CSCI 544 HOMEWORK 3

NAME: Asmita Chotani
USC ID: 3961468036

In [1]:
```
!pip install -U gensim
```

```
Requirement already satisfied: gensim in /Users/asmitachotani/opt/miniconda3/l
ib/python3.9/site-packages (4.3.0)
Requirement already satisfied: scipy>=1.7.0 in /Users/asmitachotani/opt/minico
nda3/lib/python3.9/site-packages (from gensim) (1.10.0)
Requirement already satisfied: smart-open>=1.8.1 in /Users/asmitachotani/opt/m
iniconda3/lib/python3.9/site-packages (from gensim) (6.3.0)
Requirement already satisfied: FuzzyTM>=0.4.0 in /Users/asmitachotani/opt/mini
conda3/lib/python3.9/site-packages (from gensim) (2.0.5)
Requirement already satisfied: numpy>=1.18.5 in /Users/asmitachotani/opt/minic
onda3/lib/python3.9/site-packages (from gensim) (1.23.5)
Requirement already satisfied: pyfume in /Users/asmitachotani/opt/miniconda3/l
ib/python3.9/site-packages (from FuzzyTM>=0.4.0->gensim) (0.2.25)
Requirement already satisfied: pandas in /Users/asmitachotani/opt/miniconda3/l
ib/python3.9/site-packages (from FuzzyTM>=0.4.0->gensim) (1.5.3)
Requirement already satisfied: python-dateutil>=2.8.1 in /Users/asmitachotani/
opt/miniconda3/lib/python3.9/site-packages (from pandas->FuzzyTM>=0.4.0->gensi
m) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /Users/asmitachotani/opt/minico
nda3/lib/python3.9/site-packages (from pandas->FuzzyTM>=0.4.0->gensim) (2022.
7.1)
Requirement already satisfied: six>=1.5 in /Users/asmitachotani/opt/miniconda
3/lib/python3.9/site-packages (from python-dateutil>=2.8.1->pandas->FuzzyTM>=
0.4.0->gensim) (1.16.0)
Requirement already satisfied: simpful in /Users/asmitachotani/opt/miniconda3/
lib/python3.9/site-packages (from pyfume->FuzzyTM>=0.4.0->gensim) (2.9.0)
Requirement already satisfied: fst-pso in /Users/asmitachotani/opt/miniconda3/
lib/python3.9/site-packages (from pyfume->FuzzyTM>=0.4.0->gensim) (1.8.1)
Requirement already satisfied: miniful in /Users/asmitachotani/opt/miniconda3/
lib/python3.9/site-packages (from fst-pso->pyfume->FuzzyTM>=0.4.0->gensim) (0.
0.6)
Requirement already satisfied: requests in /Users/asmitachotani/opt/miniconda
3/lib/python3.9/site-packages (from simpful->pyfume->FuzzyTM>=0.4.0->gensim)
(2.27.1)
Requirement already satisfied: idna<4,>=2.5 in /Users/asmitachotani/opt/minico
nda3/lib/python3.9/site-packages (from requests->simpful->pyfume->FuzzyTM>=0.
4.0->gensim) (3.3)
Requirement already satisfied: certifi>=2017.4.17 in /Users/asmitachotani/opt/
miniconda3/lib/python3.9/site-packages (from requests->simpful->pyfume->FuzzyT
M>=0.4.0->gensim) (2022.5.18.1)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /Users/asmitachotani/o
pt/miniconda3/lib/python3.9/site-packages (from requests->simpful->pyfume->Fuz
zyTM>=0.4.0->gensim) (1.26.9)
Requirement already satisfied: charset-normalizer~=2.0.0 in /Users/asmitachota
ni/opt/miniconda3/lib/python3.9/site-packages (from requests->simpful->pyfume-
>FuzzyTM>=0.4.0->gensim) (2.0.4)
```

In [2]:
```
pip install -U scikit-learn scipy matplotlib
```

```
Requirement already satisfied: scikit-learn in /Users/asmitachotani/opt/anacon
da3/envs/pytorch_a1/lib/python3.9/site-packages (1.2.1)
Requirement already satisfied: scipy in /Users/asmitachotani/opt/anaconda3/env
s/pytorch_a1/lib/python3.9/site-packages (1.10.1)
Requirement already satisfied: matplotlib in /Users/asmitachotani/opt/anaconda
3/envs/pytorch_a1/lib/python3.9/site-packages (3.7.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /Users/asmitachotani/op
t/anaconda3/envs/pytorch_a1/lib/python3.9/site-packages (from scikit-learn)
(3.1.0)
Requirement already satisfied: joblib>=1.1.1 in /Users/asmitachotani/opt/anaco
nda3/envs/pytorch_a1/lib/python3.9/site-packages (from scikit-learn) (1.2.0)
Requirement already satisfied: numpy>=1.17.3 in /Users/asmitachotani/opt/anaco
nda3/envs/pytorch_a1/lib/python3.9/site-packages (from scikit-learn) (1.23.5)
Requirement already satisfied: pillow>=6.2.0 in /Users/asmitachotani/opt/anaco
nda3/envs/pytorch_a1/lib/python3.9/site-packages (from matplotlib) (9.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /Users/asmitachotani/opt/an
aconda3/envs/pytorch_a1/lib/python3.9/site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: kiwisolver>=1.0.1 in /Users/asmitachotani/opt/a
naconda3/envs/pytorch_a1/lib/python3.9/site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: cycler>=0.10 in /Users/asmitachotani/opt/anacon
da3/envs/pytorch_a1/lib/python3.9/site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: contourpy>=1.0.1 in /Users/asmitachotani/opt/an
aconda3/envs/pytorch_a1/lib/python3.9/site-packages (from matplotlib) (1.0.5)
Requirement already satisfied: fonttools>=4.22.0 in /Users/asmitachotani/opt/a
naconda3/envs/pytorch_a1/lib/python3.9/site-packages (from matplotlib) (4.25.
0)
Requirement already satisfied: python-dateutil>=2.7 in /Users/asmitachotani/op
t/anaconda3/envs/pytorch_a1/lib/python3.9/site-packages (from matplotlib) (2.
8.2)
Requirement already satisfied: packaging>=20.0 in /Users/asmitachotani/opt/ana
conda3/envs/pytorch_a1/lib/python3.9/site-packages (from matplotlib) (22.0)
Requirement already satisfied: importlib-resources>=3.2.0 in /Users/asmitachot
ani/opt/anaconda3/envs/pytorch_a1/lib/python3.9/site-packages (from matplotli
b) (5.12.0)
Requirement already satisfied: zipp>=3.1.0 in /Users/asmitachotani/opt/anacond
a3/envs/pytorch_a1/lib/python3.9/site-packages (from importlib-resources>=3.2.
0->matplotlib) (3.13.0)
Requirement already satisfied: six>=1.5 in /Users/asmitachotani/opt/anaconda3/
envs/pytorch_a1/lib/python3.9/site-packages (from python-dateutil>=2.7->matplo
tlib) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

In [3]:
```python
from sklearn.metrics.pairwise import cosine_similarity
```

In [4]:
```python
pip install contractions
```

```
Requirement already satisfied: contractions in /Users/asmitachotani/opt/anacon
da3/envs/pytorch_a1/lib/python3.9/site-packages (0.1.73)
Requirement already satisfied: textsearch>=0.0.21 in /Users/asmitachotani/opt/
anaconda3/envs/pytorch_a1/lib/python3.9/site-packages (from contractions) (0.
0.24)
Requirement already satisfied: pyahocorasick in /Users/asmitachotani/opt/anaco
nda3/envs/pytorch_a1/lib/python3.9/site-packages (from textsearch>=0.0.21->con
tractions) (2.0.0)
Requirement already satisfied: anyascii in /Users/asmitachotani/opt/anaconda3/
envs/pytorch_a1/lib/python3.9/site-packages (from textsearch>=0.0.21->contract
ions) (0.3.1)
Note: you may need to restart the kernel to use updated packages.
```

In [5]:
```
import pandas as pd
import numpy as np
import nltk
nltk.download('wordnet')
nltk.download('punkt')  # for word tokenizing
nltk.download('stopwords') # for determining stop words taht have to be removed
nltk.download('omw-1.4') # for lemmatizing

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron
from sklearn.metrics import classification_report,accuracy_score
from sklearn import svm
from sklearn.svm import LinearSVC
from sklearn.model_selection import GridSearchCV

import gensim

import re
from bs4 import BeautifulSoup
import warnings
warnings.filterwarnings('ignore')
import string
import contractions
import copy
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/asmitachotani/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/asmitachotani/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/asmitachotani/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     /Users/asmitachotani/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

In [6]:
```
from torch.optim.lr_scheduler import ReduceLROnPlateau
```

In [7]:
```
# import libraries
import torch
import torchvision
from torch import nn
import torch.nn as nn
import torch.nn.functional as F
from torch.nn import CrossEntropyLoss, Softmax, Linear
from torch.optim import SGD, Adam
```

In [8]:
```
# df2 = pd.read_csv('./amazon_reviews_us_Beauty_v1_00.tsv',
#                                  sep='\t',
#                                  error_bad_lines=False
#                                 )
```

In [9]:
```
df3 = pd.read_csv('./amazon_reviews_us_Beauty_v1_00.tsv',
                                sep='\t',
                                error_bad_lines=False,
```

```
                                usecols=["star_rating", "review_body"]
                               )
display(df3)
```

	star_rating	review_body
0	5	Love this, excellent sun block!!
1	5	The great thing about this cream is that it do...
2	5	Great Product, I'm 65 years old and this is al...
3	5	I use them as shower caps & conditioning caps....
4	5	This is my go-to daily sunblock. It leaves no ...
...	...	...
5094558	5	After watching my Dad struggle with his scisso...
5094559	3	Like most sound machines, the sounds choices a...
5094560	5	I bought this product because it indicated 30 ...
5094561	5	We have used Oral-B products for 15 years; thi...
5094562	5	I love this toothbrush. It's easy to use, and ...

5094563 rows × 2 columns

In [10]:
```
# understanding the different rating values that are possible
df3['star_rating'].unique()
```

Out[10]:
```
array(['5', '4', '1', '3', '2', '2015-08-28', '2015-08-16', '2015-08-14',
       5, 4, 3, 1, 2, '2015-07-27', '2015-07-26', '2015-07-23',
       '2015-07-22', nan, '2015-06-14', '2015-06-02', '2015-04-14',
       '2015-04-09', '2015-04-08', '2015-04-03', '2015-04-02',
       '2015-04-01', '2015-03-31', '2015-03-30', '2015-03-18',
       '2015-02-28', '2015-02-10', '2014-12-30', '2014-12-03',
       '2014-10-09'], dtype=object)
```

In [11]:
```
# creating a copy of the dataframe to work with
df=df3.copy()
```

We form three classes and select 20000 reviews randomly from each class.

In [12]:
```
# 3 classes are formed for the 5 kinds of ratings possible.
def categorise(row):
    if row['star_rating'] ==  1 or row['star_rating']== '1' or row['star_rating
        return 1
    elif row['star_rating'] ==  2 or row['star_rating']== '2'or row['star_ratin
        return 1
    elif row['star_rating'] == 3 or row['star_rating']== '3'or row['star_rating
        return 2
    elif row['star_rating'] ==  4 or row['star_rating']== '4'or row['star_ratin
        return 3
    elif row['star_rating'] ==  5 or row['star_rating']== '5'or row['star_ratin
        return 3
```

```
    else:
        return 0    # the entries with invalid values in the rating column
```

In [13]:

```
%%time
df['class'] = df.apply(lambda row: categorise(row), axis=1)
display(df)
```

	star_rating	review_body	class
0	5	Love this, excellent sun block!!	3
1	5	The great thing about this cream is that it do...	3
2	5	Great Product, I'm 65 years old and this is al...	3
3	5	I use them as shower caps & conditioning caps....	3
4	5	This is my go-to daily sunblock. It leaves no ...	3
...	...	...	...
5094558	5	After watching my Dad struggle with his scisso...	3
5094559	3	Like most sound machines, the sounds choices a...	2
5094560	5	I bought this product because it indicated 30 ...	3
5094561	5	We have used Oral-B products for 15 years; thi...	3
5094562	5	I love this toothbrush. It's easy to use, and ...	3

5094563 rows × 3 columns

```
CPU times: user 1min 57s, sys: 691 ms, total: 1min 57s
Wall time: 1min 58s
```

In [14]:

```
%%time
# Creating separate dataframes for separate classes
S1_dfa = df.loc[df['class'] == 1]
S2_dfa = df.loc[df['class'] == 2]
S3_dfa = df.loc[df['class'] == 3]

# COnsidering only 20000 data entries for each class
S1_df=S1_dfa.sample(n=20000)
S2_df=S2_dfa.sample(n=20000)
S3_df=S3_dfa.sample(n=20000)
```

```
CPU times: user 508 ms, sys: 860 ms, total: 1.37 s
Wall time: 1.37 s
```

In [15]:

```
# Concatenating 20000 reviews for each class into one dataframe that we will wo
review_df = pd.concat([S1_df, S2_df, S3_df])
display(review_df)
```

	star_rating	review_body	class
3319364	1	Started out okay but after using for a little ...	1
4788188	2	I'm on the constant search for the perfect con...	1
4864622	1	I just bought this at a Home Show yesterday, a...	1
3643662	1	Ordered 2 one came smooshed and broken... If i...	1
5053184	1	It may work for some people, but it did not wo...	1
...	...	...	...
3262464	5	Very happy with this product. Looking forward ...	3
2849181	5	Keeps me dry. Just what I wanted. There is lit...	3
3060043	5	I bought these for a trip, I needed a bra that...	3
670266	5	Love it!! Does the job well and very little or...	3
1425573	5.0	good stuff	3

60000 rows × 3 columns

Data Cleaning

Reseting Index

In [16]:
```
# Since we have randomly chosen 20000 entries from each class, it is necessary
# repitition of entries.
review_df = review_df.reset_index(drop=True)
display(review_df)
```

	star_rating	review_body	class
0	1	Started out okay but after using for a little ...	1
1	2	I'm on the constant search for the perfect con...	1
2	1	I just bought this at a Home Show yesterday, a...	1
3	1	Ordered 2 one came smooshed and broken... If i...	1
4	1	It may work for some people, but it did not wo...	1
...	...	...	...
59995	5	Very happy with this product. Looking forward ...	3
59996	5	Keeps me dry. Just what I wanted. There is lit...	3
59997	5	I bought these for a trip, I needed a bra that...	3
59998	5	Love it!! Does the job well and very little or...	3
59999	5.0	good stuff	3

60000 rows × 3 columns

In [17]:
```
# Checking for null values
review_df.isnull().values.any()
```

Out[17]:
```
True
```

In [18]:
```
# Checking number of null values in the two columns
review_df.isnull().sum()
```

Out[18]:
```
star_rating    0
review_body    4
class          0
dtype: int64
```

In [19]:
```
# Filling the null values with an empty string as only empty value is in the re
review_df = review_df.fillna('')
```

In [20]:
```
wv_data=review_df.copy()
```

In [21]:
```
pre_dc = review_df['review_body'].str.len().mean()
```

In [22]:
```
#Converting the reviews into Lower Case
review_df['review_body'] = review_df['review_body'].str.lower()
display(review_df)
```

	star_rating	review_body	class
0	1	started out okay but after using for a little ...	1
1	2	i'm on the constant search for the perfect con...	1
2	1	i just bought this at a home show yesterday, a...	1
3	1	ordered 2 one came smooshed and broken... if i...	1
4	1	it may work for some people, but it did not wo...	1
...	...	...	...
59995	5	very happy with this product. looking forward ...	3
59996	5	keeps me dry. just what i wanted. there is lit...	3
59997	5	i bought these for a trip, i needed a bra that...	3
59998	5	love it!! does the job well and very little or...	3
59999	5.0	good stuff	3

60000 rows × 3 columns

In [23]:
```
# Removing well-formed tags i.e the HTML and URLs
review_df['review_body'] = review_df['review_body'].str.replace(r'<[^<>]*>', ''
review_df['review_body'] = review_df['review_body'].apply(lambda x: re.split('h
```

In [24]:
```
def remove_mention_tag_fn(text):
    text = re.sub(r'@\S*', '', text)
    return re.sub(r'#\S*', '', text)
```

```
In [25]: review_df['review_body'] = review_df['review_body'].apply(remove_mention_tag_fr
         display(review_df)
```

	star_rating	review_body	class
0	1	started out okay but after using for a little ...	1
1	2	i'm on the constant search for the perfect con...	1
2	1	i just bought this at a home show yesterday, a...	1
3	1	ordered 2 one came smooshed and broken... if i...	1
4	1	it may work for some people, but it did not wo...	1
...	...	...	...
59995	5	very happy with this product. looking forward ...	3
59996	5	keeps me dry. just what i wanted. there is lit...	3
59997	5	i bought these for a trip, i needed a bra that...	3
59998	5	love it!! does the job well and very little or...	3
59999	5.0	good stuff	3

60000 rows × 3 columns

```
In [26]: def remove_punctuations(text):
             return ''.join(char for char in text if char not in string.punctuation)
```

```
In [27]: # Remove puctuations
         review_df['review_body'] = review_df['review_body'].apply(remove_punctuations)
```

```
In [28]: def remove_alphanum(text):
             t= " ".join([re.sub('[^A-Za-z]+','', text) for text in nltk.word_tokenize(t
             return t
```

```
In [29]: # Remove non-alpabetics
         review_df['review_body']=review_df['review_body'].apply(remove_alphanum)
```

```
In [30]: # removing extra space
         review_df['review_body'] = review_df['review_body'].apply(lambda x: re.sub(' +'
```

```
In [31]: def word_contractions(text):
             t=[]
             for i in text.split():
                 t.append(contractions.fix(i))
             # Now that the review has been split into a list of words and contracted, t
             return ' '.join(t)
```

```
In [32]: # Contracting the reviews
         review_df['review_body']=review_df['review_body'].apply(word_contractions)
```

```
In [33]: post_dc = review_df['review_body'].str.len().mean()
```

```
In [34]: print("Average length of review body before and after Data Cleaning",pre_dc,pos
```

```
Average length of review body before and after Data Cleaning 271.1835333333333
4 259.4927833333333
```

In [35]:
```
clean_data=review_df.copy()
```

Word2Vec

In [36]:
```
#Splitting the dataset into testing and training dataset
Xtrain, Xtest, ytrain, ytest = train_test_split(wv_data['review_body'], wv_data

print("Training Shape ", Xtrain.shape)
print("Testing Shape ", Xtest.shape)
```

```
Training Shape  (48000,)
Testing Shape  (12000,)
```

In [37]:
```
type(ytrain)
```

Out[37]:
```
pandas.core.series.Series
```

(a) Load the pretrained “word2vec-google-news-300” Word2Vec model and learn how to extract word embeddings for your dataset. Try to check semantic similarities of the generated vectors using three examples of your own, e.g., King – Man + Woman = Queen or excellent ~ outstanding.

In [38]:
```
import gensim.downloader as a_c
```

In [39]:
```
wv_model = a_c.load('word2vec-google-news-300')
```

In [40]:
```
wv_model.save('Gensim_model.kv')
```

In [41]:
```
print(wv_model.most_similar(positive=['Woman','King'], negative=['Man']))
print(wv_model.most_similar('Excellent'))
print(wv_model.most_similar(positive=['she','father'], negative=['him']))
print(wv_model.most_similar(positive=['Google','Gmail'], negative=['Outlook']))
print(wv_model.most_similar('happy'))
```

```
[('Queen', 0.4929387867450714), ('Tupou_V.', 0.45174285769462585), ('Oprah_BFF
_Gayle', 0.4422132968902588), ('Jackson', 0.4402504861354828), ('NECN_Alison',
0.4331282675266266), ('Whitfield', 0.42834725975990295), ('Ida_Vandross', 0.42
084529995918274), ('prosecutor_Dan_Satterberg', 0.420758992433548), ('martin_L
uther_King', 0.42059651017189026), ('Coretta_King', 0.42027339339256287)]
[('excellent', 0.6091997027397156), ('definition_redistributional', 0.57536011
9342804), ('Exceptional', 0.5664600729942322), ('flexible_hou_MORE', 0.5228071
212768555), ('EXCELLENT', 0.521685779094696), ('Decent', 0.5081128478050232),
('Superb', 0.502091646194458), ('Terrific', 0.4998748004436493), ('Satisfactor
y', 0.4908524453639984), ('+_Bens', 0.48303356766700745)]
[('mother', 0.7119966745376587), ('husband', 0.6427904963493347), ('daughter',
0.6421711444854736), ('sister', 0.6059560179710388), ('eldest_daughter', 0.598
8065004348755), ('grandmother', 0.5926641225814819), ('mom', 0.579074561595916
7), ('aunt', 0.5724061131477356), ('niece', 0.5554639101028442), ('granddaught
er', 0.5517464876174927)]
[('Yahoo', 0.6514489054679871), ('Google_Nasdaq_GOOG', 0.6343342661857605),
('Google_GOOG', 0.6178034543991089), ('search_engine', 0.6156800389289856),
('GoogleGoogle', 0.6143473982810974), ('Google_NSDQ_GOOG', 0.611092627048492
4), ('Google_NASDAQ_GOOG', 0.6102906465530396), ('Yahoo_Nasdaq_YHOO', 0.608016
9677734375), ('GMail', 0.6057670712471008), ('Google_nasdaq_GOOG', 0.604456603
5270691)]
[('glad', 0.7408890724182129), ('pleased', 0.6632170677185059), ('ecstatic',
0.6626912951469421), ('overjoyed', 0.6599285006523132), ('thrilled', 0.6514049
172401428), ('satisfied', 0.6437950134277344), ('proud', 0.636042118072509 8),
('delighted', 0.627237856388092), ('disappointed', 0.6269948482513428), ('exci
ted', 0.6247665882110596)]
```

In [42]:
```
print(cosine_similarity([wv_model['queen']], [wv_model['king'] - wv_model['woma
print(cosine_similarity([wv_model['queen']], [wv_model['king'] - wv_model['man'
print(cosine_similarity([wv_model['he']], [wv_model['she']]))
print(cosine_similarity([wv_model['excellent']],[wv_model['outstanding']]))
print(cosine_similarity([wv_model['him']], [wv_model['father'] - wv_model['she'
print(cosine_similarity([wv_model['Google']], [wv_model['Gmail']]))
```

```
[[0.2858241]]
[[0.7300518]]
[[0.6129949]]
[[0.5567487]]
[[0.0734347]]
[[0.68005306]]
```

(b) Train a Word2Vec model using your own dataset. You will use these ex- tracted features in the subsequent questions of this assignment. Set the em- bedding size to be 300 and the window size to be 13. You can also consider a minimum word count of 9. Check the semantic similarities for the same two examples in part (a). What do you conclude from comparing vectors generated by yourself and the pretrained model? Which of the Word2Vec models seems to encode semantic similarities between words better?

For the rest of this assignment, use the pretrained "word2vec-google- news-300" Word2Ve features.

In [43]:
```
Xtraining_wv = list(wv_data['review_body'].str.split(" "))
```

In [44]:
```python
type(Xtraining_wv[1])
```

Out[44]:
```
list
```

In [45]:
```python
embedding_size=300
window_size=13
minimum_count=9
```

In [46]:
```python
my_model = gensim.models.Word2Vec(Xtraining_wv, vector_size=embedding_size, wir
```

In [47]:
```python
print(cosine_similarity([my_model.wv['excellent']],[my_model.wv['outstanding']]
print("*******************")
print(cosine_similarity([wv_model['excellent']],[wv_model['outstanding']]))
```

```
[[0.5882459]]
********************
[[0.5567487]]
```

In [48]:
```python
print(cosine_similarity([my_model.wv['he']], [my_model.wv['she']]))
print(cosine_similarity([my_model.wv['excellent']],[my_model.wv['outstanding']]
print(cosine_similarity([my_model.wv['him']], [my_model.wv['father'] - my_model
```

```
[[0.7792789]]
[[0.5882459]]
[[-0.62386674]]
```

In [49]:
```python
k=0
for i in range(len(Xtrain)):
    k=k+1
    if(k<5):
        print(k,wv_data['review_body'][i])
```

```
1 As seen on TV. Well it looks easy on TV.  I have no been able to enjoy this
product because it doesn't stay in place and pulls my hair out.  ouch.
2 Quality was terrible. Even it was broken when I received
3 Bad deal.
4 Purchased this item because I had a tube that I picked up at the mall, and a
ssumed it would be the same product.  But I guess you get what you paid for, a
nd while I thought I was getting was a good deal (of course I purchased 2 of t
hem), i end up with something that has either been refilled with some sort of
non skin tone colored goop which spreads like candle wax or honey on a greesy
face (Lol).  Unless you have skin the color of one of those Barbie dolls, you
may as well use this stuff for Halloween.
```

In [50]:
```python
kr=0
for i, value in enumerate(Xtrain):
    kr=kr+1
    if(kr<5):
        print(kr,value)
```

```
1 Good color; an all right product. Nothing special. Only got one in the pack
sadly; could have called the people, but was too lazy. They lied, though, so t
hat is very disappointing and people should lie. :)
2 I have been using these products for 8 weeks now, and i have noticed a big d
ifference in hairloss. Its a medicated shampoo, but I have grown to enjoy the
little tingle on my scalp when the conditioner is left on for 5 mins. At firs
t, it did not lather well, but I realized that is due to the buildup from the
other shampoo.  After a few washings, it lathers really well.  I only need a l
ittle bit to get a big lather.  I highly recommend the leave in spray conditio
ner because the rinse out one does not really leave the hair smooth and easy t
o comb through.  I have also added the mouse and the hair spray and so far, al
l of the products work really well together. I am pleased with the products an
d will try next get my sons to give it a whirl.
3 figure i'd give these a try, they werent for me. I'm personally a Bic and Gi
llette guy right now, I'm using an Edwin Jagger DE89. these are somewhat aggre
ssive for me with my DE89. I like that nice smooth glide across my face, these
almost felt like they were pulling hair.<br /><br />Give em a go though, who k
nows, you might like em! they jsut weren't for me!
4 it works okay.. really like a different brand. but this will do in a pinch!
```

In [51]:

```python
#Converting the shape of the data
def embedding_creation(data):
    word_embedding = []
    for i, rev in enumerate(data):
        word_vector = np.zeros(300)
        word_list = rev.split(" ")

        for word in word_list:
            if word in wv_model:
                word_vector += wv_model[word]

        word_vector = word_vector/len(word_list)

        word_embedding.append(word_vector)

    word_embedding_data = np.array(word_embedding)

    return word_embedding_data
```

In [52]:

```python
Xtrain_wv = embedding_creation(Xtrain)
Xtest_wv = embedding_creation(Xtest)
ytrain_wv = ytrain.copy()
ytest_wv = ytest.copy()
```

In [53]:

```python
Xtrain_wv.shape, Xtest_wv.shape, ytrain_wv.shape, ytest_wv.shape
```

Out[53]:

```
((48000, 300), (12000, 300), (48000,), (12000,))
```

TFIDF

In [54]:

```python
#Splitting the Data into train and test data (split should be of 80%-20%)
Xtrain_tf, Xtest_tf, ytrain_tf, ytest_tf = train_test_split(clean_data['review_

print("Training Data Size: ", Xtrain_tf.shape)
print("Testing Data Size: ", Xtest_tf.shape)
```

```
Training Data Size:  (48000,)
Testing Data Size:  (12000,)
```

In [55]:
```
tfID_feat_extract = TfidfVectorizer(
    sublinear_tf=True,
    strip_accents='unicode',
    analyzer='word',
    token_pattern=r'\w{1,}',
    stop_words='english',
    ngram_range=(1, 2),
    max_features=12000
)
```

In [56]:
```
Xtrain_tfid = tfID_feat_extract.fit_transform(Xtrain_tf)

Xtest_tfid = tfID_feat_extract.transform(Xtest_tf)

print("Training document-term matrix : ", Xtrain_tfid)
print("Training feature names for transformation : ", tfID_feat_extract.get_fea
```

```
Training document-term matrix :    (0, 8290)    0.14526307650114165
  (0, 3979)     0.11416736468769348
  (0, 2188)     0.147667889854224
  (0, 5700)     0.1504590096604411
  (0, 10139)    0.1504590096604411
  (0, 8592)     0.1512327912767144
  (0, 7729)     0.08373093510150022
  (0, 7434)     0.10182158419173463
  (0, 7284)     0.15910059582156408
  (0, 2178)     0.15910059582156408
  (0, 9279)     0.09135482318277315
  (0, 4047)     0.06632692969911971
  (0, 3271)     0.0932849827430395
  (0, 232)      0.15677618547577454
  (0, 596)      0.12321695124809254
  (0, 5478)     0.1458338561650719
  (0, 4099)     0.14900756763877743
  (0, 504)      0.1267100702178062
  (0, 3493)     0.07673306222365103
  (0, 156)      0.1186499187420472
  (0, 3104)     0.16803536129853702
  (0, 355)      0.14832484405123067
  (0, 8281)     0.14913224561822783
  (0, 7471)     0.11877207517410025
  (0, 4281)     0.1512327912767144
  :     :
  (47999, 2149) 0.05854195770092628
  (47999, 1239) 0.05828198050025517
  (47999, 9374) 0.042477195781095455
  (47999, 5191) 0.06371667294898073
  (47999, 11251)        0.04311598186639156
  (47999, 11478)        0.06507853410881385
  (47999, 10714)        0.06561683561930762
  (47999, 4694) 0.10257316998858444
  (47999, 3918) 0.0574687430646064
  (47999, 4633) 0.06750992515380748
  (47999, 11610)        0.10133398697215378
  (47999, 3194) 0.13975006503368811
  (47999, 7762) 0.05311697766488094
  (47999, 11166)        0.0412784266950156
  (47999, 6266) 0.056978153946672515
  (47999, 5376) 0.054880961990046946
  (47999, 8713) 0.058729607077203906
  (47999, 9622) 0.054162324016502984
  (47999, 4123) 0.0390301763419807
  (47999, 4839) 0.06904724745748447
  (47999, 4283) 0.09966224880905952
  (47999, 4047) 0.050406835443322175
  (47999, 457)  0.055805145376810465
  (47999, 6016) 0.0554250128527763
  (47999, 5631) 0.05795477772657894
Training feature names for transformation :  ['aa' 'aa batteries' 'aa battery'
... 'zippers' 'zits' 'zone']
```

Simple models

Using the Google pre-trained Word2Vec features, train a single perceptron and an SVM model for the classification problem. For this purpose, use the average Word2Vec vectors for each review as the input feature (x = N1 PNi=1 Wi for a review with N words). Report your accuracy values on the testing split for these models similar to HW1, i.e., for each of perceptron and SVM models, report two accuracy values Word2Vec and TF-IDF features. What do you conclude from comparing performances for the models trained using the two different feature types (TF-IDF and your trained Word2Vec features)?

Perceptron

TF-IDF

In [57]:
```
%%time
model_perceptron_tfid = Perceptron(
    alpha=0.00001,
    penalty= 'l2',       #Penalty for wrong prediction
    max_iter=1500,       #Maximum number of iterations
    shuffle=True,
    random_state=16,
    tol=0.001,
)
model_perceptron_tfid=model_perceptron_tfid.fit(Xtrain_tfid , ytrain_tf)
pred_percept2_tfid=model_perceptron_tfid.predict(Xtest_tfid)
result2_tfid=classification_report(ytest_tf, pred_percept2_tfid,output_dict=Tru
print(result2_tfid)

acc2_tfid=accuracy_score(ytest_tf, pred_percept2_tfid)
```

```
{'1': {'precision': 0.6404552509053285, 'recall': 0.6091020910209102, 'f1-scor
e': 0.6243853234144496, 'support': 4065}, '2': {'precision': 0.536700671808907
7, 'recall': 0.5277709811597749, 'f1-score': 0.5321983715766099, 'support': 40
87}, '3': {'precision': 0.6704738760631834, 'recall': 0.716995841995842, 'f1-s
core': 0.6929549164887605, 'support': 3848}, 'accuracy': 0.616, 'macro avg':
{'precision': 0.6158765995924732, 'recall': 0.617956304725509, 'f1-score': 0.6
165128704932733, 'support': 12000}, 'weighted avg': {'precision': 0.6147441429
753581, 'recall': 0.616, 'f1-score': 0.6149759669135076, 'support': 12000}}
CPU times: user 293 ms, sys: 293 ms, total: 585 ms
Wall time: 224 ms
```

In [58]:
```
i=1
for keys,values in result2_tfid.items():
    if i==4:
        i=i+1
        continue
    else:
        print(keys,": ",values['precision'],",",values['recall'],",",values['f1
        i=i+1
```

```
1 :  0.6404552509053285 , 0.6091020910209102 , 0.6243853234144496
2 :  0.5367006718089077 , 0.5277709811597749 , 0.5321983715766099
3 :  0.6704738760631834 , 0.716995841995842 , 0.6929549164887605
macro avg :  0.6158765995924732 , 0.617956304725509 , 0.6165128704932733
weighted avg :  0.6147441429753581 , 0.616 , 0.6149759669135076
```

Word2Vec

In [59]:
```
%%time
model_perceptron_wv = Perceptron(
    alpha=0.00001,
    penalty= 'l2',        #Penalty for wrong prediction
    max_iter=1500,        #Maximum number of iterations
    shuffle=True,
    random_state=16,
    tol=0.001,
)
model_perceptron_wv=model_perceptron_wv.fit(Xtrain_wv , ytrain_wv)
pred_percept2_wv=model_perceptron_wv.predict(Xtest_wv)
result2_wv=classification_report(ytest_wv, pred_percept2_wv,output_dict=True)
print(result2_wv)

acc2_wv=accuracy_score(ytest_wv, pred_percept2_wv)
```

```
{'1': {'precision': 0.37064079162263425, 'recall': 0.9762145748987854, 'f1-sco
re': 0.5372884896594945, 'support': 3952}, '2': {'precision': 0.52909090909090
91, 'recall': 0.07289579158316634, 'f1-score': 0.12813738441215325, 'support':
3992}, '3': {'precision': 0.8856868395773295, 'recall': 0.22731755424063116,
'f1-score': 0.36178144006278207, 'support': 4056}, 'accuracy': 0.4225833333333
333, 'macro avg': {'precision': 0.5951395134302909, 'recall': 0.42547597357419
426, 'f1-score': 0.3424024380448099, 'support': 12000}, 'weighted avg': {'prec
ision': 0.5974374282424341, 'recall': 0.4225833333333333, 'f1-score': 0.341856
1725501902, 'support': 12000}}
CPU times: user 909 ms, sys: 305 ms, total: 1.21 s
Wall time: 801 ms
```

In [60]:
```
i=1
for keys,values in result2_wv.items():
    if i==4:
        i=i+1
        continue
    else:
        print(keys,": ",values['precision'],",",values['recall'],",",values['f1
        i=i+1
```

```
1 :  0.37064079162263425 , 0.9762145748987854 , 0.5372884896594945
2 :  0.5290909090909091 , 0.07289579158316634 , 0.12813738441215325
3 :  0.8856868395773295 , 0.22731755424063116 , 0.36178144006278207
macro avg :  0.5951395134302909 , 0.42547597357419426 , 0.3424024380448099
weighted avg :  0.5974374282424341 , 0.4225833333333333 , 0.3418561725501902
```

In [61]:
```
print("Perceptron:TF-IDF",acc2_tfid)
print("Perceptron:W2V",acc2_wv)
```

```
Perceptron:TF-IDF 0.616
Perceptron:W2V 0.4225833333333333
```

SVM

TF-IDF

In [62]:
```
%%time
svm_model_tfid = LinearSVC(
    C=0.35,
    tol=0.001,
    max_iter=1000,                    #Total iterations
    random_state=16,                   #Control the random number generation to co
    penalty='l1',                     #Norm of Penalty
    class_weight="balanced",          #Provides the weight to each class
    loss='squared_hinge',             #Specifies the Loss Function
    dual=False,                       #Selects the algorithm to either the dual or
)
svm_model_tfid=svm_model_tfid.fit(Xtrain_tfid , ytrain_tf)
pred_svm_tfid=svm_model_tfid.predict(Xtest_tfid)
svm_result_tfid=classification_report(ytest_tf, pred_svm_tfid,output_dict=True)
print(svm_result_tfid)

acc3_tfid=accuracy_score(ytest_tf, pred_svm_tfid)
```

```
{'1': {'precision': 0.6853801169590643, 'recall': 0.7207872078720787, 'f1-scor
e': 0.7026378896882494, 'support': 4065}, '2': {'precision': 0.622762863534675
6, 'recall': 0.544898458527037, 'f1-score': 0.5812345034581756, 'support': 408
7}, '3': {'precision': 0.7288503253796096, 'recall': 0.7858627858627859, 'f1-s
core': 0.7562836063523822, 'support': 3848}, 'accuracy': 0.68175, 'macro avg':
{'precision': 0.6789977686244498, 'recall': 0.6838494840873005, 'f1-score': 0.
6800519998329357, 'support': 12000}, 'weighted avg': {'precision': 0.677993170
8971294, 'recall': 0.68175, 'f1-score': 0.6784923128716887, 'support': 12000}}
CPU times: user 1.63 s, sys: 177 ms, total: 1.8 s
Wall time: 1.28 s
```

In [63]:
```
i=1
for keys,values in svm_result_tfid.items():
    if i==4:
        i=i+1
        continue
    else:
        print(keys,": ",values['precision'],",",values['recall'],",",values['f1
        i=i+1
```

```
1 :  0.6853801169590643 , 0.7207872078720787 , 0.7026378896882494
2 :  0.6227628635346756 , 0.544898458527037 , 0.5812345034581756
3 :  0.7288503253796096 , 0.7858627858627859 , 0.7562836063523822
macro avg :  0.6789977686244498 , 0.6838494840873005 , 0.6800519998329357
weighted avg :  0.6779931708971294 , 0.68175 , 0.6784923128716887
```

Word2Vec

In [64]:
```
%%time
svm_model_wv = LinearSVC(
    C=0.35,
    tol=0.001,
    max_iter=1000,                    #Total iterations
    random_state=16,                   #Control the random number generation to co
    penalty='l1',                     #Norm of Penalty
    class_weight="balanced",          #Provides the weight to each class
    loss='squared_hinge',             #Specifies the Loss Function
```

```
    dual=False,                   #Selects the algorithm to either the dual or
)
```

```
CPU times: user 9 µs, sys: 1e+03 ns, total: 10 µs
Wall time: 11.9 µs
```

In [65]:
```
%%time
svm_model_wv=svm_model_wv.fit(Xtrain_wv , ytrain_wv)
```

```
CPU times: user 40.3 s, sys: 574 ms, total: 40.9 s
Wall time: 41 s
```

In [66]:
```
%%time
pred_svm_wv=svm_model_wv.predict(Xtest_wv)
```

```
CPU times: user 9.09 ms, sys: 2.25 ms, total: 11.3 ms
Wall time: 4.01 ms
```

In [67]:
```
%%time
svm_result_wv=classification_report(ytest_wv, pred_svm_wv,output_dict=True)
print(svm_result_wv)
acc3_wv=accuracy_score(ytest_wv, pred_svm_wv)
```

```
{'1': {'precision': 0.6207478890229192, 'recall': 0.6510627530364372, 'f1-scor
e': 0.6355440286525874, 'support': 3952}, '2': {'precision': 0.566740209597352
5, 'recall': 0.5147795591182365, 'f1-score': 0.5395116828563927, 'support': 39
92}, '3': {'precision': 0.690470560416174, 'recall': 0.7199211045364892, 'f1-s
core': 0.7048883524441762, 'support': 4056}, 'accuracy': 0.629, 'macro avg':
{'precision': 0.6259862196788152, 'recall': 0.628587805563721, 'f1-score': 0.6
266480213177188, 'support': 12000}, 'weighted avg': {'precision': 0.6263475972
649342, 'recall': 0.629, 'f1-score': 0.6270356497259437, 'support': 12000}}
CPU times: user 63.1 ms, sys: 19 ms, total: 82.1 ms
Wall time: 16.4 ms
```

In [68]:
```
i=1
for keys,values in svm_result_wv.items():
    if i==4:
        i=i+1
        continue
    else:
        print(keys,": ",values['precision'],",",values['recall'],",",values['f1
        i=i+1
```

```
1 :  0.6207478890229192 , 0.6510627530364372 , 0.6355440286525874
2 :  0.5667402095973525 , 0.5147795591182365 , 0.5395116828563927
3 :  0.690470560416174 , 0.7199211045364892 , 0.7048883524441762
macro avg :  0.6259862196788152 , 0.628587805563721 , 0.6266480213177188
weighted avg :  0.6263475972649342 , 0.629 , 0.6270356497259437
```

In [69]:
```
print("SVM:TF-IDF",acc3_tfid)
print("SVM:W2V",acc3_wv)
```

```
SVM:TF-IDF 0.68175
SVM:W2V 0.629
```

Feedforward Neural Networks

Using the Word2Vec features, train a feedforward multilayer perceptron net- work for classification. Consider a network with

two hidden layers, each with 100 and 10 nodes, respectively. You can use cross entropy loss and your own choice for other hyperparamters, e.g., nonlinearity, number of epochs, etc. Part of getting good results is to select suitable values for these hyper-paramters.

You can also refer to the following tutorial to familiarize yourself: https://www.kaggle.com/mishra1993/pytorch-multi-layer-perceptron-mnist Although the above tutorial is for image data but the concept of training an MLP is very similar to what we want to do.

(a) To generate the input features, use the average Word2Vec vectors similar to the “Simple models” section and train the neural network. Report accuracy values on the testing split for your MLP.

In [70]:
```
!pip install torchvision
```

```
Requirement already satisfied: torchvision in /Users/asmitachotani/opt/minicon
da3/lib/python3.9/site-packages (0.14.1)
Requirement already satisfied: requests in /Users/asmitachotani/opt/miniconda
3/lib/python3.9/site-packages (from torchvision) (2.27.1)
Requirement already satisfied: typing-extensions in /Users/asmitachotani/opt/m
iniconda3/lib/python3.9/site-packages (from torchvision) (4.5.0)
Requirement already satisfied: torch in /Users/asmitachotani/opt/miniconda3/li
b/python3.9/site-packages (from torchvision) (1.13.1)
Requirement already satisfied: numpy in /Users/asmitachotani/opt/miniconda3/li
b/python3.9/site-packages (from torchvision) (1.23.5)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /Users/asmitachotani/o
pt/miniconda3/lib/python3.9/site-packages (from torchvision) (9.4.0)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /Users/asmitachotani/o
pt/miniconda3/lib/python3.9/site-packages (from requests->torchvision) (1.26.
9)
Requirement already satisfied: certifi>=2017.4.17 in /Users/asmitachotani/opt/
miniconda3/lib/python3.9/site-packages (from requests->torchvision) (2022.5.1
8.1)
Requirement already satisfied: charset-normalizer~=2.0.0 in /Users/asmitachota
ni/opt/miniconda3/lib/python3.9/site-packages (from requests->torchvision) (2.
0.4)
Requirement already satisfied: idna<4,>=2.5 in /Users/asmitachotani/opt/minico
nda3/lib/python3.9/site-packages (from requests->torchvision) (3.3)
```

In [71]:
```
!pip install torch torchvision torchaudio
```

```
Requirement already satisfied: torch in /Users/asmitachotani/opt/miniconda3/li
b/python3.9/site-packages (1.13.1)
Requirement already satisfied: torchvision in /Users/asmitachotani/opt/minicon
da3/lib/python3.9/site-packages (0.14.1)
Requirement already satisfied: torchaudio in /Users/asmitachotani/opt/minicond
a3/lib/python3.9/site-packages (0.13.1)
Requirement already satisfied: typing-extensions in /Users/asmitachotani/opt/m
iniconda3/lib/python3.9/site-packages (from torch) (4.5.0)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /Users/asmitachotani/o
pt/miniconda3/lib/python3.9/site-packages (from torchvision) (9.4.0)
Requirement already satisfied: requests in /Users/asmitachotani/opt/miniconda
3/lib/python3.9/site-packages (from torchvision) (2.27.1)
Requirement already satisfied: numpy in /Users/asmitachotani/opt/miniconda3/li
b/python3.9/site-packages (from torchvision) (1.23.5)
Requirement already satisfied: idna<4,>=2.5 in /Users/asmitachotani/opt/minico
nda3/lib/python3.9/site-packages (from requests->torchvision) (3.3)
Requirement already satisfied: certifi>=2017.4.17 in /Users/asmitachotani/opt/
miniconda3/lib/python3.9/site-packages (from requests->torchvision) (2022.5.1
8.1)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /Users/asmitachotani/o
pt/miniconda3/lib/python3.9/site-packages (from requests->torchvision) (1.26.
9)
Requirement already satisfied: charset-normalizer~=2.0.0 in /Users/asmitachota
ni/opt/miniconda3/lib/python3.9/site-packages (from requests->torchvision) (2.
0.4)
```

In [106…

```
def accuracy23(y_pred, y_true):
    y_pred = y_pred.detach().numpy()
    y_true = y_true.detach().numpy()
    return round(np.sum(y_pred==y_true)/len(y_true), 4)*100
```

In [107…

```
def train_model(x_train_tensor,y_train_tensor,x_test_tensor,y_test_tensor,model
    train_losses = []
    test_losses = []

    valid_loss_min2 = np.Inf

    for epoch in range(epochs):

        # clear the gradients of all optimized variables
        optimizer2.zero_grad()

        # forward pass: compute predicted outputs by passing inputs to the mode
        output2 = model.forward(x_train_tensor)

        # calculate the loss
        loss2 = criterion2(output2, y_train_tensor)

        # backward pass: compute gradient of the loss with respect to model pa
        loss2.backward()

        # update running training loss
        train_loss = loss2.item()
        train_losses.append(train_loss)

        # perform a single optimization step (parameter update)
        optimizer2.step()

        # Turn off gradients for validation, saves memory and computations
```

```
        with torch.no_grad():
            model.eval()
            # forward pass: compute predicted outputs by passing inputs to the
            log_ps = model(x_test_tensor)

            # calculate the validation loss
            test_loss = criterion2(log_ps, y_test_tensor)
            test_losses.append(test_loss)


        model.train()

        print(f"Epoch: {epoch+1}/{epochs} ",
              f"Training Loss: {train_loss:.3f}.. ",
              f"Test Loss: {test_loss:.3f}.. ")

        if test_loss < valid_loss_min2:
                if not fnn_concat:
                    torch.save(model.state_dict(), 'fnn_comb_sp.pt')
                else:
                     torch.save(model.state_dict(), 'fnn_comb_concat.pt')
                valid_loss_min2 = test_loss
```

In [108…

```
%%time
class FNN(nn.Module):
    def __init__(self, input_dim,output_dim):
        super(FNN, self).__init__()
        self.layer1 = nn.Linear(input_dim, 100)
        self.act_func_relu1 = nn.ReLU()
        self.layer2 = nn.Linear(100, 10)
        self.act_func_relu2 = nn.ReLU()
        self.layer3 = nn.Linear(10, output_dim)


    def forward(self, x):
        # add hidden layer, with relu activation function
        x = self.act_func_relu1(self.layer1(x))
        # add hidden layer, with relu activation function
        x = self.act_func_relu2(self.layer2(x))
        # add output layer
        x = self.layer3(x)
        return x
```

```
CPU times: user 43 µs, sys: 9 µs, total: 52 µs
Wall time: 57 µs
```

In [109…

```
%%time
fnn = FNN(300,3)
print(fnn)
```

```
FNN(
  (layer1): Linear(in_features=300, out_features=100, bias=True)
  (act_func_relu1): ReLU()
  (layer2): Linear(in_features=100, out_features=10, bias=True)
  (act_func_relu2): ReLU()
  (layer3): Linear(in_features=10, out_features=3, bias=True)
)
CPU times: user 4.18 ms, sys: 3.21 ms, total: 7.39 ms
Wall time: 4.78 ms
```

In [110…
```
%%time
X_train_word2vec = Xtrain_wv.astype(np.float32)
X_test_word2vec = Xtest_wv.astype(np.float32)
```

```
CPU times: user 28 ms, sys: 69 ms, total: 97 ms
Wall time: 98.2 ms
```

In [111…
```
x_train_tensor = torch.tensor(X_train_word2vec)
x_test_tensor = torch.tensor(X_test_word2vec)
```

In [112…
```
ytrain2=ytrain.copy()
ytest2=ytest.copy()
ytrain2-=1
ytest2-=1
```

In [113…
```
y_train_tensor = torch.tensor(ytrain2.values)
y_test_tensor = torch.tensor(ytest2.values)
```

In [114…
```
# Define the loss
criterion2 = nn.CrossEntropyLoss()
optimizer2 = Adam(fnn.parameters(), lr=1e-2)
```

In [115…
```
train_model(x_train_tensor,y_train_tensor,x_test_tensor,y_test_tensor,fnn,100,c
```

```
Epoch: 1/100  Training Loss: 1.124..  Test Loss: 1.115..
Epoch: 2/100  Training Loss: 1.114..  Test Loss: 1.102..
Epoch: 3/100  Training Loss: 1.101..  Test Loss: 1.091..
Epoch: 4/100  Training Loss: 1.091..  Test Loss: 1.084..
Epoch: 5/100  Training Loss: 1.085..  Test Loss: 1.077..
Epoch: 6/100  Training Loss: 1.078..  Test Loss: 1.065..
Epoch: 7/100  Training Loss: 1.065..  Test Loss: 1.052..
Epoch: 8/100  Training Loss: 1.052..  Test Loss: 1.038..
Epoch: 9/100  Training Loss: 1.038..  Test Loss: 1.025..
Epoch: 10/100  Training Loss: 1.027..  Test Loss: 1.012..
Epoch: 11/100  Training Loss: 1.013..  Test Loss: 0.999..
Epoch: 12/100  Training Loss: 1.001..  Test Loss: 0.985..
Epoch: 13/100  Training Loss: 0.988..  Test Loss: 0.973..
Epoch: 14/100  Training Loss: 0.977..  Test Loss: 0.962..
Epoch: 15/100  Training Loss: 0.966..  Test Loss: 0.951..
Epoch: 16/100  Training Loss: 0.956..  Test Loss: 0.942..
Epoch: 17/100  Training Loss: 0.947..  Test Loss: 0.934..
Epoch: 18/100  Training Loss: 0.939..  Test Loss: 0.927..
Epoch: 19/100  Training Loss: 0.932..  Test Loss: 0.922..
Epoch: 20/100  Training Loss: 0.925..  Test Loss: 0.915..
Epoch: 21/100  Training Loss: 0.918..  Test Loss: 0.912..
Epoch: 22/100  Training Loss: 0.913..  Test Loss: 0.911..
Epoch: 23/100  Training Loss: 0.911..  Test Loss: 0.908..
Epoch: 24/100  Training Loss: 0.908..  Test Loss: 0.905..
Epoch: 25/100  Training Loss: 0.904..  Test Loss: 0.901..
Epoch: 26/100  Training Loss: 0.899..  Test Loss: 0.898..
Epoch: 27/100  Training Loss: 0.895..  Test Loss: 0.897..
Epoch: 28/100  Training Loss: 0.893..  Test Loss: 0.895..
Epoch: 29/100  Training Loss: 0.890..  Test Loss: 0.892..
Epoch: 30/100  Training Loss: 0.886..  Test Loss: 0.887..
Epoch: 31/100  Training Loss: 0.881..  Test Loss: 0.885..
Epoch: 32/100  Training Loss: 0.878..  Test Loss: 0.883..
Epoch: 33/100  Training Loss: 0.876..  Test Loss: 0.878..
Epoch: 34/100  Training Loss: 0.871..  Test Loss: 0.875..
Epoch: 35/100  Training Loss: 0.867..  Test Loss: 0.873..
Epoch: 36/100  Training Loss: 0.865..  Test Loss: 0.870..
Epoch: 37/100  Training Loss: 0.861..  Test Loss: 0.866..
Epoch: 38/100  Training Loss: 0.857..  Test Loss: 0.865..
Epoch: 39/100  Training Loss: 0.855..  Test Loss: 0.863..
Epoch: 40/100  Training Loss: 0.854..  Test Loss: 0.860..
Epoch: 41/100  Training Loss: 0.850..  Test Loss: 0.857..
Epoch: 42/100  Training Loss: 0.847..  Test Loss: 0.857..
Epoch: 43/100  Training Loss: 0.846..  Test Loss: 0.855..
Epoch: 44/100  Training Loss: 0.844..  Test Loss: 0.851..
Epoch: 45/100  Training Loss: 0.840..  Test Loss: 0.850..
Epoch: 46/100  Training Loss: 0.839..  Test Loss: 0.849..
Epoch: 47/100  Training Loss: 0.837..  Test Loss: 0.846..
Epoch: 48/100  Training Loss: 0.834..  Test Loss: 0.844..
Epoch: 49/100  Training Loss: 0.832..  Test Loss: 0.844..
Epoch: 50/100  Training Loss: 0.831..  Test Loss: 0.842..
Epoch: 51/100  Training Loss: 0.829..  Test Loss: 0.840..
Epoch: 52/100  Training Loss: 0.826..  Test Loss: 0.838..
Epoch: 53/100  Training Loss: 0.824..  Test Loss: 0.837..
Epoch: 54/100  Training Loss: 0.822..  Test Loss: 0.837..
Epoch: 55/100  Training Loss: 0.821..  Test Loss: 0.834..
Epoch: 56/100  Training Loss: 0.819..  Test Loss: 0.833..
Epoch: 57/100  Training Loss: 0.816..  Test Loss: 0.831..
Epoch: 58/100  Training Loss: 0.814..  Test Loss: 0.830..
Epoch: 59/100  Training Loss: 0.812..  Test Loss: 0.830..
Epoch: 60/100  Training Loss: 0.812..  Test Loss: 0.829..
```

```
Epoch: 61/100  Training Loss: 0.811..  Test Loss: 0.829..
Epoch: 62/100  Training Loss: 0.809..  Test Loss: 0.827..
Epoch: 63/100  Training Loss: 0.807..  Test Loss: 0.825..
Epoch: 64/100  Training Loss: 0.805..  Test Loss: 0.824..
Epoch: 65/100  Training Loss: 0.803..  Test Loss: 0.823..
Epoch: 66/100  Training Loss: 0.802..  Test Loss: 0.824..
Epoch: 67/100  Training Loss: 0.802..  Test Loss: 0.824..
Epoch: 68/100  Training Loss: 0.802..  Test Loss: 0.825..
Epoch: 69/100  Training Loss: 0.802..  Test Loss: 0.821..
Epoch: 70/100  Training Loss: 0.799..  Test Loss: 0.818..
Epoch: 71/100  Training Loss: 0.795..  Test Loss: 0.818..
Epoch: 72/100  Training Loss: 0.794..  Test Loss: 0.819..
Epoch: 73/100  Training Loss: 0.795..  Test Loss: 0.820..
Epoch: 74/100  Training Loss: 0.795..  Test Loss: 0.817..
Epoch: 75/100  Training Loss: 0.792..  Test Loss: 0.814..
Epoch: 76/100  Training Loss: 0.789..  Test Loss: 0.814..
Epoch: 77/100  Training Loss: 0.788..  Test Loss: 0.815..
Epoch: 78/100  Training Loss: 0.789..  Test Loss: 0.815..
Epoch: 79/100  Training Loss: 0.788..  Test Loss: 0.812..
Epoch: 80/100  Training Loss: 0.785..  Test Loss: 0.811..
Epoch: 81/100  Training Loss: 0.783..  Test Loss: 0.811..
Epoch: 82/100  Training Loss: 0.783..  Test Loss: 0.811..
Epoch: 83/100  Training Loss: 0.783..  Test Loss: 0.812..
Epoch: 84/100  Training Loss: 0.782..  Test Loss: 0.810..
Epoch: 85/100  Training Loss: 0.781..  Test Loss: 0.809..
Epoch: 86/100  Training Loss: 0.778..  Test Loss: 0.808..
Epoch: 87/100  Training Loss: 0.777..  Test Loss: 0.808..
Epoch: 88/100  Training Loss: 0.776..  Test Loss: 0.809..
Epoch: 89/100  Training Loss: 0.776..  Test Loss: 0.810..
Epoch: 90/100  Training Loss: 0.777..  Test Loss: 0.812..
Epoch: 91/100  Training Loss: 0.778..  Test Loss: 0.811..
Epoch: 92/100  Training Loss: 0.777..  Test Loss: 0.809..
Epoch: 93/100  Training Loss: 0.774..  Test Loss: 0.806..
Epoch: 94/100  Training Loss: 0.770..  Test Loss: 0.806..
Epoch: 95/100  Training Loss: 0.770..  Test Loss: 0.808..
Epoch: 96/100  Training Loss: 0.771..  Test Loss: 0.809..
Epoch: 97/100  Training Loss: 0.773..  Test Loss: 0.810..
Epoch: 98/100  Training Loss: 0.772..  Test Loss: 0.807..
Epoch: 99/100  Training Loss: 0.769..  Test Loss: 0.804..
Epoch: 100/100  Training Loss: 0.765..  Test Loss: 0.805..
```

In [116…
```python
fnn2=FNN(300,3)
```

In [117…
```python
softmax = Softmax(dim=1)
```

In [118…
```python
fnn2.load_state_dict(torch.load('fnn_comb_sp.pt'))
test_keys23 = torch.argmax(softmax(fnn2(torch.from_numpy(X_test_word2vec))), ax
```

In [119…
```python
print('Accuracy', format(accuracy23(test_keys23, y_test_tensor)))
```

```
Accuracy 64.08
```

(b) To generate the input features, concatenate the first 10 Word2Vec vectors for each review as the input feature (x = [W T , ..., W T]) and train the neural network. Report the accuracy value on the testing split for your MLP model. What do you conclude by

comparing accuracy values you obtain with those obtained in the "'Simple Models" section.

In [120…
```
#Converting the shape of the data
def concat_embedding_creation(data):
    word_embedding = []
    for i, rev in enumerate(data):
        word_vector = np.zeros((1,300))
        word_list = rev.split(" ")

        if len(word_list)==0:
            word_embedding.apped(np.zeros(10,300))
            continue

        for word in word_list[:10]:
            if word in wv_model:
                word_vector = np.concatenate([word_vector, np.expand_dims(wv_mc

        word_vector = word_vector[1:]

        if len(word_vector)<10:
            for i in range(10 - len(word_vector)):
                word_vector = np.concatenate([word_vector, np.zeros((1,300))],
        word_embedding.append(word_vector)

    word_embedding_data = np.array(word_embedding)

    return word_embedding_data.reshape(word_embedding_data.shape[0], word_embec
```

In [121…
```
Xtrain_wv_concat = concat_embedding_creation(Xtrain)

Xtest_wv_concat = concat_embedding_creation(Xtest)
```

In [122…
```
Xtrain_wv_concat.shape
```

Out[122]:
```
(48000, 3000)
```

In [123…
```
%%time
X_train_word2vec_concat = Xtrain_wv_concat.astype(np.float32)
X_test_word2vec_concat = Xtest_wv_concat.astype(np.float32)
```

```
CPU times: user 191 ms, sys: 364 ms, total: 555 ms
Wall time: 553 ms
```

In [124…
```
X_test_word2vec_concat.shape
```

Out[124]:
```
(12000, 3000)
```

In [125…
```
X_train_word2vec_concat
```

Out[125]:
```
array([[-0.10888672, -0.07470703, -0.04541016, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.07910156, -0.0050354 ,  0.11181641, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.36328125,  0.07470703,  0.07519531, ...,  0.        ,
         0.        ,  0.        ],
       ...,
       [ 0.07910156, -0.0050354 ,  0.11181641, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.08203125,  0.06445312,  0.12255859, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.13183594, -0.07519531,  0.04150391, ...,  0.        ,
         0.        ,  0.        ]], dtype=float32)
```

In [126…
```
ytrain_cc=ytrain.copy()
ytest_cc=ytest.copy()
ytrain_cc-=1
ytest_cc-=1
```

In [127…
```
y_train_tensor_concat = torch.tensor(ytrain_cc.values)
y_test_tensor_concat = torch.tensor(ytest_cc.values)
```

In [128…
```
x_train_tensor_concat = torch.tensor(X_train_word2vec_concat)
x_test_tensor_concat = torch.tensor(X_test_word2vec_concat)
```

In [129…
```
x_train_tensor_concat.shape,x_test_tensor_concat.shape
```

Out[129]:
```
(torch.Size([48000, 3000]), torch.Size([12000, 3000]))
```

In [130…
```
fnn_concat=FNN(3000,3)
```

In [131…
```
print(fnn_concat)
```

```
FNN(
  (layer1): Linear(in_features=3000, out_features=100, bias=True)
  (act_func_relu1): ReLU()
  (layer2): Linear(in_features=100, out_features=10, bias=True)
  (act_func_relu2): ReLU()
  (layer3): Linear(in_features=10, out_features=3, bias=True)
)
```

In [132…
```
# Define the loss
criterion2_concat = nn.CrossEntropyLoss()

# Optimizers require the parameters to optimize and a learning rate
optimizer2_concat = Adam(fnn_concat.parameters(), lr=0.01)
scheduler = ReduceLROnPlateau(optimizer2_concat, patience=30)
```

In [133…
```
train_model(x_train_tensor_concat,y_train_tensor_concat,x_test_tensor_concat,y_
```

```
Epoch: 1/100  Training Loss: 1.100..  Test Loss: 1.181..
Epoch: 2/100  Training Loss: 1.179..  Test Loss: 1.096..
Epoch: 3/100  Training Loss: 1.096..  Test Loss: 1.098..
Epoch: 4/100  Training Loss: 1.097..  Test Loss: 1.092..
Epoch: 5/100  Training Loss: 1.091..  Test Loss: 1.087..
Epoch: 6/100  Training Loss: 1.085..  Test Loss: 1.080..
Epoch: 7/100  Training Loss: 1.077..  Test Loss: 1.071..
Epoch: 8/100  Training Loss: 1.068..  Test Loss: 1.060..
Epoch: 9/100  Training Loss: 1.058..  Test Loss: 1.049..
Epoch: 10/100  Training Loss: 1.047..  Test Loss: 1.042..
Epoch: 11/100  Training Loss: 1.040..  Test Loss: 1.035..
Epoch: 12/100  Training Loss: 1.032..  Test Loss: 1.023..
Epoch: 13/100  Training Loss: 1.019..  Test Loss: 1.012..
Epoch: 14/100  Training Loss: 1.006..  Test Loss: 1.004..
Epoch: 15/100  Training Loss: 0.996..  Test Loss: 0.997..
Epoch: 16/100  Training Loss: 0.988..  Test Loss: 0.992..
Epoch: 17/100  Training Loss: 0.981..  Test Loss: 0.987..
Epoch: 18/100  Training Loss: 0.974..  Test Loss: 0.983..
Epoch: 19/100  Training Loss: 0.968..  Test Loss: 0.979..
Epoch: 20/100  Training Loss: 0.963..  Test Loss: 0.975..
Epoch: 21/100  Training Loss: 0.957..  Test Loss: 0.971..
Epoch: 22/100  Training Loss: 0.952..  Test Loss: 0.968..
Epoch: 23/100  Training Loss: 0.948..  Test Loss: 0.967..
Epoch: 24/100  Training Loss: 0.944..  Test Loss: 0.966..
Epoch: 25/100  Training Loss: 0.940..  Test Loss: 0.964..
Epoch: 26/100  Training Loss: 0.936..  Test Loss: 0.965..
Epoch: 27/100  Training Loss: 0.933..  Test Loss: 0.967..
Epoch: 28/100  Training Loss: 0.933..  Test Loss: 0.963..
Epoch: 29/100  Training Loss: 0.927..  Test Loss: 0.957..
Epoch: 30/100  Training Loss: 0.919..  Test Loss: 0.957..
Epoch: 31/100  Training Loss: 0.918..  Test Loss: 0.954..
Epoch: 32/100  Training Loss: 0.913..  Test Loss: 0.951..
Epoch: 33/100  Training Loss: 0.907..  Test Loss: 0.952..
Epoch: 34/100  Training Loss: 0.904..  Test Loss: 0.952..
Epoch: 35/100  Training Loss: 0.899..  Test Loss: 0.948..
Epoch: 36/100  Training Loss: 0.894..  Test Loss: 0.946..
Epoch: 37/100  Training Loss: 0.889..  Test Loss: 0.946..
Epoch: 38/100  Training Loss: 0.886..  Test Loss: 0.943..
Epoch: 39/100  Training Loss: 0.881..  Test Loss: 0.942..
Epoch: 40/100  Training Loss: 0.875..  Test Loss: 0.942..
Epoch: 41/100  Training Loss: 0.871..  Test Loss: 0.943..
Epoch: 42/100  Training Loss: 0.867..  Test Loss: 0.946..
Epoch: 43/100  Training Loss: 0.865..  Test Loss: 0.946..
Epoch: 44/100  Training Loss: 0.864..  Test Loss: 0.940..
Epoch: 45/100  Training Loss: 0.853..  Test Loss: 0.937..
Epoch: 46/100  Training Loss: 0.847..  Test Loss: 0.942..
Epoch: 47/100  Training Loss: 0.848..  Test Loss: 0.942..
Epoch: 48/100  Training Loss: 0.840..  Test Loss: 0.939..
Epoch: 49/100  Training Loss: 0.832..  Test Loss: 0.942..
Epoch: 50/100  Training Loss: 0.830..  Test Loss: 0.946..
Epoch: 51/100  Training Loss: 0.827..  Test Loss: 0.942..
Epoch: 52/100  Training Loss: 0.820..  Test Loss: 0.942..
Epoch: 53/100  Training Loss: 0.813..  Test Loss: 0.948..
Epoch: 54/100  Training Loss: 0.810..  Test Loss: 0.950..
Epoch: 55/100  Training Loss: 0.808..  Test Loss: 0.953..
Epoch: 56/100  Training Loss: 0.802..  Test Loss: 0.949..
Epoch: 57/100  Training Loss: 0.794..  Test Loss: 0.950..
Epoch: 58/100  Training Loss: 0.787..  Test Loss: 0.957..
Epoch: 59/100  Training Loss: 0.784..  Test Loss: 0.961..
Epoch: 60/100  Training Loss: 0.783..  Test Loss: 0.973..
```

```
Epoch: 61/100  Training Loss: 0.782..  Test Loss: 0.966..
Epoch: 62/100  Training Loss: 0.779..  Test Loss: 0.962..
Epoch: 63/100  Training Loss: 0.763..  Test Loss: 0.965..
Epoch: 64/100  Training Loss: 0.757..  Test Loss: 0.972..
Epoch: 65/100  Training Loss: 0.761..  Test Loss: 0.986..
Epoch: 66/100  Training Loss: 0.756..  Test Loss: 0.972..
Epoch: 67/100  Training Loss: 0.745..  Test Loss: 0.973..
Epoch: 68/100  Training Loss: 0.735..  Test Loss: 0.986..
Epoch: 69/100  Training Loss: 0.735..  Test Loss: 0.989..
Epoch: 70/100  Training Loss: 0.738..  Test Loss: 0.999..
Epoch: 71/100  Training Loss: 0.727..  Test Loss: 0.987..
Epoch: 72/100  Training Loss: 0.713..  Test Loss: 0.991..
Epoch: 73/100  Training Loss: 0.710..  Test Loss: 1.009..
Epoch: 74/100  Training Loss: 0.710..  Test Loss: 1.008..
Epoch: 75/100  Training Loss: 0.710..  Test Loss: 1.016..
Epoch: 76/100  Training Loss: 0.695..  Test Loss: 1.010..
Epoch: 77/100  Training Loss: 0.685..  Test Loss: 1.015..
Epoch: 78/100  Training Loss: 0.681..  Test Loss: 1.035..
Epoch: 79/100  Training Loss: 0.682..  Test Loss: 1.032..
Epoch: 80/100  Training Loss: 0.682..  Test Loss: 1.050..
Epoch: 81/100  Training Loss: 0.672..  Test Loss: 1.037..
Epoch: 82/100  Training Loss: 0.661..  Test Loss: 1.045..
Epoch: 83/100  Training Loss: 0.650..  Test Loss: 1.058..
Epoch: 84/100  Training Loss: 0.646..  Test Loss: 1.062..
Epoch: 85/100  Training Loss: 0.647..  Test Loss: 1.096..
Epoch: 86/100  Training Loss: 0.648..  Test Loss: 1.083..
Epoch: 87/100  Training Loss: 0.652..  Test Loss: 1.098..
Epoch: 88/100  Training Loss: 0.633..  Test Loss: 1.080..
Epoch: 89/100  Training Loss: 0.617..  Test Loss: 1.093..
Epoch: 90/100  Training Loss: 0.615..  Test Loss: 1.134..
Epoch: 91/100  Training Loss: 0.619..  Test Loss: 1.120..
Epoch: 92/100  Training Loss: 0.620..  Test Loss: 1.142..
Epoch: 93/100  Training Loss: 0.604..  Test Loss: 1.123..
Epoch: 94/100  Training Loss: 0.589..  Test Loss: 1.131..
Epoch: 95/100  Training Loss: 0.587..  Test Loss: 1.172..
Epoch: 96/100  Training Loss: 0.589..  Test Loss: 1.158..
Epoch: 97/100  Training Loss: 0.588..  Test Loss: 1.185..
Epoch: 98/100  Training Loss: 0.575..  Test Loss: 1.170..
Epoch: 99/100  Training Loss: 0.563..  Test Loss: 1.183..
Epoch: 100/100  Training Loss: 0.558..  Test Loss: 1.217..
```

In [134…
```
fnn2_concat=FNN(3000,3)
```

In [135…
```
softmax_concat = Softmax(dim=1)
```

In [136…
```
fnn2_concat.load_state_dict(torch.load('fnn_comb_concat.pt'))
test_keys23_concat = torch.argmax(softmax_concat(fnn2_concat(torch.from_numpy(X
```

In [137…
```
print('Accuracy', format(accuracy23(test_keys23_concat, y_test_tensor_concat)))
```

```
Accuracy 54.94
```

Recurrent Neural Networks

Using the Word2Vec features, train a recurrent neural network (RNN) for classification. You can refer to the following tutorial to

familiarize yourself: https://pytorch.org/tutorials/intermediate/char_rnn_classification tutorial.html

In [138…

```
def rnn_train_model(x_train_tensor,y_train_tensor,x_test_tensor,y_test_tensor,n
    train_losses = []
    test_losses = []

    valid_loss_min2 = np.Inf

    for epoch in range(epochs):

        # clear the gradients of all optimized variables
        optimizer2.zero_grad()

        # forward pass: compute predicted outputs by passing inputs to the mode
        output2 = model.forward(x_train_tensor)

        # calculate the loss
        loss2 = criterion2(output2, y_train_tensor)

        # backward pass: compute gradient of the loss with respect to model pa
        loss2.backward()

        # update running training loss
        train_loss = loss2.item()
        train_losses.append(train_loss)

        # perform a single optimization step (parameter update)
        optimizer2.step()

        # Turn off gradients for validation, saves memory and computations
        with torch.no_grad():
            model.eval()
            # forward pass: compute predicted outputs by passing inputs to the
            log_ps = model(x_test_tensor)

            # calculate the validation loss
            test_loss = criterion2(log_ps, y_test_tensor)
            test_losses.append(test_loss)


        model.train()

        print(f"Epoch: {epoch+1}/{epochs} ",
              f"Training Loss: {train_loss:.3f}.. ",
              f"Test Loss: {test_loss:.3f}.. ")

        if test_loss < valid_loss_min2:
                if not gru and not lstm:
                    torch.save(model.state_dict(), 'rnn_sp.pt')
                elif not lstm:
                    torch.save(model.state_dict(), 'rnn_gru.pt')
                else:
                    torch.save(model.state_dict(), 'rnn_lstm.pt')
                valid_loss_min2 = test_loss
```

(a) Train a simple RNN for sentiment analysis. You can consider an RNN cell with the hidden state size of 20. To feed your data into our RNN, limit the maximum review length to 20 by truncating longer reviews and padding shorter reviews with a null value (0). Report accuracy values on the testing split for your RNN model. What do you conclude by comparing accuracy values you obtain with those obtained with feedforward neural network models.

In [139…

```
#Converting the shape of the data
def rnn_embedding_creation(data,words):
    word_embedding = []

    for i, rev in enumerate(data):
        word_vector = []
        word_list = rev.split(" ")

        if len(word_list)==0:
            word_embedding.append(np.zeros((words,300)))
            continue

        for word in word_list[:words]:
            if word in wv_model:
                temp = np.reshape(wv_model[word], (1, 300))
                word_vector.append(temp)
            else:
                word_vector.append(np.zeros((1,300)))
                continue

        if len(word_vector)<words:
            for i in range(words - len(word_vector)):
                word_vector.append(np.zeros((1,300)))

        word_embedding.append(word_vector)

    word_embedding_data = np.array(word_embedding)

    return word_embedding_data
```

In [140…

```
Xtrain_wv_rnn = rnn_embedding_creation(Xtrain,20)
Xtest_wv_rnn = rnn_embedding_creation(Xtest,20)
```

In [141…

```
Xtrain_wv_rnn.shape,Xtest_wv_rnn.shape
```

Out[141]:

```
((48000, 20, 1, 300), (12000, 20, 1, 300))
```

In [142…

```
Xtrain_wv_rnn=Xtrain_wv_rnn.reshape(Xtrain_wv_rnn.shape[0], Xtrain_wv_rnn.shape
Xtest_wv_rnn=Xtest_wv_rnn.reshape(Xtest_wv_rnn.shape[0], Xtest_wv_rnn.shape[1],
```

In [143…

```
Xtrain_wv_rnn.shape,Xtest_wv_rnn.shape
```

Out[143]:

```
((48000, 20, 300), (12000, 20, 300))
```

In [144…

```
%%time
X_train_word2vec_rnn = Xtrain_wv_rnn.astype(np.float32)
X_test_word2vec_rnn = Xtest_wv_rnn.astype(np.float32)
```

```
CPU times: user 505 ms, sys: 1.25 s, total: 1.75 s
Wall time: 2.45 s
```

In [145…

```
x_train_tensor_rnn = torch.tensor(X_train_word2vec_rnn)
x_test_tensor_rn = torch.tensor(X_test_word2vec_rnn)
```

In [146…

```
ytrain_rnn=ytrain.copy()
ytest_rnn=ytest.copy()
ytrain_rnn-=1
ytest_rnn-=1
```

In [147…

```
y_train_tensor_rnn = torch.tensor(ytrain_rnn.values)
y_test_tensor_rnn = torch.tensor(ytest_rnn.values)
```

In [148…

```
x_train_tensor_rnn.shape,x_test_tensor_rn.shape,y_train_tensor_rnn.shape,y_test
```

Out[148]:

```
(torch.Size([48000, 20, 300]),
 torch.Size([12000, 20, 300]),
 torch.Size([48000]),
 torch.Size([12000]))
```

In [149…

```
class RNN(nn.Module):

    def __init__(self, input_size, output_size, n_layers):
        super(RNN, self).__init__()
        self.rnn = nn.RNN(input_size, 20, n_layers,batch_first=True)
        self.linear = nn.Linear(20, output_size)


    def forward(self, x):
        return self.linear(self.rnn(x)[0][:, -1])
```

In [150…

```
rnn = RNN(300,3, 2)
```

In [151…

```
print(rnn)
```

```
RNN(
  (rnn): RNN(300, 20, num_layers=2, batch_first=True)
  (linear): Linear(in_features=20, out_features=3, bias=True)
)
```

In [152…

```
%%time
# Define the loss
criterion_rnn = nn.CrossEntropyLoss()
optimizer_rnn = Adam(rnn.parameters(), lr=0.01)
```

```
CPU times: user 692 µs, sys: 721 µs, total: 1.41 ms
Wall time: 2.3 ms
```

In [153…

```
rnn_train_model(x_train_tensor_rnn,y_train_tensor_rnn,x_test_tensor_rn,y_test_t
```

```
Epoch: 1/80  Training Loss: 1.102..  Test Loss: 1.106..
Epoch: 2/80  Training Loss: 1.104..  Test Loss: 1.097..
Epoch: 3/80  Training Loss: 1.096..  Test Loss: 1.096..
Epoch: 4/80  Training Loss: 1.095..  Test Loss: 1.095..
Epoch: 5/80  Training Loss: 1.095..  Test Loss: 1.093..
Epoch: 6/80  Training Loss: 1.092..  Test Loss: 1.091..
Epoch: 7/80  Training Loss: 1.089..  Test Loss: 1.088..
Epoch: 8/80  Training Loss: 1.086..  Test Loss: 1.084..
Epoch: 9/80  Training Loss: 1.082..  Test Loss: 1.079..
Epoch: 10/80  Training Loss: 1.077..  Test Loss: 1.073..
Epoch: 11/80  Training Loss: 1.071..  Test Loss: 1.061..
Epoch: 12/80  Training Loss: 1.058..  Test Loss: 1.039..
Epoch: 13/80  Training Loss: 1.035..  Test Loss: 1.037..
Epoch: 14/80  Training Loss: 1.031..  Test Loss: 1.046..
Epoch: 15/80  Training Loss: 1.045..  Test Loss: 1.008..
Epoch: 16/80  Training Loss: 1.006..  Test Loss: 1.025..
Epoch: 17/80  Training Loss: 1.019..  Test Loss: 1.020..
Epoch: 18/80  Training Loss: 1.015..  Test Loss: 1.004..
Epoch: 19/80  Training Loss: 1.000..  Test Loss: 1.006..
Epoch: 20/80  Training Loss: 1.006..  Test Loss: 0.997..
Epoch: 21/80  Training Loss: 0.997..  Test Loss: 0.990..
Epoch: 22/80  Training Loss: 0.986..  Test Loss: 0.996..
Epoch: 23/80  Training Loss: 0.991..  Test Loss: 0.986..
Epoch: 24/80  Training Loss: 0.981..  Test Loss: 0.992..
Epoch: 25/80  Training Loss: 0.988..  Test Loss: 0.982..
Epoch: 26/80  Training Loss: 0.977..  Test Loss: 0.984..
Epoch: 27/80  Training Loss: 0.978..  Test Loss: 0.985..
Epoch: 28/80  Training Loss: 0.978..  Test Loss: 0.976..
Epoch: 29/80  Training Loss: 0.970..  Test Loss: 0.975..
Epoch: 30/80  Training Loss: 0.970..  Test Loss: 0.972..
Epoch: 31/80  Training Loss: 0.966..  Test Loss: 0.975..
Epoch: 32/80  Training Loss: 0.969..  Test Loss: 0.968..
Epoch: 33/80  Training Loss: 0.961..  Test Loss: 0.968..
Epoch: 34/80  Training Loss: 0.962..  Test Loss: 0.964..
Epoch: 35/80  Training Loss: 0.957..  Test Loss: 0.958..
Epoch: 36/80  Training Loss: 0.951..  Test Loss: 0.981..
Epoch: 37/80  Training Loss: 0.977..  Test Loss: 0.987..
Epoch: 38/80  Training Loss: 0.978..  Test Loss: 0.965..
Epoch: 39/80  Training Loss: 0.955..  Test Loss: 0.979..
Epoch: 40/80  Training Loss: 0.975..  Test Loss: 0.960..
Epoch: 41/80  Training Loss: 0.952..  Test Loss: 0.973..
Epoch: 42/80  Training Loss: 0.964..  Test Loss: 0.967..
Epoch: 43/80  Training Loss: 0.959..  Test Loss: 0.962..
Epoch: 44/80  Training Loss: 0.955..  Test Loss: 0.950..
Epoch: 45/80  Training Loss: 0.942..  Test Loss: 0.966..
Epoch: 46/80  Training Loss: 0.954..  Test Loss: 0.963..
Epoch: 47/80  Training Loss: 0.952..  Test Loss: 0.951..
Epoch: 48/80  Training Loss: 0.944..  Test Loss: 0.944..
Epoch: 49/80  Training Loss: 0.939..  Test Loss: 0.959..
Epoch: 50/80  Training Loss: 0.952..  Test Loss: 0.943..
Epoch: 51/80  Training Loss: 0.937..  Test Loss: 0.942..
Epoch: 52/80  Training Loss: 0.936..  Test Loss: 0.946..
Epoch: 53/80  Training Loss: 0.937..  Test Loss: 0.948..
Epoch: 54/80  Training Loss: 0.938..  Test Loss: 0.940..
Epoch: 55/80  Training Loss: 0.930..  Test Loss: 0.941..
Epoch: 56/80  Training Loss: 0.933..  Test Loss: 0.943..
Epoch: 57/80  Training Loss: 0.934..  Test Loss: 0.936..
Epoch: 58/80  Training Loss: 0.926..  Test Loss: 0.941..
Epoch: 59/80  Training Loss: 0.930..  Test Loss: 0.938..
Epoch: 60/80  Training Loss: 0.925..  Test Loss: 0.933..
```

```
Epoch: 61/80  Training Loss: 0.921..  Test Loss: 0.940..
Epoch: 62/80  Training Loss: 0.928..  Test Loss: 0.934..
Epoch: 63/80  Training Loss: 0.921..  Test Loss: 0.932..
Epoch: 64/80  Training Loss: 0.918..  Test Loss: 0.931..
Epoch: 65/80  Training Loss: 0.916..  Test Loss: 0.933..
Epoch: 66/80  Training Loss: 0.918..  Test Loss: 0.935..
Epoch: 67/80  Training Loss: 0.920..  Test Loss: 0.931..
Epoch: 68/80  Training Loss: 0.914..  Test Loss: 0.930..
Epoch: 69/80  Training Loss: 0.913..  Test Loss: 0.927..
Epoch: 70/80  Training Loss: 0.911..  Test Loss: 0.926..
Epoch: 71/80  Training Loss: 0.910..  Test Loss: 0.926..
Epoch: 72/80  Training Loss: 0.908..  Test Loss: 0.927..
Epoch: 73/80  Training Loss: 0.908..  Test Loss: 0.928..
Epoch: 74/80  Training Loss: 0.912..  Test Loss: 0.944..
Epoch: 75/80  Training Loss: 0.929..  Test Loss: 0.925..
Epoch: 76/80  Training Loss: 0.907..  Test Loss: 0.926..
Epoch: 77/80  Training Loss: 0.907..  Test Loss: 0.923..
Epoch: 78/80  Training Loss: 0.904..  Test Loss: 0.925..
Epoch: 79/80  Training Loss: 0.907..  Test Loss: 0.922..
Epoch: 80/80  Training Loss: 0.902..  Test Loss: 0.924..
```

In [154…
```
rnn2= RNN(300,3,2)
```

In [155…
```
softmax_rnn = Softmax(dim=1)
```

In [156…
```
rnn2.load_state_dict(torch.load('rnn_sp.pt'))
test_keys23_rnn = torch.argmax(softmax_rnn(rnn2(torch.from_numpy(X_test_word2ve
```

In [157…
```
print('Accuracy', format(accuracy23(test_keys23_rnn, y_test_tensor_rnn)))
```

```
Accuracy 56.37
```

(b) Repeat part (a) by considering a gated recurrent unit cell.

In [164…
```
class GatedRNN(nn.Module):
    def __init__(self, num_classes, layers, batch_size):
        super(GatedRNN, self).__init__()
        self.gru = nn.GRU(300, 300, layers, batch_first=True)
        self.linear = nn.Linear(300, num_classes)

    def forward(self, x):
        return self.linear(self.gru(x)[0][:, -1])
```

In [165…
```
gru = GatedRNN(3, 1, 60)
print(gru)
```

```
GatedRNN(
  (gru): GRU(300, 300, batch_first=True)
  (linear): Linear(in_features=300, out_features=3, bias=True)
)
```

In [166…
```
%%time
# Define the loss
criterion_gru = nn.CrossEntropyLoss()
optimizer_gru = Adam(gru.parameters(), lr=0.01)
```

```
CPU times: user 411 µs, sys: 56 µs, total: 467 µs
Wall time: 483 µs
```

In [167…

```
%%time
rnn_train_model(x_train_tensor_rnn,y_train_tensor_rnn,x_test_tensor_rn,y_test_t
```

```
Epoch: 1/50  Training Loss: 1.100..  Test Loss: 1.232..
Epoch: 2/50  Training Loss: 1.227..  Test Loss: 1.119..
Epoch: 3/50  Training Loss: 1.118..  Test Loss: 1.163..
Epoch: 4/50  Training Loss: 1.165..  Test Loss: 1.090..
Epoch: 5/50  Training Loss: 1.090..  Test Loss: 1.093..
Epoch: 6/50  Training Loss: 1.092..  Test Loss: 1.097..
Epoch: 7/50  Training Loss: 1.095..  Test Loss: 1.094..
Epoch: 8/50  Training Loss: 1.092..  Test Loss: 1.094..
Epoch: 9/50  Training Loss: 1.092..  Test Loss: 1.094..
Epoch: 10/50  Training Loss: 1.093..  Test Loss: 1.087..
Epoch: 11/50  Training Loss: 1.085..  Test Loss: 1.083..
Epoch: 12/50  Training Loss: 1.080..  Test Loss: 1.076..
Epoch: 13/50  Training Loss: 1.072..  Test Loss: 1.047..
Epoch: 14/50  Training Loss: 1.044..  Test Loss: 1.013..
Epoch: 15/50  Training Loss: 1.011..  Test Loss: 1.376..
Epoch: 16/50  Training Loss: 1.357..  Test Loss: 1.110..
Epoch: 17/50  Training Loss: 1.103..  Test Loss: 1.073..
Epoch: 18/50  Training Loss: 1.071..  Test Loss: 1.094..
Epoch: 19/50  Training Loss: 1.092..  Test Loss: 1.104..
Epoch: 20/50  Training Loss: 1.102..  Test Loss: 1.087..
Epoch: 21/50  Training Loss: 1.085..  Test Loss: 1.070..
Epoch: 22/50  Training Loss: 1.067..  Test Loss: 1.062..
Epoch: 23/50  Training Loss: 1.058..  Test Loss: 1.042..
Epoch: 24/50  Training Loss: 1.037..  Test Loss: 1.031..
Epoch: 25/50  Training Loss: 1.027..  Test Loss: 0.993..
Epoch: 26/50  Training Loss: 0.990..  Test Loss: 0.976..
Epoch: 27/50  Training Loss: 0.971..  Test Loss: 0.951..
Epoch: 28/50  Training Loss: 0.946..  Test Loss: 0.953..
Epoch: 29/50  Training Loss: 0.950..  Test Loss: 0.937..
Epoch: 30/50  Training Loss: 0.931..  Test Loss: 0.942..
Epoch: 31/50  Training Loss: 0.934..  Test Loss: 0.930..
Epoch: 32/50  Training Loss: 0.922..  Test Loss: 0.924..
Epoch: 33/50  Training Loss: 0.917..  Test Loss: 0.919..
Epoch: 34/50  Training Loss: 0.913..  Test Loss: 0.902..
Epoch: 35/50  Training Loss: 0.894..  Test Loss: 0.905..
Epoch: 36/50  Training Loss: 0.895..  Test Loss: 0.897..
Epoch: 37/50  Training Loss: 0.886..  Test Loss: 0.887..
Epoch: 38/50  Training Loss: 0.876..  Test Loss: 0.885..
Epoch: 39/50  Training Loss: 0.874..  Test Loss: 0.875..
Epoch: 40/50  Training Loss: 0.861..  Test Loss: 0.878..
Epoch: 41/50  Training Loss: 0.861..  Test Loss: 0.872..
Epoch: 42/50  Training Loss: 0.854..  Test Loss: 0.868..
Epoch: 43/50  Training Loss: 0.849..  Test Loss: 0.868..
Epoch: 44/50  Training Loss: 0.849..  Test Loss: 0.861..
Epoch: 45/50  Training Loss: 0.842..  Test Loss: 0.859..
Epoch: 46/50  Training Loss: 0.839..  Test Loss: 0.854..
Epoch: 47/50  Training Loss: 0.834..  Test Loss: 0.851..
Epoch: 48/50  Training Loss: 0.829..  Test Loss: 0.850..
Epoch: 49/50  Training Loss: 0.827..  Test Loss: 0.847..
Epoch: 50/50  Training Loss: 0.822..  Test Loss: 0.846..
CPU times: user 2h 19min 25s, sys: 1h 14min 1s, total: 3h 33min 27s
Wall time: 49min 29s
```

In [168…

```
getedRnn_model = GatedRNN(3, 1, 100)
```

In [169…

```
getedRnn_model.load_state_dict(torch.load('rnn_gru.pt'))
test_keys23_gru = torch.argmax(softmax(getedRnn_model(torch.from_numpy(X_test_w
```

In [170…

```
print('Accuracy', format(accuracy23(test_keys23_gru, y_test_tensor_rnn)))
```

```
Accuracy 60.86
```

(c) Repeat part (a) by considering an LSTM unit cell. What do you conclude by comparing accuracy values you obtain by GRU, LSTM, and simple RNN.

In [185…

```
class LSTM(nn.Module):
    def __init__(self, num_classes, layers):
        super(LSTM, self).__init__()
        self.lstm = nn.LSTM(300, 100, layers, batch_first=True)
        self.linear = nn.Linear(100, num_classes)

    def forward(self, x):
        return self.linear(self.lstm(x)[0][:, -1])
```

In [186…

```
lstm = LSTM(3, 1)
print(lstm)
```

```
LSTM(
  (lstm): LSTM(300, 100, batch_first=True)
  (linear): Linear(in_features=100, out_features=3, bias=True)
)
```

In [187…

```
%%time
# Define the loss
criterion_lstm = nn.CrossEntropyLoss()
optimizer_lstm = Adam(lstm.parameters(), lr=0.01)
```

```
CPU times: user 701 µs, sys: 719 µs, total: 1.42 ms
Wall time: 1.99 ms
```

In [188…

```
%%time
rnn_train_model(x_train_tensor_rnn,y_train_tensor_rnn,x_test_tensor_rn,y_test_t
```

```
Epoch: 1/50  Training Loss: 1.102..  Test Loss: 1.103..
Epoch: 2/50  Training Loss: 1.103..  Test Loss: 1.091..
Epoch: 3/50  Training Loss: 1.090..  Test Loss: 1.087..
Epoch: 4/50  Training Loss: 1.086..  Test Loss: 1.076..
Epoch: 5/50  Training Loss: 1.074..  Test Loss: 1.047..
Epoch: 6/50  Training Loss: 1.045..  Test Loss: 1.273..
Epoch: 7/50  Training Loss: 1.257..  Test Loss: 1.022..
Epoch: 8/50  Training Loss: 1.022..  Test Loss: 1.050..
Epoch: 9/50  Training Loss: 1.049..  Test Loss: 1.050..
Epoch: 10/50  Training Loss: 1.049..  Test Loss: 1.061..
Epoch: 11/50  Training Loss: 1.059..  Test Loss: 1.064..
Epoch: 12/50  Training Loss: 1.062..  Test Loss: 1.062..
Epoch: 13/50  Training Loss: 1.060..  Test Loss: 1.050..
Epoch: 14/50  Training Loss: 1.048..  Test Loss: 1.019..
Epoch: 15/50  Training Loss: 1.018..  Test Loss: 1.007..
Epoch: 16/50  Training Loss: 1.008..  Test Loss: 1.004..
Epoch: 17/50  Training Loss: 1.005..  Test Loss: 0.988..
Epoch: 18/50  Training Loss: 0.986..  Test Loss: 1.007..
Epoch: 19/50  Training Loss: 1.001..  Test Loss: 1.003..
Epoch: 20/50  Training Loss: 0.997..  Test Loss: 0.980..
Epoch: 21/50  Training Loss: 0.979..  Test Loss: 0.974..
Epoch: 22/50  Training Loss: 0.976..  Test Loss: 0.967..
Epoch: 23/50  Training Loss: 0.968..  Test Loss: 0.962..
Epoch: 24/50  Training Loss: 0.961..  Test Loss: 0.967..
Epoch: 25/50  Training Loss: 0.964..  Test Loss: 0.963..
Epoch: 26/50  Training Loss: 0.960..  Test Loss: 0.952..
Epoch: 27/50  Training Loss: 0.950..  Test Loss: 0.952..
Epoch: 28/50  Training Loss: 0.950..  Test Loss: 0.951..
Epoch: 29/50  Training Loss: 0.949..  Test Loss: 0.948..
Epoch: 30/50  Training Loss: 0.944..  Test Loss: 0.945..
Epoch: 31/50  Training Loss: 0.939..  Test Loss: 0.938..
Epoch: 32/50  Training Loss: 0.932..  Test Loss: 0.939..
Epoch: 33/50  Training Loss: 0.933..  Test Loss: 0.936..
Epoch: 34/50  Training Loss: 0.929..  Test Loss: 0.934..
Epoch: 35/50  Training Loss: 0.925..  Test Loss: 0.937..
Epoch: 36/50  Training Loss: 0.927..  Test Loss: 0.931..
Epoch: 37/50  Training Loss: 0.921..  Test Loss: 0.929..
Epoch: 38/50  Training Loss: 0.919..  Test Loss: 0.926..
Epoch: 39/50  Training Loss: 0.916..  Test Loss: 0.921..
Epoch: 40/50  Training Loss: 0.911..  Test Loss: 0.919..
Epoch: 41/50  Training Loss: 0.908..  Test Loss: 0.915..
Epoch: 42/50  Training Loss: 0.904..  Test Loss: 0.911..
Epoch: 43/50  Training Loss: 0.899..  Test Loss: 0.908..
Epoch: 44/50  Training Loss: 0.896..  Test Loss: 0.905..
Epoch: 45/50  Training Loss: 0.892..  Test Loss: 0.903..
Epoch: 46/50  Training Loss: 0.890..  Test Loss: 0.896..
Epoch: 47/50  Training Loss: 0.883..  Test Loss: 0.896..
Epoch: 48/50  Training Loss: 0.883..  Test Loss: 0.896..
Epoch: 49/50  Training Loss: 0.882..  Test Loss: 0.888..
Epoch: 50/50  Training Loss: 0.872..  Test Loss: 0.891..
CPU times: user 47min 28s, sys: 29min 9s, total: 1h 16min 37s
Wall time: 14min 14s
```

In [190…

```
lstm2 = LSTM(3, 1)
```

In [191…

```
lstm2.load_state_dict(torch.load('rnn_lstm.pt'))
test_keys23_lstm = torch.argmax(softmax(lstm2(torch.from_numpy(X_test_word2vec_
```

In [192…

```
print('Accuracy', format(accuracy23(test_keys23_lstm, y_test_tensor_rnn)))
```

```
Accuracy 57.940000000000005
```

References

In [183…

```
# https://www.kaggle.com/code/mishra1993/pytorch-multi-layer-perceptron-mnist/n
# https://builtin.com/machine-learning/nlp-word2vec-python
# https://python-bloggers.com/2022/05/building-a-pytorch-binary-classification-
# https://blog.floydhub.com/a-beginners-guide-on-recurrent-neural-networks-with
# https://machinelearningmastery.com/pytorch-tutorial-develop-deep-learning-moc
# https://stackoverflow.com/questions/70804697/why-is-my-pytorch-classificatior
# https://deborahmesquita.com/2017-11-05/how-pytorch-gives-the-big-picture-with
# https://medium.com/swlh/text-classification-using-scikit-learn-pytorch-and-te
# https://www.projectpro.io/recipes/develop-mlp-for-multiclass-classification-p
# https://www.analyticsvidhya.com/blog/2020/01/first-text-classification-in-pyt
# https://galhever.medium.com/sentiment-analysis-with-pytorch-part-5-mlp-model-
# https://bhadreshpsavani.medium.com/tutorial-on-sentimental-analysis-using-pyt
# https://dipikabaad.medium.com/finding-the-hidden-sentiments-using-rnns-in-pyt
#
```

In [184…

```
# https://radimrehurek.com/gensim/auto_examples/tutorials/run_word2vec.html
# https://www.guru99.com/word-embedding-word2vec.html
# https://radimrehurek.com/gensim/models/word2vec.html
# https://machinelearningmastery.com/develop-word-embedding-model-predicting-mc
# https://machinelearningmastery.com/develop-word-embedding-model-predicting-mc
# https://medium.com/data-science-lab-spring-2021/amazon-review-rating-predicti
# http://yaronvazana.com/2018/09/20/average-word-vectors-generate-document-para
# https://medium.com/analytics-vidhya/i-strongly-recommend-to-first-know-how-rn
# https://www.deeplearningwizard.com/deep_learning/practical_pytorch/pytorch_fe
```

In [ ]: