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 Score/Rating. A rating of 4 or 5 can be cosnidered as a positive review. A rating of 1 or 2 can is considered nuetral and such reviews are ignored from our analysis. This is an approximate and proxy way of d review.

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it is easier to query the data and visualise the data

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purpose above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

```
%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


from google.colab import drive


drive.mount('/content/drive')


%cd ./drive/My Drive
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=9473

Enter your authorization code:
··········
Mounted at /content/drive
/content/drive/My Drive
```

```
# using SQLite Table to read data.
con = sqlite3.connect('database.sqlite')

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data points
# you can change the number to any other number based on your computing power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000""", c
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 100000""", con

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negative rating(0)
def partition(x):
    if x < 3:
        return 0
    return 1
```

```
#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

```
Number of data points in our data (100000, 10)
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulne
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	

```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

```
print(display.shape)
display.head()
```

```
(80668, 7)
```

	UserId	ProductId	ProfileName	Time	Score	
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	Ove
1	#oc-R11D9D7SHXIJB9	B005HG9ET0	Louis E. Emory "hoppy"	1342396800	5	My wife
2	#oc-R11DNU2NBKQ23Z	B007Y59HVM	Kim Cieszykowski	1348531200	1	Th
3	#oc-R11O5J5ZVQE25C	B005HG9ET0	Penguin Chick	1346889600	5	Th
4	#oc-R12KPBODL2B5ZD	B007OSBE1U	Christopher P. Presta	1348617600	1	

```
display[display['UserId']=='AZY10LLTJ71NX']
```

	UserId	ProductId	ProfileName	Time	Score	
80638	AZY10LLTJ71NX	B006P7E5ZI	undertheshrine "undertheshrine"	1334707200	5	I w

```
display['COUNT(*)'].sum()
```

```
393063
```

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

It is observed (as shown in the table below) that the reviews data had many duplicate entries. Hence it was nece
results for the analysis of the data. Following is an example:

```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, Helpfu
and on doing analysis it was found that

ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8)

ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and s

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same pro
in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first
eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one
without sorting would lead to possibility of different representatives still existing for the same product.

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='q
```

```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', in
final.shape
```

```
(87775, 10)
```

```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

```
87.775
```

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than H possible hence these two rows too are removed from calcualtions

```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	

```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

```
#Before starting the next phase of preprocessing lets see the number of entries left
print(final.shape)

#How many positive and negative reviews are present in our dataset?
final['Score'].value_counts()
```

```
(87773, 10)
1    73592
0    14181
Name: Score, dtype: int64
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

Now that we have finished deduplication our data requires some preprocessing before we go on further with ana

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
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

```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)

sent_1000 = final['Text'].values[1000]
print(sent_1000)
print("="*50)

sent_1500 = final['Text'].values[1500]
print(sent_1500)
print("="*50)

sent_4900 = final['Text'].values[4900]
print(sent_4900)
print("="*50)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anym
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has lit
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. So if you
==================================================
```

```
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anym
```

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-from-a
from bs4 import BeautifulSoup

soup = BeautifulSoup(sent_0, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1000, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1500, 'lxml')
text = soup.get_text()
```

```
print(text)
print("="*50)

soup = BeautifulSoup(sent_4900, 'lxml')
text = soup.get_text()
print(text)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anym
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has lit
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. So if you
```

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase


sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
was way to hot for my blood, took a bite and did a jig  lol
==================================================
```

```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anym
```

```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
was way to hot for my blood took a bite and did a jig lol
```

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you',
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'hi
```

```
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'the
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 't
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'havin
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until',
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during'
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'unde
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both',
            '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', "migh
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "was
            'won', "won't", 'wouldn', "wouldn't"])
```

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 87773/87773 [00:35<00:00, 2455.93it/s]
```

```
preprocessed_reviews[1500]
```

```
'way hot blood took bite jig lol'
```

[3.2] Preprocessing Review Summary

```
## Similarly you can do preprocessing for review summary also.
```

[4] Featurization

[4.1] BAG OF WORDS

[4.2] Bi-Grams and n-Grams.

[4.3] TF-IDF

[4.4] Word2Vec

[4.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

[4.4.1.2] TFIDF weighted W2v

[5] Assignment 11: Truncated SVD

1. **Apply Truncated-SVD on only this feature set:**
    - SET 2:Review text, preprocessed one converted into vectors using (TFIDF)
    - **Procedure:**
        - Take top 2000 or 3000 features from tf-idf vectorizers using idf_ score.
        - You need to calculate the co-occurrence matrix with the selected features (Note: X.X^T doesı covariance matrix, check these bolgs blog-1, blog-2 for more information)
        - You should choose the n_components in truncated svd, with maximum explained variance. P them. (hint: plot of cumulative explained variance ratio)
        - After you are done with the truncated svd, you can apply K-Means clustering and choose the
        - Print out wordclouds for each cluster, similar to that in previous assignment.
        - You need to write a function that takes a word and returns the most similar words using cosi the matrix after truncatedSVD)

Truncated-SVD

[5.1] Taking top features from TFIDF, SET 2

```
from sklearn.feature_extraction.text import TfidfVectorizer

X=preprocessed_reviews
#applying tfidf vectorizer upon the preprocessed reviews
tf_idf_vect = TfidfVectorizer()

X_tf = tf_idf_vect.fit_transform(X)

a=tf_idf_vect.get_feature_names()


b=tf_idf_vect.idf_
b=b.tolist()

#https://codereview.stackexchange.com/questions/138702/sorting-two-lists

c,d=(list(t) for t in zip(*sorted(zip(b,a))))

idf_scores=c[0:3000]
top_features=d[0:3000]
```

[5.2] Calulation of Co-occurrence matrix

```
#Rechecking the co_occurence matrix

L_0=["abc def ijk pqr", "pqr klm opq", "lmn pqr xyz abc def pqr abc"]
top_features_0 = ["abc", "pqr", "def"]

# copied from https://github.com/niketan108/Truncated-SVD-on-amazon-food/blob/master/11%20Amazon%
n_neighbor = 2
occ_matrix_0 = np.zeros((3,3))
for row in L_0:
    words_in_row = row.split()
    for index,word in enumerate(words_in_row):
        if word in top_features_0:
            for j in range(max(index-n_neighbor,0),min(index+n_neighbor,len(words_in_row)-1) + 1)
                if words_in_row[j] in top_features_0:
                    occ_matrix_0[top_features_0.index(word),top_features_0.index(words_in_row[j])
                else:
                    continue
        else:
            continue



print(occ_matrix_0)
```

```
[[3. 3. 3.]
 [3. 4. 2.]
 [3. 2. 2.]]
```

```
# copied from https://github.com/niketan108/Truncated-SVD-on-amazon-food/blob/master/11%20Amazon%
n_neighbor = 2
occ_matrix = np.zeros((3000,3000))
for row in tqdm(X):
    words_in_row = row.split()
    for index,word in enumerate(words_in_row):
        if word in top_features:
            for j in range(max(index-n_neighbor,0),min(index+n_neighbor,len(words_in_row)-1) + 1)
                if words_in_row[j] in top_features:
                    occ_matrix[top_features.index(word),top_features.index(words_in_row[j])] += 1
                else:
                    continue
        else:
            continue
```

```
100%|██████████| 87773/87773 [07:42<00:00, 189.97it/s]
```

[5.3] Finding optimal value for number of components (n) to be retained.

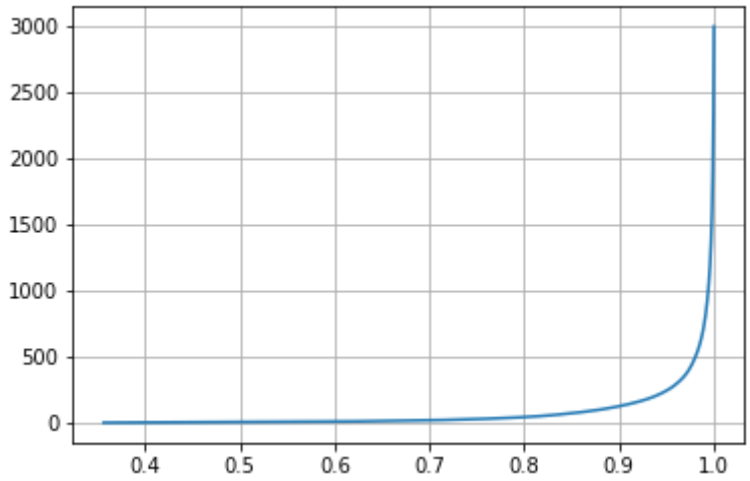
```
from sklearn.decomposition import TruncatedSVD
from tqdm import tqdm

evr=[] #explained_varaiance_ratio
for i in  tqdm(range(1,3000,10)):
  svd = TruncatedSVD(n_components=i)
  svd.fit(occ_matrix)
  evr.append(svd.explained_variance_ratio_.sum())
```

```
100%|██████████| 300/300 [1:57:40<00:00, 54.43s/it]
```

```
k=[i for i in range(1,3000,10)]
plt.plot(evr,k)
plt.grid()
plt.plot()
```

```
[]
```




```
n_components=400
svd = TruncatedSVD(n_components)
svd=svd.fit_transform(occ_matrix)
```

```
print(svd.shape)
```

```
(3000, 400)
```

[5.4] Applying k-means clustering

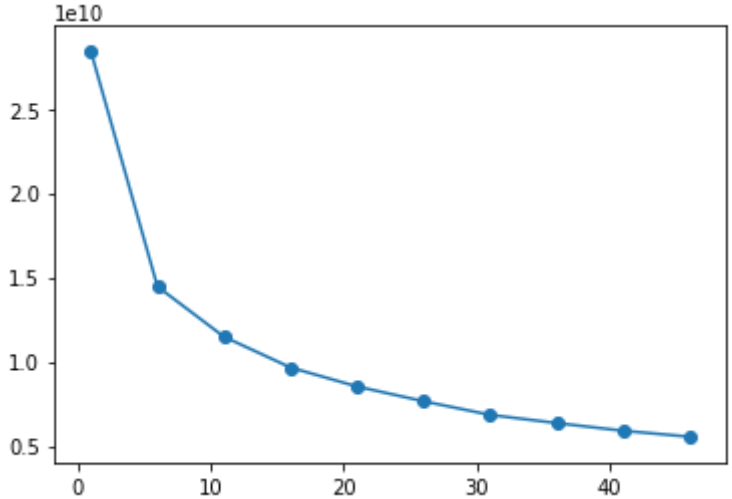
```
from sklearn.cluster import KMeans
#Hyper parameter tuning
inertia = []

K = range(1,50,5)
for i in K:

  kmeans = KMeans(n_clusters=i, random_state=0).fit(svd)
  inertia.append(kmeans.inertia_)

#plot of hyperparameter vs square of intra cluster distances
plt.plot(K, inertia)
plt.scatter(K,inertia)
```

```
<matplotlib.collections.PathCollection at 0x7f8800ccda20>
```



```
print(kmeans.labels_.shape)
```

```
(3000,)
```

[5.5] Wordclouds of clusters obtained in the above section

```
kmeans = KMeans(n_clusters=5, random_state=0).fit(svd)

#constructing a pandas dataframe containing Reviews and cluster labels
df=pd.DataFrame(top_features,columns=['Top_features'])
df['cluster']=kmeans.labels_
print(df['cluster'].value_counts())
```

```
0    2996
3       1
1       1
4       1
2       1
Name: cluster, dtype: int64
```

```
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator


for i in [0,1,3]:
  a = " ".join( df[df["cluster"]==i].Top_features)

  # Create and generate a word cloud image:
  wordcloud = WordCloud().generate(a)

  # Display the generated image:
  plt.imshow(wordcloud, interpolation='bilinear')
  plt.axis("off")
  plt.show()
```

[5.6] Function that returns most similar words for a given word.

```
from sklearn.metrics.pairwise import cosine_similarity
def similarity(num):

#obtaining cosine similarity
  sim=cosine_similarity(occ_matrix)

  print("words similar to ",top_features[num-1],"are")
#obtaining similarity scores corresponding to a particular word
  lsm=sim[num-1]
#sorting simiarity scores by argument and getting top 20 values
  k=lsm.argsort()[::-1]
  k=k[1:11]

  for i in k :
    print(top_features[i])
```

```
for i in range (0,3000,300):
  similarity(i)
  print()
```

```
words similar to  pretzel are
not
sticks
like
taste
disappoint
necessarily
ca
hesitate
bother
overpowering

words similar to  cheese are
mac
macaroni
parmesan
velveeta
cheddar
shells
shredded
kraft
cream
sauce

words similar to  told are
not
vet
friend
doctor
truth
would
hesitate
contacted
could
necessarily

words similar to  thrilled are
find
amazon
com
not
able
locally
anywhere
delighted
stores
could

words similar to  freshness are
maintain
not
ensure
taste
disappoint
quality
sealed
necessarily
hesitate
bother

words similar to  babies are
```

```
not
fur
us
love
eat
food
hesitate
necessarily
feed
disappoint

words similar to  talking are
know
not
disappoint
necessarily
bother
hesitate
ca
stop
overpowering
people

words similar to  blender are
mix
ice
put
use
milk
water
blend
add
make
cubes

words similar to  machines are
keurig
coffee
espresso
maker
not
drip
snob
tassimo
drinker
drinkers

words similar to  muffin are
mix
blueberry
pancake
trail
english
bran
brownie
like
pan
waffle
```