4        American Double / Imperial IPA                  4          4.5
…                                      …                 …            …
99995             Belgian Strong Dark Ale                4            4
99996                        Belgian IPA                 4            4
99997                        Belgian IPA                 4            4
99998                Herbed / Spiced Beer                3          2.5
```

```
99999            Herbed / Spiced Beer              3          2.5

      review/palate review/taste review/overall  review/time  \
0               1.5          1.5             1.5   1234817823
1                 3            3               3   1235915097
2                 3            3               3   1235916604
3               2.5            3               3   1234725145
4                 4          4.5               4   1293735206
…               …           …             …           …
99995             4            4             3.5   1149722515
99996           3.5            4             3.5   1201206794
99997             4            4             4.5   1195558181
99998           2.5            1               1   1298300394
99999           2.5            2             1.5   1296028407

      review/profileName                                        review/text
0                stcules   A lot of foam. But a lot.\tIn the smell some …
1                stcules   Dark red color, light beige foam, average.\tI…
2                stcules   Almost totally black. Beige foam, quite compa…
3                stcules   Golden yellow color. White, compact foam, qui…
4         johnmichaelsen   According to the website, the style for the C…
…                    …                                                  …
99995           ggaughan   Sampled at the Monk's Cafe BFM tasting. The b…
99996           Phyl21ca                                             Bottle
99997                Bov                                   Jurassian Pale Ale
99998            chefelf   I really, really wanted to like this beer. Th…
99999           corby112                                         From notes

[100000 rows x 13 columns]
```

[8]:
```
df['review/overall'] = df['review/overall'].apply(pd.to_numeric)
df['review/appearance'] = df['review/appearance'].apply(pd.to_numeric)
df['review/aroma'] = df['review/aroma'].apply(pd.to_numeric)
df['review/palate'] = df['review/palate'].apply(pd.to_numeric)
df['review/taste'] = df['review/taste'].apply(pd.to_numeric)
df['beer/ABV'] = df['beer/ABV'].apply(pd.to_numeric)
```

[242]:
```
df['review/overall'].plot(kind='hist')
plt.xlabel('review/overall')
plt.title('Overall Rating Distribution')
plt.savefig("distribution.png")
```



[10]:
```
median = df['review/overall'].median()
print(median)
```

```
4.0
```

[11]:
```
mean = df['review/overall'].mean()
print(mean)
```

```
3.89702
```

[12]:
```
punctuation = set(string.punctuation)
def standardize(string):
    string = string.lower().replace('\t', ' ')
    r = ''.join([c for c in string if not c in punctuation])
    return r
```

[13]:
```
df['review/text'] = df['review/text'].apply(standardize)
```

[14]:
```
df
```

[14]:
```
                  beer/name beer/beerId beer/brewerId  beer/ABV  \
0              Sausa Weizen       47986         10325       5.0
1                  Red Moon       48213         10325       6.2
2    Black Horse Black Beer       48215         10325       6.5
```

```
3                Sausa Pils       47969         10325       5.0
4             Cauldron DIPA       64883          1075       7.7
…                       …         …           …         …
99995          La Madragore       30968          2958       8.0
99996       CuvÃ©e Du 9Ã¨me       39651          2958       8.0
99997       CuvÃ©e Du 9Ã¨me       39651          2958       8.0
99998           La Dragonne       30972          2958       7.5
99999           La Dragonne       30972          2958       7.5

                             beer/style  review/appearance  review/aroma  \
0                            Hefeweizen                2.5           2.0
1                    English Strong Ale                3.0           2.5
2                Foreign / Export Stout                3.0           2.5
3                       German Pilsener                3.5           3.0
4        American Double / Imperial IPA                4.0           4.5
…                                   …                …             …
99995           Belgian Strong Dark Ale                4.0           4.0
99996                       Belgian IPA                4.0           4.0
99997                       Belgian IPA                4.0           4.0
99998             Herbed / Spiced Beer                 3.0           2.5
99999             Herbed / Spiced Beer                 3.0           2.5

       review/palate  review/taste  review/overall  review/time  \
0                1.5           1.5             1.5   1234817823
1                3.0           3.0             3.0   1235915097
2                3.0           3.0             3.0   1235916604
3                2.5           3.0             3.0   1234725145
4                4.0           4.5             4.0   1293735206
…                  …             …               …          …
99995            4.0           4.0             3.5   1149722515
99996            3.5           4.0             3.5   1201206794
99997            4.0           4.0             4.5   1195558181
99998            2.5           1.0             1.0   1298300394
99999            2.5           2.0             1.5   1296028407

      review/profileName                                        review/text
0                stcules   a lot of foam but a lot in the smell some ban…
1                stcules   dark red color light beige foam average in th…
2                stcules   almost totally black beige foam quite compact…
3                stcules   golden yellow color white compact foam quite …
4         johnmichaelsen   according to the website the style for the ca…
…                    …                                                  …
99995           ggaughan   sampled at the monks cafe bfm tasting the bee…
99996           Phyl21ca                                             bottle
99997                Bov                                  jurassian pale ale
99998            chefelf   i really really wanted to like this beer the …
99999           corby112                                         from notes
```

[100000 rows x 13 columns]

[16]:
```
def findStat(s):
    print("Mean: ",
    df[df['review/text'].str.contains(s)]['review/overall'].mean())
    print("Median: ",
    df[df['review/text'].str.contains(s)]['review/overall'].median())
```

[17]:
```
findStat("bad")
```

```
Mean:  3.5239204721963344
Median:  3.5
```

[18]:
```
findStat("fine")
```

```
Mean:  4.006820877817319
Median:  4.0
```

[19]:
```
findStat("ok")
```

```
Mean:  3.8610464595828486
Median:  4.0
```

[20]:
```
findStat("beer")
```

```
Mean:  3.9029024060986264
Median:  4.0
```

[21]:
```
findStat("awful")
```

```
Mean:  2.634228187919463
Median:  2.5
```

[22]:
```
findStat("amazing")
```

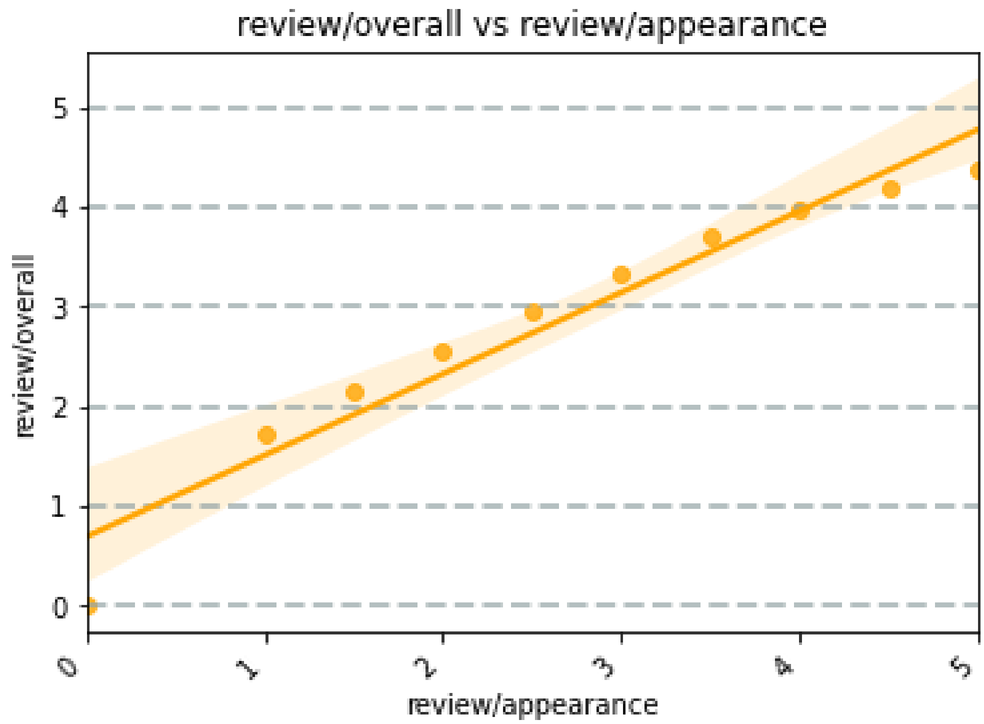
```
Mean:  4.300974512743628
Median:  4.5
```

[ ]:
```

```

[220]:
```
def bar_plot(df, s, types, c, xlabel):
    means = []
    for t in types:
        means.append(df[df[s] == t]['review/overall'].mean())
    plt.bar(xlabel, means, color=c, alpha=0.7)
    plt.grid(color='#95a5a6', linestyle='--', linewidth=2, axis='y', alpha=0.7)
    plt.xlabel(s)
```

```
    plt.ylabel("review/overall")
    plt.title("review/overall vs "+s)
    plt.rcParams["figure.figsize"] = (8, 5)
    plt.setp(plt.gca().get_xticklabels(), rotation=45,␣
 ↪horizontalalignment='right')
    plt.show()

def scatter_plot(df, s, types, c, xlabel):
    means = []
    for t in types:
        means.append(df[df[s] == t]['review/overall'].mean())
#     plt.scatter(xlabel, means, color=c, alpha=0.7)
#     plt.plot(xlabel, means, color=c)
    data = pd.DataFrame({'x':xlabel, 'y':means})
    sns.regplot(x='x', y='y', data=data, color=c)
    plt.grid(color='#95a5a6', linestyle='--', linewidth=2, axis='y', alpha=0.7)
    plt.xlabel(s)
    plt.ylabel("review/overall")
    plt.title("review/overall vs "+s)
    plt.rcParams["figure.figsize"] = (8, 5)
    plt.setp(plt.gca().get_xticklabels(), rotation=45,␣
 ↪horizontalalignment='right')
    name = s.replace("/", '')
    plt.savefig(name+".png")
    plt.show()


def scatter_noline(df, s, types, c, xlabel):
    means = []
    for t in types:
        means.append(df[df[s] == t]['review/overall'].mean())
    plt.scatter(xlabel, means, color=c, alpha=0.7)
    plt.grid(color='#95a5a6', linestyle='--', linewidth=2, axis='y', alpha=0.7)
    plt.xlabel(s)
    plt.ylabel("review/overall")
    plt.title("review/overall vs "+s)
    plt.rcParams["figure.figsize"] = (8, 5)
    plt.setp(plt.gca().get_xticklabels(), rotation=45,␣
 ↪horizontalalignment='right')
    plt.show()
```

[203]:
```
appearance = df['review/appearance'].unique()
scatter_plot(df, 'review/appearance', appearance, "orange", appearance)
```



[216]:
```
def savePlot(string, color, b):
    appearance = df[string].unique()
    if b:
        scatter_plot(df, string, appearance, color, appearance)
    else:
        scatter_plot(df, string, appearance, color, range(len(appearance)))
```

[ ]:
```
def savePlot2(string, color, b):
    appearance = df[string].unique()
    if b:
        scatter_plot(df, string, appearance, color, appearance)
    else:
        scatter_plot(df, string, appearance, color, range(len(appearance)))
```

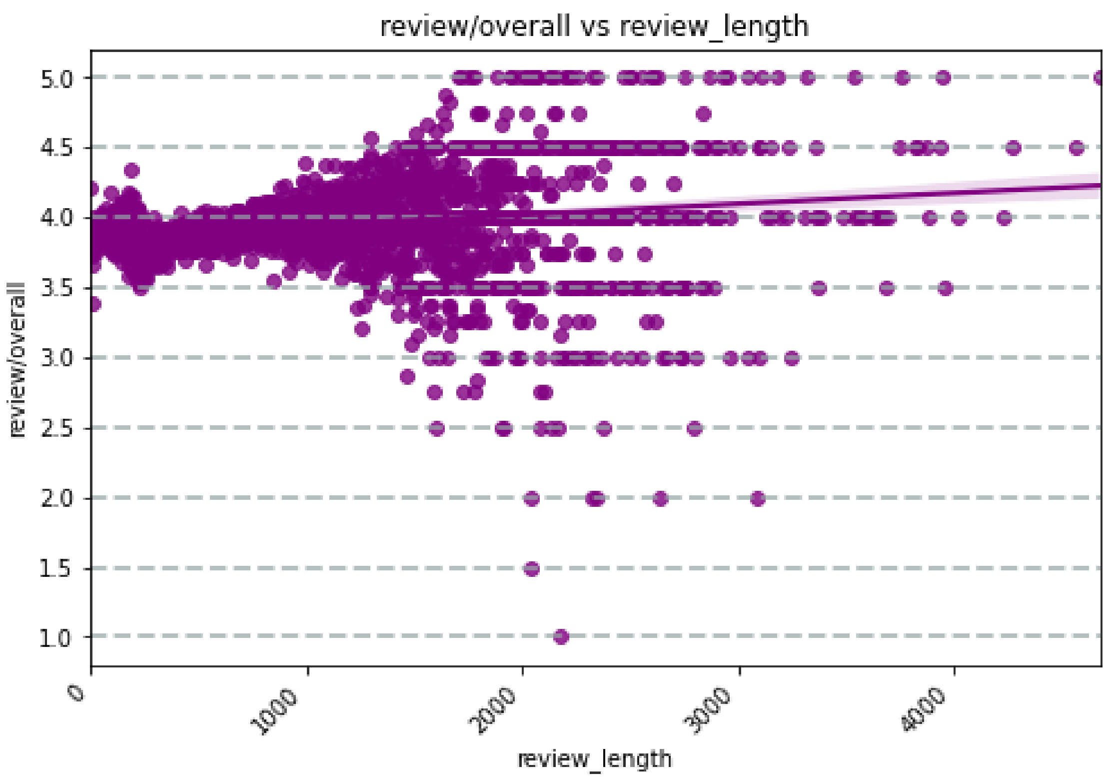
[232]:
```
df['review_length'] = df['review/text'].str.len()
```

[245]:
```
new = df[['review/overall', 'review/appearance', 'review/palate', 'review/
 ↪taste', 'review/aroma', 'beer/ABV', 'review_length']].copy()
```

[246]:
```
new.corr()
```

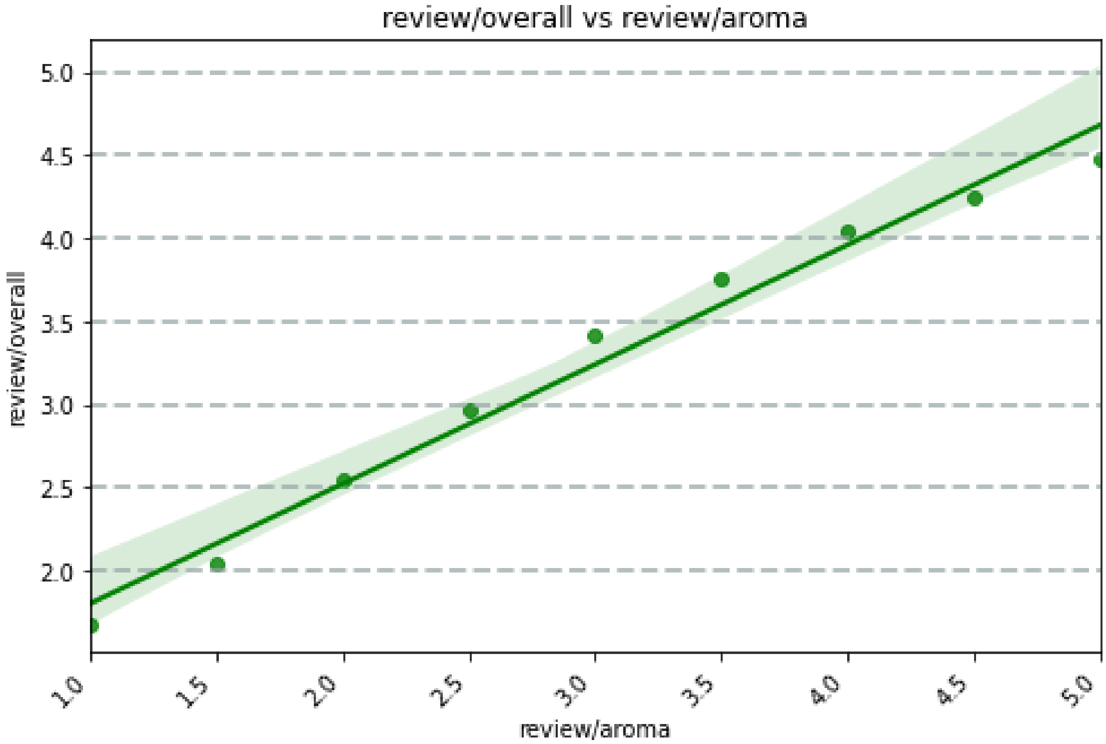
[246]:
```
                    review/overall  review/appearance  review/palate  \
review/overall            1.000000           0.489678       0.686402
review/appearance         0.489678           1.000000       0.549310
review/palate             0.686402           0.549310       1.000000
review/taste              0.775298           0.526898       0.716624
review/aroma              0.596313           0.528390       0.592861
beer/ABV                  0.131734           0.240494       0.287835
review_length             0.051897           0.081490       0.077584

                    review/taste  review/aroma  beer/ABV  review_length
review/overall          0.775298      0.596313  0.131734       0.051897
review/appearance       0.526898      0.528390  0.240494       0.081490
review/palate           0.716624      0.592861  0.287835       0.077584
review/taste            1.000000      0.699170  0.296684       0.070301
review/aroma            0.699170      1.000000  0.348777       0.090255
beer/ABV                0.296684      0.348777  1.000000       0.120151
review_length           0.070301      0.090255  0.120151       1.000000
```

[233]:
```
savePlot('review_length', 'purple', True)
```



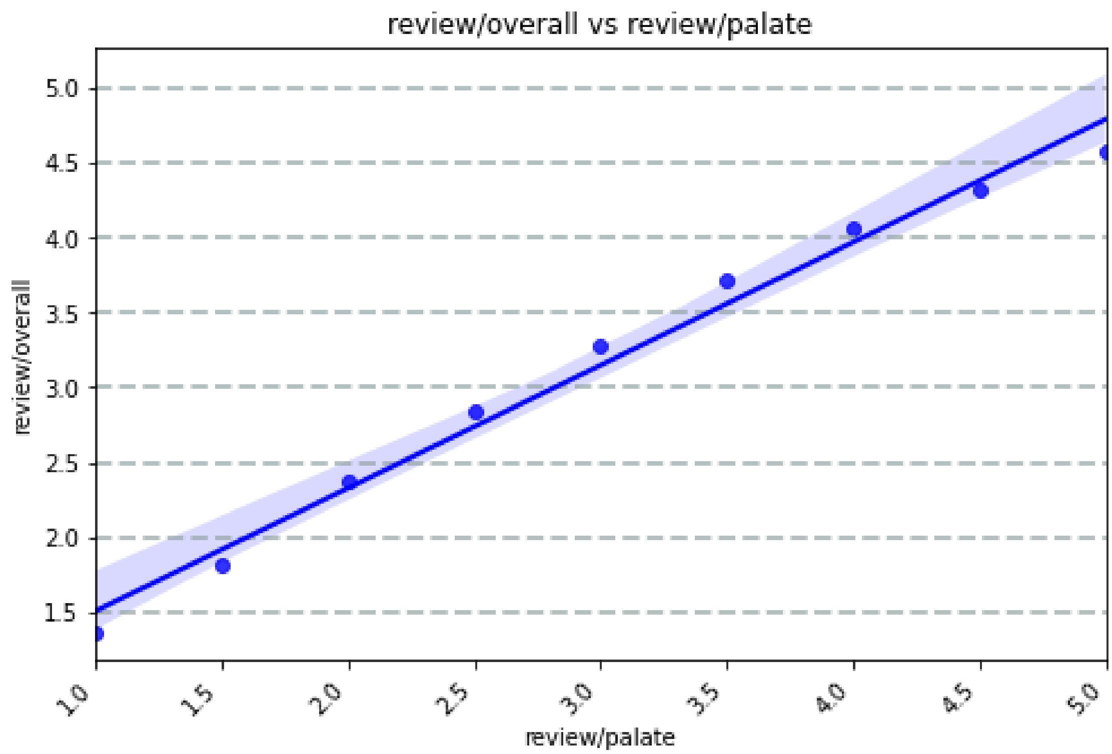

```
<Figure size 576x360 with 0 Axes>
```

[221]:
```
savePlot('review/aroma', 'green', True)
```



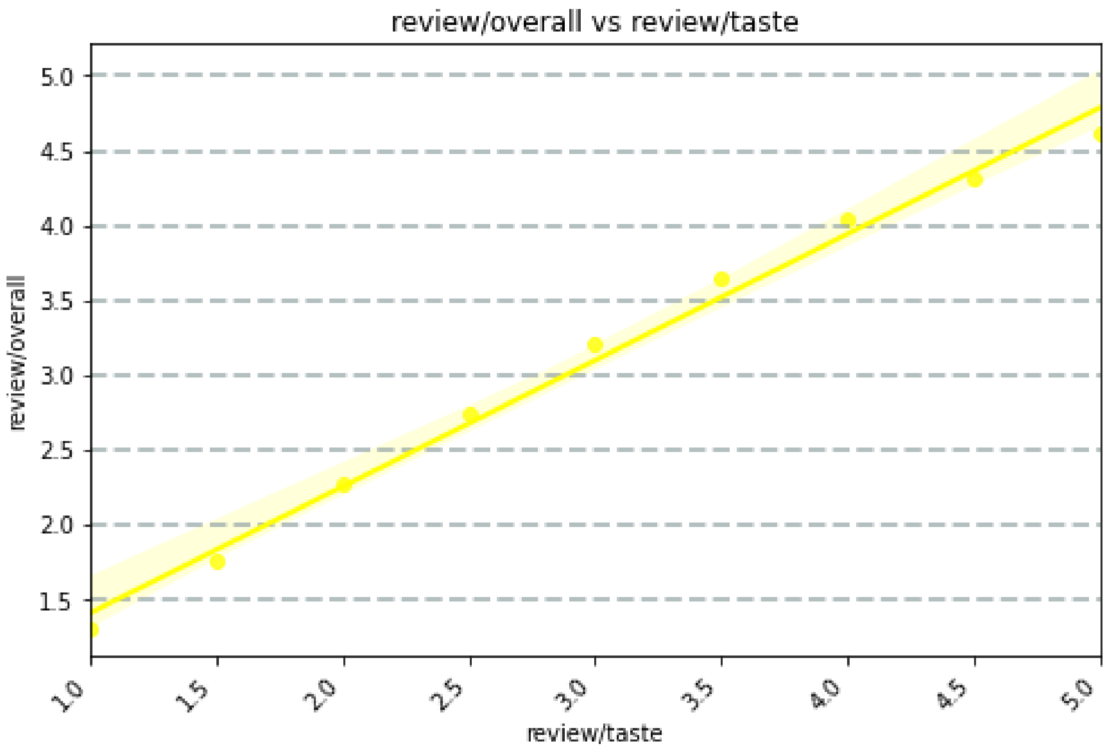

```
<Figure size 576x360 with 0 Axes>
```

[222]:
```
savePlot('review/palate', 'blue', True)
```



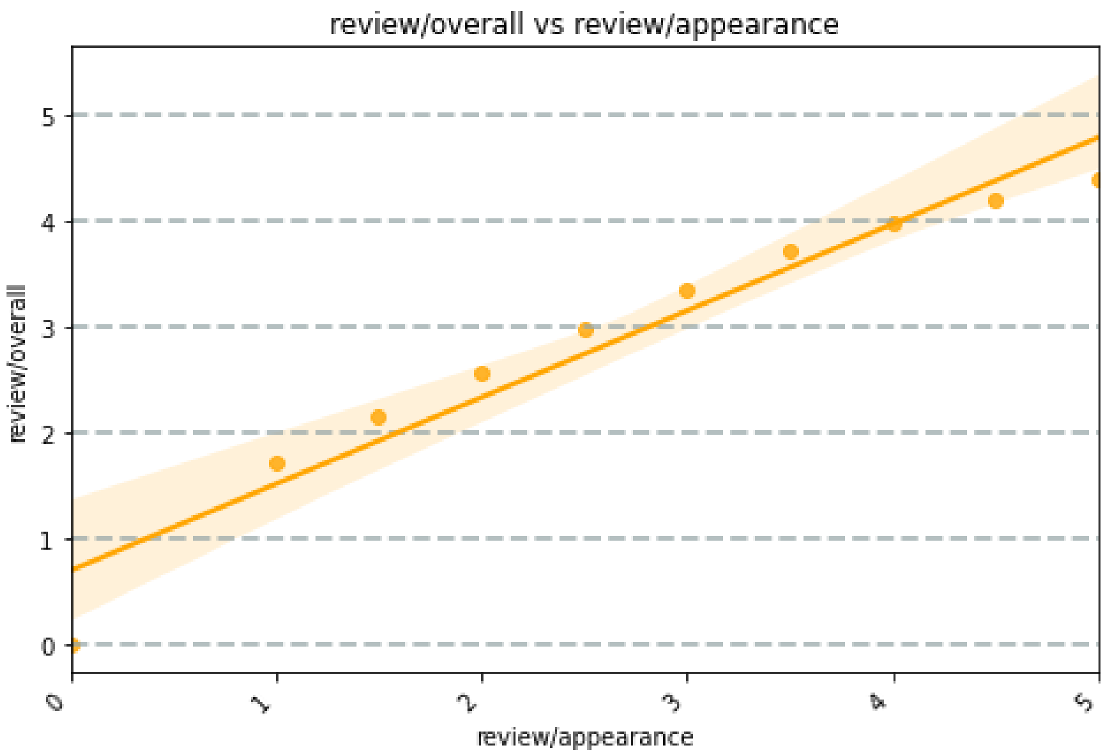

```
<Figure size 576x360 with 0 Axes>
```

[223]:
```
savePlot('review/taste', 'yellow', True)
```



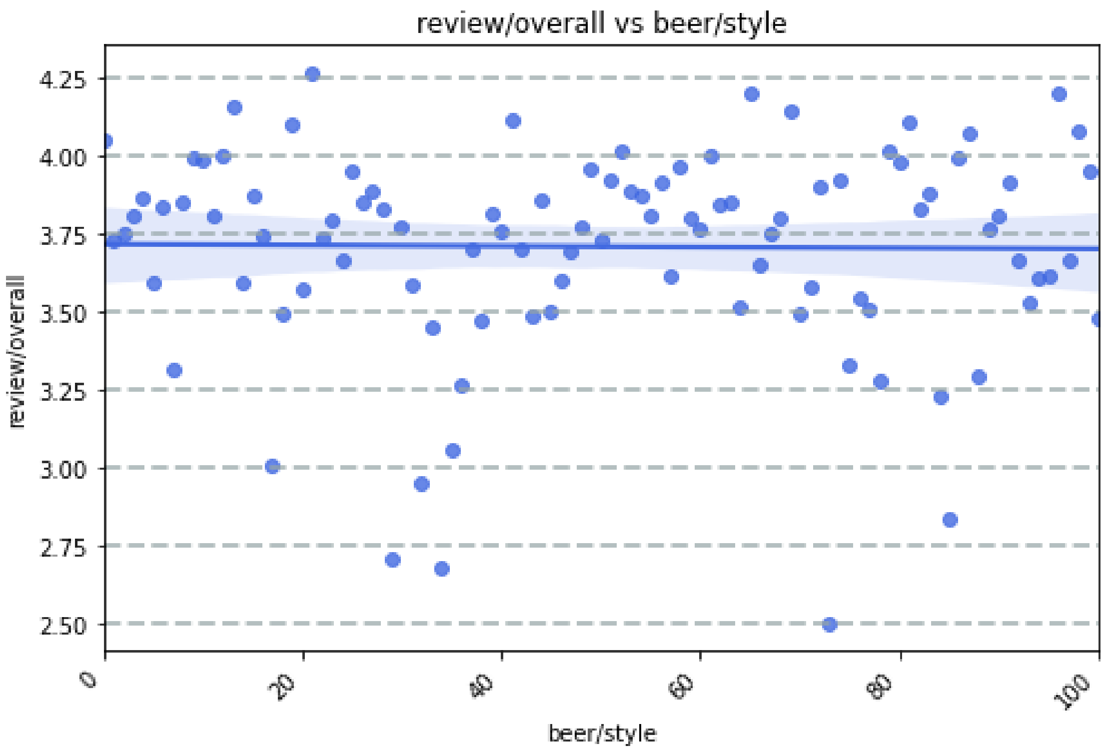

```
<Figure size 576x360 with 0 Axes>
```

[225]:
```
savePlot('review/appearance', 'orange', True)
```

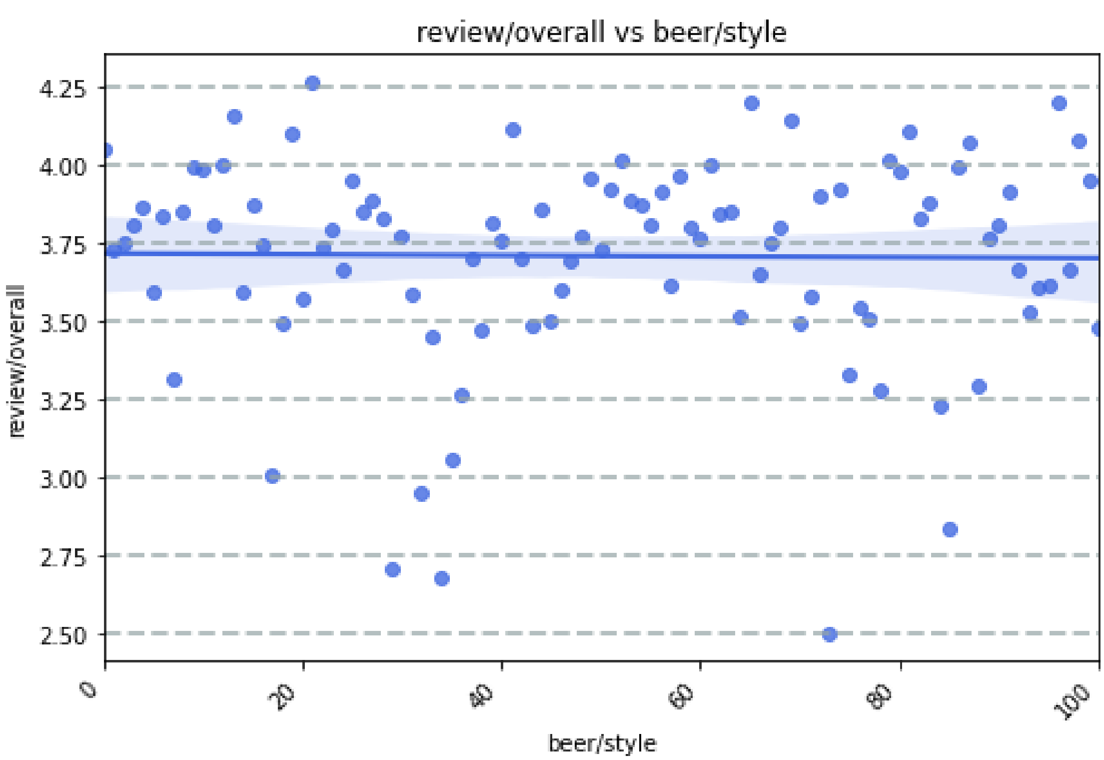



```
<Figure size 576x360 with 0 Axes>
```

[204]:
```
beer_style = df['beer/style'].unique()
scatter_plot(df, 'beer/style', beer_style, "royalblue", range(len(beer_style)))
```

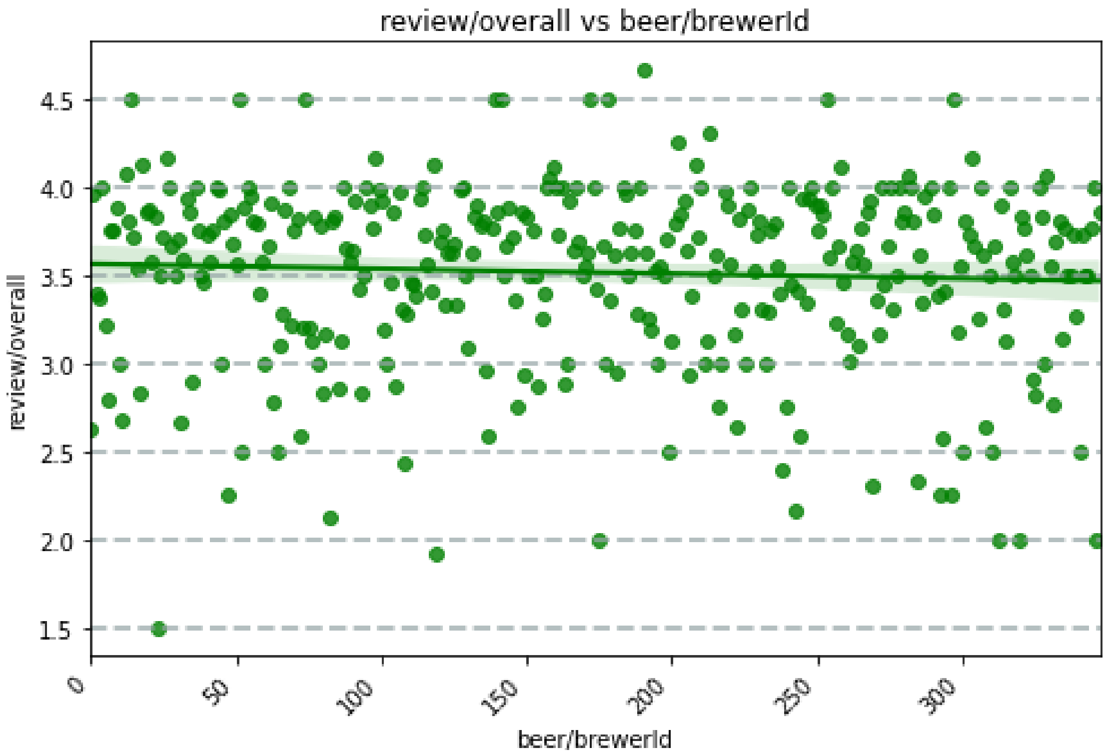



```
[228]: savePlot('beer/style', 'royalblue', False)
```



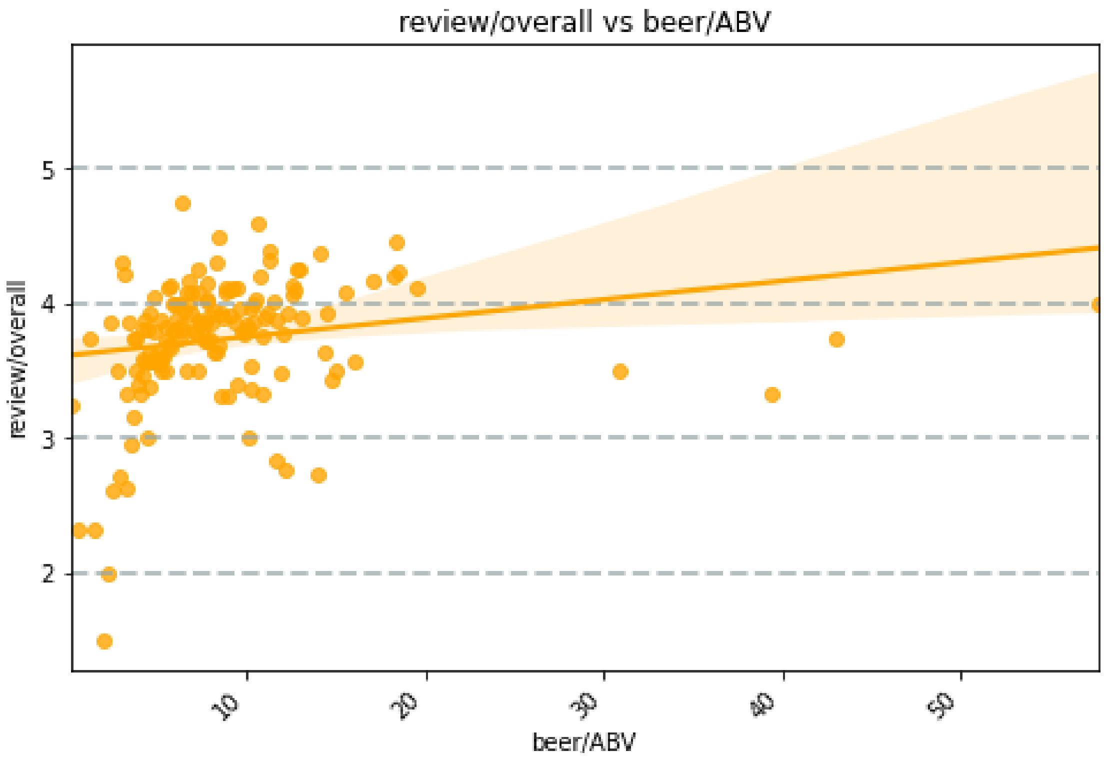
```
<Figure size 576x360 with 0 Axes>
```

[227]:
```python
savePlot('beer/brewerId', 'green', False)
```




```
<Figure size 576x360 with 0 Axes>
```

[229]:
```python
savePlot('beer/ABV', 'orange', True)
```




```
<Figure size 576x360 with 0 Axes>
```

[205]:
```
from sklearn.model_selection import train_test_split
train, test = train_test_split(df, test_size = 0.3, shuffle=True, random_state␣
↪=1)
```

[169]:
```
#where 0 is low rating, 1 is high rating

stop_words = set(stopwords.words('english'))
def create_bag(df):


    #for low rating
    unigrams1 = defaultdict(int)
    bigrams1 = defaultdict(int)

    #for high rating
    unigrams2 = defaultdict(int)
    bigrams2 = defaultdict(int)

#     for ind in df.index:
    for ind in df.index:
```

```
        count = 1
        b = ''
        for w in df['review/text'][ind].split():
            if w in stop_words: continue
            if df['review/overall'][ind] > mean:
                unigrams2[w] += 1
            else:
                unigrams1[w] += 1
            if count > 1:
                s = b + ' ' + w
                if df['review/overall'][ind] > mean:
                    bigrams2[s] += 1
                else: bigrams1[s] +=1
            b = w

            count += 1

    return unigrams1, bigrams1, unigrams2, bigrams2
```

[170]:
```
unigrams1, bigrams1, unigrams2, bigrams2 = create_bag(train)
```

[192]:
```
P_high_rating = sum(train['review/overall'] > mean)/len(train.index)
P_low_rating = 1 - P_high_rating
```

[199]:
```
P_high_rating2 = sum(test['review/overall'] > mean)/len(test.index)
print(P_high_rating2)
```

```
0.6817
```

[200]:
```
print(P_high_rating)
print(P_low_rating)
```

```
0.6806714285714286
0.3193285714285714
```

[201]:
```
def predictHighUni(text, word_bag1, word_bag2, document_len):
    p_high = 1
    p_low = 1
    bag_sum1 = sum(word_bag1.values())
    bag_sum2 = sum(word_bag2.values())


    text = text.split()
    t = set(text)
```

```
    for w in t:
#         if w in stop_words: continue
        if w in word_bag1:
#             p1 = word_bag1[w]/bag_sum1
            p_low *= word_bag1[w]/bag_sum1

        if w in word_bag2:
#             p2 = word_bag2[w]/bag_sum2
            p_high *= word_bag2[w]/bag_sum2
#             if p2 == 0:
#                 print("here: ", w, print(w in word_bag2))


#         print(p_low)
#         print(p_high)

    p_low *= P_low_rating
    p_high *= P_high_rating


#     if(p_high == p_low): return P_high_rating > P_low_rating

    return p_high > p_low

def predictHighBi(text, word_bag1, word_bag2, document_len):
    p_high = 1
    p_low = 1
    bag_sum1 = sum(word_bag1.values())
    bag_sum2 = sum(word_bag2.values())
    count = 1
    b = ''
    t = set(text.split())

    for w in t:
        if w in stop_words: continue
        if count == 1:
            b = w
            count = 2
            continue
        s = b + ' ' + w
        b = w

        if s in word_bag1:
            p_low *= word_bag1[s]/bag_sum1
        if s in word_bag2:
            p_high *= word_bag2[s]/bag_sum2
```

```
    p_low *= P_low_rating
    p_high *= P_high_rating


    return p_high > p_low

def MSE(predictions, labels):
    differences = [(x-y)**2 for x,y in zip(predictions,labels)]
    return sum(differences) / len(differences)
```

[39]:
```
r = train['review/text'].iloc[5]
```

[40]:
```
r
```

[40]:
```
' a buddy of mine stopped by with a sixer of this so i figured i would give it a
shot not much smell or taste to it very bland a bit corny and skunkykinda like
bad city water in some places ive been too carbonated for my liking i could only
finish one and wont be having another if i want a light beer ill grab a sa light
instead'
```

[41]:
```
train['review/overall'].iloc[5]
```

[41]:
```
1.5
```

[42]:
```
train_len =  len(train.index)
```

[172]:
```
"skunkykinda" in unigrams2
```

[172]:
```
False
```

[183]:
```
predictHighUni(r, unigrams1,unigrams2, train_len)
```

```
0.008734792297144968
0.006986583838060697
0.008734792297144968
0.006986583838060697
1.2570258040837995e-05
1.1986449927257011e-05
1.2570258040837995e-05
1.1986449927257011e-05
1.2570258040837995e-05
1.1986449927257011e-05
1.2570258040837995e-05
1.1986449927257011e-05
2.4331552197716275e-08
2.6189368972508198e-08
1.2490730083083058e-10
```

8.95361518203449e-11
2.2088459541071563e-14
1.3238296493342534e-14
1.5745612082236899e-16
9.658035318098583e-17
3.879006616086056e-19
7.439379943696974e-20
1.757136221960375e-23
5.344994489698513e-25
1.757136221960375e-23
5.344994489698513e-25
1.7602925029526255e-27
4.3859524297610374e-29
1.7602925029526255e-27
4.3859524297610374e-29
5.965076368659033e-31
6.833104371309734e-33
2.707293240637871e-34
3.772487691274451e-37
2.547068482695843e-38
2.767492691832297e-41
1.2425395363771145e-42
1.224416760330523e-45
1.926689746964787e-46
8.287799157993509e-50
1.926689746964787e-46
8.287799157993509e-50
3.4167022242457494e-48
1.42179244068226e-51
3.4167022242457494e-48
1.42179244068226e-51
3.5719562723050585e-50
1.3177065781765497e-53
3.5719562723050585e-50
1.3177065781765497e-53
1.9254755407542656e-52
6.3197169560083365e-56
5.702935906437894e-57
1.7206282360411082e-60
4.669903559296426e-61
1.4314185719592188e-64
2.2211716515838387e-64
2.473658648073199e-68
2.0026453165495636e-67
1.860507707521541e-71
3.628682261954124e-71
2.476458521403437e-75
1.4224696959934733e-75

8.709052893254167e-80
1.375458746511358e-78
1.0469317661413491e-82
1.375458746511358e-78
1.0469317661413491e-82
1.3779294338963094e-82
1.262891330181469e-86
1.3779294338963094e-82
1.262891330181469e-86
1.3779294338963094e-82
1.262891330181469e-86
1.3779294338963094e-82
1.262891330181469e-86
7.006453402233186e-85
6.338102759870197e-89
7.006453402233186e-85
6.338102759870197e-89
2.6855452968562776e-89
2.660352446403198e-93
1.9370658570384447e-92
1.588467112222343e-96
7.79761740213727e-95
5.1303131013160655e-99
7.79761740213727e-95
5.1303131013160655e-99
7.79761740213727e-95
5.1303131013160655e-99
7.79761740213727e-95
5.1303131013160655e-99
6.45308068208423e-99
4.888789442875709e-103
6.45308068208423e-99
4.888789442875709e-103
5.6214540176248456e-105
4.888789442875709e-103
4.3034848490848066e-107
2.9325375272627413e-105
3.6671570061926317e-109
2.002378150921832e-107
3.6671570061926317e-109
2.002378150921832e-107
2.424671164906299e-112
1.1425948411894754e-110
2.424671164906299e-112
1.1425948411894754e-110
1.527118409031894e-115
5.0681235362159786e-114
1.527118409031894e-115

```
5.0681235362159786e-114
1.527118409031894e-115
5.0681235362159786e-114
1.527118409031894e-115
5.0681235362159786e-114
```

[183]: True

[118]:
```
predictHighBi(r, bigrams1,bigrams2, train_len)
```

[118]: True

[251]:
```
def predict(x, s):
    y = []
    x_len = len(x)
    if s == "uni":
        for d in x:
            y.append(predictHighUni(d, unigrams1, unigrams2, x_len))
    else:
        for d in x:
            y.append(predictHighBi(d, bigrams1, bigrams2, x_len))
    return y

def findMSE(x, y, s):
    y = [d > mean for d in y]
    if s == "uni":
        y_pred = predict(x, "uni")
    else:
        y_pred = predict(x, "bi")
    return MSE(y_pred, y)

def findCorrect(x, y, s):
    y = [d > mean for d in y]
    if s == "uni":
        y_pred = predict(x, "uni")
    else:
        y_pred = predict(x, "bi")
    correct = [a==b for a,b in zip(y, y_pred)]
    return sum(correct)/ len(correct)
```

[132]:
```
text = list(train['review/text'])
rating = list(train['review/overall'])
```

[252]:
```
print("Correct rate by unigram of train set: ", findCorrect(text, rating,␣
↪"uni"))
```

```
Correct rate by unigram of train set:  0.4223857142857143
```

[253]:
```
print("Correct rate by bigram of train set: ", findCorrect(text, rating, "bi"))
```

```
Correct rate by bigram of train set:  0.34855714285714284
```

[ ]:
```
text_test = list(test['review/text'])
rating_test = list(test['review/overall'])
```

[254]:
```
print("Correct rate by unigram of test set: ", findCorrect(text_test,␣
 ↪rating_test, "uni"))
```

```
Correct rate by unigram of test set:  0.5993666666666667
```

[255]:
```
print("Correct rate by bigram of test set: ", findCorrect(text_test,␣
 ↪rating_test, "bi"))
```

```
Correct rate by bigram of test set:  0.36806666666666665
```

[239]:
```
def true_p(p, y, data_len):
    sum = 0
    for i in range(data_len):
        if p[i] == y[i] and p[i] == True:
            sum += 1
    return sum
def Ber(y, predictions, data_len):
    LP = sum(y)                         #labeled positive
    LN = data_len - LP                       #labeled negative
    PP = sum(predictions)                    #prediction positive
    PN = data_len - PP                       #prediction negative
    TP = true_p(predictions, y, data_len)  #true positive
    FN = LP - TP                             #false negative
    FP = PP - TP                             #false positive
    TN = PN - FN                             #false negative

    TPR = TP/LP
    TNR = TN/LN


    FPR = FP/LN
    FNR = FN/LP
    BER = 0.5*(FPR+FNR)


    print("True positive: ", TPR)
    print("True negative: ", TNR)
    return BER
# BER = Ber(y_test, predictions, data_len)
# print("Balanced Error Rate: ", BER)
```

```
def findRate(x, y):
    y = [d > mean for d in y]
    y_pred = predict(x, "uni")
    return Ber(y, y_pred, len(y))
```

[241]:
```
ber = findRate(text_test, rating_test)
print(ber)
```

```
True positive:  0.6021221456163512
True negative:  0.5934652843229657
0.4022062850303415
```

2 TFIDF MODEL

[20]:
```
uni1 = defaultdict(int)
uni2 = defaultdict(int)




for ind in train.index:
    for w in train['review/text'][ind].split():
        if w in stop_words: continue
        if train['review/overall'][ind] > mean:
            uni2[w] += 1
        else:
            uni1[w] += 1
```

[46]:
```
wordByRating = []
wordByRating.append(uni1)
wordByRating.append(uni2)


counts = []
# for i in range(2):
c = [(uni1[w], w) for w in uni1]
c.sort(reverse = True)
counts += c[:500]
c = [(uni2[w], w) for w in uni2]
c.sort(reverse = True)
counts += c[:500]

wordsUni = [x[1] for x in counts]
```

```
# print(counts[980:])

# counts = [(unigrams[w], w) for w in unigrams]
# counts.sort()
# counts.reverse()
# wordsUni = [x[1] for x in counts[:1000]]


# counts = [(bigrams[w], w) for w in bigrams]
# counts.sort()
# counts.reverse()
# wordsBi = [x[1] for x in counts[:1000]]
# counts[:5]
```

[56]:

```
def merge(list1, list2):
#     print(list1[0])
    merged_list = [(list1[i], list2[i]) for i in range(0, len(list1))]
    return merged_list

def featureTfidf(z, words, wordId):
 ↪#=============================================================================
    feat = [0]*len(words)
#     print(z[1])
    for w in z[0]:
#         print(w)
        if w in words:
#             print(w, z[1][w])
            feat[wordId[w]] = z[1][w]

    feat.append(1)
    return feat



def prepX(review, words):
#     print(text)
    Df = defaultdict(float)
    Tfidf = []

    text = []
    l = len(review)

    for r in review:
        w = r.split()
#         print(w)
        text.append(w)
```

```
#     print(text)
#     return 0
#     print(w)

    for t in words:
        Df[t] = 0
        for w in text:
            if t in w:
                Df[t] += 1


    for w in text:
        _tfidf = defaultdict(float)
        for t in words:
            _tf = w.count(t)
#             print("tf: ", _tf)
            if Df[t] == 0:
                _tfidf[t] = 0
            else:
                _tfidf[t] = _tf * math.log(1/Df[t], 10)
#             print("tfidf: ", _tfidf[t])
        Tfidf.append(_tfidf)


    wordId = dict(zip(words, range(len(words))))

#     print()
    listX = merge(text, Tfidf)
#     print(listX[0])
    X = [featureTfidf(d, words, wordId) for d in listX]
    return csr_matrix(X)

def build_train_model(text, rating, words, c):
    X = prepX(text, words)
    Y = rating
    clf = linear_model.Ridge(c, fit_intercept=False)
    clf.fit(X, y)
    return clf


def longPredict(text, rating, text_test, rating_test, words, c):
    X = prepX(text, words)
    y = [d > mean for d in rating]

    Xtest = prepX(text_test, words)
    ytest = [d > mean for d in rating_test]
    # Regularized regression
```

```
    for i in c:
        clf = linear_model.LogisticRegression(C=0.1, class_weight='balanced',␣
↪fit_intercept=False)
        clf.fit(X, y)
        predictions = clf.predict(Xtest)
        correct = [a==b for a,b in zip(predictions, ytest)]
        print("Correct rate with C=", i,":",  sum(correct)/ len(correct))
```

[53]:
```
print(text[0])
```

```
 clear shiny gold with not a whole lot of head retention after an initially
foamy pour has that semiskunked lager aroma with a good mix of light graininess
and hops very crisp and refreshing clean thru the middle ending with minimal hop
spiciness makes a great hot weather alternative to the ever present wheat beers
nothing earthshattering here just another great brew in sierra nevadas already
stellar lineup
```

[70]:
```
c = [0.01, 0.1, 1, 10, 100]
```

[66]:
```
X = prepX(text, wordsUni)
y = [d > mean for d in rating]
clf = linear_model.LogisticRegression(C=0.1, class_weight='balanced',␣
↪fit_intercept=False)
clf.fit(X, y)
predictions = clf.predict(X)
correct = [a==b for a,b in zip(predictions, y)]
print("Correct rate with C=", 0.1,":",  sum(correct)/ len(correct))
```

```
Correct rate with C= 0.1 : 0.6822857142857143
```

[72]:
```
for i in c:
    clf = linear_model.LogisticRegression(C=i, class_weight='balanced',␣
↪fit_intercept=False)
    clf.fit(X, y)
    predictions = clf.predict(X)
    correct = [a==b for a,b in zip(predictions, y)]
    print("Correct rate with C=", i,":",  sum(correct)/ len(correct))
```

```
Correct rate with C= 0.01 : 0.6820142857142857
Correct rate with C= 0.1 : 0.6822857142857143
Correct rate with C= 1 : 0.6822285714285714
Correct rate with C= 10 : 0.6822142857142857
Correct rate with C= 100 : 0.6822142857142857
```

[73]:
```
Xtest = prepX(text_test, wordsUni)
ytest = [d > mean for d in rating_test]
```

[75]:
```
for i in c:
    clf = linear_model.LogisticRegression(C=i, class_weight='balanced',␣
↪fit_intercept=False)
    clf.fit(X, y)
    predictions = clf.predict(Xtest)
    correct = [a==b for a,b in zip(predictions, ytest)]
    print("Correct rate with C=", i,":",  sum(correct)/ len(correct))
```

```
Correct rate with C= 0.01 : 0.6717
Correct rate with C= 0.1 : 0.6720666666666667
Correct rate with C= 1 : 0.6723
Correct rate with C= 10 : 0.6723333333333333
Correct rate with C= 100 : 0.6722666666666667
```

3 KNN MODEL

[128]:
```
from sklearn.neighbors import KNeighborsClassifier

def knnClass(text, rating, text_test, rating_test, words, n):
    X = prepX(text, words)
    y = [d > mean for d in rating]

    Xtest = prepX(text_test, words)
    ytest = [d > mean for d in rating_test]
    knn = KNeighborsClassifier(n_neighbors=n)

    knn.fit(X, y)

    #Predict the response for test dataset
    y_pred = knn.predict(Xtest)
    correct = [a==b for a,b in zip(y_pred, ytest)]
    print("Correct rate with n=", n,":",  sum(correct)/ len(correct))
```

[130]:
```
knnClass(text, rating, text_test, rating_test, wordsUni, 3)
```

```
Correct rate with n= 3 : 0.6388333333333334
```

[133]:
```
X = prepX(text, wordsUni)
y = [d > mean for d in rating]

Xtest = prepX(text_test, wordsUni)
ytest = [d > mean for d in rating_test]
```

[137]:
```
n = [1, 2, 3]
```

[139]:
```
def predictKnn(Xtest, ytest, n):
    for i in n:
        knn = KNeighborsClassifier(n_neighbors=i)

        knn.fit(X, y)

        #Predict the response for test dataset
        y_pred = knn.predict(Xtest)
        correct = [a==b for a,b in zip(y_pred, ytest)]
        print("Correct rate with n=", i,":",  sum(correct)/ len(correct))
```

[138]:
```
predictKnn(Xtest, ytest, n)
```

```
Correct rate with n= [1, 2, 3] : 0.5919
Correct rate with n= [1, 2, 3] : 0.5236
Correct rate with n= [1, 2, 3] : 0.6388333333333334
```

[140]:
```
predictKnn(X, y, n)
```

```
Correct rate with n= 1 : 0.9282
Correct rate with n= 2 : 0.7467
Correct rate with n= 3 : 0.7914142857142857
```

[ ]:
```

```