naiveBayes_AmazonFoodProductReview

November 21, 2018

Naive Bayes model implementation for Amazon food product reviews.

```
In [21]: import numpy as np
         import pandas as pd
         from sklearn.feature_extraction.text import CountVectorizer
         from sklearn.feature_extraction.text import TfidfVectorizer
         import sklearn
         from sklearn.naive_bayes import MultinomialNB
         from sklearn.model_selection import train_test_split
         import sqlite3
         import  warnings
         warnings.filterwarnings('ignore')
         import string
         from sklearn.model_selection import TimeSeriesSplit
         from sklearn.model_selection import cross_val_score
         from sklearn import cross_validation
         from sklearn.metrics import accuracy_score
         from sklearn.decomposition import TruncatedSVD
         import matplotlib.pyplot as plt
         import seaborn as sns
         from sklearn import metrics
         from sklearn.metrics import confusion_matrix
         from sklearn.metrics import f1_score
         import pickle
         %matplotlib inline
```

```
In [2]: con = sqlite3.connect('/home/niranjan/Downloads/database.sqlite')
        data = pd.read_sql_query('select * from Reviews where Score!=3 ',con)
        data['Score'] = [1 if i > 3 else 0 for i in data['Score']]
        #print(data.shape)
```

```
In [3]: data_sort = data.sort_values(by='ProductId',ascending=True, inplace=False, kind='quicks
        data_groupby = data_sort[['ProductId','Score']].groupby('Score').count()
        #print(data_groupby)
```

Removing duplicates data

```
In [4]: data_without_dup = data_sort.drop_duplicates(subset={'UserId','ProfileName','Time','Te
        #print(data_without_dup.shape)
```

```
data_groupby = data_without_dup[['ProductId','Score']].groupby('Score').count()
#print(data_groupby)
```

In [5]:
```
data_without_dup = data_without_dup[data_without_dup.HelpfulnessNumerator<=data_withou
#print(data_without_dup.shape)
```

In [6]:
```
import nltk
nltk.download('stopwords')
nltk.download('wordnet')
from nltk.corpus import stopwords
from nltk import WordNetLemmatizer
stop = set(stopwords.words('english'))
lemma = nltk.WordNetLemmatizer()
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/niranjan/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /home/niranjan/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

In [7]:
```
import re
def cleanhtml(words):
  tag = re.compile(r'<.?>')
  cleanSent = re.sub(tag,'',words)
  return cleanSent

def PuncRemov(words):
  tag = re.compile(r'[^a-zA-Z]')
  cleanSent = re.sub(tag,'',words)
  return cleanSent
```

****----------------------data cleaning----------------------****

---

removal of html tags, symbols other than alphabets, stopwords and performing lemmatization as part of data pre-processing.

Lemmatization :- is the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma, or dictionary form.**

In [8]:
```
final_string = []
str1 = ' '
positive_word = []
negative_word = []
i=0
```

```
for sen in data_without_dup['Text'].values:
  filtered_word = []
  sent = cleanhtml(sen)
  for word in sent.split():
    cleanWord = PuncRemov(word)
    for cleaned_words in cleanWord.split():
      if ((len(cleaned_words) > 2) & (cleaned_words.isalpha())):
        if (cleaned_words.islower() not in stop):
          w = (lemma.lemmatize(cleaned_words.lower())).encode('utf8')
          filtered_word.append(w)
          if data_without_dup['Score'].values[i]=='positive':
            positive_word.append(w)
          else:
            negative_word.append(w)
        else:
          continue
      else:
        continue
  str1 = b" ".join(filtered_word)
  final_string.append(str1)
  i = i+1
```

In [9]:
```
data_without_dup['cleaned_text'] = final_string
data_without_dup['cleaned_text'] = data_without_dup['cleaned_text'].str.decode('utf8')
```

sorting data based on time in ascending order

In [10]:
```
data_without_dup = data_without_dup.sort_values(by='Time',ascending=True,inplace=Fals
```

In [11]:
```
X = data_without_dup['cleaned_text']
y = data_without_dup['Score']
```

In [87]:
```
X_train= X[0:250000]
y_train = y[0:250000]
X_cv= X[250000:270000]
y_cv = y[250000:270000]
X_test = X[270000:290000]
y_test = y[270000:290000]
```

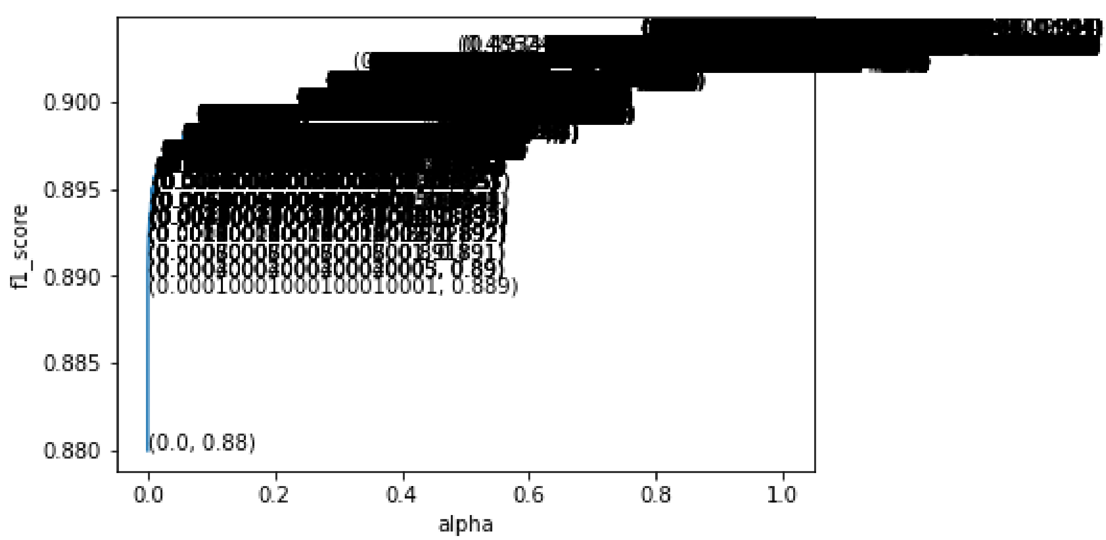
BOW as Vectorizer

In [14]:
```
count_vect = CountVectorizer()
X_train_bow = count_vect.fit_transform(X_train)
X_cv_bow = count_vect.transform(X_cv)
X_test_bow = count_vect.transform(X_test)
```

Best alpha fit using Cross validation

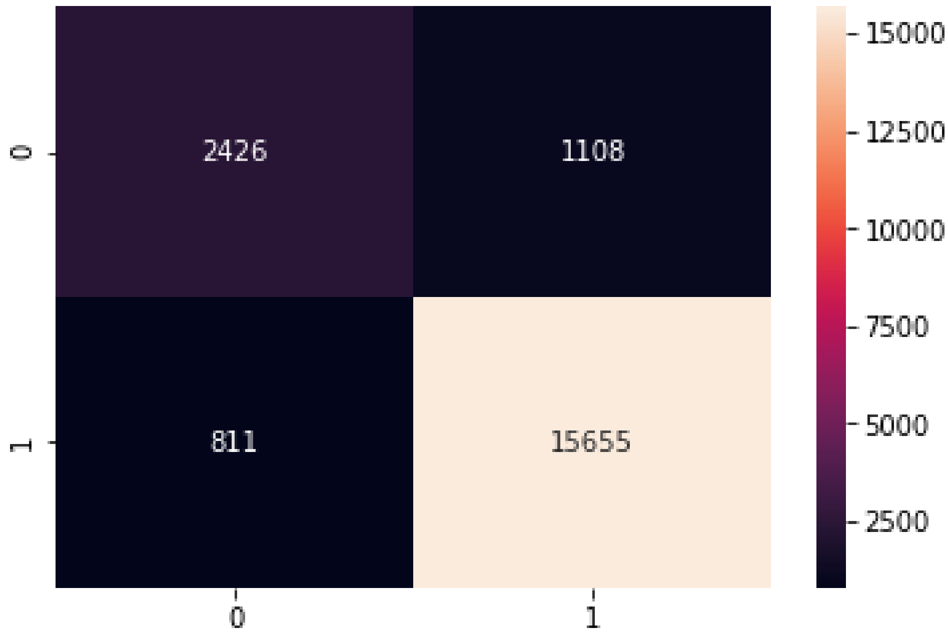
```
In [38]: param = np.linspace(0,1,10000)
         #print(param)

         cv_scores = []
         for val in param:
             classifier = MultinomialNB(alpha=val)
             classifier.fit(X_train_bow,y_train)
             y_pred = classifier.predict(X_cv_bow)
             cv_scores.append(f1_score(y_pred,y_cv,average='micro'))


         #Determining optimal value of apha

         optimal_alpha = param[cv_scores.index(max(cv_scores))]
         print("optimal number of alpha is: ", optimal_alpha)

         plt.plot(param,cv_scores)

         for xy in zip(param, np.round(cv_scores,3)):
             plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')
         plt.xlabel('alpha')
         plt.ylabel('f1_score')
         plt.show()

         #print("the f1_score for each alpha value is : ", np.round(cv_scores,3))
```

```
optimal number of alpha is:  0.7923792379237924
```

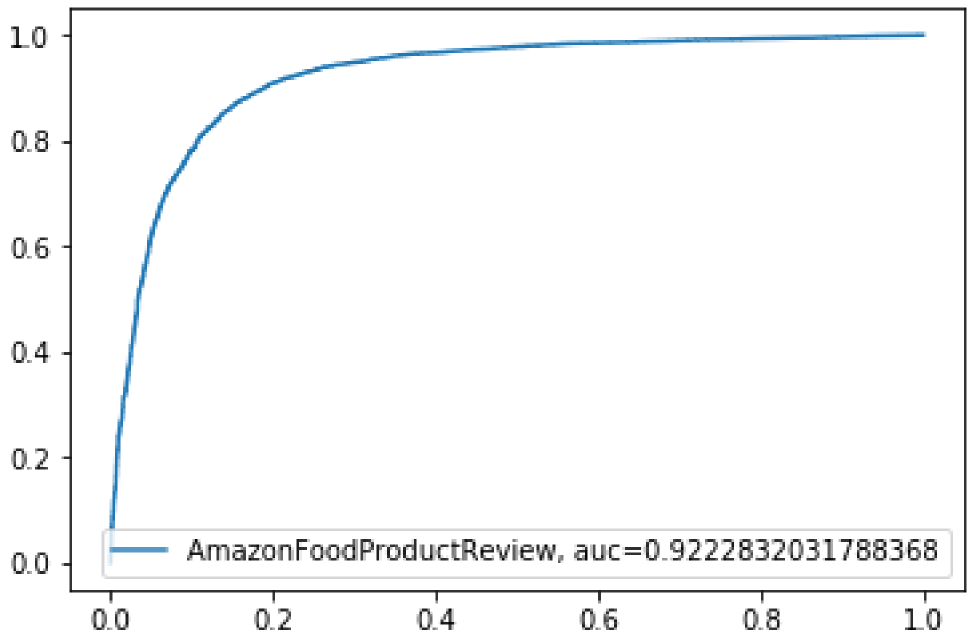


In [77]:
```
clf = MultinomialNB(alpha=optimal_alpha)
clf.fit(X_train_bow,y_train)
pred = clf.predict(X_test_bow)
# evaluate accuracy
acc = f1_score(y_test, pred,average='micro')*100
print("The accuracy of the multimonialNB classifier for alpha {a} is {b}%".format(a =
print("---------------------------------------------------------")
class_labels = clf.classes_
feature_names =count_vect.get_feature_names()
topn_class1 = sorted(zip(clf.predict_log_proba(X_test_bow)[:,0], feature_names))[:10]
topn_class2 = sorted(zip(clf.predict_log_proba(X_test_bow)[:,1], feature_names))[:10]
print("Important words in negative reviews")
for coef, feat in topn_class1:
    print(class_labels[0], coef, feat)
print("-----------------------------------------")
print("Important words in positive reviews")
for coef, feat in topn_class2:
    print(class_labels[1], coef, feat)
tn,fp,fn,tp = confusion_matrix(y_test,pred).ravel()
df = pd.DataFrame(confusion_matrix(y_test,pred))
sns.heatmap(df,annot=True,fmt="d")
plt.show()
```

```
The accuracy of the multimonialNB classifier for alpha 0.7923792379237924 is 90.405%
---------------------------------------------------------
Important words in negative reviews
0 -227.14954512541772 breakfastmorning
0 -193.00439882773208 bottler
0 -162.0512254941732 areeven
0 -111.79294854915304 barscookiescakesmeatloavesbr
0 -108.94069355239935 beetljuice
0 -106.93009055314678 briannabr
0 -103.12694280755295 broccolihaters
0 -88.09768252082904 bitteracidic
0 -86.76064505100476 accountit
0 -83.78868009150756 amt
-----------------------------------------
Important words in positive reviews
1 -148.63024181920946 author
1 -148.13608061836158 attitudebr
1 -148.13608061836158 aux
1 -124.51900736951302 buyingmove
1 -121.91314757824148 attitude
1 -118.57365795156738 barsbr
1 -73.83664242421673 bjuice
1 -71.59713280601363 airfreshners
1 -63.17997228995955 agescons
1 -59.14661111178657 biy
```




```
In [78]: y_pred_proba = classifier.predict_proba(X_test_bow)[::,1]
         fpr, tpr, thresholds = metrics.roc_curve(y_test,  y_pred_proba)
         auc = metrics.roc_auc_score(y_test, y_pred_proba)
         plt.plot(fpr,tpr,label="AmazonFoodProductReview, auc="+str(auc))
         plt.legend(loc=4)
         plt.show()
```




```
In [73]: precision = (tp/tp+fp)
         recall = tp/fn+tp
         print("precision is :",precision)
         print("recall is :",recall)
```

```
precision is : 1109.0
recall is : 15674.303329223181
```

tf-idf as Vectorizer

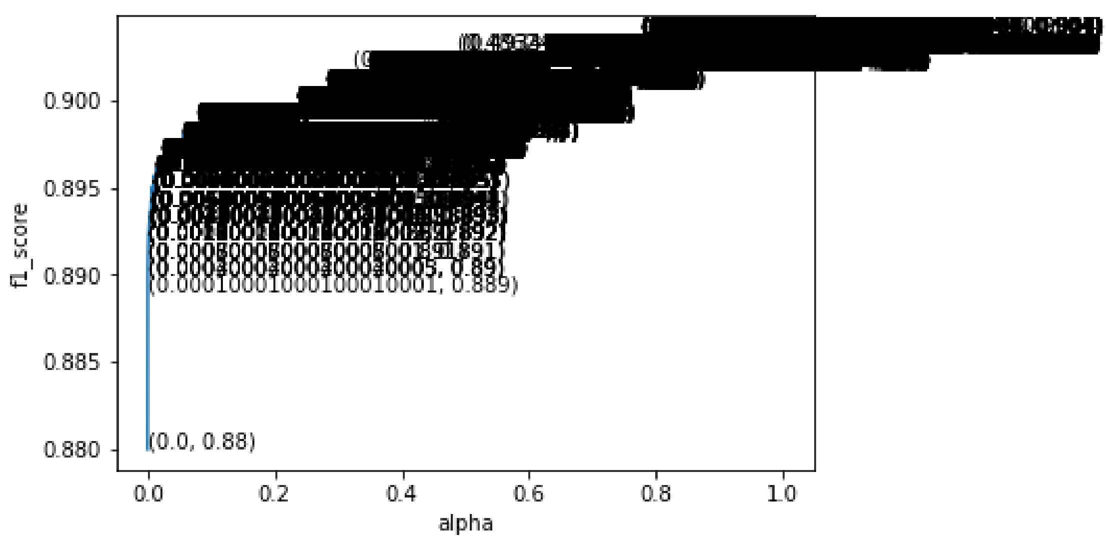
```
In [88]: vec = TfidfVectorizer()
```

```
In [94]: tfidf_vect = CountVectorizer()
         X_train_tfidf = tfidf_vect.fit_transform(X_train)
         X_cv_tfidf = tfidf_vect.transform(X_cv)
         X_test_tfidf = tfidf_vect.transform(X_test)
```

```
In [95]: param = np.linspace(0,1,10000)
         #print(param)

         cv_scores = []
         for val in param:
             classifier = MultinomialNB(alpha=val)
             classifier.fit(X_train_tfidf,y_train)
             y_pred = classifier.predict(X_cv_tfidf)
```

```
        cv_scores.append(f1_score(y_pred,y_cv,average='micro'))


    #Determining optimal value of apha

    optimal_alpha_tfidf = param[cv_scores.index(max(cv_scores))]
    print("optimal number of alpha is: ", optimal_alpha_tfidf)

    plt.plot(param,cv_scores)

    for xy in zip(param, np.round(cv_scores,3)):
        plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')
    plt.xlabel('alpha')
    plt.ylabel('f1_score')
    plt.show()

    print("the f1_score for each alpha value is : ", np.round(cv_scores,3))
```

```
optimal number of alpha is:  0.7923792379237924
```

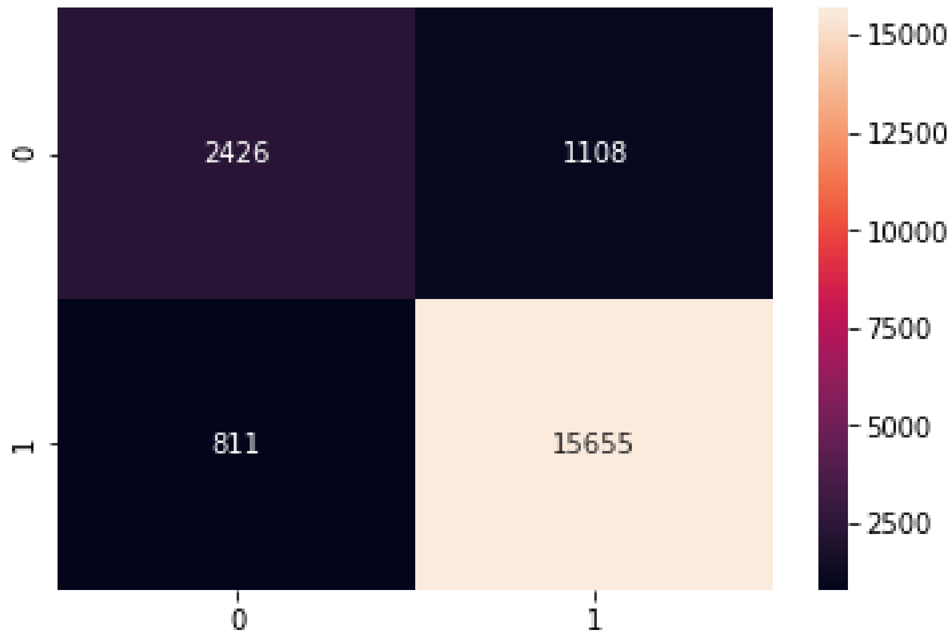



```
the f1_score for each alpha value is :  [0.88  0.889 0.89  ... 0.904 0.904 0.904]
```

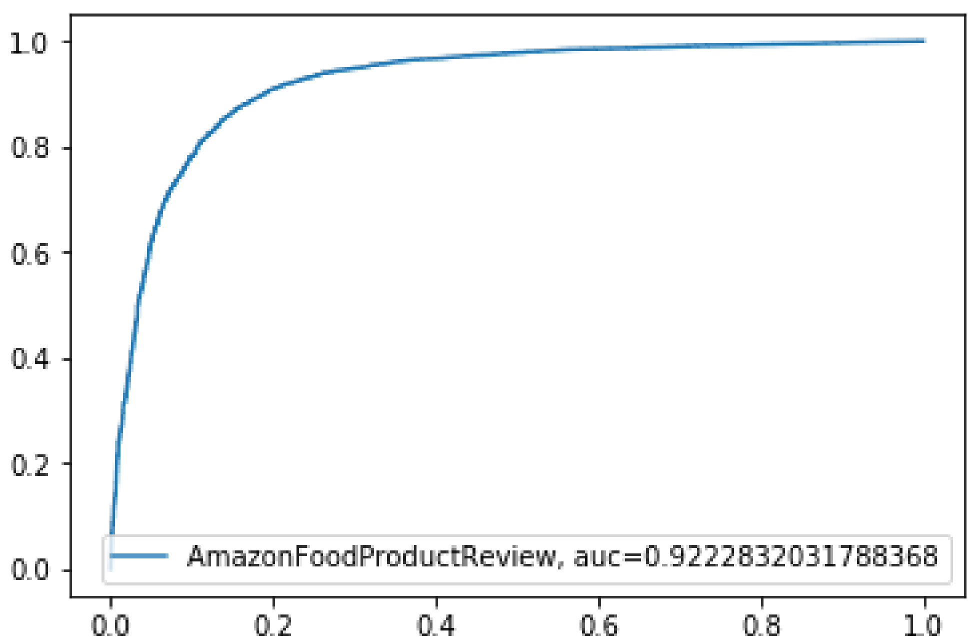
In [96]:
```
clf = MultinomialNB(alpha=optimal_alpha_tfidf)
clf.fit(X_train_tfidf,y_train)
pred = clf.predict(X_test_tfidf)
# evaluate accuracy
acc = f1_score(y_test, pred,average='micro')*100
```

```
    print("The accuracy of the multimonialNB classifier for alpha {a} is {b}%".format(a =
    print("---------------------------------------------------------")
    class_labels = clf.classes_
    feature_names =count_vect.get_feature_names()
    topn_class1 = sorted(zip(clf.predict_log_proba(X_test_tfidf)[:,0], feature_names))[:1
    topn_class2 = sorted(zip(clf.predict_log_proba(X_test_tfidf)[:,1], feature_names))[:1
    print("Important words in negative reviews")
    for coef, feat in topn_class1:
        print(class_labels[0], coef, feat)
    print("-----------------------------------------")
    print("Important words in positive reviews")
    for coef, feat in topn_class2:
        print(class_labels[1], coef, feat)
    tn,fp,fn,tp = confusion_matrix(y_test,pred).ravel()
    df = pd.DataFrame(confusion_matrix(y_test,pred))
    sns.heatmap(df,annot=True,fmt="d")
    plt.show()
```

```
The accuracy of the multimonialNB classifier for alpha 0.7923792379237924 is 90.405%
---------------------------------------------------------
Important words in negative reviews
0 -227.14954512541772 breakfastmorning
0 -193.00439882773208 bottler
0 -162.0512254941732 areeven
0 -111.79294854915304 barscookiescakesmeatloavesbr
0 -108.94069355239935 beetljuice
0 -106.93009055314678 briannabr
0 -103.12694280755295 broccolihaters
0 -88.09768252082904 bitteracidic
0 -86.76064505100476 accountit
0 -83.78868009150756 amt
-----------------------------------------
Important words in positive reviews
1 -148.63024181920946 author
1 -148.13608061836158 attitudebr
1 -148.13608061836158 aux
1 -124.51900736951302 buyingmove
1 -121.91314757824148 attitude
1 -118.57365795156738 barsbr
1 -73.83664242421673 bjuice
1 -71.59713280601363 airfreshners
1 -63.17997228995955 agescons
1 -59.14661111178657 biy
```




```
In [97]: y_pred_proba = classifier.predict_proba(X_test_tfidf)[::,1]
         fpr, tpr, thresholds = metrics.roc_curve(y_test,  y_pred_proba)
         auc = metrics.roc_auc_score(y_test, y_pred_proba)
         plt.plot(fpr,tpr,label="AmazonFoodProductReview, auc="+str(auc))
         plt.legend(loc=4)
         plt.show()
```



In [98]:
```
precision = (tp/tp+fp)
recall = tp/fn+tp
print("precision is :",precision)
print("recall is :",recall)
```

```
precision is : 1109.0
recall is : 15674.303329223181
```

In [101]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.add_column("important_features_class_[0]",topn_class1)
x.add_column("important_features_class_[1]",topn_class2)
print(x)
y = PrettyTable()
y.field_names=["Vectorizer","Model","Alpha","f1_score","Precision","recall"]
y.add_row(["BOW","MultinomialNB","0.7923792379237924","0.90405","1109.0","15674.30332
y.add_row(["tfidf","MultinomialNB","0.7923792379237924","0.90405","1109.0","15674.303
print(y)
```

```
+--------------------------------------------------------+------------------------------------
|              important_features_class_[0]              |     important_features_class_[1]
+--------------------------------------------------------+------------------------------------
|       (-227.14954512541772, 'breakfastmorning')        |   (-148.63024181920946, 'author')
|            (-193.00439882773208, 'bottler')            | (-148.13608061836158, 'attitudebr')
|            (-162.0512254941732, 'areeven')             |     (-148.13608061836158, 'aux')
| (-111.79294854915304, 'barscookiescakesmeatloavesbr') | (-124.51900736951302, 'buyingmove')
|           (-108.94069355239935, 'beetljuice')          |  (-121.91314757824148, 'attitude')
|           (-106.93009055314678, 'briannabr')           |   (-118.57365795156738, 'barsbr')
|         (-103.12694280755295, 'broccolihaters')        |    (-73.83664242421673, 'bjuice')
|           (-88.09768252082904, 'bitteracidic')         | (-71.59713280601363, 'airfreshners')
|            (-86.76064505100476, 'accountit')           |   (-63.17997228995955, 'agescons')
|              (-83.78868009150756, 'amt')               |     (-59.14661111178657, 'biy')
+--------------------------------------------------------+------------------------------------
+------------+---------------+--------------------+----------+-----------+--------------------
| Vectorizer |     Model     |       Alpha        | f1_score | Precision |       recall
+------------+---------------+--------------------+----------+-----------+--------------------
|    BOW     | MultinomialNB | 0.7923792379237924 | 0.90405  |   1109.0  | 15674.303329223181
|   tfidf    | MultinomialNB | 0.7923792379237924 | 0.90405  |   1109.0  | 15674.303329223181
+------------+---------------+--------------------+----------+-----------+--------------------
```