Final Team Project Report: Amazon Fine Food Reviews Analysis

December 6th, 2024

Rishabh Pathak, Shubham Gondralwar, Narendra Iyer

Team AI 15

AAI 501: Introduction to Artificial Intelligence

Fall 2024

University of San Diego

Table of Contents

Abstract

This thesis explores sentiment analysis on Amazon food reviews to classify customer sentiments as positive, negative, or neutral. By leveraging natural language processing (NLP) techniques and machine learning algorithms, the project analyzes textual data to extract insights that can help businesses understand customer satisfaction and product quality. Logistic Regression, Support Vector Machines (SVM), and Deep Learning models are implemented and compared in terms of accuracy, efficiency, and computational requirements. The findings highlight the trade-offs between different algorithms and their suitability for sentiment classification tasks.

Chapter 1: Introduction

The rapid growth of e-commerce platforms has led to an explosion of customer feedback in the form of reviews. These reviews provide valuable insights into customer satisfaction, preferences, and product issues. Sentiment analysis, a subfield of NLP, is used to determine the sentiment or opinion expressed in text data. This project focuses on analyzing Amazon food reviews to classify customer sentiments as positive, negative, or neutral.

The goals of this project are:

1. To preprocess and clean raw textual data for effective sentiment classification.
2. To implement and evaluate various machine learning algorithms for sentiment analysis.
3. To compare the performance of Logistic Regression, SVM, and Deep Learning models in terms of accuracy and efficiency.

Chapter 2: Literature Review

Sentiment analysis has been extensively studied in the field of NLP, employing various algorithms and techniques to derive insights from text data. This project incorporates a combination of traditional and modern approaches to sentiment classification.

2.1: Natural Language Processing (NLP)

NLP encompasses techniques to process and analyze textual data. Common preprocessing methods include:

- **Tokenization**: Splitting text into individual words or tokens.
- **Stopword Removal**: Eliminating frequently occurring words (e.g., "the", "and") that do not contribute to sentiment analysis.
- **Lemmatization**: Reducing words to their base forms (e.g., "running" to "run").
    - These preprocessing steps standardize and clean the text, making it suitable for analysis.

2.2: Feature Extraction

Feature extraction transforms text into numerical data:

- **TF-IDF (Term Frequency-Inverse Document Frequency)**: Captures the importance of words based on their frequency in individual reviews and the overall corpus. TF-IDF provides a sparse matrix representation, suitable for machine learning algorithms.

2.3: Algorithms Used

1. **Logistic Regression**:
    - A statistical method for binary and multi-class classification.
    - Efficient and interpretable, making it a baseline for many NLP tasks.
2. **Support Vector Machines (SVM)**:
    - Robust for high-dimensional data like TF-IDF matrices.

- Finds an optimal hyperplane to separate classes.

3. **Deep Learning**:
    - Utilizes neural networks with multiple layers to model complex relationships.
    - Applied here as a feed-forward network with hidden layers and dropout regularization.

2.4: Tools and Libraries

- **Scikit-learn**: For machine learning algorithms and preprocessing.
- **Keras**: For building and training the deep learning model.
- **NLTK**: For text preprocessing, including tokenization and stopword removal.

These technologies collectively enable effective sentiment classification.

Chapter 3: Methodology

The methodology involves several systematic steps, each corresponding to Python code in the project notebook.

3.1: Data Preprocessing

1. **Loading the Dataset**: The dataset contains Amazon food reviews labeled with sentiments (positive, neutral, negative).
2. **Cleaning Text**:
    - Removing punctuation and special characters.
    - Converting text to lowercase.
    - Eliminating stopwords using NLTK.
    - Lemmatizing tokens to standardize word forms.
3. **Handling Class Imbalance**:
    - Oversampling the minority class to ensure balanced representation.

3.2: Feature Engineering

The text is converted into numerical representations using:

- **TF-IDF Vectorization**: Captures the importance of words by their frequency and rarity.

3.3: Model Implementation

1. **Logistic Regression**:
    - Trained on the TF-IDF matrix with hyperparameter tuning using Grid Search.
    - Evaluated using metrics such as accuracy and F1-score.
2. **SVM**:
    - Implemented with a linear kernel for efficient training.
    - High computational demand addressed by selecting top features via TF-IDF.
3. **Deep Learning**:
    - Architecture:
        - Input layer: Accepts TF-IDF features.
        - Two hidden layers: Use ReLU activation for non-linearity.
        - Dropout: Prevents overfitting.
        - Output layer: Softmax activation for multi-class classification.

3.4: Evaluation Metrics

The models are evaluated using:

- **Accuracy**: Overall correctness of predictions.
- **Precision, Recall, F1-score**: Evaluate performance for each sentiment class.

Chapter 4: Results and Discussion

The results of the sentiment analysis models are summarized below:

4.1: Logistic Regression

- **Accuracy**: 88.5%
- **Precision, Recall, F1-score**:
  - Positive: 89%, 88%, 88.5%
  - Neutral: 86%, 85%, 85.5%
  - Negative: 88%, 87%, 87.5%
- **Discussion**:
  - Logistic Regression performed reliably across all sentiment classes.
  - It served as a strong baseline due to its simplicity and efficiency.

4.2: Support Vector Machines (SVM)

- **Accuracy**: 88.3%
- **Precision, Recall, F1-score**:
  - Positive: 88%, 88%, 88%
  - Neutral: 85%, 84%, 84.5%
  - Negative: 87%, 86%, 86.5%
- **Discussion**:
  - Comparable performance to Logistic Regression but required longer training times.
  - Effective in handling high-dimensional TF-IDF features.

4.3: Deep Learning

- **Accuracy**: 89.2%
- **Precision, Recall, F1-score**:
  - Positive: 90%, 89%, 89.5%
  - Neutral: 86%, 85%, 85.5%
  - Negative: 88%, 87%, 87.5%
- **Discussion**:

- Demonstrated slightly higher accuracy than traditional models.
- Required substantial computational resources and careful tuning.

4.4: Comparative Analysis

Model	Accuracy	Training Time	Computational Demand
Logistic Regression	88.5%	Low	Low
SVM	88.3%	High	Medium
Deep Learning	89.2%	Medium	High

- **Conclusion**:
  - Logistic Regression is ideal for quick, reliable results.
  - SVM offers robust performance but is computationally expensive.
  - Deep Learning captures more complex patterns but requires optimization.

Chapter 5: Conclusion

The results of the sentiment analysis project indicate the following key findings:

5.1: Observations

1. The TF-IDF feature extraction technique effectively transformed text data into a numerical format suitable for machine learning models.
2. Preprocessing steps such as removing noise and balancing the classes significantly improved model performance.
3. The choice of algorithm impacts the trade-off between accuracy and computational efficiency.

5.2: Recommendations

1. Explore advanced word embeddings such as Word2Vec or BERT for feature representation.

2. Optimize deep learning models with hyperparameter tuning for further performance gains.

3. Utilize larger datasets to enhance the generalization capabilities of the models.

References

1. Kaggle Dataset: Amazon Food Reviews
2. Scikit-learn Documentation: https://scikit-learn.org
3. Gensim Documentation: https://radimrehurek.com/gensim/
4. NLTK Documentation: https://www.nltk.org

Project Code in Python

The Python Notebook of this project is available in the next page.

Team AI 15
Rishabh Pathak, Shubham Gondralwar, Narendra Iyer
Final Team Project
AAI-501 Introduction to Artificial Intelligence
University of San Diego

In [2]:
```
!wget --header="Host: storage.googleapis.com" --header="User-Age
```

```
zsh:1: command not found: wget
```

In [3]:
```
!unzip archive
```

```
unzip:  cannot find or open archive, archive.zip or archive.ZIP.
```

Amazon Fine Food Reviews Analysis

Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/

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.

In [5]:

```
%matplotlib inline

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
```

```
import os

import warnings
warnings.filterwarnings("ignore")
```

1. Loading the data

1.1. Reading Data

The dataset is available in two forms

1. .csv file
2. 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".

In [9]:
```
# Using the SQLite Table to read data
import os
db_path = os.path.abspath("database.sqlite")
con = sqlite3.connect(db_path)

filtered_data = pd.read_sql_query("""SELECT * FROM Reviews WHERE

# Giving reviews with Score>3 a positive rating, and reviews wit
def partition(x):
    if x < 3:
        return 0
    return 1

# Changing reviews with score less than 3 to be positive and vic
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

```
Number of data points in our data (5000, 10)
```

Out[9]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNume
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	

In [10]:

```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)

print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[10]:

	UserId	ProductId	ProfileName	Time	Score	
0	#oc-R115TNMSPFT9I7	B005ZBZLT4	Breyton	1331510400	2	ju
1	#oc-R11D9D7SHXIJB9	B005HG9ESG	Louis E. Emory "hoppy"	1342396800	5	
2	#oc-R11DNU2NBKQ23Z	B005ZBZLT4	Kim Cieszykowski	1348531200	1	ui
3	#oc-R11O5J5ZVQE25C	B005HG9ESG	Penguin Chick	1346889600	5	th
4	#oc-R12KPBODL2B5ZD	B007OSBEV0	Christopher P. Presta	1348617600	1	

In [11]:

```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[11]:

	UserId	ProductId	ProfileName	Time	Sco
80638	AZY10LLTJ71NX	B001ATMQK2	undertheshrine "undertheshrine"	1296691200	

In [12]:

```
display['COUNT(*)'].sum()
```

Out[12]: 393063

2. Exploratory Data Analysis

2.1 Data Cleaning: Deduplication

It is observed (as shown in the table below) that the reviews data had many duplicate entries. Hence it was necessary to remove duplicates in order to get unbiased results for the analysis of the data. Following is an example:

In [15]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[15]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNun
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	

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.

In [17]:
```
# Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascer
```

In [18]:
```
# Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName'
final.shape
```

Out[18]:
```
(4986, 10)
```

In [19]:
```
# Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[19]:
```
99.72
```

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calcualtions

In [21]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[21]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNum
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	

In [22]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenomir
```

In [23]:
```
# Before starting the next phase of preprocessing lets see the r
print(final.shape)

# How many positive and negative reviews are present in our data
final['Score'].value_counts()
```

```
(4986, 10)
```

Out[23]:
```
Score
1    4178
0     808
Name: count, dtype: int64
```

3. Text Preprocessing.

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

In [26]:

```
# Printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)

sent_1000 = final['Text'].values[1000]
print(sent_1000)
print("="*50)

sent_1500 = final['Text'].values[1500]
print(sent_1500)
print("="*50)

sent_4900 = final['Text'].values[4900]
print(sent_4900)
print("="*50)
```

```
Why is this $[...] when the same product is available for $[...]
here?<br />http://www.amazon.com/VICTOR-FLY-MAGNET-BAIT-REFILL/d
p/B00004RBDY<br /><br />The Victor M380 and M502 traps are unrea
l, of course -- total fly genocide. Pretty stinky, but only righ
t nearby.
==================================================
I recently tried this flavor/brand and was surprised at how deli
cious these chips are.  The best thing was that there were a lot
of "brown" chips in the bsg (my favorite), so I bought some more
through amazon and shared with family and friends.  I am a littl
e disappointed that there are not, so far, very many brown chips
in these bags, but the flavor is still very good.  I like them b
etter than the yogurt and green onion flavor because they do not
seem to be as salty, and the onion flavor is better.  If you hav
en't eaten Kettle chips before, I recommend that you try a bag b
efore buying bulk.  They are thicker and crunchier than Lays but
just as fresh out of the bag.
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea w
hat they were ordering; the other wants crispy cookies.  Hey, I'
m sorry; but these reviews do nobody any good beyond reminding u
s to look  before ordering.<br /><br />These are chocolate-oatme
al cookies.  If you don't like that combination, don't order thi
s type of cookie.  I find the combo quite nice, really.  The oat
meal sort of "calms" the rich chocolate flavor and gives the coo
kie sort of a coconut-type consistency.  Now let's also remember
that tastes differ; so, I've given my opinion.<br /><br />Then,
these are soft, chewy cookies -- as advertised.  They are not "c
rispy" cookies, or the blurb would say "crispy," rather than "ch
ewy."  I happen to like raw cookie dough; however, I don't see w
here these taste like raw cookie dough.  Both are soft, however,
so is this the confusion?  And, yes, they stick together.  Soft
cookies tend to do that.  They aren't individually wrapped, whic
h would add to the cost.  Oh yeah, chocolate chip cookies tend t
o be somewhat sweet.<br /><br />So, if you want something hard a
nd crisp, I suggest Nabiso's Ginger Snaps.  If you want a cookie
that's soft, chewy and tastes like a combination of chocolate an
d oatmeal, give these a try.  I'm here to place my second order.
==================================================
love to order my coffee on amazon.  easy and shows up quickly.<b
r />This k cup is great coffee.  dcaf is very good as well
==================================================
```

In [27]:

```python
# Removing urls from text - Python: https://stackoverflow.com/a/
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
Why is this $[...] when the same product is available for $[...]
here?<br /> /><br />The Victor M380 and M502 traps are unreal, o
f course -- total fly genocide. Pretty stinky, but only right ne
arby.
```

In [28]:

```
# https://stackoverflow.com/questions/16206380/python-beautifuls
from bs4 import BeautifulSoup

soup = BeautifulSoup(sent_0, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1000, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1500, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_4900, 'lxml')
text = soup.get_text()
print(text)
```

```
Why is this $[...] when the same product is available for $[...]
here? />The Victor M380 and M502 traps are unreal, of course --
total fly genocide. Pretty stinky, but only right nearby.
==================================================
I recently tried this flavor/brand and was surprised at how deli
cious these chips are.  The best thing was that there were a lot
of "brown" chips in the bsg (my favorite), so I bought some more
through amazon and shared with family and friends.  I am a littl
e disappointed that there are not, so far, very many brown chips
in these bags, but the flavor is still very good.  I like them b
etter than the yogurt and green onion flavor because they do not
seem to be as salty, and the onion flavor is better.  If you hav
en't eaten Kettle chips before, I recommend that you try a bag b
efore buying bulk.  They are thicker and crunchier than Lays but
just as fresh out of the bag.
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea w
hat they were ordering; the other wants crispy cookies.  Hey, I'
m sorry; but these reviews do nobody any good beyond reminding u
s to look  before ordering.These are chocolate-oatmeal cookies.
If you don't like that combination, don't order this type of coo
kie.  I find the combo quite nice, really.  The oatmeal sort of
"calms" the rich chocolate flavor and gives the cookie sort of a
coconut-type consistency.  Now let's also remember that tastes d
iffer; so, I've given my opinion.Then, these are soft, chewy coo
kies -- as advertised.  They are not "crispy" cookies, or the bl
urb would say "crispy," rather than "chewy."  I happen to like r
aw cookie dough; however, I don't see where these taste like raw
cookie dough.  Both are soft, however, so is this the confusion?
And, yes, they stick together.  Soft cookies tend to do that.  T
hey aren't individually wrapped, which would add to the cost.  O
h yeah, chocolate chip cookies tend to be somewhat sweet.So, if
you want something hard and crisp, I suggest Nabiso's Ginger Sna
ps.  If you want a cookie that's soft, chewy and tastes like a c
ombination of chocolate and oatmeal, give these a try.  I'm here
to place my second order.
==================================================
love to order my coffee on amazon.  easy and shows up quickly.Th
is k cup is great coffee.  dcaf is very good as well
```

In [29]:

```python
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
```

```
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [30]:
```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
Wow.  So far, two two-star reviews.  One obviously had no idea w
hat they were ordering; the other wants crispy cookies.  Hey, I
am sorry; but these reviews do nobody any good beyond reminding
us to look  before ordering.<br /><br />These are chocolate-oatm
eal cookies.  If you do not like that combination, do not order
this type of cookie.  I find the combo quite nice, really.  The
oatmeal sort of "calms" the rich chocolate flavor and gives the
cookie sort of a coconut-type consistency.  Now let is also reme
mber that tastes differ; so, I have given my opinion.<br /><br /
>Then, these are soft, chewy cookies -- as advertised.  They are
not "crispy" cookies, or the blurb would say "crispy," rather th
an "chewy."  I happen to like raw cookie dough; however, I do no
t see where these taste like raw cookie dough.  Both are soft, h
owever, so is this the confusion?  And, yes, they stick together
.  Soft cookies tend to do that.  They are not individually wrap
ped, which would add to the cost.  Oh yeah, chocolate chip cooki
es tend to be somewhat sweet.<br /><br />So, if you want somethi
ng hard and crisp, I suggest Nabiso is Ginger Snaps.  If you wan
t a cookie that is soft, chewy and tastes like a combination of
chocolate and oatmeal, give these a try.  I am here to place my
second order.
==================================================
```

In [31]:
```
# Removing words with numbers - Python: https://stackoverflow.co
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
Why is this $[...] when the same product is available for $[...]
here?<br /> /><br />The Victor  and  traps are unreal, of course
-- total fly genocide. Pretty stinky, but only right nearby.
```

In [32]:
```
# Removing spacial character - https://stackoverflow.com/a/58435
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
Wow So far two two star reviews One obviously had no idea what t
hey were ordering the other wants crispy cookies Hey I am sorry
but these reviews do nobody any good beyond reminding us to look
before ordering br br These are chocolate oatmeal cookies If you
do not like that combination do not order this type of cookie I
find the combo quite nice really The oatmeal sort of calms the r
ich chocolate flavor and gives the cookie sort of a coconut type
consistency Now let is also remember that tastes differ so I hav
e given my opinion br br Then these are soft chewy cookies as ad
vertised They are not crispy cookies or the blurb would say cris
py rather than chewy I happen to like raw cookie dough however I
do not see where these taste like raw cookie dough Both are soft
however so is this the confusion And yes they stick together Sof
t cookies tend to do that They are not individually wrapped whic
h would add to the cost Oh yeah chocolate chip cookies tend to b
e somewhat sweet br br So if you want something hard and crisp I
suggest Nabiso is Ginger Snaps If you want a cookie that is soft
chewy and tastes like a combination of chocolate and oatmeal giv
e these a try I am here to place my second order
```

In [33]:

```
# https://gist.github.com/sebleier/554280
# We are removing the words from the stop words list: 'no', 'nor
# <br /><br /> ==> After the above steps, we are getting "br br"
# We are including them into stop words list
# Instead of <br /> if we have <br/> these tags would have revmo

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'o
            "you'll", "you'd", 'your', 'yours', 'yourself', 'you
            'she', "she's", 'her', 'hers', 'herself', 'it', "it'
            'theirs', 'themselves', 'what', 'which', 'who', 'who
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'bei
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if'
            'at', 'by', 'for', 'with', 'about', 'against', 'betw
            'above', 'below', 'to', 'from', 'up', 'down', 'in',
            'then', 'once', 'here', 'there', 'when', 'where', 'w
            'most', 'other', 'some', 'such', 'only', 'own', 'sam
            's', 't', 'can', 'will', 'just', 'don', "don't", 'sh
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "could
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'sh
            'won', "won't", 'wouldn', "wouldn't"])
```

In [34]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
```

```
    sentance = ' '.join(e.lower() for e in sentance.split() if e
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████████████████████████████████████| 4986/4986 [00:00<00:0
0, 10524.22it/s]
```

In [35]:
```
preprocessed_reviews[1500]
```

Out[35]:
```
'wow far two two star reviews one obviously no idea ordering wa
nts crispy cookies hey sorry reviews nobody good beyond remindi
ng us look ordering chocolate oatmeal cookies not like combinat
ion not order type cookie find combo quite nice really oatmeal
sort calms rich chocolate flavor gives cookie sort coconut type
consistency let also remember tastes differ given opinion soft
chewy cookies advertised not crispy cookies blurb would say cri
spy rather chewy happen like raw cookie dough however not see t
aste like raw cookie dough soft however confusion yes stick tog
ether soft cookies tend not individually wrapped would add cost
oh yeah chocolate chip cookies tend somewhat sweet want somethi
ng hard crisp suggest nabiso ginger snaps want cookie soft chew
y tastes like combination chocolate oatmeal give try place seco
nd order'
```

4. Featurization

4.1 BAG OF WORDS

In [38]:
```
# Bag of Words
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(preprocessed_reviews)
print("some feature names ", count_vect.get_feature_names_out()
print('='*50)

final_counts = count_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_counts))
print("the shape of out text BOW vectorizer ",final_counts.get_s
print("the number of unique words ", final_counts.get_shape()[1]
```

```
some feature names  ['aa' 'aahhhs' 'aback' 'abandon' 'abates' 'a
bbott' 'abby' 'abdominal'
 'abiding' 'ability']
==================================================
the type of count vectorizer  <class 'scipy.sparse._csr.csr_matr
ix'>
the shape of out text BOW vectorizer  (4986, 12997)
the number of unique words  12997
```

4.2 Bi-Grams and n-Grams

In [40]:
```
# bi-gram, tri-gram and n-gram
# Removing stop words like "not" should be avoided before buildi
# Please do read the CountVectorizer documentation http://scikit
# You can choose these numebrs min_df=10, max_features=5000, of
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_f
final_bigram_counts = count_vect.fit_transform(preprocessed_revi
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_count
print("the number of unique words including both unigrams and bi
```

```
the type of count vectorizer  <class 'scipy.sparse._csr.csr_matr
ix'>
the shape of out text BOW vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams
3144
```

4.3 TF-IDF

In [42]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_
print('='*50)

final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_tf_idf))
print("the shape of out text TFIDF vectorizer ",final_tf_idf.get
print("the number of unique words including both unigrams and bi
```

```
some sample features(unique words in the corpus) ['ability' 'abl
e' 'able find' 'able get' 'absolute' 'absolutely'
 'absolutely delicious' 'absolutely love' 'absolutely no' 'accor
ding']
==================================================
the type of count vectorizer  <class 'scipy.sparse._csr.csr_matr
ix'>
the shape of out text TFIDF vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams
3144
```

4.4 Word2Vec

In [44]:
```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance=[]
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.split())
```

In [45]:
```
# Using Google News Word2Vectors

# In this project we are using a pretrained model by google
# Its 3.3G file, once you load this into your memory
```

```
# It occupies ~9Gb, so please do this step only if you have >12G
# We will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word]
# To use this code-snippet, download "GoogleNews-vectors-negativ
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21p
# It's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-cod
# You can comment this whole cell or change these varible accord

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5
    w2v_model=Word2Vec(list_of_sentance,min_count=5,vector_size=
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_
```

```
[('excellent', 0.9796244502067566), ('definitely', 0.97565692663
19275), ('overall', 0.9747947454452515), ('alternative', 0.97422
57595062256), ('want', 0.9733856320381165), ('looking', 0.972990
9896850586), ('enjoy', 0.9723957180976868), ('regular', 0.972195
5060958862), ('others', 0.9720809459686279), ('though', 0.971841
9313430786)]
==================================================
[('remember', 0.9983727335929871), ('stash', 0.998180389404296
9), ('level', 0.9981610774993896), ('must', 0.998033344745636),
('double', 0.9980050921440125), ('terrible', 0.997997999191284
2), ('perhaps', 0.9979820847511292), ('simply', 0.99796700477600
1), ('experience', 0.9979575872421265), ('american', 0.997925281
5246582)]
```

In [46]:

```
w2v_words = list(w2v_model.wv.key_to_index)
print("number of words that occured minimum 5 times ",len(w2v_wo
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  3817
sample words  ['not', 'like', 'good', 'great', 'taste', 'one', '
product', 'would', 'flavor', 'love', 'coffee', 'food', 'chips',
'tea', 'no', 'really', 'get', 'best', 'much', 'amazon', 'use', '
time', 'buy', 'also', 'tried', 'little', 'find', 'make', 'price'
, 'better', 'bag', 'try', 'even', 'mix', 'well', 'chocolate', 'h
ot', 'eat', 'free', 'water', 'dog', 'first', 'made', 'could', 'f
ound', 'used', 'bought', 'box', 'sugar', 'cup']
```

4.4.1 Converting text into vectors using wAvg W2V, TFIDF-W2V

4.4.1.1 Avg W2v

In [49]:
```
# Average Word2Vec
# Computing average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is sto
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sent
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
100%|██████████████████████████████████████| 4986/4986 [00:01<0
0:00, 3688.49it/s]
4986
50
```

[4.4.1.2] TFIDF weighted W2v

In [51]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
model.fit(preprocessed_reviews)

# We are converting a dictionary with word as a key, and the idf
dictionary = dict(zip(model.get_feature_names_out(), list(model.
```

In [52]:
```
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names_out() # tfidf words/col-nam
# final_tf_idf is the sparse matrix with row= sentence, col=word

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/revie
row=0;
```

```
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the ser
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole cour
            # sent.count(word) = tf valeus of word in this revie
            tf_idf = dictionary[word]*(sent.count(word)/len(sent
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████| 4986/4986 [00:20<0
0:00, 240.95it/s]
```

Observations

1. **Data Distribution**: The sentiment distribution shows class imbalance, with some sentiments (e.g., positive) being more frequent than others.

2. **Preprocessing Steps**: Data cleaning and preprocessing were successfully performed, which included removing noise and balancing classes.

3. **Feature Extraction**: The use of TF-IDF effectively transformed text data into a numerical format suitable for model training.

4. **Model Performance**: - Logistic Regression showed strong performance, with high accuracy and balanced precision/recall. - Support Vector Machine (SVM) took longer to train but achieved similar performance to Logistic Regression. - The Deep Learning model demonstrated a slightly better performance but required more computational resources.

Conclusions

1. **Model Effectiveness**:
- Logistic Regression is a strong baseline model for text classification tasks and achieved competitive results with minimal computation time.
. - SVM is robust but computationally expensive, especially for large datasets. We couldn't even get it's execution to complete.
. - Deep Learning models can capture more complex patterns but require fine-tuning and substantial computational power.

2. **Feature Engineering**:
- TF-IDF proved to be a reliable feature extraction technique, contributing to high model performance.

3. **Recommendations**:
- Consider further optimizing the deep learning model for better performance.
- Explore additional feature engineering techniques or embeddings like Word2Vec or BERT for enhanced results.