NLP Project Report

Amazon Fine Food Review Analysis

Edera Venkata Naga Sai Bharadwaja
SC20M104
Machine Learning and Computing
Indian Institute of Space Sciene and Technology

Table of Contents

List of Figures

1. Introduction to Dataset

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

1.1 Overview

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

1.2 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

2. Objective

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

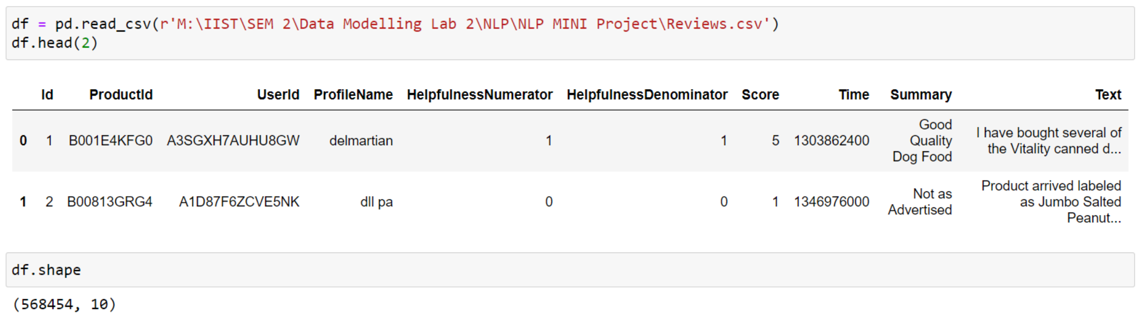
- **Ques :** How to determine if a review is positive or negative?
- **Ans :** We could use the Score/Rating. A rating of 4 or 5 could be cosnidered a positive review. A review of 1 or 2 could be considered negative. A review of 3 is nuetral and ignored. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

3. Loading the data

The dataset is available in two forms

- .csv file
- SQLite Database

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 will be set to ”positive”. Otherwise, it will be set to ”negative”.



```
df = pd.read_csv(r'M:\IIST\SEM 2\Data Modelling Lab 2\NLP\NLP MINI Project\Reviews.csv')
df.head(2)
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...

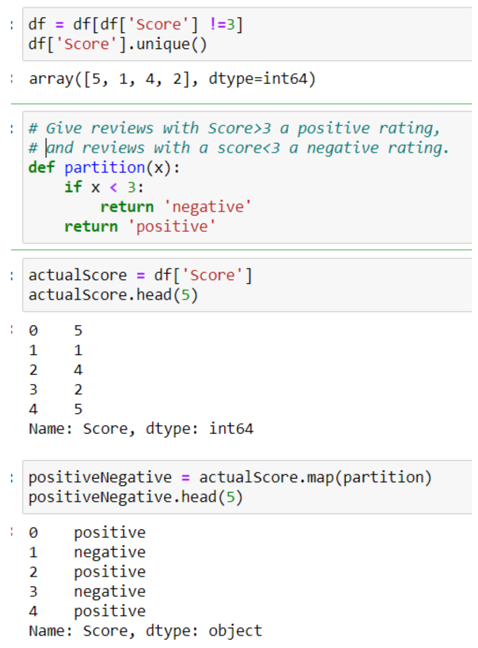
```
df.shape
```

```
(568454, 10)
```

Fig. 1. Loading the data using .csv

4. Exploratory Data Analysis

4.1 Converting Score < 3 as Negative and > 3 as positive



```
df = df[df['Score'] !=3]
df['Score'].unique()
```

```
array([5, 1, 4, 2], dtype=int64)
```

```
# Give reviews with Score>3 a positive rating,
# and reviews with a score<3 a negative rating.
def partition(x):
    if x < 3:
        return 'negative'
    return 'positive'
```

```
actualScore = df['Score']
actualScore.head(5)
```

```
0    5
1    1
2    4
3    2
4    5
Name: Score, dtype: int64
```

```
positiveNegative = actualScore.map(partition)
positiveNegative.head(5)
```

```
0    positive
1    negative
2    positive
3    negative
4    positive
Name: Score, dtype: object
```

Fig. 2. Converting Score < 3 as Negative and > 3 as positive

4.2 Cleaning: Deduplication

Every product in Amazon has got different variants and colors. If a reviwer gives a review for one particular product that he/she is bought. Then the same review is copied for all the variants of that Brand of product.Hence,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:

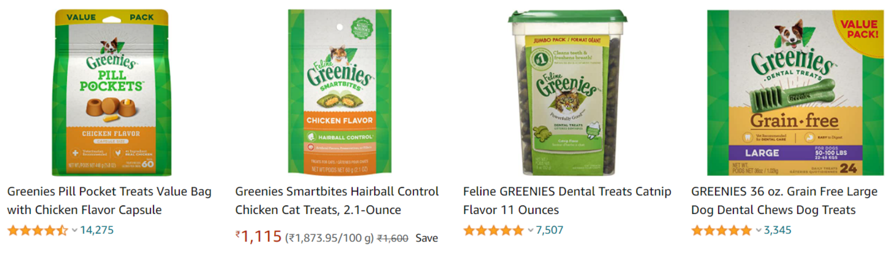
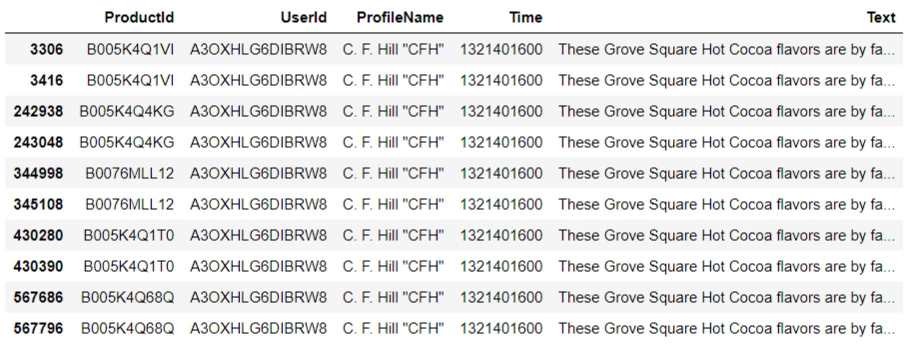
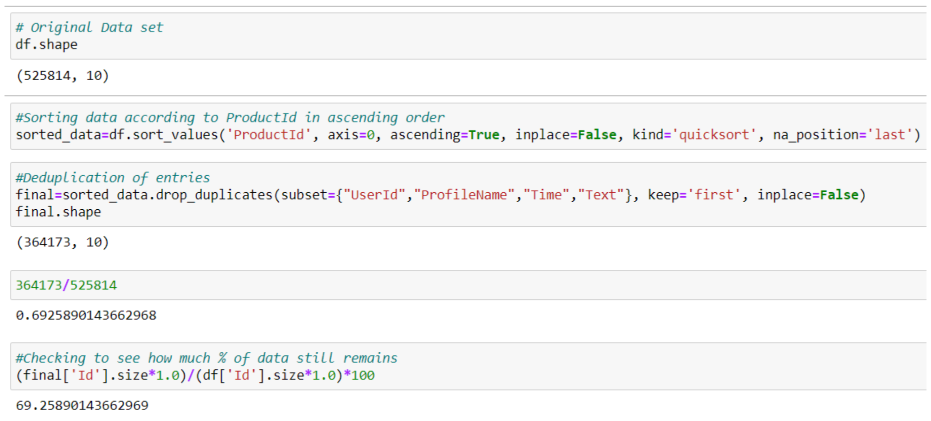

Greenies Pill Pocket Treats Value Bag with Chicken Flavor Capsule
14,275
Greenies Smartbites Hairball Control Chicken Cat Treats, 2.1-Ounce
₹1,115 (₹1,873.95/100 g) ₹1,600 Save
Feline GREENIES Dental Treats Catnip Flavor 11 Ounces
7,507
GREENIES 36 oz. Grain Free Large Dog Dental Chews Dog Treats
3,345

Fig. 3. Different variants in a particular Brand



	ProductId	UserId	ProfileName	Time	Text
3306	B005K4Q1VI	A3OXHLG6DIBRW8	C. F. Hill "CFH"	1321401600	These Grove Square Hot Cocoa flavors are by fa...
3416	B005K4Q1VI	A3OXHLG6DIBRW8	C. F. Hill "CFH"	1321401600	These Grove Square Hot Cocoa flavors are by fa...
242938	B005K4Q4KG	A3OXHLG6DIBRW8	C. F. Hill "CFH"	1321401600	These Grove Square Hot Cocoa flavors are by fa...
243048	B005K4Q4KG	A3OXHLG6DIBRW8	C. F. Hill "CFH"	1321401600	These Grove Square Hot Cocoa flavors are by fa...
344998	B0076MLL12	A3OXHLG6DIBRW8	C. F. Hill "CFH"	1321401600	These Grove Square Hot Cocoa flavors are by fa...
345108	B0076MLL12	A3OXHLG6DIBRW8	C. F. Hill "CFH"	1321401600	These Grove Square Hot Cocoa flavors are by fa...
430280	B005K4Q1T0	A3OXHLG6DIBRW8	C. F. Hill "CFH"	1321401600	These Grove Square Hot Cocoa flavors are by fa...
430390	B005K4Q1T0	A3OXHLG6DIBRW8	C. F. Hill "CFH"	1321401600	These Grove Square Hot Cocoa flavors are by fa...
567686	B005K4Q68Q	A3OXHLG6DIBRW8	C. F. Hill "CFH"	1321401600	These Grove Square Hot Cocoa flavors are by fa...
567796	B005K4Q68Q	A3OXHLG6DIBRW8	C. F. Hill "CFH"	1321401600	These Grove Square Hot Cocoa flavors are by fa...

Fig. 4. Duplicates in the Dataset



```
# Original Data set
df.shape
```

```
(525814, 10)
```

```
#Sorting data according to ProductId in ascending order
sorted_data=df.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last')
```

```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
final.shape
```

```
(364173, 10)
```

```
364173/525814
```

```
0.6925890143662968
```

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

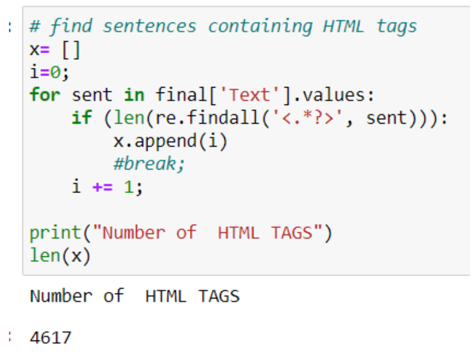
```
69.25890143662969
```

Fig. 5. Dropduplicates

Oservation : It is observed that after removing duplicates in the dataset , 69.25% of the data retain in the data set. So we are able to remove 30.75% of data in the form of duplicates

4.3 Finding and Removing HTML tags

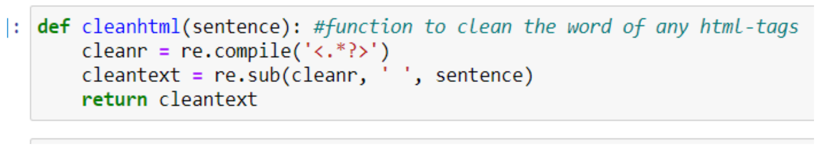
The text column in the Dataset not only contain Reviews given the user but also some html tags will be included(given by the user) to refer other product or other sources. So inorder to preprocess the Reviews we need to find HTML tags and Remove them.



```
# find sentences containing HTML tags
x= []
i=0;
for sent in final['Text'].values:
    if (len(re.findall('<.*?>', sent))):
        x.append(i)
        #break;
    i += 1;

print("Number of  HTML TAGS")
len(x)
```

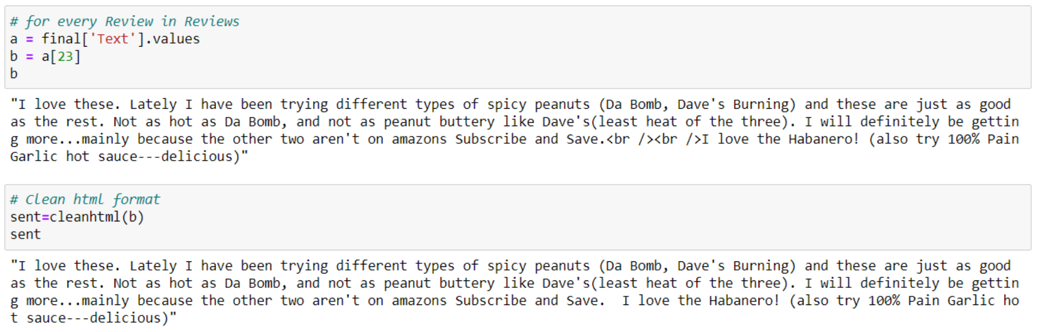
```
Number of  HTML TAGS

4617
```

Fig. 6. Finding HTML tags



```
def cleanhtml(sentence): #function to clean the word of any html-tags
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext
```

Fig. 7. Clean HTML tags using Regular Expressions



```
# for every Review in Reviews
a = final['Text'].values
b = a[23]
b
```

```
"I love these. Lately I have been trying different types of spicy peanuts (Da Bomb, Dave's Burning) and these are just as good
as the rest. Not as hot as Da Bomb, and not as peanut buttery like Dave's(least heat of the three). I will definitely be gettin
g more...mainly because the other two aren't on amazons Subscribe and Save.<br /><br />I love the Habanero! (also try 100% Pain
Garlic hot sauce---delicious)"
```

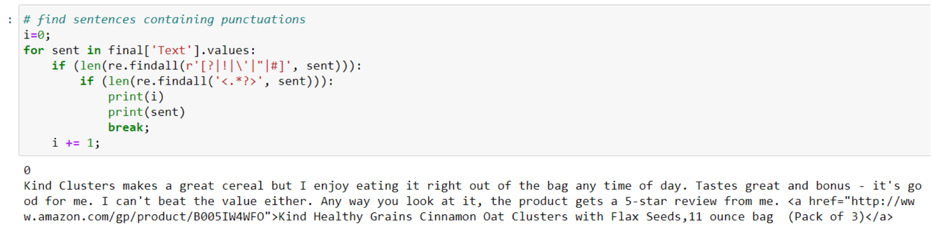
```
# Clean html format
sent=cleanhtml(b)
sent
```

```
"I love these. Lately I have been trying different types of spicy peanuts (Da Bomb, Dave's Burning) and these are just as good
as the rest. Not as hot as Da Bomb, and not as peanut buttery like Dave's(least heat of the three). I will definitely be gettin
g more...mainly because the other two aren't on amazons Subscribe and Save.  I love the Habanero! (also try 100% Pain Garlic ho
t sauce---delicious)"
```

Fig. 8. Review before and after cleaning HTML tags

4.4 Finding and Removing Punctuation’s

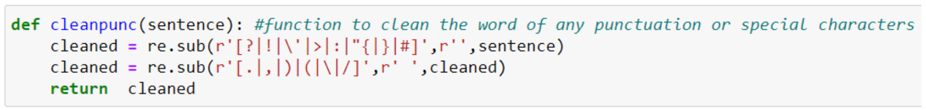
The text column in the Dataset contain Reviews with punctuation Marks . So inorder to preprocess the Reviews we need to find punctuation’s and Remove them.



```
: # find sentences containing punctuations
i=0;
for sent in final['Text'].values:
    if (len(re.findall(r'[?|!|\'|"|#]', sent))):
        if (len(re.findall('<.*?>', sent))):
            print(i)
            print(sent)
            break;
    i += 1;
```

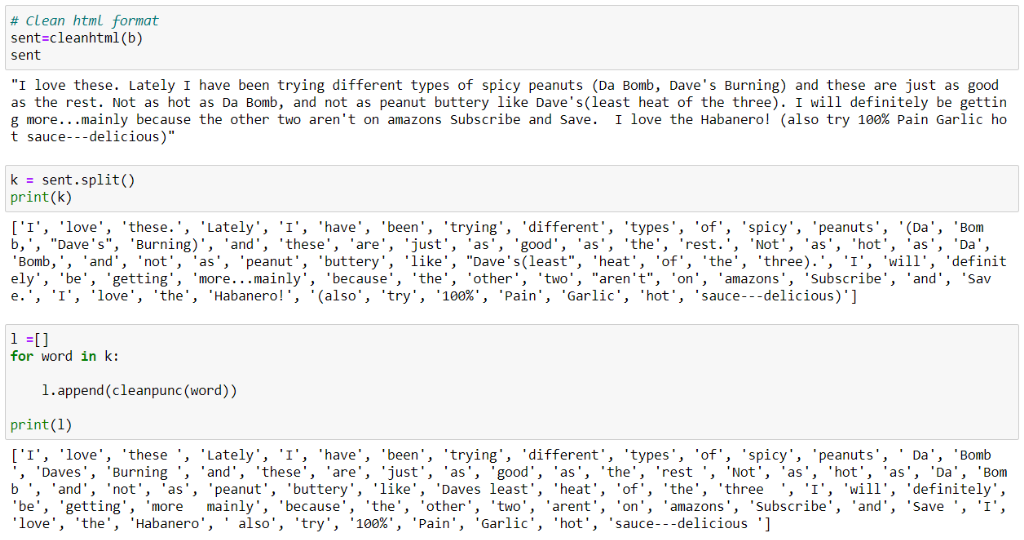
```
0
Kind Clusters makes a great cereal but I enjoy eating it right out of the bag any time of day. Tastes great and bonus - it's go
od for me. I can't beat the value either. Any way you look at it, the product gets a 5-star review from me. <a href="http://ww
w.amazon.com/gp/product/B005IW4WFO">Kind Healthy Grains Cinnamon Oat Clusters with Flax Seeds,11 ounce bag  (Pack of 3)</a>
```

Fig. 9. Finding punctuation’s



```
def cleanpunc(sentence): #function to clean the word of any punctuation or special characters
    cleaned = re.sub(r'[?|!|\'|>|:|"{|}|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
```

Fig. 10. Clean punctuation’s using Regular Expressions



```
# Clean html format
sent=cleanhtml(b)
sent
```

```
"I love these. Lately I have been trying different types of spicy peanuts (Da Bomb, Dave's Burning) and these are just as good
as the rest. Not as hot as Da Bomb, and not as peanut buttery like Dave's(least heat of the three). I will definitely be gettin
g more...mainly because the other two aren't on amazons Subscribe and Save.  I love the Habanero! (also try 100% Pain Garlic ho
t sauce---delicious)"
```

```
k = sent.split()
print(k)
```

```
['I', 'love', 'these.', 'Lately', 'I', 'have', 'been', 'trying', 'different', 'types', 'of', 'spicy', 'peanuts', '(Da', 'Bom
b,', "Dave's", 'Burning)', 'and', 'these', 'are', 'just', 'as', 'good', 'as', 'the', 'rest.', 'Not', 'as', 'hot', 'as', 'Da',
'Bomb,', 'and', 'not', 'as', 'peanut', 'buttery', 'like', "Dave's(least", 'heat', 'of', 'the', 'three).', 'I', 'will', 'definit
ely', 'be', 'getting', 'more...mainly', 'because', 'the', 'other', 'two', "aren't", 'on', 'amazons', 'Subscribe', 'and', 'Sav
e.', 'I', 'love', 'the', 'Habanero!', '(also', 'try', '100%', 'Pain', 'Garlic', 'hot', 'sauce---delicious)']
```

```
l =[]
for word in k:

    l.append(cleanpunc(word))

print(l)
```

```
['I', 'love', 'these ', 'Lately', 'I', 'have', 'been', 'trying', 'different', 'types', 'of', 'spicy', 'peanuts', ' Da', 'Bomb
', 'Daves', 'Burning ', 'and', 'these', 'are', 'just', 'as', 'good', 'as', 'the', 'rest ', 'Not', 'as', 'hot', 'as', 'Da', 'Bom
b ', 'and', 'not', 'as', 'peanut', 'buttery', 'like', 'Daves least', 'heat', 'of', 'the', 'three  ', 'I', 'will', 'definitely',
'be', 'getting', 'more   mainly', 'because', 'the', 'other', 'two', 'arent', 'on', 'amazons', 'Subscribe', 'and', 'Save ', 'I',
'love', 'the', 'Habanero', ' also', 'try', '100%', 'Pain', 'Garlic', 'hot', 'sauce---delicious ']
```

Fig. 11. Review before and after cleaning punctuation’s

5. Vector Semantics / Word Embedding

Ques : Why is Vectorizing text and transformations is so important ?

- vectors as a way of projecting words onto mathematical space while preserving the information provided by these words. In machine learning this feature is called feature vector as each value corresponds to some feature, which are used to make predictions.
- Different Algorithms to get Word Embeddings / Vector Semantics
    1. Context-independent (Bag of Words, TF-IDF, Word2Vec, GloVe)
    2. Context-aware (ELMo, Transformer, BERT, Transformer-XL)
    3. Large model (GPT-2, XLNet, Compressive Transformer)

Word embeddings are word vector representations where words with similar meaning have similar representation. Word vectors are one of the most efficient ways to represent words.

word2vec has two architectures

1. CBOW(Continuous Bag of Words)
2. Skip Gram

5.1 Skip Gram Architecture

In skipgram model, we try to predict each context word given a target word.

- collect word co-occurrence from the data
- context window : To get a words that are Close to Certain Words

Example : Deep Learning is very easy and Fun

- Let window size = 2
- RED box represents Target Word
- BLUE box represents Context Words

Window pairs :

1. 1st Window pairs: (Deep, Learning), (Deep, is)
2. 2nd Window pairs: (Learning, Deep), (Learning, is), (Learning, easy)
3. 3rd Window pairs: (is, Deep), (is, Learning), (is, very), (is, easy)

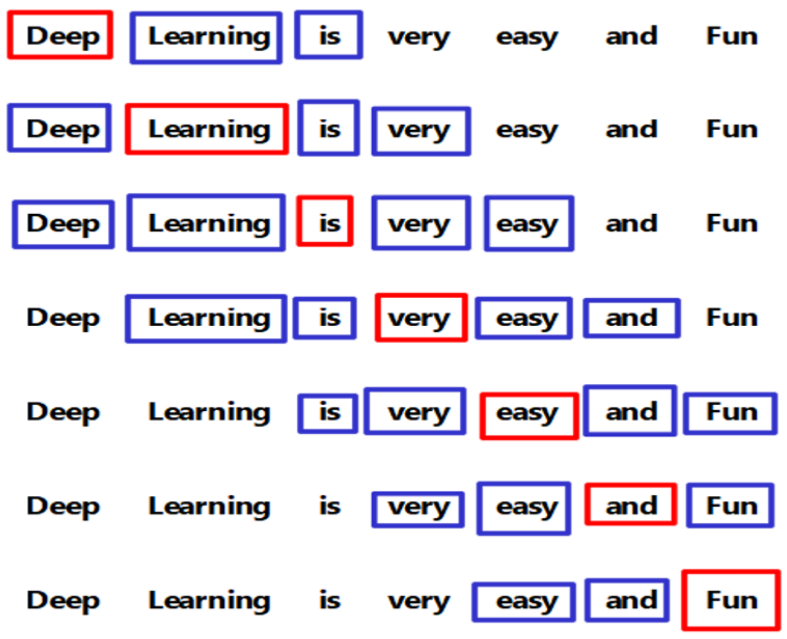

Deep Learning is very easy and Fun
Deep Learning is very easy and Fun
Deep Learning is very easy and Fun
Deep Learning is very easy and Fun
Deep Learning is very easy and Fun
Deep Learning is very easy and Fun
Deep Learning is very easy and Fun

Fig. 12. Skip gram Example

5.1.1 Target word vs Context word Data set

1. (Deep, Learning)
2. (Deep, is)
3. (Learning, Deep)
4. (Learning, is)
5. (Learning, easy)
6. (is, Deep)
7. (is, Learning)
8. (is, very)
9. (is, easy)
10. (very, learning)
11. (very, is)
12. (very, easy)
13. (very, and)
14. (easy, is)
15. (easy, very)
16. (easy, and)
17. (easy, fun)
18. (and, very)
19. (and, easy)
20. (and, fun)
21. (fun, easy)
22. (fun, and)

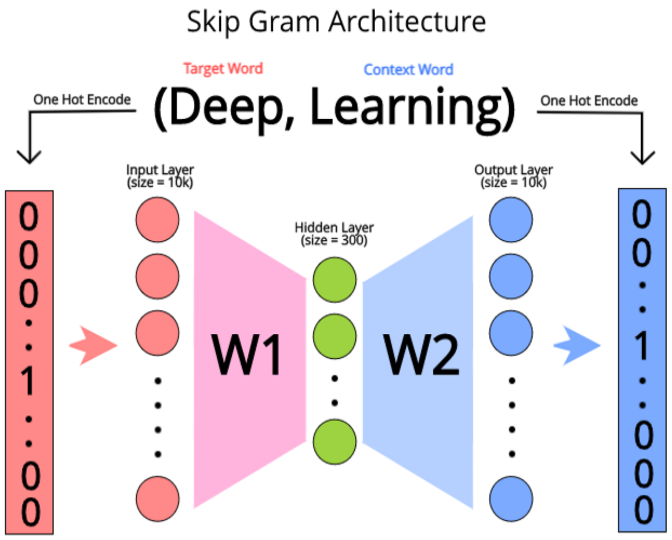


Fig. 13. Skip gram Training

5.2 How to Train Skip gram

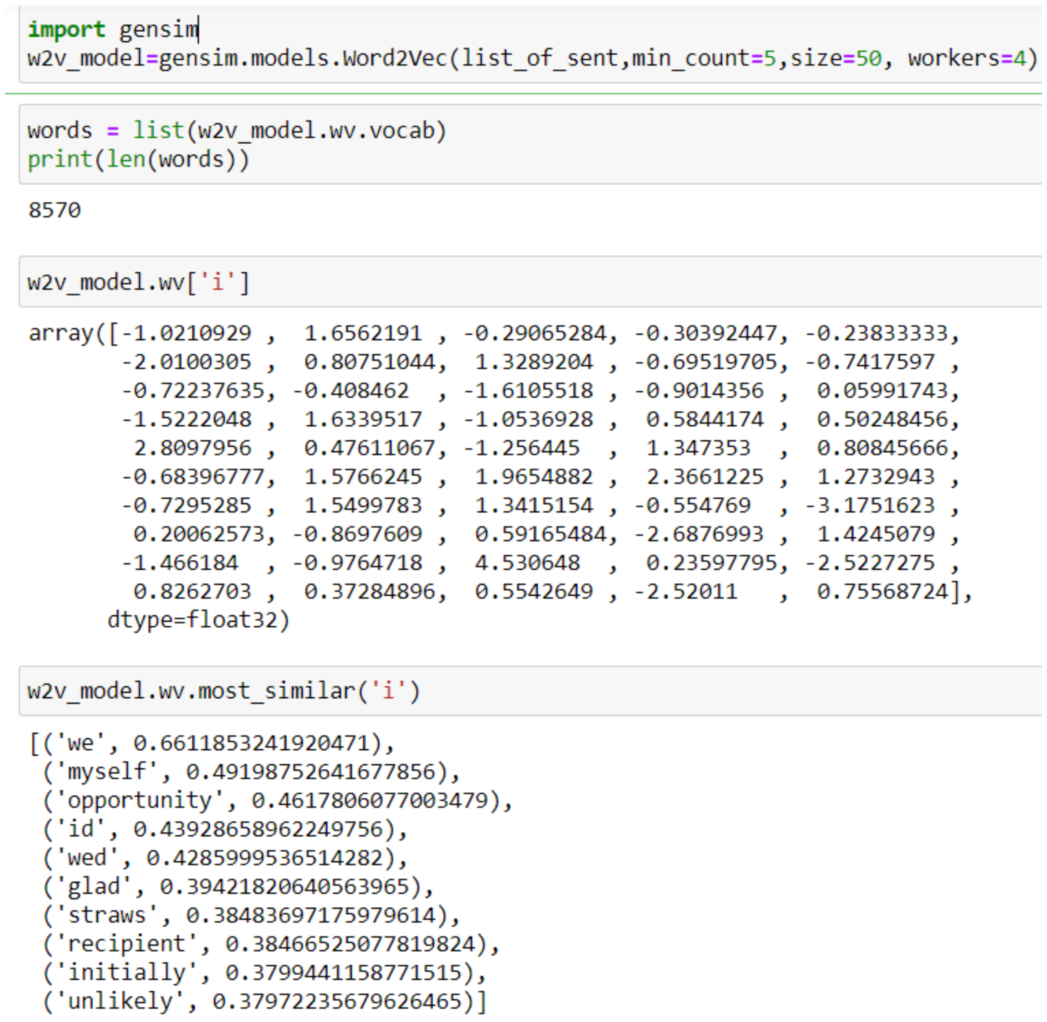
- For Every target and context word in the Target and Context word dataset , Target word is given as input and Context word is given as output(Label) to the Skip gram architecture.
- So from the Target and Context word dataset that we have , we train the skipgram architecture with a single hidden layer
- Number of Neurons in the Hidden Layer = Length of the Output vector corresponding to each word
- Once we train the skip gram architecture . So every word in the dataset is given as input and output will be the corresponding vector taken from the hidden Layer
- So finally we are able to get Vectors for each every word in the Reviews Dataset

The skip-gram objective function sums the log probabilities of the surrounding n words to the left and right of the target word w_t to produce the following objective:

$$J_\theta = \frac{1}{T} \sum_{t=1}^{T} \sum_{-n \leq j \leq n, \neq 0} \log p(w_{j+1} \mid w_t)$$

6. Train own Word2Vec model using gensim models

Gensim is an open-source library for unsupervised topic modeling and natural language processing, using modern statistical machine learning.. Gensim is implemented in Python and Cython. Gensim is designed to handle large text collections using data streaming and incremental online algorithms, which differentiates it from most other machine learning software packages that target only in-memory processing.
As ML algorithms doesn't handle the raw text,we need to convert the words into corresponding vectors . So,we train the Gensim models to convert each word into a vector of length 50



```
import gensim
w2v_model=gensim.models.Word2Vec(list_of_sent,min_count=5,size=50, workers=4)
```

```
words = list(w2v_model.wv.vocab)
print(len(words))
```

```
8570
```

```
w2v_model.wv['i']
```

```
array([-1.0210929 ,  1.6562191 , -0.29065284, -0.30392447, -0.23833333,
       -2.0100305 ,  0.80751044,  1.3289204 , -0.69519705, -0.7417597 ,
       -0.72237635, -0.408462  , -1.6105518 , -0.9014356 ,  0.05991743,
       -1.5222048 ,  1.6339517 , -1.0536928 ,  0.5844174 ,  0.50248456,
        2.8097956 ,  0.47611067, -1.256445  ,  1.347353  ,  0.80845666,
       -0.68396777,  1.5766245 ,  1.9654882 ,  2.3661225 ,  1.2732943 ,
       -0.7295285 ,  1.5499783 ,  1.3415154 , -0.554769  , -3.1751623 ,
        0.20062573, -0.8697609 ,  0.59165484, -2.6876993 ,  1.4245079 ,
       -1.466184  , -0.9764718 ,  4.530648  ,  0.23597795, -2.5227275 ,
        0.8262703 ,  0.37284896,  0.5542649 , -2.52011   ,  0.75568724],
      dtype=float32)
```

```
w2v_model.wv.most_similar('i')
```

```
[('we', 0.6611853241920471),
 ('myself', 0.49198752641677856),
 ('opportunity', 0.4617806077003479),
 ('id', 0.43928658962249756),
 ('wed', 0.4285999536514282),
 ('glad', 0.39421820640563965),
 ('straws', 0.38483697175979614),
 ('recipient', 0.38466525077819824),
 ('initially', 0.3799441158771515),
 ('unlikely', 0.37972235679626465)]
```

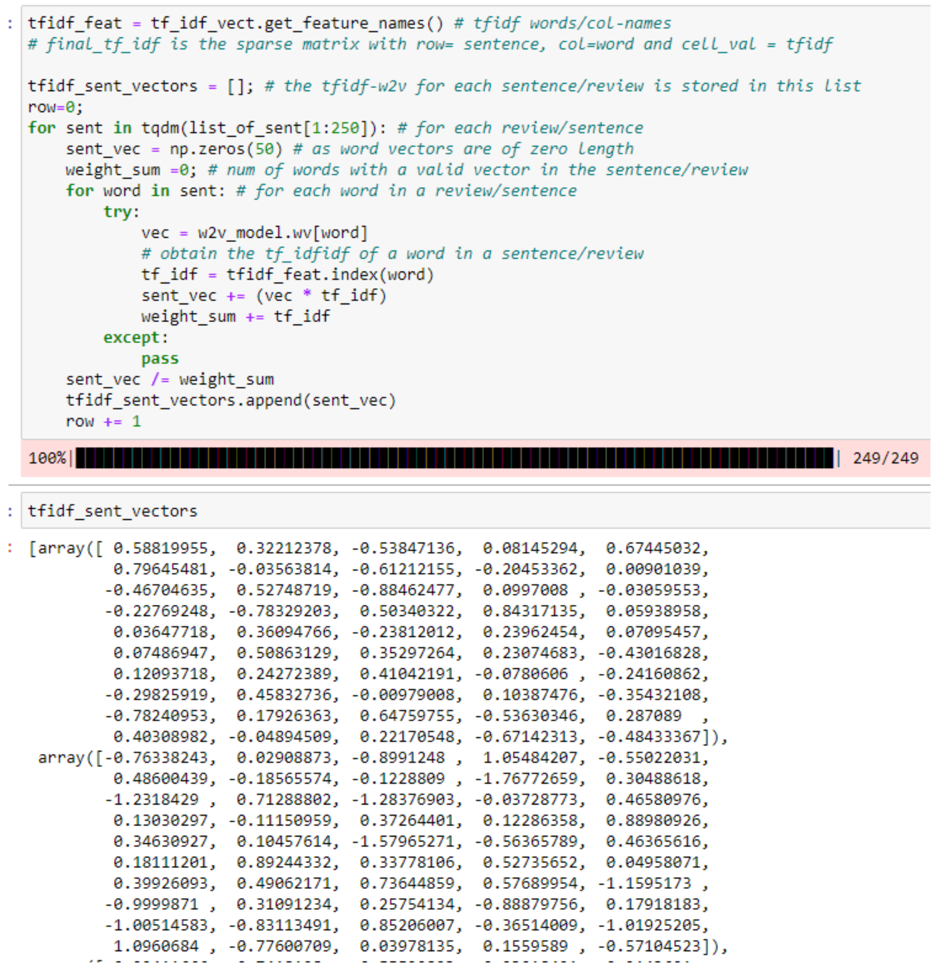
Fig. 14. Traning word2vec using gensim.models

7. TF-IDF w2v

TF-IDF is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. This is done by multiplying two metrics: how many times a word appears in a document, and the inverse document frequency of the word across a set of documents.

It has many uses, most importantly in automated text analysis, and is very useful for scoring words in machine learning algorithms for Natural Language Processing (NLP).

We can't train ML algorithm by converting words into vectors alone. So we also need to convert each Review(sentence) into a vector by making use of those vectors of word of length 50.This can be achieved by using TFIDF w2v



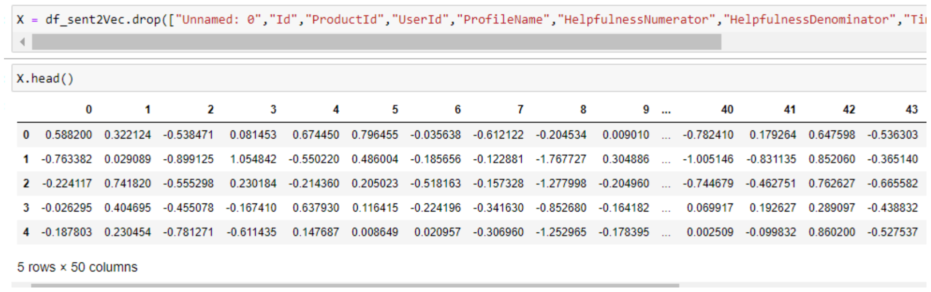
```
tfidf_feat = tf_idf_vect.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sent[1:250]): # 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 sentence/review
    for word in sent: # for each word in a review/sentence
        try:
            vec = w2v_model.wv[word]
            # obtain the tf_idfidf of a word in a sentence/review
            tf_idf = tfidf_feat.index(word)
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
        except:
            pass
    sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|████████████████████████████████████████████████████████████| 249/249
```

```
tfidf_sent_vectors
```

```
[array([ 0.58819955,  0.32212378, -0.53847136,  0.08145294,  0.67445032,
         0.79645481, -0.03563814, -0.61212155, -0.20453362,  0.00901039,
        -0.46704635,  0.52748719, -0.88462477,  0.0997008 , -0.03059553,
        -0.22769248, -0.78329203,  0.50340322,  0.84317135,  0.05938958,
         0.03647718,  0.36094766, -0.23812012,  0.23962454,  0.07095457,
         0.07486947,  0.50863129,  0.35297264,  0.23074683, -0.43016828,
         0.12093718,  0.24272389,  0.41042191, -0.0780606 , -0.24160862,
        -0.29825919,  0.45832736, -0.00979008,  0.10387476, -0.35432108,
        -0.78240953,  0.17926363,  0.64759755, -0.53630346,  0.287089  ,
         0.40308982, -0.04894509,  0.22170548, -0.67142313, -0.48433367]),
 array([-0.76338243,  0.02908873, -0.8991248 ,  1.05484207, -0.55022031,
         0.48600439, -0.18565574, -0.1228809 , -1.76772659,  0.30488618,
        -1.2318429 ,  0.71288802, -1.28376903, -0.03728773,  0.46580976,
         0.13030297, -0.11150959,  0.37264401,  0.12286358,  0.88980926,
         0.34630927,  0.10457614, -1.57965271, -0.56365789,  0.46365616,
         0.18111201,  0.89244332,  0.33778106,  0.52735652,  0.04958071,
         0.39926093,  0.49062171,  0.73644859,  0.57689954, -1.1595173 ,
        -0.9999871 ,  0.31091234,  0.25754134, -0.88879756,  0.17918183,
        -1.00514583, -0.83113491,  0.85206007, -0.36514009, -1.01925205,
         1.0960684 , -0.77600709,  0.03978135,  0.1559589 , -0.57104523]),
```

Fig. 15. TFIDF word2vec



```
X = df_sent2Vec.drop(["Unnamed: 0","Id","ProductId","UserId","ProfileName","HelpfulnessNumerator","HelpfulnessDenominator","Ti
```

```
X.head()
```

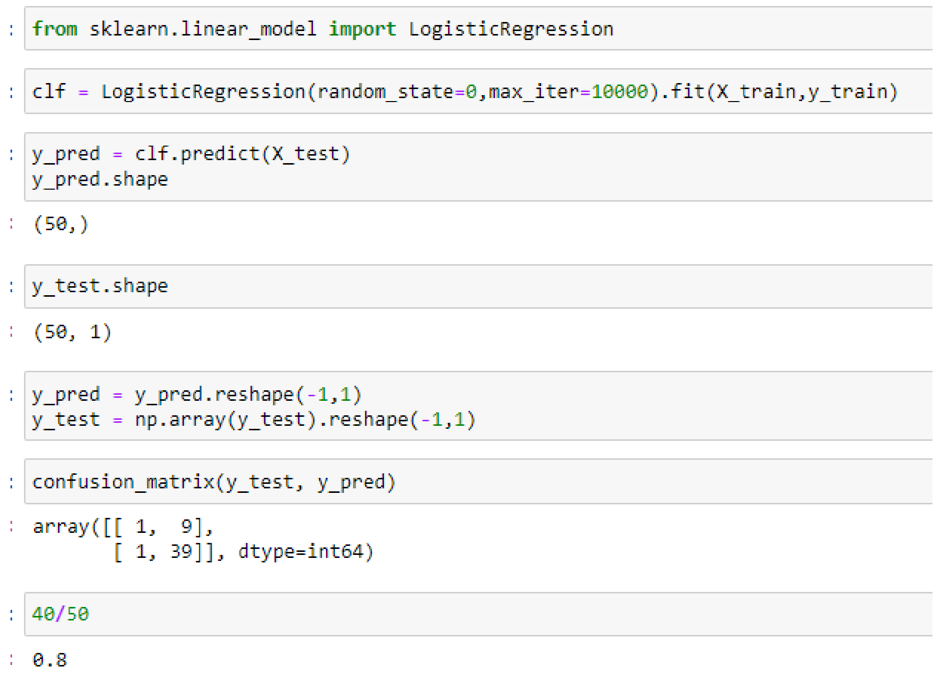
	0	1	2	3	4	5	6	7	8	9	...	40	41	42	43
0	0.588200	0.322124	-0.538471	0.081453	0.674450	0.796455	-0.035638	-0.612122	-0.204534	0.009010	...	-0.782410	0.179264	0.647598	-0.536303
1	-0.763382	0.029089	-0.899125	1.054842	-0.550220	0.486004	-0.185656	-0.122881	-1.767727	0.304886	...	-1.005146	-0.831135	0.852060	-0.365140
2	-0.224117	0.741820	-0.555298	0.230184	-0.214360	0.205023	-0.518163	-0.157328	-1.277998	-0.204960	...	-0.744679	-0.462751	0.762627	-0.665582
3	-0.026295	0.404695	-0.455078	-0.167410	0.637930	0.116415	-0.224196	-0.341630	-0.852680	-0.164182	...	0.069917	0.192627	0.289097	-0.438832
4	-0.187803	0.230454	-0.781271	-0.611435	0.147687	0.008649	0.020957	-0.306960	-1.252965	-0.178395	...	0.002509	-0.099832	0.860200	-0.527537

5 rows × 50 columns

Fig. 16. Reviews to Vectors of length 50

8. Training ML algorithm using vectors

Initially we trained our own text Corpus using the Dataset and Gensim models and able get vector of each word.Next, we converted Review(Sentence) into vector by using TFIDF w2v.Now we have New Dataset with review as a vector of length 50. So,by making use of any Classification algorithms (like Logistic Regression , GDA , Navie Bayes) we can train the Model. Finally by making use of this model we can predict whether the given Review is Positive or Negative.



```
from sklearn.linear_model import LogisticRegression
```

```
clf = LogisticRegression(random_state=0,max_iter=10000).fit(X_train,y_train)
```

```
y_pred = clf.predict(X_test)
y_pred.shape
```

```
(50,)
```

```
y_test.shape
```

```
(50, 1)
```

```
y_pred = y_pred.reshape(-1,1)
y_test = np.array(y_test).reshape(-1,1)
```

```
confusion_matrix(y_test, y_pred)
```

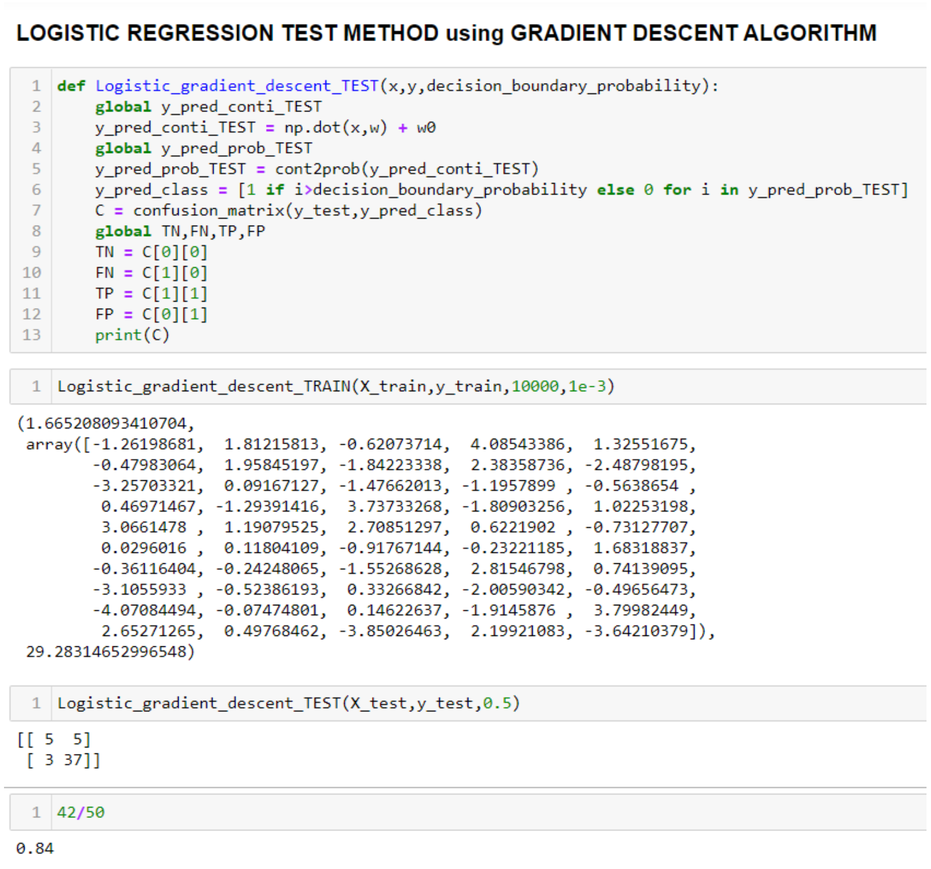
```
array([[ 1,  9],
       [ 1, 39]], dtype=int64)
```

```
40/50
```

```
0.8
```

Fig. 17. Result

The Logistic Regression Algorithm Coded from Scratch gives an Accuracy of 84.0 % with Learning rate = 10^{-3} and iterations = 10000



LOGISTIC REGRESSION TEST METHOD using GRADIENT DESCENT ALGORITHM

```
def Logistic_gradient_descent_TEST(x,y,decision_boundary_probability):
    global y_pred_conti_TEST
    y_pred_conti_TEST = np.dot(x,w) + w0
    global y_pred_prob_TEST
    y_pred_prob_TEST = cont2prob(y_pred_conti_TEST)
    y_pred_class = [1 if i>decision_boundary_probability else 0 for i in y_pred_prob_TEST]
    C = confusion_matrix(y_test,y_pred_class)
    global TN,FN,TP,FP
    TN = C[0][0]
    FN = C[1][0]
    TP = C[1][1]
    FP = C[0][1]
    print(C)
```

```
Logistic_gradient_descent_TRAIN(X_train,y_train,10000,1e-3)
```

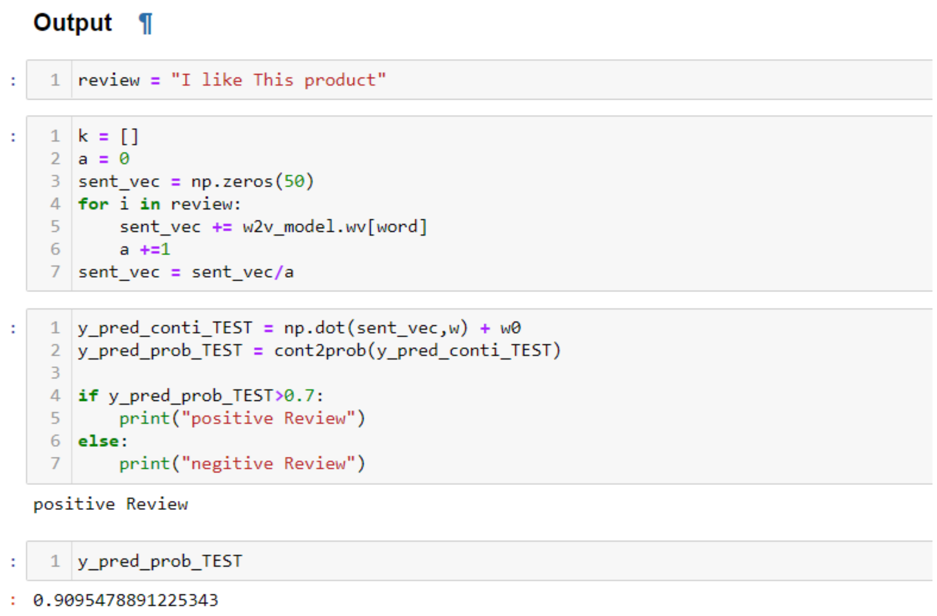
```
(1.665208093410704,
 array([-1.26198681,  1.81215813, -0.62073714,  4.08543386,  1.32551675,
        -0.47983064,  1.95845197, -1.84223338,  2.38358736, -2.48798195,
        -3.25703321,  0.09167127, -1.47662013, -1.1957899 , -0.5638654 ,
         0.46971467, -1.29391416,  3.73733268, -1.80903256,  1.02253198,
         3.0661478 ,  1.19079525,  2.70851297,  0.6221902 , -0.73127707,
         0.0296016 ,  0.11804109, -0.91767144, -0.23221185,  1.68318837,
        -0.36116404, -0.24248065, -1.55268628,  2.81546798,  0.74139095,
        -3.1055933 , -0.52386193,  0.33266842, -2.00590342, -0.49656473,
        -4.07084494, -0.07474801,  0.14622637, -1.9145876 ,  3.79982449,
         2.65271265,  0.49768462, -3.85026463,  2.19921083, -3.64210379]),
 29.28314652996548)
```

```
Logistic_gradient_descent_TEST(X_test,y_test,0.5)
```

```
[[ 5  5]
 [ 3 37]]
```

```
42/50
```

```
0.84
```



Output ¶

```
review = "I like This product"
```

```
k = []
a = 0
sent_vec = np.zeros(50)
for i in review:
    sent_vec += w2v_model.wv[word]
    a +=1
sent_vec = sent_vec/a
```

```
y_pred_conti_TEST = np.dot(sent_vec,w) + w0
y_pred_prob_TEST = cont2prob(y_pred_conti_TEST)

if y_pred_prob_TEST>0.7:
    print("positive Review")
else:
    print("negitive Review")
```

```
positive Review
```

```
y_pred_prob_TEST
```

```
0.9095478891225343
```

9. Code for the Project

https://github.com/BharadwajEdera/Bharadwaj-Machine-Learning-and-computing/blob/main/Amazon%20Fine%20Food%20Review%20analysis/NLP%20Mini%20Project.ipynb

References

- Thakkar, Kartikay, et al. ”Sentimental Analysis on Amazon Fine Food Reviews.” (2020).
- Mustaqim, Tanzilal, and Aprilia Dewi Ardiyanti. ”Sentiment Prediction Accuracy of Amazon Fine Food Review using TFIDF and LightGBM models.” Proceeding International Conference on Science and Engineering. Vol. 4. 2021.
- Sharedalal, Rutvik. ”AMAZON FINE FOOD REVIEWS-DESIGN AND IMPLEMENTATION OF AN AUTOMATED CLASSIFICATION SYSTEM.” (2019).
- https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/
- https://datascienceplus.com/scikit-learn-for-text-analysis-of-amazon-fine-food-reviews/