objective :

1. Applying logistic regression to classify the amazon food reviews.
2. Check for different types of scoring metrics
3. Getting the important features.
4. checking for multicollinearity

In [1]:
```
#importing all module
import sqlite3 as s
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.cross_validation import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import confusion_matrix


from scipy.sparse import find

#importing the needed module
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

from sklearn.feature_extraction.text import CountVectorizer
from sklearn import preprocessing

#importing the needed module
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score,make_scorer
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
```

```
C:\Users\himateja\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: D
eprecationWarning: This module was deprecated in version 0.18 in favor of the m
odel_selection module into which all the refactored classes and functions are m
oved. Also note that the interface of the new CV iterators are different from t
hat of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [2]:
```
con=s.connect("database.sqlite")
con
```

Out[2]:
```
<sqlite3.Connection at 0x1b399af73b0>
```

In [3]:
```
data=pd.read_sql_query("SELECT * FROM Reviews WHERE Score!=3",con)
data.head(5)
```

Out[3]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominat
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	

In [4]:
```
#function to change the score to positive/negative
def change(x):
    if x<3:
        return 'negative'
    else:
        return 'positive'
```

In [5]:
```
a_s=data.Score
a_s=a_s.map(change)
data.Score=a_s
data.Score.head(5)
```

Out[5]:
```
0    positive
1    negative
2    positive
3    negative
4    positive
Name: Score, dtype: object
```

Data cleaning

The data needs to get clean as it may have some unwanted things such as duplicates.

In [6]:
```
#sorting the values by product ids
data=data.sort_values("ProductId")
```

In [7]:
```
#removing the duplicates from the data
final_data=data.drop_duplicates(subset={"UserId","Text","ProfileName","Time"},kee
```

In [8]:
```
final_data=final_data[final_data['HelpfulnessNumerator']<=final_data['Helpfulness
```

In [9]:
```
print(final_data.shape)
print(final_data.Score.value_counts())
```

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

In [10]:
```
p_data=final_data[final_data.Score=="positive"]
n_data=final_data[final_data.Score=="negative"]
```

In [11]:
```
#randomly selecting points
p_data_w2v=p_data.sample(150000)
n_data_w2v=n_data.sample(12000)
```

In [12]:
```
print(p_data.shape,n_data.shape)
```

```
(307061, 10) (57110, 10)
```

In [13]:
```
d=pd.concat((p_data,n_data))
d_w2v=pd.concat((p_data_w2v,n_data_w2v))
```

In [14]:
```
print(d.shape)
```

```
(364171, 10)
```

In [15]:
```
#sorting according to time stamp
d=d.sort_values('Time')
d_w2v=d_w2v.sort_values('Time')
```

In [16]:
```
print(d.Score.value_counts())
print(d_w2v.Score.value_counts())
```

```
positive    307061
negative     57110
Name: Score, dtype: int64
positive    150000
negative     12000
Name: Score, dtype: int64
```

Data preprocessing

The data should be preprocessed after cleaning it

In [17]:
```
import string
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer
import re
```

In [18]:
```
#stopwords
stop_words=set(stopwords.words("english"))
#initializing snowball stemmer
sno=SnowballStemmer("english")
print(stop_words)
```

```
{"weren't", 'all', 'no', 'same', 'by', 'she', 'didn', 'being', 'o', 'mightn',
'under', 'his', 'if', 'y', 'as', "didn't", 'which', 'having', 'is', 'does', 'ha
sn', 'your', 'an', 'than', "hadn't", 'am', 's', 'for', 'their', 'haven', 'and',
'will', 'doing', 'himself', 'what', 'where', "you've", 'd', 'our', 'm', 'so',
'be', 'few', "she's", 'couldn', 'weren', 'during', 'such', 'can', 'only', 'thi
s', 'i', 'of', 'down', 'isn', "isn't", 'aren', 'a', "you'll", 'yourself', 'to',
'how', 'then', 'don', 'other', 'her', 'should', 'those', 'them', 'until', 'ha
d', 'further', "don't", 'between', 'll', "couldn't", "won't", 'or', 'shouldn',
'whom', 'after', 'herself', 'but', 'my', 'into', "wasn't", 'against', 'these',
'hadn', 'you', 'been', "needn't", 'while', 'we', 'theirs', 'they', 'about', 'u
p', 'each', 'own', 'he', 'just', 'in', "mightn't", 'do', 'who', 'why', 'wasn',
'off', 'wouldn', 'myself', "you're", "wouldn't", 'any', 'out', 'both', 'here',
'more', 'above', 'are', 'very', 'itself', 'not', 'yourselves', 'did', 'hers',
'when', 'below', 'has', 'it', 'at', 'most', "aren't", "haven't", 'again', 'hi
m', 'doesn', 'on', 'were', "doesn't", 'me', 'over', 'won', 'ain', "shouldn't",
'some', 'ours', "it's", 've', 'mustn', 'ma', 'now', 'have', 're', 'themselves',
'there', "hasn't", 'nor', 'that', 'because', 'through', 'before', 'once', 'ours
elves', "that'll", 'with', 'its', 'the', "you'd", 'from', 'too', 'needn', "sha
n't", 'was', 'shan', "should've", 'yours', "mustn't", 't'}
```

In [19]:
```
#function to remove html tags
def cleanhtml(s):
    cleanr=re.compile("<.*?>")
    cleant=re.sub(cleanr," ",s)
    return cleant
```

In [20]:
```
#funtion to remove punctuation and special character
def cleanpunc(s):
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',s)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
```

In [21]:
```
i=0
final=[]

for s in d.Text.values:
    f=[]
    c=cleanhtml(s)
    for w in cleanpunc(c).split():
        if w.isalpha() and len(w)>2:
            if w not in stop_words:
                sne=(sno.stem(w.lower())).encode('utf-8')
                f.append(sne)

            else:
                continue
        else:
            continue
    te=b" ".join(f)
    final.append(te)
    i+=1
```

In [22]:
```
#adding the preprocessed data into another column
d["cleaned"]=final
d
```

Out[22]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNu
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	
138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	
417839	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	
346055	374359	B00004CI84	A344SMIA5JECGM	Vincent P. Ross	

In [23]:
```
#checking if new column is added
d.columns
```

Out[23]:
```
Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator',
       'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text',
       'cleaned'],
      dtype='object')
```

In [24]:
```
import sqlite3
conn=sqlite3.connect("future.sqlite")
c=conn.cursor()
conn.text_factory=str
d.to_sql('Reviews',conn,if_exists='replace',index=True)
```

In [25]:
```
#replacing positive and negative with 1,0 to make it work for different metrics
d=d.replace(['positive','negative'],[1,0])
d.head(10)
```

Out[25]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessD
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	
138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	
417839	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	
346055	374359	B00004CI84	A344SMIA5JECGM	Vincent P. Ross	1	
417838	451855	B00004CXX9	AJH6LUC1UT1ON	The Phantom of the Opera	0	
346116	374422	B00004CI84	A1048CYU0OV4O8	Judy L. Eans	2	
346041	374343	B00004CI84	A1B2IZU1JLZA6	Wes	19	
70688	76882	B00002N8SM	A32DW342WBJ6BX	Buttersugar	0	
346141	374450	B00004CI84	ACJR7EQF9S6FP	Jeremy Robertson	2	
346094	374400	B00004CI84	A2DEE7F9XKP3ZR	jerome	0	

Bag of words

In [99]:
```
#splitting
x_1, x_test, y_1, y_test = train_test_split(d.cleaned.values, d.Score, test_size=

#x_tr, x_cv, y_tr, y_cv = train_test_split(x_1, y_1, test_size=0.3)
```

In [100]:
```
print(x_1.shape,x_test.shape,y_1.shape,y_test.shape)
```

```
(254919,) (109252,) (254919,) (109252,)
```

In [101]:
```
#bigrams
count_vect=CountVectorizer(ngram_range=(1,2))
```

In [102]:
```
#transforming the data
bdata=count_vect.fit_transform(x_1)
#preprocessing the data
bdata=preprocessing.normalize(bdata)
#transforming the data
test_data=count_vect.transform(x_test)
#preprocessing the data
test_data=preprocessing.normalize(test_data)
print(bdata.shape,y_1.shape,test_data.shape,y_test.shape)
```

```
(254919, 2351241) (254919,) (109252, 2351241) (109252,)
```

In [103]:
```
#using the gridsearchcv for finding the best c
lr = LogisticRegression()
param_grid = {'C':[0.0001,0.001,0.01,0.1,1,10,100],"penalty":["l1","l2"]}
#For time based splitting
t = TimeSeriesSplit(n_splits=5)

gsv = GridSearchCV(lr,param_grid,cv=t,n_jobs=-1,verbose=1,scoring="f1")
gsv.fit(bdata,y_1)
print("Best HyperParameter: ",gsv.best_params_)
#assinging best alpha to optimal print("Best Accuracy: %.2f%%"%(gsv.best_score_*1
print("best estimator: ",gsv.estimator)
```

```
Fitting 5 folds for each of 14 candidates, totalling 70 fits

[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  2.3min
[Parallel(n_jobs=-1)]: Done  70 out of  70 | elapsed: 11.6min finished

Best HyperParameter:  {'C': 100, 'penalty': 'l2'}
best estimator:  LogisticRegression(C=1.0, class_weight=None, dual=False, fit_i
ntercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [104]:
```
gsv.best_score_*100
```

Out[104]:
```
95.9740680876888
```

In [105]:
```
gsv.cv_results_
```

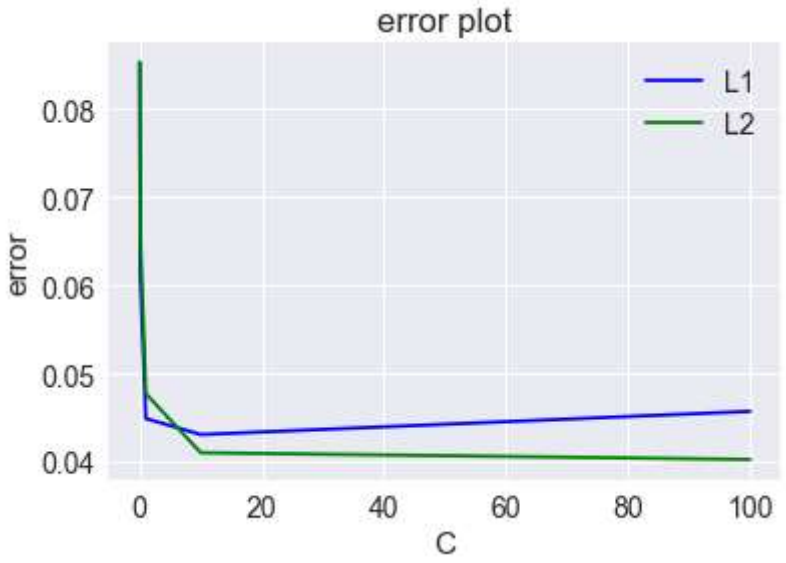
```
 'split2_train_score': array([0.93043161, 0.93043161, 0.93043161, 0.93043161,
0.93067379,
        0.93043161, 0.94999935, 0.94674546, 0.96459449, 0.96760948,
        0.99405003, 0.99226085, 0.99999098, 0.99997745]),
 'split3_train_score': array([0.9257425 , 0.9257425 , 0.9257425 , 0.9257425 ,
0.9272306 ,
        0.92586786, 0.95028654, 0.94776761, 0.96406356, 0.96729214,
        0.99369546, 0.99164274, 0.99999317, 0.9999761 ]),
 'split4_train_score': array([0.92248034, 0.92248034, 0.92248034, 0.92248034,
0.92631079,
        0.92338001, 0.95080625, 0.94875985, 0.96405682, 0.96717254,
        0.99369473, 0.99119119, 0.99999175, 0.99997526]),
 'mean_train_score': array([0.93128707, 0.93128707, 0.93128707, 0.93128707,
0.93239922,
        0.93149208, 0.94948755, 0.9464256 , 0.9645562 , 0.96734083,
        0.99404987, 0.99250031, 0.99999251, 0.99998041]),
 'std_train_score': array([6.93522268e-03, 6.93522268e-03, 6.93522268e-03, 6.
93522268e-03,
        5.77692104e-03, 6.69126518e-03, 1.39338197e-03, 1.84723452e-03,
```

we can see that l1 and l2 are alternatively executed . so , lets separate the values.

In [106]:
```
#separating the regularization values
mean_test_score=gsv.cv_results_["mean_test_score"]
mean_test_score=1-mean_test_score
y_l1=[]
y_l2=[]
for i in mean_test_score[0:14:2]:
    y_l1.append(i)
for i in mean_test_score[1:14:2]:
    y_l2.append(i)
print(y_l1)
print("\n\n")
print(y_l2)
```

```
[0.08524499261752805, 0.08524499261752805, 0.08339273461120578, 0.0611150594149
4089, 0.04490093654388516, 0.04307863051324812, 0.04570984604551631]



[0.08524499261752805, 0.08524499261752805, 0.0848666413108411, 0.06595224081153
384, 0.04772442038977054, 0.041022223684298975, 0.040259319123111187]
```

In [107]:
```
#plotting the error
x=param_grid["C"]
plt.plot(x,y_l1,"b",label="L1")
plt.plot(x,y_l2,"g",label="L2")
plt.xlabel("C")
plt.ylabel("error")
plt.title("error plot")
plt.legend()
plt.show()
```



now lets test the optimal value on test data

In [108]:
```
lr=LogisticRegression(C=100,penalty="l2")
lr.fit(bdata,y_1)
```

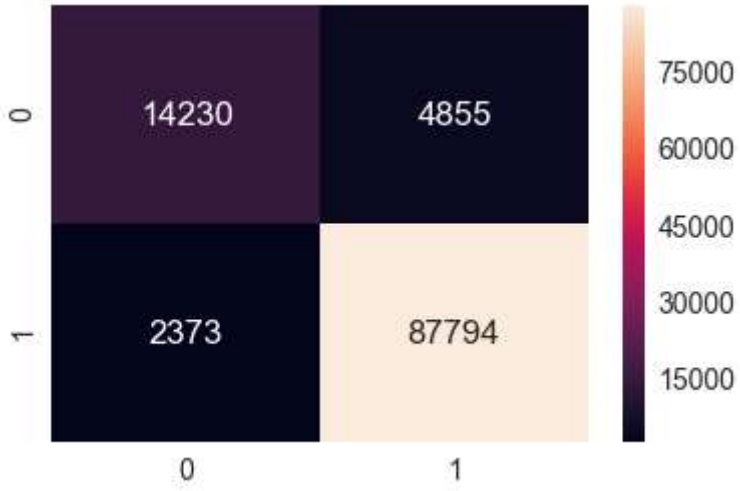
Out[108]:
```
LogisticRegression(C=100, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [109]:
```
pred=lr.predict(test_data)
```

In [110]:
```
#different metrics
acc=accuracy_score(y_test,pred)*100
print("\nthe accuracy is %.2f%%"%acc)

re=recall_score(y_test,pred,) * 100
print("\nthe recall is %.2f%%"%re)

pre=precision_score(y_test,pred) * 100
print("the precision is %.2f%%"%pre)

f1=f1_score(y_test,pred) * 100
print("the f1 score is %.2f%%"%f1)

df_cm=pd.DataFrame(confusion_matrix(y_test,pred))
#sns.set(font_scale=1.4)
sns.heatmap(df_cm,annot=True,fmt="d")
```

```
the accuracy is 93.38%

the recall is 97.37%
the precision is 94.76%
the f1 score is 96.05%
```

Out[110]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2754d03ceb8>
```



In [111]:
```
scoring_data=pd.DataFrame({"accuracy":"93.38%","recall":"97.37%","precision":"94.
scoring_data.T
```

Out[111]:

	score
accuracy	93.38%
recall	97.37%
precision	94.76
f1	96.05%

Lets see what happens to sparsity when lambda increases (c decreases) with L1 regularization

In [112]:
```
lr=LogisticRegression(C=1,penalty="l1")
lr.fit(bdata,y_1)
np.count_nonzero(lr.coef_)
```

Out[112]: 2634

In [113]:
```
lr=LogisticRegression(C=0.1,penalty="l1")
lr.fit(bdata,y_1)
np.count_nonzero(lr.coef_)
```

Out[113]: 453

In [114]:
```
lr=LogisticRegression(C=0.01,penalty="l1")
lr.fit(bdata,y_1)
np.count_nonzero(lr.coef_)
```

Out[114]: 57

In [115]:
```
lr=LogisticRegression(C=0.001,penalty="l1")
lr.fit(bdata,y_1)
np.count_nonzero(lr.coef_)
```

Out[115]: 1

sparsity increases with increase in lambda(decrease in C)

In [116]:
```
#randomizedsearchcv is used to find optimal c and regulariztion
lr = LogisticRegression()
param_grid = {'C':randint(10**-2, 10**2),"penalty":["l1","l2"]}
#For time based splitting
t = TimeSeriesSplit(n_splits=5)

rsv = RandomizedSearchCV(lr,param_grid,cv=t,n_jobs=-1,verbose=1)
rsv.fit(bdata,y_1)
print("Best HyperParameter: ",rsv.best_params_)
#assinging best alpha to optimal print("Best Accuracy: %.2f%%"%(gsv.best_score_*1
print("best estimator: ",rsv.estimator)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits

[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed: 14.9min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed: 20.1min finished

Best HyperParameter:  {'C': 79, 'penalty': 'l2'}
best estimator:  LogisticRegression(C=1.0, class_weight=None, dual=False, fit_i
ntercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

Pertubation test

This test is for multicollinearity check

In [117]:
```
#the data before adding noise
clf=LogisticRegression()
clf.fit(bdata,y_1)
```

Out[117]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [118]:
```
np.count_nonzero(clf.coef_)
```

Out[118]:
```
2351241
```

In [119]:
```
wt_test=find(clf.coef_)[2]
print(wt_test[0:20])
```

```
[ 2.95206467e-02  2.92571924e-02  2.63454358e-04  1.61430043e-01
  1.15755606e-01  3.77762247e-03 -2.40949890e-02  2.48134930e-03
  6.32093536e-03  2.82106843e-03  1.56434515e-03  5.28477633e-04
  3.84257308e-02  1.56737879e-04  1.29260824e-03  1.29260824e-03
  6.07704500e-03  6.07704500e-03  9.78693576e-05  9.78693576e-05]
```

In [120]:
```
#randomling getting noise for all nonzero values
bdata_t=bdata
epsilon = np.random.uniform(low=-0.001, high=0.001, size=(find(bdata_t)[0].size,)
i,j,v=find(bdata_t)
print(epsilon)
```

```
[-0.00096668 -0.0002961   0.0001949  ...  0.00041346 -0.00064359
 -0.00018327]
```

In [121]:
```
#adding the noise to nonzero value
bdata_t[i,j] = epsilon + bdata_t[i,j]
```

In [122]:
```
#the data after adding noise
cl=LogisticRegression()
cl.fit(bdata_t,y_1)
```

Out[122]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [123]:
```
np.count_nonzero(cl.coef_)
```

Out[123]:
```
2351241
```

In [124]:
```
wt_test_noise=find(cl.coef_[0])[2]
print(wt_test_noise[0:20])
```

```
[ 2.95171555e-02  2.93880598e-02  2.55116481e-04  1.61207681e-01
  1.15946295e-01  3.77784478e-03 -2.39667566e-02  2.44735048e-03
  6.33426888e-03  2.77482622e-03  1.54081980e-03  5.32318649e-04
  3.83345599e-02  1.57880619e-04  1.31856287e-03  1.31021722e-03
  5.99185334e-03  5.98170349e-03  9.69787143e-05  9.73391520e-05]
```

As the difference between the values is not significant , features are not multicollinear

Feature importance

as the features are not mutlicollinear , we can use l1 regularization to get imp features

In [125]:
```
countvect=CountVectorizer(ngram_range=(1,2),stop_words="english",analyzer='word')
feature_data=countvect.fit_transform(x_1)
all_feat = countvect.get_feature_names()
```

In [126]:
```
clf=LogisticRegression(C=1,penalty="l1")
clf.fit(feature_data,y_1)
```

Out[126]:
```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l1', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [127]:
```
#shape of the array
clf.coef_.shape
```

Out[127]:
```
(1, 2209266)
```

In [128]:
```
#converting the array to one dimensional
check=clf.coef_.ravel()
```

In [129]:
```
#creating the dataframe for features importance
feature_importance=pd.DataFrame({"word":all_feat,"coef_":check})
feature_importance.head(4)
```

Out[129]:

	word	coef_
0	aa	0.000000
1	aa pleas	0.000000
2	aa state	0.000000
3	aaa	0.028596

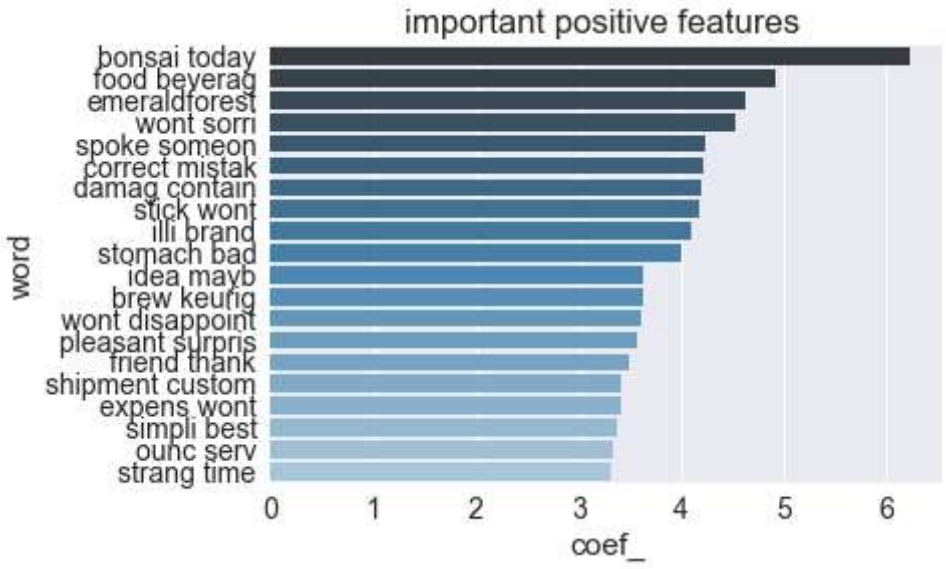
In [130]:
```
#sorting in the descending order
feature_importance.sort_values(by="coef_",ascending=False,inplace=True)
feature_importance.head(10)
```

Out[130]:

	word	coef_
205612	bonsai today	6.236173
757031	food beverag	4.913268
632314	emeraldforest	4.634703
2170887	wont sorri	4.536133
1821192	spoke someon	4.239659
445276	correct mistak	4.223517
492910	damag contain	4.200272
1851394	stick wont	4.184966
963200	illi brand	4.107079
1855900	stomach bad	3.997220

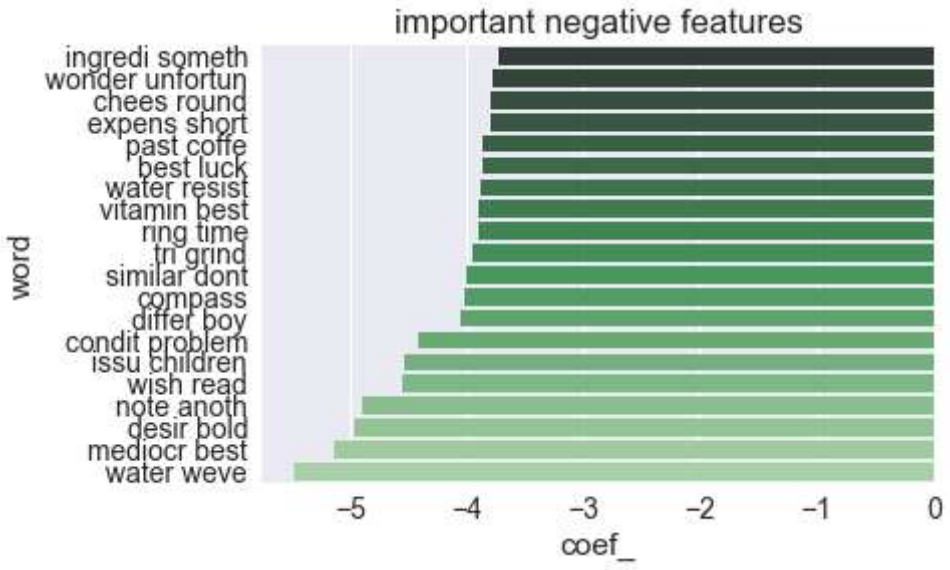
In [131]:
```
#visualising positive features
sns.barplot(y='word', x='coef_', data=feature_importance.head(20), palette="Blues_
```

Out[131]: Text(0.5,1,'important positive features ')



In [132]:
```
#visualing the negative features
sns.barplot(y='word', x='coef_', data=feature_importance.tail(20), palette="Green
```

Out[132]: Text(0.5,1,'important negative features')



In [133]:
```
summary=pd.DataFrame({"gridsearchcv":["100","L2"],"randomizedsearchcv":["90","L2"
summary.T
```

Out[133]:

	C	regularization
gridsearchcv	100	L2
randomizedsearchcv	90	L2

The model performs good for bag of words

Tfidf

In [26]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [27]:
```
#splitting
x_1, x_test, y_1, y_test = train_test_split(d.cleaned.values, d.Score, test_size=
```

In [28]:
```
tfidf=TfidfVectorizer(ngram_range=(1,2))
```

In [72]:
```
#transforming the data and normalizing
tdata=tfidf.fit_transform(x_1)
tdata=preprocessing.normalize(tdata)
test_data=tfidf.transform(x_test)
test_data=preprocessing.normalize(test_data)
print(tdata.shape,test_data.shape)
```

```
(254919, 2351241) (109252, 2351241)
```

In [138]:
```
#gridsearchcv for finding optimal c
lr = LogisticRegression()
param_grid = {'C':[0.0001,0.001,0.01,0.1,1,10,100],"penalty":["l1","l2"]}
#For time based splitting
t = TimeSeriesSplit(n_splits=5)

gsv = GridSearchCV(lr,param_grid,cv=t,n_jobs=-1,verbose=1,scoring="f1")
gsv.fit(tdata,y_1)
print("Best HyperParameter: ",gsv.best_params_)
#assinging best alpha to optimal print("Best Accuracy: %.2f%%"%(gsv.best_score_*1
print("best estimator: ",gsv.estimator)
```

```
Fitting 5 folds for each of 14 candidates, totalling 70 fits

[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  2.0min
[Parallel(n_jobs=-1)]: Done  70 out of  70 | elapsed:  9.1min finished

Best HyperParameter:  {'C': 100, 'penalty': 'l2'}
best estimator:  LogisticRegression(C=1.0, class_weight=None, dual=False, fit_i
ntercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [139]:
```
gsv.best_score_*100
```

Out[139]:
```
95.881560903948
```

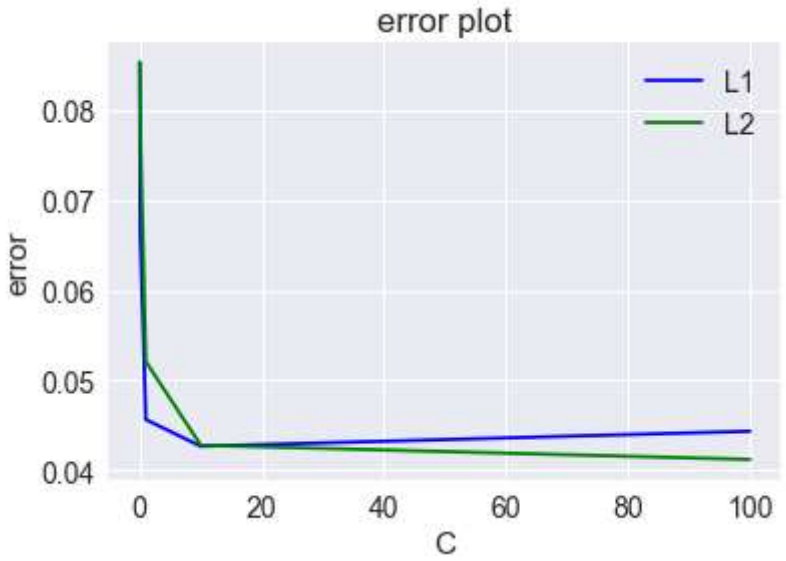
In [140]:
```
gsv.cv_results_
```

```
        0.99808528, 0.99924674, 1.        , 1.        ]),
 'split1_train_score': array([0.93607698, 0.93607698, 0.93607698, 0.93607698,
0.93607698,
        0.93607698, 0.94445747, 0.93659294, 0.96405152, 0.9665533 ,
        0.99816303, 0.99871747, 0.99998662, 0.99997994]),
 'split2_train_score': array([0.93043161, 0.93043161, 0.93043161, 0.93043161,
0.93043161,
        0.93043161, 0.94539179, 0.93437351, 0.96407992, 0.96894454,
        0.99820258, 0.99856332, 0.99999098, 0.99998647]),
 'split3_train_score': array([0.9257425 , 0.9257425 , 0.9257425 , 0.9257425 ,
0.9257425 ,
        0.9257425 , 0.94650316, 0.93528192, 0.96383388, 0.96987432,
        0.99832537, 0.99828503, 0.99999317, 0.99998976]),
 'split4_train_score': array([0.92248034, 0.92248034, 0.92248034, 0.92248034,
0.92263628,
        0.92248034, 0.9477968 , 0.93741255, 0.9641811 , 0.97041871,
        0.99841507, 0.99812767, 0.99999175, 0.99999175]),
 'mean_train_score': array([0.93128707, 0.93128707, 0.93128707, 0.93128707,
0.93131826,
        0.93128707, 0.94538858, 0.93707297, 0.96375548, 0.96737726,
```

In [141]:
```
#separating the values of regulerization
mean_test_score=gsv.cv_results_["mean_test_score"]
mean_test_score=1-mean_test_score
y_l1=[]
y_l2=[]
for i in mean_test_score[0:14:2]:
    y_l1.append(i)
for i in mean_test_score[1:14:2]:
    y_l2.append(i)
print(y_l1)
print("\n\n")
print(y_l2)
```

```
[0.08524499261752805, 0.08524499261752805, 0.08519099054896606, 0.0663395199085
2582, 0.04561982694223621, 0.04264709122839683, 0.04431391213889535]



[0.08524499261752805, 0.08524499261752805, 0.08524499261752805, 0.0779251663402
1995, 0.0520627897955962, 0.042760045170896044, 0.041184390960519957]
```

In [142]:
```
x=param_grid["C"]
plt.plot(x,y_l1,"b",label="L1")
plt.plot(x,y_l2,"g",label="L2")
plt.xlabel("C")
plt.ylabel("error")
plt.title("error plot")
plt.legend()
plt.show()
```



now lets try the optimal parameteres on the test data

In [143]:
```
lr=LogisticRegression(C=100,penalty="l2")
lr.fit(tdata,y_1)
```

Out[143]:
```
LogisticRegression(C=100, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [144]:
```
pred=lr.predict(test_data)
```

In [145]:
```
acc=accuracy_score(y_test,pred)*100
print("\nthe accuracy is %.2f%%"%acc)

re=recall_score(y_test,pred,) * 100
print("\nthe recall is %.2f%%"%re)

pre=precision_score(y_test,pred) * 100
print("the precision is %.2f%%"%pre)

f1=f1_score(y_test,pred) * 100
print("the f1 score is %.2f%%"%f1)

df_cm=pd.DataFrame(confusion_matrix(y_test,pred))
sns.set(font_scale=1.4)
sns.heatmap(df_cm,annot=True,fmt="d")
```

```
the accuracy is 93.46%

the recall is 97.60%
the precision is 94.65%
the f1 score is 96.10%
```

Out[145]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2750a3944e0>
```

In [146]:
```
scoring_data=pd.DataFrame({"accuracy":"93.46%","recall":"97.60%","precision":"94.
scoring_data.T
```

Out[146]:

	score
accuracy	93.46%
recall	97.60%
precision	94.65%
f1	96.10%

lets see what happens to sparsity with increase in lambda or decrease in C

In [147]:
```
lr=LogisticRegression(C=10,penalty="l1")
lr.fit(tdata,y_1)
np.count_nonzero(lr.coef_)
```

Out[147]: 31092

In [148]:
```
lr=LogisticRegression(C=1,penalty="l1")
lr.fit(tdata,y_1)
np.count_nonzero(lr.coef_)
```

Out[148]: 2737

In [149]:
```
lr=LogisticRegression(C=0.1,penalty="l1")
lr.fit(tdata,y_1)
np.count_nonzero(lr.coef_)
```

Out[149]: 347

In [150]:
```
lr=LogisticRegression(C=0.01,penalty="l1")
lr.fit(tdata,y_1)
np.count_nonzero(lr.coef_)
```

Out[150]: 20

sparsity increases with increase in lambda

In [151]:
```
#randomizedsearchcv for finding optimal c
lr = LogisticRegression()
param_grid = {'C':randint(10**-2, 10**2),"penalty":["l1","l2"]}
#For time based splitting
t = TimeSeriesSplit(n_splits=5)

rsv = RandomizedSearchCV(lr,param_grid,cv=t,n_jobs=-1,verbose=1)
rsv.fit(tdata,y_1)
print("Best HyperParameter: ",rsv.best_params_)
#assinging best alpha to optimal print("Best Accuracy: %.2f%%"%(gsv.best_score_*1
print("best estimator: ",rsv.estimator)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits

[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed: 11.8min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed: 18.2min finished

Best HyperParameter:  {'C': 95, 'penalty': 'l2'}
best estimator:  LogisticRegression(C=1.0, class_weight=None, dual=False, fit_i
ntercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

Pertubation technique

multicollinearity check

In [152]:
```
#before noise
clf=LogisticRegression()
clf.fit(tdata,y_1)
```

Out[152]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [153]:
```
#nonzero values
np.count_nonzero(clf.coef_)
```

Out[153]:
```
2351241
```

In [154]:
```
wt_test=find(clf.coef_[0])[2]
print(wt_test[0:20])
```

```
[ 0.04645525  0.04744264  0.00053778  0.17007897  0.12798862  0.00935973
 -0.04212703  0.00659661  0.01312656  0.00701524  0.00392476  0.00213447
  0.0492709   0.00070833  0.0033779   0.0033779   0.01774095  0.01774095
  0.00050136  0.00050136]
```

In [155]:
```
#generating random noise for adding to nonzero value
tdata_t=tdata
epsilon = np.random.uniform(low=-0.001, high=0.001, size=(find(tdata_t)[0].size,)
i,j,v=find(tdata_t)
print(epsilon)
```

```
[-0.00039151 -0.00071995  0.00052939 ... -0.00093318 -0.00011491
  0.00067021]
```

In [156]:
```
#addition of noise
tdata_t[i,j] = epsilon + tdata_t[i,j]
```

In [157]:
```
cl=LogisticRegression()
cl.fit(tdata_t,y_1)
```

Out[157]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [158]:
```
np.count_nonzero(clf.coef_)
```

Out[158]:
```
2351241
```

In [159]:
```
wt_test_noise=find(cl.coef_[0])[2]
print(wt_test_noise[0:20])
```

```
[ 0.04579204  0.04705734  0.00057517  0.17021508  0.1278883   0.00933912
 -0.04190056  0.0065739   0.01337778  0.00713104  0.00397958  0.00215312
  0.04955252  0.00070725  0.00334899  0.00332952  0.01773055  0.01776998
  0.00050474  0.00050525]
```

The difference is very small.so, the feadtures are not multicollinear

Feature importance

as the features are not multicollinear we can use L1 regularizaiton

In [160]:
```
tf=TfidfVectorizer(ngram_range=(1,2),stop_words="english",analyzer='word')
feature_data=tf.fit_transform(x_1)
all_feat = tf.get_feature_names()
```

In [161]:
```
clf=LogisticRegression(C=1,penalty="l1")
clf.fit(feature_data,y_1)
```

Out[161]:
```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l1', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [162]:
```
#converting array to 1D
check=clf.coef_.ravel()
```

In [163]:
```
#creating dataframe for feature importance
feature_importance=pd.DataFrame({"word":all_feat,"coef_":check})
feature_importance.head(4)
```

Out[163]:

	word	coef_
0	aa	0.0
1	aa pleas	0.0
2	aa state	0.0
3	aaa	0.0

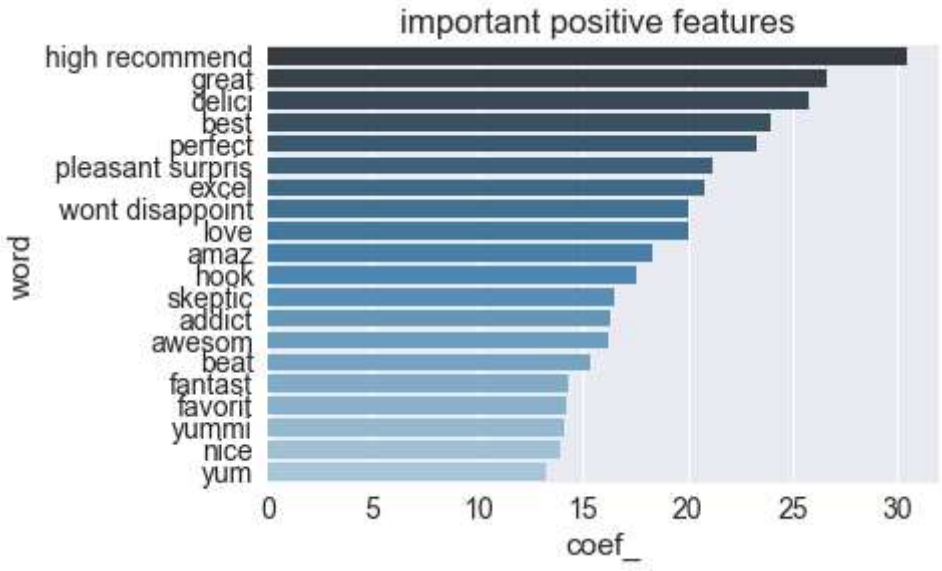
In [164]:
```
#sorting in descending order
feature_importance.sort_values(by="coef_",ascending=False,inplace=True)
feature_importance.head(10)
```

Out[164]:

	word	coef_
919679	high recommend	30.415174
849395	great	26.614199
518289	delici	25.806726
167357	best	23.982681
1407016	perfect	23.296294
1442136	pleasant surpris	21.142059
665793	excel	20.791440
2170289	wont disappoint	20.072441
1142736	love	20.056110
55385	amaz	18.324004

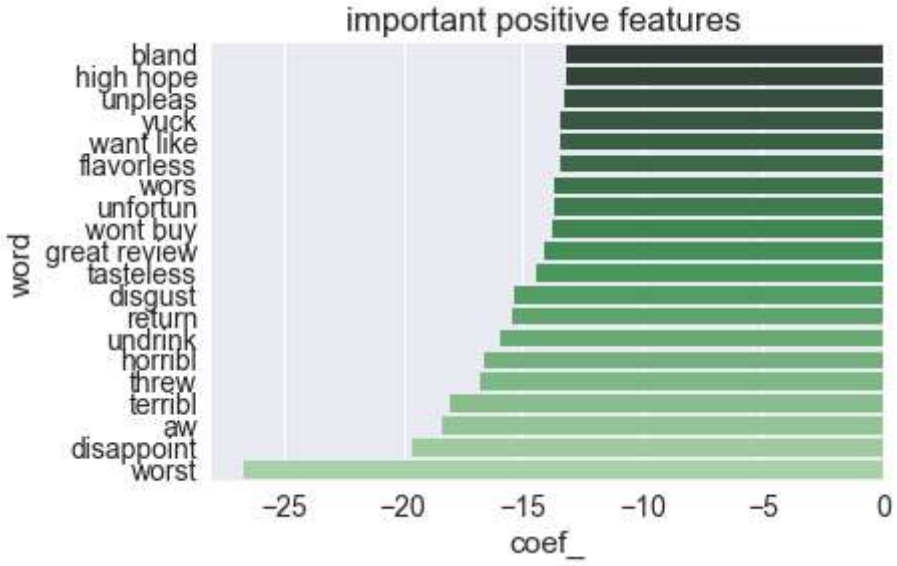
In [165]:
```
#visualising positive features
sns.barplot(y='word', x='coef_', data=feature_importance.head(20), palette="Blues_
```

Out[165]:
```
Text(0.5,1,'important positive features ')
```



In [166]:
```
#visualising negative features
sns.barplot(y='word', x='coef_', data=feature_importance.tail(20), palette="Green
```

Out[166]:
```
Text(0.5,1,'important positive features ')
```



In [167]:
```
summary=pd.DataFrame({"gridsearchcv":["100","L2"],"randomizedsearchcv":["90","L2"
summary.T
```

Out[167]:

	C	regularization
gridsearchcv	100	L2
randomizedsearchcv	90	L2

The model performs quiet good

Word2vec

In [26]:
```
from gensim.models import Word2Vec
#making list of sentences
import string
i=0
list_s=[]
for s in d.Text.values:
    filtered=[]
    s=cleanhtml(s)
    for w in s.split():
        for c_w in cleanpunc(w).split():
            if c_w.isalpha():
                filtered.append(c_w.lower())
            else:
                continue
    list_s.append(filtered)
#training our own model
w2v_model=Word2Vec(list_s,min_count=3,size=100,workers=8)
```

```
C:\Users\himateja\Anaconda3\lib\site-packages\gensim\utils.py:1209: UserWarnin
g: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

In [27]:
```
#splitting the data
x_1, x_test, y_1, y_test = train_test_split(list_s, d.Score, test_size=0.3, rando
```

Avg Word2vec

In [32]:
```
def avg(l):

    sv=[]
    for s in l:
        sum=np.zeros(100)
        i=0
        for w in s:
            try:

                x=w2v_model.wv[w]
                sum+=x
                i+=1
            except:
                pass
        sum/=i
        sv.append(sum)
    return np.matrix(sv)
```

In [33]:
```
x_1=avg(x_1)
```

In [34]:
```
np.asarray(x_1).shape
```

Out[34]:
```
(254919, 100)
```

In [35]:
```
x_test=avg(x_test)
```

```
C:\Users\himateja\Anaconda3\lib\site-packages\ipykernel_launcher.py:15: Runtime
Warning: invalid value encountered in true_divide
  from ipykernel import kernelapp as app
```

In [36]:
```
np.asarray(x_test).shape
```

Out[36]:
```
(109252, 100)
```

In [37]:
```
#changing nan values if any
train=np.nan_to_num(x_1)
test=np.nan_to_num(x_test)
```

In [39]:
```
#gridsearchcv
lr = LogisticRegression()
param_grid = {'C':[0.0001,0.001,0.01,0.1,1,10,100],"penalty":["l1","l2"]}
#For time based splitting
t = TimeSeriesSplit(n_splits=5)

gsv = GridSearchCV(lr,param_grid,cv=t,n_jobs=-1,verbose=1,scoring="f1")
gsv.fit(train,y_1)
print("Best HyperParameter: ",gsv.best_params_)
#assinging best alpha to optimal print("Best Accuracy: %.2f%%"%(gsv.best_score_*1
print("best estimator: ",gsv.estimator)
```

```
Fitting 5 folds for each of 14 candidates, totalling 70 fits

[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  2.9min
[Parallel(n_jobs=-1)]: Done  70 out of  70 | elapsed: 53.8min finished

Best HyperParameter:  {'C': 100, 'penalty': 'l2'}
best estimator:  LogisticRegression(C=1.0, class_weight=None, dual=False, fit_i
ntercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [40]:
```
gsv.best_score_*100
```

Out[40]:
```
94.00437890007348
```

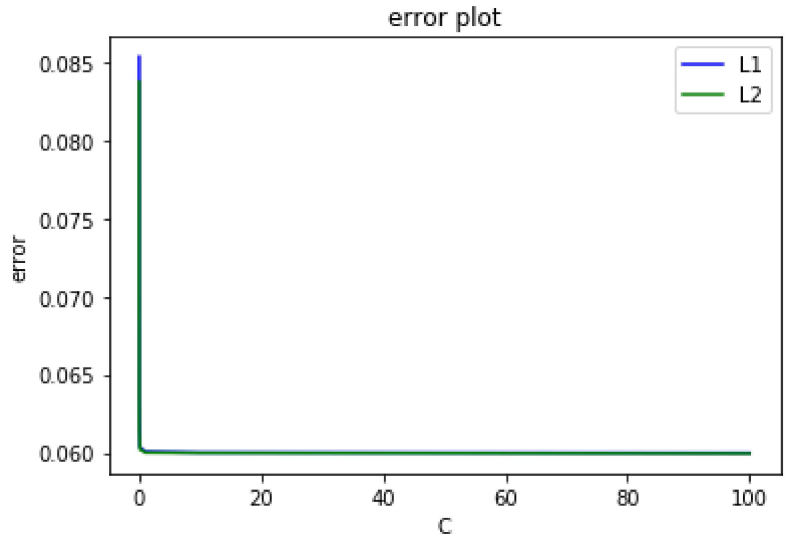
In [41]:
```
gsv.cv_results_
```

```
0.94680596,
        0.94779292, 0.94876772, 0.94870556, 0.94882997, 0.94880777,
        0.94881308, 0.94878412, 0.94882425, 0.94874625]),
 'split3_train_score': array([0.92573863, 0.92718704, 0.93208279, 0.94052266,
0.94439744,
        0.94538878, 0.94598843, 0.94596975, 0.94606549, 0.94608536,
        0.94613906, 0.94611272, 0.94615129, 0.94607488]),
 'split4_train_score': array([0.9224819 , 0.9255225 , 0.93191318, 0.9397927 ,
0.94363814,
        0.944303  , 0.9445567 , 0.94462853, 0.94468365, 0.94471312,
        0.94476286, 0.94475458, 0.94477121, 0.94471471]),
 'mean_train_score': array([0.93115747, 0.93231704, 0.93513384, 0.94228986,
0.94686082,
        0.94845073, 0.94948606, 0.94952913, 0.94968899, 0.94967839,
        0.9496983 , 0.9497039 , 0.94972079, 0.9496622 ]),
 'std_train_score': array([0.00674604, 0.00595939, 0.00368176, 0.00191893, 0.
00258174,
        0.00352219, 0.0041384 , 0.00421402, 0.00430744, 0.00428264,
        0.0042491 , 0.00427999, 0.00426379, 0.00426597])}
```

In [42]:
```
#separating the l1 and l2 values
mean_test_score=gsv.cv_results_["mean_test_score"]
mean_test_score=1-mean_test_score
y_l1=[]
y_l2=[]
for i in mean_test_score[0:14:2]:
    y_l1.append(i)
for i in mean_test_score[1:14:2]:
    y_l2.append(i)
print(y_l1)
print("\n\n")
print(y_l2)
```

```
[0.08533665718223216, 0.07964863606371586, 0.06354612686837757, 0.0603584734260
6504, 0.06006661715558015, 0.060012820364547914, 0.05999178671175853]



[0.08374508708523853, 0.0700839115999966, 0.061501063760627916, 0.0602301498629
757, 0.06001650006946502, 0.059995866149874955, 0.05995621099926529]
```

In [43]:
```
#error plot
x=param_grid["C"]
plt.plot(x,y_l1,"b",label="L1")
plt.plot(x,y_l2,"g",label="L2")
plt.xlabel("C")
plt.ylabel("error")
plt.title("error plot")
plt.legend()
plt.show()
```



lets apply best hyperparameter in action

In [45]:
```
lr=LogisticRegression(C=100,penalty="l2")
lr.fit(train,y_1)
```

Out[45]:
```
LogisticRegression(C=100, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [46]:
```
pred=lr.predict(test)
```

In [47]:
```
acc=accuracy_score(y_test,pred)*100
print("\nthe accuracy is %.2f%%"%acc)

re=recall_score(y_test,pred,) * 100
print("\nthe recall is %.2f%%"%re)

pre=precision_score(y_test,pred) * 100
print("the precision is %.2f%%"%pre)

f1=f1_score(y_test,pred) * 100
print("the f1 score is %.2f%%"%f1)

df_cm=pd.DataFrame(confusion_matrix(y_test,pred))
sns.set(font_scale=1.4)
sns.heatmap(df_cm,annot=True,fmt="d")
```

```
the accuracy is 89.00%

the recall is 96.47%
the precision is 90.77%
the f1 score is 93.54%
```

Out[47]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1c72f806ba8>
```

In [48]:
```
scoring_data=pd.DataFrame({"accuracy":"89.00%","recall":"96.47%","precision":"90.
scoring_data.T
```

Out[48]:

	score
accuracy	89.00%
recall	96.47%
precision	90.77%
f1	93.54%

lets see what happens to sparsity with increase in lambda

In [49]:
```
lr=LogisticRegression(C=10,penalty="l1")
lr.fit(train,y_1)
np.count_nonzero(lr.coef_)
```

Out[49]: 99

In [50]:
```
lr=LogisticRegression(C=1,penalty="l1")
lr.fit(train,y_1)
np.count_nonzero(lr.coef_)
```

Out[50]: 100

In [51]:
```
lr=LogisticRegression(C=.1,penalty="l1")
lr.fit(train,y_1)
np.count_nonzero(lr.coef_)
```

Out[51]: 97

In [52]:
```
lr=LogisticRegression(C=.01,penalty="l1")
lr.fit(train,y_1)
np.count_nonzero(lr.coef_)
```

Out[52]: 81

sparsity increases with increase in lambda

In [38]:
```
#randomizedsearchcv
lr = LogisticRegression()
param_grid = {'C':randint(10**-2, 10**2),"penalty":["l1","l2"]}
#For time based splitting
t = TimeSeriesSplit(n_splits=3)

rsv = RandomizedSearchCV(lr,param_grid,cv=t,n_jobs=-1,verbose=1)
rsv.fit(train,y_1)
print("Best HyperParameter: ",rsv.best_params_)
#assinging best alpha to optimal print("Best Accuracy: %.2f%%"%(gsv.best_score_*1
print("best estimator: ",rsv.estimator)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 49.3min finished

Best HyperParameter:  {'C': 33, 'penalty': 'l1'}
best estimator:  LogisticRegression(C=1.0, class_weight=None, dual=False, fit_i
ntercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

Pertubation technique

multicollinearity check

In [38]:
```
clf=LogisticRegression()
clf.fit(train,y_1)
```

Out[38]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [39]:
```
wt_test=find(clf.coef_[0])[2]
print(wt_test[0:20])
```

```
[-0.07323414 -0.27828451  0.27906553 -1.57250832  0.00744133 -0.3886471
  1.18881694 -0.15708664 -2.49435854  0.56476661  0.16660721 -0.32184196
 -1.32037805 -0.68279559  1.28570245  0.50028978  1.10845098  1.63209455
  0.33067109  1.66735704]
```

In [40]:
```
x_1_t=train
epsilon = np.random.uniform(low=-0.001, high=0.001, size=(find(x_1_t)[0].size,))
i,j,v=find(x_1_t)
print(epsilon)
```

```
[ 7.52625773e-04  8.58448417e-05 -8.90916048e-04 ...  7.79184291e-04
  5.51188742e-05  1.84753387e-04]
```

In [41]:
```
x_1_t[i,j] = epsilon + x_1_t[i,j]
```

In [42]:
```
clf=LogisticRegression()
clf.fit(x_1_t,y_1)
```

Out[42]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [43]:
```
wt_test_noise=find(clf.coef_[0])[2]
print(wt_test_noise[0:20])
```

```
[-0.07950654 -0.27767067  0.27879303 -1.56715728  0.00895534 -0.39113137
  1.18629003 -0.1553853  -2.48665692  0.56516244  0.16581946 -0.31727856
 -1.31674371 -0.67986191  1.27892829  0.49484372  1.10606222  1.62444072
  0.33085647  1.6653512 ]
```

the differences are not significant ,so the features are not multicollinear .

In [39]:
```
summary=pd.DataFrame({"gridsearchcv":["100","L2"],"randomizedsearchcv":["33","L1"
summary.T
```

Out[39]:

	C	regularization
gridsearchcv	100	L2
randomizedsearchcv	33	L1

Tfidf word2vec

In [30]:
```
#splitting the data
x_1, x_test, y_1, y_test = train_test_split(list_s, d.Score, test_size=0.3, rando
```

In [35]:
```
def tf(l):

    tf=tfidf.fit_transform(d.Text.values)
    tfidfsv = []
    row=0;
    for s in l:
        sum = np.zeros(100)
        i=0;
        for word in s:
            try:
                vec = w2v_model.wv[word]
                tf_idf = tf[row, tfidf_feat.index(word)]
                sum += (vec * tf_idf)
                i += tf_idf
            except:
                pass
        sum /= i
        tfidfsv.append(sum)
        row += 1
    return np.matrix(tfidfsv)
```

In [36]:
```
x_1=tf(x_1)
x_test=tf(x_test)
```

```
C:\Users\himateja\Anaconda3\lib\site-packages\ipykernel_launcher.py:17: Runtime
Warning: invalid value encountered in true_divide
```

In [37]:
```
x_1.shape
```

Out[37]: (254919, 100)

In [38]:
```
x_test.shape
```

Out[38]: (109252, 100)

In [39]:
```
# changing 'NaN' to numeric value
train=np.nan_to_num(x_1)
test=np.nan_to_num(x_test)
```

In [40]:
```
#gridsearchcv
lr = LogisticRegression()
param_grid = {'C':[0.0001,0.001,0.01,0.1,1,10,100,1000],"penalty":["l1","l2"]}
#For time based splitting
t = TimeSeriesSplit(n_splits=5)

gsv = GridSearchCV(lr,param_grid,cv=t,n_jobs=-1,verbose=1,scoring="f1")
gsv.fit(train,y_1)
print("Best HyperParameter: ",gsv.best_params_)
#assinging best alpha to optimal print("Best Accuracy: %.2f%%"%(gsv.best_score_*1
print("best estimator: ",gsv.estimator)
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits

[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   26.3s
[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed:   52.8s finished

Best HyperParameter:  {'C': 0.0001, 'penalty': 'l1'}
best estimator:  LogisticRegression(C=1.0, class_weight=None, dual=False, fit_i
ntercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [41]:
```
gsv.best_score_*100
```

Out[41]:
```
91.47550073824719
```

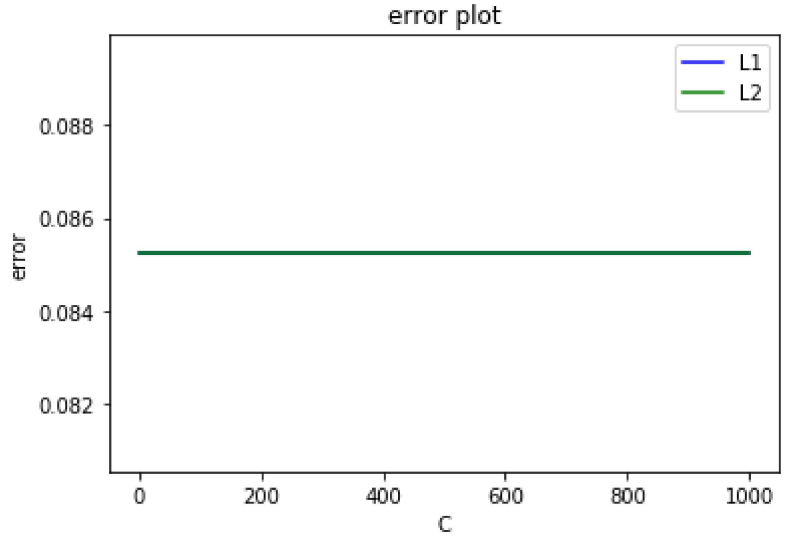
In [42]:
```
gsv.cv_results_
```

```
 'split1_test_score': array([0.91895881, 0.91895881, 0.91895881, 0.91895881,
0.91895881,
        0.91895881, 0.91895881, 0.91895881, 0.91895881, 0.91895881,
        0.91895881, 0.91895881, 0.91895881, 0.91895881, 0.91895881,
        0.91895881]),
 'split2_test_score': array([0.91142484, 0.91142484, 0.91142484, 0.91142484,
0.91142484,
        0.91142484, 0.91142484, 0.91142484, 0.91142484, 0.91142484,
        0.91142484, 0.91142484, 0.91142484, 0.91142484, 0.91142484,
        0.91142484]),
 'split3_test_score': array([0.90923095, 0.90923095, 0.90923095, 0.90923095,
0.90923095,
        0.90923095, 0.90923095, 0.90923095, 0.90923095, 0.90923095,
        0.90923095, 0.90923095, 0.90923095, 0.90923095, 0.90923095,
        0.90923095]),
 'split4_test_score': array([0.90377098, 0.90377098, 0.90377098, 0.90377098,
0.90377098,
        0.90377098, 0.90377098, 0.90377098, 0.90377098, 0.90377098,
        0.90377098, 0.90377098, 0.90377098, 0.90377098, 0.90377098,
        0.90377098]),
```

In [43]:
```
#separating regulerisation values
mean_test_score=gsv.cv_results_["mean_test_score"]
mean_test_score=1-mean_test_score
y_l1=[]
y_l2=[]
for i in mean_test_score[0:16:2]:
    y_l1.append(i)
for i in mean_test_score[1:16:2]:
    y_l2.append(i)
print(y_l1)
print("\n\n")
print(y_l2)
```

```
[0.08524499261752805, 0.08524499261752805, 0.08524499261752805, 0.0852449926175
2805, 0.08524499261752805, 0.08524499261752805, 0.08524499261752805, 0.08524499
261752805]



[0.08524499261752805, 0.08524499261752805, 0.08524499261752805, 0.0852449926175
2805, 0.08524499261752805, 0.08524499261752805, 0.08524499261752805, 0.08524499
261752805]
```

In [44]:
```
#error plot
x=param_grid["C"]
plt.plot(x,y_l1,"b",label="L1")
plt.plot(x,y_l2,"g",label="L2")
plt.xlabel("C")
plt.ylabel("error")
plt.title("error plot")
plt.legend()
plt.show()
```



lets use the optimal value of c and see how it performs

In [45]:
```
lr=LogisticRegression(C=0.0001,penalty="l1")
lr.fit(train,y_1)
```

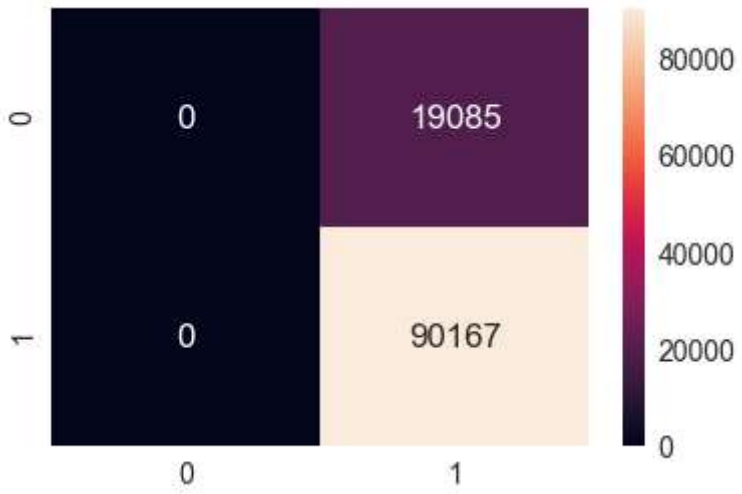
Out[45]:
```
LogisticRegression(C=0.0001, class_weight=None, dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l1', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
```

In [46]:
```
pred=lr.predict(test)
```

In [47]:
```
acc=accuracy_score(y_test,pred)*100
print("\nthe accuracy is %.2f%%"%acc)

re=recall_score(y_test,pred,) * 100
print("\nthe recall is %.2f%%"%re)

pre=precision_score(y_test,pred) * 100
print("the precision is %.2f%%"%pre)

f1=f1_score(y_test,pred) * 100
print("the f1 score is %.2f%%"%f1)

df_cm=pd.DataFrame(confusion_matrix(y_test,pred))
sns.set(font_scale=1.4)
sns.heatmap(df_cm,annot=True,fmt="d")
```

```
the accuracy is 82.53%

the recall is 100.00%
the precision is 82.53%
the f1 score is 90.43%
```

Out[47]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1f915d86b38>
```



In [41]:
```
scoring_data=pd.DataFrame({"accuracy":"82.53%","recall":"100%","precision":"82.53
scoring_data.T
```

Out[41]:

	score
accuracy	82.53%
recall	100%
precision	82.53%
f1	90.43%

lets check whats happening to sparsity with increase in lambda

In [49]:
```
lr=LogisticRegression(C=10,penalty="l1")
lr.fit(train,y_1)
np.count_nonzero(lr.coef_)
```

Out[49]: 0

In [50]:
```
lr=LogisticRegression(C=1,penalty="l1")
lr.fit(train,y_1)
np.count_nonzero(lr.coef_)
```

Out[50]: 0

In [51]:
```
lr=LogisticRegression(C=.1,penalty="l1")
lr.fit(train,y_1)
np.count_nonzero(lr.coef_)
```

Out[51]: 0

In [52]:
```
lr=LogisticRegression(C=.01,penalty="l1")
lr.fit(train,y_1)
np.count_nonzero(lr.coef_)
```

Out[52]: 0

there is no weight in vectors this is the reason it is behaving as dumb model and it is not useful to do pertubation test as there is no weight.

Randomizedsearchcv

In [54]:
```
#randomizedsearchcv
lr = LogisticRegression()
param_grid = {'C':randint(10**-2, 10**2),"penalty":["l1","l2"]}
#For time based splitting
t = TimeSeriesSplit(n_splits=5)

rsv = RandomizedSearchCV(lr,param_grid,cv=t,n_jobs=-1,verbose=1)
rsv.fit(train,y_1)
print("Best HyperParameter: ",rsv.best_params_)
#assinging best alpha to optimal print("Best Accuracy: %.2f%%"%(gsv.best_score_*1
print("best estimator: ",rsv.estimator)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits

[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   24.2s
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:   33.2s finished

Best HyperParameter:  {'C': 95, 'penalty': 'l1'}
best estimator:  LogisticRegression(C=1.0, class_weight=None, dual=False, fit_i
ntercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [40]:
```
summary=pd.DataFrame({"gridsearchcv":["0.0001","L2"],"randomizedsearchcv":["95","
summary.T
```

Out[40]:

	C	regularization
gridsearchcv	0.0001	L2
randomizedsearchcv	95	L1

This model is dumb . it has very poor performance.

conclusion:

1.The sparcity increases with increase in lambda.
2.L1 is very useful when the features are not multicollinear.

In [ ]: