Amazon Fine Food Reviews - Kaggle

Introduction:

The objective of this project is to “Predict the Rating” of food products with the help of customer’s reviews.

Dataset:

The Amazon Fine Food Reviews dataset consists of 568,454 food reviews Amazon users left up to October 2012.

The features of this dataset are as follows:

* Product Id - unique identifier for the product
* User Id - unique identifier for the user
* Profile Name
* Helpfulness Numerator - number of users who found the review helpful
* Helpfulness Denominator - number of users who indicated whether they found the review helpful
* Score - rating between 1 and 5
* Time - timestamp for the review
* Summary - brief summary of the review
* Text - text of the review

Since the data which is provided is very large, in order to reduce our training time, only 10% of the data was taken for this project, i.e. 56846 customer reviews.

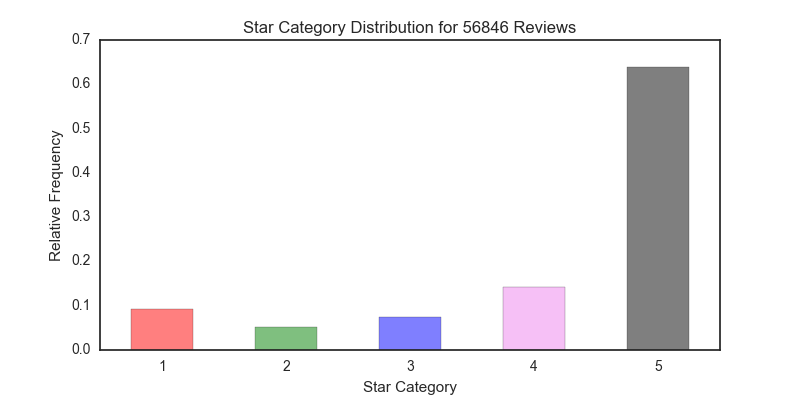
NOTE: the distribution of data for this 10% is similar for every “Score Category” as compared to the whole data.

The features that will be used for this project are:

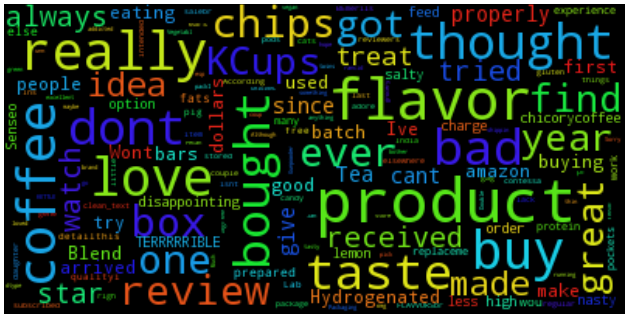
1. Text - customer’s reviews
2. Score - rating between 1 and 5

Exploratory Data Analysis:

After analyzing the score or rating given by customers, it was discovered that majority of the ratings belong to Star Category 4 and 5, about 80%.



Then, WordClouds were constructed to analyze words i.e. how customers described the products for every Rating Category.

Rating = 1 Rating = 2

Rating = 3 Rating = 4

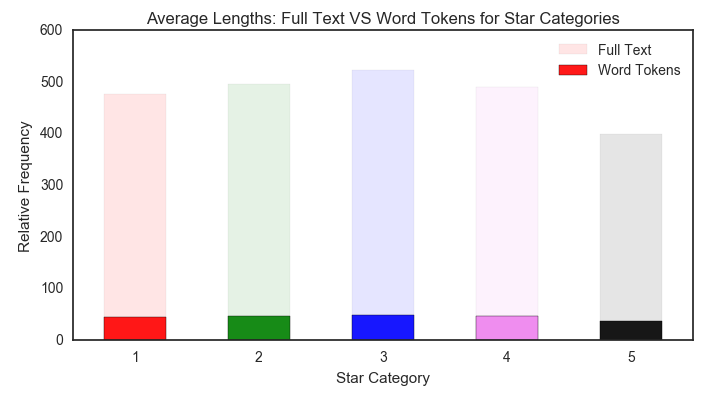


Rating = 5

Reviews with 1 and 2 ratings contains a lot of negative words, while rating 3 contains a mix of negative as well as positive words.

In contrast, 4 and 5 ratings mostly contains positive words, as expected.

As a part of EDA, factor analysis was done on the review text, i.e. finding the average lengths of tokens for each Rating Category with respect to the average lengths of full text review. It was observed that after preprocessing the reviews by stemming, removing stopwords and removing punctuation by Regular expression operations, the average lengths of tokens are almost same for all Rating Categories.



Data Preparation:

Original Dataset contains 568,454 food reviews Amazon users left up to October 2012. And only 10% data (i.e. 56846 reviews) were considered with same data distribution as in original dataset for this project to reduce the training time.

Splitting the new data into training and testing set:

Training data : consists of 70% data, i.e. 39,792 reviews

Testing data : consists of 30% data, i.e. 17,054 reviews

Steps for data (text review) preparation:

1. Removed punctuation with the help of Regular Expression Operations.
2. Converted to lower case and splitted every word.
3. Lemmatization
4. Removed stopwords
5. Joined back

Workflow:

C:\Users\DHRUBAJIT\Desktop\Kaggle Datasets\Amazon cuisine reviews\plots\workflow2.png

Recall : ratio of a number of events you can correctly recall to a number of all correct events.

If you can recall all 10 events correctly, then, your recall ratio is 1.0 (100%).

If you can recall 7 events correctly out of 10 events, your recall ratio is 0.7 (70%).

Precision : ratio of a number of events you can correctly recall to a number all events you recall (mix of correct and wrong recalls).

In other words, it is how precise is your recall.

If 15 events are predicted as correct, out of which 10 events are correctly recalled, and 5 are wrong then precision

score is [10/(10+5)], i.e. 66.67%.

Part 1: Bag of Words Models

Explanation:

Bag of words is a simple representation of a bag full of words, ignoring the grammar and even word order but keeping word multiplicity. Occurrence of every word is taken into account and is used as a feature for training a classifier.

Example: John likes to watch movies. Mary likes movies too.

John also likes to watch football games.

Based on these two text a list is created with unique words:

“John” “likes” “to” “watch” “movies” “Mary” “too” “also” “football” “games”

Occurrence in Sentence 1 : 1 2 1 1 2 1 1 0 0 0

Occurrence in Sentence 2 : 1 1 1 1 0 0 0 1 1 1

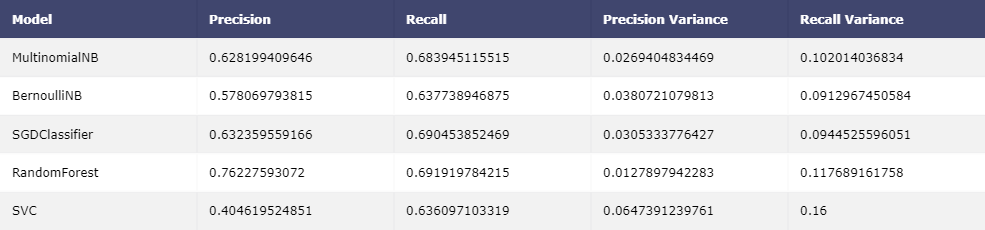
In Scikit-learn, this can be done using CountVectorizer(). It converts a collection of text documents to a matrix of token counts.

For the baseline bag of words models, ngram is kept at (1,1) and max\_features at 1000.

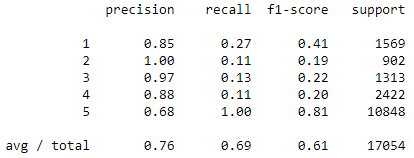
Max\_features : only considers the top max\_features ordered by term frequency across the corpus.

ngram\_range : will be explained later in this report.

Results:



**RandomForestClassifier** with a precision score of 76.2%, varying by just 1.2% and a recall of 69.12% with a variance of 11.7%, clearly outperforms the other models.

 But, still the results are not much desirable.

Comparing the recall values for all the Rating Categories, this model predicts Rating 5 perfectly, this is

because of the imbalanced data problem.

Rating 1 gets an F1 - score of 41%, which is okay, although not good. But for rating 2,3 and 4, the

results are very poor.

NOTE: From the workflow diagram, the model that performs best in Bag of Words Models will be taken forward, in this case it is RandomForestClassifier.

Part 2: Term Frequency – Inverse Document Frequency Analysis (tf – idf)

This is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor.

The goal of using tf-idf instead of the raw frequencies of occurrence of a token in a given document is to scale down the impact of tokens that occur very frequently in a given corpus and that are hence empirically less informative than features that occur in a small fraction of the training corpus.

Mathematically, tf(t) = (number of times term ‘t’ appears in a document) / (total number of terms in a document)

idf(t) = log\_e [(total number of documents) / (number of documents with term ‘t’ on it)]

Example: Suppose there are 10,000,000 documents. Consider a particular document has 100 words and the term “cat” appears 3 times in that document. And, there are

Total 1000 documents where the term “cat” appears.

Now, tf(t) = 3 / 100 = 0.03

And, idf(t) = log(10,000,000 / 1000) = 4

Therefore, tf-idf(t) = 4 \* 0.03 = 0.12

Similarly, another term “liked” appeared 20 times in that document of 100 words and this term appeared in total 1500 documents out of 10,000,000.

Now, tf(t) = 20/100 = 0.2

And, idf(t) = log(10,000,000 / 1500) = 3.82

Therefore, tf-idf(t) = 3.82 \* 0.2 = 0.76

As we can see, the tf-idf value for term “liked” is much higher than the term “cat”, that means the term “liked” appeared more number of times and hence is

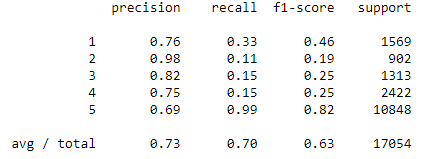
less informative than “cat”.

In Scikit-learn, this can be done by TfidfTransformer**().**

Results**:** Here we trained a RandomForestClassifier with different learners, i.e. 100, 500 and 1000.



RandomForest with 1000 learners performed slightly better than the others. Comparing this model with the Bag of Words model, we notice that there has been a slight increase in Recall value, which is good. Also Precision value decreased by 3% with a decrease in precision variance as well.

 We can also see an increase in recall values for Ratings 1,2,3 and 4.

Part 3: “ngram” Analysis

An ***n*-gram** is a contiguous sequence of *n* items from a given sequence of text or speech. The items can be [phonemes](https://en.wikipedia.org/wiki/Phoneme), syllables, [letters](https://en.wikipedia.org/wiki/Letter_(alphabet)), [words](https://en.wikipedia.org/wiki/Word) or [base pairs](https://en.wikipedia.org/wiki/Base_pairs) according to the application. The *n*-grams typically are collected from a [text](https://en.wikipedia.org/wiki/Text_corpus) or [speech corpus](https://en.wikipedia.org/wiki/Speech_corpus).

An *n*-gram of size 1 is referred to as a "unigram", size 2 is a "bigram", size 3 is a "[trigram](https://en.wikipedia.org/wiki/Trigram)" and so on.

C:\Users\DHRUBAJIT\Desktop\Kaggle Datasets\Amazon cuisine reviews\plots\ngram_diagram.png

Why “n-gram” is useful?

Lets take an example, “ Red, not blue “ , now this sentence means that it is actually color Red and not blue.

But, when we take 1-gram (uni-gram), after removing punctuation, each word will be a token, i.e. “Red” “not” “blue”. The whole meaning of the original sentence is gone.

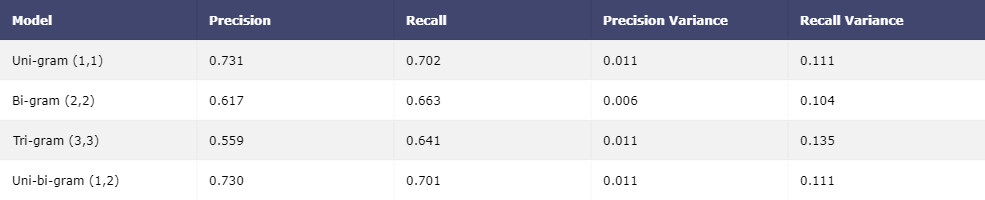
Now, when we take 2-gram (bi-gram), we get tokens “Red not” “not blue” , in this case the token “not blue” still maintains the true meaning of the original sentence.

With (1,2) - gram, we get single as well as double tokens, i.e. “Red” “not” “blue” “Red not” “not blue” , with this we can keep the actual meaning of the original, also divide each word into a token for the machine to learn.

This is how “ngram” helps in understanding deeper meanings of texts or corpus.

In Scikit-learn, “ngram” can be accessed from the CountVectorizer() method.

Results: Here we trained the RandomForestClassifier with uni-gram, bi-gram, tri-gram and uni-bi-gram.

****

The bi-gram and tri-gram models performed very poorly. In contrast, uni-gram and uni-bi-gram models have higher precision and recall values, with almost similar results.

Lets take Uni-gram model as our best performing model, since its precision and recall values are slightly higher than the other by 0.1%

Conclusion:

Lets summarize our best performing Model for this exercise.

Model:

RandomForestClassifier() { n\_estimators: 1000 }

CountVectorizer(): { 1. Max\_features = 1000

2. ngram\_range = (1,1) }

TfidfTransformer()

Results:

Confusion Matrix Classification Report

C:\Users\DHRUBAJIT\Desktop\Kaggle Datasets\Amazon cuisine reviews\plots\cmatrix.png 