Amazon Fine Food Reviews Analysis

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

Number of reviews: 568,454 Number of users: 256,059 Number of products: 74,258 Timespan: Oct 1999 - Oct 2012 Number of Attributes/Columns in data: 10

Attribute Information:

1)Id

2)ProductId - unique identifier for the product

3)UserId - unqiue identifier for the user

4)ProfileName

5)HelpfulnessNumerator - number of users who found the review helpful

6)HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not

7)Score - rating between 1 and 5

8)Time - timestamp for the review

9)Summary - brief summary of the review

10)Text - text of the review

Objective

Given a review, determine whether the review is positive (Rating of 4 or 5) or negative (rating of 1 or 2).

[Q] How to determine if a review is positive or negative?

[Ans] We could use the Score/Rating. A rating of 4 or 5 could be cosnidered a positive review. A review of 1 or 2 could be considered negative. A review of 3 is nuetral and ignored. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

Loading the data

The dataset is available in two forms

.csv file

SQLite Database

In order to load the data, We have used the SQLITE dataset as it easier to query the data and visualise the data efficiently.

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score id above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

Reading Data

```
con = sqlite3.connect("../input/amazon-fine-food-reviews/database.sqlite")
```

Exploratory Data Analysis

Data Cleaning¶

```
display = pd.read_sql_query("""SELECT * FROM REVIEWS WHERE SCORE != 3 AND User
Id ="AR5J8UI46CURR" ORDER BY ProductId""",con)
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATI VANILLA WAFERS
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATI VANILLA WAFERS
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATI VANILLA WAFERS
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATI VANILLA WAFERS

As can be seen above the same user has multiple reviews of the with the same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that

ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8)

ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delelte the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

```
#Deduplication of entries
final = sorted_data.drop_duplicates(subset=('UserId','ProfileName','Time','Tex
t'),keep='first',inplace=False)
```

Text Preprocessing: stemming, stop-word removal and Lemmatization

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1)Begin by removing the html tags

2)Remove any punctuations or limited set of special characters like , or . or # etc.

3)Check if the word is made up of english letters and is not alpha-numeric

4)Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)

5)Convert the word to lowercase

6)Remove Stopwords

7)Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

After which we collect the words used to describe positive and negative reviews

Bag of Words (BOW)

```
#Bag of Words
count_vect = CountVectorizer()
final_counts = count_vect.fit_transform(final['Text'].values)
```

To get Most common positive and negative words

```
freq_dist_positive = nltk.FreqDist(all_positive_words)
freq_dist_negative = nltk.FreqDist(all_negative_words)
print("Most Common Positive words : ",freq_dist_positive.most_common(20))
print("Most Common Negative Words : ",freq_dist_negative.most_common(20))
```

```
Most Common Positive words :  [(b'like', 139429), (b'tast', 129047), (b'good', 112766), (b'flavor', 1096
24), (b'love', 107357), (b'use', 103888), (b'great', 103870), (b'one', 96726), (b'product', 91033), (b't
ri', 86791), (b'tea', 83888), (b'coffe', 78814), (b'make', 75107), (b'get', 72125), (b'food', 64802),
(b'would', 55568), (b'time', 55264), (b'buy', 54198), (b'realli', 52715), (b'eat', 52004)]
Most Common Negative Words :  [(b'tast', 34585), (b'like', 32330), (b'product', 28218), (b'one', 20569),
(b'flavor', 19575), (b'would', 17972), (b'tri', 17753), (b'use', 15302), (b'good', 15041), (b'coffe', 14
716), (b'get', 13786), (b'buy', 13752), (b'order', 12871), (b'food', 12754), (b'dont', 11877), (b'tea',
11665), (b'even', 11085), (b'box', 10844), (b'amazon', 10073), (b'make', 9840)]
```

Observation:- From the above it can be seen that the most common positive and the negative wordsoverlap for eg:**'like'** could be used as **'not like'**

so, its a goog idea to consider pairs of consequent words(bi-grams) or a q sequence of n consecutive words(**n_grams**)

Bi-Grams and N-Grams

Now that we have our list of words describing positive and negative reviews lets analyse them.

```
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features
=5000)
final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_counts.get_s
hape())
print("the number of unique words including both unigrams and bigrams "
, final_bigram_counts.get_shape()[1])
```

To get Most common positive and negative words

```
freq_dist_positive = nltk.FreqDist(all_positive_words)
freq_dist_negative = nltk.FreqDist(all_negative_words)
print("Most Common Positive words : ",freq_dist_positive.most_common(20))
print("Most Common Negative Words : ",freq_dist_negative.most_common(20))
```

```
Most Common Positive words :  [(b'like', 139429), (b'tast', 129047), (b'good', 112766), (b'flavor', 1096
24), (b'love', 107357), (b'use', 103888), (b'great', 103870), (b'one', 96726), (b'product', 91033), (b't
ri', 86791), (b'tea', 83888), (b'coffe', 78814), (b'make', 75107), (b'get', 72125), (b'food', 64802),
(b'would', 55568), (b'time', 55264), (b'buy', 54198), (b'realli', 52715), (b'eat', 52004)]
Most Common Negative Words :  [(b'tast', 34585), (b'like', 32330), (b'product', 28218), (b'one', 20569),
(b'flavor', 19575), (b'would', 17972), (b'tri', 17753), (b'use', 15302), (b'good', 15041), (b'coffe', 14
716), (b'get', 13786), (b'buy', 13752), (b'order', 12871), (b'food', 12754), (b'dont', 11877), (b'tea',
11665), (b'even', 11085), (b'box', 10844), (b'amazon', 10073), (b'make', 9840)]
```

Observation:- From the above it can be seen that the most common positive and the negative wordsoverlap for eg:**'like'** could be used as **'not like'**

so, its a goog idea to consider pairs of consequent words(bi-grams) or a q sequence of n consecutive words(**n_grams**)

TF-IDF (Term frequency–Inverse document frequency)

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
final_tf_idf = tf_idf_vect.fit_transform(final['Text'].values)
```

```
final_tf_idf.shape
```

```
(364171, 2910192)
```