PRODUCT RECOMMENDER SYSTEM AND SENTIMENT ANALYSIS

Advising Professor:
Dr. Handan Liu

Submitted by:
Aman Kapoor (001448282)
Dimple Zatkia (001474071)
Harshad Jaju (001439898)
Pallavi Dalvi (001442156)

Table of Contents

1. ABSTRACT

Almost every online business relies on customer reviews and ratings. Explicit feedback is especially important in the ecommerce industry where all customer engagements are impacted by these ratings and reviews. Giant eCommerce like Amazon relies on such rating data to power its recommendation engine to provide the best product recommendations that are personalized and most relevant to the user.

We have implemented

- Product Recommendation using KNN algorithm
- Sentiment Analysis using KNN, Logistic Regression, and Naïve Bayes

2. INTRODUCTION

In today's world, every customer is faced with multiple choices. For example, if a person is looking for a book to read without any specific idea of what he wants, there's a wide range of possibilities how his search might pan out. He might waste a lot of time browsing around on the internet and trawling through various sites hoping to strike gold or he might look for recommendations from other people. The later always dominates the former.

This is what recommendation engines do and their power is being harnessed by most businesses these days. From Amazon to Netflix, Google to Goodreads, recommendation engines are one of the most widely used applications of machine learning techniques.

2.1 What are recommendation engines?

A recommendation engine filters the data using different algorithms and recommends the most relevant items to users. It first captures the past behavior of a customer and based on that, recommends products which the users might be likely to buy.

Until recently, people generally tend to buy products recommended to them by their friends or the people they trust. This used to be the primary method of purchase when there was any doubt about the product. But with the advent of the digital age, that circle has expanded to include online sites that utilize some sort of recommendation engine.

If a completely new user visits an e-commerce site, that site will not have any past history of that user. So how does the site go about recommending products to the user in such a scenario? One possible solution could be to recommend the best-selling products, i.e. the products which are

high in demand. Another possible solution could be to recommend the products which would bring the maximum profit to the business.

If we can recommend a few items to a customer based on their needs and interests, it will create a positive impact on the user experience and lead to frequent visits. Hence, businesses nowadays are building smart and intelligent recommendation engines by studying the past behavior of their users.

Recommendation engines are nothing but an automated form of a "shop counter guy". You ask him for the product. Not only he shows that product, but also the related ones which you could buy. They are well trained in cross selling and up selling. So, does our recommendation engines.

The ability of these engines to recommend personalized content, based on past behavior is incredible. It brings customer delight and gives them a reason to keep returning to the website.

Recommendation systems are one of the most common, easily comprehendible applications of big data and machine learning. Among the most known applications are Amazon's recommendation engine that provides us with a personalized webpage when we visit the site, and Spotify's recommendation list of songs when we listen using their app.

2.2 How does a recommendation engine work?

Before we deep dive into this topic, first we'll think of how we can recommend items to users:

- We can recommend items to a user which are most popular among all the users
- We can divide the users into multiple segments based on their preferences (user features) and recommend items to them based on the segment they belong to

Both above methods have their drawbacks. In the first case, the most popular items would be the same for each user so everybody will see the same recommendations. While in the second case, as the number of users increases, the number of features will also increase. So classifying the users into various segments will be a very difficult task.

The main problem here is that we are unable to tailor recommendations based on the specific interest of the users. It's like Amazon is recommending you buy a laptop just because it's been bought by most of the shoppers. But thankfully, Amazon (or any other big firm) does not recommend products using the above-mentioned approach. They use some personalized methods which help them in recommending products more accurately.

2.3 Type of Recommendation Engines

Before looking at the different types of recommendation engines, lets take a step back and see if we can make some intuitive recommendations. Consider the following cases:

Case 1: Recommend the most popular items

A simple approach could be to recommend the items which are liked by most number of users. This is a blazing fast and dirty approach and thus has a major drawback. The things is, there is no personalization involved with this approach.

Surprisingly, such approach still works in places like news portals. Whenever you login to say BBC news, you'll see a column of "Popular News" which is subdivided into sections and the most read articles of each sections are displayed. This approach can work in this case because:

- There is division by section so user can look at the section of his interest.
- At a time there are only a few hot topics and there is a high chance that a user wants to read the news which is being read by most others

Case 2: Using a classifier to make recommendation

We already know lots of classification algorithms. Let's see how we can use the same technique to make recommendations. Classifiers are parametric solutions so we just need to define some parameters (features) of the user and the item. The outcome can be 1 if the user likes it or 0 otherwise. This might work out in some cases because of following advantages:

- Incorporates personalization
- It can work even if the user's past history is short or not available

But has some major drawbacks as well because of which it is not used much in practice:

- The features might actually not be available or even if they are, they may not be sufficient to make a good classifier
- As the number of users and items grow, making a good classifier will become exponentially difficult

Case 3: Recommendation Algorithms

Now let's come to the special class of algorithms which are tailor-made for solving the recommendation problem. There are typically two types of algorithms – Content Based and Collaborative Filtering. You should refer to our previous article to get a complete sense of how they work. I'll give a short recap here.

One of the main algorithm as part of this case which we have implemented:

Collaborative filtering algorithms:

- Idea: If a person A likes item 1, 2, 3 and B like 2,3,4 then they have similar interests and A should like item 4 and B should like item 1.
- This algorithm is entirely based on the past behavior and not on the context. This makes it one of the most commonly used algorithm as it is not dependent on any additional information.
- For instance: product recommendations by e-commerce player like Amazon and merchant recommendations by banks like American Express.
- Further, there are several types of collaborative filtering algorithms :

User-User Collaborative filtering: Here we find look alike customers (based on similarity) and offer products which first customer's look alike has chosen in past. This algorithm is very effective but takes a lot of time and resources. It requires to compute every customer pair information which takes time. Therefore, for big base platforms, this algorithm is hard to implement without a very strong parallelizable system.

Item-Item Collaborative filtering: It is quite similar to previous algorithm, but instead of finding customer look alike, we try finding item look alike. Once we have item look alike matrix, we can easily recommend alike items to customer who have purchased any item from the store. This algorithm is far less resource consuming than user-user collaborative filtering. Hence, for a new customer the algorithm takes far lesser time than user-user collaborate as we don't need all similarity scores between customers. And with fixed number of products, product-product look alike matrix is fixed over time.

3. GOALS

In solving these problems, we will build collaborative filtering models for recommending products to customers using purchase data. In particular, we'll cover in details the step-by-step process in constructing a recommendation system with Python and machine learning module to create. These steps include:

- Transforming and normalizing data
- Training models
- Evaluating model performance
- Selecting the optimal model

We are trying to achieve two important goals through this project

- Recommending a product to a user based on product ratings
- Recommending a product to a user based on the reviews of the product

4. METHODOLOGY

4.1 Product Recommendation

Recommendation system - is nothing but analyzing and predicting the user response to the options. We will be using item-based K-nearest neighbor (KNN) algorithm. In order to determine the rating of products by any user, we will find other products that have similar ratings, and based these ratings we will predict his rating on selected product. KNN finds the nearest K neighbors of each product based on similarity function, and uses the weighted means to predict the rating

4.1.1 Data Analysis

We are using the cell phone and accessories dataset from below database.
http://jmcauley.ucsd.edu/data/amazon/

This dataset consists of reviews from Amazon. The data file is in JSON format. It has ~194,439 records and 9 columns.

Below is the screenshot of data.

In [2]:
```
#read cell phone and accessories dataset
mobile_df = pd.read_json('Cell_Phones_and_Accessories_5.json',lines=True)
```

In [3]:
```
print(mobile_df.shape)
mobile_df.head()
```

```
(194439, 9)
```

Out[3]:

	asin	helpful	overall	reviewText	reviewTime	reviewerID	reviewerName	summary	unixReviewTime
0	120401325X	[0, 0]	4	They look good and stick good! I just don't li...	05 21, 2014	A30TL5EWN6DFXT	christina	Looks Good	1400630400
1	120401325X	[0, 0]	5	These stickers work like the review says they ...	01 14, 2014	ASY55RVNIL0UD	emily l.	Really great product.	1389657600
2	120401325X	[0, 0]	5	These are awesome and make my phone look so st...	06 26, 2014	A2TMXE2AFO7ONB	Erica	LOVE LOVE LOVE	1403740800
3	120401325X	[4, 4]	4	Item arrived in great time and was in perfect ...	10 21, 2013	AWJ0WZQYMYFQ4	JM	Cute!	1382313600
4	120401325X	[2, 3]	5	awesome! stays on, and looks great. can be use...	02 3, 2013	ATX7CZYFXI1KW	patrice m rogoza	leopard home button sticker for iphone 4s	1359849600

Below table says what each column represents.

Column #	Column Name	Column Meaning
1	asin	Product ID
2	helpful	Helpfulness rating of the review e.g. 2/3
3	overall	Rating of the product
4	reviewtext	Text of the review
5	reviewTime	Time of the review (Raw)
6	reviwerID	User ID
7	reviewerName	User name
8	summary	Summary of the review
9	unixReviewTime	Time of the review (Unix Time)

Rating is ranged from 1 to 5 where 1 being the worst and 5 being the best rating.

4.1.2 Data pre-processing

To evaluate prediction and comparison we need to include some aggregation in our data. We have calculated count and mean of all the ratings by all users to a product. And then merged this with the original data frame.

```
#get count and mean of rating by product
count = mobile_df.groupby("asin", as_index=False).count()
mean = mobile_df.groupby("asin", as_index=False).mean()

#Merge count into dataset
mobileMerged_df = pd.merge(mobile_df, count, how='right', on=['asin'])
mobileMerged_df.head()
```

	asin	helpful_x	overall_x	reviewText_x	reviewTime_x	reviewerID_x	reviewerName_x	summary_x	unixReviewTime_x	helpful_y	overall_y	rev
0	120401325X	[0, 0]	4	They look good and stick good! I just don't li...	05 21, 2014	A30TL5EWN6DFXT	christina	Looks Good	1400630400	7	7	
1	120401325X	[0, 0]	5	These stickers work like the review says they ...	01 14, 2014	ASY55RVNIL0UD	emily l.	Really great product.	1389657600	7	7	
2	120401325X	[0, 0]	5	These are awesome and make my phone look so st...	06 26, 2014	A2TMXE2AFO7ONB	Erica	LOVE LOVE LOVE	1403740800	7	7	

For clear understanding we have renamed column names.

```
#rename column for understanding
mobileMerged_df.rename(columns={'helpful_x':'helpful', 'overall_x':'overallRating', 'reviewText_x':'reviewText',
                        'reviewTime_x':'reviewTime', 'reviewerID_x':'reviewerId', 'reviewerName_x':'reviewrName',
                        'summary_x':'summary', 'unixReviewTime_x':'unixReviewTime', 'helpful_y':'helpfulCount',
                        'overall_y':'overallRatingCount', 'reviewText_y':'reviewTextCount',
                        'reviewTime_y':'reviewTimeCount', 'reviewerID_y':'reviewerIdCount',
                        'reviewerName_y':'reviewerNameCount', 'summary_y':'summaryCount', 'unixReviewTime_y':'unixReviewTimeCo
                inplace=True)
```

For better and accurate results, we will keep those records which have count of total reviews equal to or more than 100.

```
#Sorting by no of reviews and get the products with more than or equal to 100 reviews
mobileMerged_df = mobileMerged_df.sort_values(by='reviewerIdCount', ascending=False)
reviews100Count = mobileMerged_df[mobileMerged_df.reviewerIdCount >= 100]
reviews100Count.head()
```

	asin	helpful	overallRating	reviewText	reviewTime	reviewerId	reviewrName	summary	unixReviewTime	helpfulCount	overallRatin
60401	B005SUHPO6	[0, 1]	5	This product its good to protect the fragile i...	07 13, 2013	AZ1NSCXQKRGUH	Sam	Really Good	1373673600	837	
60498	B005SUHPO6	[0, 0]	5	this a very good case and i recomend it to all...	02 3, 2013	A2N6XPEZYSGAU0	tractorman6070	case	1359849600	837	
60526	B005SUHPO6	[0, 0]	5	The color is great and as usual, an Otter	03 15, 2014	A30J79TFWG8XMY	Walawa05	Love it	1394841600	837	

For review analysis for a product, we have grouped all the reviews for a product into a data frame and a CSV file.

Grouping all the summary reviews by product

```
#group by summary data and save to csv
productReviewSummary = reviews100Count.groupby("asin")["summary"].apply(list)
productReviewSummary = pd.DataFrame(productReviewSummary)
productReviewSummary.to_csv("ProductReviewSummary.csv")
```

4.1.3 Data Preparation

We will only include those columns which will be required for our KNN model and exclude the remaining per performance point of view.

Dataframe with needed features

```
#include Product Id , Summary and Rating
df_new = pd.read_csv("ProductReviewSummary.csv")
df_new = pd.merge(df_new, mean, on="asin", how='inner')
df_new = df_new[['asin','summary','overall']]
df_new.head()
```

	asin	summary	overall
0	B0009B0IX4	['Solid Headset', 'Poor ergonomics mar this ot...	4.228395
1	B000S5Q9CA	['Best Micro-USB charger for Your Car. The Pr...	4.507962
2	B0013G8PTS	['Great Battery back-up', 'Does the job, has g...	4.605960
3	B0015RB39O	['I love these', 'Affordable', 'Would not reco...	3.628755
4	B001630QZE	['Newcomer to bluetooth but impressed', "Looke...	4.050000

Text Cleaning

We have a lot noise in our data like extra spaces, junk characters or typos. We need to clean it before processing. We have done this in 2 steps –

- Tokenizing

We have tokenized the text into words. For text formatting, we have used the Regex. RegEx (Regular Expression) is a sequence of characters that forms a search pattern and can be used to check if a string contains the specified search pattern. We have used Python built-in RE package. The sub() function replaces the matched string with the text of our choice. And then we have removed all the duplicates

Text Cleaning - Summary column

For text formatting, we will use the Regex. RegEx (Regular Expression) is a sequence of characters that forms a search pattern and can be used to check if a string contains the specified search pattern. For that we will use Python built-in RE package. The sub() function replaces the matched string with the text of our choice.

```
#tokenize summary
regEx = re.compile('[^a-z]+')
def cleanReviews(reviewText):
    reviewText = reviewText.lower()
    reviewText = regEx.sub(' ', reviewText).strip()
    return reviewText
```

```
#drop duplicates
df_new["cleanSummary"] = df_new["summary"].apply(cleanReviews)
df_new = df_new.drop_duplicates(['overall'], keep='last')
df_new = df_new.reset_index()
df_new.head()
```

	index	asin	summary	overall	cleanSummary
0	0	B0009B0IX4	['Solid Headset', 'Poor ergonomics mar this ot...	4.228395	solid headset poor ergonomics mar this otherwi...
1	1	B000S5Q9CA	['Best Micro-USB charger for Your Car. The Pr...	4.507962	best micro usb charger for your car the price ...
2	2	B0013G8PTS	['Great Battery back-up', 'Does the job, has g...	4.605960	great battery back up does the job has good fe...
3	3	B0015RB39O	['I love these', 'Affordable', 'Would not reco...	3.628755	i love these affordable would not recommend go...
4	4	B001630QZE	['Newcomer to bluetooth but impressed', "Looke...	4.050000	newcomer to bluetooth but impressed looked goo...

- CountVectorizing

The CountVectorizer provides a simple way to both tokenize a collection of text documents and build a vocabulary of known words, but also to encode new documents using that vocabulary. An encoded vector is returned with a length of the entire vocabulary and an integer count for the number of times each word appeared in the document. Stop words are words like “and”, “the”, “him”, which are presumed to be uninformative in representing the content of a text, and which may be removed to avoid them being construed as signal for prediction. Sometimes, however, similar words are useful for prediction, such as in classifying writing style or personality.

We have also used stop_words parameter in vectorization. It is part of **Natural Language Toolkit (NLTK)**. It is used to remove the not required stop words as per English language, which do not have any significance or meaning for prediction like ‘and’, ‘it’, ‘has’ etc.

CountVectorizer: ¶

The CountVectorizer provides a simple way to both tokenize a collection of text documents and build a vocabulary of known words, but also to encode new documents using that vocabulary. An encoded vector is returned with a length of the entire vocabulary and an integer count for the number of times each word appeared in the document. Stop words are words like “and”, “the”, “him”, which are presumed to be uninformative in representing the content of a text, and which may be removed to avoid them being construed as signal for prediction. Sometimes, however, similar words are useful for prediction, such as in classifying writing style or personality.

```
#apply CountVectorizer on cleaned data
cleanSummary = df_new["cleanSummary"]
countVector = CountVectorizer(max_features = 300, stop_words='english')
transformedReviews = countVector.fit_transform(cleanSummary)

vect_Reviews_df = DataFrame(transformedReviews.A, columns=countVector.get_feature_names())
vect_Reviews_df = vect_Reviews_df.astype(int)
```

4.1.4 Data distribution to train, test and validation data set

We have divided data into training data and test data with every time different data size to analyze the accuracy of the model. We have used train-test ratio as 75-25 and 85-15.

```
#create train and test dataset
X = np.array(vect_Reviews_df)
tpercent = 0.75
tsize = int(np.floor(tpercent * len(vect_Reviews_df)))
reviews_train_df = X[:tsize]
reviews_test_df = X[tsize:]

train_len = len(reviews_train_df)
test_len = len(reviews_test_df)
```

4.1.5 Model Implementation

We have chosen KNN (K-nearest neighbor) for our model as for recommendation as this is based on collaborative filtering method, which will be useful in our case.
After applying the algorithm, we have set our model to recommend 2 similar products based on mean rating value.

KNN Implementation

After applying algorithm we will give 2 related products based on our trained model

```
#Apply the KNN Algorithm
neighbor = NearestNeighbors(n_neighbors=3).fit(reviews_train_df)

#nearest k neighbors of each datapoint in object X
distances, indices = neighbor.kneighbors(reviews_train_df)
```

```
#recommend 2 related products based on mean rating
for i in range(test_len):
    h = neighbor.kneighbors([reviews_test_df[i]])
    related_product_list = h[1]

    first_related_product = [item[0] for item in related_product_list]
    first_related_product = str(first_related_product).strip('[]')
    first_related_product = int(first_related_product)
    second_related_product = [item[1] for item in related_product_list]
    second_related_product = str(second_related_product).strip('[]')
    second_related_product = int(second_related_product)

    print ("Based on product reviews, for ", df_new["asin"][train_len + i] ," average rating is ",df_new["overall"][train_len + i
    print ("The first similar product is ", df_new["asin"][first_related_product] ," average rating is ",df_new["overall"][first_
    print ("The second similar product is ", df_new["asin"][second_related_product] ," average rating is ",df_new["overall"][sec
    print ("********************************************************************************")
```

```
Based on product reviews, for  B00A9LVA0Y  average rating is  4.508196721311475
The first similar product is  B005QXX6N4  average rating is  4.467889908256881
The second similar product is  B004JQUZC4  average rating is  4.358333333333333
********************************************************************************
Based on product reviews, for  B00ABCV34O  average rating is  3.6037735849056602
The first similar product is  B008GVL9YQ  average rating is  4.123893805309734
The second similar product is  B004T0GHOU  average rating is  3.6422018348623855
********************************************************************************
```

Strategy

Since we are using KNN, the value of K, which means number of nearest neighbors to the data

point on the plane, is very important. Hence, we have tried our model with different data size and different values of K to know which is giving the best results.

First, we have tried data size for train-test data as 75-25 with the value of K=3, then we tried same data size with the value of K=5 and then we tested with 85-15 train-test size with K=5. The result metrics is mentioned in the next section.

4.1.6 Metric evaluation

As mentioned above, we have used multiple data set size and value of K. Below are their evaluation metric

Evaluation metric with data size 75-25 and K=3

	precision	recall	f1-score	support
3	0.56	0.62	0.59	8
4	0.93	0.91	0.92	44
micro avg	0.87	0.87	0.87	52
macro avg	0.74	0.77	0.75	52
weighted avg	0.87	0.87	0.87	52

Accuracy and Mean square error of the model

Accuracy of the model

```
print (accuracy_score(test_target_df, knnpreds_test))
```

```
0.8653846153846154
```

```
print(mean_squared_error(test_target_df, knnpreds_test))
```

```
0.1346153846153846
```

Evaluation metric with data size 75-25 and K=5

	precision	recall	f1-score	support
3	0.71	0.62	0.67	8
4	0.93	0.95	0.94	44
micro avg	0.90	0.90	0.90	52
macro avg	0.82	0.79	0.81	52
weighted avg	0.90	0.90	0.90	52

Accuracy and Mean square error of the model

```
print (accuracy_score(test_target_df, knnpreds_test))
```

```
0.9038461538461539
```

```
print(mean_squared_error(test_target_df, knnpreds_test))
```

```
0.09615384615384616
```

Evaluation metric with data size 85-15 and K=5

	precision	recall	f1-score	support
3	0.50	0.33	0.40	3
4	0.93	0.97	0.95	29
micro avg	0.91	0.91	0.91	32
macro avg	0.72	0.65	0.67	32
weighted avg	0.89	0.91	0.90	32

Accuracy and Mean square error of the model

```
print (accuracy_score(test_target_df, knnpreds_test))
```

```
0.90625
```

```
print(mean_squared_error(test_target_df, knnpreds_test))
```

```
0.09375
```

4.2 Sentiment Analysis

Sentiment Analysis - We have done sentiment analysis of the users using two algorithms. The two algorithms that we have used are logistic regression and Naive Bayes. We have compared the results of the two to check which algorithm gives us a better result. Then, We have also analyzed the user's behavior based on the number of ratings the user has given and what is the mean of his ratings.

Sentiment analysis is contextual mining of text which identifies and extracts subjective information in source material and helping a business to understand the social sentiment of their brand, product or service while monitoring online conversations.

Sentiment Column indicates the Numeric Score of Being Positive or Negative. Useful Score Column indicates the Boolean Value of the Total Number of Votes.

4.2.1 Data Analysis

- Loading of data

```
mobile_df = pd.read_json('Cell_Phones_and_Accessories_5.json',lines=True)
```

```
mobile_df
```

There are in total 194439 rows × 9 columns

- Cleaning the data and removing duplicates:

We will clean the data by dropping the duplicate entries based on reviewerID asin and unixReviewTime. Then we will add the helpfulness and upvote percentages. If the helpfulness denominator < 0, we will assign it helpful % of -1, else will take helpfulnessNumerator / helpfulnessDenominator and assign it a range in % upvote.

```
In [10]: #Cleaning the data by eliminating duplicates
reviews.drop_duplicates(subset=['reviewerID', 'asin','unixReviewTime'],inplace=True)

#Adding the helpfulness and upvote percentages for metrics
reviews['Helpful %'] = np.where(reviews['HelpfulnessDenominator'] > 0, reviews['HelpfulnessNumerator'] / reviews['Helpf
reviews['% Upvote'] = pd.cut(reviews['Helpful %'], bins = [-1, 0, 0.2, 0.4, 0.6, 0.8, 1.0], labels = ['Empty', '0-20%',
reviews['Id'] = reviews.index;
reviews
```

	asin	helpful	overall	reviewText	reviewTime	reviewerID	reviewerName	summary	unixReviewTime	HelpfulnessNumerator	Helpfu
0	7806397051	[3, 4]	1	Very oily and creamy. Not at all what I expect...	01 30, 2014	A1YJEY40YUW4SE	Andrea	Don't waste your money	1391040000	3	
1	7806397051	[1, 1]	3	This palette was a decent price and I was look...	04 18, 2014	A60XNB876KYML	Jessica H.	OK Palette!	1397779200	1	
2	7806397051	[0, 1]	4	The texture of this concealer pallet is fantas...	09 6, 2013	A3G6XNM240RMWA	Karen	great quality	1378425600	0	
3	7806397051	[2, 2]	2	I really can't tell what exactly this thing is...	12 8, 2013	A1PQFP6SAJ6D80	Norah	Do not work on my face	1386460800	2	
4	7806397051	[0, 0]	3	It was a little smaller than I expected, but t...	10 19, 2013	A38FVHZTNQ271F	Nova Amor	It's okay.	1382140800	0	
				I was very happy							

Getting the sentiments based on overall review score

```
In [36]: reviews["sentiment"] = reviews["overall"].apply(lambda score: "positive" if score > 3 else "negative")
reviews.head(5)
```

wText	reviewTime	reviewerID	reviewerName	summary	unixReviewTime	HelpfulnessNumerator	HelpfulnessDenominator	Helpful %	% Upvote	Id	summaryClean	sentiment
ry oily and eamy. t at all what I pect...	01 30, 2014	A1YJEY40YUW4SE	Andrea	Don't waste your money	1391040000	3	4	0.75	60-80%	0	don t waste your money	negative
This alette was a decent and I look...	04 18, 2014	A60XNB876KYML	Jessica H.	OK Palette!	1397779200	1	1	1.00	80-100%	1	ok palette	negative
exture of this cealer allet is ntas...	09 6, 2013	A3G6XNM240RMWA	Karen	great quality	1378425600	0	1	0.00	Empty	2	great quality	positive

Based on the heatmap, we can see that there are users that have given overall = 3, this is a neutral review. Therefore, we will now remove the neutral reviews and assign value to 1 to reviews with overall < 3 as 0 and overall > 3 as 1. This will help us in performing regression.

```
In [12]: df =  reviews[reviews['overall'] != 3]
         X = df['reviewText']
         y_dict = {1:0, 2:0, 4:1, 5:1}
         y = df['overall'].map(y_dict)
```

4.2.2 Data pre-processing

To achieve the models, the Text had to be pre-processed using Natural Language Toolkit (NLTK) so that it can be fed to the Machine Learning algorithm. The Text has a lot of stop-words and duplicates. These unnecessary portions were removed using the NLTK library which is a platform used for Python programs that work with human language data for applying in Statistical Natural Language Processing (NLP). It contains Text processing libraries for tokenization, parsing, classification, stemming, tagging and semantic reasoning. We processed the Text by converting it into the lower case to avoid errors using Regex.

```
regEx = re.compile('[^a-z]+')
def cleanReviews(reviewText):
    reviewText = reviewText.lower()
    reviewText = regEx.sub(' ', reviewText).strip()
    return reviewText

review["summaryClean"] = review["summary"].apply(cleanReviews)

train, test = train_test_split(review, test_size=0.2)
print("%d items in training data, %d in test data" % (len(train), len(test)))
```

```
692000 items in training data, 173000 in test data
```

```
stopwords = set(STOPWORDS)
mpl.rcParams['font.size']=12
mpl.rcParams['savefig.dpi']=100
mpl.rcParams['figure.subplot.bottom']=.1


def show_wordcloud(data, title = None):
    wordcloud = WordCloud(
        background_color='white',
        stopwords=stopwords,
        max_words=200,
        max_font_size=40,
        scale=3,
        random_state=1
    ).generate(str(data))

    fig = plt.figure(1, figsize=(8, 8))
    plt.axis('off')
    if title:
        fig.suptitle(title, fontsize=20)
        fig.subplots_adjust(top=2.3)

    plt.imshow(wordcloud)
    plt.show()

show_wordcloud(review["summaryClean"],title = "All Reviews")
```

The New Data Frame with One Useful Attribute Along With its Product ID and the total reviews are displayed for user convenience

```
assist = mobile_df[['helpful','asin']]
```

```
assist
```

	helpful	asin
0	[0, 0]	120401325X
1	[0, 0]	120401325X
2	[0, 0]	120401325X
3	[4, 4]	120401325X
4	[2, 3]	120401325X
5	[1, 2]	120401325X
6	[0, 0]	120401325X
7	[1, 2]	3998899561
8	[2, 3]	3998899561

```
df = pd.DataFrame(assist)
```

```
df[['helpful','total']] = pd.DataFrame(df.helpful.values.tolist(), index=df.index)
dfhelpful = df[['asin', 'helpful','total']]
```

```
dfhelpful
```

4.2.3 Merging of data

Once, the Text is processed and duplicates are removed we merged the necessary derived columns with the rest of data set to achieve the result set comprising of [asin, overall, reviewText, reviewTime, reviewerID, reviewerName, summary, unixReviewTime, helpful, Total]

```
dfNew = mobile_df.drop('helpful',1)
```

```
frames = [dfNew, dfhelpful]
```

```
mobile_df = dfNew.join(dfhelpful[['helpful','total']])
```

```
mobile_df
```

	asin	overall	reviewText	reviewTime	reviewerID	reviewerName	summary	unixReviewTime	helpful	total
0	120401325X	4	They look good and stick good! I just don't li...	05 21, 2014	A30TL5EWN6DFXT	christina	Looks Good	1400630400	0	0
1	120401325X	5	These stickers work like the review says they ...	01 14, 2014	ASY55RVNIL0UD	emily l.	Really great product.	1389657600	0	0
2	120401325X	5	These are awesome and make my phone look so st...	06 26, 2014	A2TMXE2AFO7ONB	Erica	LOVE LOVE LOVE	1403740800	0	0
3	120401325X	4	Item arrived in great time and was in perfect ...	10 21, 2013	AWJ0WZQYMYFQ4	JM	Cute!	1382313600	4	4
4	120401325X	5	awesome! stays on, and looks great. can be use...	02 3, 2013	ATX7CZYFXI1KW	patrice m rogoza	leopard home button sticker for iphone 4s	1359849600	2	3

4.2.4 Creating Train, Test Data

```
train, test = train_test_split(review, test_size=0.2)
print("%d items in training data, %d in test data" % (len(train), len(test)))
```

```
692000 items in training data, 173000 in test data
```

We split the data into Train and Test data for feature Extraction from reviewsText. We Assign New Dimension to Each Word and Give the Word Counts.

4.2.5 Building Machine Learning Models

- **Logistic Regression**

Logistic Regression is used when the dependent variable(target) is categorical.

For example,

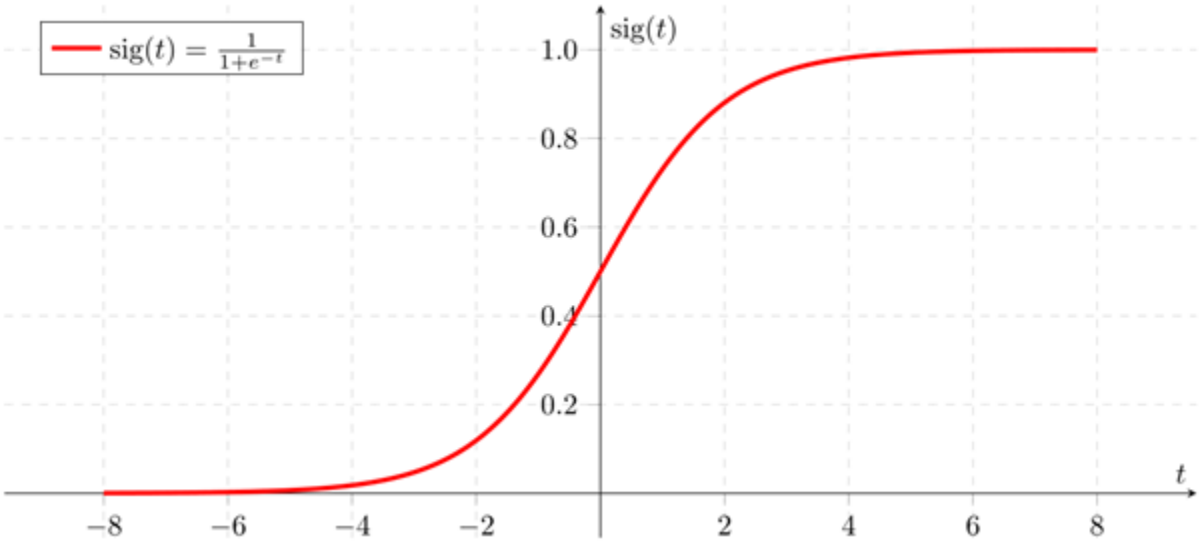
To predict whether an email is spam (1) or (0)

Whether the tumor is malignant (1) or not (0)

Consider a scenario where we need to classify whether an email is spam or not. If we use linear regression for this problem, there is a need for setting up a threshold based on which classification can be done. Say if the actual class is malignant, predicted continuous value 0.4

and the threshold value is 0.5, the data point will be classified as not malignant which can lead to serious consequence in real time.

From this example, it can be inferred that linear regression is not suitable for classification problem. Linear regression is unbounded, and this brings logistic regression into picture. Their value strictly ranges from 0 to 1.



If ‘Z’ goes to infinity, Y(predicted) will become 1 and if ‘Z’ goes to negative infinity, Y(predicted) will become 0

We will now perform logistic regression. The CountVectorizer provides a simple way to both tokenize a collection of text documents and build a vocabulary of known words, but also to encode new documents using that vocabulary. We will create test train model and sort the words based. Stop words are words like “and”, “the”, “him”, which are presumed to be uninformative in representing the content of a text, and which may be removed to avoid them being construed as signal for prediction. Sometimes, however, similar words are useful for prediction, such as in classifying writing style or personality.

Getting the top 20 negatives and positive words from reviews

```
In [13]: c = CountVectorizer(stop_words = 'english')

def text_fit(X, y, model,clf_model,coef_show=1):

    X_c = model.fit_transform(X)
    print('# features: {}'.format(X_c.shape[1]))
    X_train, X_test, y_train, y_test = train_test_split(X_c, y, random_state=0)
    print('# train records: {}'.format(X_train.shape[0]))
    print('# test records: {}'.format(X_test.shape[0]))
    clf = clf_model.fit(X_train, y_train)
    acc = clf.score(X_test, y_test)
    print ('Model Accuracy: {}'.format(acc))

    if coef_show == 1:
        w = model.get_feature_names()
        coef = clf.coef_.tolist()[0]
        coeff_df = pd.DataFrame({'Word' : w, 'Coefficient' : coef})
        coeff_df = coeff_df.sort_values(['Coefficient', 'Word'], ascending=[0, 1])
        print('')
        print('-Top 20 positive-')
        print(coeff_df.head(20).to_string(index=False))
        print('')
        print('-Top 20 negative-')
        print(coeff_df.tail(20).to_string(index=False))


text_fit(X, y, c, LogisticRegression())
```

```
# features: 72694
# train records: 132190
# test records: 44064
C:\Users\JAYESH\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning: Default solver will
be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
Model Accuracy: 0.9249500726216412

-Top 20 positive-
Word  Coefficient
 pleasantly     2.384222
     hooked     2.257317
     staple     2.200133
  skeptical     2.153173
 complaints     2.011397
    pleased     1.894847
  subscribe     1.871199
 noticeably     1.864546
complaining     1.842625
    amazing     1.820935
    awesome     1.805127
      evens     1.785057
```

TF-IDF:

Tf-idf stands for term frequency-inverse document frequency, and the tf-idf weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Variations of the tf-idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query.

One of the simplest ranking functions is computed by summing the tf-idf for each query term; many more sophisticated ranking functions are variants of this simple model.

Tf-idf can be successfully used for stop-words filtering in various subject fields including text summarization and classification.

```
In [15]: tfidf = TfidfVectorizer(stop_words = 'english')
         text_fit(X, y, tfidf, LogisticRegression())

         # features: 72694
         # train records: 132190
         # test records: 44064
         Model Accuracy: 0.9281726579520697

         -Top 20 positive-
         Word  Coefficient
               love    12.217327
              great     9.826824
            perfect     8.124969
            amazing     7.989484
               best     6.973668
             highly     6.916465
              works     6.566374
            awesome     6.181700
               nice     5.998487
               easy     5.993309
              helps     5.984361
            pleased     5.908600
          excellent     5.626778
              happy     5.440191
               glad     5.375078
               soft     5.336826
         definitely     4.874709
          beautiful     4.687433
          wonderful     4.570399
             smooth     4.397937

         -Top 20 negative-
         Word  Coefficient
                reviews    -4.637585
                  hopes    -4.661567
                  broke    -4.811503
                  threw    -4.969979
               returned    -4.974605
               terrible    -5.148619
                useless    -5.245654
                    did    -5.251954
                  worse    -5.310728
               horrible    -5.383293
                  awful    -5.389344
                 return    -5.429208
                   didn    -5.484571
                  maybe    -5.840430
         disappointment    -6.163732
          unfortunately    -6.493056
```

N-grams:

N-grams are contiguous sequences of n-items in a sentence. N can be 1, 2 or any other positive integers, although usually we do not consider very large N because those n-grams rarely appears in many different places.

When performing machine learning tasks related to natural language processing, we usually need to generate n-grams from input sentences. For example, in text classification tasks, in addition to using each individual token found in the corpus, we may want to add bi-grams or tri-grams as features to represent our documents. This post describes several different ways to generate n-grams quickly from input sentences in Python.

```
In [16]: tfidf_n = TfidfVectorizer(ngram_range=(1,2),stop_words = 'english')
         text_fit(X, y, tfidf_n, LogisticRegression())
```

```
# features: 2177780
# train records: 132190
# test records: 44064
Model Accuracy: 0.9222267610748003

-Top 20 positive-
Word  Coefficient
             love    17.207290
            great    14.233636
          perfect    10.064959
            works     9.137805
          amazing     9.008506
             best     8.857616
             nice     8.613009
             easy     7.549170
             soft     7.205425
            happy     6.837663
            helps     6.653958
          awesome     6.578585
          pleased     6.210442
       definitely     6.142294
            price     5.930533
           highly     5.907272
        excellent     5.786386
highly recommend     5.758696
           smooth     5.670389
        beautiful     5.659967

-Top 20 negative-
Word  Coefficient
          hoping    -5.517294
disappointment    -5.532951
         thought    -5.665248
    waste money    -5.973677
         reviews    -6.284454
 disappointing    -6.363196
           broke    -6.443598
        terrible    -6.503862
           worse    -6.628761
           awful    -6.677277
          return    -6.888687
        horrible    -7.186969
           money    -7.187139
           worst    -7.635661
 unfortunately    -7.844501
             did    -7.913779
```

It can be observed that accuracy of Logistic regression has not improved with n-grams. So, using n-grams did not help us much.

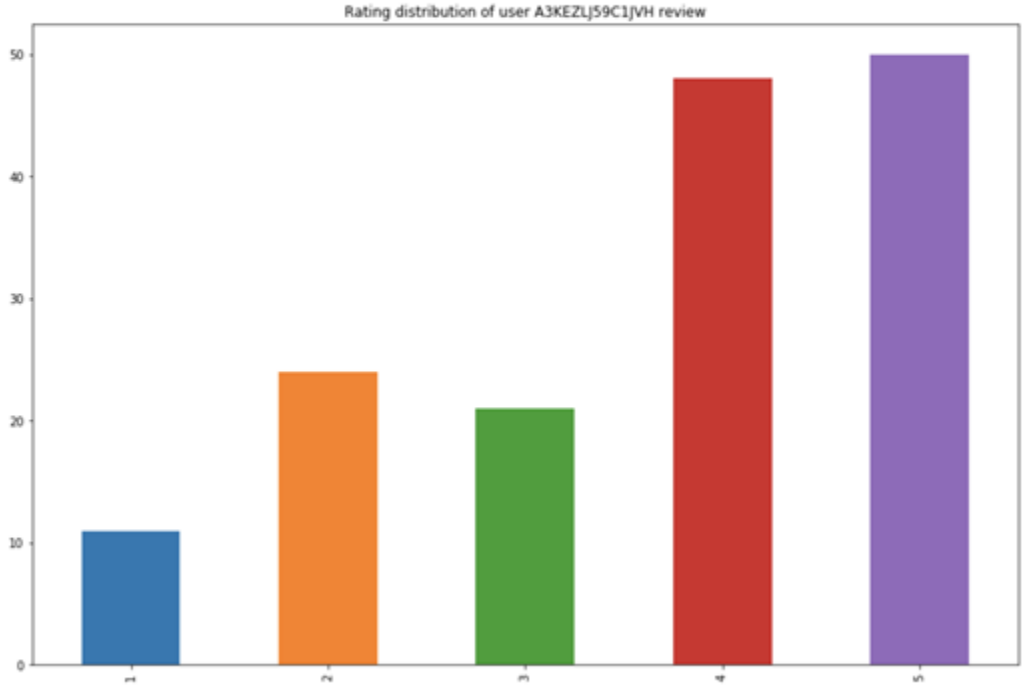
Study of user behavior

The user behavior has to be analyzed to improve the model performance and understand the underlying reasons for the bad or good reviews. This also gives importance to word choices of a user when giving the reviews. As, there are some user who explicitly rate product bad or good if it vice versa.

Count of ratings is considered to select the user and user with the id A3KEZLJ59C1JVH is randomly chosen

```
]: def plot_user(reviewerID):
       df_luser = reviews[reviews['reviewerID'] == reviewerID]['overall']
       df_luser_plot = df_luser.value_counts(sort=False)
       ax = df_luser_plot.plot(kind = 'bar', figsize = (15,10), title = 'Rating distribution of user {} review'.format(re
       plt.show()

   plot_user('A3KEZLJ59C1JVH')
```



It can be observed that the user is generous with his ratings and probably happy with his products as most of his ratings are 4 and 5. This gives us idea about user behavioral aspetcs.

Most popular words used by the user for different ratings are observed. (2-grams and 3-grams are chosen for analysis) Here, we will analyzed the most pouplar words used by th user. An N-gram is simply a sequence of N words. For instance, let us take a look at the following examples.

San Francisco (is a 2-gram)

The Three Musketeers (is a 3-gram)

She stood up slowly (is a 4-gram)

N-gram model predicts the occurrence of a word based on the occurrence of its N – 1 previous words. So here we are answering the question – how far back in the history of a sequence of words should we go to predict the next word? For instance, a bigram model (N = 2) predicts the occurrence of a word given only its previous word (as N – 1 = 1 in this case). Similarly, a trigram model (N = 3) predicts the occurrence of a word based on its previous two words (as N – 1 = 2 in this case).

```
]: def get_token_ngram(score, benchmark, userid='all'):
       '''This functions returns the top used words by the user  we have applied nltk word_tokenize to split the
          sentence or text  into the words and then we will take that in total text if that word are not in stop word and
          len is >=3 and apply bigrams and trigrams.
           '''

       if userid != 'all':
           df = reviews[(reviews['reviewerID'] == userid) & (reviews['overall'] == score)]['reviewText']
       else:
           df = reviews[reviews['overall'] == score]['reviewText']

       count = len(df)
       total_text = ' '.join(df)
       total_text = total_text.lower()
       stop = set(stopwords.words('english'))
       total_text = nltk.word_tokenize(total_text)
       total_text = [word for word in total_text if word not in stop and len(word) >= 3]
       lemmatizer = WordNetLemmatizer()
       total_text = [lemmatizer.lemmatize(w,'v') for w in total_text]
       bigrams = ngrams(total_text,2)
       trigrams = ngrams(total_text, 3)

       # look at 2-gram and 3-gram together
       combine = chain(bigrams, trigrams)
       text = nltk.Text(combine)
       fdist = nltk.FreqDist(text)

       # return only phrase occurs more than benchmark of his reviews
       return sorted([(w,fdist[w],str(round(fdist[w]/count*100,2))+'%') for w in set(text) if fdist[w] >= count*benchmark]
```

- **Naïve Bayes:**

A Naive Bayes Classifier is a supervised machine-learning algorithm that uses the Bayes' Theorem, which assumes that features are statistically independent. The theorem relies on the *naive* assumption that input variables are independent of each other, i.e. there is no way to know anything about other variables when given an additional variable. Regardless of this assumption, it has proven itself to be a classifier with good results.

- A Naive Bayes Classifier is a supervised machine-learning algorithm that uses the Bayes' Theorem, which assumes that features are statistically independent.
- The theorem relies on the *naive* assumption that input variables are independent of each other, i.e. there is no way to know anything about other variables when given an additional variable. Regardless of this assumption, it has proven itself to be a classifier with good results.

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

- "P" is the symbol to denote probability.
- P (A | B) = Probability of event A (hypothesis) occurring given that B (evidence) has occurred.
- P (B | A) = Probability of the event B (evidence) occurring given that A (hypothesis) has occurred.
- P (A) = Probability of event B (hypothesis) occurring.
- P (B) = Probability of event A (evidence) occurring.

- **Naive Bayes Classifier**

- Given a vector **x** of features, Naive Bayes calculates the probability that the vector belongs to each class.

    - P (C_k | x_1, x_2…..x_n)
- Bayes' Theorem, we know that it is equal to: P (C_k | **x**) = (P(**x** | C_k) *P (C_k)) / P(**x**)
- We know P (C_k) because of the class distribution in our data.
- P(**x** | C_k) is equivalent to its joint probability P (C_k, x_1, x_2….., x_n). By the chain rule in probabilities we can expand it to:

$$= P(C_k, x_1, x_2\ldots., x_n) = P(x_1 \mid x_2 \ldots, x_n, C_k) * P(x_2\ldots, x_n, C_k)$$
$$= P(x_1 \mid x_2 \ldots, x_n, C_k) * P(x_2, \mid x_{3\ldots..,} x_n, C_k) * P(x_{3,} \ldots, x_n, C_k)$$
$$= P(x_1 \mid x_2\ldots, x_n, C_k) * P(x_2, \mid x_3 \ldots, x_n, C_k) * P(x_{n-1,} \mid x_n, C_k) * P(x_n \mid C_k) * P(C_k)$$

```
review["sentiment"] = review["overall"].apply(lambda score: "positive" if score > 3 else "negative")
review["usefulScore"] = (review["helpful"]/review["total"]).apply(lambda n: "useful" if n > 0.6 else "useless")

review.head(15)
```

	overall	summary	helpful	total	sentiment	usefulScore
0	4	Looks Good	0	0	positive	useless
1	5	Really great product.	0	0	positive	useless
2	5	LOVE LOVE LOVE	0	0	positive	useless
3	4	Cute!	4	4	positive	useful
4	5	leopard home button sticker for iphone 4s	2	3	positive	useful
5	5	best thing ever..	0	0	positive	useless
6	1	not a good Idea	1	2	negative	useless
7	5	Solid Case	2	3	positive	useful
8	5	Perfect Case	1	1	positive	useful
9	5	Just what I needed	0	0	positive	useless
10	5	A Winner	2	3	positive	useful

- **Multinomial Naive Bayes Classifier**

MultinomialNB implements the naive Bayes algorithm for multinomially distributed data, and is one of the two classic naive Bayes variants used in text classification (where the data are typically represented as word vector counts, although tf-idf vectors are also known to work well in practice). The distribution is parametrized by vectors $\theta y=(\theta y1,\ldots,\theta yn)$ class y, where n is the number of features (in text classification, the size of the vocabulary) and θyi is the probability $P(xi|y)$ of feature i appearing in a sample belonging to class y.

The parameters θy is estimated by a smoothed version of maximum likelihood, i.e. relative frequency counting:

```
model = MultinomialNB().fit(X_train_tfidf, y_train)
prediction['Multinomial'] = model.predict(X_test_tfidf)
```

Visualize the accuracy, recall and f1-score for Naive Bayes Multinomial

```
print(metrics.classification_report(y_test, prediction['Multinomial'], target_names = ["positive", "negative"]))
```

```
              precision    recall  f1-score   support

    positive       0.95      0.81      0.88     24323
    negative       0.97      0.99      0.98    148677

 avg / total       0.97      0.97      0.97    173000
```

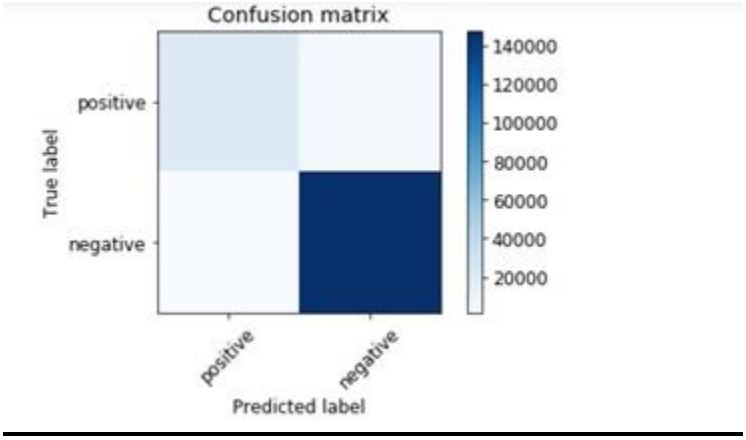
```
accuracy_score(y_test, prediction['Multinomial'])
```

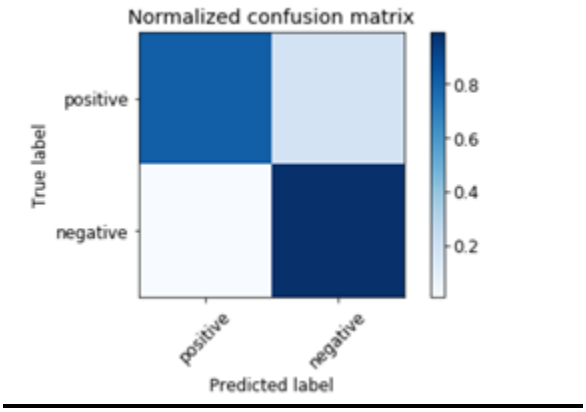
```
0.9674161849710983
```

```
def plot_confusion_matrix(matrix, title='Confusion matrix', cmap=plt.cm.Blues, labels=["positive", "negative"]):
    plt.imshow(matrix, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(labels))
    plt.xticks(tick_marks, labels, rotation=45)
    plt.yticks(tick_marks, labels)
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

# Compute confusion matrix
matrix = confusion_matrix(y_test, prediction['Multinomial'])
np.set_printoptions(precision=2)
plt.figure()
plot_confusion_matrix(matrix)

matrix_normalized = matrix.astype('float') / matrix.sum(axis=1)[:, np.newaxis]
plt.figure()
plot_confusion_matrix(matrix_normalized, title='Normalized confusion matrix')
plt.show()
```





- **Binomial Naive Bayes Classifier**

BernoulliNB implements the naive Bayes training and classification algorithms for data that is distributed according to multivariate Bernoulli distributions; i.e., there may be multiple features but each one is assumed to be a binary-valued (Bernoulli, boolean) variable. Therefore, this class requires samples to be represented as binary-valued feature vectors; if handed any other kind of data, a BernoulliNB instance may binarize its input (depending on the binarize parameter).

The decision rule for Bernoulli naive Bayes is based on

$P(x_i|y)=P(i|y)x_i+(1-P(i|y))(1-x_i)$

which differs from multinomial NB's rule in that it explicitly penalizes the non-occurrence of a feature i that is an indicator for class y, where the multinomial variant would simply ignore a non-occurring feature.

In the case of text classification, word occurrence vectors (rather than word count vectors) may be used to train and use this classifier. BernoulliNB might perform better on some datasets, especially those with shorter documents. It is advisable to evaluate both models, if time permits.

```
model = BernoulliNB().fit(X_train_tfidf, y_train)
prediction['Bernoulli'] = model.predict(X_test_tfidf)
```

Visualize the accuracy, recall and f1-score for Naive Bayes Bernoulli

```
print(metrics.classification_report(y_test, prediction['Bernoulli'], target_names = ["positive", "negative"]))
```

```
             precision    recall  f1-score   support

   positive       0.79      0.68      0.73     24323
   negative       0.95      0.97      0.96    148677

avg / total       0.93      0.93      0.93    173000
```

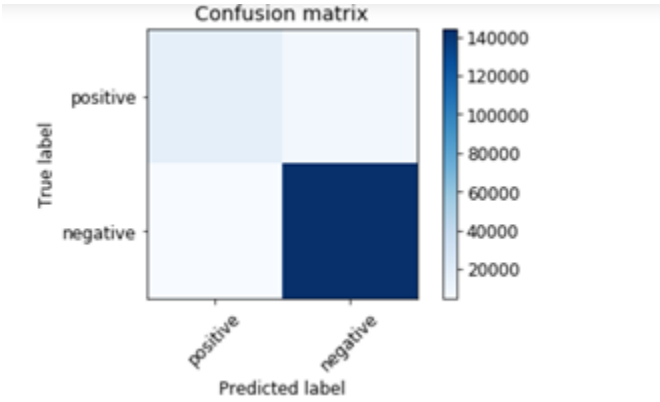
```
accuracy_score(y_test, prediction['Bernoulli'])
```

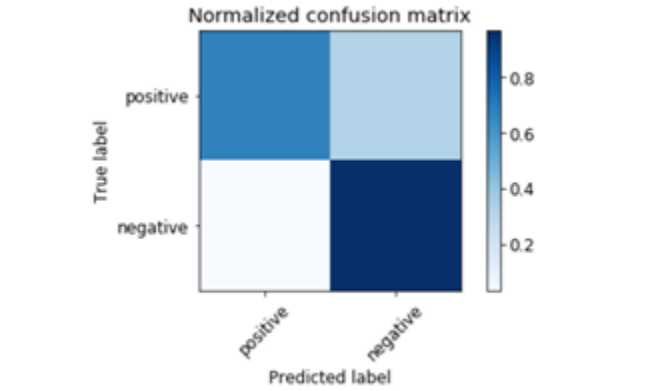
```
0.9288959537572254
```

```
def plot_confusion_matrix(matrix, title='Confusion matrix', cmap=plt.cm.Blues, labels=["positive", "negative"]):
    plt.imshow(matrix, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(labels))
    plt.xticks(tick_marks, labels, rotation=45)
    plt.yticks(tick_marks, labels)
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

# Compute confusion matrix
matrix = confusion_matrix(y_test, prediction['Bernoulli'])
np.set_printoptions(precision=2)
plt.figure()
plot_confusion_matrix(matrix)

matrix_normalized = matrix.astype('float') / matrix.sum(axis=1)[:, np.newaxis]
plt.figure()
plot_confusion_matrix(matrix_normalized, title='Normalized confusion matrix')
plt.show()
```





Testing the Sentiments of Few Reviews

```
def testSentiments(model, testData):
    testCounts = countVector.transform([testData])
    testTfidf = tfidf_transformer.transform(testCounts)
    result = model.predict(testTfidf)[0]
    probability = model.predict_proba(testTfidf)[0]
    print("Sample estimated as %s: negative prob %f, positive prob %f" % (result.upper(), probability[0], probability[1]))

testSentiments(logreg, "They look good and stick good! I just don't like the rounded shape because I was always bumping it and Sir
testSentiments(logreg, "This is the cutest case EVER!  I love it because it was easy to apply.  I had to switch from the rubber on
testSentiments(logreg, "Use it for my Nokia N95-3 and it works very well.A bit bulky since its not really pocketable but fits in a
```

```
Sample estimated as POSITIVE: negative prob 0.145477, positive prob 0.854523
Sample estimated as POSITIVE: negative prob 0.000008, positive prob 0.999992
Sample estimated as POSITIVE: negative prob 0.000104, positive prob 0.999896
```

- **K-Nearest Neighbours**

In pattern recognition, the K-Nearest Neighbors algorithm (KNN) is a nonparametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether KNN is used for classification or regression.

In KNN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor. • In KNN regression, the output is the property value for the object. This value is the average of the values of k nearest neighbors. KNN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The KNN algorithm is among the simplest of all machine learning algorithms. A peculiarity of the KNN algorithm is that it is sensitive to the local structure of the data.

$$y = \frac{1}{K}\sum_{i=1}^{K} y_i$$

Formula for KNN

Here yi is the ith case of the examples sample and y is the prediction (outcome) of the query point. The KNN algorithm was used because although it is not very interpretable it was believed to produce good results for the problem. The KNN model is using the alpha values as n_neighbors which are used to predict the point based on the number of the nearest Personalized Medicine: Redefining Cancer Treatment College of Engineering, INFO-6105 22 neighbors. The best alpha/ n_neighbors value for KNN Model is 5 which produces the train, cross-validation, and test log-loss values as 0.48, 1, 1.07 respectively. When the KNN Model was tested on the testing data using the best alpha the log-loss for the model was generated as 1.008 and

misclassified points were 31.95% which means that the model predicts 68.05% points correctly. The confusion, precision and recall matrix support these results.

Sentiment Analysis using KNN algorithm

Cleaning the data by using regex function.

```
#Cleaning Data using regex

regEx = re.compile('[^a-z]+')
def cleanReviews(reviewText):
   reviewText = reviewText.lower()
   reviewText = regEx.sub(' ', reviewText).strip()
   return reviewText
mobile_df["summaryClean"] = mobile_df["summary"].apply(cleanReviews)
mobile_df["cleanreviewText"] = mobile_df["reviewText"].apply(cleanReviews)
```

```
#Adding helpful and upvote %ages for metrics
mobile_df['Helpful %'] = np.where(mobile_df['HelpfulnessDen'] > 0, mobile_df['HelpfulnessNum'] / mobile_df['HelpfulnessDen'], -1
mobile_df['% Upvote'] = pd.cut(mobile_df['Helpful %'], bins = [-1, 0, 0.2, 0.4, 0.6, 0.8, 1.0], labels = ['Empty', '0-20%', '20-
mobile_df['Id'] = mobile_df.index;
mobile_df
```

Creating the heat map with overall ratings and reviews for every ID.

```
#Creating heat map of overall ratings vs reviews
df_m = mobile_df.groupby(['overall', '% Upvote']).agg({'Id': 'count'})
df_m = df_m.unstack()
df_m.columns = df_m.columns.get_level_values(1)

fig = plt.figure(figsize=(15,10))
sns.heatmap(df_m[df_m.columns[::-1]].T, cmap = 'YlGnBu', linewidths=.5, annot = True, fmt = 'd', cbar_kws={'label': '# reviews_d
plt.yticks(rotation=0)
plt.title('How helpful users find among the user scores')
plt.show()
```


```
df =  mobile_df[mobile_df['overall'] != 3]
X = df['cleanreviewText']
y_dict = {1:0, 2:0, 4:1, 5:1}
y = df['overall'].map(y_dict)
```

```
model = CountVectorizer(stop_words = 'english')
model.fit_transform(X)
```

```
<173000x70327 sparse matrix of type '<class 'numpy.int64'>'
        with 5554703 stored elements in Compressed Sparse Row format>
```

KNN model with test and train records

```
mean_acc = np.zeros((15))
std_acc = np.zeros((15))

for i in range(1,16):

    knn = KNeighborsClassifier(n_neighbors = i)
    X_c = model.fit_transform(X)
    print('# features: {}'.format(X.shape[0]))
    X_train, X_test, y_train, y_test = train_test_split(X_c, y, random_state=0)
    print('# train records: {}'.format(X_train.shape[0]))
    print('# test records: {}'.format(X_test.shape[0]))
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    print(f1_score(y_test, y_pred))
    knn_model_1 = knn.fit(X_train, y_train)
    print(i)
    mean_acc[i-1] = metrics.accuracy_score(y_test, y_pred)
    std_acc[i-1]=np.std(y_pred==y_test)/np.sqrt(y_pred.shape[0])

    print('k-NN accuracy for test set: %f' % knn_model_1.score(X_test, y_test))
    print('==============================================================')
mean_acc
```

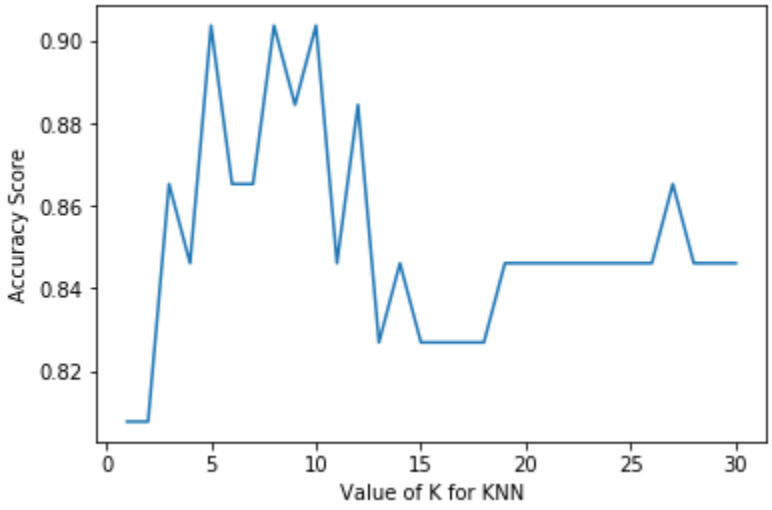
```
array([0.83708671, 0.804     , 0.86330636, 0.85181503, 0.86712139,
       0.8643237 , 0.86991908, 0.86910983, 0.87045087, 0.8700578 ,
       0.8699422 , 0.8699422 , 0.86906358, 0.86945665, 0.86855491])
```

5. RESULT ANALYSIS

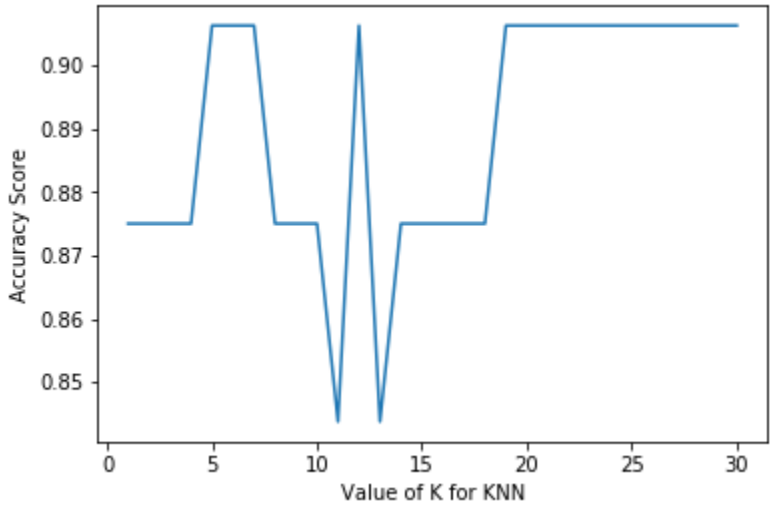
- **Product Recommendation**

We have plotted graph to analyze the accuracy score against value of K and data set size. We have kept the range of K as 1 to 31.

- **Data size 75-25**



- **Data size 85-15**



- **Sentiment Analysis**

  - **Logistic Regression**

Logistic Regression

Model Accuracy: 0.7825662672476398

Logistic Regression with TFIDF –

Model Accuracy: 0.9281726579520697

Logistic Regression with TFIDF + ngrams –

Model Accuracy: 0.9222267610748003

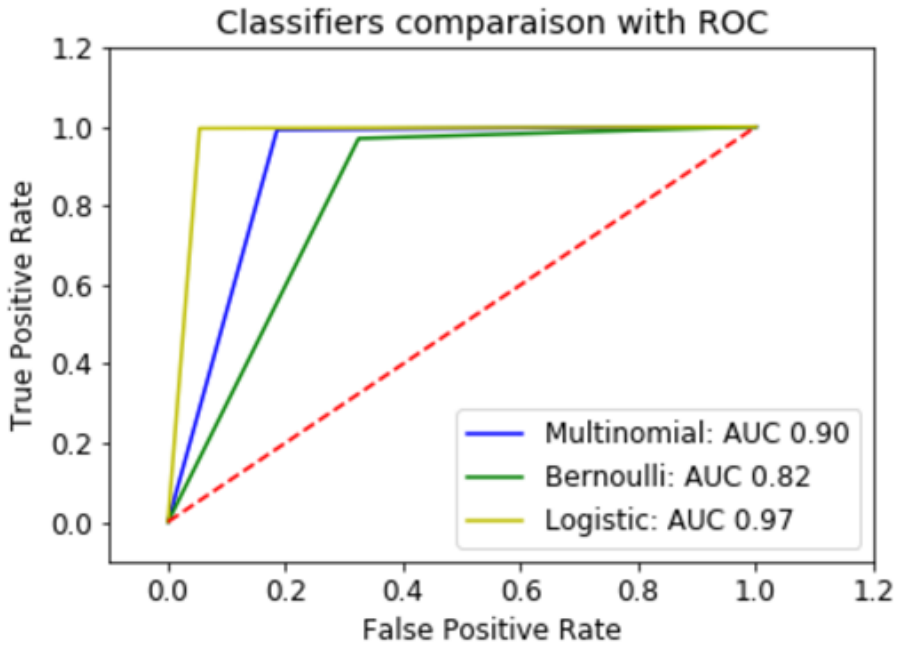
  - **Naïve Bayes**

Graphical comparison of Multinomial, Bernoulli and Logistic Regression

```
def formatt(x):
    if x == 'negative':
        return 0
    return 1
vfunc = np.vectorize(formatt)

cmp = 0
colors = ['b', 'g', 'y', 'm', 'k']
for model, predicted in prediction.items():
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test.map(formatt), vfunc(predicted))
    roc_auc = auc(false_positive_rate, true_positive_rate)
    plt.plot(false_positive_rate, true_positive_rate, colors[cmp], label='%s: AUC %0.2f'% (model,roc_auc))
    cmp += 1

plt.title('Classifiers comparaison with ROC')
plt.legend(loc='lower right')
plt.plot([0,1],[0,1],'r--')
plt.xlim([-0.1,1.2])
plt.ylim([-0.1,1.2])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



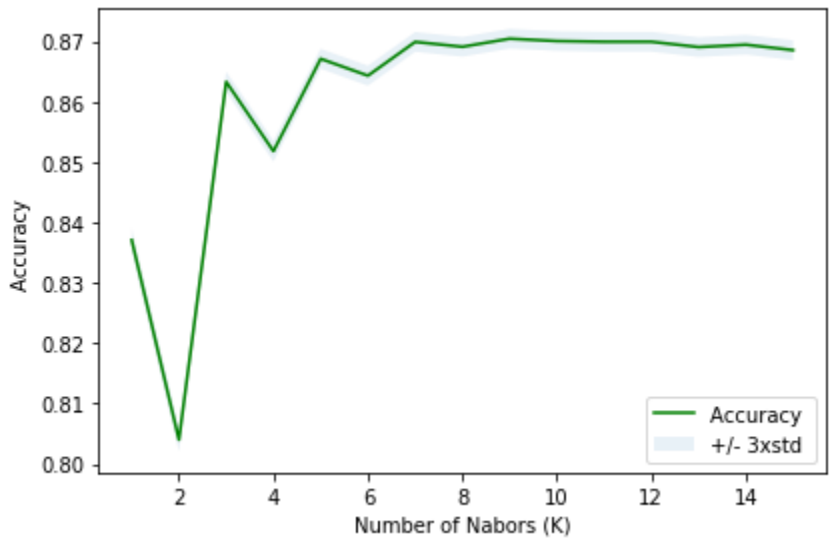
- **KNN**

K	Accuracy Score	
1	0.837087	
2	0.804	Least Accuracy
3	0.863306	
4	0.851815	
5	0.867121	
6	0.864324	
7	0.869919	
8	0.86911	
9	0.870451	Greatest Accuracy
10	0.870058	
11	0.869942	
12	0.869942	
13	0.869064	
14	0.869457	
15	0.868555	

K	F1 Score	
1	0.907316303	
2	0.883849664	Least F1 Score
3	0.924284726	
4	0.916280224	
5	0.926932221	
6	0.924736424	
7	0.928869446	
8	0.928043929	
9	0.929407466	Greatest F1 Score
10	0.928974041	
11	0.92934571	
12	0.929174903	
13	0.929023525	
14	0.929107757	
15	0.928867256	

Plot of different values of K vs accuracy score:

```
plt.plot(range(1,16),mean_acc,'g')
plt.fill_between(range(1,16),mean_acc - 1 * std_acc,mean_acc + 1 * std_acc, alpha=0.10)
plt.legend(('Accuracy ', '+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Nabors (K)')
plt.tight_layout()
plt.show()
```



6. CONCLUSION

- **Product Recommendation**

We used the Item based collaborative approach with KNN model for our product recommendation system. The reason why we preferred item based approach over user based approach is because user based approach is often harder to evaluate because of the dynamic behavior of users, whereas items usually don't change very often, and it can be computed offline and served without rigorous training of the model.

We predicted the rating score of the product by taking number of neighbors = 3 and test train split of 75-25. The accuracy of the model was 0.8653846153846154 and the MSE was 0.1346153846153846.

Then we predicted the rating score of the product by taking number of neighbors = 5 and test train split of 75-25. The accuracy of the model was 0.9038461538461539 and the MSE was 0.09615384615384616.

Further, we predicted the rating score of the product by taking n_neighbors = 5 and test train split of 85-15. The accuracy of the model was 0.90625 and the MSE was 0.09375.

So, our analysis says that changing the test train split size did not make much difference for model evaluation but changing the value of K (number of neighbors) did affect the model very much. Based on this dataset and the way KNN is implemented here, the best value for K would be 5.

- **Sentiment Analysis**

We have used three algorithms: Logistic Regression, Naïve Bayes, and KNN

For KNN, we have tested the accuracy score with the values of K from 1 to 15 and we have concluded that K = 9 has the highest accuracy score. Also, the lowest accuracy is of K=2.

From the plot analysis, we have derived that for the value of k<9, the accuracy score almost remains the same.

For Logistic Regression, apart from basic, we have also implemented tfidf with Logistic Regression and N-gram + tfidf with Logistic Regression and we have concluded that the highest accuracy is for tfidf with Logistic Regression and lowest with basic Logistic Regression.

For Naïve Bayes, we have implemented BernoulliNB, MultinomialNB, and Logistic and we have concluded that Logistic have the highest accuracy score.

From all the respective algorithms, we are able to conclude that Naïve Bayes with Logistic have the highest accuracy score and will be used for Sentiment Analysis.

Algorithm	Accuracy Score	
Logistic Regression with TFIDF	0.928172658	
Naïve Bayes (Logistic)	0.97	Greatest Accuracy
Knn (k = 9)	0.870451	

7. REFERENCES

- https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-recommendation-engine-python/
- https://www.analyticsvidhya.com/blog/2016/06/quick-guide-build-recommendation-engine-python/
- https://medium.com/datadriveninvestor/how-to-build-a-recommendation-system-for-purchase-data-step-by-step-d6d7a78800b6
- https://www.lynda.com/Python-tutorials/Introduction-Python-Recommendation-Systems-Machine-Learning/563080-2.html?srchtrk=index%3a2%0alinktypeid%3a2%0aq%3arecommendation+system+machine+learning%0apage%3a1%0as%3arelevance%0asa%3atrue%0aproducttypeid%3a2
- https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc
- http://snap.stanford.edu/data/web Amazo n.html
- https://www.kaggle.com/linkma/starter amazon reviews for sentiment b9ebb399 9 h ttps://www.kaggle.com/vijayabhaskar96/fully convolutional accuracy 94 1 15 mi 789a46
- https://www.datacamp.com/community/tutorials/wordcloud python
- http://www.albertauyeung.com/post/generating ngrams python/ https ://codereview.stackexchange.com/questions/124245/splitting text into n grams and analyzingstatistics on them
- https://www.kaggle.com/rafjaa/resampling strategies for imbalanced datasets https://stackoverflow.com/questions/13423919/computing n grams using python
- https://www.analyticsvidhya.com/blog/2018/03/introduction k neighbours algorithm clustering/