Student Names
IST 652 Scripting for Data

Amazon Reviews: Exactly How Helpful Are Reviews?

Amazonian Team Members:
- Neal Staalberg
- Kirby Hood

Purpose of Investigation:

Amazon.com is a place where many people go to shop for many different products. People often decide to make a purchase (online and offline) based on the reviews written about the products (and services) in which they are interested in. These reviews can be considered helpful or not and Amazon solicits feedback on reviews based off if people found the reviews to be helpful and for amazon it is a means of encouraging others to purchase the same product, weeding out bad and/or fraudulent products and prioritizing which reviews are presented to prospective customers.

Given that grading/labeling reviews is a very time intensive endeavor requiring a significant amount of purchasers electing to review items and then other individuals rating those reviews as helpful or not we decided to look into an algorithmic to evaluate the feasibility of labeling reviews based off of their predicted (prospective) helpfulness to others. The ultimate end goal of this analysis is to capitalize on a large and engaged customer base and the prioritize which reviews are most likely to factor into a purchase being made or not for product/service and provide relevant information for customers.

Holistically, this information creates a strong value add for both doing business with Amazon from a third party sellers point of view as potential customers will see both aggregate reviews (1-5 stars) and customer stories (leveraging brand loyalty and word of mouth). In addition to this are the benefits to amazon itself as this provides Amazon information on which brands and products will drive the most sales, minimize returns and generate the most traffic. To note, this information more than likely informed the product choice and manufacturing of Amazon's Basic product line which competes with products already offered by other sellers within Amazon's marketplace but adds to Amazon's bottom line.

Analysis Questions:

Can a prediction model generate the same classification of helpfulness as dozens or hundreds of prospective buyers? This would be beneficial to determine what reviews will be helpful before they are seen by potential Amazon customers and allow for prioritizing reviews which may positively boost sales , might represent defects/issues with the product or customer service opportunities for Amazon or it's third party sellers.

Can the prediction model of reviews, either helpful or not be applied to other businesses and their products to compare how their product reviews are similar to those from Amazon? Can the predictive model trained on electronics reviews generalize well to products in other categories?

Can a modestly accurate classification algorithm provide keywords that can guide or add value to the current human review process from existing Amazon shoppers?

Scope:

An analysis is to be conducted to determine the feasibility of predicting relative helpfulness or (helpful/unhelpful) ratings of online product reviews based off the review text entered by the user. Jupyter Notebooks and Rodeo were both used during the course of this project with code executed in python and pyspark (spark-2.4).

Data Preparation:

The Amazon Electronics reviews were first loaded in to both Jupiter Notebooks and Rodeo. There were several packages that were used to take the initial look at the overall data and prepare for further analysis. Some of these packages were numpy, pandas, seaborn, json, calendar, and datetime which facilitated the analysis and processing of the text data.

Two functions were also created to read the json file of the Amazon reviews. These functions first parsed the original path of the data, and then read the data in through the parse function that was created and creation of the dataframe. The original pandas dataframe that was created returned 1,689,188 rows and 9 columns. Using isna and isnull to check for missing values and NaN within the Amazon review data set returned 24,730 missing values and NaN. These missing values and NaN only represented 1.46% of the overall data within these reviews. It was chosen to remove these missing values from the original dataset. This resulted in 1,664,458 rows and 9 columns.

Binning was also done on the Amazon dataset to incorporate the helpfulness ratio (helpful/total reviews) of the reviews. Bins were created to categorize the helpfulness of the reviews regardless if the reviews were positive or negative. This created 3 bins 0, 1, and 2. The image below shows the min and max helpfulness ratio of each bin.

helpfulnessBucket	min(helpfulnessRatio)	max(helpfulnessRatio)
0.0	0.0	0.32894736842105265
1.0	0.3333333333333333	0.6694915254237288
2.0	0.6702702702702703	1.0

The sum of each bin was done to show how many reviews were categorized to each bin of helpfulness.

Sum of Reviews for each Helpfulness Bin

helpfulnessBucket	count
0.0	110000
1.0	121815
2.0	494146

Due to the size of the data and the significant skew of the classes a sample size was taken from original data set to perform further analysis. The data set after sample was taken consisted of 330,308 data points.

helpfulnessBucket	count
0.0	110000
1.0	110085
2.0	110223

Training and testing data sets were then created using spark.read.parquet.

Training Data Set

helpfulnessBucket	count
2.0	73944
1.0	73964
0.0	73585

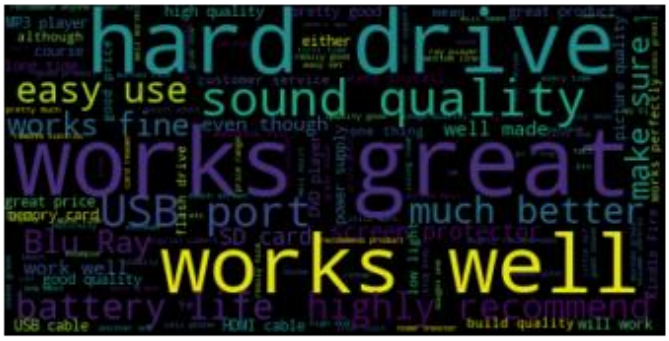
Test Data Set

helpfulnessBucket	count
2.0	35900
1.0	36182
0.0	36415

The Amazon reviews then went through another round of tokenizing, removal of stop words, and hashing.

Student Names
IST 652 Scripting for Data

Initial Data Exploration:
A word cloud was created to see the overall word frequency before more preparation was performed. This was executed to have an initial view of the most popular words to be able to set a base for comparison of further word clouds and prediction models.



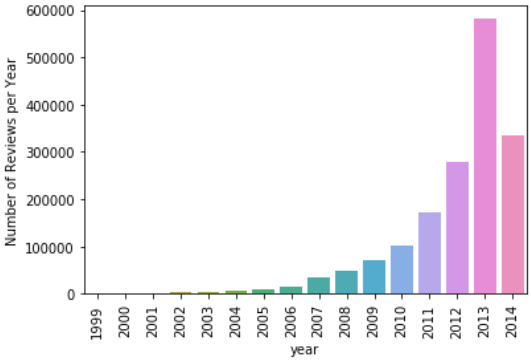
From this initial word cloud we can see that words such as “works well”, “works great”, “hard drive”, “sound quality”, and “easy use” are terms that seem fairly positive and could deal with maybe computers or sound systems. Further analysis would need to be conducted to determine exactly what each product was that is being described as that is not included within the data that was retrieved (only the proprietary ASIN# was provided).

The data was cleaned more following this initial word cloud. All words were made lowercase, punctuation was removed as well as stop words. The top 50 common words were evaluated, as these words can be removed because they will not be significant in further analysis. This was also done for the rare words. All misspelled words were corrected, and tokenization applied. Bigrams we also created using TextBlob.

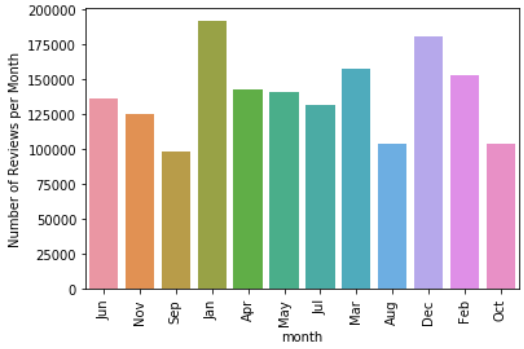
Next, the term frequency was counted to see the top words now within the dataset.

Several other columns were created and added to the dataframe. The columns extracted information from the timestamp in which the Amazon reviews were done. The columns that were added were for the year the review was created, the day and the month. These were then plotted to see if this would tell us about the frequency distribution of the time that the reviews were done by users.

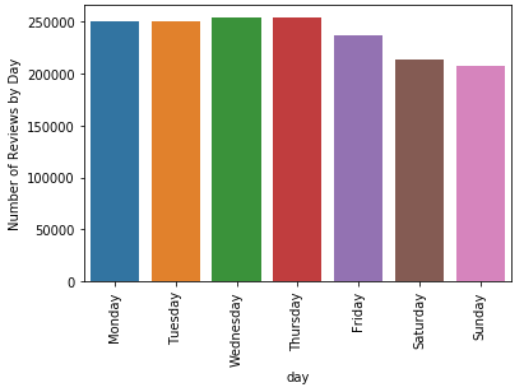
Number of Reviews per Year



Number of Reviews per Month



Number of Reviews per Day

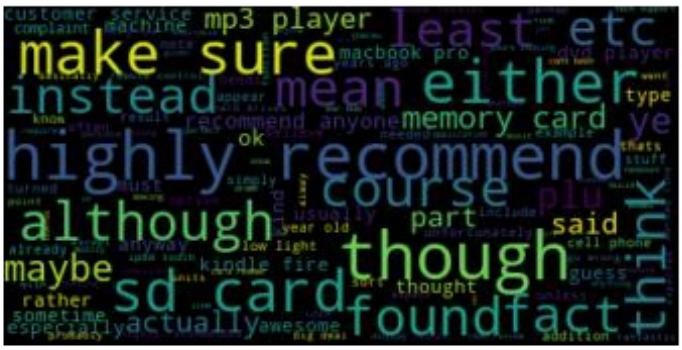


From the above visualizations above this indicates that in 2013 there must have been very hot electronic products offered online, or a surge of people to Amazon to start purchasing items here instead of traditional retailers. Also, December and January are the two months in which most reviews were written. This too indicates that people got many electronic items around the holidays, probably as presents and wrote the reviews after setting them up and using them. When looking at the number of reviews by day, this is very intuitive that many people might have Amazon Prime and order things on Friday, and they are delivered over the weekend on Sundays and reviews are written at the earlier parts of the week.

A sentiment analysis was done to show a give value to the reviews. Using the overall column of the Amazon data set containing the ratings, new columns were added to the dataframe to indicate if the rating was good, neutral, or bad. The ratings were originally on a scale of 1 – 5 and the columns were created using ratings of 4 and 5 for good, 3 for neutral, and 2 and 1 for the bad columns.

Using the new columns, three more word clouds were created to visually see if there is a difference within the words used in the different review ratings.

Good Word Cloud



Neutral Word Cloud

Bad Word Cloud

Further Data Exploration:

Random Forest Classification:

After initial cleaning of the Amazon Review data set and examining the word clouds, a Random Forest model was created to predict the evaluations of the reviews. The Random Forest model resulted in the creation of a prediction column of the bin in which the review would be placed, if the review was helpful, and the probability that this would occur.

The Random Forest model was evaluated using MulticlassClassificationEvaluator from pyspark for accuracy. The training set resulted in an accuracy rate of 42.20% and the test set had an accuracy of 42.10%. These are very close in their accuracy levels. Below shows the prediction of each helpfulness bin as well as the actual helpfulness bin from the Random Forest Model

Random Forest Classification Probability Outcomes of Testing Data:

helpfulnessBucket	prediction	helpful	probability
2.0	0.0	[3, 3]	[0.35891642397244...
2.0	2.0	[6, 6]	[0.27689797845050...
2.0	0.0	[2, 2]	[0.35646111625804...
2.0	0.0	[1, 1]	[0.35819608175869...
2.0	1.0	[5, 5]	[0.31788998177583...
2.0	0.0	[4, 4]	[0.38773555915324...
2.0	0.0	[1, 1]	[0.36621724705657...
2.0	1.0	[3, 3]	[0.32351453141812...
2.0	0.0	[1, 1]	[0.38078040049001...
2.0	1.0	[3, 3]	[0.33714283673766...
2.0	2.0	[11, 12]	[0.29158368687517...
2.0	2.0	[1, 1]	[0.31053774123212...
2.0	0.0	[3, 3]	[0.35625376854951...
2.0	0.0	[1, 1]	[0.38868291205595...
2.0	0.0	[6, 7]	[0.38269502645233...
2.0	1.0	[16, 16]	[0.30961703628264...
2.0	2.0	[1, 1]	[0.24813524759229...
2.0	2.0	[5, 6]	[0.24436377079670...
2.0	1.0	[1, 1]	[0.32942843730071...
2.0	2.0	[10, 10]	[0.25361662541302...

only showing top 20 rows

Random Forest Predictions:

helpfulnessBucket	prediction	count
2.0	2.0	15014
2.0	1.0	3959
2.0	0.0	16927
1.0	2.0	12298
1.0	1.0	4489
1.0	0.0	19395
0.0	2.0	6780
0.0	1.0	3469
0.0	0.0	26166

From these results we can see that there were more reviews that were predicted incorrectly for the 2.0 helpfulness bin, as well as the 1.0 and 0.0 bins.

Logistic Regression:

A Logistic Regression model was then created using the same helpfulness bins to predict if the review will be helpful to someone when making the purchase.

When the model ran using the training model, it produced an accuracy rate of 45.90% and the test data returned a 45.00% accuracy rate. This accuracy rates of the logistic model were better than the Random Forest model that was previously created, but there are still parameters that need further tuning. The visualization below shows the counts of the correctly predicted helpfulness of the reviews that were done by users on electronics bought from Amazon.

Logistic Regression Probability Outcomes of Testing Data:

helpfulnessBucket	prediction	helpful	probability
2.0	0.0	[3, 3]	[0.42089876721117...
2.0	2.0	[6, 6]	[0.19964061612559...
2.0	0.0	[2, 2]	[0.40076839151984...
2.0	0.0	[1, 1]	[0.45524505614658...
2.0	0.0	[5, 5]	[0.45636613388866...
2.0	1.0	[4, 4]	[0.31876451721533...
2.0	0.0	[1, 1]	[0.42303660160205...
2.0	1.0	[3, 3]	[0.25782900456960...
2.0	0.0	[1, 1]	[0.36422653515408...
2.0	2.0	[3, 3]	[0.20208792512383...
2.0	2.0	[11, 12]	[0.20605338428404...
2.0	0.0	[1, 1]	[0.33557264812448...
2.0	2.0	[3, 3]	[0.31997892711091...
2.0	1.0	[1, 1]	[0.24275213336486...
2.0	2.0	[6, 7]	[0.27357770968880...
2.0	2.0	[16, 16]	[0.22391820569913...
2.0	1.0	[1, 1]	[0.13481180542337...
2.0	2.0	[5, 6]	[0.11334368897644...
2.0	0.0	[1, 1]	[0.42905551344645...
2.0	2.0	[10, 10]	[0.13585466321949...

only showing top 20 rows

Logistic Regression Predictions:

helpfulnessBucket	prediction	count
2.0	2.0	15548
2.0	1.0	6540
2.0	0.0	13812
1.0	2.0	10544
1.0	1.0	8312
1.0	0.0	17326
0.0	2.0	6418
0.0	1.0	5033
0.0	0.0	24964

The predictions made by the Logistic Regression, although were better than the Random Forest, are still lacking in certain parameters. However, as each model is created and trained and tested,

there is the potential that a pattern is emerging even without having fine tuned parameters for better accuracy rates.

Naive Bayes Classification:

The Naive Bayes Classification model based mostly on probability and was the first model to ever use probability within the methods of doing predictions. This model can be considered one of the more reliable models that can be used.

Again the same variables were used for the creation of the Naive Bayes Classification model to predict the helpfulness of the Amazon reviews.

Naive Bayes Probability Outcomes of Testing Data:

helpfulnessBucket	prediction	helpful	probability
2.0	0.0	[3, 3]	[0.53831403022942...
2.0	2.0	[6, 6]	[0.01321563828031...
2.0	2.0	[2, 2]	[0.24647943551165...
2.0	0.0	[1, 1]	[0.55972568851594...
2.0	0.0	[5, 5]	[0.87351978054736...
2.0	0.0	[4, 4]	[0.74984803731730...
2.0	2.0	[1, 1]	[0.14960194809958...
2.0	2.0	[3, 3]	[1.14602031696084...
2.0	2.0	[1, 1]	[0.05248305457619...
2.0	0.0	[3, 3]	[0.89238945516624...
2.0	2.0	[11, 12]	[0.01331585569378...
2.0	2.0	[1, 1]	[0.12902485318988...
2.0	0.0	[3, 3]	[0.55185079336854...
2.0	1.0	[1, 1]	[0.01336240711234...
2.0	2.0	[6, 7]	[0.05576966213734...
2.0	2.0	[16, 16]	[0.01520406119350...
2.0	2.0	[1, 1]	[0.02052194994716...
2.0	2.0	[5, 6]	[1.78457319291109...
2.0	0.0	[1, 1]	[0.77662245812227...
2.0	1.0	[10, 10]	[0.28832349170972...

only showing top 20 rows

Naive Bayes Predictions:

helpfulnessBucket	prediction	count
2.0	2.0	15690
2.0	1.0	6478
2.0	0.0	13732
1.0	2.0	12134
1.0	1.0	8151
1.0	0.0	15897
0.0	2.0	10031
0.0	1.0	6128
0.0	0.0	20256

The Naive Bayes Classification model had an accuracy rate of 41.06% on the training data and 40.64% on the test data. The accuracy outcomes of this model were the lowest of the three models. As with the other models we are also seeing that there are more inaccurate predictions for each helpfulness bin than there are correct predictions.

The helpfulness bin 0.0 had the most accurate predictions throughout all models that were created. All three prediction models create a pattern in which show that helpfulness depends on other factors, not just wording of reviews. The reviews can be considered helpful even if the review is negative or neutral.

Amazon Review Analysis Findings and Conclusion:

After creating word frequencies, word clouds, Random Forest model, running a Logistic Regression, and Naive Bayes, it can be concluded that further exploration and refinements with the models and with the text analysis are likely need to incorporate specificity in terms of attributing n-grams. The n-grams are instrumental in determining and understanding how the text should be interpreted. The context of the reviews should be evaluated. As we have found when evaluating the visuals created with the word clouds, the words “highly recommended” revealed themselves in the good, neutral, and bad clouds. This indicates that the words “highly recommended” could be referred to in a positive and negative context. I.E. ‘i would highly recommend’ vs ‘i would highly recommend you avoid this’ would have the same ‘highly recommend’ bigram as we frequently observed in the word cloud. This concludes that the context in which reviews are written weigh heavily on the analysis of Amazon Reviews.

Can a prediction model generate the same classification of helpfulness as dozens or hundreds of prospective buyers?

During the analysis of the Amazon Review, it would be useful and important to weight the impact by the severity and the volume of people finding the review helpful. The current analysis indicated that a rating of 1 star could be due to receiving a product which was defective upon arrival. It could even mean that someone who experienced a defective battery that caused a device catch fire and cause serious damage to their home and property. However the helpfulness ratings would be weighted equally regardless of the total ratings of helpful/unhelpful (only the

ratio was considered). By doing further analysis, there is the possibility that weighting the words written in the reviews, would indicate a completely different outcome than the one that is currently produced.

For further analysis the incorporation of more text analysis and complex text analytics tools i.e. SyntaxNet could potentially determine more relevant terminology to the actual words that should be included in the helpfulness bins, and return better prediction models and accuracy rates.

Can the prediction model of reviews, either helpful or not be applied to other businesses and their products to compare how their product reviews are similar to those from Amazon? Can the predictive model trained on electronics reviews generalize well to products in other categories?

The prediction models of reviews could potentially be applied to other business industries to help them evaluate their products and services, and determine expectations of potential consumers. The predictive and classification models can be tuned and trained to evaluate the context of the reviews for better models. Considering the context of reviews, is crucial to understanding what needs to be trained and tuned. More frequent words that appear in reviews need to be removed as well as words that are unique. The middle of the frequencies of the words in the reviews, are the words that should ultimately be evaluated and used for the model creation and prediction. This would help to alleviate any word or group of words that could be used in good, neutral, and bad reviews.
Prediction models can also be trained to put the electronics into generalized categories, only if specific words are used. These models could potentially use the rare words within the reviews that are only reflected of certain electronics. However, it is hard to tell exactly what people will write and if they will use correct/consistent terminology especially when it comes to electronics.

Can a modestly accurate classification algorithm provide keywords that can guide or add value to the current human review process from existing Amazon shoppers?

The accuracy rates in which were produced from our models ranged from 40.06% to 45.00%. When dealing with real world data it is appropriate to conclude that these rating are not as bad as they appear. Real world data passed through prediction models output low accuracy rates. Having these rates on the Amazon reviews, is a good thing. With finer tuning they could output higher rates. These models have the potential to provide keywords that would indicate guidance of a consumer. The better these models are able to be tuned and reviewing the context in which the reviews were written by the users, we would be able to formulate sets of keywords that could be indicators of the current purchasing and reviewing process.

These words that could be formulated into certain lists, if in the wrong hands, could potentially sway people into a purchase based on false pretenses. This has the potential to be detrimental to the reputation of Amazon if the reviews were falsely written or biased towards any algorithmic approach.

As the Amazon Review Analysis stands, more research and parsing needs to take place to determine the actual context in which the reviews were written. This will would help to better

train the models as well as the removal of more frequent words. These tasks could lead to more developed findings as well as a more accurate prediction of helpfulness.

Why project is larger in scope than the homework:

For our project the data in which we are starting with is 1.37 GB and most of the data that we are using for the homework are much smaller. We have also conducted stratified sampling to begin and perform the analysis. The stated objective of the project requires building an analysis on top of building the scripting for the project. The homework assignments have been focused on scripting. We intend to focus on the analysis as the end goal for this project with scripting in python and pyspark as a means of accomplishing that analysis.

We are focusing our programs on knowledge, predication and classifications models, as well as the parameters within the programs to give the best accuracy.

Data Acquisition:
http://jmcauley.ucsd.edu/data/amazon/

Reference:
Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering
R. He, J. McAuley
WWW, 2016

Software and dependencies:
Apache Spark
Dockerized (Docker CE) pyspark Spark-2.4 on Kubernetes
https://hub.docker.com/r/alexmilowski/jupyter-spark-2.4.0/tags/

Rodeo
https://www.yhat.com/products/rodeo

Python & pyspark packages used throughout the Amazon Review Analysis
numpy (python)
pandas (python)
seaborn (python)
json (python)
calendar (python)
matplotlib (python)
collections (python)
gzip (python)
urllib (python)
datetime (python)
RegexTokenizer (pyspark)
Tokenizer (pyspark)
IDF aka Inverse Document Frequency (pyspark)
Hashing (pyspark)
StopWordsRemover (pyspark)
Pipeline (pyspark)
RandomForestClassifier (pyspark)

Code:
SET 1:

```
import urllib.request
from urllib import request
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from pandas.io.json import json_normalize
from collections import OrderedDict
import calendar
from datetime import date
import json
import gzip

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')

reviewDF = getDF('/Users/kwhood/Project/reviews_Electronics_5.json.gz')
display(reviewDF)
# returns 1,689,188 rows and 9 columns

# checking for NaN and missing values
pd.isna(reviewDF)
pd.isnull(reviewDF)
reviewDF.isnull().sum(axis = 0)
# returns that there are 24,730 missing reviewrNames, although this is not a big deal at the
moment, as the reviewer name might not be something that is analyzed
# the missing values in the reviewerName column totals 1.46% of the data and can be removed
reviewDF.isnull().sum(axis = 1)
# nothing shown to be missing in the rows at this time

# removing the 24,730 rows with the missing reviewerNames (only 1.46% of the total data and
should not have an impact on the rest of the data)
reviewDF2 = reviewDF.dropna()
```

```
display(reviewDF2)
#results in 1,664,458 rows and 9 columns
print(reviewDF2.columns)
print(reviewDF2.head(7))


frame = [reviewDF2.reviewText]
print(frame)

# word cloud
from wordcloud import WordCloud
import PIL
import itertools

complete_review = pd.concat(frame, keys=['electronics'])
review_names = ['electronics']
for reviews, name in zip(frame, review_names):
    raw_str = complete_review.loc[name].str.cat(sep=',')
    wordcloud = WordCloud(max_words=800, margin=0).generate(raw_str)
    plt.figure()
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    plt.show()


# cleaning data set
# Preprocessing of text in reviewText column making all words lower case
reviewDF2['reviewText'] = reviewDF2['reviewText'].apply(lambda x: " ".join(x.lower() for x in
x.split()))
reviewDF2['reviewText'].head(5)

# Removal of punctuation
reviewDF2['reviewText'] = reviewDF2['reviewText'].str.replace('[^\w\s]','')
reviewDF2['reviewText'].head(5)

# removing stop words
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
stop = stopwords.words('english')
reviewDF2['reviewText'] = reviewDF2['reviewText'].apply(lambda x: " ".join(x for x in x.split()
if x not in stop))
reviewDF2['reviewText'].head(4)

# removing common words
# looking to see what the top 50 common words are
```

```
common = pd.Series(''.join(reviewDF2['reviewText']).split()).value_counts()[:50]
common
# now removing these words because they will not be of use when classification occurs
reviewDF2['reviewText']=reviewDF2['reviewText'].apply(lambda x: " ".join(x for x in x.split() if
x not in common))
reviewDF2['reviewText'].head(4)

# removing rare words
rare = pd.Series(''.join(reviewDF2['reviewText']).split()).value_counts()[-50:]
rare
rare = list(rare.index)
reviewDF2['reviewText']=reviewDF2['reviewText'].apply(lambda x: " ".join(x for x in x.split() if
x not in rare))
reviewDF2['reviewText'].head(4)

# using textblob this will correct any misspelled words
from textblob import TextBlob
reviewDF2['reviewText'][:5].apply(lambda x: str(TextBlob(x).correct()))

# tokenization applied to make the blob that was just created and convert them into series of
words
TextBlob(reviewDF2['reviewText'][1]).words

# creating bigrams
TextBlob(reviewDF2['reviewText'][0]).ngrams(2)

# Creating the term frequency (number of times the term appears in each row/ number of terms
in the row)
termFreq = (reviewDF2['reviewText'][1:2]).apply(lambda x: pd.value_counts(x.split("
"))).sum(axis = 0).reset_index()
termFreq.columns = ['words','Term_Frequency']
termFreq

#using calendar to create a new column for year, month, day of week
print(reviewDF2['reviewTime'].dtype)
reviewDF2['reviewTime'] = pd.to_datetime(reviewDF2['reviewTime'])
print(reviewDF2['reviewTime'].dtype)

reviewDF2['year'] = reviewDF2['reviewTime'].dt.year
reviewDF2['month'] = (reviewDF2['reviewTime'].dt.month).apply(lambda x:
calendar.month_abbr[x])
reviewDF2['day'] = reviewDF2['reviewTime'].dt.weekday_name
reviewDF2['day'] = pd.Categorical(reviewDF2['day'],
categories=['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday'],
ordered=True)
reviewDF2.head(4)
```

```
# Reviews per day of week
current_palette = sns.color_palette('bright')
popDays = sns.countplot(x='day', data=reviewDF2),
plt.xticks(rotation='vertical'),plt.ylabel('Number of Reviews by Day'),
(sns.color_palette(current_palette))

# reviews per month
popMon = sns.countplot(x='month', data=reviewDF2),
plt.xticks(rotation='vertical'),plt.ylabel('Number of Reviews per Month')

# reviews per year
popYear = sns.countplot(x='year', data=reviewDF2),
plt.xticks(rotation='vertical'),plt.ylabel('Number of Reviews per Year')


# Sentiment Analysis
reviewDF2['reviewText'][:5].apply(lambda x: TextBlob(x).sentiment)
reviewDF2['sentiment'] = reviewDF2['reviewText'].apply(lambda x: TextBlob(x).sentiment[0])
reviewDF2[['reviewText', 'sentiment']].head(4)

# breaking the overall column into 3 different categories depending on review (good, neutral,
bad)
reviewDF2['good'] = np.where(reviewDF2['overall'] >= 4, 'yes','no')
reviewDF2['neutral'] = np.where(reviewDF2['overall'] == 3, 'yes','no')
reviewDF2['bad'] = np.where(reviewDF2['overall'] <= 2, 'yes','no')

reviewDF2.head(10)

# word clouds for each category of overall review rating
# Good Word Cloud
frame = [reviewDF2.reviewText, reviewDF2.good]
print(frame)
complete_review = pd.concat(frame, keys=['reviewText','good'])
review_names = ['reviewText','good']
for reviews, name in zip(frame, review_names):
    raw_str = complete_review.loc[name].str.cat(sep=',')
    wordcloud = WordCloud(max_words=800, margin=0).generate(raw_str)
    plt.figure()
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    plt.show()

# Neutral Word Cloud
frame2 = [reviewDF2.reviewText, reviewDF2.neutral]
print(frame2)
```

```
complete_review2 = pd.concat(frame2, keys=['reviewText','neutral'])
review_names2 = ['reviewText','neutral']
for reviews, name in zip(frame2, review_names2):
    raw_str2 = complete_review2.loc[name].str.cat(sep=',')
    wordcloud2 = WordCloud(max_words=800, margin=0).generate(raw_str2)
    plt.figure()
    plt.imshow(wordcloud2, interpolation='bilinear')
    plt.axis('off')
    plt.show()

# Bad Word Cloud
frame3 = [reviewDF2.reviewText, reviewDF2.bad]
print(frame3)
complete_review3 = pd.concat(frame3, keys=['reviewText','bad'])
review_names3 = ['reviewText','bad']
for reviews, name in zip(frame3, review_names3):
    raw_str3 = complete_review.loc[name].str.cat(sep=',')
    wordcloud3 = WordCloud(max_words=800, margin=0).generate(raw_str3)
    plt.figure()
    plt.imshow(wordcloud3, interpolation='bilinear')
    plt.axis('off')
    plt.show()
```

SET 2:

```
# coding: utf-8

# In[2]:
from pyspark.sql import SparkSession
spark = SparkSession.builder .getOrCreate()

# In[2]:
from pyspark import SparkConf, SparkContext
from pyspark.sql import *
from pyspark.sql.functions import *
from pyspark.ml.feature import Bucketizer
from pyspark.sql import Window
from pyspark.sql.functions import col

# In[3]:
sc = spark.sparkContext
# In[4]:
sc._conf.getAll()

# In[5]:
spark.version
```

```
# For our purposes there might not be any gain from reviews where there wasn't any rating of the reviews so omit for now

# In[6]:
df = spark.read.json("reviews_Electronics_5.json.gz")

# In[7]:
df_array = df.filter("helpful[1]>0").selectExpr("*","helpful[0] as positive","helpful[1] as totalReviews","helpful[0]/helpful[1] as helpfulnessRatio")

# In[8]:
df_array.head()

# In[30]:
df_array.show()

# helpfulnessBucket|min(helpfulnessRatio)|max(helpfulnessRatio)|
# +-----------------+---------------------+---------------------+
# |              0.0|                  0.0|  0.32894736842105265|
# |              1.0|   0.3333333333333333|   0.6694915254237288|
# |              2.0|   0.6702702702702703|                  1.0

# In[9]:
bucketizer =
Bucketizer(splits=[0,0.33,0.67,1],inputCol="helpfulnessRatio",outputCol="helpfulnessBucket")

# In[10]:
df_bucket = bucketizer.transform(df_array)

# In[11]:
df_bucket.select("positive","totalReviews","helpfulnessRatio","helpfulnessBucket").show()

# In[12]:
df_bucket.groupBy("helpfulnessBucket").agg(min("helpfulnessRatio"),max("helpfulnessRatio"))
.show()

# In[13]:
df_bucket.groupBy("helpfulnessBucket").count().show()
# calculated in excel because didn't want to have to implement window function to find count - min(count) and percentages

# In[14]:
df_bucket_sample =
df_bucket.sampleBy("helpfulnessBucket",fractions={0:1,1:0.903008661,2:0.222606274})

# In[15]:
```

```
df_bucket_sample    .groupBy("helpfulnessBucket").count() .show()

# In[16]:
df_bucket_sample_train =
df_bucket_sample.sampleBy("helpfulnessBucket",fractions={0:0.67,1:0.67,2:0.67})

# sampling and writing out to parquet to run once and have common result set. commented out as
to not run again accidentally

# df_bucket_sample_train.coalesce(1).write.mode("overwrite").format("parquet").save("train")

#
df_bucket_sample.subtract(df_bucket_sample_train).coalesce(1).write.mode("overwrite").format
("parquet").save("test")

# In[17]:
train = spark.read.parquet("train")
test  = spark.read.parquet("test")

# In[18]:
train.groupBy("helpfulnessBucket").count().sort(desc("helpfulnessBucket")).show()

# In[19]:
test.groupBy("helpfulnessBucket").count().sort(desc("helpfulnessBucket")).show()

# In[20]:
from pyspark.ml.feature import RegexTokenizer, Tokenizer, IDF, HashingTF,
StopWordsRemover

# In[21]:
regexTokenizer =
RegexTokenizer(inputCol="reviewText",outputCol="documentRegex",pattern="\\W")

# In[22]:
remover   =
StopWordsRemover(inputCol="documentRegex",outputCol="documentWordsFiltered")

# In[23]:
hashingTF =
HashingTF(inputCol="documentWordsFiltered",outputCol="wordsHashed",numFeatures=1000)

# In[24]:
idf =  IDF(inputCol="wordsHashed",outputCol="features")

# In[25]:
from pyspark.ml import Pipeline
```

```
from pyspark.ml.classification import RandomForestClassifier , LogisticRegression ,
NaiveBayes

# In[26]:
rf =
RandomForestClassifier(labelCol="helpfulnessBucket",featuresCol="features",numTrees=10)

# In[27]:
rf_pipeline =  Pipeline(stages=[regexTokenizer,remover,hashingTF,idf,rf])

# In[29]:
rf_model = rf_pipeline.fit(train)

# In[38]:
rf_model.transform(test).select("helpfulnessBucket","prediction","helpful","probability").sort(de
sc("helpfulnessBucket")).show()

# In[39]:
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# In[40]:
eval = MulticlassClassificationEvaluator(labelCol="helpfulnessBucket",
predictionCol="prediction", metricName="accuracy")

# In[41]:
eval.evaluate(rf_model.transform(test))

# In[42]:
eval.evaluate(rf_model.transform(train))

# In[43]:
rf_model.transform(test)    .groupBy("helpfulnessBucket","prediction")
.count().sort(desc("helpfulnessBucket"),desc("prediction")) .show(50)

# In[44]:
lr = LogisticRegression(maxIter=20,regParam=0.001,labelCol="helpfulnessBucket")
# In[45]:
lr_pipeline =  Pipeline(stages=[regexTokenizer,remover,hashingTF,idf,lr])

# In[46]:
lr_model = lr_pipeline.fit(train)

# In[47]:
lr_model.transform(test).select("helpfulnessBucket","prediction","helpful","probability").sort(de
sc("helpfulnessBucket")).show()
```

```
# In[48]:
eval.evaluate(lr_model.transform(test))

# In[49]:
eval.evaluate(lr_model.transform(train))

# In[50]:
lr_model.transform(test)    .groupBy("helpfulnessBucket","prediction")
.count().sort(desc("helpfulnessBucket"),desc("prediction")).show(50)

# In[51]:
nb = NaiveBayes(labelCol="helpfulnessBucket",featuresCol="features")

# In[52]:
nb_pipeline =  Pipeline(stages=[regexTokenizer,remover,hashingTF,idf,nb])

# In[53]:
nb_model = nb_pipeline.fit(train)

# In[54]:
nb_model.transform(test).select("helpfulnessBucket","prediction","helpful","probability").sort(d
esc("helpfulnessBucket")).show()

# In[55]:
eval.evaluate(nb_model.transform(test))

# In[56]:
eval.evaluate(nb_model.transform(train))

# In[57]:
nb_model.transform(test)    .groupBy("helpfulnessBucket","prediction")
.count().sort(desc("helpfulnessBucket"),desc("prediction")).show(50)
```