

## DSE220 Amazon Product Review Kaggle Competition

**Model 1:** [ensemble\\_gb\\_0.168\\_rounding1\\_6.10.2017.csv](#)

**Public Score: 0.16171**

**Private Score: 0.16457**

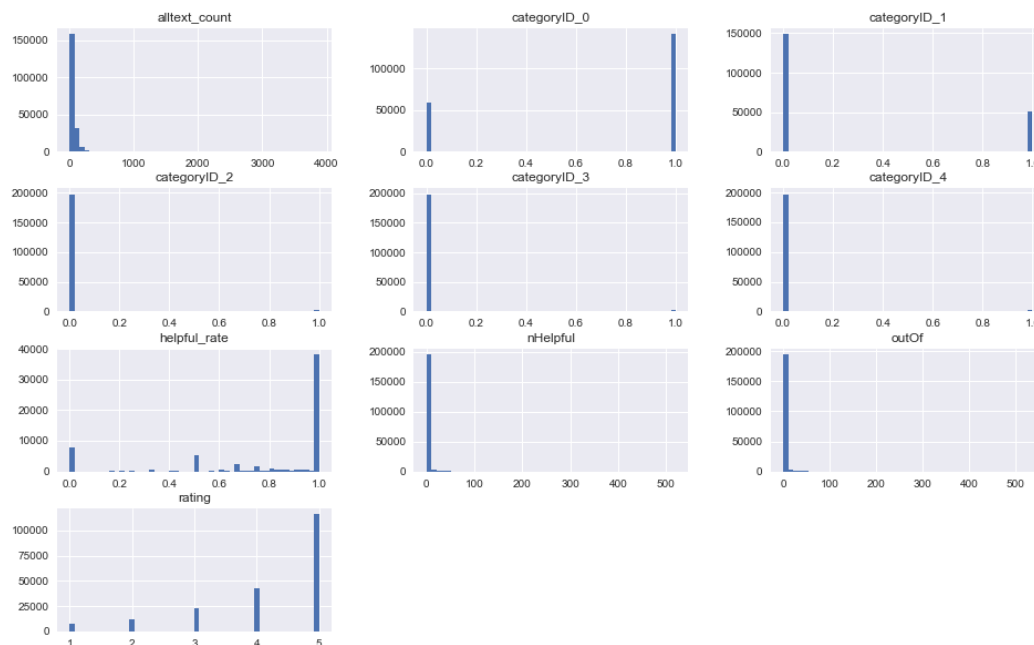
### Methodology

#### I. Data Preprocessing

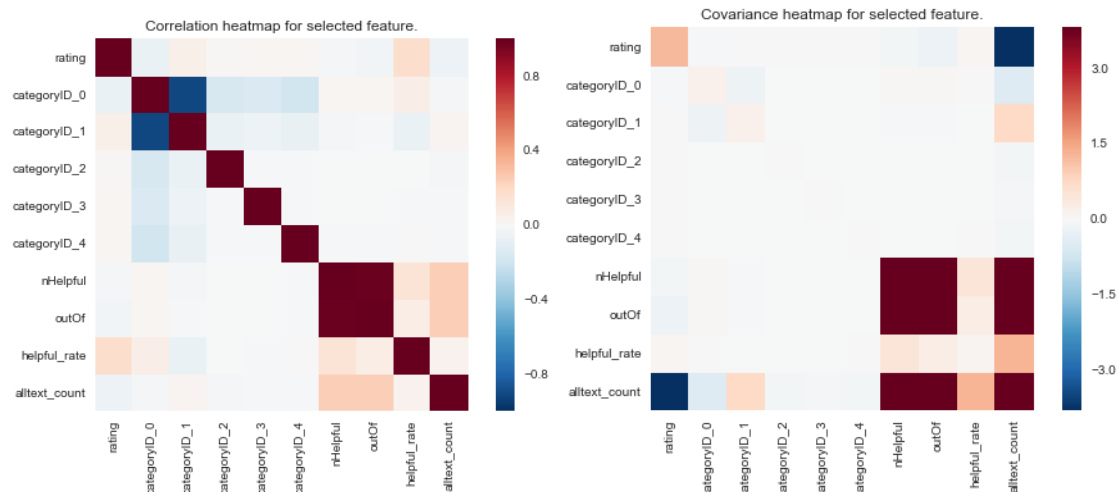
Following the Baseline.pynb file, the data was loaded into Jupyter Notebook appropriately as a pandas dataframe and the shape and the contents of the dataframe examined. The distribution of null values and unique values was evaluated for the training dataset, finding that the column price should be dropped due to ~63% of the values being null and reviewHash dropped due values from the feature being completely unique. The function data\_preprocessing turned the categorical feature, categoryID, into binarized features in order for models to be able to use it, removed categoryID, converted reviewTime into a pandas datetime value, and removed reviewHash and price. The helpful\_ratio (nHelpful/outOf) was computed for the training dataset. Furthermore, the reviewText was inputted into functions associated with counting the number of words and the number of key words (without punctuation and stopwords) in each reviewText instances. This final dataframe was used for data exploration.

#### II. Data Exploration

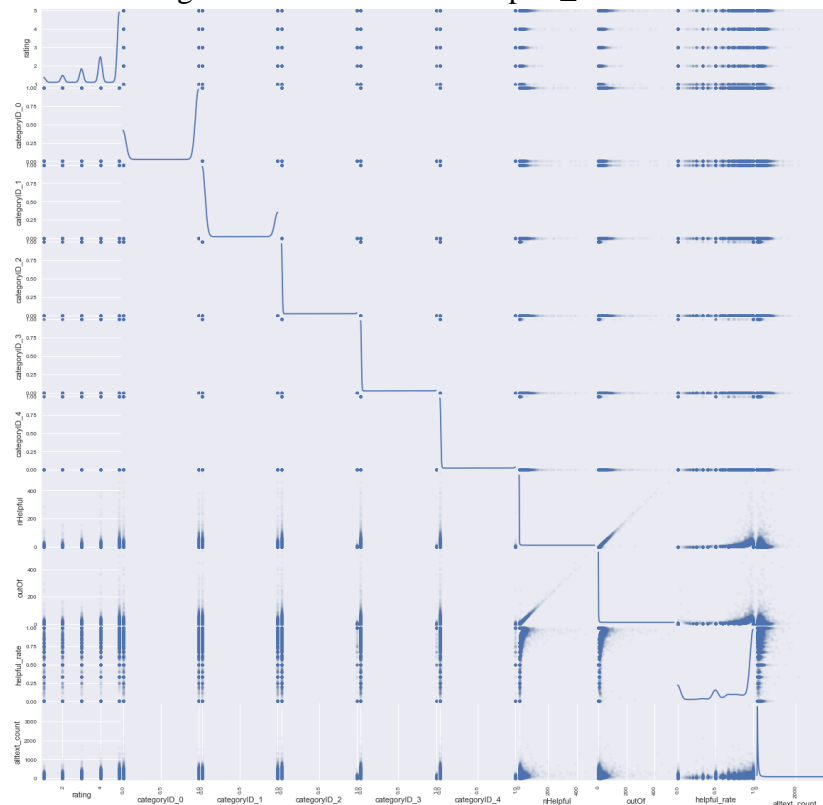
The dataframe with the alltext\_count, categoryID\_0-4, helpful\_rate, nHelpful, out of, and rating. The distribution of instances among the features was examined using simple histograms, finding that the class attribute, nHelpful, following a head-and-tail distribution with the majority of the instances falling below  $nHelpful < 10$ .



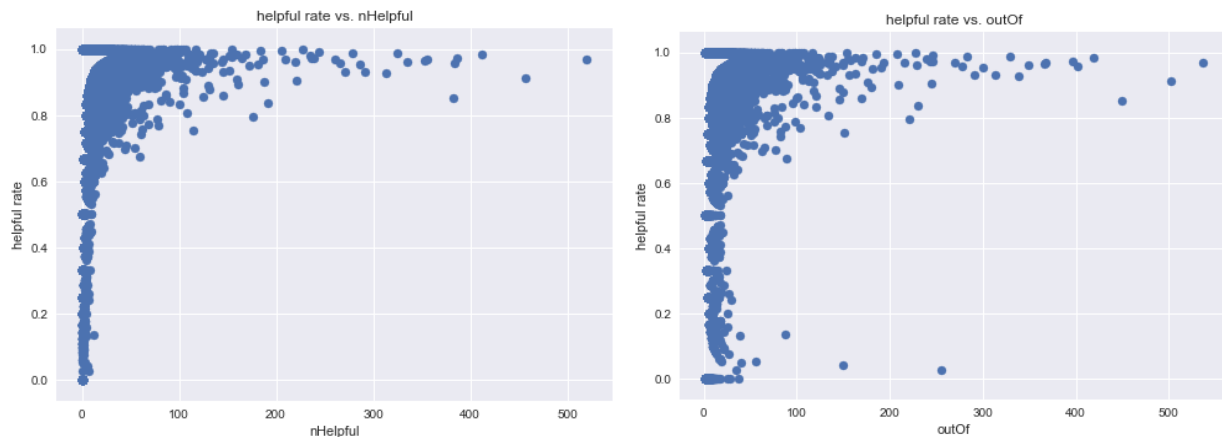
Next, simple statistics were analyzed in the dataset, finding that the nHelpful feature varied from 0 to 520 with an average of 1.1153 (this is due to the majority of the outOf values being equal to 1). While visualizing the covariance and correlation heatmap matrices among the features, it was noted that the relationship between out of and the helpful\_rate is significant, of course as expected.



Scatter matrices were generated to compare the distribution changes between each of the features, most notably seeing that the relationship between outOf and helpful\_rate showed that the majority of the data belonged to outOf  $\leq 1$  and helpful\_rate  $> 0.9$ .



The scatter plots for helpful rate vs. outOf and helpful rate vs. nHelpful were analyzed. This provided the idea that an ensemble of algorithms should be used to predict nHelpful, that the helpful\_rate should be used to indirectly predict nHelpful, and that certain constraints or thresholds should be made on the models depending on the value of out of.



### III. Modeling

The following features were used to predict nHelpful (following feature relationship analysis): 'rating', 'outOf', 'categoryID\_0', 'categoryID\_1', 'categoryID\_2', 'categoryID\_3', 'categoryID\_4'

To prevent overfitting, the training set was iteratively split into training and validation sets based on the random\_state feature. GridSearchCV was used to iteratively determine the model which decreased the mae the most and was done by using 10-fold cross-validation on the inputted training set and testing on varied validation sets. Majority vote for the different models was used to create the final model and the model with the lowest mae was chosen.

#### Chosen Model

If outOf == 0 then helpful\_rate = 0;

Elif outOf == 1 then predict helpful\_rate using the optimized gradient boosting classifier:

```
gbclf = ensemble.GradientBoostingClassifier(criterion='friedman_mse', init=None,
learning_rate=0.1, loss='deviance', max_depth=4,
max_features=None, max_leaf_nodes=None,
min_impurity_split=1e-07, min_samples_leaf=1,
min_samples_split=2, min_weight_fraction_leaf=0.0,
n_estimators=100, presort='auto', random_state=None,
subsample=1.0, verbose=0, warm_start=False)
```

Else then predict helpful\_rate using the optimized gradient boosting regressor:

```
gref = ensemble.GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse',
init=None,
learning_rate=0.1, loss='lad', max_depth=4, max_features=None,
max_leaf_nodes=None, min_impurity_split=1e-07,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=100,
```

```
presort='auto', random_state=None, subsample=1.0, verbose=0,  
warm_start=False)
```

Following acquiring predictions, the helpful\_rate values were multiplied with outOf values. The resulting values were rounded and exported as a .csv file. After 10 iterations of different validation sets, the “mean” mae was 0.168.

**Model 2: [predictions\\_gbreg\\_rounding\\_Helpful.csv](#)**

**Public Score: 0.16271**

**Private Score: 0.16857**

The same data pre-processing, data exploration, and data preparation was used to create the model, but this model was directly predicting nHelpful (without use of helpful\_rate).

**Chosen Model**

The following features were used:

'rating', 'outOf', 'categoryID\_0', 'categoryID\_1', 'categoryID\_2', 'categoryID\_3', 'categoryID\_4'

If outOf == 0 then nHelpful = 0;

Else predict nHelpful using the optimized gradient boosting classifier:

```
gbreg1 = ensemble.GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse',  
init=None,  
learning_rate=0.1, loss='lad', max_depth=4, max_features=None,  
max_leaf_nodes=None, min_impurity_split=1e-07,  
min_samples_leaf=1, min_samples_split=2,  
min_weight_fraction_leaf=0.0, n_estimators=100,  
presort='auto', random_state=None, subsample=1.0, verbose=0,  
warm_start=False)
```

Following acquiring predictions, the nHelpful values were rounded and exported as a .csv file.

**Model 1 worked significantly better due to the use of cross-fold validation and through iteratively creating different training and validation sets.**

# Final Model 1

June 12, 2017

## 1 Final: Amazon Product Review Kaggle Competition

### 2 Model 1

#### 2.1 DSE 220: Machine Learning

#### 2.2 Due Date: 11 June 2017

#### 2.3 Orysyia Stus

```
In [72]: import pandas as pd
import numpy as np
from collections import defaultdict
import gzip
import matplotlib as plt
%pylab inline

from sklearn.cross_validation import StratifiedKFold, cross_val_predict
from sklearn import preprocessing
```

Populating the interactive namespace from numpy and matplotlib

```
C:\Users\Orysyia\Anaconda\envs\py36\lib\site-packages\IPython\core\magics\pylab.py:1
`%matplotlib` prevents importing * from pylab and numpy
"\n`matplotlib` prevents importing * from pylab and numpy"
```

```
In [2]: def readGz(f):
        for l in gzip.open(f):
            yield eval(l)

        def parse(path):
            g = gzip.open(path, 'rb')
            for l in g:
                yield eval(l)

        def getDF(path):
            i = 0
```

```

df = {}
for d in parse(path):
    df[i] = d
    i += 1
return pd.DataFrame.from_dict(df, orient='index')

train_df = getDF('train.json.gz')
test_df = getDF('test_Helpful.json.gz')

```

## 2.4 Data Processing & Feature Engineering

```

In [3]: print('The shape of train_df is', train_df.shape)
        train_df.head(2)

```

The shape of train\_df is (200000, 12)

```

Out[3]:
categoryID      categories      itemID
0           0  [[Clothing, Shoes & Jewelry, Women], [Clothing...  I65535532
1           0  [[Clothing, Shoes & Jewelry, Women, Clothing, ...  I24109231

reviewerID  rating      reviewText
0  U745881038    3.0  These are cute, but they are a little small. ...
1  U023577405    4.0  I love the look of this bra, it is what I want...

reviewHash  reviewTime      summary  unixReviewTime
0  R115160670  05 20, 2014      Cute      1400544000
1  R800651687  02 7, 2013  Beautiful but size runs small      1360195200

helpful  price
0  {'outOf': 0, 'nHelpful': 0}    NaN
1  {'outOf': 0, 'nHelpful': 0}    NaN

```

```

In [10]: # Examine the number of null values in the train_df
         print(train_df.isnull().sum())
         print('\n Column price needs to be dropped since', train_df['price'].isnu

```

```

categoryID      0
categories      0
itemID          0
reviewerID      0
rating          0
reviewText      0
reviewHash      0
reviewTime      0
summary         0
unixReviewTime  0
helpful         0
price          125851

```

dtype: int64

Column price needs to be dropped since 62.9255 % of the data is null.

```
In [66]: # Determine which items might be too unique for predictive purposes
print('For column itemID, there are ', train_df['itemID'].nunique(), 'unique values for 200000 instances.')
print('For column itemID, there are ', train_df['reviewerID'].nunique(), 'unique values for 200000 instances.')
print('For column itemID, there are ', train_df['reviewHash'].nunique(), 'unique values for 200000 instances.')
print('For column itemID, there are ', train_df['categoryID'].nunique(), 'unique values for 200000 instances.')

print('\n All values for reviewHash are unique, thus column reviewHash does not provide any predictive power')
```

For column itemID, there are 19913 unique values for 200000 instances.  
For column itemID, there are 39249 unique values for 200000 instances.  
For column itemID, there are 200000 unique values for 200000 instances.  
For column itemID, there are 5 unique values for 200000 instances.

All values for reviewHash are unique, thus column reviewHash does not provide any predictive power

```
In [4]: print('The shape of train_df is', test_df.shape)
        test_df.head(2)
```

The shape of train\_df is (14000, 12)

```
Out[4]:
```

	categoryID	categories	itemID
0	0	[[Sports & Outdoors, Other Sports, Dance, Clot...	I52093239
1	0	[[Sports & Outdoors, Clothing, Women, Hoodies]...	I96953233

	reviewerID	rating	reviewText
0	U816789534	3.0	I ordered according to the size chart but it's...
1	U987148846	4.0	Super thin but really cute and not cheap-looki...

	reviewHash	reviewTime	summary	unixReviewTime	helpful	price
0	R157684793	07 15, 2011	Too small	1310688000	{'outOf': 2}	NaN
1	R732719858	07 17, 2013	Fun hoodie	1374019200	{'outOf': 0}	NaN

```
In [83]: def data_preprocessing(dataframe):
        """The function breakdown the helpful colum into outOf & nHelpful. Co
        reviewHash (unique, provides no predictive power). Takes categoryID an
        deleted categoryID."""
        dummies = pd.get_dummies(dataframe['categoryID']).rename(columns=lambda x: x + '_categoryID')
        dataframe = pd.concat([dataframe, dummies], axis=1)
        del dataframe['categoryID']
        helpful = pd.DataFrame.from_dict(dict(dataframe['helpful'])).T
        dataframe = pd.concat([dataframe, helpful], axis=1)
        del dataframe['helpful']
```

```

dataframe['reviewTime'] = pd.to_datetime(dataframe['reviewTime'])
del dataframe['unixReviewTime']
del dataframe['reviewHash']
del dataframe['price']
return dataframe

```

```

In [84]: def get_helpful_rate(dataframe):
        """The function creates the field helpful_rate."""
        dataframe['helpful_rate'] = dataframe['nHelpful']/dataframe['outOf']
        return dataframe

```

```

In [85]: train = data_preprocessing(train_df)
        # The helpful_rate (nHelpful/outof) might be a better feature holding more
train = get_helpful_rate(train)
train.head(2)

```

```

Out[85]:
              categories      itemID  reviewer
0  [[Clothing, Shoes & Jewelry, Women], [Clothing...  I655355328  U7458810
1  [[Clothing, Shoes & Jewelry, Women, Clothing, ...  I241092314  U0235774

      rating      reviewText  reviewTime \
0        3.0  These are cute, but they are a little small.  ...  2014-05-20
1        4.0  I love the look of this bra, it is what I want...  2013-02-07

      summary  categoryID_0  categoryID_1  categoryID_2
0          Cute           1           0           0
1  Beautiful but size runs small           1           0           0

      categoryID_3  categoryID_4  nHelpful  outOf  helpful_rate
0                0             0          0      0          NaN
1                0             0          0      0          NaN

```

```

In [86]: test = data_preprocessing(test_df)
test.head(2)

```

```

Out[86]:
              categories      itemID  reviewer
0  [[Sports & Outdoors, Other Sports, Dance, Clot...  I520932398  U8167895
1  [[Sports & Outdoors, Clothing, Women, Hoodies]...  I969532331  U9871488

      rating      reviewText  reviewTime \
0        3.0  I ordered according to the size chart but it's...  2011-07-15
1        4.0  Super thin but really cute and not cheap-looki...  2013-07-17

      summary  categoryID_0  categoryID_1  categoryID_2  categoryID_3 \
0  Too small           1           0           0           0
1  Fun hoodie           1           0           0           0

      categoryID_4  outOf
0                0      2
1                0      0

```



```

In [112]: from nltk.corpus import stopwords
import string
from itertools import chain

def reviewText_listed(row):
    all_words = row.split()
    all_words = [w.lower() for w in all_words]
    # subset_list = [''.join(c for c in s if c not in string.punctuation)
    # subset_list = [w for w in subset_list if w != '']
    # subset_list = [word for word in subset_list if word not in stopwords]
    # return all_words, len(all_words), subset_list, len(subset_list)
    return len(all_words)

def reviewText_processing(dataframe):
    """The function returns 4 dictionaries (all review text words, all review
    words count, content (non-punctuation & non-stopwords) text words, content
    words count, & a vocabulary list of the content words)."""
    # review_allText = {}
    review_allText_count = {}
    # review_keyText = {}
    # review_keyText_count = {}
    # vocabulary = []
    count = 0
    # if subset_on == 'True':
    #     dataframe = dataframe[dataframe['outOf'] != 0]
    # else:
    #     dataframe = dataframe
    reviewText = list(dataframe['reviewText'])
    for text in reviewText:
        # all_, all_count, subset_, subset_count = reviewText_listed(text)
        all_count = reviewText_listed(text)
        # review_allText[count] = all_
        review_allText_count[count] = all_count
        # review_keyText[count] = subset_
        # review_keyText_count[count] = subset_count
        # vocabulary.append(subset_)
        count += 1
    # vocabulary = set(list(chain.from_iterable(vocabulary)))
    return dataframe, review_allText_count
    # return dataframe, review_allText, review_allText_count, review_keyText_count

In [113]: train, train_alltext_count = reviewText_processing(train)
train['alltext_count'] = list(train_alltext_count.values())
# train['keytext_count'] = list(train_keytext_count.values())
# train['key_vs_all_ratio'] = train['keytext_count']/train['alltext_count']

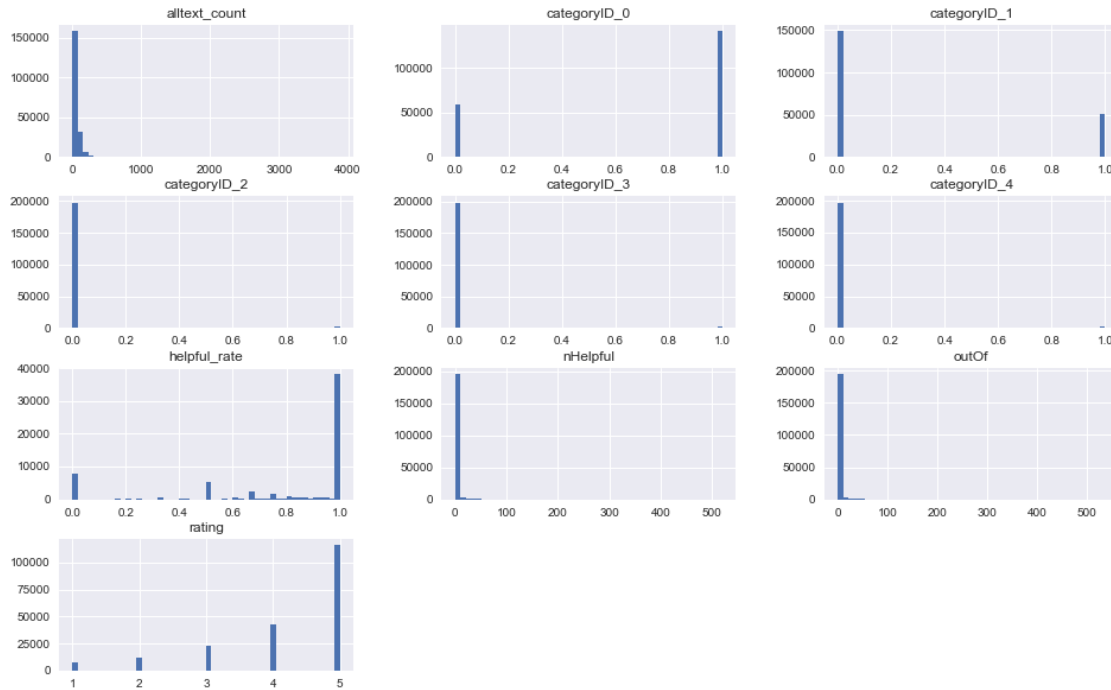
test, test_alltext_count = reviewText_processing(test)
test['alltext_count'] = list(test_alltext_count.values())

```

```
# test['keytext_count'] = list(test_keytext_count.values())
# test['key_vs_all_ratio'] = test['keytext_count']/test['alltext_count']
```

## 2.5 Data Exploration

```
In [121]: train.hist(bins = 50, figsize=(16,10));
```



```
In [116]: train.describe()
```

```
Out[116]:
```

	rating	categoryID_0	categoryID_1	categoryID_2	\
count	200000.000000	200000.000000	200000.000000	200000.000000	
mean	4.233590	0.707050	0.256990	0.011730	
std	1.107719	0.455117	0.436975	0.107668	
min	1.000000	0.000000	0.000000	0.000000	
25%	4.000000	0.000000	0.000000	0.000000	
50%	5.000000	1.000000	0.000000	0.000000	
75%	5.000000	1.000000	1.000000	0.000000	
max	5.000000	1.000000	1.000000	1.000000	

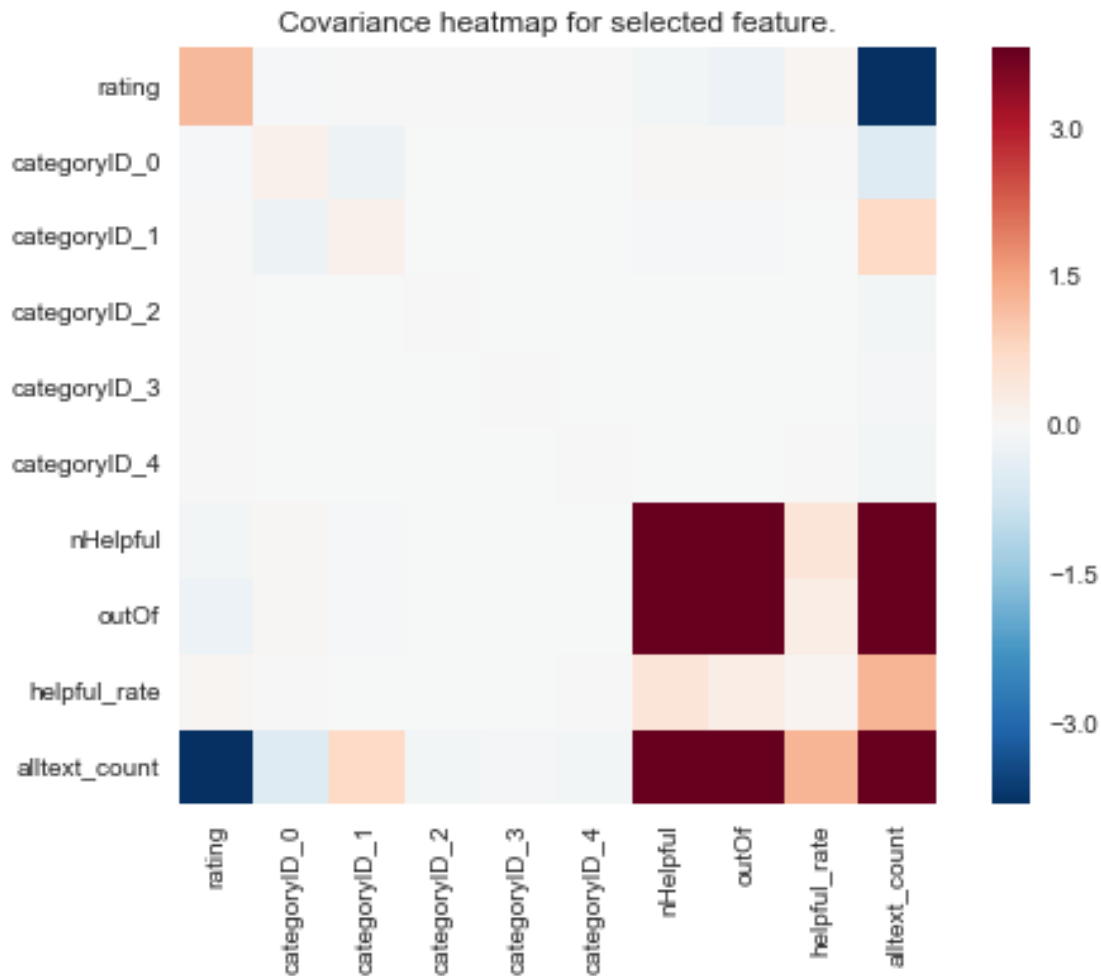
	categoryID_3	categoryID_4	nHelpful	outOf	\
count	200000.000000	200000.000000	200000.000000	200000.000000	
mean	0.009540	0.014690	1.115355	1.309145	
std	0.097206	0.120309	5.863531	6.307235	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	

50%	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	1.000000	1.000000
max	1.000000	1.000000	520.000000	537.000000

	helpful_rate	alltext_count
count	63016.000000	200000.000000
mean	0.782856	59.164485
std	0.344661	59.263191
min	0.000000	0.000000
25%	0.666667	26.000000
50%	1.000000	41.000000
75%	1.000000	69.000000
max	1.000000	3882.000000

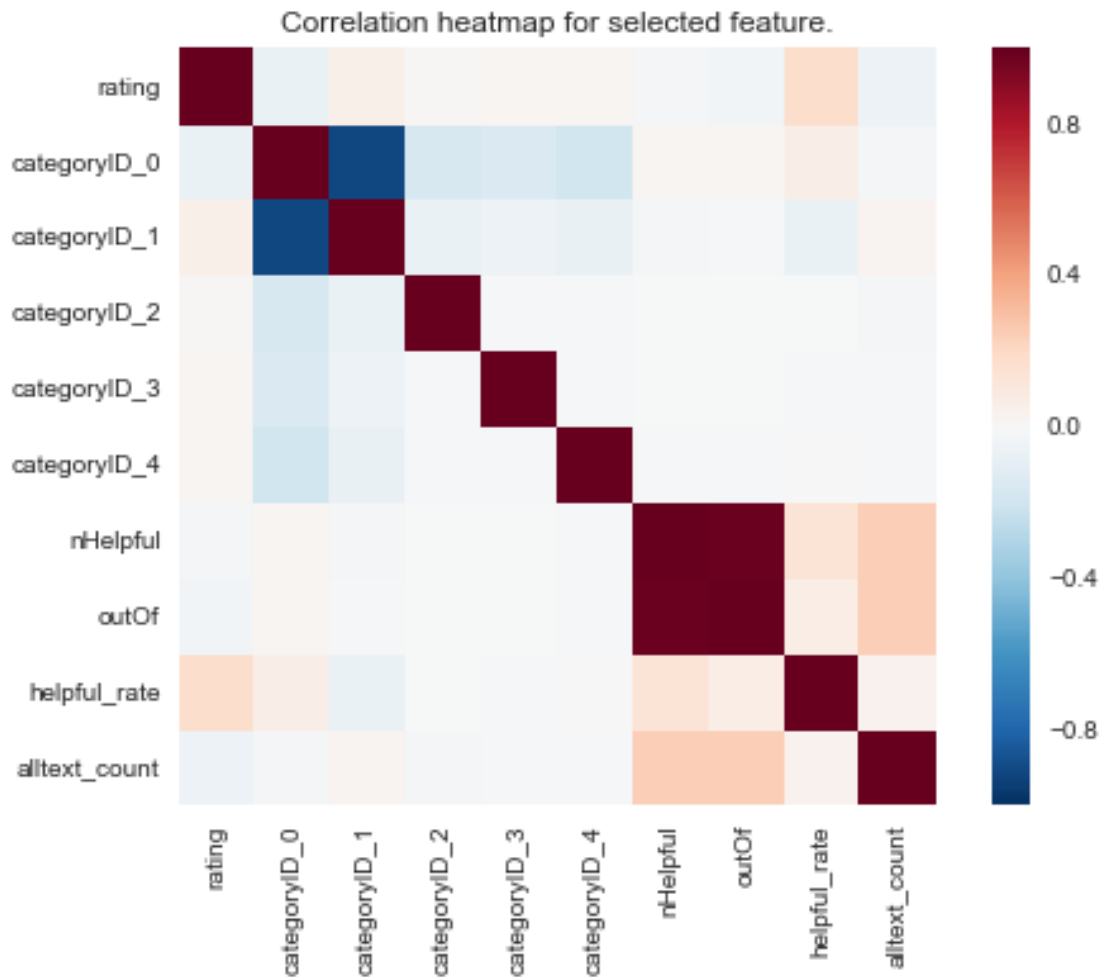
```
In [117]: import seaborn as sns
mat = train.cov() # to get a heatmap of the covariance matrix
a = sns.heatmap(mat, vmax=1, square = True)
a.set_title('Covariance heatmap for selected feature.')
```

Out[117]: <matplotlib.text.Text at 0x21ca7109c50>

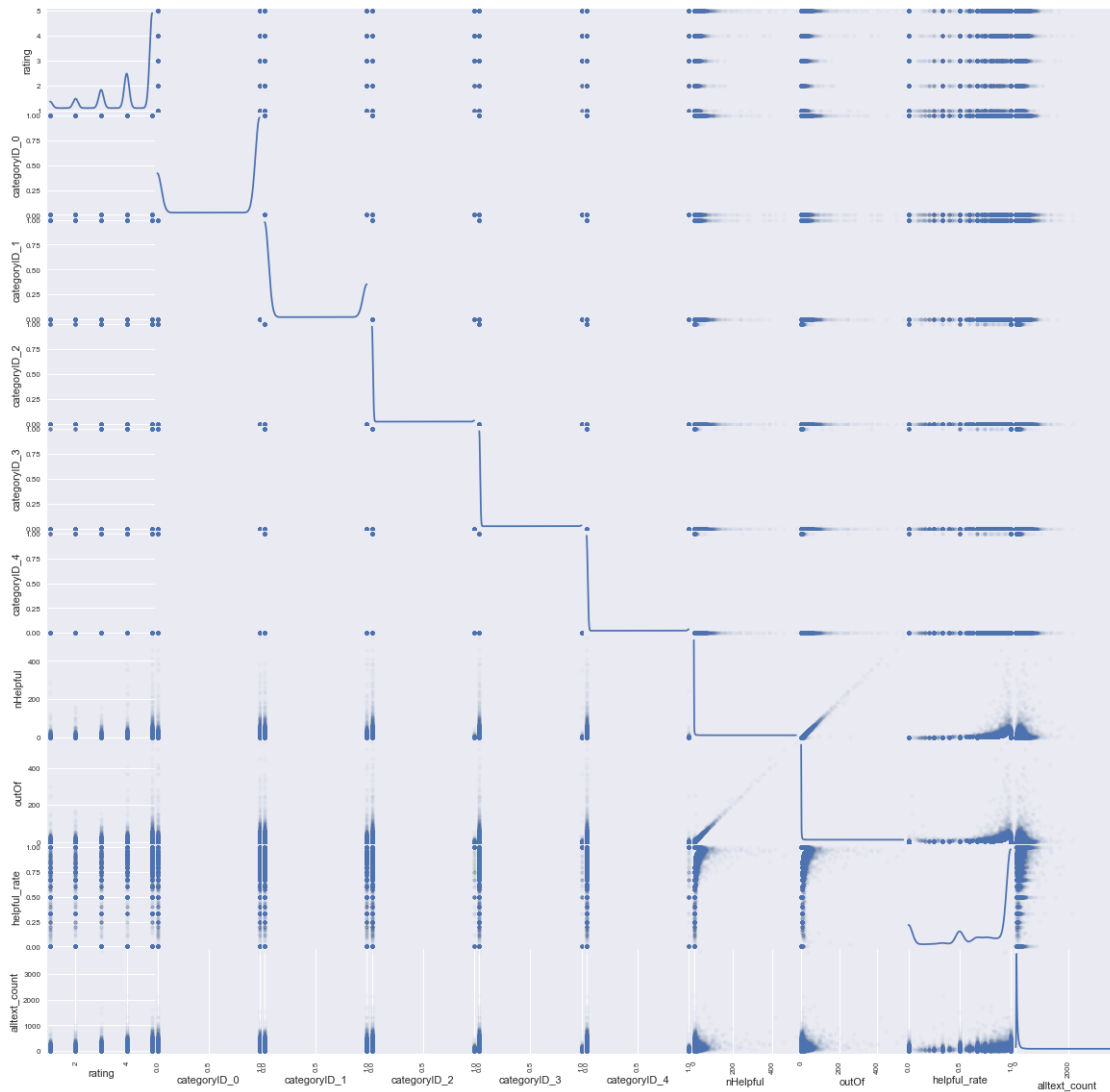


```
In [118]: mat = train.corr() # to get a heatmap of the correlation matrix
a = sns.heatmap(mat, vmax=1, square = True)
a.set_title('Correlation heatmap for selected feature.')
```

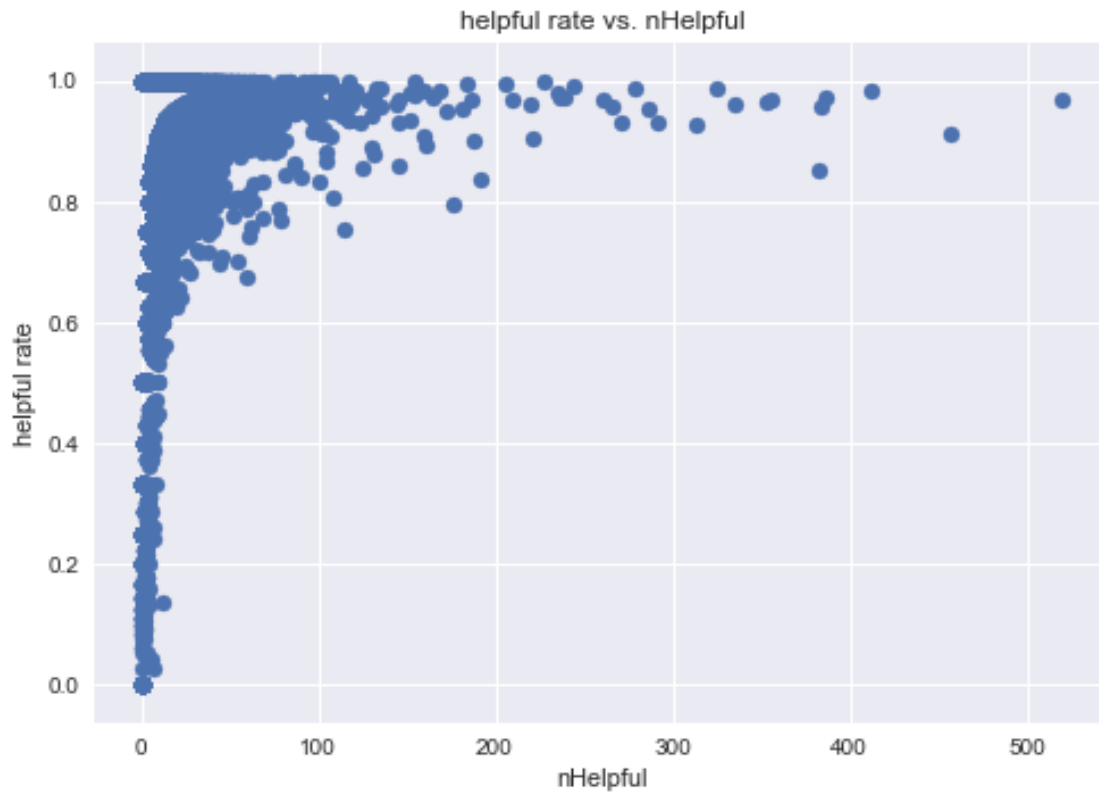
```
Out[118]: <matplotlib.text.Text at 0x21ca7793eb8>
```



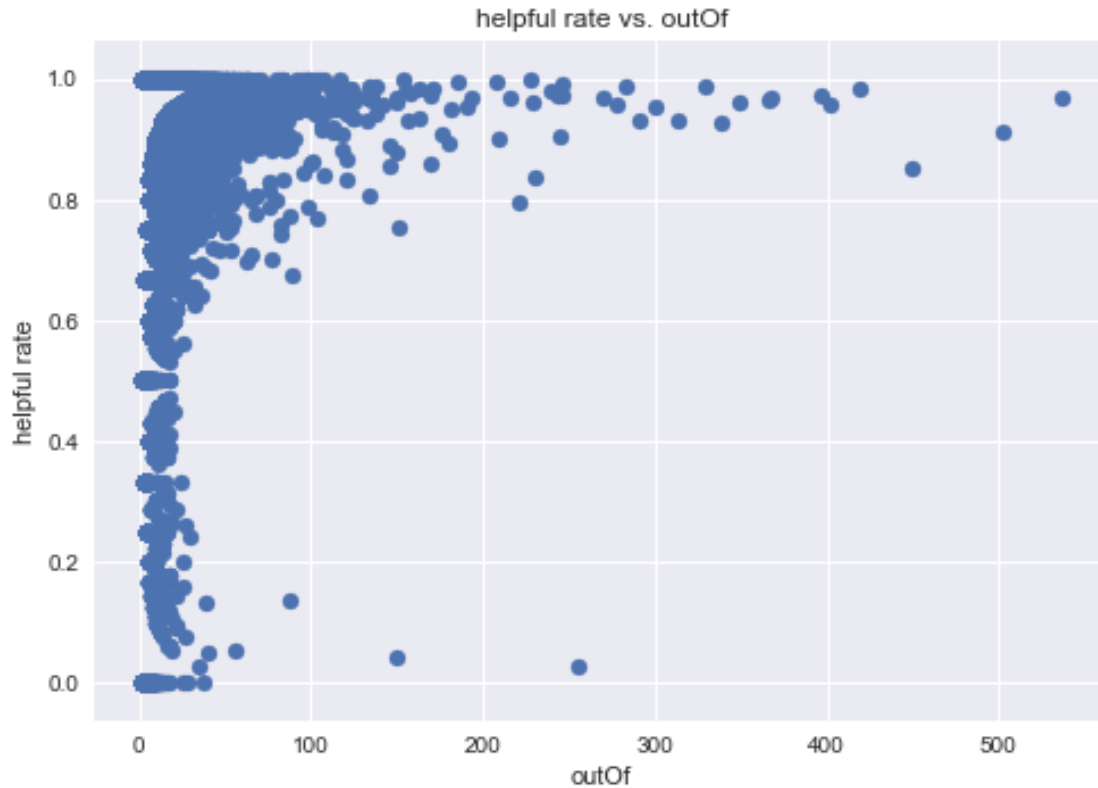
```
In [122]: from pandas.tools.plotting import scatter_matrix
scatter_matrix(train, alpha=0.03, figsize=(20, 20), diagonal='kde');
```



```
In [127]: plt.scatter(train['nHelpful'], train['helpful_rate']);
plt.title('helpful rate vs. nHelpful');
plt.xlabel('nHelpful');
plt.ylabel('helpful rate');
```



```
In [128]: plt.scatter(train['outOf'], train['helpful_rate']);  
plt.title('helpful rate vs. outOf');  
plt.xlabel('outOf');  
plt.ylabel('helpful rate');
```



## 2.6 Modeling

```
In [130]: from sklearn.grid_search import GridSearchCV
          from sklearn import ensemble
          from sklearn.cross_validation import cross_val_score, cross_val_predict,
          from sklearn.metrics import mean_absolute_error
```

```
In [163]: from sklearn.model_selection import train_test_split
```

```
def data_to_model(dataframe, test_train, features, class_attribute, random_state):
    """The function prepares the train & test dataframes for modeling."""
    if test_train == 'train':
        X_train = pd.DataFrame(dataframe, columns=features)
        y_train = pd.DataFrame(dataframe[class_attribute])
        X_train1, X_valid, y_train1, y_valid = train_test_split(X_train,
                                                                y_train,
                                                                return_X_test=False,
                                                                random_state=random_state)
        return X_train, y_train, X_train1, y_train1, X_valid, y_valid
    elif test_train == 'test':
        X_test = pd.DataFrame(dataframe, columns=features)
        return X_test
    else:
        pass
```

```

In [346]: import warnings
          warnings.filterwarnings('ignore')

mae = []
mae_rounding = []
columns = ['rating', 'outOf', 'categoryID_0', 'categoryID_1', 'categoryID_2']
iteration = 0
for i in range(10):
    print('\n ***** For round', iteration, 'validation set.')
    #For outOf ==0, always remove 0 therefore
    X_train, y_train, X_train1, y_train1, X_valid, y_valid = data_to_model(X, y, columns)

    # For outOf == 1 binary classifier: Gradient Boosting Classifier
    one_X_train = X_train1[X_train1['outOf'] == 1 ]
    one_y_train = np.array(y_train1['helpful_rate'][X_train1['outOf'] == 1])
    one_kf = StratifiedKFold(one_y_train, n_folds=10, shuffle=True, random_state=0)
    gridparams = dict(learning_rate=[0.01, 0.1], loss=['deviance', 'exponential'])
    params = {'n_estimators': 100, 'max_depth': 4}
    gbclf = GridSearchCV(ensemble.GradientBoostingClassifier(**params), gridparams)
    gbclf = GridSearchCV(ensemble.GradientBoostingClassifier(**params), gridparams)
    gbclf.fit(one_X_train, one_y_train)
    print("Best model:")
    print(gbclf.best_estimator_)
    print("")

    # Rest of data regressor: Gradient Boosting Regressor
    rest_X_train = X_train1[(X_train1['outOf'] != 1) & (X_train1['outOf'] != 2)]
    rest_y_train = np.array(y_train1['helpful_rate'][(X_train1['outOf'] != 1) & (X_train1['outOf'] != 2)])
    rest_kf = StratifiedKFold(rest_y_train, n_folds=10, shuffle=True, random_state=0)
    gridparams = dict(learning_rate=[0.01, 0.1], loss=['ls', 'lad'])
    params = {'n_estimators': 100, 'max_depth': 4}
    gbreg = GridSearchCV(ensemble.GradientBoostingRegressor(**params), gridparams)
    gbreg = GridSearchCV(ensemble.GradientBoostingRegressor(**params), gridparams)
    gbreg.fit(rest_X_train, rest_y_train)
    print("Best model:")
    print(gbreg.best_estimator_)
    print("")

    # Predicting on the validation set:
    y_pred = {}
    X_valid1 = X_valid.reset_index(drop = True)
    for j in range(len(X_valid1)):
        if X_valid1['outOf'][j] == 0:
            y_pred[j] = 0
        elif X_valid1['outOf'][j] == 1:
            y_pred[j] = (gbclf.predict(np.array(X_valid1.iloc[j]))[0]) * (X_valid1['rating'][j])
        else:
            y_pred[j] = (gbreg.predict(np.array(X_valid1.iloc[j]))[0]) * (X_valid1['rating'][j])

```



```

print("Without rounding, mean absolute error: %0.3f" % mean_absolute_
mae.append(mean_absolute_error(list(y_valid['nHelpful']), list(y_pred
print("With rounding, mean absolute error: %0.3f" % mean_absolute_err
mae_rounding.append(mean_absolute_error(list(y_valid['nHelpful']), np
iteration += 1

print('\n Using the following features: ', columns)
print('Without rounding, "mean" mae: %0.3f' % mean(mae))
print('With rounding, "mean" mae: %0.3f' % mean(mae_rounding))

```

\*\*\*\*\* For round 0 validation set. \*\*\*\*\*

Best model:

```

GradientBoostingClassifier(criterion='friedman_mse', init=None,
learning_rate=0.01, loss='deviance', max_depth=4,
max_features=None, max_leaf_nodes=None,
min_impurity_split=1e-07, min_samples_leaf=1,
min_samples_split=2, min_weight_fraction_leaf=0.0,
n_estimators=100, presort='auto', random_state=None,
subsample=1.0, verbose=0, warm_start=False)

```

Best model:

```

GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
learning_rate=0.1, loss='lad', max_depth=4, max_features=None,
max_leaf_nodes=None, min_impurity_split=1e-07,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=100,
presort='auto', random_state=None, subsample=1.0, verbose=0,
warm_start=False)

```

Without rounding, mean absolute error: 0.163

With rounding, mean absolute error: 0.162

\*\*\*\*\* For round 1 validation set. \*\*\*\*\*

Best model:

```

GradientBoostingClassifier(criterion='friedman_mse', init=None,
learning_rate=0.1, loss='deviance', max_depth=4,
max_features=None, max_leaf_nodes=None,
min_impurity_split=1e-07, min_samples_leaf=1,
min_samples_split=2, min_weight_fraction_leaf=0.0,
n_estimators=100, presort='auto', random_state=None,
subsample=1.0, verbose=0, warm_start=False)

```

Best model:

```

GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
learning_rate=0.1, loss='lad', max_depth=4, max_features=None,
max_leaf_nodes=None, min_impurity_split=1e-07,

```

```
min_samples_leaf=1, min_samples_split=2,  
min_weight_fraction_leaf=0.0, n_estimators=100,  
presort='auto', random_state=None, subsample=1.0, verbose=0,  
warm_start=False)
```

Without rounding, mean absolute error: 0.170

With rounding, mean absolute error: 0.169

\*\*\*\*\* For round 2 validation set. \*\*\*\*\*

Best model:

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,  
                           learning_rate=0.1, loss='deviance', max_depth=4,  
                           max_features=None, max_leaf_nodes=None,  
                           min_impurity_split=1e-07, min_samples_leaf=1,  
                           min_samples_split=2, min_weight_fraction_leaf=0.0,  
                           n_estimators=100, presort='auto', random_state=None,  
                           subsample=1.0, verbose=0, warm_start=False)
```

Best model:

```
GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,  
                           learning_rate=0.1, loss='lad', max_depth=4, max_features=None,  
                           max_leaf_nodes=None, min_impurity_split=1e-07,  
                           min_samples_leaf=1, min_samples_split=2,  
                           min_weight_fraction_leaf=0.0, n_estimators=100,  
                           presort='auto', random_state=None, subsample=1.0, verbose=0,  
                           warm_start=False)
```

Without rounding, mean absolute error: 0.169

With rounding, mean absolute error: 0.168

\*\*\*\*\* For round 3 validation set. \*\*\*\*\*

Best model:

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,  
                           learning_rate=0.1, loss='deviance', max_depth=4,  
                           max_features=None, max_leaf_nodes=None,  
                           min_impurity_split=1e-07, min_samples_leaf=1,  
                           min_samples_split=2, min_weight_fraction_leaf=0.0,  
                           n_estimators=100, presort='auto', random_state=None,  
                           subsample=1.0, verbose=0, warm_start=False)
```

Best model:

```
GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,  
                           learning_rate=0.1, loss='lad', max_depth=4, max_features=None,  
                           max_leaf_nodes=None, min_impurity_split=1e-07,  
                           min_samples_leaf=1, min_samples_split=2,  
                           min_weight_fraction_leaf=0.0, n_estimators=100,  
                           presort='auto', random_state=None, subsample=1.0, verbose=0,  
                           warm_start=False)
```

Without rounding, mean absolute error: 0.162

With rounding, mean absolute error: 0.160

\*\*\*\*\* For round 4 validation set. \*\*\*\*\*

Best model:

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=4,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_split=1e-07, min_samples_leaf=1,
                           min_samples_split=2, min_weight_fraction_leaf=0.0,
                           n_estimators=100, presort='auto', random_state=None,
                           subsample=1.0, verbose=0, warm_start=False)
```

Best model:

```
GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='lad', max_depth=4, max_features=None,
                           max_leaf_nodes=None, min_impurity_split=1e-07,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           presort='auto', random_state=None, subsample=1.0, verbose=0,
                           warm_start=False)
```

Without rounding, mean absolute error: 0.174

With rounding, mean absolute error: 0.173

\*\*\*\*\* For round 5 validation set. \*\*\*\*\*

Best model:

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.01, loss='deviance', max_depth=4,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_split=1e-07, min_samples_leaf=1,
                           min_samples_split=2, min_weight_fraction_leaf=0.0,
                           n_estimators=100, presort='auto', random_state=None,
                           subsample=1.0, verbose=0, warm_start=False)
```

Best model:

```
GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='lad', max_depth=4, max_features=None,
                           max_leaf_nodes=None, min_impurity_split=1e-07,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           presort='auto', random_state=None, subsample=1.0, verbose=0,
                           warm_start=False)
```

Without rounding, mean absolute error: 0.169

With rounding, mean absolute error: 0.168

\*\*\*\*\* For round 6 validation set. \*\*\*\*\*

Best model:

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.01, loss='deviance', max_depth=4,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_split=1e-07, min_samples_leaf=1,
                           min_samples_split=2, min_weight_fraction_leaf=0.0,
                           n_estimators=100, presort='auto', random_state=None,
                           subsample=1.0, verbose=0, warm_start=False)
```

Best model:

```
GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='lad', max_depth=4, max_features=None,
                           max_leaf_nodes=None, min_impurity_split=1e-07,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           presort='auto', random_state=None, subsample=1.0, verbose=0,
                           warm_start=False)
```

Without rounding, mean absolute error: 0.169

With rounding, mean absolute error: 0.169

\*\*\*\*\* For round 7 validation set. \*\*\*\*\*

Best model:

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=4,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_split=1e-07, min_samples_leaf=1,
                           min_samples_split=2, min_weight_fraction_leaf=0.0,
                           n_estimators=100, presort='auto', random_state=None,
                           subsample=1.0, verbose=0, warm_start=False)
```

Best model:

```
GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='lad', max_depth=4, max_features=None,
                           max_leaf_nodes=None, min_impurity_split=1e-07,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           presort='auto', random_state=None, subsample=1.0, verbose=0,
                           warm_start=False)
```

Without rounding, mean absolute error: 0.166

With rounding, mean absolute error: 0.165

\*\*\*\*\* For round 8 validation set. \*\*\*\*\*

Best model:

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.01, loss='deviance', max_depth=4,
```

```

max_features=None, max_leaf_nodes=None,
min_impurity_split=1e-07, min_samples_leaf=1,
min_samples_split=2, min_weight_fraction_leaf=0.0,
n_estimators=100, presort='auto', random_state=None,
subsample=1.0, verbose=0, warm_start=False)

```

Best model:

```

GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
learning_rate=0.1, loss='lad', max_depth=4, max_features=None,
max_leaf_nodes=None, min_impurity_split=1e-07,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=100,
presort='auto', random_state=None, subsample=1.0, verbose=0,
warm_start=False)

```

Without rounding, mean absolute error: 0.174

With rounding, mean absolute error: 0.173

\*\*\*\*\* For round 9 validation set. \*\*\*\*\*

Best model:

```

GradientBoostingClassifier(criterion='friedman_mse', init=None,
learning_rate=0.1, loss='deviance', max_depth=4,
max_features=None, max_leaf_nodes=None,
min_impurity_split=1e-07, min_samples_leaf=1,
min_samples_split=2, min_weight_fraction_leaf=0.0,
n_estimators=100, presort='auto', random_state=None,
subsample=1.0, verbose=0, warm_start=False)

```

Best model:

```

GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
learning_rate=0.1, loss='lad', max_depth=4, max_features=None,
max_leaf_nodes=None, min_impurity_split=1e-07,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=100,
presort='auto', random_state=None, subsample=1.0, verbose=0,
warm_start=False)

```

Without rounding, mean absolute error: 0.175

With rounding, mean absolute error: 0.174

Using the following features: ['rating', 'outOf', 'categoryID\_0', 'categoryID\_1',  
Without rounding, "mean" mae: 0.169  
With rounding, "mean" mae: 0.168

Predict on test data set: - Method: if X\_train1['outOf'] == 0 then y\_pred == 0, elif  
X\_train1['outOf'] == 1 then perform GradientBoostingClassifier, else perform GradientBoostin-  
gRegresor - Using the following features: ['rating', 'outOf', 'categoryID\_0', 'categoryID\_1', 'cate-

goryID\_2', 'categoryID\_3', 'categoryID\_4'] - Without rounding, "mean" mae: 0.169, - With rounding, "mean" mae: 0.168

```
In [347]: ### Predict on test data set: Without rounding, "mean" mae: 0.169, With rounding, "mean" mae: 0.168
warnings.filterwarnings('ignore')

one_X_train = X_train[X_train['outOf'] == 1 ]
one_y_train = np.array(y_train['helpful_rate'][X_train['outOf'] == 1])
gbclf = ensemble.GradientBoostingClassifier(criterion='friedman_mse', initial
learning_rate=0.1, loss='deviance', max_depth=4,
max_features=None, max_leaf_nodes=None,
min_impurity_split=1e-07, min_samples_leaf=1,
min_samples_split=2, min_weight_fraction_leaf=0.0,
n_estimators=100, presort='auto', random_state=None,
subsample=1.0, verbose=0, warm_start=False)
gbclf.fit(one_X_train, one_y_train)

rest_X_train = X_train[(X_train['outOf'] != 1) & (X_train['outOf'] != 0)]
rest_y_train = np.array(y_train['helpful_rate'][(X_train['outOf'] != 1) &
gbreg = ensemble.GradientBoostingRegressor(alpha=0.9, criterion='friedman
learning_rate=0.1, loss='lad', max_depth=4, max_features=None,
max_leaf_nodes=None, min_impurity_split=1e-07,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=100,
presort='auto', random_state=None, subsample=1.0, verbose=0,
warm_start=False)
gbreg.fit(rest_X_train, rest_y_train)

X_test = pd.DataFrame(test, columns=columns)
y_pred = {}
for j in range(len(X_test)):
    if X_test['outOf'][j] == 0:
        y_pred[j] = 0
    elif X_test['outOf'][j] == 1:
        y_pred[j] = (gbclf.predict(np.array(X_test.iloc[j]))[0]) * (X_test
    else:
        y_pred[j] = (gbreg.predict(np.array(X_test.iloc[j]))[0]) * (X_test

In [353]: def predictions_exported(test, y_pred, rounding_on, output_file):
    predictions = []
    test_user_id = list(test['reviewerID'])
    test_item_id = list(test['itemID'])
    outOf = list(test['outOf'])

    if rounding_on == True:
        predictions = np.round(list(y_pred.values()), 0)
    else:
        predictions = list(y_pred.values())
```

```

output = open(output_file, 'w')
output.write('userID-itemID-outOf,prediction\n')
for i in range(len(predictions)):
    user = test_user_id[i]
    item = test_item_id[i]
    outof = outOf[i]
    prediction = predictions[i]
    output.write(user + '-' + item + '-' + str(outof) + ',' + str(pre

In [355]: predictions_exported(test, y_pred, True, 'ensemble_gb_0.168_rounding1_6.1
          predictions_exported(test, y_pred, False, 'ensemble_gb_0.169_norounding1_

In [ ]:

```

# Final Model 2

June 12, 2017

## 1 Final: Amazon Product Review Kaggle Competition

## 2 Model 2

### 2.1 DSE 220: Machine Learning

### 2.2 Due Date: 11 June 2017

### 2.3 Orysyia Stus

```
In [ ]: import pandas as pd
import numpy as np
from collections import defaultdict
import gzip

def readGz(f):
    for l in gzip.open(f):
        yield eval(l)

def parse(path):
    g = gzip.open(path, 'rb')
    for l in g:
        yield eval(l)

def getDF(path):
    i = 0
    df = {}
    for d in parse(path):
        df[i] = d
        i += 1
    return pd.DataFrame.from_dict(df, orient='index')

train_df = getDF('train.json.gz')
test_df = getDF('test_Helpful.json.gz')

In [2]: print(train_df.shape)

(200000, 12)
```



```
In [3]: train_df.isnull().sum()
```

```
Out[3]: categoryID          0
        categories          0
        itemID              0
        reviewerID          0
        rating              0
        reviewText          0
        reviewHash          0
        reviewTime          0
        summary             0
        unixReviewTime      0
        helpful             0
        price               125851
        dtype: int64
```

```
In [4]: print('Column price might need to be dropped since', train_df['price'].isnull().sum())
Column price might need to be dropped since 62.9255 % of the data is null.
```

```
In [5]: print('Delete column reviewHash since', train_df['reviewHash'].nunique(), 'exists')
del train_df['reviewHash']
```

Delete column reviewHash since 200000 unique values, out of 200000 exists.

### 2.3.1 Data Pre-Processing & Feature Engineering

Data Cleaning Done - deleted 'reviewHash' because it is completely unique - create helpful\_rate (if 0/0 then fillna with 0), delete 'helpful' - convert 'reviewTime' to datetime - delete 'unixReviewTime' since 'reviewTime' is present - created review\_keyText dictionary -> removed punctuation, stopwords, list of remaining words - created review\_allText dictionary -> list of all words - created review\_keyText\_count dictionary -> length of each wordlist in review\_keyText - created review\_allText\_count dictionary -> length of each wordlist in review\_allText - created a vocabulary list comprised of the set of review\_keyText - created a dataframe called itemID\_RPD -> calculates the reviews per day for each itemID - created a dataframe called reviewerID\_RPD -> calculates the reviews per day for each reviewerID

To Do - determine which users with best nHelpful count use which words -> count the number of words in each reviewText - see distribution of helpful\_rate via histogram - match itemID either by itemID if not present then via categories, price ie. NearestNeighbor - determine which words are used through distribution of helpful\_rate - build a dataframe for item which contains the categoryID, categories (binary w/dummy variables), price

```
In [6]: a = pd.DataFrame.from_dict(dict(train_df['helpful']))
        train_df1 = pd.concat([train_df, a], axis=1)
        train_df1['helpful_rate'] = train_df1['nHelpful']/train_df1['outOf']
        train_df1['helpful_rate'].fillna(0, inplace=True)
        del train_df1['helpful']
        train_df1['reviewTime'] = pd.to_datetime(train_df1['reviewTime'])
        del train_df1['unixReviewTime']
```

```

In [7]: from nltk.corpus import stopwords
import string

def reviewText_listed(row):
    all_words = row.split()
    all_words = [w.lower() for w in all_words]
    subset_list = [''.join(c for c in s if c not in string.punctuation) for s in all_words]
    subset_list = [w for w in subset_list if w != '']
    subset_list = [word for word in subset_list if word not in stopwords.words('english')]
    return all_words, len(all_words), subset_list, len(subset_list)

review_allText = {}
review_allText_count = {}
review_keyText = {}
review_keyText_count = {}
vocabulary = []
count = 0
reviewText = list(train_df1['reviewText'])
for text in reviewText:
    all_, all_count, subset_, subset_count = reviewText_listed(text)
    review_allText[count] = all_
    review_allText_count[count] = all_count
    review_keyText[count] = subset_
    review_keyText_count[count] = subset_count
    vocabulary.append(subset_)
    count += 1

In [8]: from itertools import chain
vocabulary = set(list(chain.from_iterable(vocabulary)))

In [9]: print('There are', train_df1['reviewerID'].nunique(), 'unique reviewerIDs out of',
            train_df1['reviewerID'].nunique(), 'total reviewerIDs')
print('There are', train_df1['itemID'].nunique(), 'unique itemIDs out of',
            train_df1['itemID'].nunique(), 'total itemIDs')

There are 39249 unique reviewerIDs out of 200000 training records.
There are 19913 unique itemIDs out of 200000 training records.

In [10]: def RPD(row):
    if (row['max'] - row['min']).days == 0:
        return 0
    else:
        return row['count'] / (row['max'] - row['min']).days

In [11]: rt_count = train_df1.groupby('reviewerID')['reviewTime'].count()
rt_max = train_df1.groupby('reviewerID')['reviewTime'].max()
rt_min = train_df1.groupby('reviewerID')['reviewTime'].min()

reviewerID_RDP = pd.concat([rt_count, rt_max, rt_min], axis=1, join="inner")
reviewerID_RDP.columns.values[0] = 'count'

```

```
reviewerID_RDP.columns.values[1] = 'max'
reviewerID_RDP.columns.values[2] = 'min'

reviewerID_RDP['reviewerID_RPD'] = reviewerID_RDP.apply(RPD, axis=1)
reviewerID_RDP.head()
```

```
Out[11]:
```

	count	max	min	reviewerID_RPD
reviewerID				
U000005418	13	2014-01-25	2011-12-29	0.017150
U000025708	7	2013-12-03	2013-05-30	0.037433
U000095100	20	2014-06-07	2012-02-14	0.023697
U000129529	3	2014-01-01	2013-12-31	3.000000
U000130531	7	2013-08-15	2013-03-29	0.050360

```
In [12]: rt_count = train_df1.groupby('itemID')['reviewTime'].count()
rt_max = train_df1.groupby('itemID')['reviewTime'].max()
rt_min = train_df1.groupby('itemID')['reviewTime'].min()

itemID_RPD = pd.concat([rt_count, rt_max, rt_min], axis=1, join="inner")
itemID_RPD.columns.values[0] = 'count'
itemID_RPD.columns.values[1] = 'max'
itemID_RPD.columns.values[2] = 'min'

itemID_RPD['itemID_RPD'] = itemID_RPD.apply(RPD, axis=1)
itemID_RPD.head()
```

```
Out[12]:
```

	count	max	min	itemID_RPD
itemID				
I000059267	4	2014-02-26	2010-12-21	0.003439
I000139473	9	2014-04-10	2013-02-20	0.021739
I000159068	7	2014-02-09	2013-01-03	0.017413
I000235837	24	2014-07-02	2010-01-26	0.014833
I000384418	13	2014-05-24	2010-06-26	0.009104

```
In [13]: train_df1.head(3)
```

```
Out[13]:
```

	categoryID	categories	itemID
0	0	[[Clothing, Shoes & Jewelry, Women], [Clothing, Shoes & Jewelry, Women, Clothing, ...	I6553553
1	0	[[Clothing, Shoes & Jewelry, Women, Clothing, ...	I2410923
2	0	[[Clothing, Shoes & Jewelry, Wedding Party Gif...	I4082608

	reviewerID	rating	reviewText
0	U745881038	3.0	These are cute, but they are a little small. ...
1	U023577405	4.0	I love the look of this bra, it is what I want...
2	U441384838	3.0	it's better on a man's hand.I didn't find it v...

	reviewTime	summary	price	nHelpful	outOf	\
0	2014-05-20	Cute	NaN	0	0	
1	2013-02-07	Beautiful but size runs small	NaN	0	0	

2	2014-05-13	Good price but...	19.99	2	2
---	------------	-------------------	-------	---	---

	helpful_rate
0	0.0
1	0.0
2	1.0

In [14]: train\_df1.shape

Out[14]: (200000, 12)

In [62]: review\_keyword\_length = pd.DataFrame.from\_dict(review\_keyText\_count, orient='columns')  
review\_allword\_length = pd.DataFrame.from\_dict(review\_allText\_count, orient='columns')

In [32]: categories = list(train\_df1['categories'])  
g = list(chain.from\_iterable(categories))  
print('There are', len(set(list(chain.from\_iterable(g)))), 'unique categories')

There are 1042 unique categories values

In [34]: print('There are', train\_df1['categoryID'].nunique(), 'unique categoriesID')

Out[34]: 5

In [295]: dummies = pd.get\_dummies(train\_df1['categoryID']).rename(columns=lambda x: x + '\_categoryID')  
# master = pd.concat([train\_df1, dummies], axis=1)  
master = pd.concat([train\_df1, dummies, review\_keyword\_length, review\_allword\_length], axis=1)  
master.columns.values[17] = 'review\_content\_len'  
master.columns.values[18] = 'review\_all\_len'  
master['review\_content\_ratio'] = master['review\_content\_len']/master['review\_all\_len']  
del master['categoryID']  
master.head(3)

Out[295]:

	categories	itemID	reviewTime
0	[[Clothing, Shoes & Jewelry, Women], [Clothing, Shoes & Jewelry, Women, Clothing, ...	I655355328	U745881
1	[[Clothing, Shoes & Jewelry, Women, Clothing, ...	I241092314	U023577
2	[[Clothing, Shoes & Jewelry, Wedding Party Gif...	I408260822	U441384

	rating	reviewText	reviewTime
0	3.0	These are cute, but they are a little small.	2014-05-20
1	4.0	I love the look of this bra, it is what I want...	2013-02-07
2	3.0	it's better on a man's hand.I didn't find it v...	2014-05-13

	summary	price	nHelpful	outOf	helpful_rate
0	Cute	NaN	0	0	0.0
1	Beautiful but size runs small	NaN	0	0	0.0
2	Good price but...	19.99	2	2	1.0

	categoryID_0	categoryID_1	categoryID_2	categoryID_3	categoryID_4
0	1	0	0	0	0
1	1	0	0	0	0
2	1	0	0	0	0

	review_content_len	review_all_len	review_contentratio
0	10	24	0.416667
1	25	57	0.438596
2	15	28	0.535714

```
In [281]: # create a model just using rating, outOf, categoryID_0-5; predict on nH
# obviously is outOf == 0 then nHelpful == 0, do not need to model this
```

```
In [282]: model = master[master['outOf'] != 0]
print('With all data', master.shape, ', when outOf != 0', model.shape)
```

With all data (200000, 19) , when outOf != 0 (63016, 19)

```
In [322]: columns = ['rating', 'outOf', 'categoryID_0', 'categoryID_1', 'categoryID_2', 'categoryID_3', 'categoryID_4', 'categoryID_5', 'categoryID_6', 'categoryID_7', 'categoryID_8', 'categoryID_9', 'categoryID_10', 'categoryID_11', 'categoryID_12', 'categoryID_13', 'categoryID_14', 'categoryID_15', 'categoryID_16', 'categoryID_17', 'categoryID_18', 'categoryID_19', 'categoryID_20', 'categoryID_21', 'categoryID_22', 'categoryID_23', 'categoryID_24', 'categoryID_25', 'categoryID_26', 'categoryID_27', 'categoryID_28', 'categoryID_29', 'categoryID_30', 'categoryID_31', 'categoryID_32', 'categoryID_33', 'categoryID_34', 'categoryID_35', 'categoryID_36', 'categoryID_37', 'categoryID_38', 'categoryID_39', 'categoryID_40', 'categoryID_41', 'categoryID_42', 'categoryID_43', 'categoryID_44', 'categoryID_45', 'categoryID_46', 'categoryID_47', 'categoryID_48', 'categoryID_49', 'categoryID_50', 'categoryID_51', 'categoryID_52', 'categoryID_53', 'categoryID_54', 'categoryID_55', 'categoryID_56', 'categoryID_57', 'categoryID_58', 'categoryID_59', 'categoryID_60', 'categoryID_61', 'categoryID_62', 'categoryID_63', 'categoryID_64', 'categoryID_65', 'categoryID_66', 'categoryID_67', 'categoryID_68', 'categoryID_69', 'categoryID_70', 'categoryID_71', 'categoryID_72', 'categoryID_73', 'categoryID_74', 'categoryID_75', 'categoryID_76', 'categoryID_77', 'categoryID_78', 'categoryID_79', 'categoryID_80', 'categoryID_81', 'categoryID_82', 'categoryID_83', 'categoryID_84', 'categoryID_85', 'categoryID_86', 'categoryID_87', 'categoryID_88', 'categoryID_89', 'categoryID_90', 'categoryID_91', 'categoryID_92', 'categoryID_93', 'categoryID_94', 'categoryID_95', 'categoryID_96', 'categoryID_97', 'categoryID_98', 'categoryID_99']
X_train = pd.DataFrame(model, columns=columns)
y_train = pd.DataFrame(model.ix[:, 'nHelpful'])
```

```
In [323]: from sklearn.model_selection import train_test_split
X_train1, X_valid, y_train1, y_valid = train_test_split(X_train, y_train,
```

```
In [299]: ### Test data
a = pd.DataFrame.from_dict(dict(test_df['helpful'])).T
test_df1 = pd.concat([test_df, a], axis=1)
```

```
review1_allText = {}
review1_allText_count = {}
review1_keyText = {}
review1_keyText_count = {}
vocabulary1 = []
count = 0
reviewText = list(test_df1['reviewText'])
for text in reviewText:
    all_, all_count, subset_, subset_count = reviewText_listed(text)
    review1_allText[count] = all_
    review1_allText_count[count] = all_count
    review1_keyText[count] = subset_
    review1_keyText_count[count] = subset_count
    vocabulary1.append(subset_)
    count += 1
```

```
In [300]: review1_keyword_length = pd.DataFrame.from_dict(review1_keyText_count, or
review1_allword_length = pd.DataFrame.from_dict(review1_allText_count, or
```

```
In [324]: dummies = pd.get_dummies(test_df1['categoryID']).rename(columns=lambda x:
master = pd.concat([test_df1, dummies, review1_keyword_length, review1_al
master.columns.values[18] = 'review_content_len'
master.columns.values[19] = 'review_all_len'
master['review_content_ratio'] = master['review_content_len']/master['revi
del master['categoryID']
X_test = pd.DataFrame(master, columns=columns)
X_test.head()
```

```
Out[324]:
```

	rating	outOf	categoryID_0	categoryID_1	categoryID_2	categoryID_3
0	3.0	2	1	0	0	0
1	4.0	0	1	0	0	0
2	5.0	1	1	0	0	0
3	5.0	1	1	0	0	0
4	4.0	0	1	0	0	0

	categoryID_4	review_all_len
0	0	27
1	0	27
2	0	23
3	0	135
4	0	38

```
In [311]: modell = X_test[X_test['outOf'] != 0]
print('With all data', X_test.shape, ', when outOf != 0', modell.shape)
```

With all data (14000, 8) , when outOf != 0 (4400, 8)

Really have a training size of 63,016 to predict 4,400 values.

### 3 Gradient Boosting Regression

- without rounding 0.16543
- with rounding 0.16271

```
In [312]: from sklearn.grid_search import GridSearchCV
from sklearn import ensemble
from sklearn.metrics import mean_absolute_error
```

```
In [ ]: # kf = StratifiedKFold(y, n_folds=10, random_state=None, shuffle=True)
gridparams = dict(learning_rate=[0.01, 0.1], loss=['ls', 'lad', 'huber', 'c
# gridparams = dict(learning_rate=[0.01, 0.1, 1, 10], loss=['ls', 'lad', 'h
params = {'n_estimators': 100, 'max_depth': 4}
gbclf = GridSearchCV(ensemble.GradientBoostingRegressor(**params), gridpara
# gbclf = GridSearchCV(ensemble.GradientBoostingRegressor(n_estimators= 200
gbclf.fit(X_train1, y_train1)
```

```

print("Best model:")
print(gbclf.best_estimator_)
print("")

y_pred = gbclf.predict(X_valid)
print("Mean absolute error: %0.3f" % mean_absolute_error(np.array(y_valid['

In [314]: gbreg1 = ensemble.GradientBoostingRegressor(alpha=0.9, criterion='friedma
            learning_rate=0.1, loss='lad', max_depth=4, max_features=None,
            max_leaf_nodes=None, min_impurity_split=1e-07,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100,
            presort='auto', random_state=None, subsample=1.0, verbose=0,
            warm_start=False)
            gbreg1.fit(X_train, y_train)

C:\Users\Orysy\Anaconda\envs\py36\lib\site-packages\sklearn\utils\validation.py:52
y = column_or_1d(y, warn=True)

Out[314]: GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
            learning_rate=0.1, loss='lad', max_depth=4, max_features=None,
            max_leaf_nodes=None, min_impurity_split=1e-07,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100,
            presort='auto', random_state=None, subsample=1.0, verbose=0,
            warm_start=False)

In [319]: gbreg1_predictions = []
            for i in range(len(X_test)):
                if X_test['outOf'][i] == 0:
                    gbreg1_predictions.append(0)
                else:
                    gbreg1_predictions.append(round(gbreg1.predict(X_test.ix[i])[0]))

C:\Users\Orysy\Anaconda\envs\py36\lib\site-packages\sklearn\utils\validation.py:39
DeprecationWarning)
C:\Users\Orysy\Anaconda\envs\py36\lib\site-packages\sklearn\utils\validation.py:39
DeprecationWarning)
C:\Users\Orysy\Anaconda\envs\py36\lib\site-packages\sklearn\utils\validation.py:39
DeprecationWarning)
C:\Users\Orysy\Anaconda\envs\py36\lib\site-packages\sklearn\utils\validation.py:39
DeprecationWarning)
C:\Users\Orysy\Anaconda\envs\py36\lib\site-packages\sklearn\utils\validation.py:39
DeprecationWarning)
C:\Users\Orysy\Anaconda\envs\py36\lib\site-packages\sklearn\utils\validation.py:39
DeprecationWarning)
C:\Users\Orysy\Anaconda\envs\py36\lib\site-packages\sklearn\utils\validation.py:39
DeprecationWarning)
C:\Users\Orysy\Anaconda\envs\py36\lib\site-packages\sklearn\utils\validation.py:39
DeprecationWarning)

```

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]



[illegible]













[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]





[illegible]





[illegible]



[illegible]











[illegible]













[illegible]

[illegible]

[illegible]











[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

```
C:\Users\Orysy\Anaconda\envs\py36\lib\site-packages\sklearn\utils\validation.py:39  
DeprecationWarning)
```

```
In [320]: # with rounding  
          predictions = open("predictions_gbreg_rounding_Helpful.csv", 'w')  
          predictions.write('userID-itemID-outOf,prediction\n')  
          for i in range(len(gbregl_predictions)):  
              user = test_user_id[i]  
              item = test_item_id[i]  
              outof = outOf[i]  
              prediction = gbregl_predictions[i]  
              predictions.write(user + '-' + item + '-' + str(outof) + ',' + str(pr  
  
In [ ]:
```