Data Preprocessing

Amazon Fine Food Reviews

Data Source: 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. Id
2. ProductId - unique identifier for the product
3. UserId - unqiue identifier for the user
4. ProfileName
5. HelpfulnessNumerator - number of users who found the review helpful
6. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
7. Score - rating between 1 and 5
8. Time - timestamp for the review
9. Summary - brief summary of the review
10. Text - text of the review

Objective:

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

In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")



import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os
```

Loading the data which we have saved after Data cleaning

In [2]:

```
conn = sqlite3.connect('final.sqlite')
final_data = pd.read_sql_query(""" SELECT * FROM Reviews """, conn)
conn.close
```

Out[2]:

```
<function Connection.close>
```

In [3]:

```
final_data.head()
```

Out[3]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpful
0	138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	
1	138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	
2	138689	150507	0006641040	A1S4A3IQ2MU7V4	sally sue "sally sue"	1	
3	138690	150508	0006641040	AZGXZ2UUK6X	Catherine Hallberg " (Kate)"	1	
4	138691	150509	0006641040	A3CMRKGE0P909G	Teresa	3	

We have two text features in our dataset (Summary and Text). These Text features needs to be preprocessed

Text Preprocessing: Stemming, stop-word removal and Lemmatization.

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

After which we collect the words used to describe positive and negative reviews

In [ ]:

```
```

Text Preprocessing

[1] HTML Tag Removal

In [4]:

```
#Function to clean html tags from a sentence
import re
def removeHtml(sentence):
    pattern = re.compile('<.*?>')
    cleaned_text = re.sub(pattern,' ',sentence)
    return cleaned_text

removeHtml('<a href="foo.com" class="bar">I Want This <b>text!</b></a><>')
```

Out[4]:

```
' I Want This  text!   '
```

[2] Punctuations Removal

In [5]:

```
#Function to keep only words containing letters A-Z and a-z. This will remove all punctuati
def removePunctuations(sentence):
    cleaned_text  = re.sub('[^a-zA-Z]',' ',sentence)
    return (cleaned_text)

removePunctuations("fsd*?~,,,( sdfsdfdsvv)#")
```

Out[5]:

```
'fsd        sdfsdfdsvv  '
```

[3] Removal of words with numbers

In [6]:

```
#Remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
def removeNumbers(sentence):
    sentence = re.sub("\S*\d\S*", " ", sentence).strip()
    return (sentence)

removeNumbers("The5 number is removed removed55")
```

Out[6]:

```
'number is removed'
```

[4] Removal URL from sentences.

In [7]:

```
#Remove URL from sentences.
def removeURL(sentence):
    text = re.sub(r"http\S+", " ", sentence)
    sentence = re.sub(r"www.\S+", " ", text)
    return (sentence)

removeURL("https://www.google.com/search?client=ubuntu&channel=fs&q=google+drive+storage+pl
```

Out[7]:

```
'  notice the URL is removed'
```

[5] Remove words like 'zzzzzzzzzzzzzzzzzzzzzzz', 'testtting', 'grrrrrrreeeettttt' etc. Preserves words like 'looks', 'goods', 'soon' etc. We will remove all such words which has three consecutive repeating characters.

In [8]:

```
#https://stackoverflow.com/questions/37012948/regex-to-match-an-entire-word-that-contains-r
#Remove words like 'zzzzzzzzzzzzzzzzzzzzzz', 'testtting', 'grrrrrrreeeettttt' etc. Preserv
def removePatterns(sentence):
    cleaned_text  = re.sub("\\s*\\b(?=\\w*(\\w)\\1{2,})\\w*\\b",' ',sentence)
    return (cleaned_text)

removePatterns("This looks soooooooo good! I am so happpyyy")
```

Out[8]:

```
'This looks  good! I am so '
```

[6] Expand the most common english contractions

In [9]:

```
# https://stackoverflow.com/a/47091490/4084039
# https://en.wikipedia.org/wiki/Wikipedia:List_of_English_contractions
import re

#Expand the reviews x is aninput string of any length. Convert all the words to lower case
def decontracted(x):
    x = str(x).lower()
    x = x.replace(",000,000", " m").replace(",000", " k").replace("’", "'").replace("’", "'
                           .replace("won't", " will not").replace("cannot", " can not").rep
                           .replace("n't", " not").replace("what's", " what is").replace("i
                           .replace("'ve", " have").replace("'m", " am").replace("'re", " a
                           .replace("he's", " he is").replace("she's", " she is").replace("
                           .replace("%", " percent ").replace("₹", " rupee ").replace("$",
                           .replace("€", " euro ").replace("'ll", " will").replace("how's",
                           .replace("o'clock"," of the clock").replace("ne'er"," never").re
                           .replace("finna"," fixing to").replace("gonna"," going to").repl
                           .replace("daresn't"," dare not").replace("dasn't"," dare not").r
                           .replace("'cause'"," because")

    x = re.sub(r"([0-9]+)000000", r"\1m", x)
    x = re.sub(r"([0-9]+)000", r"\1k", x)
    return x
```

[7] Stopwords

Stop words usually refers to the most common words in a language are generally filtered out before or after processing of natural language data. Sometimes it is avoided to remove the stop words to support phrase search.

In [10]:

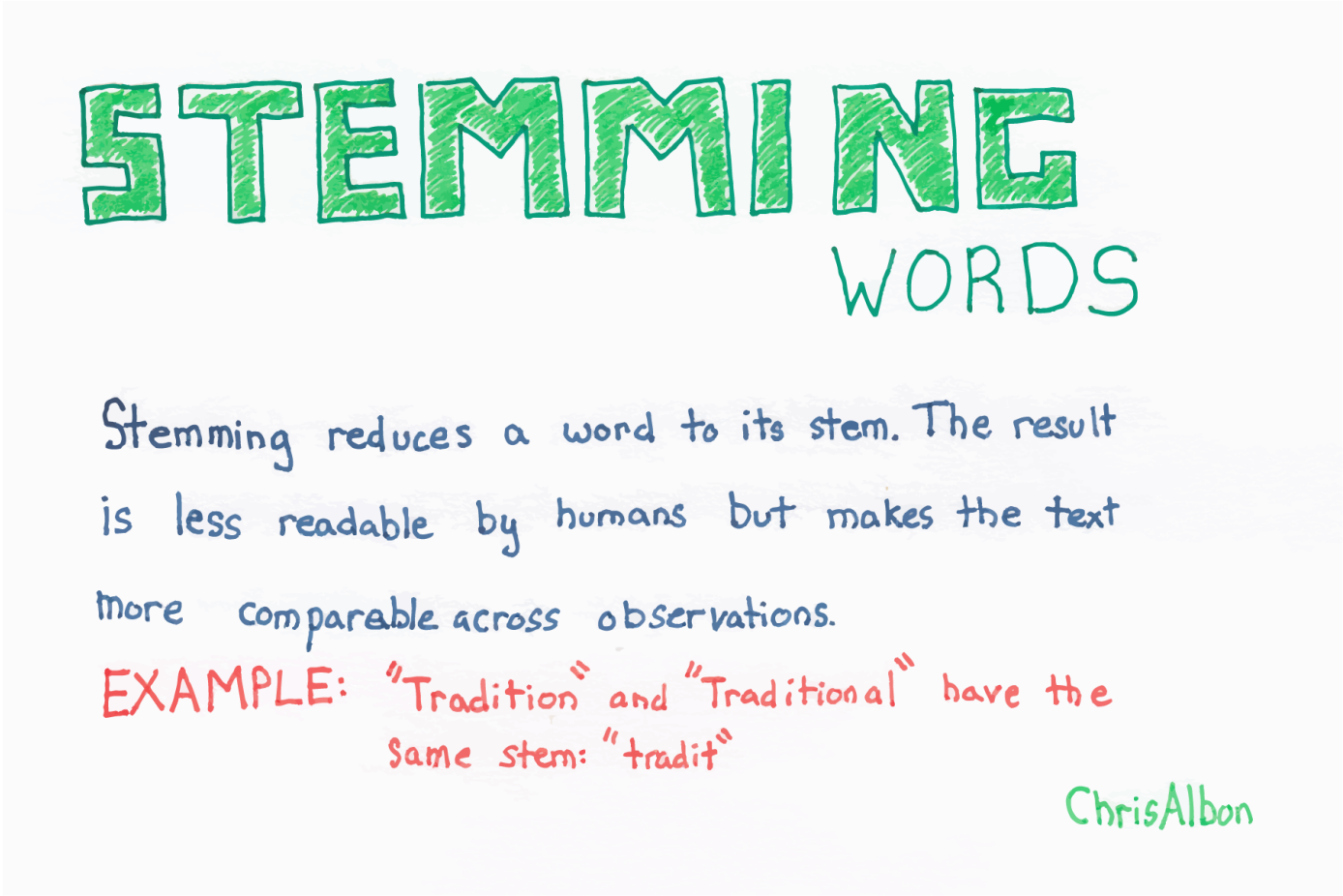
```
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer



#Stemming and stopwords removal
from nltk.stem.snowball import SnowballStemmer
sno = SnowballStemmer(language='english')

#Removing the word 'not' from stopwords
default_stopwords = set(stopwords.words('english'))
#excluding some useful words from stop words list as we doing sentiment analysis
excluding = set(['against','not','don', "don't",'ain', 'aren', "aren't", 'couldn', "couldn'
             'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'i
             'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't",'shouldn', "shoul
             "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"])

custom_stopwords = default_stopwords - excluding

print(custom_stopwords)
```

```
{'s', 'doing', 'been', 'i', 'shan', 'here', 'ourselves', 'or', 'into', 'ho
w', 'myself', 'those', 'both', "it's", 'after', 'nor', 'why', 'very', 'suc
h', 're', "you've", 'did', 'he', "you're", 'whom', 'has', 'up', 'should', 'b
elow', 'now', 'am', 'does', 'this', 'm', 'ours', 'to', "you'd", 'just', 'the
mselves', 'yours', 'we', 'under', 'more', 'for', 'of', 'will', 'our', 'betwe
en', 'when', 'as', 'o', 'off', 'she', 'herself', 'some', 'at', 'any', 'onl
y', 'other', 'do', 'are', 'through', 'being', 'ma', 'no', 'own', 'y', 'fro
m', 'd', 'with', 'itself', 'they', 've', 'have', 'there', 'it', 'her', 'if',
'few', 'theirs', 'having', 'their', 'but', 'you', 'on', 'out', 'down', 'wh
o', 'before', "shan't", "that'll", 'what', 'your', 'and', 'so', 'about', 'ea
ch', 'be', 'where', 'further', 'which', 'were', 'in', 'until', 'then', 'duri
ng', 'than', 'had', 'hers', 'while', 'once', 'too', "you'll", 'because', 'ca
n', 'these', 'its', 'him', 'himself', 'his', 't', 'a', 'is', "she's", 'me',
'an', 'all', "should've", 'above', 'll', 'yourself', 'them', 'yourselves',
'my', 'was', 'again', 'most', 'that', 'the', 'same', 'over', 'by'}
```

[8] Stemming

Porter Stemmer: Most commonly used stemmer without a doubt, also one of the most gentle stemmers. Though it is also the most computationally intensive of the algorithms. It is also the oldest stemming algorithm by a large margin.

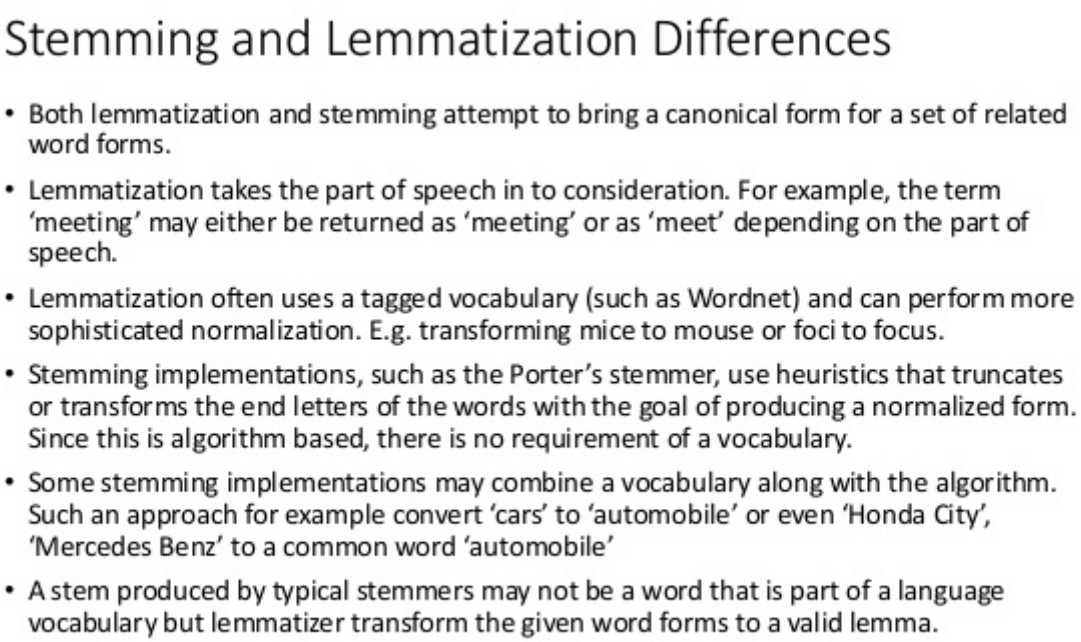
SnowBall Stemmer(Porter2): Nearly universally regarded as an improvement over porter, and for good reason. Porter himself in fact admits that it is better than his original algorithm. Slightly faster computation time than Porter, with a fairly large community around it.



In [11]:

```
from nltk.stem import SnowballStemmer
snow = SnowballStemmer('english') #initialising the snowball stemmer
print("Stem/Root words of the some of the words using SnowBall Stemmer:")
print(snow.stem('tasty'))
print(snow.stem('tasteful'))
print(snow.stem('tastiest'))
print(snow.stem('delicious'))
```

```
Stem/Root words of the some of the words using SnowBall Stemmer:
tasti
tast
tastiest
delici
```



Preprocessing output for one review

In [12]:

```
str1=' '
final_string=[]
all_positive_words=[] # store words from +ve reviews here
all_negative_words=[] # store words from -ve reviews here.
s=''
for sent in final_data['Text'][2:3].values: #Running only for 2nd review
    filtered_sentence=[]
    print("Review:",sent) #Each review
    sent=removeHtml(sent)# remove HTML tags
    sent=removePunctuations(sent)# remove Punctuation Symbols
    print("\nReview Split: ",sent.split())
    print()
    for w in sent.split():
        print("================================>",w)
        if((w.isalpha()) and (len(w)>2)):#If it is a numerical value or character of lenght
            if(w.lower() not in custom_stopwords):# If it is a stopword
                s=(snow.stem(w.lower())).encode('utf8') #Stemming the word using SnowBall S
                print("Selected: Stem Word->",s)
                print()
                filtered_sentence.append(s)
            else:
                print("Eliminated as it is a stopword")
                print()
                continue
        else:
            print("Eliminated as it is a numerical value or character of length less than 2
            print()
            continue
#     print(filtered_sentence)
    str1 = b" ".join(filtered_sentence) #final string of cleaned words

    final_string.append(str1)
    print("**********************************************************************")
    print("\nFinally selected words from the review: ",final_string)
```

```
Review: This is a fun way for children to learn their months of the year!
We will learn all of the poems throughout the school year.  they like the
handmotions which I invent for each poem.

Review Split:  ['This', 'is', 'a', 'fun', 'way', 'for', 'children', 'to',
'learn', 'their', 'months', 'of', 'the', 'year', 'We', 'will', 'learn', 'a
ll', 'of', 'the', 'poems', 'throughout', 'the', 'school', 'year', 'they',
'like', 'the', 'handmotions', 'which', 'I', 'invent', 'for', 'each', 'poe
m']

================================> This
Eliminated as it is a stopword

================================> is
Eliminated as it is a numerical value or character of length less than 2

================================> a
Eliminated as it is a numerical value or character of length less than 2
```

[3.1] Preprocessing Review Text

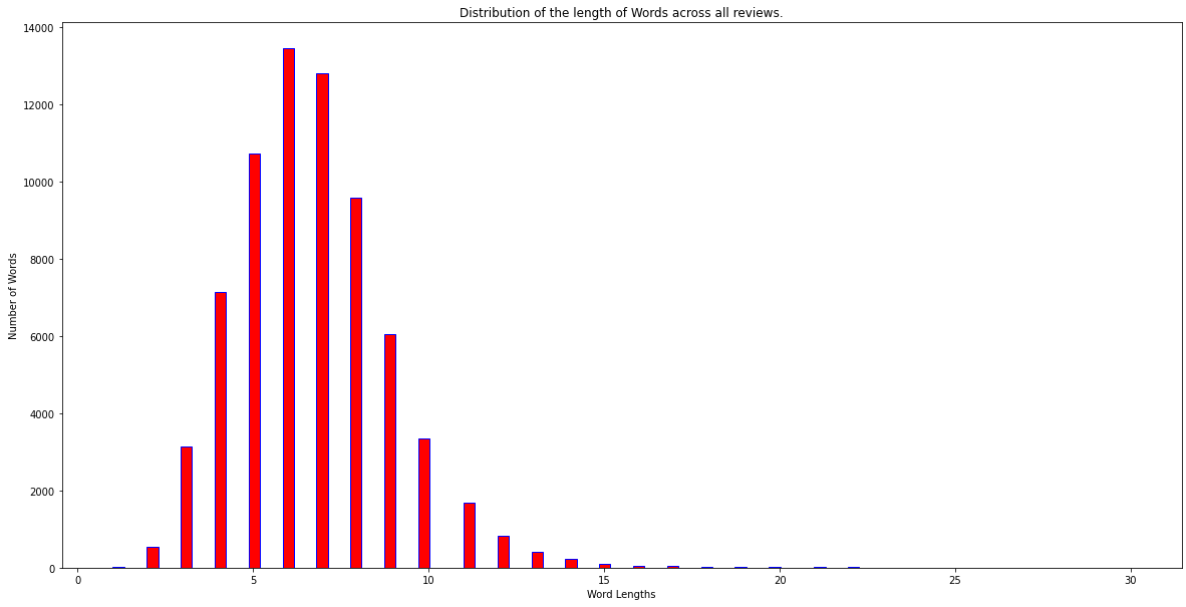
In [13]:

```
#We will check the distribution of stemmed word lengths across the whole review dataset to
#In other words we will keep only those words which has a length less than that of a speici
from tqdm import tqdm
total_words = []

for review in tqdm(final_data['Text'].values):
    filtered_sentence=[]
    review = decontracted(review)
    review = removeNumbers(review)
    review = removeHtml(review)
    review = removeURL(review)
    review = removePunctuations(review)
    review = removePatterns(review)

    for cleaned_words in review.split():
        if((cleaned_words not in custom_stopwords)):
            stemed_word=(sno.stem(cleaned_words.lower()))
            total_words.append(stemed_word)

total_words = list(set(total_words)) #Get list of unique words.

#A list to hold the length of each words used in all the reviews used across the whole data
dist = []
for i in tqdm(total_words):
    length = len(i)
    dist.append(length)

# matplotlib histogram to see the distribution of the length of words
plt.figure(figsize=(20,10))
plt.hist(dist, color = 'red', edgecolor = 'blue', bins =90)
plt.title('Distribution of the length of Words across all reviews.')
plt.xlabel('Word Lengths')
plt.ylabel('Number of Words')
```

```
100%|██████████| 364171/364171 [10:39<00:00, 569.05it/s]
100%|██████████| 70255/70255 [00:00<00:00, 833371.03it/s]
```

Out[13]:

```
Text(0, 0.5, 'Number of Words')
```



Conclusion from the above histogram:

We can see that most stemmed words present in the reviews has lengths between 4 and 10. Words which has length greater than 15 are very very very few as compared to other words. So we will discard these words from the reviews when we process them. It means we will consider only those words whose length is greater than 2 and less than 16.

In [14]:

```
# Combining all the above data cleaning methodologies as discussed above.

#Processing review Texts
preprocessed_reviews = [] #Store all the processed reviews
all_positive_words=[] #Store all the relevant words from Positive reviews
all_negative_words=[] #Store all the relevant words from Negative reviews

count=0     #Iterator to iterate through the list of reviews and check if a given review be
string=' '
stemed_word=' '

for review in tqdm(final_data['Text'].values):
    filtered_sentence=[]
    review = decontracted(review)
    review = removeNumbers(review)
    review = removeHtml(review)
    review = removeURL(review)
    review = removePunctuations(review)
    review = removePatterns(review)

    for cleaned_words in review.split():
        if((cleaned_words not in custom_stopwords) and (2<len(cleaned_words)<16)):
            stemed_word=(sno.stem(cleaned_words.lower()))
            filtered_sentence.append(stemed_word)
            if (final_data['Score'].values)[count] == 'positive':
                all_positive_words.append(stemed_word) #List of all the relevant words from
            if(final_data['Score'].values)[count] == 'negative':
                all_negative_words.append(stemed_word) #List of all the relevant words from
        else:
            continue
    review = " ".join(filtered_sentence) #Final string of cleaned words
    preprocessed_reviews.append(review.strip()) #Data corpus containg cleaned reviews from
    count+=1

#Save the list of positive words and negative words
import pickle
with open('all_positive_words.pkl', 'wb') as file:
    pickle.dump(all_positive_words, file)

with open('all_negative_words.pkl', 'wb') as file:
    pickle.dump(all_negative_words, file)

#Adding a column of CleanedText to the table final which stores the data_corpus after pre-p
final_data['CleanedText']=preprocessed_reviews

print("The length of the data corpus is : {}".format(len(preprocessed_reviews)))
```

```
100%|██████████| 364171/364171 [14:09<00:00, 428.94it/s]

The length of the data corpus is : 364171
```

[3.2] Preprocessing Review Summary + Saving the Processed DB for future use.

In [15]:

```
## Similartly you can do preprocessing for summary summary also.
import warnings
warnings.filterwarnings("ignore")

preprocessed_summary = [] #Store all the processed summary

count=0     #Iterator to iterate through the list of summarys and check if a given summary
string=' '
stemed_word=' '

for summary in tqdm(final_data['Summary'].values):
    filtered_sentence=[]
    summary = decontracted(summary)
    summary = removeNumbers(summary)
    summary = removeHtml(summary)
    summary = removeURL(summary)
    summary = removePunctuations(summary)
    summary = removePatterns(summary)

    for cleaned_words in summary.split():
        if(2<len(cleaned_words)<16):
            stemed_word=(sno.stem(cleaned_words.lower()))
            filtered_sentence.append(stemed_word)
    summary = " ".join(filtered_sentence) #Final string of cleaned words
    preprocessed_summary.append(summary.strip()) #Data corpus containg cleaned summarys fr
    count+=1

#Adding a column of CleanedSummary to the table final which stores the data_corpus after pr
final_data['CleanedSummary']=preprocessed_summary

#Combing the reviews with summary for each sentence
final_data['Combined_Reviews'] = final_data['CleanedText'].values + " " + final_data['Clean

#Store final table into an SQLLite table for future.

conn = sqlite3.connect('totally_processed_DB.sqlite')
sqlite_table = "Reviews"
final_data.to_sql(sqlite_table, conn, if_exists='replace')
conn.close

print("File is saved as 'totally_processed_DB.sqlite'")
```

```
100%|██████████| 364171/364171 [01:01<00:00, 5969.32it/s]

File is saved as 'totally_processed_DB.sqlite'
```

In [16]:

```
con = sqlite3.connect('totally_processed_DB.sqlite')
final_data1 = pd.read_sql_query(""" SELECT * FROM Reviews """, con)
conn.close
```

Out[16]:

```
<function Connection.close>
```

In [17]:

```
final_data1.head()
```

Out[17]:

	level_0	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator
0	0	138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0
1	1	138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1
2	2	138689	150507	0006641040	A1S4A3IQ2MU7V4	sally sue "sally sue"	1
3	3	138690	150508	0006641040	AZGXZ2UUK6X	Catherine Hallberg " (Kate)"	1
4	4	138691	150509	0006641040	A3CMRKGE0P909G	Teresa	3

In [18]:

```
final_data1.columns
```

Out[18]:

```
Index(['level_0', 'index', 'Id', 'ProductId', 'UserId', 'ProfileName',
       'HelpfulnessNumerator', 'HelpfulnessDenominator', 'Score', 'Time',
       'Summary', 'Text', 'CleanedText', 'CleanedSummary', 'Combined_Review
s'],
      dtype='object')
```

Analysis after performing preprocessing on TEXT

Positive and Negative words in reviews

Positive and Negative words in reviews

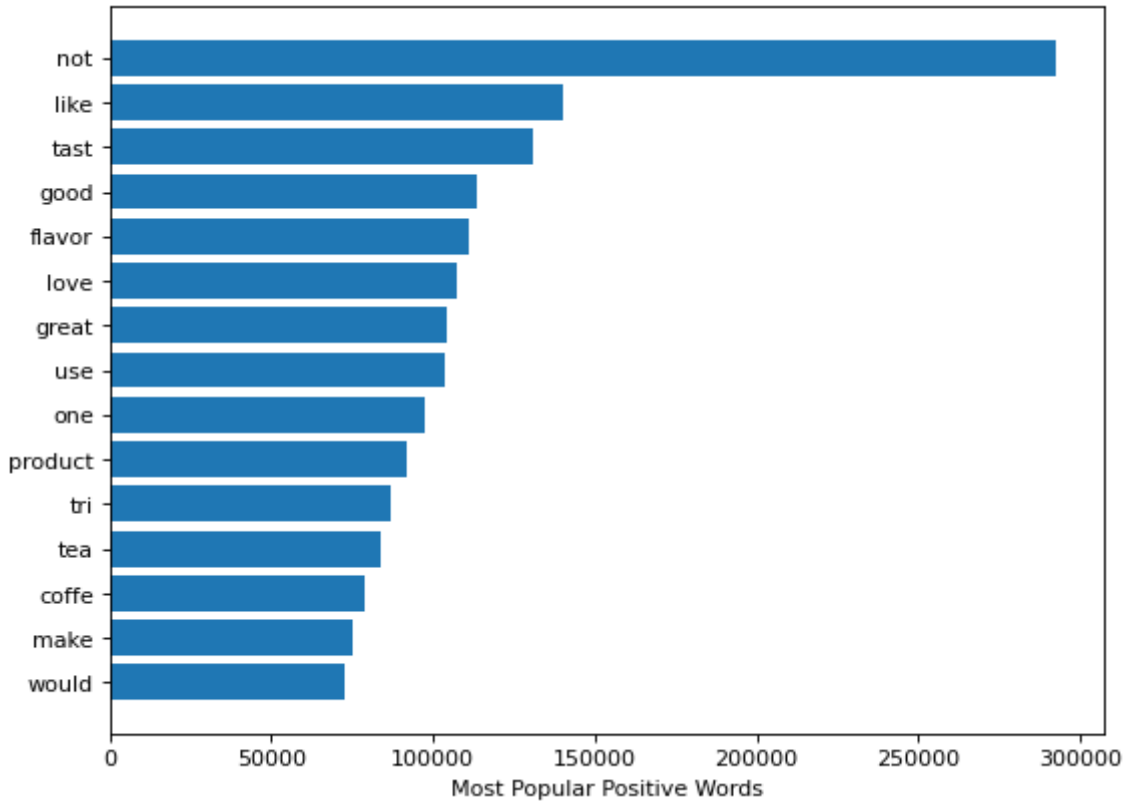
In [19]:

```
from collections import Counter
print("No. of positive words:",len(all_positive_words))
print("No. of negative words:",len(all_negative_words))
# print("Sample postive words",all_positive_words[:9])
# print("Sample negative words",all_negative_words[:9])
positive = Counter(all_positive_words)
print("\nMost Common postive words",positive.most_common(10))
negative = Counter(all_negative_words)
print("\nMost Common negative words",negative.most_common(10))
```

```
No. of positive words: 11791002
No. of negative words: 2426140

Most Common postive words [('not', 292389), ('like', 140360), ('tast', 13061
7), ('good', 113380), ('flavor', 110911), ('love', 107545), ('great', 10430
0), ('use', 103675), ('one', 97181), ('product', 91900)]

Most Common negative words [('not', 95184), ('tast', 35014), ('like', 3259
5), ('product', 28579), ('would', 23273), ('one', 20696), ('flavor', 19860),
('tri', 17740), ('use', 15271), ('good', 15116)]
```

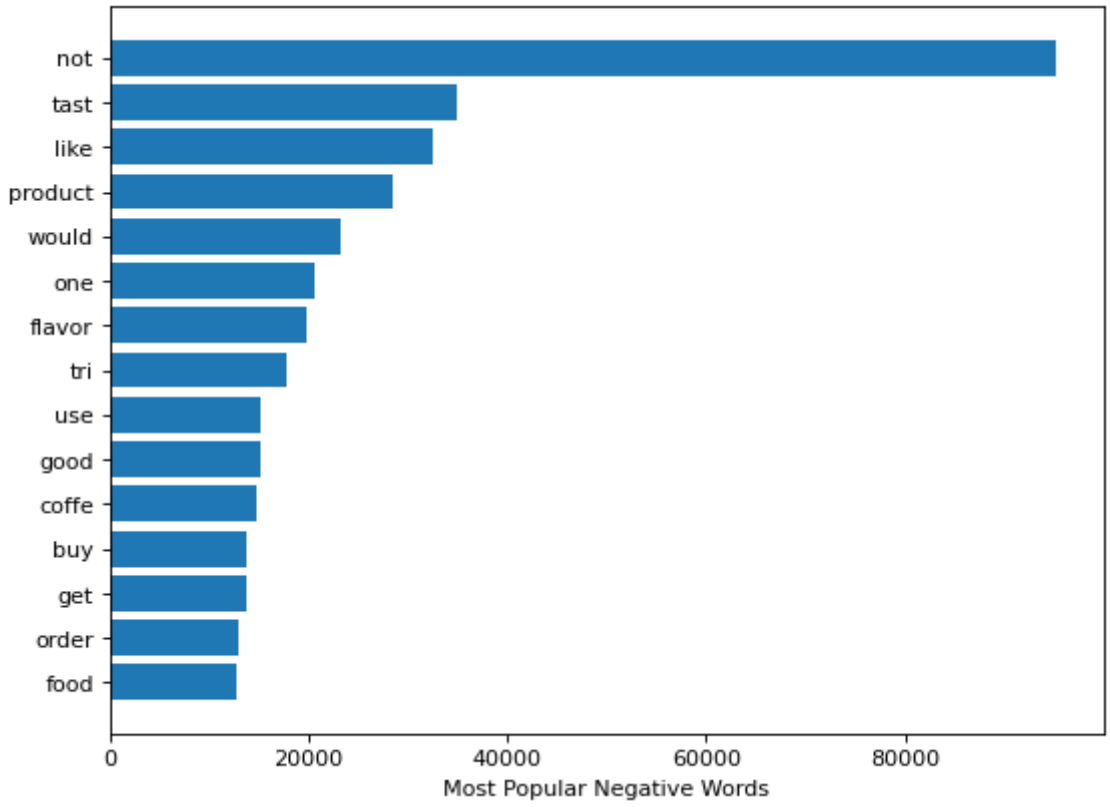
In [20]:

```
from matplotlib.pyplot import figure
figure(num=None, figsize=(8, 6), dpi=80, facecolor='w', edgecolor='k')
pos_words = positive.most_common(15)
pos_words.sort(key=lambda x: x[1], reverse=False)
words=[]
times=[]
for w,t in pos_words:
    words.append(w)
    times.append(t)
plt.barh(range(len(words)),times)
plt.yticks(range(len(words)),words)
plt.xlabel('Most Popular Positive Words')
plt.show()
```



In [21]:

```
neg_words = negative.most_common(15)
neg_words.sort(key=lambda x: x[1], reverse=False)
words=[]
times=[]
for w,t in neg_words:
    words.append(w)
    times.append(t)
figure(num=None, figsize=(8, 6), dpi=80, facecolor='w', edgecolor='k')
plt.barh(range(len(words)),times)
plt.yticks(range(len(words)),words)
plt.xlabel('Most Popular Negative Words')
plt.show()
```



Observation:

- "tast" , "like" , "flavor", "good" and "one" are some of the most common words in both negative and positve reviews
- "good" and "great" are some of the most common words in positive reviews
- "would" and "coffe" are some of the most common words in negative reviews
- tasty, good, etc are some of the words common in both **because there may be a not before it like "not tasty" , "not good"**

Word Cloud of Whole Dataset

In [22]:

```
from wordcloud import WordCloud, STOPWORDS
stopwords = set(STOPWORDS)

df2=final_data

plt.rcParams['figure.figsize']=(8.0,6.0)    #(6.0,4.0)
figure(num=None, figsize=(12, 10), dpi=80, facecolor='w', edgecolor='k')
plt.rcParams['font.size']=12                #10
plt.rcParams['savefig.dpi']=100             #72
plt.rcParams['figure.subplot.bottom']=.1


def show_wordcloud(data, title = None):
    wordcloud = WordCloud(
        background_color='white',
        stopwords=stopwords,
        max_words=200,
        max_font_size=40,
        scale=3,
        random_state=1 # chosen at random by flipping a coin; it was heads
    ).generate(str(data))

    fig = plt.figure(1, figsize=(8, 8))
    plt.axis('off')
    if title:
        fig.suptitle(title, fontsize=20)
        fig.subplots_adjust(top=2.3)

    plt.imshow(wordcloud)
    plt.show()

show_wordcloud(df2['CleanedText'])
#df2.loc[df2['Score'] == 1]['CleanedText']
```

Word Cloud of only Positive Reviews

In [25]:

```
from wordcloud import WordCloud, STOPWORDS
stopwords = set(STOPWORDS)

plt.rcParams['figure.figsize']=(8.0,6.0)    #(6.0,4.0)
figure(num=None, figsize=(12, 10), dpi=80, facecolor='w', edgecolor='k')
plt.rcParams['font.size']=12                #10
plt.rcParams['savefig.dpi']=100             #72
plt.rcParams['figure.subplot.bottom']=.1


def show_wordcloud(data, title = None):
    wordcloud = WordCloud(
        background_color='white',
        stopwords=stopwords,
        max_words=200,
        max_font_size=40,
        scale=3,
        random_state=1 # chosen at random by flipping a coin; it was heads
    ).generate(str(data))

    fig = plt.figure(1, figsize=(8, 8))
    plt.axis('off')
    if title:
        fig.suptitle(title, fontsize=20)
        fig.subplots_adjust(top=2.3)

    plt.imshow(wordcloud)
    plt.show()

show_wordcloud(df2.loc[df2['Score'] == 'positive']['CleanedText'])
```

Word Cloud of only Negative Reviews

In [26]:

```
from wordcloud import WordCloud, STOPWORDS
stopwords = set(STOPWORDS)

plt.rcParams['figure.figsize']=(8.0,6.0)    #(6.0,4.0)
figure(num=None, figsize=(12, 10), dpi=80, facecolor='w', edgecolor='k')
plt.rcParams['font.size']=12                #10
plt.rcParams['savefig.dpi']=100             #72
plt.rcParams['figure.subplot.bottom']=.1


def show_wordcloud(data, title = None):
    wordcloud = WordCloud(
        background_color='white',
        stopwords=stopwords,
        max_words=200,
        max_font_size=40,
        scale=3,
        random_state=1 # chosen at random by flipping a coin; it was heads
    ).generate(str(data))

    fig = plt.figure(1, figsize=(8, 8))
    plt.axis('off')
    if title:
        fig.suptitle(title, fontsize=20)
        fig.subplots_adjust(top=2.3)

    plt.imshow(wordcloud)
    plt.show()

show_wordcloud(df2.loc[df2['Score'] == 'negative']['CleanedText'])
```

In [ ]:

```
```

In [ ]:

```
```