Amazon Fine Food Reviews Analysis

Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/

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).

[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.

Imports

```
import numpy as np
import pandas as pd
import sqlite3
import re
import string
from functools import reduce
```

```
from bs4 import BeautifulSoup
from tqdm import tqdm

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')

from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

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

from ipykernel import kernelapp as app
```

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

SQLITE connection and data extract

```
con = sqlite3.connect("/content/drive/MyDrive/AAIC/Datasets/AmazonReviews.sqlite")
```

```
data = pd.read_sql_query("select * from Reviews where Score != 3 limit 5000", con)
print(data.shape)
print(data.columns)

#replace score with proper positive/negative values
data['Score'] = data['Score'].map(lambda x: "Positive" if x>3 else "Negative")
data.head(5)
```

```
(5000, 10)
Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator',
       'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text'],
      dtype='object')
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenom:
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	

Data Cleaning - Deduplication

It's mostly about the gut feeling. We need to use common sense

```
pd.read_sql_query("select * from Reviews where UserID='AR5J8UI46CURR' and Score != 3 order by
```

Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDeno

This user added same summary/text at exactly the same time, but different product it. Is there any error? So check the info of the product. If we check it in https://www.amazon.com/dp/B000HDL1RQ , they are all same products with different flavours. So the person has given a review for 1 product and duplicated under different products. Amazon is sharing the reviews for the products.

But it is not good for us. So, let's remove the unwanted data.

```
fdata = data.drop_duplicates(subset={"UserId",  "ProfileName", "Time", "Text"}, keep='first',
fdata = fdata.copy()
print(fdata.shape)
print(fdata.shape[0]/data.shape[0]*100)
```

```
(4986, 10)
99.72
```

3	73791	B000HDOPZG	AR5J8UI46CURR	Krishnan	2

```
#from main data
print("In Main data\n", pd.read_sql_query("select * from Reviews where HelpfulnessNumerator >

print("\nIn the fdata\n", fdata[(fdata['HelpfulnessNumerator'] > fdata['HelpfulnessDenominator
```

```
In Main data
      Id  ...                                              Text
0  44737  ...  It was almost a 'love at first bite' - the per...
1  64422  ...  My son loves spaghetti so I didn't hesitate or...

[2 rows x 10 columns]

In the fdata
 Empty DataFrame
Columns: [Id, ProductId, UserId, ProfileName, HelpfulnessNumerator, HelpfulnessDenominatc
Index: []
```

```
#no of positive and negative reviews in the fdata
fdata['Score'].value_counts()
```

```
Positive    4178
Negative     808
Name: Score, dtype: int64
```

```
print(fdata.shape)
```

```
(4986, 10)
```

Text Preprocessing

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)

```
'''
[('ab*', 'a followed by zero or more b'),
     ('ab+', 'a followed by one or more b'),
     ('ab?', 'a followed by zero or one b'),
     ('ab{3}', 'a followed by three b'),
     ('ab{2,3}', 'a followed by two to three b')
     (r'\d+', 'sequence of digits'),
     (r'\D+', 'sequence of non-digits'),
     (r'\s+', 'sequence of whitespace'),
     (r'\S+', 'sequence of non-whitespace'),
     (r'\w+', 'alphanumeric characters'),
     (r'\W+', 'non-alphanumeric'),
     (r'^\w+', 'word at start of string'),
     (r'\A\w+', 'word at start of string'),
     (r'\w+\S*$', 'word near end of string'),
     (r'\w+\S*\Z', 'word near end of string'),
     (r'\w*t\w*', 'word containing t'),
     (r'\bt\w+', 't at start of word'),
     (r'\w+t\b', 't at end of word'),
     (r'\Bt\B', 't, not start or end of word'),
     ('a(ab)', 'a followed by literal ab'),
     ('a(a*b*)', 'a followed by 0-n a and 0-n b'),
     ('a(ab)*', 'a followed by 0-n ab'),
     ('a(ab)+', 'a followed by 1-n ab')],
'''


def stop_words():
  stop_words = set(stopwords.words('english'))
  preserve = ["not", "nor", "no"]
  remove = ['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "
             "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
             'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', '
             'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
             'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'ha
             'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'unti
             'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'duri
             'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'u
             'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'bot
             'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very
```

```
             'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very
             's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd'
             've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn
             "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "m
             "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "
             'won', "won't", 'wouldn', "wouldn't"]
  for pword in preserve:
    if pword in stop_words:
      stop_words.remove(pword.lower())
  for pword in remove:
    if pword not in stop_words:
      stop_words.add(pword.lower())
  return stop_words

def remove_html_tags(sentence, usebs=False):
  #use beautifulsoup to remove tags
  if usebs:
    return BeautifulSoup(sentence, "lxml").get_text()

  #use normal way to remove html tags
  comp = re.compile("<.*?>")
  correct = re.sub(comp,' ', sentence)
  return correct

def remove_punc(sentence, use_direct_re=True):
  if use_direct_re:
    return re.sub('[^A-Za-z0-9]+', ' ', sentence);
  punc = np.array(["\\"+char if char in ["[", "]"] else char for char in "[]{}!@#$%^&*()\"':;?
  regex = "["+reduce(lambda x,y: "{}|{}".format(x,y), a)+"]" #[!|@|] is the sample regex
  comp = re.compile(regex)
  correct = re.sub(comp,' ', sentence)
  return correct

def remove_urls(sentence):
  return re.sub(r'(http://\S+)|(https://\S+)', '', sentence)

def stem(word):
  ps = PorterStemmer()
  return ps.stem(word)

def decontracted(word):
  word_mappings = {"n\'t":" not", "\'re":" are", "\'s":" is", "\'d":" would", "\'ll":" will",
  specific_mappings = {"won\'t":"will not", "can\'t":"can not"}

  for (k,v) in specific_mappings.items():
    comp = re.compile(k)
    word = re.sub(comp, v, word)
  for (k,v) in word_mappings.items():
    comp = re.compile(k)
    word = re.sub(comp, v, word)

  return word;

def remove_words_with_num(word):
  return re.sub("\S*\d\S*", " ", word)
```

```python
def clean_data(data, stem_w=True):
  sw = stop_words()
  preprocessed_reviews = []
  for review in tqdm(data):
    review = remove_html_tags(review, True)
    review = remove_urls(review)
    review = decontracted(review)
    review = remove_punc(review, True)
    review = remove_words_with_num(review)
    review = ' '.join(stem(word.lower()) if stem_w else word.lower() for word in review.split(
    preprocessed_reviews.append(review.strip())
  return preprocessed_reviews

DEBUG = False
if DEBUG:
  print("not" in stop_words())
  print(remove_html_tags('Raghu<html><lksdf<adf>'))
  print(remove_html_tags("<a href=''>Link</a>https://www.google.com"))
  print(remove_html_tags("<a href=''>Link</a>https://www.google.com", True))
  print(remove_urls("<a href=''>Link</a> https://www.google.com"))
  print(remove_punc('Raghu!@#$%^&*(){}][\":;\'?><,./<html><lksdf<adf>'))
  print(stem("friendly"))
  print(stem("tasty"))
  print(decontracted("I won't be attending. You're welcome"))
  print(remove_punc("a 123 a34 $5t"))
  print(remove_punc("a 123 a34 $5t", True))
  print(remove_words_with_num("a 123 a34 $5t"))
  print(remove_urls("<a href=''>Link</a> httpasdfs://www.google.com"))
```

```python
if "Processed Text" in fdata.columns:
  fdata.drop(labels="Processed Text", axis=1,inplace=True)

fdata['Processed Text'] = clean_data(fdata['Text'])
```

```
100%|██████████| 4986/4986 [00:08<00:00, 609.65it/s]
```

```python
print(fdata.shape)
print("and" in stop_words())
```

```
(4986, 11)
True
```

```python
print(fdata['Text'])
print(fdata['Processed Text'])
```

```
0       I have bought several of the Vitality canned d...
1       Product arrived labeled as Jumbo Salted Peanut...
2       This is a confection that has been around a fe...
3       If you are looking for the secret ingredient i...
4       Great taffy at a great price.  There was a wid...
                              ...
4995    My baby didn't seem into these dinners, so I t...
4996    This is great!  Organic baby food options - de...
4997    My little guy loves to try new foods..so this ...
4998    We ordered the Earth's best 2nd dinner variety...
```

```
4999    My baby loves this food.  At whole foods they ...
Name: Text, Length: 4986, dtype: object
0       bought sever vital can dog food product found ...
1       product arriv label jumbo salt peanut peanut a...
2       confect around centuri light pillowi citru gel...
3       look secret ingredi robitussin believ found go...
4       great taffi great price wide assort yummi taff...
                               ...
4995    babi not seem dinner tri not terribl not good ...
4996    great organ babi food option deliv doorstep di...
4997    littl guy love tri new food varieti pack great...
4998    order earth best dinner varieti pack along fru...
4999    babi love food whole food sell flat that retai...
Name: Processed Text, Length: 4986, dtype: object
```

```
 reduce(lambda x,y: x+y, tqdm([ review.split(" ") for review in fdata[fdata['Score'] == 'Posit
 reduce(lambda x,y: x+y, tqdm([ review.split(" ") for review in fdata[fdata['Score'] == 'Negat
ords[0:100])
ve_words))
ve_words))
ositive_words)
```

```
100%|██████████| 4178/4178 [00:02<00:00, 1830.89it/s]
100%|██████████| 808/808 [00:00<00:00, 10543.93it/s]['bought', 'sever', 'vital', 'can', '
150287
35156
False
```

Featurization

Frequency of words

```
apw = nltk.FreqDist(all_positive_words)
anw = nltk.FreqDist(all_negative_words)
print(apw.most_common(20))
print(anw.most_common(20))
```

```
[('not', 3614), ('like', 1822), ('tast', 1651), ('good', 1577), ('flavor', 1577), ('love'
[('not', 1290), ('like', 446), ('tast', 436), ('product', 411), ('would', 327), ('one', 2
```

```
#https://buhrmann.github.io/tfidf-analysis.html
def top_features(row, features, top_n=25):
    ''' Get top n tfidf values in row and return them with their corresponding feature names.'
    topn_ids = np.argsort(row)[::-1][:top_n]
    top_feats = [(features[i], row[i]) for i in topn_ids]
    df = pd.DataFrame(top_feats)
    df.columns = ['feature', 'tfidf']
    return df
```

Unigram

```
count_vec = CountVectorizer()
count_vec.fit(fdata['Processed Text'])
print("some feature names ", count_vec.get_feature_names()[200:210])

fdata_bow = count_vec.transform(fdata['Processed Text'])
print("Type of fdata_bow  : ", type(fdata_bow))
print("BoW matrix's shape : ", fdata_bow.get_shape())
print("No of unique words : ", fdata_bow.shape[1])
```

```
some feature names  ['alley', 'alli', 'allianc', 'allot', 'allow', 'allspic', 'allud', 'a
Type of fdata_bow  :  <class 'scipy.sparse.csr.csr_matrix'>
BoW matrix's shape :  (4986, 9050)
No of unique words :  9050
```

```
all_words = count_vec.get_feature_names()
print(fdata.iloc[0]['Text'])
cx = fdata_bow[0]

print_data = "Words and their counts ::: "
for (index, count) in zip(cx.indices, cx.data):
  print_data = print_data + ("{}-{}".format(all_words[index], count))+", "

print(print_data)
```

```
I have bought several of the Vitality canned dog food products and have found them all tc
Words and their counts ::: appreci-1, better-2, bought-1, can-1, dog-1, finicki-1, food-1
```

```
print(cx.indices)
print(cx.indptr)
print(cx.data)
```

```
[ 366  715  877 1129 2324 2953 3059 3111 3378 4383 4539 4623 4854 6172
 6179 6302 7023 7234 7552 8644]
[ 0 20]
[1 2 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1]
```

Bigram & n-grams

```
count_vec = CountVectorizer(ngram_range=(1,2)) #both 1-gram and bigram
count_vec.fit(fdata['Processed Text'])
print(count_vec.get_feature_names()[0:20])
```

```
['aahhh', 'aahhh get', 'aback', 'aback brand', 'abandon', 'abat', 'abat steep', 'abbi', '
```

```
fdata_bow_2 = count_vec.transform(fdata['Processed Text'])
print("Type of fdata_bow  : ", type(fdata_bow_2))
print("BoW matrix's shape : ", fdata_bow_2.get_shape())
print("No of unique words : ", fdata_bow_2.shape[1])
```

```
Type of fdata_bow  :  <class 'scipy.sparse.csr.csr_matrix'>
BoW matrix's shape :  (4986, 122685)
```

No of unique words : 122685

```
all_words = count_vec.get_feature_names()
print(fdata.iloc[0]['Text'])
cx = fdata_bow_2[0]

print_data = "Words and their counts ::: "
for (index, count) in zip(cx.indices, cx.data):
  print_data = print_data + ("{}-{}".format(all_words[index], count))+", "

print(print_data)
```

```
I have bought several of the Vitality canned dog food products and have found them all tc
Words and their counts ::: appreci-1, appreci product-1, better-2, better labrador-1, bou
```

```
top_features(fdata_bow_2[1,:].toarray()[0], count_vec.get_feature_names())
```

	feature	tfidf
0	jumbo	2
1	product	2

TF-IDF

4	intend	1

```
count_vec = TfidfVectorizer(ngram_range=(1,2))
count_vec.fit(fdata['Processed Text'])
print(count_vec.get_feature_names()[0:20])
fdata_tfidf = count_vec.transform(fdata['Processed Text'])
```

```
['aahhh', 'aahhh get', 'aback', 'aback brand', 'abandon', 'abat', 'abat steep', 'abbi', '
```

```
print(fdata_tfidf.shape)
```

```
(4986, 122685)
```

```
print(fdata.iloc[0]['Text'])
cx = fdata_tfidf[0]

all_words = count_vec.get_feature_names()
print_data = "Words and their counts ::: "
for (index, count) in zip(cx.indices, cx.data):
  print_data = print_data + ("'{}':{}".format(all_words[index], count))+", "

print(print_data)
```

```
I have bought several of the Vitality canned dog food products and have found them all to
Words and their counts ::: 'vital can':0.19887104163834124, 'vital':0.18973021605851162,
```

```
top_features(fdata_tfidf[1,:].toarray()[0], count_vec.get_feature_names())
```

	feature	tfidf
0	jumbo	0.368562
1	peanut	0.218650
2	product jumbo	0.193159
3	size unsalt	0.193159
4	unsalt not	0.193159
5	label jumbo	0.193159
6	sure error	0.193159
7	error vendor	0.193159
8	repres product	0.193159
9	peanut actual	0.193159
10	jumbo salt	0.193159
11	peanut peanut	0.193159
12	arriv label	0.193159
13	intend repres	0.193159
14	[illegible]	[illegible]
16	actual small	0.193159

Word2Vec

```
'''
data = pd.read_sql_query("select * from Reviews where Score != 3", con)
print(data.shape)
print(data.columns)

#replace score with proper positive/negative values
data['Score'] = data['Score'].map(lambda x: "Positive" if x>3 else "Negative")
fdata = data.drop_duplicates(subset={"UserId",  "ProfileName", "Time", "Text"}, keep='first',
'''
```

```
'\ndata = pd.read_sql_query("select * from Reviews where Score != 3", con)\nprint(data.s
hape)\nprint(data.columns)\n\n#replace score with proper positive/negative values\ndata
[\'Score\'] = data[\'Score\'].map(lambda x: "Positive" if x>3 else "Negative")\nfdata =
data.drop_duplicates(subset={"UserId",\t"ProfileName", "Time", "Text"}, keep=\'first\',
inplace=False)\n'
```

```
if "TFIDF Text" not in fdata.columns:
  #fdata.drop(labels="TFIDF Text", axis=1,inplace=True)
  fdata['TFIDF Text'] = clean_data(fdata['Text'], stem_w=False)
```

```
100%|██████████| 4986/4986 [00:01<00:00, 2626.28it/s]
```

```
list_of_sentences = []
for sentence in tqdm(fdata['TFIDF Text']):
  list_of_sentences.append(sentence.split())
print(len(list_of_sentences))
```

```
100%|██████████| 4986/4986 [00:00<00:00, 259641.19it/s]4986
```

```
dimensions = 50
w2v_model = Word2Vec(tqdm(list_of_sentences), min_count=5, size=dimensions, workers=4)
```

```
100%|██████████| 4986/4986 [00:00<00:00, 98723.52it/s]
```

```
w2v_model.wv.most_similar('flavor')
```

```
[('taste', 0.9956306219100952),
 ('like', 0.9906103610992432),
 ('sweet', 0.9892805814743042),
 ('chocolate', 0.985688328742981),
 ('strong', 0.9819727540016174),
 ('hot', 0.9797263741493225),
 ('tastes', 0.977389395236969),
 ('coffee', 0.9772028923034668),
 ('cup', 0.9750277996063232),
 ('milk', 0.9654900431632996)]
```

```
w2v_model.wv.similarity("flavor", "taste")
```

```
0.99563056
```

```
w2v_model.wv['taste']
```

```
array([ 0.31400645,  0.35826513,  0.11272779, -0.36115277, -0.22070578,
        0.1877151 , -0.19483876,  0.67671824,  0.09352529,  0.53199154,
        0.40332803, -0.81506085,  0.21533908,  0.02029351, -0.49281758,
       -0.19159071,  0.532863  ,  0.57779664,  0.10145646,  0.3251961 ,
       -0.04902818, -0.02896426, -0.09993342, -0.58901256,  0.66139174,
       -0.89219964,  0.12459397, -0.7702163 , -0.00181521,  1.129645  ,
        0.40801245, -0.8234921 ,  0.37883705, -0.22841604,  0.7392538 ,
        0.5194731 ,  0.11135306, -0.44463718, -0.85259855,  0.61243194,
       -0.48818505,  0.40567145,  0.35114944,  0.34433004,  0.5892858 ,
       -0.34284878, -0.22800311,  0.49495897,  1.0624708 , -0.1053092 ],
      dtype=float32)
```

Avg & tf-idf weighted Word2Vec

```
vectors = []
w2v_words = list(w2v_model.wv.vocab)

for sentence in list_of_sentences:
  temp_vec = np.zeros(dimensions)
  count = 0
  for word in sentence:
    try:
      if word in w2v_words:
        temp_vec += w2v_model.wv[word]
        count += 1
    except:
```

```
    except:
      pass
  if count > 0:
    temp_vec /= count
  vectors.append(temp_vec)

print(len(vectors))
print(len(vectors[0]))
```

```
4986
50
```

```
w2v_model.wv.similar_by_word("bought")
```

```
[('pet', 0.9936877489089966),
 ('box', 0.9931565523147583),
 ('carrying', 0.9926332831382751),
 ('cheaper', 0.9923766851425171),
 ('found', 0.9922744035720825),
 ('ordered', 0.9920716881752014),
 ('com', 0.9913408160209656),
 ('expensive', 0.9908621311187744),
 ('online', 0.990604043006897),
 ('order', 0.9905717968940735)]
```

```
print(vectors[0])
```

```
[ 0.14641876  0.52827878 -0.0993959  -0.60903482 -0.08402478  0.60290023
 -0.02877809  0.18023748 -0.23364483  0.24489117  0.43388439 -0.58803605
 -0.11811611 -0.0392796   0.07474669 -0.1017498   0.47070879  0.47110651
  0.22118492  0.09940586  0.36695456  0.48987006 -0.09364202 -0.25199192
  0.32419532 -0.16184854 -0.02182628 -0.79156137  0.27281518  0.63075193
  0.26040206 -0.50866942 -0.10360876 -0.09779876  0.48042295  0.50790102
  0.00669679 -0.68515561 -0.92944834  0.13607911 -0.22111696  0.26557272
  0.13511326  0.54070132  0.53600736 -0.33063295 -0.19789684  0.56785565
  0.47458754 -0.02977765]
```

```
tf_idf_model = TfidfVectorizer()
tf_idf_model.fit(fdata['Processed Text'])
fdata_tfidf = tf_idf_model.transform(fdata['Processed Text'])
```

```
print(len(tf_idf_model.idf_))
print(len(tf_idf_model.get_feature_names()))
words_vs_idf = dict(zip(tf_idf_model.get_feature_names(), list(tf_idf_model.idf_)))
```

```
9050
9050
```

```
vectors = []
for sentence in tqdm(list_of_sentences):
  temp_vec = np.zeros(dimensions)
  tf_add = 0
  for word in sentence:
    try:
      if word in w2v_words and word in words_vs_idf:
```

```
        tf_val = words_vs_idf[word] #across corpus
        tf_val = tf_val * sentence.count(word)/len(sentence)
        w2v_val = w2v_model.wv[word]
        temp_vec += (w2v_val*tf_val)
        tf_add += tf_val
    except:
      pass
  vectors.append(temp_vec/tf_add)

print(len(vectors[0]))
print(vectors[0])
```

```
  0%|          | 0/4986 [00:00<?, ?it/s]/usr/local/lib/python3.7/dist-packages/ipykernel_
  from ipykernel import kernelapp as app
100%|██████████| 4986/4986 [00:06<00:00, 791.80it/s]50
[ 0.15859548  0.53520589 -0.11226889 -0.64228139 -0.09953286  0.62202132
 -0.0294062   0.15563168 -0.2675561   0.22790046  0.45630022 -0.61150571
 -0.11868267 -0.05107184  0.09024587 -0.10530289  0.46926174  0.49347503
  0.21566485  0.07872581  0.38972335  0.50289167 -0.09117014 -0.2514417
  0.32642028 -0.14351982 -0.00683054 -0.82131423  0.26984106  0.62049635
  0.27244595 -0.49327634 -0.13240512 -0.0893658   0.46390479  0.51806883
  0.00250215 -0.71116737 -0.91896131  0.13529064 -0.20075444  0.27946065
  0.11988801  0.55603091  0.54397109 -0.32905527 -0.19511487  0.61551135
  0.45882511 -0.02013739]
```

✓ 6s completed at 10:33 AM