Matrix Factorisation on Amazon reviews dataset

Data Source: 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:

Find similar word cluster for a given word

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 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"

Also we sort data by time-based slicing

Loading Preprocessed Data

I have preprocessed the data separately for 250k points and stored in cleanedreviews.csv

```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Mounted at /content/drive/
```

```
!pip install glove_python
```

```
Requirement already satisfied: glove_python in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (f
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (f
```

```
from glove import Corpus, Glove
```

```
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
```

```
#Loading preprocessed data
df=pd.read_csv('drive/My Drive/amazon/cleanedreviews.csv')
df.head()
```

	Unnamed: 0	Id	ProductId	UserId	ProfileName	HelpfulnessNume
0	0	150524	0006641040	ACITT7DI6IDDL	shari zychinski	
1	1	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	
2	2	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	

```
df['Score'].value_counts()
```

```
1    215035
0     34965
Name: Score, dtype: int64
```

```
df['Class']=['positive' if s==1 else 'negative' for s in df['Score']]
```

```
sub_data=df[0:250000]
sub_data.head()
```

	Unnamed: 0	Id	ProductId	UserId	ProfileName	HelpfulnessNume
0	0	150524	0006641040	ACITT7DI6IDDL	shari zychinski	
1	1	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	

```
#coverting reviews in lists of words i-e, for each review a list of words will created
list_of_sent=[]
for sent in sub_data['Text'].values:
    filtered_sentence=[]
    for w in sent.split():
        for cleaned_words in w.split():
            if(cleaned_words.isalpha()):
                filtered_sentence.append(cleaned_words.lower())
            else:
                continue
    list_of_sent.append(filtered_sentence)
```

```
#Using Corpus to construct  co-occurance matrix
corpus=Corpus()
corpus.fit(list_of_sent,window=5)
```

```
#creating a Glove object which will use the matrix created in the above lines to create
#We can set the learning rate as it uses Gradient Descent and number of components

glove = Glove(no_components=5, learning_rate=0.05)

glove.fit(corpus.matrix, epochs=30, no_threads=4, verbose=True)
glove.add_dictionary(corpus.dictionary)
glove.save('glove.model')
```

```
#After the training glove object has the word vectors for the lines we have provided. Bu
#We need to add the dictionary to the glove object to make it complete. by using :glove.
```

```
Performing 30 training epochs with 4 threads
Epoch 0
Epoch 1
Epoch 2
Epoch 3
Epoch 4
Epoch 5
Epoch 6
Epoch 7
Epoch 8
Epoch 9
Epoch 10
Epoch 11
```

```
!pip install wordcloud
```

```
Collecting wordcloud
  Downloading https://files.pythonhosted.org/packages/ae/af/849edf14d573eba9c8082
    100% |████████████████████████████████| 368kB 23.2MB/s
Requirement already satisfied: numpy>=1.6.1 in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: pillow in /usr/local/lib/python3.6/dist-packages (
Requirement already satisfied: olefile in /usr/local/lib/python3.6/dist-packages
Installing collected packages: wordcloud
Successfully installed wordcloud-1.5.0
```

```
#importing word cloud
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
```

```
Epoch 27
```

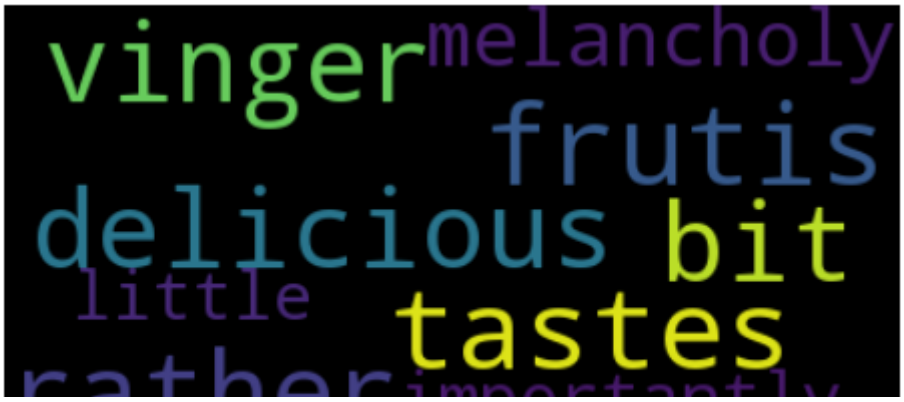
```
#function to print wordcloud for differnt words
def cloud(c):
  wordcloud = WordCloud(max_font_size=50, max_words=100,collocations=False).generate_frc
  plt.figure()
  plt.imshow(wordcloud, interpolation="bilinear")
  plt.axis("off")
  plt.show()
```

```
similar=glove.most_similar('yummy',number=10)
```

```
similar
```

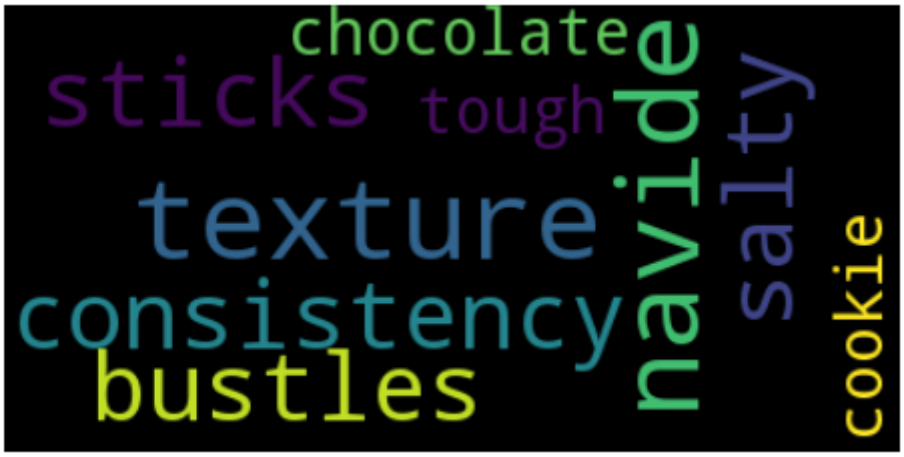
```
[('frutis', 0.996492443083793),
 ('tastes', 0.9944467571247136),
 ('delicious', 0.9935245182171553),
 ('vinger', 0.9925299089629726),
 ('rather', 0.9910518457238957),
 ('bit', 0.9905045859027032),
 ('melancholy', 0.9889789964338971),
 ('importantly', 0.9885956397653726),
 ('little', 0.9873806119969238)]
```

```
#wordcloud for Yummy
words=dict(similar)
cloud(words)
```



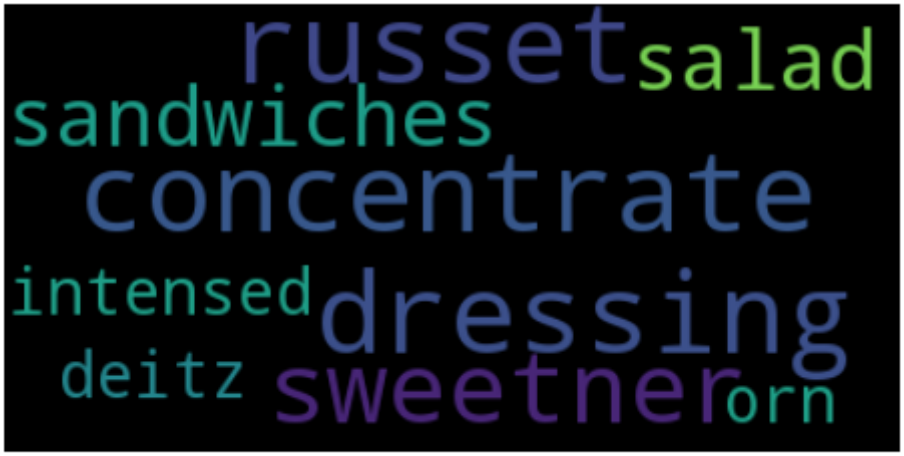

```
#tasty
words=glove.most_similar('tasty',number=10)
words=dict(words)
cloud(words)
```


```
#spicy
words=glove.most_similar('spicy',number=10)
words=dict(words)
cloud(words)
```

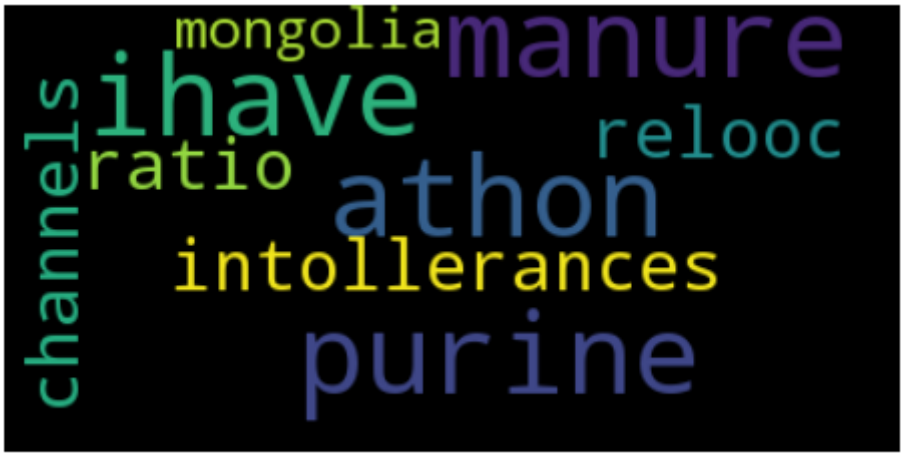



```
words=glove.most_similar('biryani',number=10)
words=dict(words)
cloud(words)
```

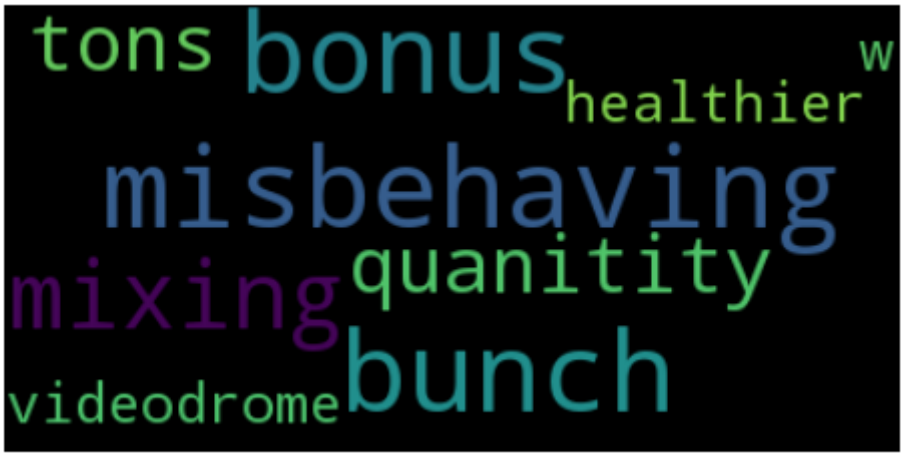

```
words=glove.most_similar('pizza',number=10)
words=dict(words)
cloud(words)
```




```
words=glove.most_similar('protien',number=10)
words=dict(words)
cloud(words)
```




```
words=glove.most_similar('healthy',number=10)
words=dict(words)
cloud(words)
```


tons
bonus
w
healthier
misbehaving
quanitity
mixing
bunch
videodrome