

## ▼ LOGISTIC REGRESSION ALGORITHM on Amazon Fine Food Reviews

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 - unique 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).

## ▼ Loading,Cleaning & Preprocessing the 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

3. If the score is above 3, then the recommendation will be set to "positive". Otherwise, it will be set to "negative". *italicized text*

```
1 %matplotlib inline
2 import warnings
3
4 warnings.filterwarnings("ignore")
```

## ▼ Import all Required Libraries

```
1 import sqlite3
2 import pandas as pd
3 import numpy as np
4 import nltk
5 import string
6 import matplotlib.pyplot as plt
7 import seaborn as sns
8 from sklearn.feature_extraction.text import TfidfTransformer
9 from sklearn.feature_extraction.text import TfidfVectorizer
```

```

9 from sklearn.feature_extraction.text import TfidfVectorizer
10 from sklearn.feature_extraction.text import CountVectorizer
11 from sklearn.metrics import confusion_matrix
12 from sklearn import metrics
13 from sklearn.metrics import roc_curve, auc
14 from nltk.stem.porter import PorterStemmer
15
16 import re
17 import string
18 from nltk.corpus import stopwords
19 from nltk.stem import PorterStemmer
20 from nltk.stem.wordnet import WordNetLemmatizer
21
22 from gensim.models import Word2Vec
23 from gensim.models import KeyedVectors
24 import pickle
25
26 from tqdm import tqdm
27 import os

```

### ▼ Pull the dataset from Google Drive & mount

```

1 from google.colab import drive
2 drive.mount('/content/drive')

```

Go to this URL in a browser: [https://accounts.google.com/o/oauth2/auth?client\\_id=947318989803-6bn6qk8qdgf4n4g3pf](https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pf)

Enter your authorization code:

.....

Mounted at /content/drive

### ▼ Connect to sqlite and fetch the data using SQL Query

```

1 con = sqlite3.connect("/content/drive/My Drive/Colab Notebooks/database.sqlite")
2
3 filtered_data=pd.read_sql_query("""SELECT * FROM Reviews WHERE Score != 3""",con);
4 filtered_data.head(3)

```

```

↳

```

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

```
1 filtered_data.shape
```

```
↳ (525814, 10)
```

```

1 def partition(x):
2     if x < 3 :
3         return 'negative'
4     return 'positive'
5
6 actualScore=filtered_data['Score']
7 positive_negative=actualScore.map(partition)
8 filtered_data['Score']=positive_negative
9 print("Number of datapoints", filtered_data.shape)

```

```
1 print( number of datapoints , filtered_data.shape,  
10 filtered_data.head(3))
```

Number of datapoints (525814, 10)

		<b>Id</b>	<b>ProductId</b>	<b>UserId</b>	<b>ProfileName</b>	<b>HelpfulnessNumerator</b>	<b>HelpfulnessDenominator</b>	<b>Score</b>
0	1	B001E4KFG0	A3SGXH7AUHU8GW		delmartian	1	1	posi
1	2	B00813GRG4	A1D87F6ZCVE5NK		dll pa	0	0	nega
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres	"Natalia Corres"	1	1	posi

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

```
1 print(display.shape)  
2 display.head(3)
```

(80668, 7)

		<b>UserId</b>	<b>ProductId</b>	<b>ProfileName</b>	<b>Time</b>	<b>Score</b>
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when considering t
1	#oc-R11D9D7SHXIJB9	B005HG9ET0	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle sp
2	#oc-R11DNU2NBKQ23Z	B007Y59HVM	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortun

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

	UserId	ProductId	ProfileName	Time	Score	
80638	AZY10LLTJ71NX	B006P7E5ZI	undertheshrine "undertheshrine"	1334707200	5	I was recommended to try green tea

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

393063

```
1 display= pd.read_sql_query("""  
2 SELECT *  
3 FROM Reviews  
4 WHERE Score != 3 AND UserId="AR5J8UI46CURR"  
5 ORDER BY ProductID  
6 """, con)  
7 display.head()
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5 1199
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5 1199
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5 1199
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5 1199
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5 1199

```

1 sorted_data=filtered_data.sort_values('ProductId',axis=0,ascending=True,inplace=False,kind='quicksort',na_positior
2
3 final_data=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"},keep='first',inplace=False)
4 final_data.shape

```

```

(364173, 10)

```

```

1 (final_data['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100

```

```

69.25890143662969

```

```

1 display= pd.read_sql_query("""
2 SELECT *
3 FROM Reviews
4 WHERE Score != 3 AND Id=44737 OR Id=64422
5 ORDER BY ProductID
6 """, con)
7
8 display.head()

```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5 1
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4 1

```

1 final_data=final_data[final_data.HelpfulnessNumerator<=final_data.HelpfulnessDenominator]

```

## ▼ Count the Positive and Negative Review Counts

```

1 print(final_data.shape)
2
3 #How many positive and negative reviews are present in our dataset?
4 final_data['Score'].value_counts()

```

```

(364171, 10)
positive    307061
negative    57110
Name: Score, dtype: int64

```

## ▼ Import nltk library

```
1 nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

## ▼ Find the Stopping Words

```
1 stopping_words = set(stopwords.words('english'))
2 print(stopping_words)
```

```
{'what', "hasn't", 'which', 'in', 'd', 'haven', 'y', 'with', 'after', 'his', 'other', 'don', 'll', "shouldn't",
```

```
1 def clean_html(text):
2     clean_r = re.compile('<.*?>')
3     clean_text = re.sub(clean_r, '', text)
4     return clean_text
5
6 def Clean_punc(text):
7     clean_sentence = re.sub(r'[?|!|\'|\"|#]', r' ', text)
8     clean_data = re.sub(r'[,|,|)(|\\|/)]', r' ', clean_sentence)
9     return clean_data
```

```
1 from tqdm import tqdm
2 import os
3 import pdb
4 import pickle
5
6 from tqdm import tqdm
7 import os
8 import pdb
9 import pickle
10
11 stem_no = nltk.stem.SnowballStemmer('english')
12
13 if not os.path.isfile('final_data.sqlite'):
14     final_string=[]
15     all_positive_words=[]
16     all_negative_words=[]
17     for i,sentence in enumerate(tqdm(final_data['Text'].values)):
18         filtered_sentence=[]
19         sent_without_html_tags=clean_html(sentence)
20         #pdb.set_trace()
21         for w in sent_without_html_tags.split():
22             for cleaned_words in Clean_punc(w).split():
23                 if ((cleaned_words.isalpha()) & (len(cleaned_words) > 2)):
24                     if(cleaned_words.lower() not in stopping_words):
25                         stemming=(stem_no.stem(cleaned_words.lower())).encode('utf8')
26                         filtered_sentence.append(stemming)
27                         if(final_data['Score'].values[i]=='positive':
28                             all_positive_words.append(stemming)
29                         if(final_data['Score'].values[i]=='negative':
30                             all_negative_words.append(stemming)
31         str1 = b" ".join(filtered_sentence)
32         final_string.append(str1)
33
34     final_data['Cleaned_text']=final_string
```


```
35 final_data['Cleaned_text']=final_data['Cleaned_text'].str.decode("utf-8")
36
37 conn = sqlite3.connect('final_data.sqlite')
38 cursor=conn.cursor()
39 conn.text_factory = str
40 final_data.to_sql('Reviews',conn,schema=None,if_exists='replace',index=True,index_label=None,chunksize=None,dtypes=None)
41 conn.close()
42
43
44 with open('positive_words.pkl','wb') as f :
45     pickle.dump(all_positive_words,f)
46 with open('negative_words.pkl','wb') as f :
47     pickle.dump(all_negative_words,f)
```

100%|██████████| 364171/364171 [06:15<00:00, 970.86it/s]

## ▼ Find the Total Number of Words in the Review Text

```
1 final_data['total_words'] = [len(x.split()) for x in final_data['Cleaned_text'].tolist()]
```

```
1 final_data.head(3)
```



	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0	positive
138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	1	positive
138689	150507	0006641040	A1S4A3IQ2MU7V4	sally sue "sally sue"	1	1	positive

```
1 final_data.sort_values(by=['Time'], inplace=True, ascending=True)
```

```
1 final_data.head(3)
```

↗

	<b>Id</b>	<b>ProductId</b>	<b>UserId</b>	<b>ProfileName</b>	<b>HelpfulnessNumerator</b>	<b>HelpfulnessDenominator</b>	<b>Score</b>
<b>138706</b>	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0	positive
<b>138683</b>	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	2	positive
<b>417839</b>	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	0	positive

```
1 final_data.shape
```

```
(364171, 12)
```

```
1 final_data_100K=final_data[0:100000]
2 amazon_polarity_labels=final_data_100K['Score'].values
3 final_data_100K.head(2)
```

```
(364171, 12)
```

	<b>Id</b>	<b>ProductId</b>	<b>UserId</b>	<b>ProfileName</b>	<b>HelpfulnessNumerator</b>	<b>HelpfulnessDenominator</b>	<b>Score</b>
<b>138706</b>	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0	positive
<b>138683</b>	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	2	positive

### ▼ Split the data into Train , Test and CV

```
1 from sklearn.model_selection import train_test_split
2 from sklearn.neighbors import KNeighborsClassifier
3 from sklearn.metrics import accuracy_score
4 from sklearn.model_selection import cross_val_score
5 from collections import Counter
6 from sklearn.metrics import confusion_matrix
7 from sklearn.metrics import classification_report
8
9 X_1,X_Test,Y_1,Y_Test = train_test_split(final_data_100K,amazon_polarity_labels,test_size=0.2,random_state=0)
10 X_Train,X_CV,Y_Train,Y_CV = train_test_split(X_1,Y_1,test_size=0.2)
```

```
1 print(X_Train.shape, Y_Train.shape)
2 print(X_CV.shape, Y_CV.shape)
3 print(X_Test.shape, Y_Test.shape)
4
5 print("="*100)
```

```

6
7
8 count_vector=CountVectorizer(min_df=1)
9 X_Train_data_bow=(count_vector.fit_transform(X_Train['Cleaned_text'].values))
10 X_Test_data_bow=(count_vector.transform(X_Test['Cleaned_text'].values))
11 X_CV_data_bow=(count_vector.transform(X_CV['Cleaned_text'].values))
12
13 print("After vectorizations")
14 print(X_Train_data_bow.shape, Y_Train.shape)
15 print(X_CV_data_bow.shape, Y_CV.shape)
16 print(X_Test_data_bow.shape, Y_Test.shape)
17 print(""*100)

```

```

(64000, 12) (64000,)
(16000, 12) (16000,)
(20000, 12) (20000,)
=====
After vectorizations
(64000, 29469) (64000,)
(16000, 29469) (16000,)
(20000, 29469) (20000,)
=====

```

## ▼ Use Logistic Regression Algorithm and GridSearchCV using 10 Fold Cross Validation to find Best Alpha

```

1 from sklearn.model_selection import GridSearchCV
2 from scipy.stats import randint as sp_randint
3 from sklearn.model_selection import cross_val_score
4 from sklearn.linear_model import LogisticRegression
5
6 def Logistic_Regression_Optimal_C(x_training_data,y_training_data):
7     grid_params = { 'C' : [10**x for x in range(-5,4)]
8                     }
9     logistic_regression = LogisticRegression(random_state=None, class_weight=None)
10    clf=GridSearchCV(logistic_regression,grid_params,scoring='roc_auc',return_train_score=True,cv=10)
11    clf.fit(x_training_data,y_training_data)
12    results = pd.DataFrame.from_dict(clf.cv_results_)
13    results = results.sort_values(['param_C'])
14    train_auc= results['mean_train_score']
15    train_auc_std= results['std_train_score']
16    cv_auc = results['mean_test_score']
17    cv_auc_std= results['std_test_score']
18    C = results['param_C']
19    #print(type(alpha))
20    #print(alpha)
21    log_c=np.log10(list(results["param_C"]))
22    print(clf.best_score_)
23    print(clf.best_params_)
24    plt.plot(log_c, train_auc, label='Train AUC')
25    plt.plot(log_c, cv_auc, label='CV AUC')
26    plt.scatter(log_c, train_auc, label='Train AUC points')
27    plt.scatter(log_c, cv_auc, label='CV AUC points')
28    plt.legend()
29    plt.xlabel("C: hyperparameter")
30    plt.ylabel("AUC")
31    plt.title("Hyper parameter Vs AUC plot")
32    plt.grid()
33    plt.show()
34    return results,clf,logistic_regression

```

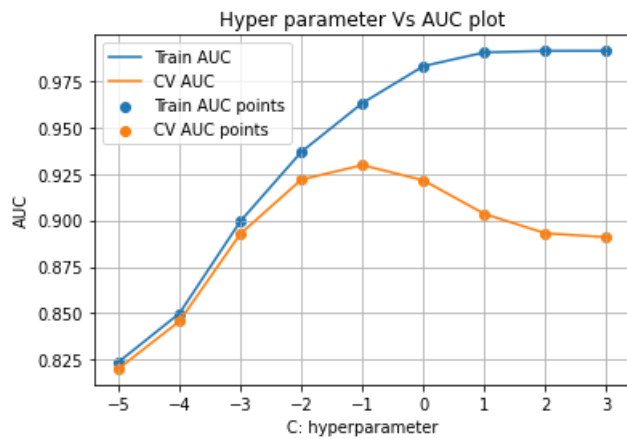
```

1 results,best_c,Logistic_Regression_Optimal_C = Logistic_Regression_Optimal_C(X_Train_data_bow,Y_Train)
2 results.head()

```



```
0.9296777946680457
{'C': 0.1}
```



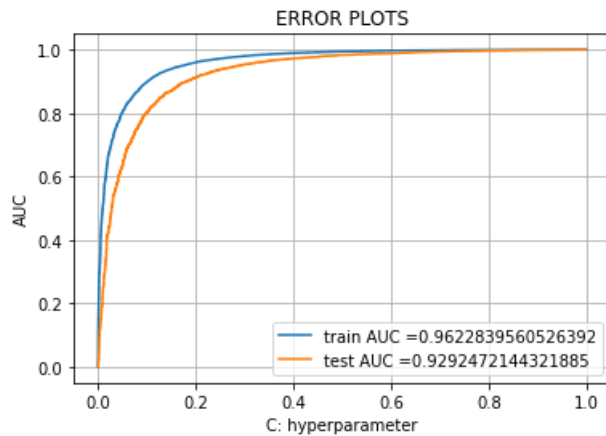
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split0_test_score	split1_test
0	0.429887	0.007426	0.028884	0.000816	1e-05	{'C': 1e-05}	0.816206	0
1	0.412549	0.011324	0.028783	0.000618	0.0001	{'C': 0.0001}	0.840353	0
2	0.543292	0.012814	0.028957	0.000383	0.001	{'C': 0.001}	0.886338	0
3	1.095692	0.052711	0.028767	0.000523	0.01	{'C': 0.01}	0.921654	0
4	2.264165	0.028399	0.028764	0.000349	0.1	{'C': 0.1}	0.934950	0

## ▼ Using Logistic Regression on Bag OF Words , The Optimal Value of C = 0.1 and Accuracy

```
1 best_C = best_c.best_params_
2 best_c=best_C.get("C")
3 print(best_c)
```

```
0.1
```

```
1 from sklearn.metrics import roc_curve, auc
2
3 logistic_reg = LogisticRegression(C=best_c,random_state=None, class_weight=None)
4 logistic_reg.fit(X_Train_data_bow,Y_Train)
5 pred_test_data=logistic_reg.predict(X_Test_data_bow)
6 y_train_predicted_prob = logistic_reg.predict_proba(X_Train_data_bow)[:,-1]
7 y_test_predicted_prob=logistic_reg.predict_proba(X_Test_data_bow)[:,-1]
8 train_fpr, train_tpr, train_thresholds=roc_curve(Y_Train,y_train_predicted_prob,pos_label='positive')
9 test_fpr, test_tpr, test_thresholds = roc_curve(Y_Test, y_test_predicted_prob,pos_label='positive')
10 plt.plot(train_fpr, train_tpr, label="train AUC =" +str(auc(train_fpr, train_tpr)))
11 plt.plot(test_fpr, test_tpr, label="test AUC =" +str(auc(test_fpr, test_tpr)))
12 plt.legend()
13 plt.xlabel("C: hyperparameter")
14 plt.ylabel("AUC")
15 plt.title("ERROR PLOTS")
16 plt.grid()
17 plt.show()
```



BY LOOKING AT THE GRAPH , THE TRAIN AUC = 0.9614.. AND TEST AUC = 0.9300..

#### ▼ FIND THE TOP 20 POSITIVE AND NEGATIVE FEATURES

```
1 feature_names = count_vector.get_feature_names()
2 print(feature_names)
3 coefs_with_fns = sorted(zip(logistic_reg.coef_[0], feature_names))
4 top_features = zip(coefs_with_fns[:20], coefs_with_fns[:-(20 + 1):-1])
5 list(top_features)
```

```
[('aaa', 'aaaaaaaaagghh', 'aaaaah', 'aaaaahhhhhhhhhhhhhhh', 'aaaah', 'aaah', 'aaahhhhhh', 'aachen', 'aacur', 'aach', 'worst', 'delici'),
((-1.57737780496826, 'terribl'), (1.3382979566189583, 'excel')),
((-1.5235684925772264, 'disappoint'), (1.2743398099781318, 'perfect')),
((-1.5205848303736005, 'bland'), (1.2722028214899557, 'best')),
((-1.4750041539836685, 'horribl'), (1.1706319430190184, 'great')),
((-1.3409612561369968, 'return'), (1.1029968456002295, 'amaz')),
((-1.3219831368478137, 'threw'), (1.0949674364031627, 'addict')),
((-1.3098381626032183, 'aw'), (1.0142716547466408, 'yummi')),
((-1.2412542391072634, 'tasteless'), (0.9649815643441443, 'beat')),
((-1.15974152981675, 'unfortun'), (0.9628143526724355, 'hook')),
((-1.1319660226677657, 'stale'), (0.9595063431094906, 'awesom')),
((-1.1077658603574856, 'sorri'), (0.9497114203720983, 'satisfi')),
((-1.0308874481764474, 'money'), (0.9268681019088533, 'wonder')),
((-1.025668735420536, 'ruin'), (0.9245726512896841, 'nice')),
((-0.9968855390231548, 'unpleas'), (0.8790290626229103, 'thank')),
((-0.9507004340734386, 'disgust'), (0.87285749292844, 'refresh')),
((-0.9506042394000396, 'wors'), (0.8707387895436062, 'uniqu')),
((-0.9341227475388825, 'poor'), (0.8696152132731848, 'smooth')),
((-0.9268692049162756, 'cancel'), (0.8631700227755539, 'alway')),
((-0.9210385646173204, 'gross'), (0.8136641532234451, 'favorit'))]
```

#### ▼ ROC AUC SCORE = 0.9300..

```
1 from sklearn.metrics import roc_auc_score
2
3 roc_auc_score(Y_Test,y_test_predicted_prob)
```

```
0.9292472144321885
```

```
1 from sklearn.metrics import classification_report,confusion_matrix
2
3 print(classification_report(Y_Test,y_test_predicted_prob))
```

```
3 print(classification_report(y_test,pred_test_data))
4 print(confusion_matrix(Y_Test,pred_test_data))
```

```

└─ precision    recall  f1-score   support

   negative     0.79      0.52      0.63      2457
   positive     0.94      0.98      0.96     17543

   accuracy                   0.92     20000
  macro avg       0.87      0.75      0.79     20000
 weighted avg     0.92      0.92      0.92     20000

[[ 1285  1172]
 [  333 17210]]
```

## ▼ Find the Weight W for the Model

```
1 weight_w1=logistic_reg.coef_
```

```
1 weight_w1
```

```
└─ array([[ -9.54484428e-03,  1.34033485e-03,  9.57209191e-03, ...,
          -4.18750044e-02,  3.49458247e-05,  3.24138399e-04]])
```

```
1 print(weight_w1.shape)
```

```
└─ (1, 29469)
```

## ▼ Adding Small Noise using random function

```
1 import copy
2 noise_data = copy.deepcopy(X_Train_data_bow)
```

```
1 from scipy.sparse import csr_matrix
2 noise_data=csr_matrix(noise_data,dtype=np.float64)
```

```
1 type(noise_data)
```

```
└─ scipy.sparse.csr.csr_matrix
```

```
1 add_noise=np.random.normal(0,0.01)
```

```
1 print(add_noise)
```

```
└─ 0.011160311560311106
```

```
1 type(add_noise)
```

```
└─ float
```

## ▼ Adding the Noise to BoW Trainig Data

```
1 noise_data.data += add_noise
2 print(noise_data.shape)
```

```
└─ (64000, 29469)
```

## ▼ Fit the Model on NOISE DATA and get the Weight W'

```
1 logistic_reg_new = LogisticRegression(C=best_c,random_state=None, class_weight=None)
2 logistic_reg_new.fit(noise_data,Y_Train)
3 weight_w2=logistic_reg_new.coef_
```

```
1 weight_w2.shape
```

```
↳ (1, 29469)
```

## ▼ CALCULATE THE AVERAGE DIFFERENCE IN WEIGHT VECTORS

```
1 coef_diff = weight_w2 - weight_w1
2 print ("Average difference in weight vectors : ",np.mean(coef_diff))
```

```
↳ Average difference in weight vectors : 1.5990948651124464e-06
```

## ▼ Add the small eps value to avoid the divide by zer error

$$W1' = W1 + 10^{-6}$$

$$W2' = W2 + 10^{-6}$$

```
1 weight_w1_dash = weight_w1 + 0.000001
2 weight_w2_dash = weight_w2 + 0.000001
```

## ▼ Find the Percentage Change between W1 & W2 using $(|(W1-W2) / (W1)|) * 100$

```
1 percentage_difference = (abs((weight_w1_dash- weight_w2_dash)/weight_w1_dash)) * 100
```

```
1 sorted_array=np.sort(percentage_difference)
2
3 reverse_array=sorted_array[::-1]
```

```
1 type(percentage_difference)
```

```
↳ numpy.ndarray
```

```
1 percentage_difference.shape
```

```
↳ (1, 29469)
```

## ▼ FIND THE NTH PERCENTILE

```
1 print('The First 100th Percentile')
2
3 for i in range(100):
4     print(" The Percentile = ",i,np.percentile(percentage_difference,i))
```

```
↳
```

The First 100th Percentile

```
The Percentile = 0 0.00012988504120639484
The Percentile = 1 0.01704106652529466
The Percentile = 2 0.031667209716234816
The Percentile = 3 0.04680951695573926
The Percentile = 4 0.06296934141459282
The Percentile = 5 0.07710280742676966
The Percentile = 6 0.09144412489112096
The Percentile = 7 0.10553738467497611
The Percentile = 8 0.12134112333411497
The Percentile = 9 0.13595506168880706
The Percentile = 10 0.15081270267368269
The Percentile = 11 0.16697508577206996
The Percentile = 12 0.18258076809661966
The Percentile = 13 0.19822051870137644
The Percentile = 14 0.21427523126538356
The Percentile = 15 0.22964289513366137
The Percentile = 16 0.2443782103723866
The Percentile = 17 0.26154048360674576
The Percentile = 18 0.27601476351142334
The Percentile = 19 0.2924510292565669
The Percentile = 20 0.3093105177933104
The Percentile = 21 0.32385186650583647
The Percentile = 22 0.33842006387510865
The Percentile = 23 0.3539653576772072
The Percentile = 24 0.36935937449626266
The Percentile = 25 0.38539926331194163
The Percentile = 26 0.400044166967558
The Percentile = 27 0.41514934737875137
The Percentile = 28 0.4304017617690613
The Percentile = 29 0.4448634669705822
The Percentile = 30 0.45959783426313466
The Percentile = 31 0.47621909483400354
The Percentile = 32 0.49080194651982306
The Percentile = 33 0.5075306621060133
The Percentile = 34 0.5235854034451344
The Percentile = 35 0.538902653489705
The Percentile = 36 0.5545227732983823
The Percentile = 37 0.5694745982128888
The Percentile = 38 0.5842990749517557
The Percentile = 39 0.5974049922299296
The Percentile = 40 0.6144258545130906
The Percentile = 41 0.6295019573548023
The Percentile = 42 0.6459883365194344
The Percentile = 43 0.6615943199804594
The Percentile = 44 0.6783678855075483
The Percentile = 45 0.6951289976148123
The Percentile = 46 0.7107863288136497
The Percentile = 47 0.7282140315256589
The Percentile = 48 0.7453277764271679
The Percentile = 49 0.7632029262751474
The Percentile = 50 0.7800833501172941
The Percentile = 51 0.7944572327544914
The Percentile = 52 0.8104452672524756
The Percentile = 53 0.8267020259940581
The Percentile = 54 0.8460909511594523
The Percentile = 55 0.8670759059456585
The Percentile = 56 0.8867652520131792
The Percentile = 57 0.9070800666739709
The Percentile = 58 0.9272390269407805
The Percentile = 59 0.9481353440124766
The Percentile = 60 0.9693808453167522
The Percentile = 61 0.988501926914398
The Percentile = 62 1.008777127887971
The Percentile = 63 1.0314838393616867
The Percentile = 64 1.0527566305018365
The Percentile = 65 1.076803774712028
The Percentile = 66 1.0997270884226287
The Percentile = 67 1.1248088224542439
The Percentile = 68 1.1497393568293308
The Percentile = 69 1.171559026820088
The Percentile = 70 1.1960885515105154
```

```

The Percentile = 70 1.12000033127377
The Percentile = 71 1.22004325721293
The Percentile = 72 1.245962475452935
The Percentile = 73 1.2785163342734733
The Percentile = 74 1.3129796397308124
The Percentile = 75 1.3503683540688045
The Percentile = 76 1.3906112977042164
The Percentile = 77 1.4308618930525472
The Percentile = 78 1.4778195295121621
The Percentile = 79 1.535333528759989
The Percentile = 80 1.589585366978879
The Percentile = 81 1.654389886294069
The Percentile = 82 1.7288723906914822
The Percentile = 83 1.802666241568125
The Percentile = 84 1.8932814305710686
The Percentile = 85 1.9917682086908624
The Percentile = 86 2.11877569719489
The Percentile = 87 2.2602520969381055
The Percentile = 88 2.426616826431457
The Percentile = 89 2.6254518784448084
The Percentile = 90 2.861125748158766
The Percentile = 91 3.1543233986897072
The Percentile = 92 3.574850501485807
The Percentile = 93 4.079814012828142
The Percentile = 94 4.5617997834832344
The Percentile = 95 5.204422022609738
The Percentile = 96 6.127016713603429
The Percentile = 97 7.838587197272339
The Percentile = 98 11.114065868273794
The Percentile = 99 26.473223295000746

```

▼ BY LOOKING AT THE PERCENTILE LOOKS LIKE THERE IS SHARP INCREASE IN THE PERCENTAGE CH

98TH = 11.11

99TH = 26.47

```

1 print(" The Percentile = ",np.percentile(percentage_difference,98.1))
2 print(" The Percentile = ",np.percentile(percentage_difference,98.2))
3 print(" The Percentile = ",np.percentile(percentage_difference,98.3))
4 print(" The Percentile = ",np.percentile(percentage_difference,98.4))
5 print(" The Percentile = ",np.percentile(percentage_difference,98.5))
6 print(" The Percentile = ",np.percentile(percentage_difference,98.6))
7 print(" The Percentile = ",np.percentile(percentage_difference,98.7))
8 print(" The Percentile = ",np.percentile(percentage_difference,98.8))
9 print(" The Percentile = ",np.percentile(percentage_difference,98.9))
10 print(" The Percentile = ",np.percentile(percentage_difference,99))

```

```

↗ The Percentile = 11.93693577866307
The Percentile = 12.991766230275505
The Percentile = 13.761437085315986
The Percentile = 14.921986156096573
The Percentile = 15.543394057304209
The Percentile = 16.88162252881774
The Percentile = 18.32623247587617
The Percentile = 21.27902948057568
The Percentile = 24.78234190054234
The Percentile = 26.473223295000746

```

LOOKS LIKE THERE IS INCREASE IN THE VALUE FOR 98.8TH, 98.9TH & 99TH PERCENTILE

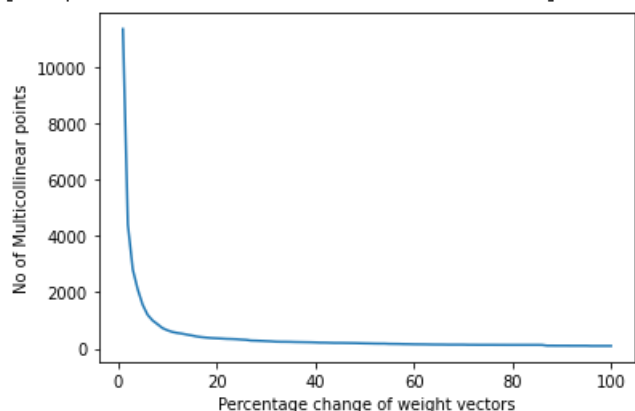
## ▼ PLOT THE PERCENTAGE CHANGE OF WEIGHT VECTOR WITH NO OF MULTICOLLINEAR POINTS

```

1 percentage_change = []
2 delta_i = []
3
4 for i in range(1,101,1):
5     f= np.where(reverse_array > i )[1].size
6     percentage_change.append(i)
7     delta_i.append(f)
8
9
10 plt.xlabel('Percentage change of weight vectors')
11 plt.ylabel('No of Multicollinear points')
12 plt.plot(percentage_change,delta_i)

```

↳ [matplotlib.lines.Line2D at 0x7fd9e3d1ef28]



1

## ▼ DISPLAY THE LIST OF MULTICOLLINEAR FEATURES

```

1 feat = count_vector.get_feature_names()
2 print("No of features have weight changes greater than 30%: ", percentage_difference[np.where(percentage_difference
3 feature_names=[]
4 print("\nHence below features are multi collinear:")
5 for i in np.where(percentage_difference > 1)[1]:
6     feature_names.append(feat[i])
7 print(feature_names)

```

↳ No of features have weight changes greater than 30%: 190

Hence below features are multi collinear:

['aaaah', 'aaah', 'aaahhhhhh', 'ab', 'aberddeen', 'abi', 'abita', 'abnorm', 'abovement', 'abruzzo', 'abscond', 'a

1

## ▼ FIND THE TOP 20 POSITIVE AND NEGATIVE FEATURES

```

1 feature_names = count_vector.get_feature_names()
2 coefs_with_fns = sorted(zip(logistic_reg_new.coef_[0], feature_names))
3 top_features = zip(coefs_with_fns[:20], coefs_with_fns[-(20 + 1):-1])
4 list(top_features)

```

```

[((-2.1897087580522707, 'worst'), (1.4799395787392793, 'delici')),
 ((-1.622620925667345, 'disappoint'), (1.4381802487889659, 'excel')),
 ((-1.4680016561065057, 'horribl'), (1.338581505468097, 'perfect')),
 ((-1.4536479455687588, 'terribl'), (1.2757553991400303, 'best')),
 ((-1.3420854783555103, 'aw'), (1.2052891739908673, 'great')),
 ((-1.324114576615227, 'bland'), (1.136149304462042, 'yummi')),
 ((-1.3144308941094085, 'return'), (1.0849946002814685, 'addict')),
 ((-1.266831934326799, 'threw'), (1.0585243927196422, 'amaz')),
 ((-1.1413144677765867, 'unfortun'), (1.0050624821153975, 'awesom')),
 ((-1.124400385988919, 'stale'), (0.9842678362137236, 'refresh')),
 ((-1.0949639099000745, 'yuck'), (0.9643267000433828, 'beat')),
 ((-1.086558885923567, 'tasteless'), (0.9237078205821216, 'nice')),
 ((-1.0851015614217796, 'poor'), (0.9139233994354666, 'wonder')),
 ((-1.0567803708722718, 'dissapoint'), (0.8976226027764952, 'glad')),
 ((-1.031581732109198, 'unpleas'), (0.8933028806429271, 'smooth')),
 ((-1.029185841192754, 'weak'), (0.8929004659141696, 'hook')),
 ((-1.009397479392787, 'disgust'), (0.8548719395738693, 'favorit')),
 ((-0.9879711261291, 'money'), (0.8440101397384687, 'uniqu')),
 ((-0.9877179182942307, 'sorri'), (0.8206886668205972, 'satisfi')),
 ((-0.9655446083447002, 'stuck'), (0.8099747518424147, 'happi'))]

```

```

1 def imp_features(model,classifier):
2     voc = model.get_feature_names()
3     w = list(classifier.coef_[0])
4     pos_coef = []
5     neg_coef = []
6     pos_words = []
7     neg_words = []
8     for i,c in enumerate(w):
9         if c > 0:
10             pos_coef.append(c)
11             pos_words.append(voc[i])
12         if c < 0:
13             neg_coef.append(abs(c))
14             neg_words.append(voc[i])
15     pos_df = pd.DataFrame(columns = ['Words','Coef'])
16     neg_df = pd.DataFrame(columns = ['Words','Coef'])
17     pos_df['Words'] = pos_words
18     pos_df['Coef'] = pos_coef
19     neg_df['Words'] = neg_words
20     neg_df['Coef'] = neg_coef
21     pos_df = pos_df.sort_values("Coef",axis = 0,ascending = False).reset_index(drop=True)
22     neg_df = neg_df.sort_values("Coef",axis = 0,ascending = False).reset_index(drop=True)
23     print("Shape of Positive dataframe:- ",pos_df.shape)
24     print("Shape of Negative dataframe:- ",neg_df.shape)
25     print("Top ten positive predictors:- \n",pos_df.head(10))
26     print("\nTop ten negative predictors:- \n",neg_df.head(10))

```

▼ THE POSTIVE POINT = (21767,2)

THE NEGATIVE POINT = (7626,2)

```
1 imp_features(count_vector,logistic_reg_new)
```

↗



Shape of Positive dataframe:- , (21767, 2)

Shape of Negative dataframe:- (7626, 2)

Top ten positive predictors:-

	Words	Coef
0	delici	1.479940
1	excel	1.438180
2	perfect	1.338582
3	best	1.275755
4	great	1.205289
5	yummi	1.136149
6	addict	1.084995
7	amaz	1.058524
8	awesom	1.005062
9	refresh	0.984268

Top ten negative predictors:-

	Words	Coef
0	worst	2.189709
1	disappoint	1.622621
2	horribl	1.468002
3	terribl	1.453648
4	aw	1.342085
5	bland	1.324115
6	return	1.314431
7	threw	1.266832
8	unfortun	1.141314
9	stale	1.124400

## ▼ SPARSITY OF WEIGHT VECTOR ON BAG OF WORDS VECTORIZATION TECHNIQUE

```
1 logistic_reg_l1 = LogisticRegression(C=best_c,random_state=None, class_weight=None,penalty='l1',solver='liblinear')
2 logistic_reg_l1.fit(X_Train_data_bow,Y_Train)
3 weight_w1_l1=logistic_reg_l1.coef_
```

```
1 imp_features(count_vector,logistic_reg_l1)
```

↳ Shape of Positive dataframe:- , (382, 2)

Shape of Negative dataframe:- (373, 2)

Top ten positive predictors:-

	Words	Coef
0	delici	1.576014
1	excel	1.445582
2	perfect	1.383116
3	best	1.229966
4	yummi	1.215222
5	great	1.168915
6	amaz	1.160982
7	addict	1.103060
8	awesom	1.036212
9	refresh	1.012890

Top ten negative predictors:-

	Words	Coef
0	worst	2.644614
1	disappoint	1.603224
2	horribl	1.597010
3	terribl	1.553291
4	aw	1.440146
5	threw	1.365178
6	yuck	1.350889
7	return	1.318721
8	bland	1.274205
9	unfortun	1.198295

```
1 logistic_reg_new_l1 = LogisticRegression(C=best_c, random_state=None, class_weight=None, penalty='l1', solver='liblinear')
2 logistic_reg_new_l1.fit(noise_data, Y_Train)
3 weight_w2=logistic_reg_new_l1.coef_
```

## ▼ AFTER SPARSITY , THE NUMBER OF ZEROS REMOVED

THE POSITIVE DATAPOINT = (382,2)

AND THE NEGATIVE DATAPOINT = (372,2)

```
1 imp_features(count_vector, logistic_reg_new_l1)
```

```
↳ Shape of Positive dataframe:- , (382, 2)
```

```
Shape of Negative dataframe:- (372, 2)
```

```
Top ten positive predictors:-
```

	Words	Coef
0	delici	1.585864
1	excel	1.454302
2	perfect	1.391040
3	best	1.236530
4	yummi	1.221394
5	great	1.175115
6	amaz	1.166880
7	addict	1.107871
8	awesom	1.040926
9	refresh	1.015497

```
Top ten negative predictors:-
```

	Words	Coef
0	worst	2.658635
1	disappoint	1.613068
2	horribl	1.607020
3	terribl	1.562023
4	aw	1.448828
5	threw	1.373219
6	yuck	1.354739
7	return	1.325813
8	bland	1.280382
9	unfortun	1.205609

## ▼ TF-IDF Vectorization Technique on Logistic Regression Algorithm

```
1 from sklearn.feature_extraction.text import TfidfVectorizer
2
3 print(X_Train.shape, Y_Train.shape)
4 print(X_CV.shape, Y_CV.shape)
5 print(X_Test.shape, Y_Test.shape)
6
7 print("="*100)
8
9
10 tfidf_vector=TfidfVectorizer(min_df=10)
11 X_Train_data_tfidf=(tfidf_vector.fit_transform(X_Train['Cleaned_text']).values)
12 X_Test_data_tfidf=(tfidf_vector.transform(X_Test['Cleaned_text']).values)
13 X_CV_data_tfidf=(tfidf_vector.transform(X_CV['Cleaned_text']).values)
14
15 print("After vectorizations")
16 print(X_Train_data_tfidf.shape, Y_Train.shape)
17 print(X_CV_data_tfidf.shape, Y_CV.shape)
18 print(X_Test_data_tfidf.shape, Y_Test.shape)
19 print("="*100)
```

```

(64000, 12) (64000,)
(16000, 12) (16000,)
(20000, 12) (20000,)

```

After vectorizations

```

(64000, 6794) (64000,)
(16000, 6794) (16000,)
(20000, 6794) (20000,)

```

```

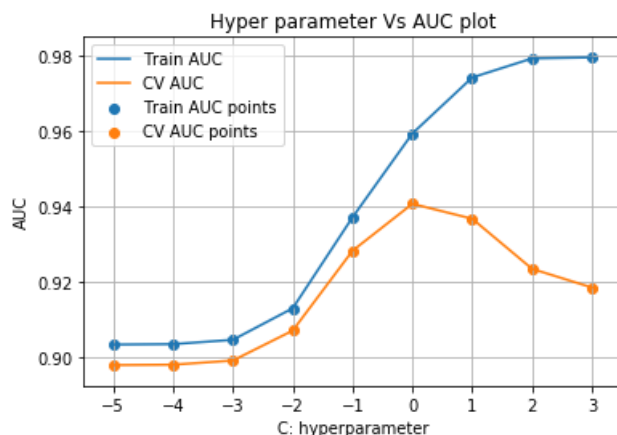
1 results,best_c,Logistic_Regression_Optimal_C = Logistic_Regression_Optimal_C(X_Train_data_tfidf,Y_Train)
2 results.head()

```

```

0.940703952138698
{'C': 1}

```



	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split0_test_score	split1_test_score
0	0.223514	0.006442	0.014594	0.000381	1e-05	{'C': 1e-05}	0.898880	0
1	0.182328	0.001241	0.014044	0.000183	0.0001	{'C': 0.0001}	0.898969	0
2	0.206218	0.002766	0.013947	0.000166	0.001	{'C': 0.001}	0.899903	0
3	0.281825	0.003033	0.014030	0.000241	0.01	{'C': 0.01}	0.906732	0
4	0.614663	0.043245	0.014329	0.000435	0.1	{'C': 0.1}	0.926466	0

```

1 best_C = best_c.best_params_
2 best_c=best_C.get("C")
3 print(best_c)

```

```
1
```

```

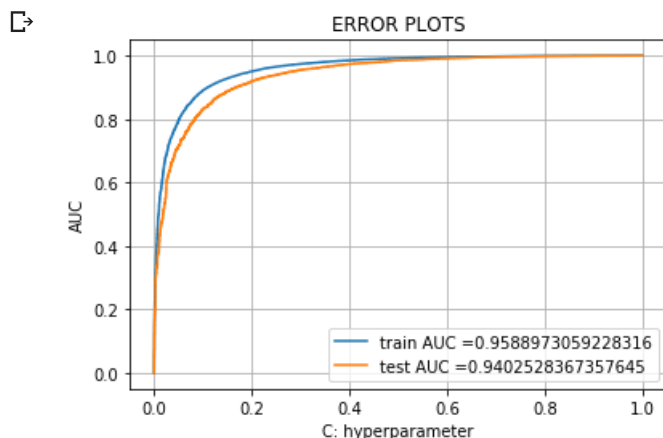
1 logistic_reg = LogisticRegression(C=best_c,random_state=None, class_weight=None)
2 logistic_reg.fit(X_Train_data_tfidf,Y_Train)
3 pred_test_data=logistic_reg.predict(X_Test_data_tfidf)
4 y_train_predicted_prob = logistic_reg.predict_proba(X_Train_data_tfidf)[:,-1]
5 y_test_predicted_prob=logistic_reg.predict_proba(X_Test_data_tfidf)[:,-1]
6 train_fpr, train_tpr, train_thresholds=roc_curve(Y_Train,y_train_predicted_prob,pos_label='positive')
7 test_fpr, test_tpr, test_thresholds = roc_curve(Y_Test, y_test_predicted_prob,pos_label='positive')
8 plt.plot(train_fpr, train_tpr, label="train AUC =" +str(auc(train_fpr, train_tpr)))
9 plt.plot(test_fpr, test_tpr, label="test AUC =" +str(auc(test_fpr, test_tpr)))
10 plt.legend()
11 plt.xlabel("C: hyperparameter")

```

```

12 plt.ylabel("AUC")
13 plt.title("ERROR PLOTS")
14 plt.grid()
15 plt.show()

```



```

1 feature_names = tfidf_vector.get_feature_names()
2 print(feature_names)
3 coefs_with_fns = sorted(zip(logistic_reg.coef_[0], feature_names))
4 top_features = zip(coefs_with_fns[:20], coefs_with_fns[:-(20 + 1):-1])
5 list(top_features)

```

```

[('abandon', 'abc', 'abdomin', 'abil', 'abl', 'abnorm', 'abroad', 'absenc', 'absolut', 'absorb', 'absorpt', 'absu
[(-7.748687485272045, 'disappoint'), (10.339448027879326, 'great')),
((-7.622673795053763, 'worst'), (8.486116395757557, 'best')),
((-5.9005000634955875, 'terribl'), (7.68466693659498, 'delici')),
((-5.579915691392207, 'horribl'), (7.3817152671250845, 'love')),
((-5.295137172971102, 'aw'), (6.762914343760864, 'perfect')),
((-5.040232601297126, 'bland'), (6.3577495135318225, 'excel')),
((-4.9987251867463165, 'return'), (5.1498910605369135, 'nice')),
((-4.8335094257523865, 'stale'), (5.148119141672898, 'favorit')),
((-4.77273337902024, 'threw'), (5.074128994392021, 'amaz')),
((-4.675868642865835, 'tasteless'), (5.068752716930844, 'good')),
((-4.530295948623052, 'unfortun'), (5.023669033728019, 'wonder')),
((-4.205660835924637, 'weak'), (4.240576065719856, 'addict')),
((-4.173129569188836, 'money'), (4.089643153159445, 'yummi')),
((-4.120068588911006, 'wast'), (3.8270174695946784, 'awesom')),
((-3.8335856171096605, 'wors'), (3.7711143383792556, 'alway')),
((-3.760560772946229, 'disgust'), (3.7388272339085047, 'tasti')),
((-3.64913321087239, 'bad'), (3.7093388092836515, 'beat')),
((-3.520109301365229, 'stuck'), (3.5431488028219014, 'find')),
((-3.512229111711405, 'unpleas'), (3.4367258483602066, 'smooth')),
((-3.501584488910644, 'sorri'), (3.4217965398833345, 'keep'))]

```

```
1 roc_auc_score(Y_Test,y_test_predicted_prob)
```

```
0.9402528367357645
```

```

1 print(classification_report(Y_Test,pred_test_data))
2 print(confusion_matrix(Y_Test,pred_test_data))

```

```


```

	precision	recall	f1-score	support
negative	0.83	0.50	0.62	2457
positive	0.93	0.99	0.96	17543
accuracy			0.93	20000
macro avg	0.88	0.74	0.79	20000
weighted avg	0.92	0.93	0.92	20000

```
[[ 1218 1239]
 [ 255 17288]]
```

## ▼ Avg Word2Vec Vectorization Technique on Logistic Regression Algorithm

```
1 from gensim.models import Word2Vec
2 from gensim.models import KeyedVectors
3 import pickle
4
5 list_of_sent_train_avgw2v=[]
6 list_of_sent_test_avgw2v=[]
7 list_of_sent_cv_avgw2v=[]
8 for sent_train_avgw2v in tqdm(X_Train['Cleaned_text'].values):
9     list_of_sent_train_avgw2v.append(sent_train_avgw2v.split())
```

100%|██████████| 64000/64000 [00:00<00:00, 101284.36it/s]

```
1 for sent_test_avgw2v in tqdm(X_Test['Cleaned_text'].values):
2     list_of_sent_test_avgw2v.append(sent_test_avgw2v.split())
3
4 for sent_cv_avgw2v in tqdm(X_CV['Cleaned_text'].values):
5     list_of_sent_cv_avgw2v.append(sent_cv_avgw2v.split())
```

100%|██████████| 20000/20000 [00:00<00:00, 176735.48it/s]  
100%|██████████| 16000/16000 [00:00<00:00, 201002.38it/s]

```
1 w2v_model_train = Word2Vec(list_of_sent_train_avgw2v,min_count=5,size=50,workers=4)
2 w2v_words_Logistic_reg_train=list(w2v_model_train.wv.vocab)
```

```
1 w2v_model_test = Word2Vec(list_of_sent_test_avgw2v,min_count=5,size=50,workers=4)
2 w2v_words_Logistic_reg_test=list(w2v_model_test.wv.vocab)
```

```
1 w2v_model_cv = Word2Vec(list_of_sent_cv_avgw2v,min_count=5,size=50,workers=4)
2 w2v_words_Logistic_reg_cv=list(w2v_model_cv.wv.vocab)
```

```
1 print(len(w2v_words_Logistic_reg_train))
2 print(w2v_words_Logistic_reg_train[0:50])
```

10246  
['continu', 'enjoy', 'dove', 'dark', 'chocol', 'order', 'receiv', 'par', 'one', 'reason', 'close', 'expir', 'dat

```
1 train_vectors=[];
2 for sent in list_of_sent_train_avgw2v:
3     sent_vec=np.zeros(50)
4     cnt_words=0;
5     for word in sent:
6         if word in w2v_words_Logistic_reg_train:
7             vec=w2v_model_train.wv[word]
8             sent_vec+=vec
```

```

9         cnt_words+=1
10     if cnt_words !=0:
11         sent_vec/=cnt_words
12     train_vectors.append(sent_vec)
13 print(len(train_vectors))
14 print(len(train_vectors[0]))

```

↗ 64000  
50

```

1 test_vectors=[];
2 for sent in list_of_sent_test_avgw2v:
3     sent_vec=np.zeros(50)
4     cnt_words=0;
5     for word in sent:
6         if word in w2v_words_Logistic_reg_test:
7             vec=w2v_model_test.wv[word]
8             sent_vec+=vec
9             cnt_words+=1
10    if cnt_words !=0:
11        sent_vec/=cnt_words
12    test_vectors.append(sent_vec)
13 print(len(test_vectors))
14 print(len(test_vectors[0]))

```

↗ 20000  
50

```

1 cv_vectors=[];
2 for sent in list_of_sent_cv_avgw2v:
3     sent_vec=np.zeros(50)
4     cnt_words=0;
5     for word in sent:
6         if word in w2v_words_Logistic_reg_cv:
7             vec=w2v_model_cv.wv[word]
8             sent_vec+=vec
9             cnt_words+=1
10    if cnt_words !=0:
11        sent_vec/=cnt_words
12    cv_vectors.append(sent_vec)
13 print(len(cv_vectors))
14 print(len(cv_vectors[0]))

```

↗ 16000  
50

```

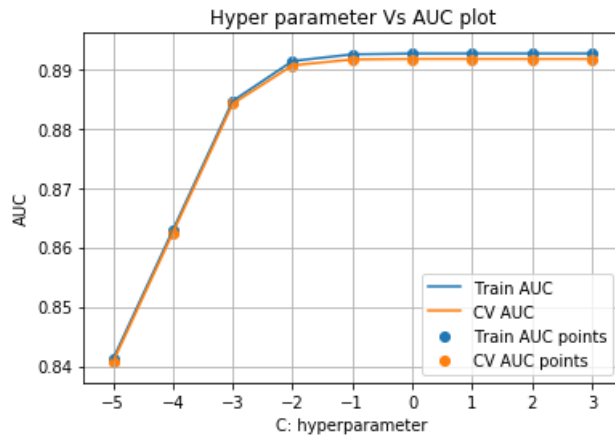
1 results,best_c,Logistic_Regression_Optimal_C = Logistic_Regression_Optimal_C(train_vectors,Y_Train)
2 results.head()

```

↗

0.8916826145952774

{ 'C': 1 }



	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split0_test_score	split1_test
0	0.440008	0.010672	0.040468	0.006611	1e-05	{'C': 1e-05}	0.839766	0
1	0.460969	0.008436	0.039006	0.000848	0.0001	{'C': 0.0001}	0.863514	0
2	0.691445	0.008001	0.038812	0.000781	0.001	{'C': 0.001}	0.886444	0
3	1.172471	0.044348	0.038245	0.000290	0.01	{'C': 0.01}	0.891425	0
4	1.584745	0.055563	0.038497	0.000528	0.1	{'C': 0.1}	0.891924	0

```

1 best_C = best_c.best_params_
2 best_c=best_C.get("C")
3 print(best_c)

```

```

↳ 1

```

```

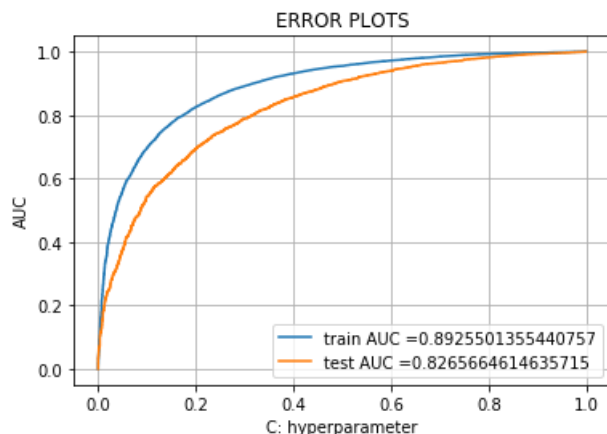
1 from sklearn.metrics import roc_curve, auc
2
3 logistic_reg = LogisticRegression(C=best_c,random_state=None, class_weight=None)
4 logistic_reg.fit(train_vectors,Y_Train)
5 pred_test_data=logistic_reg.predict(test_vectors)
6 y_train_predicted_prob = logistic_reg.predict_proba(train_vectors)[: ,1]
7 y_test_predicted_prob=logistic_reg.predict_proba(test_vectors)[: ,1]
8 train_fpr, train_tpr, train_thresholds=roc_curve(Y_Train,y_train_predicted_prob,pos_label='positive')
9 test_fpr, test_tpr, test_thresholds = roc_curve(Y_Test, y_test_predicted_prob,pos_label='positive')
10 plt.plot(train_fpr, train_tpr, label="train AUC =" +str(auc(train_fpr, train_tpr)))
11 plt.plot(test_fpr, test_tpr, label="test AUC =" +str(auc(test_fpr, test_tpr)))
12 plt.legend()
13 plt.xlabel("C: hyperparameter")
14 plt.ylabel("AUC")
15 plt.title("ERROR PLOTS")
16 plt.grid()
17 plt.show()

```

```

↳

```



```
1 from sklearn.metrics import roc_auc_score
2 from sklearn.metrics import classification_report, confusion_matrix
3
4 roc_auc_score(Y_Test, y_test_predicted_prob)
```

0.8265664614635715

```
1 print(classification_report(Y_Test, pred_test_data))
2 print(confusion_matrix(Y_Test, pred_test_data))
```

```
precision    recall  f1-score   support

negative     0.75     0.01     0.01      2457
positive     0.88     1.00     0.93     17543

accuracy          0.88      20000
macro avg     0.81     0.50     0.47      20000
weighted avg  0.86     0.88     0.82      20000

[[ 15 2442]
 [  5 17538]]
```

## ▼ TF-IDF Word2Vec Vectorization Technique for Logistic Regression on Amazon Fine Food

```
1 model_Avgw2v = TfidfVectorizer()
2 X_Train_Avgw2v=model_Avgw2v.fit_transform(X_Train['Cleaned_text'].values)
```

```
1 X_Test_Avgw2v=model_Avgw2v.transform(X_Test['Cleaned_text'].values)
2 X_CV_Avgw2v=model_Avgw2v.transform(X_CV['Cleaned_text'].values)
```

```
1 dictionary = dict(zip(model_Avgw2v.get_feature_names(), list(model_Avgw2v.idf_)))
```

```
1 tfidf_feature=model_Avgw2v.get_feature_names()
2
3 tfidf_sent_vectors_train=[];
4 #final_tf_idf = [];
5 row=0;
6
7 for sent in tqdm(list_of_sent_train_avgw2v):
8     sent_vec=np.zeros(50)
9     weight_sum=0;
10    for word in sent :
11        if word in w2v_words_Logistic_reg_train and word in tfidf_feature :
12            vec=w2v model_train.wv[word]
```



```

13         #tf_idf=final_tf_idf[row,tfidf_feature.index(word)]
14         tf_idf=dictionary[word]*(sent.count(word)/len(sent))
15         sent_vec+=(vec*tf_idf)
16         weight_sum+=tf_idf
17
18     if weight_sum!=0:
19         sent_vec/=weight_sum
20     tfidf_sent_vectors_train.append(sent_vec)
21     row+=1

```

100% |██████████| 64000/64000 [12:17<00:00, 110.80it/s]

```

1 tfidf_sent_vectors_test=[];
2 #final_tf_idf = [];
3 row=0;
4
5 for sent in tqdm(list_of_sent_test_avg2v):
6     sent_vec=np.zeros(50)
7     weight_sum=0;
8     for word in sent :
9         if word in w2v_words_Logistic_reg_test and word in tfidf_feature :
10             vec=w2v_model_test.wv[word]
11             #tf_idf=final_tf_idf[row,tfidf_feature.index(word)]
12             tf_idf=dictionary[word]*(sent.count(word)/len(sent))
13             sent_vec+=(vec*tf_idf)
14             weight_sum+=tf_idf
15
16     if weight_sum!=0:
17         sent_vec/=weight_sum
18     tfidf_sent_vectors_test.append(sent_vec)
19     row+=1

```

100% |██████████| 20000/20000 [03:48<00:00, 87.35it/s]

```

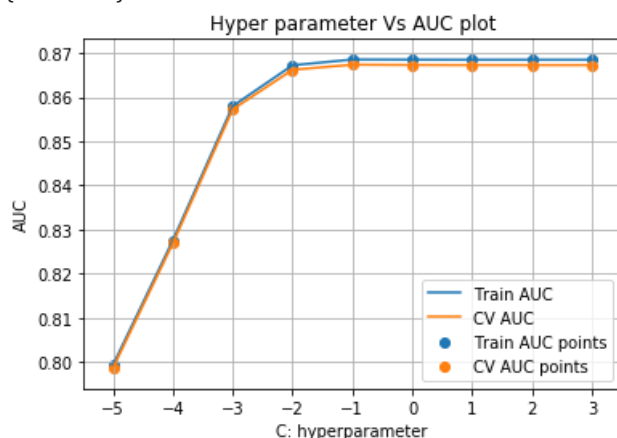
1 results,best_c,Logistic_Regression_Optimal_C = Logistic_Regression_Optimal_C(tfidf_sent_vectors_train,Y_Train)
2 results.head()

```

↵

0.8673362513641042

{ 'C': 0.1 }



	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split0_test_score	split1_test_score
0	0.467593	0.010707	0.039096	0.001151	1e-05	{'C': 1e-05}	0.809282	0
1	0.480854	0.005201	0.039248	0.001363	0.0001	{'C': 0.0001}	0.834692	0
2	0.718839	0.010308	0.038745	0.000836	0.001	{'C': 0.001}	0.863065	0
3	1.343019	0.066285	0.039566	0.000806	0.01	{'C': 0.01}	0.871818	0
4	1.728029	0.072872	0.038878	0.000352	0.1	{'C': 0.1}	0.872574	0

```

1 best_C = best_c.best_params_
2 best_c=best_C.get("C")
3 print(best_c)

```

```

0.1

```

```

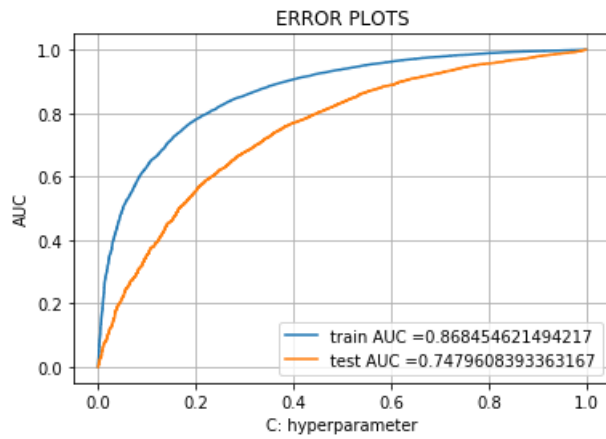
1 from sklearn.metrics import roc_curve, auc
2
3 logistic_reg = LogisticRegression(C=best_c,random_state=None, class_weight=None)
4 logistic_reg.fit(tfidf_sent_vectors_train,Y_Train)
5 pred_test_data=logistic_reg.predict(tfidf_sent_vectors_test)
6 y_train_predicted_prob = logistic_reg.predict_proba(tfidf_sent_vectors_train)[: ,1]
7 y_test_predicted_prob=logistic_reg.predict_proba(tfidf_sent_vectors_test)[: ,1]
8 train_fpr, train_tpr, train_thresholds=roc_curve(Y_Train,y_train_predicted_prob,pos_label='positive')
9 test_fpr, test_tpr, test_thresholds = roc_curve(Y_Test, y_test_predicted_prob,pos_label='positive')
10 plt.plot(train_fpr, train_tpr, label="train AUC =" +str(auc(train_fpr, train_tpr)))
11 plt.plot(test_fpr, test_tpr, label="test AUC =" +str(auc(test_fpr, test_tpr)))
12 plt.legend()
13 plt.xlabel("C: hyperparameter")
14 plt.ylabel("AUC")
15 plt.title("ERROR PLOTS")
16 plt.grid()
17 plt.show()

```

```

0.1

```



```
1 from sklearn.metrics import roc_auc_score
2 from sklearn.metrics import classification_report, confusion_matrix
3
4 roc_auc_score(Y_Test, y_test_predicted_prob)
```

→ 0.7479608393363167

```
1 print(classification_report(Y_Test,pred_test_data))
2 print(confusion_matrix(Y_Test,pred_test_data))
```

	precision	recall	f1-score	support
negative	0.38	0.05	0.09	2457
positive	0.88	0.99	0.93	17543
accuracy			0.87	20000
macro avg	0.63	0.52	0.51	20000
weighted avg	0.82	0.87	0.83	20000

```
[[ 130 2327]
 [ 214 17329]]
```

```
1 pip install -U PTable
```

```

➤ Collecting PTable
  Downloading https://files.pythonhosted.org/packages/ab/b3/b54301811173ca94119eb474634f120a49cd370f257d1aae5a4f
Building wheels for collected packages: PTable
  Building wheel for PTable (setup.py) ... done
  Created wheel for PTable: filename=PTable-0.9.2-cp36-none-any.whl size=22908 sha256=47f54d5b1c93581e6d9d5a49ef
  Stored in directory: /root/.cache/pip/wheels/22/cc/2e/55980bfe86393df3e9896146a01f6802978d09d7ebc5a5ea56
Successfully built PTable
Installing collected packages: PTable
Successfully installed PTable-0.9.2

```

```
1 from prettytable import PrettyTable
2
3 x= PrettyTable()
4 x.field_names = ["Vectorizer" , "Hyperparameter", "AUC"]
5 x.add_row(["Bag Of Words",0.1,0.923486])
6 x.add_row(["Tf-Idf",1,0.945869])
7 x.add_row(["Avg Word2Vec ", 1,0.826496])
8 x.add_row(["Avg Tf-Idf",0.1,0.723985])
9 print(x)
```



Vectorizer	Hyperparameter	AUC
Bag Of Words	0.1	0.923486
Tf-Idf	1	0.945869
Avg Word2Vec	1	0.826496
Avg Tf-Idf	0.1	0.723985

## CONCLUSION

BEST C = 1 , ACCURACY = 0.94589 USING TF-IDF EATUREIZATION TECHNIQUE

BAG OF WORDS AND TF-IDF PERFORMED WELL

AFTER SPARSITY THE DATAPOINT DRACTICALLY GET REDUCED.