Final - Raul G. Martinez (PID: A12461871)

DSE 220: Machine Learning

Due Date: 06/12 11:59 PM

In [ ]:

Report Section

Abstract

This report aims to provide a Machine Learning approach to predict how useful an Amazon Review is going to be, the data used in this analysis contains a total of 200,000 reviews in the years of 2003 to 2014 from a total of 39,249 reviewers and 19,913 products. Natural language processing techniques were used to learn new features from the review body text and summary text data, in addition to feature engineering in many other numerical features. Then, a Random Forest Regressor Model was implemented to predict the ratio between the number of helpful votes with the total number of votes for each review; train and validation split was done 80% and 20% respectively. After generating a baseline model with three fundamental features (itemID, reviewerID, and outOf_feature), model optimization was achieved with feature selection in two ways. First, each new feature was tested individually along with the baseline features and their model performance was evaluated with MAE (Mean Absolute Error) and MSE (Mean Squared Error), and any feature performing worst than the baseline was excluded. Second, an ablation study was performed by removing one feature at time, starting with the highest MAE, and the group of features yielding the lowest MAE were selected for downstream optimization. Next, hyper-parameter tunning was performed on max_depth and n_estimators parameters and the model with the lowest MAE was again selected. After final model selection, the confusion matrix was computed to evaluate the model's ability to classify the number of helpful votes; precision and recal metrics were observed in detail. All in all, this analysis and implementation emphasizes on feature engineering and feature selection rather than model or hyper-parameter optimization. A total of 36 new features were generated, tested, and selected to provide an automated optimal performance where the best MAE was found to be 0.15479 for the train dataset and 0.16357 for 60% of the test dataset.

Data Preparation and Pruning

The training data consists of 200,000 entries inside the compressed file 'train.json.gz', while the test data is stored also in a compressed file named 'test_Helpful.json.gz' and it contains 14,000 entries. Both datasets contain a total of 12 raw features summarized in the table below (observations are based on training data):

Raw Feature	Observations
categoryID	unique values are 0, 1, 2, 3, and 4

Raw Feature	Observations
categories	there are 1042 unique categories (i.e. 'Active Hoodies') forming 1847 unique lists, each review has a collection of lists
itemID	there are a total of 19,913 unique items
reviewerID	there are a total of 39,249 unique reviewers
rating	values assigned are 1, 2, 3, 4, and 5. Their frequency increases monotonically with the number, 1 being the less frequent
reviewText	the min number of characters is 0 and the max is 22,646
reviewHash	each review has a unique hash ID
reviewTime	time goes from 2003 to 2014, the most reviews are observed in 2013 and 2014
summary	the min number of characters is 1 and the max is 201
unixReviewTime	there are 2,532 unique times
helpful	it contains a dictionary with 'outOf' and 'nHelpful'. Votes 0 and 1 comprise of 68.5% and 14.3% from the total number of reviews
price	62.9% of the reviews have missing values, they were filled with -999 in this analysis

Raw features were read in a dataframe for the train (raw) dataset (see table 1 in the report section) then split into train and validation, 80% and 20% respectively, in order to validate feature selection and model optimization. The newly defined train data was then pruned to only include reviews where the number of votes (helpful - outOf) is greater than zero, therefore, the original train data (after the split) was used to compute performance metrics while the newly filtered data was only used to train the model. This is because the entries equal to zero are always predicted zero by the regression model from multiplying predicted label times the known 'outOf' feature (with value zero). Better performance was found empirically when implementing this approach, it basically allows the model to learn on the remaining 31.5% of the data where the predictions are not given or easily predicted.

Lastly, the labels are extracted from the 'helpful' column and are defined as the ratio between 'nHelpful' and 'outOf'. Divisio by zero is prevented with the approach mentioned above of removing entries with votes equal to zero.

Feature Engineering

A total of 36 new features were generated as candidates to be included in the final model for test data predictions. The table below summarizes the new feature names (when applicable), the raw features used to generate it (when applicable), and lastly some description about it. Positive and negative word lists are taken from reference 1 and 2 below, and the list of stop words was downloaded from the NLTK library on python. A couple features were also inspired from reference 3.

Feature Name	Features Used From Raw Data	Description
itemID	NA	**Baseline Feature** - ID for each product or item
reviewerID	NA	**Baseline Feature** - ID for each reviewer or user
outOf_feature	helpful	**Baseline Feature** - Number of votes
categoryID	NA	IDs for the collection of category lists, the unique IDs are 0, 1, 2, 3, and 4

Feature Name	Features Used From Raw Data	Description
categories_count	categories	Number collection of category lists inside 'categories' column
category_numtrans	categories	Each category list gets a unique ID, then numeric transformation is applied by summation of IDs
rating	NA	Rating for the given review
rating_deviation	rating	Deviation from the mean rating for each product
itemID_helpfulRate	itemID, helpful	Helpful rate calculated for each item on labeled data
itemID_numReviews	itemID	Number of reviews for each product
reviewerID_helpfulRate	reviewerID, helpful	Helpful rate calculated for each reviewer on labeled data
reviewerID_numReviews	reviewerID	Number of reviews for each reviewer
price	NA	Price for the item being reviewed, missing values are assigned the value -999
reviewText_count_words (or summary)	reviewText, summary	Count for the number of words, stop words removed
reviewText_posWordCount (or summary)	reviewText, summary	Count for the number of positive words found, with pre-defined list of positive words
reviewText_negWordCount (or summary)	reviewText, summary	Count for the number of negative words found, with pre-defined list of negative words
reviewText_posWordRate (or summary)	reviewText, summary	Positive word rate for the text, with pre-defined list of positive words
reviewText_negWordRate (or summary)	reviewText, summary	Negative word rate for the text, with pre-defined list of negative words
reviewText_count_char (or summary)	reviewText, summary	Count for the number of characters
reviewText_count_punctu (or summary)	reviewText, summary	Count for the number of punctuation symbols
reviewText_count_firstCapital (or summary)	reviewText, summary	Count for the number of words where the first letter is capital
reviewText_avgWordLength (or summary)	reviewText, summary	Average word length for the text, stop words removed
reviewText_capitalwords (or summary)	reviewText, summary	Count for the number of capital words in the text
reviewText_ExclQue_countchar (or summary)	reviewText, summary	Count for the number of exclamation and question symbols in the text
reviewText_PunctChar_ratio (or summary)	reviewText, summary	Ratio for the count of punctuation symbols to characters in the text
summary_reviewText_charRatio	reviewText, summary	Ratio for the count of characters present in summary text by reviewText
summary_reviewText_wordsRatio	reviewText, summary	Ratio for the count of words present in summary text by reviewText
unixReviewTime	NA	Time elapsed in Unix time before the review was posted
unixReviewTime_delta_firstreview	unixReviewTime	Time elapsed in Unix time from first review for each item
unixReviewTime_delta_lastreview	unixReviewTime	Time elapsed in Unix time from the last review for each item
votes_time	helpful, unixReviewTime	Ratio between the number of votes and the time elapsed in Unix time

Model Selection

Multiple regression models were evaluated including Linear Regression, Multi-Later Perceptron Regressor, Decision Trees, and Random Forest Regressor from different sklearn libraries. After parameter tunning and different feature testing, the top performer and simpler to implement turned out to be Random Forest Regressor; data not shown in this report. The model is trained to predict a regressor label, which is the ratio between 'nHelpful' by 'outOf', for every individual review entry. The regressor label was then transformed to a classification label by multiplying it with the known 'outOf' values and rounding to the nearest integer; rounding was shown to improve MAE consistently (data not shown in the report). Accuracy was then calculated for the classification and only reported in the data table 2 from the report section. In summary, a simple approach was selected for model selection in this analysis in order to spend more time with feature engineering and feature selection, after all, experimenting with the features showed higher contribution for error predictions MAE and MSE as opposed to model optimization.

Feature Selection

A two step approach was taken to subselect features, both used random forest models with hyper-parameters 'max_depth'=10 and 'n_estimators'=100; these were found to be relatively stable and fast to test empirically. The first approach compares the performance of a baseline model with only three features (itemID, reviewerID, and outOf_feature) with other 4 feature models by combining the baseline features and adding one of the 40 feature candidates (new and existing features) at a time. The top performing models are then ranked according to lower MAE (ascending) on the validation dataset and the features whose models performed lower than the baseline model are eliminated. Table 2 in the code section illustrates the approach, in this example a total of 7 features were eliminated and 33 selected for downstream selection. Next, the second approach aims to continue feature subselection by performing an ablation study, where features are eliminated one-by-one and model performance is measured. The features previously selected on baseline comparison are ranked by MAE (descending) and one feature is eliminated at a time, starting with no features eliminated to the first with the highest MAE and further continuing to eliminate all features. Again, the criteria to select a group of features is the test case with the lowest MAE on the validation dataset. Figure 1 shows scatter plots for MAE and MSE, both error metrics trend similarly and also clearly illustrate how model performance decreases as the number of eliminated features increases, the lowest MAE for this datset occured when 11 were removed, therefore a total of 22 features were finally selected for downstream model evaluation.

Model Optimization and Evaluation

Many features were generated and then narrowed down to only subselect the ones with optimal performance. After selecting the final features, the same model used for testing above was then optimized by tuning the hyper-parameters 'max_depth' and 'n_estimators'; these parameters were found to be the most significant across 6 different ones tested by performing an expensive Randomized Search Cross-Validation with sklearn model_selection library (not shown in this report). The values 10 and 12 are tested for max_depth, and the values 100, 200, 400, and 600 for n_estimators. By testing every possible scenario, the optimal combination was found by taking the one with the lowest MAE on the validation dataset. To illustrate this approach, a printout from the model is shown in the 'Hyperparameter Tuning' part from the code section, where the optimal parameters were found to be 12 for 'max_depth' and 200 for 'n_estimators' with a score of 0.172575 for MAE on the validation dataset.

To further evaluate the model's ability to classify, the confusion matrix was computed for the optimal predictions on the validation dataset. More evaluation metrics for the classes were also calculated such as True Positive Rate (recall), True Negative Rate, Precision, False Positive Rate, False Negative Rate, and Accuracy. Table 3 in the code section summarizes the results for these metrics for a total of 122 different classes. Is important to note that some labels show NaN values as the prediction rate, since not all predicted labels exist in the dataset and are marked as False Positives, also how accuracy is not a good metric for this analysis as a value of greater than 0.9 for every feature does not seem to be real. Conversely, when looking at precision and recall (see Figure 2) there is something interesting happening, it basically shows how the performance of both metrics is well correlated with label value magnitudes; or possibly label ranges. For instance, labels with values less than 2 have really good precision and recall (with greater than 0.7) and it then decreases to approximately 0.1 to 0.5 for labels with values around 3 to 50, and finally almost every label greater than 50 has precision and recall of zero. This behavior is not very surprising and somewhat expected due to the nature of the predictions made in the regression model and data inconsistency for the distribution of number of votes in the training set; for example, larger number of votes for each review are significantly less common, and to put it in perspective there are only around 350 out of 200,000 reviews where the number of helpful votes is greater than 50. Additionally, the model is very good at predicting smaller values because of the rounding approach used for integers which basically reduces the error on the prediction, and again, performance progressively reduces with medium size values and it finally completely breaks down for larger value labels and is no longer useful. In summary, this model has the potential to generate good predictions when the number of votes is very small, however for predicting larger ranges such as 0 to 384 votes it will likely face limitations.

Conclusion

To summarize, a random forest regressor model was presented in this report to predict the number of helpful votes for Amazon reviews. A total of 36 features were learned, tested, and subselected by comparing model performance with baseline features and by performing an ablation study on a selected group of features; as a matter of fact, most of the time spent in this analysis was around the former two steps along with natural language processing, feature engineering, and data exploration. Next, model optimization was done with hyper-parameter tunning for two variables which are 'max_depth' and 'n_estimators'. And lastly, performance metrics such as precision and recall where used to assess model performance on the validation dataset. The final model obtained an MAE of 0.15479 for the training dataset and for 60% of the testing dataset the MAE was found to be 0.16357, the error for the remaining 40% of the test dataset is not reported in this analysis.

Future Steps

The approach presented above revealed a significant amount of new features and information from the raw data, however, the results seem to show that a single Random Forest Regressor is not sufficient for capturing most of the patterns in the data. Therefore, as next steps it would be useful to use multiple models, either implemented individually or together as an averaged prediction, that could possibly capture information better whether is through classification or regression. For example, a logistic regression model could be used to classify binary labels that occur very frequently such as 1 and 2 (not zero because it stays the same). Similarly, other models could be used to predict different ranges for the number of total votes such as 5 to 10 or even for greater than 50, depending on how they perform during testing. Lastly, the use of ensemble methods could also be considered for model optimization, some techniques widely used are bagging and boosting.

References

1. Minqing Hu and Bing Liu. "Mining and Summarizing Customer Reviews."Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2004), Aug 22-25, 2004, Seattle, Washington, USA.
2. Bing Liu, Minqing Hu and Junsheng Cheng. "Opinion Observer: Analyzing and Comparing Opinions on the Web." Proceedings of the 14th International World Wide Web conference (WWW-2005), May 10-14, 2005, Chiba, Japan.
3. Song, Xia. “Predict Amazon Review Helpfulness WihtXgboost, Neural Network, and LSTM Neural Network.” Medium, Medium, 11 Aug. 2019, medium.com/@songxia.sophia/predict-amazon-review-helpfulness-wihtxgboost-neural-network-and-lstm-neural-network-837a1da44f49.

Code Section

In [1]:
```
%%javascript
IPython.OutputArea.prototype._should_scroll = function(lines) {
    return false;
}
```

In [2]:
```
# import libraries

import pandas as pd
import numpy as np
pd.options.mode.chained_assignment = None  # default='warn'
import gzip
from collections import defaultdict
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, accuracy_score, confusion_matrix
from sklearn.ensemble import RandomForestRegressor
import string
import nltk
from nltk.corpus import stopwords
import os
from sklearn.model_selection import train_test_split
from itertools import compress
import matplotlib.pyplot as plt
import itertools
```

Define Functions

In [3]:

```
# parse raw data

"""
These functions parse data from compressed files.
"""

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')
```

In [4]:
```
def GetPosNegWords_count_Rate(text):

    """
    This function finds the positive and negative word rates for the text by:
        - using pre-defined lists of positive and negative words
        - making all words lower-case
        - removes blank spaces and punctuations
    """

    # import positive and negative word lists, define as set for higher efficiency
    posWords_list = set([i.strip() for i in open("positive-words.txt", "r").readlines()])
    negWords_list = set([i.strip() for i in open("negative-words.txt", "r", encoding="ISO-8859-1").readlines()])

    # count the number of positive and negative words present in each review text
    dict_ = defaultdict(list)
    for review_text in text:

        # remove punctuation symbols and spaces
        words = [n.lower().translate(str.maketrans('','',string.punctuation)) for n in review_text.split(' ')]
        words = [i for i in words if i != ''] # remove spaces

        # count number of positive and negative words in each review
        pos_count, neg_count = 0, 0
        for word in words:
            if word in posWords_list:
                pos_count+=1
            elif word in negWords_list:
                neg_count+=1
            else:
                continue

        # save count and rate
        if len(words) == 0: # prevent division by zero
            dict_['pos_count'].append(-1)
            dict_['neg_count'].append(-1)
            dict_['pos_rate'].append(-1)
            dict_['neg_rate'].append(-1)
        else:
            dict_['pos_count'].append(pos_count)
            dict_['neg_count'].append(neg_count)
            dict_['pos_rate'].append(pos_count/len(words))
            dict_['neg_rate'].append(neg_count/len(words))

    return dict_
```

In [5]:
```python
def Get_NumWords(text):

    """
    This function counts the number of words in the text by:
        - words are converted to lower-case
        - blank spaces and punctuations are removed
        - stopwords are removed using NLTK library
    """

    # get stopwords from nltk library
#     nltk.download('stopwords')
    stop_words = set(stopwords.words('english'))

    # loop over each review
    list_ = []
    for review_text in text:

        # remove punctuation symbols and spaces
        words = [n.lower().translate(str.maketrans('','',string.punctuation)) for n in review_text.split(' ')]
        words = [i for i in words if i != ''] # remove spaces

        # count number of words, excluding stopwords
        word_count = 0
        for word in words:
            if word in stop_words:
                continue
            else:
                 word_count+=1

        # save counts
        list_.append(word_count)

    return list_
```

In [6]:
```python
def Get_category_numtrans(category_lists):

    """
    This function assigns an ID number to each unique list of categories,
    then those IDs are used to create a sum for the collection of lists
    found in each product Category column.
    """

    # get unique category lists
    lists = []
    for i in category_lists:
        for j in i:
            lists.append(tuple(j))

    # make dictionary with IDs
    categorylists_dict = {k:v for v,k in enumerate(set(lists))}

    # transform list occurance to numbers using dictionary
    category_numtrans = []
    for i in category_lists:
        sum_ = 0
        for j in i:
            sum_ += categorylists_dict[tuple(j)]
        category_numtrans.append(sum_)

    # return transformation
    return category_numtrans
```

In [7]:
```python
def get_helpfulRate(col):

    """
    This function computes the ratio between number
    of helpfull votes and total votes.
    """

    allHelpful = []
    colHelpful = defaultdict(list)

    col_data = X_train_raw[col]
    allHelpful = y_train_raw

    for x,y in zip(col_data, allHelpful):
         colHelpful[x].append(y)

    averageRate = sum([x['nHelpful'] for x in allHelpful]) * 1.0 / sum([x['outOf'] for x in allHelpful])

    rate = {}
    for u in colHelpful:
        totalU = sum([x['outOf'] for x in colHelpful[u]])
        if totalU > 0:
            rate[u] = sum([x['nHelpful'] for x in colHelpful[u]]) * 1.0 / totalU
        else:
            rate[u] = averageRate

    return rate, averageRate
```

In [8]:
```python
def find_Helpful_rate(data, colname):

    """
    This function computes the helpful rate for a specified column
    For example: 'reviewerID' and 'itemID'.
    """

    rate_dict, avg_rate = get_helpfulRate(colname)

    ratehelpful = []
    for i in data:
        # use average for entries not present
        try:
            ratehelpful.append(rate_dict[i])
        except:
            ratehelpful.append(avg_rate)

    return ratehelpful
```

In [9]:
```python
def Get_punctuation_count(data):

    """
    This function counts the number of punctuation characters found in the text
    punctuation characters are taken from the 'string' library.
    """

    punct_set = set(string.punctuation)
    list_ = []
    for str_ in data:
        count_ = 0
        for c in str_:
            if c in punct_set:
                count_+=1
            else:
                continue
        list_.append(count_)

    return list_
```

In [10]:
```
def Get_numreviews_summarized(data):

    """
    This function finds the number of reviews encountered for the column specified:
    For example: 'itemID' and 'reviewerID'.
    """

    # initialize dict
    dict_ = {k:0 for k in set(data)}

    # count number
    for r in data:
        dict_[r]+=1

    # create list
    list_ = []
    for r in data:
        try:
            list_.append(dict_[r])
        except:
            list_.append(0)

    return list_
```

In [11]:
```
def Get_product_ratingDeviation(df):

    """
    This function finds the difference in rating for each review
    as compared to the mean of all ratings for the item.
    """

    ProdRating_mean_dict = df[['itemID', 'rating']].groupby(['itemID']).mean().rating.to_dict()

    rating_deviation = []
    for r, p in zip(df.rating, df.itemID):
        try:
            rating_deviation.append(r - ProdRating_mean_dict[p])
        except:
            rating_deviation.append(-444)

    return rating_deviation
```

In [12]:
```
def Get_count_firstLetterCapital(data):

    """
    This function counts the number of words in the text
    whose fist letter is capital.
    """

    list_ = []
    for review in data:
        count_ = 0
        for word in review.split():
            if word[0].isupper():
                count_+=1
        list_.append(count_)

    return list_
```

In [13]:
```
def Get_avg_word_length(data):

    """
    This function computes the average word length for the text,
    stop words are removed using the list from NLTK library.
    """

    # get stopwords from nltk library
#     nltk.download('stopwords')
    stop_words = set(stopwords.words('english'))

    # loop over each review
    list_ = []
    for review_text in data:

        # remove punctuation symbols and spaces
        words = [n.lower().translate(str.maketrans('','',string.punctuation)) for n in review_text.split(' ')]
        words = [i for i in words if i != ''] # remove spaces

        # remove stopwords
        words = set(words) - stop_words

        # save average length
        try:
            list_.append(sum([len(w) for w in words])/len(words))
        except:
            list_.append(-1)

    return list_
```

In [14]:
```
def Get_delta_sinceFirstReview(df):

    """
    This function finds the Unix time difference
    since the first review for each item.
    """

    # find time for first review in each product
    first_product_reviewtime = df[['itemID','unixReviewTime']].groupby(['itemID']).min().unixReviewTime.to_dict()

    # find delta for each review
    delta = []
    for i,t in zip(df.itemID, df.unixReviewTime):
        delta.append(t-first_product_reviewtime[i])

    return delta
```

In [15]:
```python
def Get_delta_sinceLastReview(df):

    """
    This function finds the Unix time difference
    since the last review for each item.
    """

    # create reduced df
    new_df = df[['itemID','unixReviewTime']].groupby(['itemID'])['unixReviewTime'].apply(list)

    # create dictionary with deltas
    delta_dict = defaultdict(dict)

    for idx, times_list in zip(new_df.index, new_df):

        times_list = sorted(times_list)
        times_list_deltas = np.append(np.array([0]) , np.diff(times_list))

        for t, d in zip(times_list, times_list_deltas):
            delta_dict[idx][t] = d

    # generate list with deltas matching input dataframe
    list_ = []
    for u,t in zip(df.itemID, df.unixReviewTime):
        list_.append(delta_dict[u][t])

    return list_
```

In [16]:
```python
def Get_colsRatio(col1, col2):

    """
    This function finds the ratio element-wise for two columns.
    """

    list_ = []
    for i,j in zip(col1, col2):

        try:
            list_.append(i/j)
        except:
            list_.append(-1)

    return list_
```

In [17]:
```python
def Get_numCapitalwords(data):

    """
    This function finds the number of words
    in the text where all letters are capital.
    """

    # get stopwords from nltk library
#     nltk.download('stopwords')
    stop_words = set(stopwords.words('english'))

    # loop over each review
    list_ = []
    for review_text in data:

        # remove punctuation symbols and spaces
        words = [n.translate(str.maketrans('','',string.punctuation)) for n in review_text.split(' ')]
        words = [i for i in words if i != ''] # remove spaces

        # remove stopwords
        words = set(words) - stop_words

        # append count of upper case words
        list_.append(len([i for i in words if i.isupper()]))

    return list_
```

In [18]:
```python
def Get_ExclQues_charCount(data):

    """
    This function counts the number of exclamation
    and question characters in the text.
    """

    # loop over each review
    list_ = []
    for review_text in data:
        # append count of question and exclamation characters
        list_.append(review_text.count('?') + review_text.count('!'))

    return list_
```

In [19]:
```
def Get_features(df):

    """
    This function learns all the features from the raw data,
    each feature is added as a new column to the input dataframe.
    """

    # Modify ----------> "categories"

    # get number of characters
    df.loc[:,'categories_count'] = [len(i) for i in df['categories']]

    # generate numerical category by transforming combination of lists to numbers
    df.loc[:,'category_numtrans'] = Get_category_numtrans(df.categories)

    # Modify ----------> "itemID" and "reviewerID"

    # create dictionaries for itemID and reviewerID, convert from categorical to numeric
    items_dict = {k:v for v,k in enumerate(set(df.itemID))}
    reviewer_dict = {k:v for v,k in enumerate(set(df.reviewerID))}

    # change item and reviewer IDs to numeric
    df.loc[:,'itemID'] = [items_dict[i] for i in df['itemID']]
    df.loc[:,'reviewerID'] = [reviewer_dict[i] for i in df['reviewerID']]

    # add helpful rate for itemID and reviewerID
    df.loc[:,'itemID_helpfulRate'] = find_Helpful_rate(df['itemID'], 'itemID')
    df.loc[:,'reviewerID_helpfulRate'] = find_Helpful_rate(df['reviewerID'], 'reviewerID')

    # get number of reviews for each user
    df.loc[:,'reviewerID_numReviews'] = Get_numreviews_summarized(df['reviewerID'])

    # get number of reviews for each product
    df.loc[:,'itemID_numReviews'] = Get_numreviews_summarized(df['itemID'])

    # Modify ----------> "reviewText"

    # get number of words, remove stopwords
    df.loc[:,'reviewText_count_words'] = Get_NumWords(df['reviewText'])

    # get character count
    df.loc[:,'reviewText_count_char'] = [len(i) for i in df['reviewText']]

    # get punctuation count
```

```python
df.loc[:,'reviewText_count_punctu'] = Get_punctuation_count(df['reviewText'])

# get number of words that start with a capital leter
df.loc[:,'reviewText_count_firstCapital'] = Get_count_firstLetterCapital(df['reviewText'])

# get average word length
df.loc[:,'reviewText_avgWordLength'] = Get_avg_word_length(df['reviewText'])

# get number of capital words
df.loc[:,'reviewText_capitalwords'] = Get_numCapitalwords(df['reviewText'])

# get number of question and exclamation characters
df.loc[:,'reviewText_ExclQue_countchar'] = Get_ExclQues_charCount(df['reviewText'])

# get ratio between puctuations with character numbers
df.loc[:,'reviewText_PunctChar_ratio'] = Get_colsRatio(df['reviewText_count_punctu'], df['reviewText_count_char'])

# get positive and negative word rate
reviewText_PosNeg = GetPosNegWords_count_Rate(df.reviewText)
df.loc[:,'reviewText_posWordCount'] = reviewText_PosNeg['pos_count']
df.loc[:,'reviewText_negWordCount'] = reviewText_PosNeg['neg_count']
df.loc[:,'reviewText_posWordRate'] = reviewText_PosNeg['pos_rate']
df.loc[:,'reviewText_negWordRate'] = reviewText_PosNeg['neg_rate']

# Modify ----------> "summary"

# get number of words, remove stopwords
df.loc[:,'summary_count_words'] = Get_NumWords(df['summary'])

# get character count
df.loc[:,'summary_count_char'] = [len(i) for i in df['summary']]

# get punctuation count
df.loc[:,'summary_count_punctu'] = Get_punctuation_count(df['summary'])

# get number of words that start with a capital leter
df.loc[:,'summary_count_firstCapital'] = Get_count_firstLetterCapital(df['summary'])

# get average word length
df.loc[:,'summary_avgWordLength'] = Get_avg_word_length(df['summary'])

# get number of capital words
df.loc[:,'summary_capitalwords'] = Get_numCapitalwords(df['summary'])

# get number of question and exclamation characters
```

```python
df.loc[:,'summary_ExclQue_countchar'] = Get_ExclQues_charCount(df['summary'])

# get ratio between puctuations with character numbers
df.loc[:,'summary_PunctChar_ratio'] = Get_colsRatio(df['summary_count_punctu'], df['summary_count_char'])

# get positive and negative word rate
summary_PosNeg = GetPosNegWords_count_Rate(df.summary)
df.loc[:,'summary_posWordCount'] = summary_PosNeg['pos_count']
df.loc[:,'summary_negWordCount'] = summary_PosNeg['neg_count']
df.loc[:,'summary_posWordRate'] = summary_PosNeg['pos_rate']
df.loc[:,'summary_negWordRate'] = summary_PosNeg['neg_rate']

# Modify ----------> "helpful"

# parse helpful votes
df.loc[:,'outOf_feature'] = [i['outOf'] for i in df['helpful']]

# Modify ----------> "price"

# change NA values to -999
df.loc[df.price.isna(),'price'] = -999

# Modify ----------> "unixReviewTime"

# find time since first review
df.loc[:,'unixReviewTime_delta_firstreview'] = Get_delta_sinceFirstReview(df)

# find time since last review for same product
df.loc[:,'unixReviewTime_delta_lastreview'] = Get_delta_sinceLastReview(df)

# Modify ----------> "reviewText"

# get rating deviation from the mean
df.loc[:,'rating_deviation'] = Get_product_ratingDeviation(df)

# Add -------------> New Columns

# votes over time
df.loc[:,'votes_time'] = df.outOf_feature/df.unixReviewTime

# ratio of summary to reviewText for characters and words
df.loc[:,'summary_reviewText_charRatio'] = Get_colsRatio(df.summary_count_char, df.reviewText_count_char)
df.loc[:,'summary_reviewText_wordsRatio'] = Get_colsRatio(df.summary_count_words, df.reviewText_count_words)

# define columns to keep
```

```python
    cols = ['categoryID','categories_count', 'category_numtrans','summary_count_words', 'itemID', 'reviewerID', 'rating',
            'reviewText_count_words', 'itemID_helpfulRate', 'reviewerID_helpfulRate',
            'reviewerID_numReviews', 'rating_deviation',
            'outOf_feature', 'unixReviewTime', 'price', 'reviewText_posWordCount', 'reviewText_negWordCount',
            'reviewText_posWordRate', 'reviewText_negWordRate', 'summary_count_char', 'reviewText_count_char',
            'reviewText_count_punctu','summary_count_punctu','summary_posWordCount',
            'summary_negWordCount', 'summary_posWordRate','summary_negWordRate',
           'reviewText_count_firstCapital', 'summary_count_firstCapital',
           'reviewText_avgWordLength','summary_avgWordLength', 'unixReviewTime_delta_firstreview',
           'votes_time', 'summary_reviewText_charRatio', 'summary_reviewText_wordsRatio', 'itemID_numReviews',
           'reviewText_capitalwords', 'summary_capitalwords',
           'reviewText_ExclQue_countchar', 'summary_ExclQue_countchar',
           'reviewText_PunctChar_ratio', 'summary_PunctChar_ratio', 'unixReviewTime_delta_lastreview']

    # return features
    return df[cols]
```

In [20]:

```python
def Get_labels_ratio(df):

    """
    This function finds the ratio between helpful votes
    with total votes from raw data. This ratio is the
    regression label used to train the model.
    """

    return [i['nHelpful']/i['outOf'] if i['outOf']!=0 else 0 for i in df]
```

In [21]:
```python
def save_predictions(pred):

    """
    This function reads a formatted file and writes
    the predictions found for the test data. The file
    is used as input to the DSE 220 Kaggle competition.
    """

    predictions = open("predictions_Helpful.txt", 'w')

    count = 0
    for l in open("pairs_Helpful.txt"):

        if l.startswith("userID"):
            #header
            predictions.write(l)
            continue

        u,i,outOf = l.strip().split('-')

        predictions.write(u + '-' + i + '-' + str(outOf) + ',' + str(pred[count]) + '\n')

        count+=1

    predictions.close()
```

In [22]:
```python
def perfmetrics_RFmodel(max_depth, n_estimators, X_train_filt, y_train_filt, X_train, y_train, X_test, y_test):

    """
    This function applies the random forest model,
    the inputs and outputs are listed below:

    Inputs:
        Random Forest Hyperparameters
            - max_depth: max depth of the tree
            - n_estimators: number of trees in the forest
        Trainning Data
            - X_train_filt: train the model with train data where number of votes (labels) are > 0
            - y_train_filt train the model with train data where number of votes (labels) are > 0
            - X_train: data includes all labels
            - y_train: data includes all labels
        Testing Data (or Validation)
            - X_test: data includes all labels
            - y_test: data includes all labels

    Outputs:
        - train_accuracy
        - test_accuracy
        - train_mae (Mean Absolute Error)
        - test_mae
        - train_mse (Mean Squared Error)
        - test_mse
        - y_pred_test (or validation predictions)
    """

    # define model
    regr = RandomForestRegressor(max_depth=max_depth, n_estimators=n_estimators, n_jobs=-1)

    # evaluate, with filtered data
    regr.fit(X_train_filt, y_train_filt)

    # predict test and train, multiply prediction by the number of votes
    y_pred_train = regr.predict(X_train)*np.array(X_train.outOf_feature)
    y_pred_test = regr.predict(X_test)*np.array(X_test.outOf_feature)

    # round to the nearest integer, labels are integers
    y_pred_train = [int(round(i)) for i in y_pred_train]
    y_pred_test = [int(round(i)) for i in y_pred_test]

    # calculate accuracy, mean absolute error, and mean squared error
```

```
    train_accuracy = accuracy_score(y_train, y_pred_train)
    train_mae = mean_absolute_error(y_train, y_pred_train)
    train_mse = mean_squared_error(y_train, y_pred_train)

    if y_test != None: # case when test labels are known
        test_accuracy = accuracy_score(y_test, y_pred_test)
        test_mae = mean_absolute_error(y_test, y_pred_test)
        test_mse = mean_squared_error(y_test, y_pred_test)
    else: # case when test labels are predicted
        return (train_accuracy, train_mae, train_mse, y_pred_test)

    # return all performance metrics
    return (train_accuracy, test_accuracy, train_mae, test_mae, train_mse, test_mse, y_pred_test)
```

Main Section

Read Data

In [23]:
```
%%time

# read train
train = getDF('train.json.gz')

# define features and labels for train data
train_features = train
train_labels = train['helpful']

# read test
test = getDF('test_Helpful.json.gz')
```

```
CPU times: user 27.3 s, sys: 1.47 s, total: 28.8 s
Wall time: 31.1 s
```

In [24]:
```
# split train into train and validation
X_train_raw, X_val_raw, y_train_raw, y_val_raw = train_test_split(train_features, train_labels, test_size=0.20)
```

Table 1. Trainning data example (raw).

```
In [25]: X_train_raw.head()
```

Out[25]:

	categoryID	categories	itemID	reviewerID	rating	reviewText	reviewHash	reviewTime	summary	unixReviewTime	helpful	price
44011	0	[[Clothing, Shoes & Jewelry, D, Dreams], [Clot...	I322658212	U656051947	5.0	These pajamas are a perfect weight for me. I'm...	R196940592	03 25, 2014	Nice weight & comfy.	1395705600	{'outOf': 0, 'nHelpful': 0}	NaN
53593	0	[[Clothing, Shoes & Jewelry, Shoes & Accessori...	I913405870	U604395367	3.0	I returned these because I thought they looked...	R612741609	09 16, 2013	Nice quality	1379289600	{'outOf': 0, 'nHelpful': 0}	NaN
141214	0	[[Clothing, Shoes & Jewelry, Women, Shoes, San...	I036574902	U886882212	3.0	Saddle is very pretty, and I think it will be ...	R077184873	05 5, 2014	Very pretty	1399248000	{'outOf': 0, 'nHelpful': 0}	NaN
179392	0	[[Clothing, Shoes & Jewelry, Women], [Clothing...	I134433773	U154865904	5.0	This ring is truly stunning and dramatic while...	R611116635	03 16, 2014	Stainless Steel Band With CZ	1394928000	{'outOf': 0, 'nHelpful': 0}	5.94
32982	1	[[Clothing, Shoes & Jewelry, Men, Accessories,...	I532478566	U578222660	4.0	At first the tension on this was too tight, bu...	R286930585	05 16, 2014	No nonsense, just how I like it.	1400198400	{'outOf': 0, 'nHelpful': 0}	7.99

Data Preparation - Train and Validation

```
In [26]: %%time

# learn features
X_train = Get_features(X_train_raw)
X_val = Get_features(X_val_raw)

# create labels
y_train = [i['nHelpful'] for i in y_train_raw]
y_val = [i['nHelpful'] for i in y_val_raw]
```

```
CPU times: user 2min 39s, sys: 2.78 s, total: 2min 42s
Wall time: 2min 43s
```

In [27]:
```
%%time

# model is only trained for reviews where number of votes is > 0

# find indices
idx = np.array([i['outOf'] for i in X_train_raw.helpful]) > 0

# filter new train data and learn features on filtered data
X_train_filt = Get_features(X_train_raw.loc[idx,:])
y_train_ratio_filt = Get_labels_ratio(list(compress(y_train_raw, idx)))
```

```
CPU times: user 52 s, sys: 302 ms, total: 52.3 s
Wall time: 52.6 s
```

Feature Selection - Keep Features with Better Performance than Baseline

In [28]:

```
%%time

# define hyperparameters to test
max_depth = 10
n_estimators = 100

# define baseline and other features
baseline_features = ['itemID', 'reviewerID', 'outOf_feature']
features = [i for i in X_train.columns if i not in baseline_features]

# run model with baseline features + one feature, see MAE if improves
metrics = []
for i in range(len(features)+1):

    # define features to test
    if i == 0: # baseline
        curr_features = baseline_features
        feature_name = 'baseline'
    else: # other features
        curr_features = baseline_features + [features[i-1]]
        feature_name = features[i-1]

    # get metrics (train_accuracy, test_accuracy, train_mae, test_mae, train_mse, test_mse, y_pred_test)
    metrics_tuple = perfmetrics_RFmodel(max_depth, n_estimators, X_train_filt[curr_features],
                                        y_train_ratio_filt, X_train[curr_features], y_train,
                                        X_val[curr_features], y_val)

    # save metrics
    metrics.append((feature_name,) + metrics_tuple[:-1])
```

```
CPU times: user 12min 7s, sys: 7.57 s, total: 12min 15s
Wall time: 3min 43s
```

Table 2. Features tested one-by-one along with baseline, those with higher than baseline performance (lower MAE) are selected.

In [29]:
```python
# create dataframe
df_metrics = pd.DataFrame(metrics, columns=['features', 'train_accur', 'val_accur', 'train_mae', 'val_mae',
                                            'train_mse', 'val_mse'])

# sort by MAE in validation, descending
df_metrics = df_metrics.sort_values(by='val_mae')

df_metrics
```

Out[29]:

	features	train_accur	val_accur	train_mae	val_mae	train_mse	val_mse
10	rating_deviation	0.857513	0.863350	0.183419	0.179500	0.753856	0.742850
21	summary_posWordCount	0.858394	0.859800	0.182506	0.183150	0.615256	0.812350
23	summary_posWordRate	0.856988	0.858400	0.183706	0.183825	0.643056	0.802025
11	unixReviewTime	0.855681	0.857625	0.183688	0.184100	0.582037	0.828750
15	reviewText_posWordRate	0.856012	0.858625	0.181719	0.184650	0.514431	0.914550
5	rating	0.859163	0.864450	0.178381	0.184875	0.482044	1.891625
24	summary_negWordRate	0.855675	0.857600	0.184012	0.184975	0.616337	0.858525
13	reviewText_posWordCount	0.855444	0.856925	0.184200	0.185400	0.524800	0.831150
22	summary_negWordCount	0.853819	0.856400	0.186306	0.185575	0.627181	0.846875
38	reviewText_PunctChar_ratio	0.854006	0.855700	0.186356	0.185800	0.554556	0.836600
18	reviewText_count_char	0.857587	0.857500	0.180156	0.185800	0.554769	0.914700
12	price	0.854738	0.856050	0.184037	0.185850	0.553400	0.857850
36	reviewText_ExclQue_countchar	0.853450	0.854650	0.187562	0.186000	0.685662	0.841750
31	summary_reviewText_charRatio	0.854962	0.855250	0.184113	0.186175	0.617812	0.879275
6	reviewText_count_words	0.856363	0.856500	0.182669	0.186300	0.607556	0.888850
37	summary_ExclQue_countchar	0.853762	0.854300	0.185938	0.186375	0.623075	0.849875
29	unixReviewTime_delta_firstreview	0.854838	0.854300	0.184275	0.186425	0.657975	0.854025
1	categoryID	0.847069	0.856250	0.193944	0.186500	0.687056	0.848750
25	reviewText_count_firstCapital	0.855750	0.854925	0.184306	0.186550	0.617419	0.846350
26	summary_count_firstCapital	0.855481	0.855225	0.187044	0.186625	0.674181	0.845625

	features	train_accur	val_accur	train_mae	val_mae	train_mse	val_mse
35	summary_capitalwords	0.854475	0.853825	0.186550	0.186725	0.665575	0.836625
39	summary_PunctChar_ratio	0.854675	0.854650	0.184806	0.186825	0.661294	0.849475
20	summary_count_punctu	0.853744	0.854475	0.187250	0.186900	0.677963	0.847800
40	unixReviewTime_delta_lastreview	0.854044	0.854100	0.186406	0.186900	0.647744	0.851700
28	summary_avgWordLength	0.855537	0.854900	0.183356	0.186925	0.608669	0.877225
19	reviewText_count_punctu	0.855206	0.853725	0.184331	0.187200	0.549556	0.739850
32	summary_reviewText_wordsRatio	0.854050	0.854550	0.185087	0.187500	0.623687	0.881500
14	reviewText_negWordCount	0.854550	0.853525	0.186031	0.187500	0.654481	0.846400
33	itemID_numReviews	0.851650	0.852875	0.201437	0.187550	0.829775	0.857650
3	category_numtrans	0.853400	0.853425	0.187638	0.187575	0.558163	0.858025
30	votes_time	0.853525	0.852875	0.187888	0.187750	0.688375	0.866300
34	reviewText_capitalwords	0.853900	0.852875	0.185938	0.187875	0.655900	0.853925
8	reviewerID_helpfulRate	0.857487	0.853150	0.182163	0.187925	0.671488	0.857475
0	baseline	0.853556	0.852500	0.186312	0.188325	0.677188	0.865975
4	summary_count_words	0.854725	0.853525	0.186587	0.188350	0.650500	0.866250
7	itemID_helpfulRate	0.856669	0.852975	0.184888	0.188400	0.697700	0.860200
17	summary_count_char	0.854369	0.853175	0.186069	0.188425	0.657131	0.869125
2	categories_count	0.853712	0.852300	0.188206	0.188775	0.575481	0.860575
16	reviewText_negWordRate	0.855213	0.856225	0.183631	0.189450	0.501481	1.175300
9	reviewerID_numReviews	0.853731	0.851750	0.187300	0.189550	0.643663	0.859650
27	reviewText_avgWordLength	0.854031	0.853475	0.185463	0.190525	0.592287	1.120425

In [30]:
```
# keep features with better performance than baseline
baseline_valMAE = df_metrics.val_mae[df_metrics.features == 'baseline'].values[0]

features_aboveBaseline = list(df_metrics.features[df_metrics.val_mae < baseline_valMAE])
all_features = baseline_features + features_aboveBaseline

print(all_features)
```

```
['itemID', 'reviewerID', 'outOf_feature', 'rating_deviation', 'summary_posWordCount', 'summary_posWordRate', 'unixReviewTime', 'reviewText_p
osWordRate', 'rating', 'summary_negWordRate', 'reviewText_posWordCount', 'summary_negWordCount', 'reviewText_PunctChar_ratio', 'reviewText_c
ount_char', 'price', 'reviewText_ExclQue_countchar', 'summary_reviewText_charRatio', 'reviewText_count_words', 'summary_ExclQue_countchar',
'unixReviewTime_delta_firstreview', 'categoryID', 'reviewText_count_firstCapital', 'summary_count_firstCapital', 'summary_capitalwords', 'su
mmary_PunctChar_ratio', 'summary_count_punctu', 'unixReviewTime_delta_lastreview', 'summary_avgWordLength', 'reviewText_count_punctu', 'summ
ary_reviewText_wordsRatio', 'reviewText_negWordCount', 'itemID_numReviews', 'category_numtrans', 'votes_time', 'reviewText_capitalwords', 'r
eviewerID_helpfulRate']
```

Feature Selection - Ablation Study with Features Elimination (Ordered by MAE descending)

In [31]:
```
%%time

# define hyperparameters to test
max_depth = 10
n_estimators = 100

# save list of features eliminated and the calulcated MAE
features_list_mae_mse = []

# eliminate each feature, one-by-one, starting with the highest MAE feature to the lowest on the validation dataset
for f in range(len(all_features)-2): # features list is sorted with MAE ascending

    # eliminate starting from the end
    temp_featurelist = all_features[:len(all_features)-f]

    # get feature name
    if f == 0:
        f_eliminated = 'None'
    else:
        f_eliminated = all_features[-f:]

    # get metrics (train_accuracy, test_accuracy, train_mae, test_mae, train_mse, test_mse, y_pred_test)
    metrics_tuple = perfmetrics_RFmodel(max_depth, n_estimators, X_train_filt[temp_featurelist],
                                        y_train_ratio_filt, X_train[temp_featurelist],
                                        y_train, X_val[temp_featurelist], y_val)

    # print metrics
    print('Features Eliminated: {}'.format(f_eliminated))
    print('\tTrain Accuracy: {}'.format(metrics_tuple[0]))
    print('\tValidation Accuracy: {}'.format(metrics_tuple[1]))
    print('\tTrain MAE: {}'.format(metrics_tuple[2]))
    print('\tValidation MAE: {}'.format(metrics_tuple[3]))
    print('\tTrain MSE: {}'.format(metrics_tuple[4]))
    print('\tValidation MSE: {}'.format(metrics_tuple[5]))
    print('\n')

    # save mae and eliminated features lists
    features_list_mae_mse.append((metrics_tuple[3], metrics_tuple[5], f_eliminated))
```

```
Features Eliminated: None
        Train Accuracy: 0.8679625
        Validation Accuracy: 0.86735
        Train MAE: 0.1681
        Validation MAE: 0.1741
```

```
        Train MSE: 0.5105125
        Validation MSE: 0.769


Features Eliminated: ['reviewerID_helpfulRate']
        Train Accuracy: 0.86815
        Validation Accuracy: 0.867725
        Train MAE: 0.1678125
        Validation MAE: 0.173675
        Train MSE: 0.5429125
        Validation MSE: 0.772075


Features Eliminated: ['reviewText_capitalwords', 'reviewerID_helpfulRate']
        Train Accuracy: 0.8676375
        Validation Accuracy: 0.8675
        Train MAE: 0.1694
        Validation MAE: 0.17405
        Train MSE: 0.5346125
        Validation MSE: 0.71645


Features Eliminated: ['votes_time', 'reviewText_capitalwords', 'reviewerID_helpfulRate']
        Train Accuracy: 0.86820625
        Validation Accuracy: 0.86715
        Train MAE: 0.1682625
        Validation MAE: 0.17425
        Train MSE: 0.54125
        Validation MSE: 0.72675


Features Eliminated: ['category_numtrans', 'votes_time', 'reviewText_capitalwords', 'reviewerID_helpfulRate']
        Train Accuracy: 0.8680375
        Validation Accuracy: 0.86735
        Train MAE: 0.168725
        Validation MAE: 0.1749
        Train MSE: 0.5389875
        Validation MSE: 0.80905


Features Eliminated: ['itemID_numReviews', 'category_numtrans', 'votes_time', 'reviewText_capitalwords', 'reviewerID_helpfulRate']
        Train Accuracy: 0.8701875
        Validation Accuracy: 0.8668
        Train MAE: 0.162575
        Validation MAE: 0.17375
```

```
        Train MSE: 0.48335
        Validation MSE: 0.7025


Features Eliminated: ['reviewText_negWordCount', 'itemID_numReviews', 'category_numtrans', 'votes_time', 'reviewText_capitalwords', 'revie
werID_helpfulRate']
        Train Accuracy: 0.87045
        Validation Accuracy: 0.86705
        Train MAE: 0.16189375
        Validation MAE: 0.173525
        Train MSE: 0.51855625
        Validation MSE: 0.708675


Features Eliminated: ['summary_reviewText_wordsRatio', 'reviewText_negWordCount', 'itemID_numReviews', 'category_numtrans', 'votes_time',
'reviewText_capitalwords', 'reviewerID_helpfulRate']
        Train Accuracy: 0.87003125
        Validation Accuracy: 0.866925
        Train MAE: 0.16253125
        Validation MAE: 0.173325
        Train MSE: 0.50564375
        Validation MSE: 0.708675


Features Eliminated: ['reviewText_count_punctu', 'summary_reviewText_wordsRatio', 'reviewText_negWordCount', 'itemID_numReviews', 'categor
y_numtrans', 'votes_time', 'reviewText_capitalwords', 'reviewerID_helpfulRate']
        Train Accuracy: 0.870525
        Validation Accuracy: 0.8671
        Train MAE: 0.1616125
        Validation MAE: 0.173825
        Train MSE: 0.479925
        Validation MSE: 0.715575


Features Eliminated: ['summary_avgWordLength', 'reviewText_count_punctu', 'summary_reviewText_wordsRatio', 'reviewText_negWordCount', 'ite
mID_numReviews', 'category_numtrans', 'votes_time', 'reviewText_capitalwords', 'reviewerID_helpfulRate']
        Train Accuracy: 0.869575
        Validation Accuracy: 0.866725
        Train MAE: 0.16244375
        Validation MAE: 0.174
        Train MSE: 0.50869375
        Validation MSE: 0.73725


Features Eliminated: ['unixReviewTime_delta_lastreview', 'summary_avgWordLength', 'reviewText_count_punctu', 'summary_reviewText_wordsRati
```

```
o', 'reviewText_negWordCount', 'itemID_numReviews', 'category_numtrans', 'votes_time', 'reviewText_capitalwords', 'reviewerID_helpfulRat
e']
        Train Accuracy: 0.87056875
        Validation Accuracy: 0.867375
        Train MAE: 0.16161875
        Validation MAE: 0.17375
        Train MSE: 0.48356875
        Validation MSE: 0.7448


Features Eliminated: ['summary_count_punctu', 'unixReviewTime_delta_lastreview', 'summary_avgWordLength', 'reviewText_count_punctu', 'summ
ary_reviewText_wordsRatio', 'reviewText_negWordCount', 'itemID_numReviews', 'category_numtrans', 'votes_time', 'reviewText_capitalwords',
'reviewerID_helpfulRate']
        Train Accuracy: 0.870425
        Validation Accuracy: 0.867075
        Train MAE: 0.16193125
        Validation MAE: 0.1733
        Train MSE: 0.48360625
        Validation MSE: 0.7058


Features Eliminated: ['summary_PunctChar_ratio', 'summary_count_punctu', 'unixReviewTime_delta_lastreview', 'summary_avgWordLength', 'revi
ewText_count_punctu', 'summary_reviewText_wordsRatio', 'reviewText_negWordCount', 'itemID_numReviews', 'category_numtrans', 'votes_time',
'reviewText_capitalwords', 'reviewerID_helpfulRate']
        Train Accuracy: 0.8704125
        Validation Accuracy: 0.867275
        Train MAE: 0.1619375
        Validation MAE: 0.1736
        Train MSE: 0.5141
        Validation MSE: 0.735


Features Eliminated: ['summary_capitalwords', 'summary_PunctChar_ratio', 'summary_count_punctu', 'unixReviewTime_delta_lastreview', 'summa
ry_avgWordLength', 'reviewText_count_punctu', 'summary_reviewText_wordsRatio', 'reviewText_negWordCount', 'itemID_numReviews', 'category_n
umtrans', 'votes_time', 'reviewText_capitalwords', 'reviewerID_helpfulRate']
        Train Accuracy: 0.869875
        Validation Accuracy: 0.867325
        Train MAE: 0.16215
        Validation MAE: 0.17355
        Train MSE: 0.4518375
        Validation MSE: 0.7451


Features Eliminated: ['summary_count_firstCapital', 'summary_capitalwords', 'summary_PunctChar_ratio', 'summary_count_punctu', 'unixReview
Time_delta_lastreview', 'summary_avgWordLength', 'reviewText_count_punctu', 'summary_reviewText_wordsRatio', 'reviewText_negWordCount', 'i
```

```
temID_numReviews', 'category_numtrans', 'votes_time', 'reviewText_capitalwords', 'reviewerID_helpfulRate']
        Train Accuracy: 0.8698
        Validation Accuracy: 0.867225
        Train MAE: 0.1622875
        Validation MAE: 0.173975
        Train MSE: 0.4795625
        Validation MSE: 0.744025


Features Eliminated: ['reviewText_count_firstCapital', 'summary_count_firstCapital', 'summary_capitalwords', 'summary_PunctChar_ratio', 's
ummary_count_punctu', 'unixReviewTime_delta_lastreview', 'summary_avgWordLength', 'reviewText_count_punctu', 'summary_reviewText_wordsRati
o', 'reviewText_negWordCount', 'itemID_numReviews', 'category_numtrans', 'votes_time', 'reviewText_capitalwords', 'reviewerID_helpfulRat
e']
        Train Accuracy: 0.86995
        Validation Accuracy: 0.8667
        Train MAE: 0.1619125
        Validation MAE: 0.174425
        Train MSE: 0.47415
        Validation MSE: 0.744225


Features Eliminated: ['categoryID', 'reviewText_count_firstCapital', 'summary_count_firstCapital', 'summary_capitalwords', 'summary_PunctC
har_ratio', 'summary_count_punctu', 'unixReviewTime_delta_lastreview', 'summary_avgWordLength', 'reviewText_count_punctu', 'summary_review
Text_wordsRatio', 'reviewText_negWordCount', 'itemID_numReviews', 'category_numtrans', 'votes_time', 'reviewText_capitalwords', 'reviewerI
D_helpfulRate']
        Train Accuracy: 0.86934375
        Validation Accuracy: 0.8664
        Train MAE: 0.1634125
        Validation MAE: 0.176625
        Train MSE: 0.554575
        Validation MSE: 0.921025


Features Eliminated: ['unixReviewTime_delta_firstreview', 'categoryID', 'reviewText_count_firstCapital', 'summary_count_firstCapital', 'su
mmary_capitalwords', 'summary_PunctChar_ratio', 'summary_count_punctu', 'unixReviewTime_delta_lastreview', 'summary_avgWordLength', 'revie
wText_count_punctu', 'summary_reviewText_wordsRatio', 'reviewText_negWordCount', 'itemID_numReviews', 'category_numtrans', 'votes_time',
'reviewText_capitalwords', 'reviewerID_helpfulRate']
        Train Accuracy: 0.869575
        Validation Accuracy: 0.867475
        Train MAE: 0.16406875
        Validation MAE: 0.175425
        Train MSE: 0.57839375
        Validation MSE: 0.907625
```

```
Features Eliminated: ['summary_ExclQue_countchar', 'unixReviewTime_delta_firstreview', 'categoryID', 'reviewText_count_firstCapital', 'sum
mary_count_firstCapital', 'summary_capitalwords', 'summary_PunctChar_ratio', 'summary_count_punctu', 'unixReviewTime_delta_lastreview', 's
ummary_avgWordLength', 'reviewText_count_punctu', 'summary_reviewText_wordsRatio', 'reviewText_negWordCount', 'itemID_numReviews', 'catego
ry_numtrans', 'votes_time', 'reviewText_capitalwords', 'reviewerID_helpfulRate']
        Train Accuracy: 0.86935625
        Validation Accuracy: 0.8679
        Train MAE: 0.16398125
        Validation MAE: 0.174725
        Train MSE: 0.54790625
        Validation MSE: 0.894625


Features Eliminated: ['reviewText_count_words', 'summary_ExclQue_countchar', 'unixReviewTime_delta_firstreview', 'categoryID', 'reviewText
_count_firstCapital', 'summary_count_firstCapital', 'summary_capitalwords', 'summary_PunctChar_ratio', 'summary_count_punctu', 'unixReview
Time_delta_lastreview', 'summary_avgWordLength', 'reviewText_count_punctu', 'summary_reviewText_wordsRatio', 'reviewText_negWordCount', 'i
temID_numReviews', 'category_numtrans', 'votes_time', 'reviewText_capitalwords', 'reviewerID_helpfulRate']
        Train Accuracy: 0.86899375
        Validation Accuracy: 0.867075
        Train MAE: 0.16439375
        Validation MAE: 0.176375
        Train MSE: 0.52345625
        Validation MSE: 0.984725


Features Eliminated: ['summary_reviewText_charRatio', 'reviewText_count_words', 'summary_ExclQue_countchar', 'unixReviewTime_delta_firstrevi
ew', 'categoryID', 'reviewText_count_firstCapital', 'summary_count_firstCapital', 'summary_capitalwords', 'summary_PunctChar_ratio', 'summar
y_count_punctu', 'unixReviewTime_delta_lastreview', 'summary_avgWordLength', 'reviewText_count_punctu', 'summary_reviewText_wordsRatio', 're
viewText_negWordCount', 'itemID_numReviews', 'category_numtrans', 'votes_time', 'reviewText_capitalwords', 'reviewerID_helpfulRate']
        Train Accuracy: 0.868775
        Validation Accuracy: 0.867775
        Train MAE: 0.16463125
        Validation MAE: 0.17545
        Train MSE: 0.53184375
        Validation MSE: 0.96365


Features Eliminated: ['reviewText_ExclQue_countchar', 'summary_reviewText_charRatio', 'reviewText_count_words', 'summary_ExclQue_countchar',
'unixReviewTime_delta_firstreview', 'categoryID', 'reviewText_count_firstCapital', 'summary_count_firstCapital', 'summary_capitalwords', 'su
mmary_PunctChar_ratio', 'summary_count_punctu', 'unixReviewTime_delta_lastreview', 'summary_avgWordLength', 'reviewText_count_punctu', 'summ
ary_reviewText_wordsRatio', 'reviewText_negWordCount', 'itemID_numReviews', 'category_numtrans', 'votes_time', 'reviewText_capitalwords', 'r
eviewerID_helpfulRate']
        Train Accuracy: 0.86899375
        Validation Accuracy: 0.8674
        Train MAE: 0.1639875
```

```
        Validation MAE: 0.17685
        Train MSE: 0.512375
        Validation MSE: 1.0431


Features Eliminated: ['price', 'reviewText_ExclQue_countchar', 'summary_reviewText_charRatio', 'reviewText_count_words', 'summary_ExclQue_co
untchar', 'unixReviewTime_delta_firstreview', 'categoryID', 'reviewText_count_firstCapital', 'summary_count_firstCapital', 'summary_capitalw
ords', 'summary_PunctChar_ratio', 'summary_count_punctu', 'unixReviewTime_delta_lastreview', 'summary_avgWordLength', 'reviewText_count_punc
tu', 'summary_reviewText_wordsRatio', 'reviewText_negWordCount', 'itemID_numReviews', 'category_numtrans', 'votes_time', 'reviewText_capital
words', 'reviewerID_helpfulRate']
        Train Accuracy: 0.8684375
        Validation Accuracy: 0.867325
        Train MAE: 0.16644375
        Validation MAE: 0.1771
        Train MSE: 0.56099375
        Validation MSE: 0.9935


Features Eliminated: ['reviewText_count_char', 'price', 'reviewText_ExclQue_countchar', 'summary_reviewText_charRatio', 'reviewText_count_wo
rds', 'summary_ExclQue_countchar', 'unixReviewTime_delta_firstreview', 'categoryID', 'reviewText_count_firstCapital', 'summary_count_firstCa
pital', 'summary_capitalwords', 'summary_PunctChar_ratio', 'summary_count_punctu', 'unixReviewTime_delta_lastreview', 'summary_avgWordLengt
h', 'reviewText_count_punctu', 'summary_reviewText_wordsRatio', 'reviewText_negWordCount', 'itemID_numReviews', 'category_numtrans', 'votes_
time', 'reviewText_capitalwords', 'reviewerID_helpfulRate']
        Train Accuracy: 0.86693125
        Validation Accuracy: 0.8672
        Train MAE: 0.16788125
        Validation MAE: 0.178075
        Train MSE: 0.54553125
        Validation MSE: 1.050725


Features Eliminated: ['reviewText_PunctChar_ratio', 'reviewText_count_char', 'price', 'reviewText_ExclQue_countchar', 'summary_reviewText_ch
arRatio', 'reviewText_count_words', 'summary_ExclQue_countchar', 'unixReviewTime_delta_firstreview', 'categoryID', 'reviewText_count_firstCa
pital', 'summary_count_firstCapital', 'summary_capitalwords', 'summary_PunctChar_ratio', 'summary_count_punctu', 'unixReviewTime_delta_lastr
eview', 'summary_avgWordLength', 'reviewText_count_punctu', 'summary_reviewText_wordsRatio', 'reviewText_negWordCount', 'itemID_numReviews',
'category_numtrans', 'votes_time', 'reviewText_capitalwords', 'reviewerID_helpfulRate']
        Train Accuracy: 0.8660875
        Validation Accuracy: 0.8668
        Train MAE: 0.16935
        Validation MAE: 0.178025
        Train MSE: 0.5467
        Validation MSE: 1.176475


Features Eliminated: ['summary_negWordCount', 'reviewText_PunctChar_ratio', 'reviewText_count_char', 'price', 'reviewText_ExclQue_countcha
```

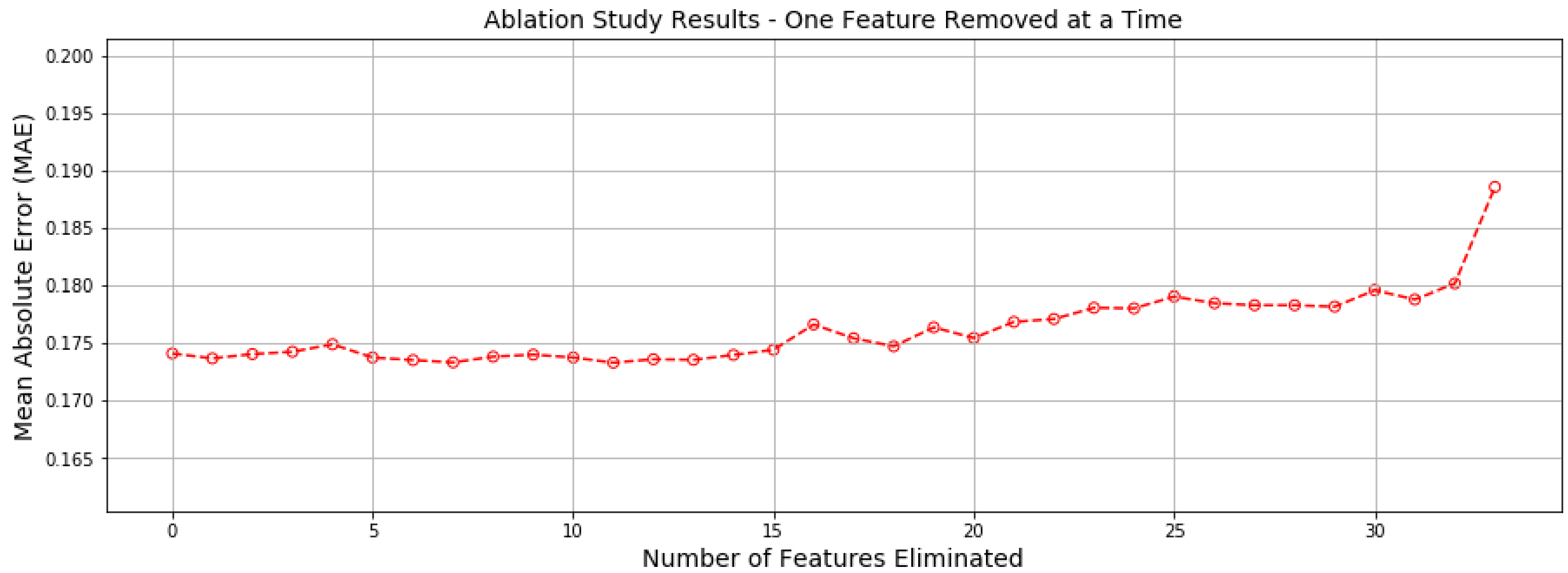
```
r', 'summary_reviewText_charRatio', 'reviewText_count_words', 'summary_ExclQue_countchar', 'unixReviewTime_delta_firstreview', 'categoryID',
'reviewText_count_firstCapital', 'summary_count_firstCapital', 'summary_capitalwords', 'summary_PunctChar_ratio', 'summary_count_punctu', 'u
nixReviewTime_delta_lastreview', 'summary_avgWordLength', 'reviewText_count_punctu', 'summary_reviewText_wordsRatio', 'reviewText_negWordCou
nt', 'itemID_numReviews', 'category_numtrans', 'votes_time', 'reviewText_capitalwords', 'reviewerID_helpfulRate']
        Train Accuracy: 0.8662375
        Validation Accuracy: 0.8667
        Train MAE: 0.168975
        Validation MAE: 0.17905
        Train MSE: 0.52705
        Validation MSE: 1.25515


Features Eliminated: ['reviewText_posWordCount', 'summary_negWordCount', 'reviewText_PunctChar_ratio', 'reviewText_count_char', 'price', 're
viewText_ExclQue_countchar', 'summary_reviewText_charRatio', 'reviewText_count_words', 'summary_ExclQue_countchar', 'unixReviewTime_delta_fi
rstreview', 'categoryID', 'reviewText_count_firstCapital', 'summary_count_firstCapital', 'summary_capitalwords', 'summary_PunctChar_ratio',
'summary_count_punctu', 'unixReviewTime_delta_lastreview', 'summary_avgWordLength', 'reviewText_count_punctu', 'summary_reviewText_wordsRati
o', 'reviewText_negWordCount', 'itemID_numReviews', 'category_numtrans', 'votes_time', 'reviewText_capitalwords', 'reviewerID_helpfulRate']
        Train Accuracy: 0.866
        Validation Accuracy: 0.867
        Train MAE: 0.17045
        Validation MAE: 0.178475
        Train MSE: 0.5603875
        Validation MSE: 1.141925


Features Eliminated: ['summary_negWordRate', 'reviewText_posWordCount', 'summary_negWordCount', 'reviewText_PunctChar_ratio', 'reviewText_co
unt_char', 'price', 'reviewText_ExclQue_countchar', 'summary_reviewText_charRatio', 'reviewText_count_words', 'summary_ExclQue_countchar',
'unixReviewTime_delta_firstreview', 'categoryID', 'reviewText_count_firstCapital', 'summary_count_firstCapital', 'summary_capitalwords', 'su
mmary_PunctChar_ratio', 'summary_count_punctu', 'unixReviewTime_delta_lastreview', 'summary_avgWordLength', 'reviewText_count_punctu', 'summ
ary_reviewText_wordsRatio', 'reviewText_negWordCount', 'itemID_numReviews', 'category_numtrans', 'votes_time', 'reviewText_capitalwords', 'r
eviewerID_helpfulRate']
        Train Accuracy: 0.86541875
        Validation Accuracy: 0.8664
        Train MAE: 0.1709375
        Validation MAE: 0.1783
        Train MSE: 0.5847125
        Validation MSE: 1.08385


Features Eliminated: ['rating', 'summary_negWordRate', 'reviewText_posWordCount', 'summary_negWordCount', 'reviewText_PunctChar_ratio', 'rev
iewText_count_char', 'price', 'reviewText_ExclQue_countchar', 'summary_reviewText_charRatio', 'reviewText_count_words', 'summary_ExclQue_cou
ntchar', 'unixReviewTime_delta_firstreview', 'categoryID', 'reviewText_count_firstCapital', 'summary_count_firstCapital', 'summary_capitalwo
rds', 'summary_PunctChar_ratio', 'summary_count_punctu', 'unixReviewTime_delta_lastreview', 'summary_avgWordLength', 'reviewText_count_punct
u', 'summary_reviewText_wordsRatio', 'reviewText_negWordCount', 'itemID_numReviews', 'category_numtrans', 'votes_time', 'reviewText_capitalw
ords', 'reviewerID_helpfulRate']
```

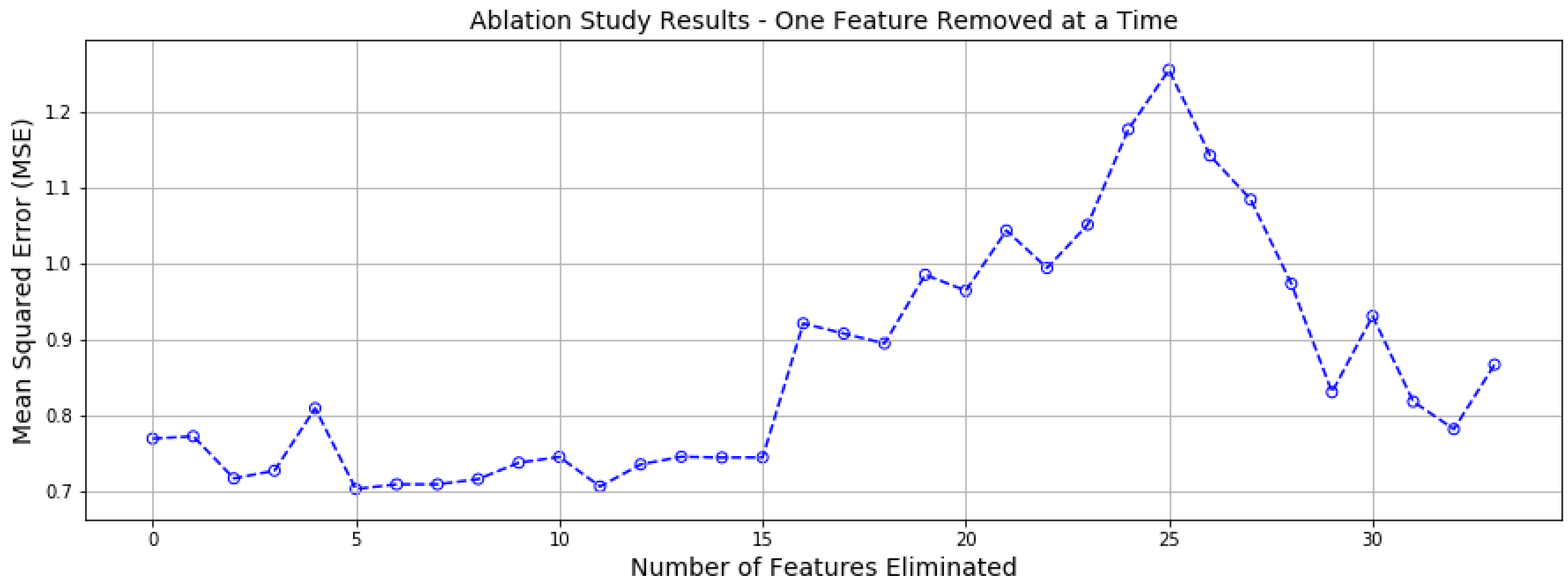
```
        Train Accuracy: 0.86361875
        Validation Accuracy: 0.864475
        Train MAE: 0.1736625
        Validation MAE: 0.1783
        Train MSE: 0.729225
        Validation MSE: 0.97285


Features Eliminated: ['reviewText_posWordRate', 'rating', 'summary_negWordRate', 'reviewText_posWordCount', 'summary_negWordCount', 'reviewT
ext_PunctChar_ratio', 'reviewText_count_char', 'price', 'reviewText_ExclQue_countchar', 'summary_reviewText_charRatio', 'reviewText_count_wo
rds', 'summary_ExclQue_countchar', 'unixReviewTime_delta_firstreview', 'categoryID', 'reviewText_count_firstCapital', 'summary_count_firstCa
pital', 'summary_capitalwords', 'summary_PunctChar_ratio', 'summary_count_punctu', 'unixReviewTime_delta_lastreview', 'summary_avgWordLengt
h', 'reviewText_count_punctu', 'summary_reviewText_wordsRatio', 'reviewText_negWordCount', 'itemID_numReviews', 'category_numtrans', 'votes_
time', 'reviewText_capitalwords', 'reviewerID_helpfulRate']
        Train Accuracy: 0.8612375
        Validation Accuracy: 0.863875
        Train MAE: 0.17938125
        Validation MAE: 0.178175
        Train MSE: 0.77003125
        Validation MSE: 0.830275


Features Eliminated: ['unixReviewTime', 'reviewText_posWordRate', 'rating', 'summary_negWordRate', 'reviewText_posWordCount', 'summary_negWo
rdCount', 'reviewText_PunctChar_ratio', 'reviewText_count_char', 'price', 'reviewText_ExclQue_countchar', 'summary_reviewText_charRatio', 'r
eviewText_count_words', 'summary_ExclQue_countchar', 'unixReviewTime_delta_firstreview', 'categoryID', 'reviewText_count_firstCapital', 'sum
mary_count_firstCapital', 'summary_capitalwords', 'summary_PunctChar_ratio', 'summary_count_punctu', 'unixReviewTime_delta_lastreview', 'sum
mary_avgWordLength', 'reviewText_count_punctu', 'summary_reviewText_wordsRatio', 'reviewText_negWordCount', 'itemID_numReviews', 'category_n
umtrans', 'votes_time', 'reviewText_capitalwords', 'reviewerID_helpfulRate']
        Train Accuracy: 0.85985625
        Validation Accuracy: 0.863275
        Train MAE: 0.1816125
        Validation MAE: 0.1796
        Train MSE: 0.7793125
        Validation MSE: 0.9308



Features Eliminated: ['summary_posWordRate', 'unixReviewTime', 'reviewText_posWordRate', 'rating', 'summary_negWordRate', 'reviewText_posW
ordCount', 'summary_negWordCount', 'reviewText_PunctChar_ratio', 'reviewText_count_char', 'price', 'reviewText_ExclQue_countchar', 'summar
y_reviewText_charRatio', 'reviewText_count_words', 'summary_ExclQue_countchar', 'unixReviewTime_delta_firstreview', 'categoryID', 'reviewT
ext_count_firstCapital', 'summary_count_firstCapital', 'summary_capitalwords', 'summary_PunctChar_ratio', 'summary_count_punctu', 'unixRev
iewTime_delta_lastreview', 'summary_avgWordLength', 'reviewText_count_punctu', 'summary_reviewText_wordsRatio', 'reviewText_negWordCount',
'itemID_numReviews', 'category_numtrans', 'votes_time', 'reviewText_capitalwords', 'reviewerID_helpfulRate']
        Train Accuracy: 0.8595125
```

```
        Validation Accuracy: 0.864
        Train MAE: 0.1830875
        Validation MAE: 0.1788
        Train MSE: 0.8068875
        Validation MSE: 0.81785


Features Eliminated: ['summary_posWordCount', 'summary_posWordRate', 'unixReviewTime', 'reviewText_posWordRate', 'rating', 'summary_negWor
dRate', 'reviewText_posWordCount', 'summary_negWordCount', 'reviewText_PunctChar_ratio', 'reviewText_count_char', 'price', 'reviewText_Exc
lQue_countchar', 'summary_reviewText_charRatio', 'reviewText_count_words', 'summary_ExclQue_countchar', 'unixReviewTime_delta_firstrevie
w', 'categoryID', 'reviewText_count_firstCapital', 'summary_count_firstCapital', 'summary_capitalwords', 'summary_PunctChar_ratio', 'summa
ry_count_punctu', 'unixReviewTime_delta_lastreview', 'summary_avgWordLength', 'reviewText_count_punctu', 'summary_reviewText_wordsRatio',
'reviewText_negWordCount', 'itemID_numReviews', 'category_numtrans', 'votes_time', 'reviewText_capitalwords', 'reviewerID_helpfulRate']
        Train Accuracy: 0.858275
        Validation Accuracy: 0.86325
        Train MAE: 0.1827875
        Validation MAE: 0.18015
        Train MSE: 0.77225
        Validation MSE: 0.7815


Features Eliminated: ['rating_deviation', 'summary_posWordCount', 'summary_posWordRate', 'unixReviewTime', 'reviewText_posWordRate', 'rati
ng', 'summary_negWordRate', 'reviewText_posWordCount', 'summary_negWordCount', 'reviewText_PunctChar_ratio', 'reviewText_count_char', 'pri
ce', 'reviewText_ExclQue_countchar', 'summary_reviewText_charRatio', 'reviewText_count_words', 'summary_ExclQue_countchar', 'unixReviewTim
e_delta_firstreview', 'categoryID', 'reviewText_count_firstCapital', 'summary_count_firstCapital', 'summary_capitalwords', 'summary_PunctC
har_ratio', 'summary_count_punctu', 'unixReviewTime_delta_lastreview', 'summary_avgWordLength', 'reviewText_count_punctu', 'summary_review
Text_wordsRatio', 'reviewText_negWordCount', 'itemID_numReviews', 'category_numtrans', 'votes_time', 'reviewText_capitalwords', 'reviewerI
D_helpfulRate']
        Train Accuracy: 0.85410625
        Validation Accuracy: 0.852775
        Train MAE: 0.1873
        Validation MAE: 0.18855
        Train MSE: 0.6837875
        Validation MSE: 0.8667


CPU times: user 30min 25s, sys: 14.3 s, total: 30min 39s
Wall time: 9min 39s
```

Figure 1. Ablation study results, MAE and MSE.

In [32]:
```python
# plot metrics vs. number of features removed

# get number of features and mae values
feature_counts = [len(i[2]) if i[2] != 'None' else 0 for i in features_list_mae_mse]
mae_values = [i[0] for i in features_list_mae_mse]
mse_values = [i[1] for i in features_list_mae_mse]

# plot MAE
plt.figure(figsize=(15,5)) # define figure
plt.scatter(feature_counts, mae_values, facecolors='none', edgecolors='r')
plt.plot(feature_counts, mae_values, '--', color='r')
plt.title('Ablation Study Results - One Feature Removed at a Time', fontsize=14)
plt.xlabel('Number of Features Eliminated', fontsize=14)
plt.ylabel('Mean Absolute Error (MAE)', fontsize=14)
plt.grid()
plt.show()

# plot MSE
plt.figure(figsize=(15,5)) # define figure
plt.scatter(feature_counts, mse_values, facecolors='none', edgecolors='b')
plt.plot(feature_counts, mse_values, '--', color='b')
plt.title('Ablation Study Results - One Feature Removed at a Time', fontsize=14)
plt.xlabel('Number of Features Eliminated', fontsize=14)
plt.ylabel('Mean Squared Error (MSE)', fontsize=14)
plt.grid()
plt.show()
```


Ablation Study Results - One Feature Removed at a Time
Mean Absolute Error (MAE)
0.200
0.195
0.190
0.185
0.180
0.175
0.170
0.165
0
5
10
15
20
25
30
Number of Features Eliminated


Ablation Study Results - One Feature Removed at a Time
Mean Squared Error (MSE)
1.2
1.1
1.0
0.9
0.8
0.7
0
5
10
15
20
25
30
Number of Features Eliminated

In [33]:
```python
# define features to keep based on ablation study results, criteria = lowest MAE
features_tokeep = set(all_features) - set(sorted(features_list_mae_mse, key= lambda x: x[0])[0][2])

print('A total of {} features were eliminated and not included downstream'
          .format(len(all_features)-len(features_tokeep)))
print('\n{} new features were selected, they are listed below:'.format(len(features_tokeep)-3))
print(features_tokeep)
```

```
A total of 11 features were eliminated and not included downstream

22 new features were selected, they are listed below:
{'itemID', 'reviewText_count_char', 'summary_negWordCount', 'summary_PunctChar_ratio', 'summary_negWordRate', 'price', 'reviewerID', 'summar
y_capitalwords', 'reviewText_posWordRate', 'reviewText_ExclQue_countchar', 'outOf_feature', 'unixReviewTime', 'rating_deviation', 'summary_p
osWordCount', 'reviewText_PunctChar_ratio', 'summary_posWordRate', 'summary_ExclQue_countchar', 'categoryID', 'reviewText_posWordCount', 'un
ixReviewTime_delta_firstreview', 'summary_count_firstCapital', 'summary_reviewText_charRatio', 'rating', 'reviewText_count_words', 'reviewTe
xt_count_firstCapital'}
```

Hyperparameter Tuning - Traning and Validation Datasets

In [34]:
```
%%time

# run test and validation dataset to tune parameters: max_depth, n_estimators
hyper_parameter_list = []
y_pred_val_list = []
for max_depth in [10,12]:
    for n_estimators in [100, 200, 400, 600]:

        # get metrics (train_accuracy, test_accuracy, train_mae, test_mae, train_mse, test_mse, y_pred_test)
        metrics_tuple = perfmetrics_RFmodel(max_depth, n_estimators, X_train_filt[features_tokeep],
                                            y_train_ratio_filt, X_train[features_tokeep],
                                            y_train, X_val[features_tokeep], y_val)
        # print metrics
        print('max_depth: {}, n_estimators:{}'.format(max_depth, n_estimators))
        print('\tTrain Accuracy: {}'.format(metrics_tuple[0]))
        print('\tValidation Accuracy: {}'.format(metrics_tuple[1]))
        print('\tTrain MAE: {}'.format(metrics_tuple[2]))
        print('\tValidation MAE: {}'.format(metrics_tuple[3]))
        print('\tTrain MSE: {}'.format(metrics_tuple[4]))
        print('\tValidation MSE: {}'.format(metrics_tuple[5]))
        print('\n')

        # save hyperparameters, mae, and predictions
        hyper_parameter_list.append((max_depth, n_estimators, metrics_tuple[3], metrics_tuple[6]))
```

```
max_depth: 10, n_estimators:100
	Train Accuracy: 0.87045625
	Validation Accuracy: 0.86675
	Train MAE: 0.1618625
	Validation MAE: 0.172775
	Train MSE: 0.5269875
	Validation MSE: 0.668575


max_depth: 10, n_estimators:200
	Train Accuracy: 0.87031875
	Validation Accuracy: 0.86695
	Train MAE: 0.1617375
	Validation MAE: 0.1735
	Train MSE: 0.492075
	Validation MSE: 0.7034


max_depth: 10, n_estimators:400
```

```
        Train Accuracy: 0.87043125
        Validation Accuracy: 0.867
        Train MAE: 0.1617375
        Validation MAE: 0.1734
        Train MSE: 0.4919125
        Validation MSE: 0.70715


max_depth: 10, n_estimators:600
        Train Accuracy: 0.8703125
        Validation Accuracy: 0.867025
        Train MAE: 0.1616375
        Validation MAE: 0.17395
        Train MSE: 0.489175
        Validation MSE: 0.7656


max_depth: 12, n_estimators:100
        Train Accuracy: 0.8752625
        Validation Accuracy: 0.8676
        Train MAE: 0.15469375
        Validation MAE: 0.173075
        Train MSE: 0.48120625
        Validation MSE: 0.726925


max_depth: 12, n_estimators:200
        Train Accuracy: 0.875175
        Validation Accuracy: 0.867475
        Train MAE: 0.15435625
        Validation MAE: 0.172575
        Train MSE: 0.43398125
        Validation MSE: 0.701275


max_depth: 12, n_estimators:400
        Train Accuracy: 0.87535
        Validation Accuracy: 0.8675
        Train MAE: 0.15424375
        Validation MAE: 0.1726
        Train MSE: 0.46011875
        Validation MSE: 0.7209


max_depth: 12, n_estimators:600
```

```
    Train Accuracy: 0.87516875
    Validation Accuracy: 0.867825
    Train MAE: 0.15446875
    Validation MAE: 0.172625
    Train MSE: 0.46076875
    Validation MSE: 0.702175


CPU times: user 30min 28s, sys: 10.9 s, total: 30min 39s
Wall time: 9min
```

In [35]:
```
# select best hyper parameters for test data evaluation
best_params = sorted(hyper_parameter_list, key = lambda x: x[2])[0]

max_depth = best_params[0]
n_estimators = best_params[1]
```

Evaluate Classification on Validation Data - Precision and Recall

In [36]:
```
# get predictions on validation data
y_pred_val = [i for i in hyper_parameter_list if i[0] == max_depth and i[1] == n_estimators][0][3]

len(y_pred_val)
```

Out[36]:
```
40000
```

In [37]:
```python
# compute confusion matrix
cnf_matrix = confusion_matrix(y_val, y_pred_val)

# compute evaluation metrics
FP = cnf_matrix.sum(axis=0) - np.diag(cnf_matrix)
FN = cnf_matrix.sum(axis=1) - np.diag(cnf_matrix)
TP = np.diag(cnf_matrix)
TN = cnf_matrix.sum() - (FP + FN + TP)

# TPR, sensitivity, or recall
TPR = TP / (TP + FN)

# TNR or specificity
TNR = TN / (TN + FP)

# precision
precision = TP / (TP + FP)

# FPR
FPR = FP / (FP + TN)

# FNR
FNR = FN / (TP + FN)

# accuracy
accuracy = (TP+TN)/(TP+FP+FN+TN)
```

```
/Users/gio/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:11: RuntimeWarning: invalid value encountered in true_divide
  # This is added back by InteractiveShellApp.init_path()
/Users/gio/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:17: RuntimeWarning: invalid value encountered in true_divide
/Users/gio/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:23: RuntimeWarning: invalid value encountered in true_divide
```

Table 3. Confusion matrix and classification metrics for each number of vote (outOf feature, label).

In [38]:
```python
# create dataframe
columns = ['FP', 'FN', 'TP', 'TN', 'TPR', 'TNR', 'precision', 'FPR', 'FNR', 'accuracy']
rows = set(y_val).union(set(y_pred_val))
data = list(zip(FP, FN, TP, TN, TPR, TNR, precision, FPR, FNR, accuracy))
df = pd.DataFrame(data=data, index=rows, columns=columns)
df.index.names = ['Label (Number of Votes)']

# print
pd.set_option('display.max_rows', df.shape[0]+1)
df
```

Out[38]:

Label (Number of Votes)	FP	FN	TP	TN	TPR	TNR	precision	FPR	FNR	accuracy
0	36	1530	27595	10839	0.947468	0.996690	0.998697	0.003310	0.052532	0.960850
1	1928	625	4689	32758	0.882386	0.944416	0.708629	0.055584	0.117614	0.936175
2	863	795	1034	37308	0.565336	0.977391	0.545071	0.022609	0.434664	0.958550
3	620	387	651	38342	0.627168	0.984087	0.512195	0.015913	0.372832	0.974825
4	397	360	219	39024	0.378238	0.989929	0.355519	0.010071	0.621762	0.981075
5	182	315	89	39414	0.220297	0.995404	0.328413	0.004596	0.779703	0.987575
6	209	169	101	39521	0.374074	0.994739	0.325806	0.005261	0.625926	0.990550
7	138	161	56	39645	0.258065	0.996531	0.288660	0.003469	0.741935	0.992525
8	111	109	46	39734	0.296774	0.997214	0.292994	0.002786	0.703226	0.994500
9	82	97	29	39792	0.230159	0.997944	0.261261	0.002056	0.769841	0.995525
10	67	88	28	39817	0.241379	0.998320	0.294737	0.001680	0.758621	0.996125
11	71	59	32	39838	0.351648	0.998221	0.310680	0.001779	0.648352	0.996750
12	44	65	13	39878	0.166667	0.998898	0.228070	0.001102	0.833333	0.997275
13	49	49	14	39888	0.222222	0.998773	0.222222	0.001227	0.777778	0.997550
14	38	41	9	39912	0.180000	0.999049	0.191489	0.000951	0.820000	0.998025
15	46	34	13	39907	0.276596	0.998849	0.220339	0.001151	0.723404	0.998000
16	35	48	7	39910	0.127273	0.999124	0.166667	0.000876	0.872727	0.997925
17	28	22	10	39940	0.312500	0.999299	0.263158	0.000701	0.687500	0.998750

Label (Number of Votes)	FP	FN	TP	TN	TPR	TNR	precision	FPR	FNR	accuracy
18	25	27	6	39942	0.181818	0.999374	0.193548	0.000626	0.818182	0.998700
19	28	16	8	39948	0.333333	0.999300	0.222222	0.000700	0.666667	0.998900
20	15	17	4	39964	0.190476	0.999625	0.210526	0.000375	0.809524	0.999200
21	18	18	5	39959	0.217391	0.999550	0.217391	0.000450	0.782609	0.999100
22	22	13	6	39959	0.315789	0.999450	0.214286	0.000550	0.684211	0.999125
23	15	15	3	39967	0.166667	0.999625	0.166667	0.000375	0.833333	0.999250
24	13	14	1	39972	0.066667	0.999675	0.071429	0.000325	0.933333	0.999325
25	15	17	4	39964	0.190476	0.999625	0.210526	0.000375	0.809524	0.999200
26	6	12	1	39981	0.076923	0.999850	0.142857	0.000150	0.923077	0.999550
27	14	10	3	39973	0.230769	0.999650	0.176471	0.000350	0.769231	0.999400
28	11	10	3	39976	0.230769	0.999725	0.214286	0.000275	0.769231	0.999475
29	14	9	2	39975	0.181818	0.999650	0.125000	0.000350	0.818182	0.999425
30	9	12	0	39979	0.000000	0.999775	0.000000	0.000225	1.000000	0.999475
31	7	9	0	39984	0.000000	0.999825	0.000000	0.000175	1.000000	0.999600
32	8	9	3	39980	0.250000	0.999800	0.272727	0.000200	0.750000	0.999575
33	4	11	0	39985	0.000000	0.999900	0.000000	0.000100	1.000000	0.999625
34	4	5	1	39990	0.166667	0.999900	0.200000	0.000100	0.833333	0.999775
35	5	2	2	39991	0.500000	0.999875	0.285714	0.000125	0.500000	0.999825
36	12	2	3	39983	0.600000	0.999700	0.200000	0.000300	0.400000	0.999650
37	4	9	0	39987	0.000000	0.999900	0.000000	0.000100	1.000000	0.999675
38	7	8	1	39984	0.111111	0.999825	0.125000	0.000175	0.888889	0.999625
39	2	3	0	39995	0.000000	0.999950	0.000000	0.000050	1.000000	0.999875
40	3	3	0	39994	0.000000	0.999925	0.000000	0.000075	1.000000	0.999850
41	2	4	1	39993	0.200000	0.999950	0.333333	0.000050	0.800000	0.999850
42	4	2	0	39994	0.000000	0.999900	0.000000	0.000100	1.000000	0.999850
43	4	1	0	39995	0.000000	0.999900	0.000000	0.000100	1.000000	0.999875

Label (Number of Votes)	FP	FN	TP	TN	TPR	TNR	precision	FPR	FNR	accuracy
44	4	5	0	39991	0.000000	0.999900	0.000000	0.000100	1.000000	0.999775
45	8	3	1	39988	0.250000	0.999800	0.111111	0.000200	0.750000	0.999725
46	4	4	1	39991	0.200000	0.999900	0.200000	0.000100	0.800000	0.999800
47	2	4	0	39994	0.000000	0.999950	0.000000	0.000050	1.000000	0.999850
48	3	4	1	39992	0.200000	0.999925	0.250000	0.000075	0.800000	0.999825
49	1	3	0	39996	0.000000	0.999975	0.000000	0.000025	1.000000	0.999900
50	0	3	1	39996	0.250000	1.000000	1.000000	0.000000	0.750000	0.999925
51	1	2	0	39997	0.000000	0.999975	0.000000	0.000025	1.000000	0.999925
52	7	3	0	39990	0.000000	0.999825	0.000000	0.000175	1.000000	0.999750
53	4	3	0	39993	0.000000	0.999900	0.000000	0.000100	1.000000	0.999825
54	2	3	0	39995	0.000000	0.999950	0.000000	0.000050	1.000000	0.999875
55	3	4	0	39993	0.000000	0.999925	0.000000	0.000075	1.000000	0.999825
56	2	2	0	39996	0.000000	0.999950	0.000000	0.000050	1.000000	0.999900
57	1	1	0	39998	0.000000	0.999975	0.000000	0.000025	1.000000	0.999950
58	2	2	1	39995	0.333333	0.999950	0.333333	0.000050	0.666667	0.999900
59	2	1	0	39997	0.000000	0.999950	0.000000	0.000050	1.000000	0.999925
60	2	0	0	39998	NaN	0.999950	0.000000	0.000050	NaN	0.999950
62	1	3	0	39996	0.000000	0.999975	0.000000	0.000025	1.000000	0.999900
63	1	0	0	39999	NaN	0.999975	0.000000	0.000025	NaN	0.999975
64	0	4	0	39996	0.000000	1.000000	NaN	0.000000	1.000000	0.999900
65	1	0	0	39999	NaN	0.999975	0.000000	0.000025	NaN	0.999975
66	1	2	0	39997	0.000000	0.999975	0.000000	0.000025	1.000000	0.999925
67	1	0	0	39999	NaN	0.999975	0.000000	0.000025	NaN	0.999975
68	1	0	0	39999	NaN	0.999975	0.000000	0.000025	NaN	0.999975
69	0	1	0	39999	0.000000	1.000000	NaN	0.000000	1.000000	0.999975
70	1	0	0	39999	NaN	0.999975	0.000000	0.000025	NaN	0.999975

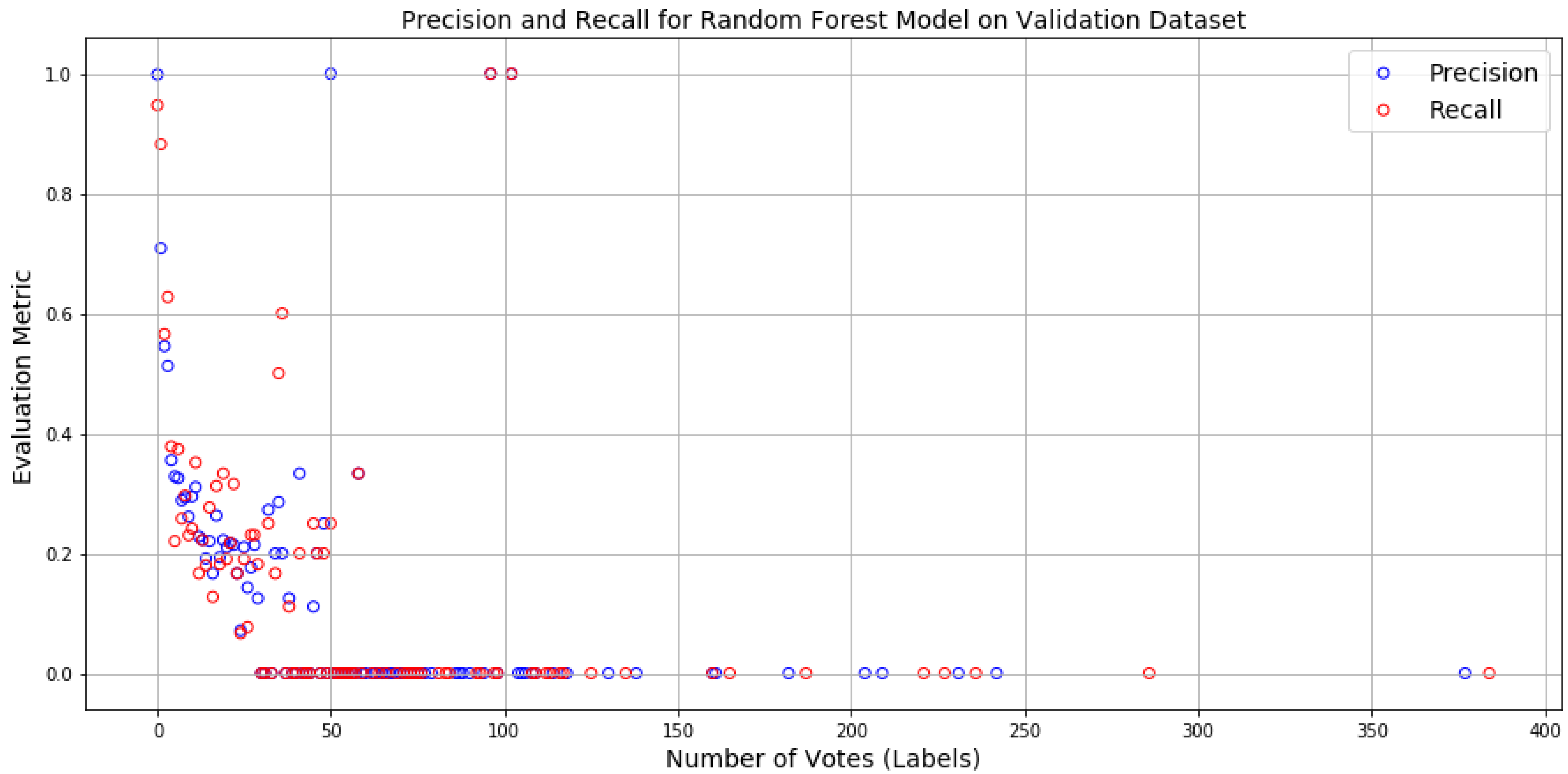
	FP	FN	TP	TN	TPR	TNR	precision	FPR	FNR	accuracy
Label (Number of Votes)										
71	1	1	0	39998	0.000000	0.999975	0.000000	0.000025	1.000000	0.999950
72	1	2	0	39997	0.000000	0.999975	0.000000	0.000025	1.000000	0.999925
73	1	1	0	39998	0.000000	0.999975	0.000000	0.000025	1.000000	0.999950
74	0	1	0	39999	0.000000	1.000000	NaN	0.000000	1.000000	0.999975
75	1	1	0	39998	0.000000	0.999975	0.000000	0.000025	1.000000	0.999950
76	1	1	0	39998	0.000000	0.999975	0.000000	0.000025	1.000000	0.999950
77	1	0	0	39999	NaN	0.999975	0.000000	0.000025	NaN	0.999975
79	2	0	0	39998	NaN	0.999950	0.000000	0.000050	NaN	0.999950
81	0	1	0	39999	0.000000	1.000000	NaN	0.000000	1.000000	0.999975
83	0	1	0	39999	0.000000	1.000000	NaN	0.000000	1.000000	0.999975
84	0	1	0	39999	0.000000	1.000000	NaN	0.000000	1.000000	0.999975
86	1	0	0	39999	NaN	0.999975	0.000000	0.000025	NaN	0.999975
87	1	0	0	39999	NaN	0.999975	0.000000	0.000025	NaN	0.999975
88	1	0	0	39999	NaN	0.999975	0.000000	0.000025	NaN	0.999975
90	1	0	0	39999	NaN	0.999975	0.000000	0.000025	NaN	0.999975
92	0	3	0	39997	0.000000	1.000000	NaN	0.000000	1.000000	0.999925
93	0	1	0	39999	0.000000	1.000000	NaN	0.000000	1.000000	0.999975
94	1	0	0	39999	NaN	0.999975	0.000000	0.000025	NaN	0.999975
96	0	0	1	39999	1.000000	1.000000	1.000000	0.000000	0.000000	1.000000
97	0	1	0	39999	0.000000	1.000000	NaN	0.000000	1.000000	0.999975
98	1	1	0	39998	0.000000	0.999975	0.000000	0.000025	1.000000	0.999950
102	0	0	1	39999	1.000000	1.000000	1.000000	0.000000	0.000000	1.000000
104	1	0	0	39999	NaN	0.999975	0.000000	0.000025	NaN	0.999975
105	1	0	0	39999	NaN	0.999975	0.000000	0.000025	NaN	0.999975
106	1	0	0	39999	NaN	0.999975	0.000000	0.000025	NaN	0.999975
108	1	1	0	39998	0.000000	0.999975	0.000000	0.000025	1.000000	0.999950

Label (Number of Votes)	FP	FN	TP	TN	TPR	TNR	precision	FPR	FNR	accuracy
109	1	1	0	39998	0.000000	0.999975	0.000000	0.000025	1.000000	0.999950
112	0	1	0	39999	0.000000	1.000000	NaN	0.000000	1.000000	0.999975
113	0	1	0	39999	0.000000	1.000000	NaN	0.000000	1.000000	0.999975
114	1	0	0	39999	NaN	0.999975	0.000000	0.000025	NaN	0.999975
116	0	1	0	39999	0.000000	1.000000	NaN	0.000000	1.000000	0.999975
117	0	1	0	39999	0.000000	1.000000	NaN	0.000000	1.000000	0.999975
118	1	0	0	39999	NaN	0.999975	0.000000	0.000025	NaN	0.999975
125	0	1	0	39999	0.000000	1.000000	NaN	0.000000	1.000000	0.999975
130	1	0	0	39999	NaN	0.999975	0.000000	0.000025	NaN	0.999975
135	0	1	0	39999	0.000000	1.000000	NaN	0.000000	1.000000	0.999975
138	1	0	0	39999	NaN	0.999975	0.000000	0.000025	NaN	0.999975
160	1	1	0	39998	0.000000	0.999975	0.000000	0.000025	1.000000	0.999950
161	1	0	0	39999	NaN	0.999975	0.000000	0.000025	NaN	0.999975
165	0	1	0	39999	0.000000	1.000000	NaN	0.000000	1.000000	0.999975
182	1	0	0	39999	NaN	0.999975	0.000000	0.000025	NaN	0.999975
187	0	1	0	39999	0.000000	1.000000	NaN	0.000000	1.000000	0.999975
204	1	0	0	39999	NaN	0.999975	0.000000	0.000025	NaN	0.999975
209	1	0	0	39999	NaN	0.999975	0.000000	0.000025	NaN	0.999975
221	0	1	0	39999	0.000000	1.000000	NaN	0.000000	1.000000	0.999975
227	0	1	0	39999	0.000000	1.000000	NaN	0.000000	1.000000	0.999975
231	1	0	0	39999	NaN	0.999975	0.000000	0.000025	NaN	0.999975
236	0	1	0	39999	0.000000	1.000000	NaN	0.000000	1.000000	0.999975
242	1	0	0	39999	NaN	0.999975	0.000000	0.000025	NaN	0.999975
286	0	1	0	39999	0.000000	1.000000	NaN	0.000000	1.000000	0.999975
377	1	0	0	39999	NaN	0.999975	0.000000	0.000025	NaN	0.999975
384	0	1	0	39999	0.000000	1.000000	NaN	0.000000	1.000000	0.999975

Figure 2. Precision and recall for random forest model implementation (max_depth: 12, n_estimators: 200, number of features: 22).

In [39]:

```
# plot precision and recall

plt.figure(figsize=(15,7)) # define figure

# precision
plt.scatter(list(rows), precision, facecolors='none', edgecolors='b')

# recall
plt.scatter(list(rows), TPR, facecolors='none', edgecolors='r')

# add labels
plt.legend(['Precision', 'Recall'], fontsize=14)
plt.title('Precision and Recall for Random Forest Model on Validation Dataset', fontsize=14)
plt.xlabel('Number of Votes (Labels)', fontsize=14)
plt.ylabel('Evaluation Metric', fontsize=14)

# show plot
plt.grid()
plt.show()
```



Evaluate on Test

In [40]:
```
%%time

# get indices
idx = np.array([i['outOf'] for i in train_features.helpful]) > 0

# create features
X_train_val = Get_features(train_features)
X_test = Get_features(test)

# create labels
y_train_val = [i['nHelpful'] for i in train_labels]
y_test = None

# find filtered test data
X_train_val_filt = Get_features(train_features.loc[idx,:])
y_train_val_ratio_filt = Get_labels_ratio(list(compress(train_labels, idx)))
```

```
CPU times: user 4min 4s, sys: 4.61 s, total: 4min 8s
Wall time: 4min 11s
```

In [41]:
```
%%time

# (train_accuracy, train_mae, train_mse, y_pred_test)
metrics_tuple = perfmetrics_RFmodel(max_depth, n_estimators, X_train_val_filt[features_tokeep],
                                    y_train_val_ratio_filt, X_train_val[features_tokeep],
                                    y_train_val, X_test[features_tokeep], y_test)
# print metrics
print('max_depth: {}, n_estimators:{}'.format(max_depth, n_estimators))
print('Train Accuracy: {}'.format(metrics_tuple[0]))
print('Train MAE: {}'.format(metrics_tuple[1]))
print('Train MSE: {}'.format(metrics_tuple[2]))
print('\n')
```

```
max_depth: 12, n_estimators:200
Train Accuracy: 0.875125
Train MAE: 0.15479
Train MSE: 0.41986


CPU times: user 3min 18s, sys: 1.59 s, total: 3min 20s
Wall time: 1min 5s
```

In [42]:
```
# save predictions on file "predictions_Helpful.txt"
save_predictions(metrics_tuple[3])
```

In [ ]:
```
```