# Naive Bayes on Amazon Food Reviews Dataset

## Using Binary Bag of Words(BoW) and TFIDF Techniques

Data Source: [https://www.kaggle.com/snap/amazon-fine-food-reviews (https://www.kaggle.com/snap/amazon-fine-food-reviews)](https://www.kaggle.com/snap/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. index
2. Id
3. ProductId - unique identifier for the product
4. UserId - unqiue identifier for the user
5. ProfileName
6. HelpfulnessNumerator - number of users who found the review helpful
7. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
8. Score - rating between 1 and 5
9. Time - timestamp for the review
10. Summary - brief summary of the review
11. Text - text of the review
12. ProcessedText - Cleaned & Preprocessed Text of the review

**Objective: Given Amazon Food reviews, convert all the reviews into a vector using binary BoW and tfidf techniques and then apply forward chaining cross validation to determine the right value of alpha. Finally apply Naive Bayes on the top of it. Find top 10 important features. Also calculate followings performance metric scores: Accuracy, precision, recall, F1 Score and confusion metric.**

[Q] How to determine if a review is positive or negative?

[Ans] We could use the Score/Rating. A rating of 4 or 5 could be cosnidered a positive review. A review of 1 or 2 could be considered negative. A review of 3 is nuetral and ignored. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

Loading the data

SQLite Database

In order to load the data, We have used the SQLITE dataset as it easier to query the data and visualise the data efficiently. Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score id above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

In [1]:

```
import sqlite3
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

from sklearn.naive_bayes import BernoulliNB, MultinomialNB

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

from wordcloud import WordCloud
```

In [2]:

```
connection = sqlite3.connect('FinalAmazonFoodReviewsDataset.sqlite')
```

In [3]:

```
data = pd.read_sql_query("SELECT * FROM Reviews", connection)
```

In [4]:
```
data.head()
```

Out[4]:

| | index | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | HelpfulnessDenominator | Score | Time | Summary | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 1 | B001E4KFG0 | A3SGXH7AUHU8GW | delmartian | 1 | 1 | Positive | 1303862400 | Good Quality Dog Food | se |
| **1** | 1 | 2 | B00813GRG4 | A1D87F6ZCVE5NK | dll pa | 0 | 0 | Negative | 1346976000 | Not as Advertised | lal |
| **2** | 2 | 3 | B000LQOCH0 | ABXLMWJIXXAIN | Natalia Corres "Natalia Corres" | 1 | 1 | Positive | 1219017600 | "Delight" says it all | F . cc a |
| **3** | 4 | 5 | B006K2ZZ7K | A1UQRSCLF8GW1T | Michael D. Bigham "M. Wassir" | 0 | 0 | Positive | 1350777600 | Great taffy | t |
| **4** | 5 | 6 | B006K2ZZ7K | ADT0SRK1MGOEU | Twoapennything | 0 | 0 | Positive | 1342051200 | Nice Taffy | . |

In [5]:
```
data.shape
```

Out[5]: (364171, 12)

In [6]:
```
data["Score"].value_counts()
```

Out[6]:
```
Positive    307061
Negative     57110
Name: Score, dtype: int64
```

In [7]:
```
def changingScores(score):
    if score == "Positive":
        return 1
    else:
        return 0
```

In [8]:
```
# changing score
# Positive = 1
# Negative = 0
actualScore = list(data["Score"])
positiveNegative = list(map(changingScores, actualScore)) #map(function, list of numbers)
data['Score'] = positiveNegative
```

In [9]:
```
data.head()
```

Out[9]:

| | index | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | HelpfulnessDenominator | Score | Time | Summary | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 1 | B001E4KFG0 | A3SGXH7AUHU8GW | delmartian | 1 | 1 | 1 | 1303862400 | Good Quality Dog Food | I b seve V ca |
| **1** | 1 | 2 | B00813GRG4 | A1D87F6ZCVE5NK | dll pa | 0 | 0 | 0 | 1346976000 | Not as Advertised | Pr a label J S Pea |
| **2** | 2 | 3 | B000LQOCH0 | ABXLMWJIXXAIN | Natalia Corres "Natalia Corres" | 1 | 1 | 1 | 1219017600 | "Delight" says it all | Thi conf tha aro |
| **3** | 4 | 5 | B006K2ZZ7K | A1UQRSCLF8GW1T | Michael D. Bigham "M. Wassir" | 0 | 0 | 1 | 1350777600 | Great taffy | taff |
| **4** | 5 | 6 | B006K2ZZ7K | ADT0SRK1MGOEU | Twoapennything | 0 | 0 | 1 | 1342051200 | Nice Taffy | I wil fo or tl |

In [10]:
```
allPositiveReviews = data[(data["Score"] == 1)]
```

In [11]:
```
allPositiveReviews.shape
```

Out[11]:
```
(307061, 12)
```

In [12]:
```
positiveReviews_30000 = allPositiveReviews[:30000]
```

In [13]:
```
positiveReviews_30000.shape
```

Out[13]:
```
(30000, 12)
```

In [14]:
```
positiveReviews_30000.head()
```

Out[14]:

| | index | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | HelpfulnessDenominator | Score | Time | Summary | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 1 | B001E4KFG0 | A3SGXH7AUHU8GW | delmartian | 1 | 1 | 1 | 1303862400 | Good Quality Dog Food | I bo seve V ca |
| **2** | 2 | 3 | B000LQOCH0 | ABXLMWJIXXAIN | Natalia Corres "Natalia Corres" | 1 | 1 | 1 | 1219017600 | "Delight" says it all | Thi confe tha arou |
| **3** | 4 | 5 | B006K2ZZ7K | A1UQRSCLF8GW1T | Michael D. Bigham "M. Wassir" | 0 | 0 | 1 | 1350777600 | Great taffy | ( taffy I T v |
| **4** | 5 | 6 | B006K2ZZ7K | ADT0SRK1MGOEU | Twoapennything | 0 | 0 | 1 | 1342051200 | Nice Taffy | I wilc for orc th |
| **5** | 6 | 7 | B006K2ZZ7K | A1SP2KVKFXXRU1 | David C. Sullivan | 0 | 0 | 1 | 1340150400 | Great! Just as good as the expensive brands! | salty taffy fla and v |

In [15]:
```
allNegativeReviews = data[(data["Score"] == 0)]
```

In [16]:
```
allNegativeReviews.shape
```
Out[16]:
```
(57110, 12)
```

In [17]:
```
negativeReviews_30000 = allNegativeReviews[:30000]
```

In [18]:
```
negativeReviews_30000.shape
```
Out[18]:
```
(30000, 12)
```

In [19]:
```
negativeReviews_30000.head()
```

Out[19]:

| | index | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | HelpfulnessDenominator | Score | Time | Summary | T |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 1 | 2 | B00813GRG4 | A1D87F6ZCVE5NK | dll pa | 0 | 0 | 0 | 1346976000 | Not as Advertised | Prod arri labe Jun Sal Peanu |
| **11** | 12 | 13 | B0009XLVG0 | A327PCT23YH90 | LT | 1 | 1 | 0 | 1339545600 | My Cats Are Not Fans of the New Food | My c ha be hap eat Felic Pla |
| **15** | 16 | 17 | B001GVISJM | A3KLWF6WQ5BNYO | Erica Neathery | 0 | 0 | 0 | 1348099200 | poor taste | I lo eat th and tl are go watc |
| **25** | 26 | 27 | B001GVISJM | A3RXAU2N8KV45G | lady21 | 0 | 1 | 0 | 1332633600 | Nasty No flavor | - cand just re No fla . J pla |
| **45** | 47 | 51 | B001EO5QW8 | A108P30XVUFKXY | Roberto A | 0 | 7 | 0 | 1203379200 | Don't like it | T oatm is good. mus so |

In [20]:
```
frames_60000 = [positiveReviews_30000, negativeReviews_30000]
```

In [21]:
```
FinalPositiveNegative = pd.concat(frames_60000)
```

In [22]:
```
FinalPositiveNegative.shape
```

Out[22]:
```
(60000, 12)
```

In [23]:
```
#Sorting FinalDataframe by "Time"
FinalSortedPositiveNegative_60000 = FinalPositiveNegative.sort_values('Time', axis=0, ascending=True, inplace=False)
```

In [24]:
```
FinalSortedPositiveNegativeScore_60000 = FinalSortedPositiveNegative_60000["Score"]
```

In [25]:
```
FinalSortedPositiveNegative_60000.shape
```

Out[25]:
```
(60000, 12)
```

In [26]:
```
FinalSortedPositiveNegativeScore_60000.shape
```

Out[26]:
```
(60000,)
```

In [27]:
```
FinalSortedPositiveNegative_60000.head()
```

Out[27]:

| | index | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | HelpfulnessDenominator | Score | Time | Summa |
|---|---|---|---|---|---|---|---|---|---|---|
| **772** | 1146 | 1245 | B00002Z754 | A29Z5PI9BW2PU3 | Robbie | 7 | 7 | 1 | 961718400 | Gre Produ |
| **771** | 1145 | 1244 | B00002Z754 | A3B8RCEI0FXFI6 | B G Chase | 10 | 10 | 1 | 962236800 | WOW Ma your ov 'slickers |
| **19460** | 28087 | 30630 | B00008RCMI | A284C7M23F0APC | A. Mendoza | 0 | 0 | 1 | 1067040000 | Be sugarle gum eve |
| **19459** | 28086 | 30629 | B00008RCMI | A19E94CF5O1LY7 | Andrew Arnold | 0 | 0 | 1 | 1067040000 | I've chew this gu many time but use |
| **85400** | 121056 | 131233 | B00004RAMX | A1PYZPS1QYR036 | Kazantzakis "hinterlands" | 5 | 8 | 0 | 1067385600 | Woodstrea Gopher Tr 06 |

In [28]:
```
Final_Data = FinalSortedPositiveNegative_60000
```

In [29]:
```
Final_Data.shape
```
Out[29]:
```
(60000, 12)
```

In [30]:
```
Final_Data_Labels = FinalSortedPositiveNegativeScore_60000
```

In [31]:
```
Final_Data_Labels.shape
```
Out[31]:
```
(60000,)
```

# Binary BoW

In [32]:
```
positive_reviews = Final_Data[(Final_Data["Score"] == 1)]
negative_reviews = Final_Data[(Final_Data["Score"] == 0)]
```

In [33]:
```
positive_reviews.shape, negative_reviews.shape
```
Out[33]:
```
((30000, 12), (30000, 12))
```

In [34]:
```
positive_bow_vect =  CountVectorizer(stop_words = "english")
positive_bow = positive_bow_vect.fit_transform(positive_reviews["ProcessedText"].values)
```

In [35]:
```
positive_bow.shape
```
Out[35]:
```
(30000, 20285)
```

In [36]:
```
features_positive = positive_bow_vect.get_feature_names()
len(features_positive), type(features_positive)
```
Out[36]:
```
(20285, list)
```

In [37]:
```
count = []
for i in range(len(features_positive)):
    total = positive_bow.getcol(i).sum() # it will give sum of all the values in 'i'th column
    count.append(total)
```

In [38]:
```
positive_dict = dict(zip(features_positive, count))
```

In [39]:
```
sortedDict_Positive = sorted(positive_dict.items(), key = lambda positive_dict: positive_dict[1], reverse = True)
```

In [40]:
```
for i in range(200):
    print(sortedDict_Positive[i])
```

```
('like', 13895)
('tast', 12635)
('good', 11457)
('love', 10510)
('flavor', 10373)
('great', 10208)
('use', 9922)
('veri', 8959)
('tri', 8710)
('product', 8691)
('just', 8638)
('tea', 8437)
('coffe', 7759)
('make', 7602)
('food', 6632)
('time', 5416)
('eat', 5313)
('buy', 5269)
('realli', 5183)
('onli', 5119)
('dog', 5041)
('price', 4705)
('amazon', 4644)
('best', 4612)
('littl', 4501)
('chocol', 4492)
('dont', 4443)
('drink', 4361)
('order', 4336)
('becaus', 4037)
('ive', 3990)
('mix', 3987)
('treat', 3931)
('store', 3818)
('bag', 3785)
('better', 3560)
('recommend', 3489)
('ani', 3423)
```

```
('day', 3333)
('year', 3306)
('sugar', 3303)
('sweet', 3164)
('cup', 3163)
('high', 3098)
('want', 2943)
('box', 2922)
('look', 2889)
('water', 2876)
('enjoy', 2875)
('work', 2838)
('brand', 2808)
('delici', 2797)
('bar', 2674)
('nice', 2664)
('favorit', 2618)
('cat', 2617)
('bit', 2607)
('think', 2605)
('way', 2603)
('add', 2590)
('need', 2583)
('packag', 2571)
('purchas', 2560)
('sinc', 2468)
('perfect', 2429)
('thing', 2389)
('bought', 2381)
('pack', 2348)
('lot', 2325)
('free', 2310)
('snack', 2295)
('know', 2293)
('differ', 2291)
('say', 2289)
('mani', 2244)
('come', 2239)
('easi', 2202)
('milk', 2193)
('hot', 2182)
('organ', 2178)
```

```
('alway', 2078)
('everi', 2055)
('fresh', 2045)
('review', 1996)
('wonder', 1951)
('ship', 1931)
('got', 1925)
('calori', 1902)
('local', 1892)
('stuff', 1880)
('right', 1877)
('qualiti', 1870)
('doe', 1861)
('healthi', 1838)
('oil', 1837)
('regular', 1826)
('befor', 1799)
('natur', 1775)
('ingredi', 1771)
('small', 1752)
('definit', 1743)
('did', 1737)
('cooki', 1734)
('someth', 1717)
('size', 1710)
('sauc', 1705)
('ad', 1688)
('bread', 1680)
('cook', 1671)
('feel', 1637)
('hard', 1637)
('tasti', 1631)
('quick', 1624)
('excel', 1605)
('sure', 1586)
('dark', 1579)
('doesnt', 1578)
('butter', 1572)
('long', 1571)
('chicken', 1561)
('month', 1551)
('happi', 1548)
```

```
('help', 1529)
('serv', 1521)
('didnt', 1508)
('strong', 1497)
('actual', 1484)
('big', 1484)
('start', 1483)
('far', 1472)
('contain', 1471)
('quit', 1438)
('problem', 1437)
('howev', 1433)
('thank', 1433)
('rice', 1431)
('old', 1428)
('bake', 1409)
('dri', 1403)
('pretti', 1390)
('bottl', 1386)
('usual', 1375)
('diet', 1373)
('peanut', 1356)
('textur', 1355)
('low', 1352)
('fat', 1348)
('salt', 1347)
('candi', 1332)
('smell', 1331)
('bean', 1318)
('chip', 1291)
('peopl', 1286)
('open', 1244)
('friend', 1239)
('real', 1227)
('groceri', 1221)
('green', 1220)
('fruit', 1212)
('juic', 1212)
('famili', 1207)
('anoth', 1195)
('blend', 1191)
('ice', 1191)
```

```
('anyth', 1187)
('new', 1174)
('worth', 1169)
('chew', 1165)
('varieti', 1165)
('gluten', 1162)
('roast', 1161)
('protein', 1160)
('expens', 1158)
('meal', 1147)
('arriv', 1142)
('minut', 1142)
('nut', 1139)
('save', 1132)
('sever', 1132)
('thought', 1122)
('morn', 1121)
('ginger', 1109)
('item', 1102)
('light', 1101)
('kid', 1097)
('week', 1087)
('recip', 1081)
('expect', 1080)
('came', 1078)
('half', 1075)
('home', 1068)
('onc', 1056)
('prefer', 1049)
('reason', 1048)
('amaz', 1047)
('receiv', 1039)
('larg', 1028)
('theyr', 1028)
('cereal', 1025)
('absolut', 1023)
```

In [44]:
```
def PlotWordCloud(frequency):
    worcloudPlot = WordCloud(background_color="white", width=1500, height=1000)
    worcloudPlot.generate_from_frequencies(frequencies=frequency)
    plt.figure(figsize=(15,10))
    plt.imshow(worcloudPlot, interpolation="bilinear")
    plt.axis("off")
    plt.show()
```

In [45]:
```
PlotWordCloud(positive_dict)
```


**This is a Word Cloud for all the positive reviews in the corpus.**
**This Word Cloud plot correponds to the most frequent words in all positive reviews.**

How I have plotted this word cloud. Now BoW representation is based on the count of the word in a document. If a word Wi occurs in many document then the sum of its column will be high. Therefore, I have just calculated the sum of all the column and created the dictionary where keys are the features and values are the sum of that column. I feeded this dictionary to the wordcloud and plotted the same. the same procedure is followed for negative reviews as well

In [46]:
```
negative_bow_vect =  CountVectorizer(stop_words = "english")
negative_bow = negative_bow_vect.fit_transform(negative_reviews["ProcessedText"].values)
```

In [47]:
```
negative_bow.shape
```

Out[47]:
```
(30000, 21669)
```

In [48]:
```
features_negative = negative_bow_vect.get_feature_names()
len(features_negative), type(features_negative)
```

Out[48]:
```
(21669, list)
```

In [49]:
```
count = []
for i in range(len(features_negative)):
    total = negative_bow.getcol(i).sum() # it will give sum of all the values in 'i'th column
    count.append(total)
```

In [50]:
```
negative_dict = dict(zip(features_negative, count))
```

In [51]:
```
sortedDict_Negative = sorted(negative_dict.items(), key = lambda negative_dict: negative_dict[1], reverse = True)
```

In [52]:
```
for i in range(200):
    print(sortedDict_Negative[i])
```

```
('tast', 18519)
('like', 17349)
('product', 14820)
('flavor', 10387)
('just', 9860)
('tri', 9551)
('veri', 8848)
('good', 8065)
('use', 8041)
('coffe', 7995)
('buy', 7165)
('food', 6949)
('order', 6571)
('dont', 6387)
('tea', 5895)
('box', 5586)
('becaus', 5462)
('amazon', 5284)
('onli', 5245)
('time', 5220)
('make', 5208)
('eat', 5192)
('bag', 5151)
('dog', 5103)
('realli', 5075)
('look', 4843)
('love', 4635)
('packag', 4621)
('review', 4408)
('purchas', 4207)
('did', 4153)
('bought', 4046)
('ani', 3995)
('bad', 3983)
('better', 3870)
('chocol', 3841)
('disappoint', 3812)
('want', 3798)
```

```
('drink', 3771)
('water', 3730)
('think', 3694)
('price', 3554)
('say', 3505)
('know', 3490)
('didnt', 3460)
('ingredi', 3370)
('smell', 3268)
('brand', 3202)
('sugar', 3175)
('great', 3141)
('way', 3139)
('thought', 3125)
('got', 3100)
('ive', 3089)
('littl', 3077)
('someth', 3011)
('store', 2877)
('thing', 2863)
('receiv', 2828)
('befor', 2762)
('money', 2755)
('differ', 2729)
('item', 2724)
('open', 2696)
('mix', 2691)
('cup', 2680)
('pack', 2659)
('day', 2626)
('sweet', 2536)
('doe', 2483)
('year', 2467)
('contain', 2374)
('cooki', 2348)
('stuff', 2335)
('howev', 2334)
('sinc', 2319)
('recommend', 2311)
('ship', 2309)
('away', 2307)
('work', 2276)
```

```
('compani', 2257)
('old', 2242)
('doesnt', 2219)
('dri', 2192)
('sure', 2161)
('cat', 2139)
('bottl', 2124)
('come', 2088)
('expect', 2076)
('actual', 2065)
('mani', 2064)
('hard', 2063)
('lot', 2042)
('bar', 2040)
('anoth', 1988)
('return', 1986)
('treat', 1985)
('hope', 1964)
('new', 1955)
('qualiti', 1954)
('read', 1932)
('natur', 1910)
('wast', 1904)
('need', 1903)
('problem', 1903)
('mayb', 1859)
('bit', 1827)
('small', 1827)
('star', 1818)
('peopl', 1764)
('whi', 1763)
('wont', 1755)
('noth', 1746)
('high', 1744)
('organ', 1738)
('list', 1702)
('anyth', 1686)
('textur', 1675)
('month', 1655)
('oil', 1652)
('candi', 1644)
('said', 1632)
```

```
('arriv', 1593)
('piec', 1567)
('hot', 1557)
('local', 1557)
('real', 1556)
('ill', 1554)
('free', 1544)
('milk', 1523)
('best', 1521)
('green', 1508)
('feel', 1506)
('enjoy', 1504)
('end', 1481)
('salt', 1467)
('label', 1465)
('ad', 1464)
('stick', 1451)
('coconut', 1433)
('right', 1432)
('case', 1426)
('chang', 1421)
('instead', 1418)
('sauc', 1418)
('bitter', 1403)
('strong', 1399)
('pretti', 1386)
('bean', 1372)
('came', 1370)
('sever', 1351)
('big', 1346)
('kind', 1344)
('chew', 1343)
('worth', 1342)
('regular', 1336)
('half', 1331)
('fresh', 1322)
('gave', 1318)
('start', 1314)
('week', 1308)
('far', 1307)
('everi', 1295)
('size', 1294)
```

```
('chip', 1289)
('color', 1286)
('wasnt', 1270)
('probabl', 1247)
('terribl', 1246)
('fruit', 1244)
('care', 1238)
('horribl', 1234)
('juic', 1231)
('chicken', 1224)
('date', 1222)
('peanut', 1216)
('unfortun', 1210)
('isnt', 1197)
('guess', 1188)
('cereal', 1187)
('aw', 1181)
('plastic', 1180)
('usual', 1177)
('butter', 1175)
('quit', 1175)
('calori', 1170)
('rice', 1170)
('second', 1164)
('corn', 1154)
('throw', 1153)
('notic', 1145)
('went', 1145)
('wouldnt', 1142)
('save', 1134)
('cost', 1130)
('long', 1127)
('mouth', 1126)
('leav', 1125)
('nice', 1125)
('custom', 1121)
```

In [53]:
```
PlotWordCloud(negative_dict)
```


This is a Word Cloud for all the negative reviews in the corpus.
This Word Cloud plot correponds to the most frequent words in all negative reviews.

## Converting Rows to binary BoW

In [32]:
```
count_vect = CountVectorizer(binary=True)
```

In [33]:
```
Final_Data_BoW = count_vect.fit_transform(Final_Data["ProcessedText"].values)
```

In [34]:
```
Final_Data_BoW.shape
```

Out[34]:
```
(60000, 29748)
```

In [35]:
```
print(type(Final_Data_BoW))
```

```
<class 'scipy.sparse.csr.csr_matrix'>
```

## Applying Forward chaining cross validation to find best value of alpha

In [36]:
```
alphaValue = []
for i in range(1, 500, 2):
    alphaValue.append(i)
```

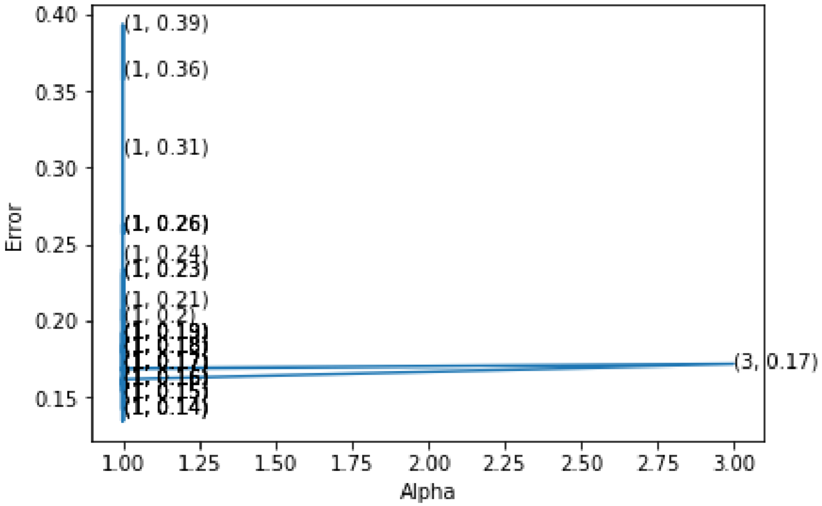
In [37]:
```
# Applying forward chaining cross validation on 80% of data, means 48000 rows. Rest 12000 will be test data.
end1 = 1000
end2 = 2000
FinalScoreOfScores = []

for i in range(47):
    train = Final_Data_BoW[0:end1]
    train_labels = Final_Data_Labels[0:end1]
    test = Final_Data_BoW[end1:end2]
    test_labels = Final_Data_Labels[end1:end2]

    scoreOfScores = []
    for a in alphaValue:
        clf = BernoulliNB(alpha=a, binarize=None, class_prior=None, fit_prior=True)
        clf.fit(train, train_labels)
        score = clf.score(test, test_labels)
        scoreOfScores.append(score)
    FinalScoreOfScores.append(scoreOfScores)

    end1 += 1000
    end2 += 1000
```

In [42]:
```
#finding best alpha using brute force
maximum = FinalScoreOfScores[0][0]

for i in range(len(FinalScoreOfScores)):
    Alpha = -1
    for j in range(len(FinalScoreOfScores[i])):
        Alpha += 2
        if maximum < FinalScoreOfScores[i][j]:
            maximum = FinalScoreOfScores[i][j]
            best_alpha = Alpha

print("Maximum Score = "+str(maximum))
print("Value of best alpha = "+str(best_alpha))
```

```
Maximum Score = 0.865
Value of best alpha = 1
```

In [43]:
```
error = []
maxAlpha = []
splitNumber = []

for i in range(len(FinalScoreOfScores)):
    a = -1
    maximum = -10          #any small negative number can be given
    for j in range(len(FinalScoreOfScores[i])):
        a += 2
        if maximum < FinalScoreOfScores[i][j]:
            maximum = FinalScoreOfScores[i][j]
            max_alpha = a

    error.append(1- maximum)
    maxAlpha.append(max_alpha)
    splitNumber.append(i+1)
```

In [44]:
```
plt.plot(maxAlpha, error)

minError1 = min(error)
minError = np.round(minError1, 2)

plt.xlabel("Alpha")
plt.ylabel("Error")
for xy in zip(maxAlpha, np.round(error,2)):
        plt.annotate(xy,xy)

plt.show()
```



In [45]:
```
for xy in zip(maxAlpha, np.round(error,2), splitNumber):
        print(xy)
```

```
(1, 0.36, 1)
(1, 0.39, 2)
(1, 0.31, 3)
(1, 0.26, 4)
(1, 0.26, 5)
(1, 0.26, 6)
(1, 0.24, 7)
(1, 0.2, 8)
(1, 0.23, 9)
(1, 0.16, 10)
(1, 0.21, 11)
(1, 0.17, 12)
(1, 0.19, 13)
(1, 0.18, 14)
(1, 0.18, 15)
(1, 0.19, 16)
(1, 0.19, 17)
(1, 0.19, 18)
(1, 0.18, 19)
(1, 0.17, 20)
(1, 0.19, 21)
(1, 0.23, 22)
(1, 0.18, 23)
(1, 0.18, 24)
(1, 0.18, 25)
(1, 0.17, 26)
(1, 0.15, 27)
(1, 0.17, 28)
(1, 0.17, 29)
(1, 0.17, 30)
(1, 0.15, 31)
(1, 0.14, 32)
(1, 0.16, 33)
(1, 0.16, 34)
(1, 0.15, 35)
(1, 0.14, 36)
(1, 0.15, 37)
(1, 0.16, 38)
```

```
(1, 0.16, 39)
(1, 0.14, 40)
(1, 0.14, 41)
(1, 0.19, 42)
(1, 0.17, 43)
(1, 0.15, 44)
(1, 0.17, 45)
(3, 0.17, 46)
(1, 0.16, 47)
```

In forward chaining cross validation that has been applied above--in this dataset--contains 47 splits.

Splits are as follows:

train0: 0:1000 test0: 1000:2000

train1: 0:2000 test1: 2000:3000

train2: 0:3000 test2: 3000:4000

train3: 0:4000 test3: 4000:5000

. .

. .

. .

train46: 0:47000 test46: 47000:48000

Now from every split, maximum score is taken and corresponding "alpha" value is taken. So, we will have 47 maximum scores and 47 corresponding alpha values.

Finally, we have plotted them above.

**We can also see above that error is decreasing from split1 towards split47**

## Applying Naive Bayes

In [46]:
```
clf = BernoulliNB(alpha=best_alpha, binarize=None, class_prior=None, fit_prior=True)
clf.fit(Final_Data_BoW[0:48000], Final_Data_Labels[0:48000])

prediction = clf.predict(Final_Data_BoW[48000:60000])

score = clf.score(Final_Data_BoW[48000:60000], Final_Data_Labels[48000:60000])
print(score)
```

```
0.8388333333333333
```

## Accuracy

In [47]:
```
Accuracy = accuracy_score(Final_Data_Labels[48000:60000], prediction) * 100
print("Accuracy = "+str(Accuracy)+"%")
```

```
Accuracy = 83.88333333333333%
```

## Precision

In [48]:
```
Precision = precision_score(Final_Data_Labels[48000:60000], prediction)
print("Precision Score = "+str(Precision))
```

```
Precision Score = 0.7919807538091419
```

## Recall

In [49]:
```
Recall = recall_score(Final_Data_Labels[48000:60000], prediction)
print("Recall Score = "+str(Recall))
```

```
Recall Score = 0.8857399103139013
```

## F1 Score

In [50]:
```
F1_Score = f1_score(Final_Data_Labels[48000:60000], prediction)
print("F1 Score = "+str(F1_Score))
```

```
F1 Score = 0.8362404741744284
```

## Confusion Matrix

In [51]:
```
Confusion_Matrix = confusion_matrix(Final_Data_Labels[48000:60000], prediction)
print("Confusion Matrix  \n"+str(Confusion_Matrix))
```

```
Confusion Matrix
[[5128 1297]
 [ 637 4938]]
```

In [52]:
```
tn, fp, fn, tp = confusion_matrix(Final_Data_Labels[48000:60000], prediction).ravel()
tn, fp, fn, tp
```

Out[52]:
```
(5128, 1297, 637, 4938)
```

## Getting top 10 Best features

In [56]:
```
featuresNames = count_vect.get_feature_names()
len(featuresNames)
```

Out[56]:
```
29748
```

In [58]:
```
featuresProbabilities = clf.feature_log_prob_
featuresProbabilities.shape
```

Out[58]:
```
(2, 29748)
```

In [59]:
```
featuresProbabilitiesTrans = featuresProbabilities.T
featuresProbabilitiesTrans.shape
```

Out[59]:
```
(29748, 2)
```

In [74]:
```
features_prob_dataFrame = pd.DataFrame(featuresProbabilitiesTrans, index = None, columns = ["Class 0 Probability Scores",
features_prob_dataFrame["Feature Names"] = featuresNames
```

In [77]:
```
impFeatures_Class0 = features_prob_dataFrame.sort_values(by = 'Class 0 Probability Scores', axis = 0, ascending = False)
impFeatures_Class0.drop(["Class 1 Probability Scores"], axis = 1, inplace = True)
impFeatures_Class0.head(10)
```

Out[77]:

| | Class 0 Probability Scores | Feature Names |
|---|---|---|
| **26097** | -0.130283 | the |
| **963** | -0.260555 | and |
| **26231** | -0.382136 | this |
| **17661** | -0.582558 | not |
| **3558** | -0.688957 | but |
| **9798** | -0.705567 | for |
| **28562** | -0.740971 | was |
| **26085** | -0.742930 | that |
| **11737** | -0.922865 | have |
| **25768** | -0.952217 | tast |

**Here, above we got all the probability scores of all the features given class label 0. They are nothing but "likelihood" probabilities of all the features given class label 0. Here, in above dataframe values corresponding to column "Class 0" has all the probability scores of features given class 0. The above dataFrame has been sorted according to the probability scores of Class 0. On Features column we got important feature names for points belong to class label 0.**

In [78]:
```
impFeatures_Class1 = features_prob_dataFrame.sort_values(by = 'Class 1 Probability Scores', axis = 0, ascending = False)
impFeatures_Class1.drop(["Class 0 Probability Scores"], axis = 1, inplace = True)
impFeatures_Class1.head(10)
```

Out[78]:

| | Class 1 Probability Scores | Feature Names |
|---|---|---|
| **26097** | -0.196612 | the |
| **963** | -0.202510 | and |
| **26231** | -0.450536 | this |
| **9798** | -0.607925 | for |
| **11737** | -0.900229 | have |
| **29103** | -0.910963 | with |
| **26085** | -0.917089 | that |
| **3558** | -0.936325 | but |
| **1297** | -1.108279 | are |
| **29536** | -1.108651 | you |

**Here, above we got all the probability scores of all the features given class label 1. They are nothing but "likelihood" probabilities of all the features given class label 1. Here, in above dataframe values corresponding to column "Class 1" has all the probability scores of features given class 1. The above dataFrame has been sorted according to the probability scores of Class 1. On Features column we got important feature names for points belong to class label 1.**

# TFIDF

In [79]:
```
positiveReviews_5000 = allPositiveReviews[:5000]
```

In [80]:
```
positiveReviews_5000.shape
```

Out[80]:
```
(5000, 12)
```

In [81]:
```
positiveReviews_5000.head()
```

Out[81]:

| | index | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | HelpfulnessDenominator | Score | Time | Summary | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 1 | B001E4KFG0 | A3SGXH7AUHU8GW | delmartian | 1 | 1 | 1 | 1303862400 | Good Quality Dog Food | I bo seve V ca |
| **2** | 2 | 3 | B000LQOCH0 | ABXLMWJIXXAIN | Natalia Corres "Natalia Corres" | 1 | 1 | 1 | 1219017600 | "Delight" says it all | Thi confe tha arou |
| **3** | 4 | 5 | B006K2ZZ7K | A1UQRSCLF8GW1T | Michael D. Bigham "M. Wassir" | 0 | 0 | 1 | 1350777600 | Great taffy | ( taffy I T v |
| **4** | 5 | 6 | B006K2ZZ7K | ADT0SRK1MGOEU | Twoapennything | 0 | 0 | 1 | 1342051200 | Nice Taffy | I wil fo orc th |
| **5** | 6 | 7 | B006K2ZZ7K | A1SP2KVKFXXRU1 | David C. Sullivan | 0 | 0 | 1 | 1340150400 | Great! Just as good as the expensive brands! | salty taffy fl and v |

In [82]:
```
negativeReviews_5000 = allNegativeReviews[:5000]
```

In [83]:
```
negativeReviews_5000.shape
```

Out[83]:
```
(5000, 12)
```

In [84]:
```
negativeReviews_5000.head()
```

Out[84]:

| | index | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | HelpfulnessDenominator | Score | Time | Summary | T |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 1 | 2 | B00813GRG4 | A1D87F6ZCVE5NK | dll pa | 0 | 0 | 0 | 1346976000 | Not as Advertised | Prod<br>arri<br>labe<br><br>Jun<br>Sal<br>Peanu |
| **11** | 12 | 13 | B0009XLVG0 | A327PCT23YH90 | LT | 1 | 1 | 0 | 1339545600 | My Cats Are Not Fans of the New Food | My c<br>ha<br>be<br>hap<br>eat<br>Felic<br>Pla |
| **15** | 16 | 17 | B001GVISJM | A3KLWF6WQ5BNYO | Erica Neathery | 0 | 0 | 0 | 1348099200 | poor taste | I lo<br>eat<br>th<br>and tl<br>are go |
| **25** | 26 | 27 | B001GVISJM | A3RXAU2N8KV45G | lady21 | 0 | 1 | 0 | 1332633600 | Nasty No flavor | watc<br><br>-<br>cand<br>just re<br>No fla<br>. J<br>pla |
| **45** | 47 | 51 | B001EO5QW8 | A108P30XVUFKXY | Roberto A | 0 | 7 | 0 | 1203379200 | Don't like it | T<br>oatm<br>is<br>good.<br>mus<br>so |

In [85]:
```
frames_10000 = [positiveReviews_5000, negativeReviews_5000]
```

In [86]:
```
FinalPositiveNegative = pd.concat(frames_10000)
```

In [87]:
```
FinalPositiveNegative.shape
```

Out[87]:
```
(10000, 12)
```

In [88]:
```
#Sorting FinalDataframe by "Time"
FinalSortedPositiveNegative_10000 = FinalPositiveNegative.sort_values('Time', axis=0, ascending=True, inplace=False)
```

In [89]:
```
FinalSortedPositiveNegativeScore_10000 = FinalSortedPositiveNegative_10000["Score"]
```

In [90]:
```
FinalSortedPositiveNegative_10000.shape
```

Out[90]:
```
(10000, 12)
```

In [91]:
```
FinalSortedPositiveNegativeScore_10000.shape
```

Out[91]:
```
(10000,)
```

In [92]:
```
FinalSortedPositiveNegative_10000.head()
```

Out[92]:

| | index | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | HelpfulnessDenominator | Score | Time | Summary |
|---|---|---|---|---|---|---|---|---|---|---|
| **772** | 1146 | 1245 | B00002Z754 | A29Z5PI9BW2PU3 | Robbie | 7 | 7 | 1 | 961718400 | Great Product |
| **771** | 1145 | 1244 | B00002Z754 | A3B8RCEI0FXFI6 | B G Chase | 10 | 10 | 1 | 962236800 | WOW Make your own 'slickers' ! |
| **5822** | 7427 | 8111 | B0000EIE2Z | A3M174IC0VXOS2 | Gail Cooke | 3 | 3 | 1 | 1075420800 | BEST BLUEBERRIES |
| **2418** | 3481 | 3783 | B00016UX0K | AF1PV3DIC0XM7 | Robert Ashton | 1 | 2 | 1 | 1081555200 | Classic Condiment |
| **5206** | 6790 | 7432 | B0001E1IME | A2IKCTD1I73PLW | Adeba | 2 | 8 | 1 | 1083456000 | amazon monopoly/ripoff |

In [93]:
```
Final_Data = FinalSortedPositiveNegative_10000
```

In [94]:
```
Final_Data.shape
```

Out[94]:
```
(10000, 12)
```

In [95]:
```
Final_Data_Labels = FinalSortedPositiveNegativeScore_10000
```

In [96]:
```
Final_Data_Labels.shape
```
Out[96]:
```
(10000,)
```

In [72]:
```
positive_reviews = Final_Data[(Final_Data["Score"] == 1)]
negative_reviews = Final_Data[(Final_Data["Score"] == 0)]
```

In [73]:
```
Positive_tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), stop_words = "english")
Positive_tf_idf = Positive_tf_idf_vect.fit_transform(positive_reviews["ProcessedText"].values)
```

In [74]:
```
Positive_tf_idf.shape
```
Out[74]:
```
(5000, 114651)
```

In [75]:
```
features = Positive_tf_idf_vect.get_feature_names()
```

In [76]:
```
idfValues = Positive_tf_idf_vect.idf_
```

In [78]:
```
d = dict(zip(features, 9 - idfValues))
```

In [79]:
```
sortedDict = sorted(d.items(), key = lambda d: d[1], reverse = True)
```

In [80]:
```
for i in range(200):
    print(sortedDict[i])
```

```
('like', 6.809072442421418)
('tast', 6.765368008186689)
('great', 6.763992492151379)
('good', 6.73466078043391)
('love', 6.689725684788852)
('flavor', 6.590032302691801)
('use', 6.568508292946707)
('just', 6.50636578331954)
('veri', 6.503690792870237)
('product', 6.470097075582087)
('tri', 6.467323148699362)
('make', 6.391361607896317)
('buy', 6.114608605976726)
('onli', 6.072907876777782)
('time', 6.060468186302143)
('realli', 6.049279258384337)
('best', 5.968767617525185)
('price', 5.95030555468545)
('littl', 5.909095286038787)
('dont', 5.886181026515912)
('order', 5.886181026515912)
('amazon', 5.849077276312534)
('store', 5.833492545295836)
('eat', 5.817661080079155)
('coffe', 5.8141086784747875)
('becaus', 5.806965790962407)
('recommend', 5.797964830103432)
('better', 5.790705270090627)
('mix', 5.726773729244833)
('ive', 5.650123319469437)
('ani', 5.631075124498743)
('high', 5.624644234168453)
('food', 5.594074168083775)
('bag', 5.591854411345462)
('drink', 5.580681110747337)
('year', 5.569381555493403)
('delici', 5.544063747509114)
('want', 5.529979007627374)
```

```
('work', 5.515693050379898)
('look', 5.501200043077331)
('favorit', 5.474071375689078)
('pack', 5.47156824547096)
('nice', 5.458957737879031)
('enjoy', 5.448753567704788)
('sweet', 5.441031521610878)
('day', 5.433249381168823)
('purchas', 5.428027437187671)
('sugar', 5.3879686766356665)
('say', 5.379760696217836)
('tea', 5.379760696217836)
('perfect', 5.377009662845947)
('snack', 5.377009662845947)
('brand', 5.371484786913977)
('bought', 5.3603426103607354)
('easi', 5.357537559433126)
('way', 5.349074885514393)
('sinc', 5.337678750783524)
('mani', 5.3290456036388205)
('thing', 5.3290456036388205)
('fresh', 5.302689758933458)
('need', 5.302689758933458)
('cup', 5.281699483041622)
('differ', 5.281699483041622)
('bit', 5.278664579346469)
('lot', 5.278664579346469)
('box', 5.269504209947804)
('think', 5.269504209947804)
('free', 5.254047951711112)
('packag', 5.247797931365941)
('add', 5.2094545761682935)
('come', 5.2094545761682935)
('regular', 5.2094545761682935)
('wonder', 5.1963396340904655)
('chip', 5.189717093329972)
('water', 5.189717093329972)
('alway', 5.166186595919777)
('local', 5.1627794375981635)
('everi', 5.155930095752589)
('got', 5.152487751561615)
('right', 5.135096008849747)
```

```
('know', 5.131581066742303)
('ship', 5.12451389951921)
('tasti', 5.120961497914842)
('calori', 5.113818610402461)
('review', 5.077318208182936)
('definit', 5.066103137362795)
('hard', 5.058555931727413)
('excel', 5.050951332342193)
('did', 5.039434890280634)
('befor', 5.031682913476317)
('qualiti', 5.027784273060659)
('healthi', 5.000059725045804)
('natur', 5.000059725045804)
('sure', 5.000059725045804)
('stuff', 4.975668271921645)
('famili', 4.971544554737783)
('high recommend', 4.967403762071751)
('someth', 4.963245751923088)
('ad', 4.954877502252571)
('doe', 4.954877502252571)
('chocol', 4.950666969716227)
('hot', 4.950666969716227)
('long', 4.933645282146797)
('happi', 4.929344200247407)
('small', 4.911952457535537)
('thank', 4.911952457535537)
('treat', 4.898707230785517)
('doesnt', 4.894252880436136)
('milk', 4.894252880436136)
('ingredi', 4.876234374933458)
('serv', 4.867101891370185)
('actual', 4.857885236265261)
('feel', 4.857885236265261)
('far', 4.853244856708759)
('howev', 4.843898994290521)
('month', 4.839193103253109)
('problem', 4.839193103253109)
('size', 4.839193103253109)
('didnt', 4.824941080545907)
('quick', 4.820144908282414)
('light', 4.800726822425313)
('usual', 4.800726822425313)
```

```
('pancak', 4.795812807622884)
('veri good', 4.795812807622884)
('help', 4.7859117366401716)
('groceri', 4.780924195129133)
('expens', 4.770873859275632)
('varieti', 4.770873859275632)
('start', 4.750464987644424)
('contain', 4.745297017485981)
('old', 4.745297017485981)
('pretti', 4.745297017485981)
('salt', 4.745297017485981)
('quit', 4.740102200608877)
('bake', 4.734880256627726)
('cook', 4.724353843640738)
('friend', 4.724353843640738)
('big', 4.719048791411046)
('gluten', 4.719048791411046)
('sauc', 4.7137154454356835)
('came', 4.7083535022942975)
('worth', 4.697542586190082)
('juic', 4.692092981422517)
('thought', 4.686613515657892)
('dog', 4.664390378873181)
('pleas', 4.664390378873181)
('recip', 4.664390378873181)
('peopl', 4.653090823619248)
('reason', 4.653090823619248)
('open', 4.64739280250461)
('arriv', 4.641662127795625)
('bean', 4.641662127795625)
('kid', 4.641662127795625)
('low', 4.641662127795625)
('abl', 4.635898423078875)
('anyth', 4.635898423078875)
('home', 4.635898423078875)
('case', 4.6301013053945494)
('new', 4.6301013053945494)
('altern', 4.618405265631358)
('groceri store', 4.618405265631358)
('strong', 4.606570807984355)
('expect', 4.600600640997851)
('diet', 4.5945946169376395)
```

```
('save', 4.588552302481677)
('tast like', 4.588552302481677)
('husband', 4.582473256405295)
('real', 4.582473256405295)
('anoth', 4.576357029387859)
('bad', 4.576357029387859)
('absolut', 4.57020316381348)
('organ', 4.57020316381348)
('tast great', 4.564011193565559)
('item', 4.551511030801327)
('especi', 4.538852633929404)
('smell', 4.532462835830634)
('textur', 4.532462835830634)
('chicken', 4.519559430994725)
('instead', 4.519559430994725)
('sever', 4.519559430994725)
('week', 4.513044749973531)
('oil', 4.506487349427372)
('amaz', 4.499886665396021)
('blend', 4.486553134526556)
('fruit', 4.486553134526556)
('theyr', 4.486553134526556)
('bottl', 4.479819102345211)
('butter', 4.479819102345211)
('kind', 4.479819102345211)
('onc', 4.479819102345211)
('orang', 4.479819102345211)
('wont', 4.459340571001671)
('cat', 4.445451458841004)
('fat', 4.445451458841004)
('prefer', 4.445451458841004)
('ago', 4.438433886182358)
('avail', 4.438433886182358)
('gluten free', 4.438433886182358)
('soda', 4.438433886182358)
('wish', 4.438433886182358)
```

In [82]:
```
PlotWordCloud(d)
```


**This is a Word Cloud for all the positive reviews in the corpus.**
**This Word Cloud plot correponds to the most frequent words in all positive reviews based on IDF values. More is the IDF Value for a word the less frequent is the word in the corpus.**

How I have plotted this word cloud. Now formulae for IDF(D,Wi) = (ln(N+1 / ni+1) + 1) where 'N' is total number of documents in a corpus and 'ni' is the total number of documents where word 'Wi' occurs. Hence, I got all the idf values from idf_ attribute and I got corresponding features from get_features_names() function. Now since, the highest possible idf value can be 8.88, hence, I subtracted all the idf values from '9' which leads to the highest idf value of the most frequently occuring word. Now I created dictionary where features are the keys and modified idf value are the values and I feeded this to the word cloud and plot the same. The same is done for negative reviews as well.

In [83]:
```
Negative_tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), stop_words = "english")
Negative_tf_idf = Negative_tf_idf_vect.fit_transform(negative_reviews["ProcessedText"].values)
```

In [84]:
```
Negative_tf_idf.shape
```

Out[84]:
```
(5000, 139161)
```

In [85]:
```
features_neg = Negative_tf_idf_vect.get_feature_names()
```

In [86]:
```
negIDF = Negative_tf_idf_vect.idf_
```

In [87]:
```
NegD = dict(zip(features_neg, 9 - negIDF))
```

In [88]:
```
sortedDictNeg = sorted(NegD.items(), key = lambda NegD: NegD[1], reverse = True)
```

In [89]:
```
for i in range(200):
    print(sortedDictNeg[i])
```

```
('tast', 7.0050070599682215)
('like', 7.004466080782725)
('product', 6.762615081465285)
('just', 6.6496447054933165)
('veri', 6.595748937288184)
('tri', 6.565994676206391)
('good', 6.44957396719508)
('flavor', 6.4170040385096545)
('use', 6.376263183183732)
('buy', 6.343270500029383)
('dont', 6.271578571573267)
('becaus', 6.170961542527858)
('onli', 6.1211746177476165)
('make', 6.113290214223468)
('time', 6.082477327793933)
('order', 6.054889371275104)
('realli', 6.047871798616457)
('look', 6.009101688151887)
('love', 5.951857145376868)
('food', 5.95030555468545)
('eat', 5.9314962227279535)
('bought', 5.923553369214017)
('disappoint', 5.889486814650411)
('ani', 5.862729365480861)
('packag', 5.861033012232683)
('better', 5.8473575854330075)
('review', 5.8473575854330075)
('did', 5.835236224900663)
('think', 5.828243189409692)
('purchas', 5.822966132308848)
('amazon', 5.810543612310291)
('box', 5.794341637734011)
('want', 5.783392623244341)
('bad', 5.781556075437039)
('say', 5.751703112287357)
('didnt', 5.742188292646019)
('know', 5.7287135940626595)
('thought', 5.720931453620604)
```

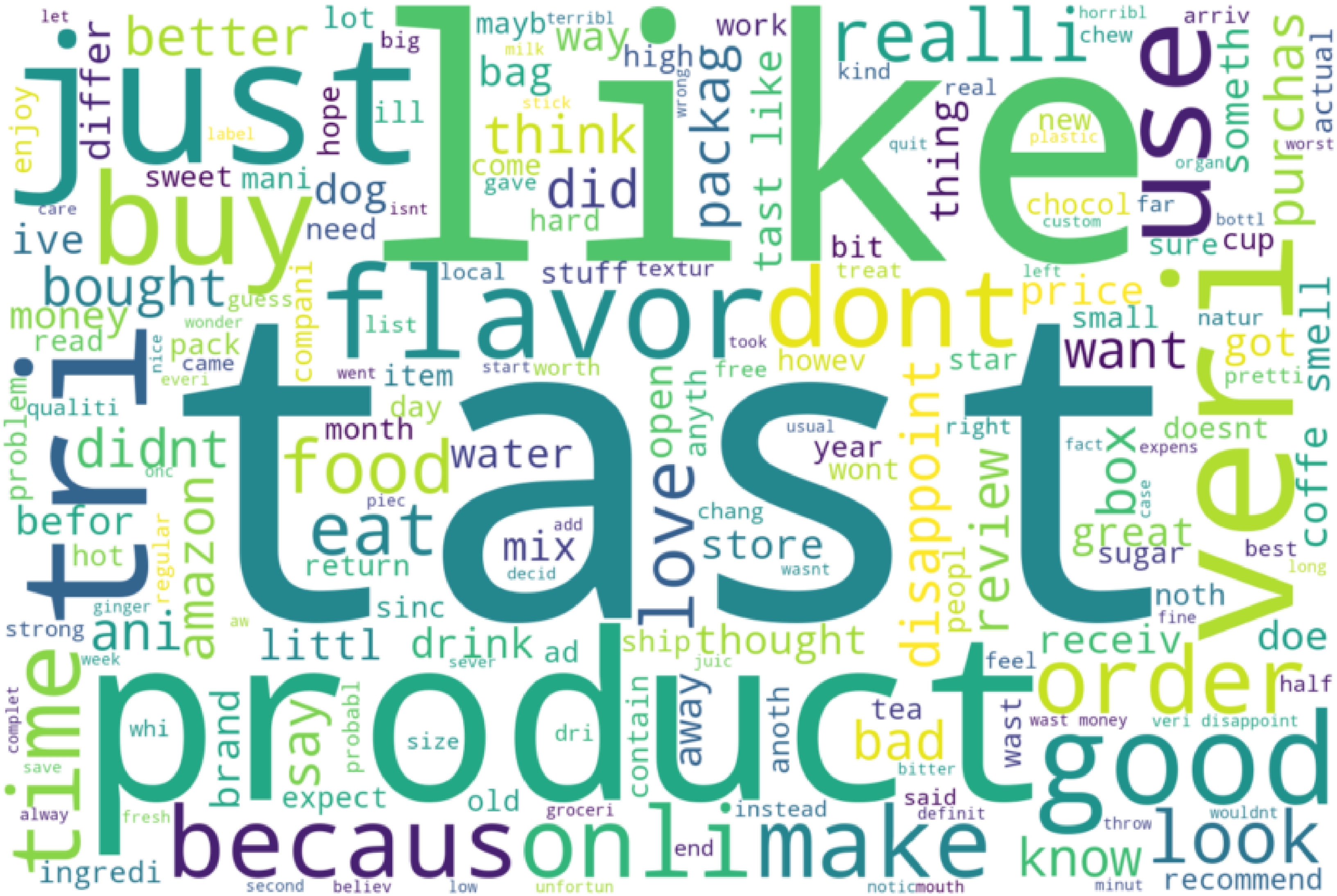
```
('bag', 5.713088276159578)
('drink', 5.674969318055968)
('great', 5.66881545248159)
('way', 5.645921632615737)
('price', 5.618171719662835)
('littl', 5.613833318064237)
('got', 5.609476012695281)
('ive', 5.5761765986262315)
('someth', 5.5761765986262315)
('coffe', 5.564825738957542)
('tast like', 5.557952859669781)
('thing', 5.555651362681502)
('water', 5.548714918684844)
('dog', 5.544063747509114)
('store', 5.520477748503234)
('mix', 5.513292088842359)
('money', 5.5036301779306225)
('befor', 5.484021706542246)
('receiv', 5.484021706542246)
('differ', 5.456416440450358)
('open', 5.443612168204369)
('smell', 5.443612168204369)
('away', 5.441031521610878)
('brand', 5.438444198045927)
('doe', 5.4358501628688805)
('ingredi', 5.430641817761742)
('recommend', 5.409532854551507)
('tea', 5.3879686766356665)
('item', 5.382504182163588)
('sugar', 5.382504182163588)
('howev', 5.365929217069375)
('sinc', 5.357537559433126)
('cup', 5.349074885514393)
('old', 5.337678750783524)
('sweet', 5.331931608527956)
('come', 5.314489305864614)
('doesnt', 5.314489305864614)
('year', 5.311552446191303)
('pack', 5.305652724064116)
('work', 5.302689758933458)
('day', 5.29074931856154)
('sure', 5.284725203958159)
```

```
('need', 5.266432010910833)
('chocol', 5.263350344373426)
('mani', 5.250927824374869)
('actual', 5.244658211361273)
('hope', 5.222399740760331)
('anoth', 5.219179126060288)
('expect', 5.2094545761682935)
('mayb', 5.193033845955966)
('hard', 5.189717093329972)
('lot', 5.183050401971783)
('ship', 5.166186595919777)
('stuff', 5.159360630849378)
('problem', 5.155930095752589)
('return', 5.152487751561615)
('compani', 5.149033516693528)
('contain', 5.145567308717043)
('wast', 5.135096008849747)
('enjoy', 5.131581066742303)
('anyth', 5.117396431750346)
('bit', 5.106624334768435)
('high', 5.0993779262476675)
('month', 5.095734934969167)
('peopl', 5.088408894877094)
('ill', 5.084725649460797)
('small', 5.077318208182936)
('wont', 5.073593809091953)
('hot', 5.069855486981346)
('noth', 5.069855486981346)
('new', 5.054760860758861)
('star', 5.050951332342193)
('ad', 5.04712723590379)
('read', 5.031682913476317)
('qualiti', 5.019941095599632)
('whi', 5.015996317308616)
('strong', 5.00805976771288)
('treat', 4.996035574746079)
('natur', 4.971544554737783)
('dri', 4.963245751923088)
('textur', 4.963245751923088)
('local', 4.946438633606706)
('right', 4.946438633606706)
('pretti', 4.9379279439387975)
```

```
('said', 4.933645282146797)
('kind', 4.929344200247407)
('big', 4.916328832135337)
('gave', 4.916328832135337)
('best', 4.903141827853382)
('real', 4.889778600041215)
('worth', 4.889778600041215)
('chew', 4.885284210453376)
('regular', 4.885284210453376)
('chang', 4.871678558397598)
('feel', 4.871678558397598)
('list', 4.867101891370185)
('half', 4.862504182121556)
('probabl', 4.862504182121556)
('end', 4.839193103253109)
('instead', 4.834464962057163)
('size', 4.834464962057163)
('far', 4.824941080545907)
('came', 4.810482997370677)
('guess', 4.810482997370677)
('arriv', 4.800726822425313)
('free', 4.795812807622884)
('veri disappoint', 4.795812807622884)
('wasnt', 4.795812807622884)
('piec', 4.790874525982302)
('unfortun', 4.780924195129133)
('bitter', 4.770873859275632)
('decid', 4.770873859275632)
('horribl', 4.770873859275632)
('care', 4.765810557319085)
('long', 4.765810557319085)
('everi', 4.7607214878116135)
('sever', 4.7607214878116135)
('start', 4.7607214878116135)
('went', 4.7607214878116135)
('fine', 4.745297017485981)
('usual', 4.740102200608877)
('isnt', 4.734880256627726)
('label', 4.729630900741583)
('case', 4.719048791411046)
('expens', 4.719048791411046)
('aw', 4.7137154454356835)
```

```
('notic', 4.7137154454356835)
('throw', 4.7137154454356835)
('bottl', 4.7083535022942975)
('mouth', 4.7083535022942975)
('save', 4.7083535022942975)
('terribl', 4.702962653659421)
('plastic', 4.697542586190082)
('week', 4.697542586190082)
('minut', 4.686613515657892)
('organ', 4.686613515657892)
('wouldnt', 4.686613515657892)
('left', 4.681103859846922)
('complet', 4.675563679471306)
('stick', 4.675563679471306)
('milk', 4.669992634421851)
('let', 4.664390378873181)
('juic', 4.653090823619248)
('second', 4.64739280250461)
('definit', 4.635898423078875)
('took', 4.6301013053945494)
('groceri', 4.61250554350417)
('wast money', 4.61250554350417)
('add', 4.600600640997851)
('low', 4.600600640997851)
('believ', 4.5945946169376395)
('quit', 4.5945946169376395)
('alway', 4.588552302481677)
('onc', 4.588552302481677)
('custom', 4.582473256405295)
('worst', 4.576357029387859)
('ginger', 4.557780643814923)
('nice', 4.557780643814923)
('fact', 4.551511030801327)
('fresh', 4.551511030801327)
('wrong', 4.545201861608064)
('wonder', 4.538852633929404)
```

In [90]:
```
PlotWordCloud(NegD)
```




**This is a Word Cloud for all the negative reviews in the corpus.**
**This Word Cloud plot correponds to the most frequent words in all negative reviews based on IDF values. More is the IDF Value for a word the less frequent is the word in the corpus.**

## Converting rows to tfidf

In [97]:
```
tfidf_vect = TfidfVectorizer(ngram_range = (1,2))
```

In [98]:
```
Final_Data_tfidf = tfidf_vect.fit_transform(Final_Data["ProcessedText"].values)
```

In [99]:
```
Final_Data_tfidf.shape
```

Out[99]:
```
(10000, 237703)
```

In [100]:
```
print(type(Final_Data_tfidf))
```

```
<class 'scipy.sparse.csr.csr_matrix'>
```

## Applying Forward chaining cross validation to find best value of alpha

In [101]:
```
alphaValue = []
for i in range(1, 500, 2):
    alphaValue.append(i)
```

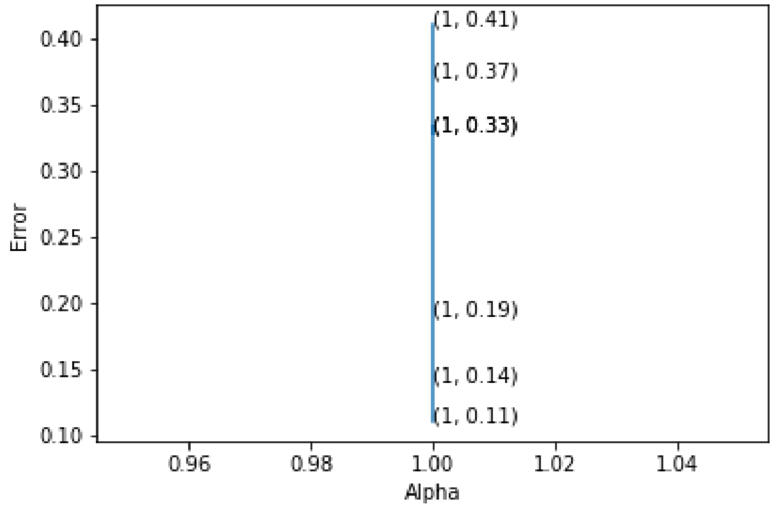
In [102]:
```
# Applying forward chaining cross validation on 80% of data, means 8000 rows. Rest 2000 will be test data.
end1 = 1000
end2 = 2000
FinalScoreOfScores = []

for i in range(7):
    train = Final_Data_tfidf[0:end1]
    train_labels = Final_Data_Labels[0:end1]
    test = Final_Data_tfidf[end1:end2]
    test_labels = Final_Data_Labels[end1:end2]

    scoreOfScores = []
    for a in alphaValue:
        clf = MultinomialNB(alpha=a, fit_prior=True, class_prior=None)
        clf.fit(train, train_labels)
        score = clf.score(test, test_labels)
        scoreOfScores.append(score)
    FinalScoreOfScores.append(scoreOfScores)

    end1 += 1000
    end2 += 1000
```

In [103]:
```
#finding best alpha using brute force
maximum = FinalScoreOfScores[0][0]

for i in range(len(FinalScoreOfScores)):
    Alpha = -1
    for j in range(len(FinalScoreOfScores[i])):
        Alpha += 2
        if maximum < FinalScoreOfScores[i][j]:
            maximum = FinalScoreOfScores[i][j]
            best_alpha = Alpha

print("Maximum Score = "+str(maximum))
print("Value of best alpha = "+str(best_alpha))
```

```
Maximum Score = 0.89
Value of best alpha = 1
```

In [104]:
```
error = []
maxAlpha = []
splitNumber = []

for i in range(len(FinalScoreOfScores)):
    a = -1
    maximum = -10   #any small negative number can be given
    for j in range(len(FinalScoreOfScores[i])):
        a += 2
        if maximum < FinalScoreOfScores[i][j]:
            maximum = FinalScoreOfScores[i][j]
            max_alpha = a

    error.append(1- maximum)
    maxAlpha.append(max_alpha)
    splitNumber.append(i+1)
```

In [105]:
```
plt.plot(maxAlpha, error)

minError1 = min(error)
minError = np.round(minError1, 2)

plt.xlabel("Alpha")
plt.ylabel("Error")
for xy in zip(maxAlpha, np.round(error,2)):
        plt.annotate(xy,xy)

plt.show()
```




In [106]:
```
for xy in zip(maxAlpha, np.round(error,2), splitNumber):
        print(xy)
```

```
(1, 0.41, 1)
(1, 0.37, 2)
(1, 0.33, 3)
(1, 0.33, 4)
(1, 0.19, 5)
(1, 0.14, 6)
(1, 0.11, 7)
```

In forward chaining cross validation that has been applied above--in this dataset--contains 7 splits.

Splits are as follows:

train0: 0:1000 test0: 1000:2000

train1: 0:2000 test1: 2000:3000

train2: 0:3000 test2: 3000:4000

train3: 0:4000 test3: 4000:5000

. .

. .

. .

train6: 0:7000 test6: 7000:8000

Now from every split, maximum score is taken and corresponding "alpha" value is taken. So, we will have 7 maximum scores and 7 corresponding alpha values.

Finally, we have plotted them above.

**We can also see above that error becomes mimimum at split 7.**

## Applying Naive Bayes

In [107]:

```
clf = MultinomialNB(alpha=best_alpha, fit_prior=True, class_prior=None)
clf.fit(Final_Data_tfidf[0:8000], Final_Data_Labels[0:8000])

prediction = clf.predict(Final_Data_tfidf[8000:10000])

score = clf.score(Final_Data_tfidf[8000:10000], Final_Data_Labels[8000:10000])
print(score)
```

```
0.889
```

## Accuracy

In [108]:
```
Accuracy = accuracy_score(Final_Data_Labels[8000:10000], prediction) * 100
print("Accuracy = "+str(Accuracy)+"%")
```

```
Accuracy = 88.9%
```

## Precision

In [109]:
```
Precision = precision_score(Final_Data_Labels[8000:10000], prediction)
print("Precision Score = "+str(Precision))
```

```
Precision Score = 0.8651564185544768
```

## Recall

In [110]:
```
Recall = recall_score(Final_Data_Labels[8000:10000], prediction)
print("Recall Score = "+str(Recall))
```

```
Recall Score = 0.8921023359288098
```

## F1 Score

In [111]:
```
F1_Score = f1_score(Final_Data_Labels[8000:10000], prediction)
print("F1 Score = "+str(F1_Score))
```

```
F1 Score = 0.8784227820372398
```

## Confusion Matrix

In [112]:
```
Confusion_Matrix = confusion_matrix(Final_Data_Labels[8000:10000], prediction)
print("Confusion Matrix  \n"+str(Confusion_Matrix))
```

```
Confusion Matrix
[[976 125]
 [ 97 802]]
```

In [113]:
```
tn, fp, fn, tp = confusion_matrix(Final_Data_Labels[8000:10000], prediction).ravel()
tn, fp, fn, tp
```

Out[113]: (976, 125, 97, 802)

## Getting top 10 Best features

In [115]:
```
featureNames = tfidf_vect.get_feature_names()
len(featureNames)
```

Out[115]: 237703

In [116]:
```
FeatureProbabilities = clf.feature_log_prob_
FeatureProbabilities.shape
```

Out[116]: (2, 237703)

In [117]:
```
FeatureProbabilitiesTrans = FeatureProbabilities.T
FeatureProbabilitiesTrans.shape
```

Out[117]: (237703, 2)

In [119]:
```
features_prob_dataFrame = pd.DataFrame(FeatureProbabilitiesTrans, index = None, columns = ["Class 0 Probability Scores",
features_prob_dataFrame["Feature Names"] = featureNames
```

In [120]:
```
impFeatures_Class0 = features_prob_dataFrame.sort_values(by = 'Class 0 Probability Scores', axis = 0, ascending = False)
impFeatures_Class0.drop(["Class 1 Probability Scores"], axis = 1, inplace = True)
impFeatures_Class0.head(10)
```

Out[120]:

| | Class 0 Probability Scores | Feature Names |
|---|---|---|
| **197889** | -7.130319 | the |
| **8036** | -7.672842 | and |
| **206446** | -7.873337 | this |
| **133366** | -7.986996 | not |
| **222275** | -8.018136 | was |
| **196540** | -8.174226 | that |
| **76371** | -8.212109 | for |
| **30555** | -8.226640 | but |
| **192872** | -8.238006 | tast |
| **113416** | -8.367527 | like |

**Here, above we got all the probability scores of all the features given class label 0. They are nothing but "likelihood" probabilities of all the features given class label 0. Here, in above dataframe values corresponding to column "Class 0" has all the probability scores of features given class 0. The above dataFrame has been sorted according to the probability scores of Class 0. On Features column we got important feature names for points belong to class label 0.**

In [121]:
```
impFeatures_Class1 = features_prob_dataFrame.sort_values(by = 'Class 1 Probability Scores', axis = 0, ascending = False)
impFeatures_Class1.drop(["Class 0 Probability Scores"], axis = 1, inplace = True)
impFeatures_Class1.head(10)
```

Out[121]:

| | Class 1 Probability Scores | Feature Names |
|---|---|---|
| **197889** | -7.290579 | the |
| **8036** | -7.456165 | and |
| **206446** | -7.951256 | this |
| **76371** | -8.020146 | for |
| **13193** | -8.203751 | are |
| **235852** | -8.263928 | you |
| **92550** | -8.299243 | have |
| **87443** | -8.314475 | great |
| **196540** | -8.343133 | that |
| **230790** | -8.345903 | with |

**Here, above we got all the probability scores of all the features given class label 1. They are nothing but "likelihood" probabilities of all the features given class label 1. Here, in above dataframe values corresponding to column "Class 1" has all the probability scores of features given class 1. The above dataFrame has been sorted according to the probability scores of Class 1. On Features column we got important feature names for points belong to class label 1.**