CSCI544 HW3 Report

I preprocessed the dataset by:

1. Converting reviews into lowercase
2. remove the HTML and URLs from the reviews
3. Remove non- alphabetical characters
4. Remove extra spaces
5. Perform contractions

Simple models(Part 3)

1. Accuracy values with features Word2Vec for Perceptron is 0.5750833333333333
   Accuracy values with features TF-IDF for Perceptron is 0.51075
2. Accuracy values with features Word2Vec for SVC is 0.6169166666666667 Accuracy values with features TF-IDF for SVC is 0.5506666666666666

```
Accuracy values with features Word2Vec for Perceptron is  0.5750833333333333
Accuracy values with features TF-IDF for Perceptron is  0.51075
Accuracy values with features Word2Vec for Perceptron is  0.6169166666666667
Accuracy values with features TF-IDF for Perceptron is  0.5506666666666666
```

3. There is a typo at the third and fourth line. ‘Perceptron’ should be ‘SVC’
4. I conclude that the models using Word2Vec features have better accuracy than the ones using Word2Vec.

FeedForward Neural Network(Part 4)

1. Accuracy for 4a FNN is 0.6174166798591614
2. Accuracy for 4b FNN is 0.531416654586792
3. The model from 4a has better accuracy about 61%. The second is the model from the simple model perceptron. 4b has the least accuracy model about 53%.

Recurrent Neural Networks(Part 5)

1. Accuracy for 5a RNN is 0.5809166431427002
2. I conclude that the RNN simple model has worse performance(about 58% accuracy) than the Feedforward neural network.
3. Accuracy for 5b GRU is 0.6243333220481873
4. Accuracy for 5c LSTM is 0.6214166879653931
5. I conclude that GRU and LSTM layers have similar and higher accuracy(62%) than any other models. Simple RNN model has the least performance(58%)

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

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remount=
```

```
pip install contractions
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting contractions
  Downloading contractions-0.1.73-py2.py3-none-any.whl (8.7 kB)
Collecting textsearch>=0.0.21
  Downloading textsearch-0.0.24-py2.py3-none-any.whl (7.6 kB)
Collecting pyahocorasick
  Downloading pyahocorasick-2.0.0-cp38-cp38-manylinux_2_5_x86_64.manylinux1_x86_64.whl (104 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 104.5/104.5 KB 3.1 MB/s eta 0:00:00
Collecting anyascii
  Downloading anyascii-0.3.1-py3-none-any.whl (287 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 287.5/287.5 KB 12.5 MB/s eta 0:00:00
Installing collected packages: pyahocorasick, anyascii, textsearch, contractions
Successfully installed anyascii-0.3.1 contractions-0.1.73 pyahocorasick-2.0.0 textsearch-0.0.24
```

```
import pandas as pd
import numpy as np
import warnings
from sklearn.model_selection import train_test_split
from nltk.corpus import stopwords
from bs4 import BeautifulSoup
import re
import contractions
import gensim.downloader as api
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
import gensim.models
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score
from sklearn.svm import LinearSVC
import tensorflow as tf
warnings.filterwarnings('ignore')
from keras.layers import Dense, Embedding, LSTM, Flatten, Dropout, Conv2D, MaxPooling2D
from keras import Input
from tensorflow.keras.optimizers import Adam
```

```
pip install tensorflow
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: tensorflow in /usr/local/lib/python3.8/dist-packages (2.11.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: flatbuffers>=2.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (23.1.21)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: tensorboard<2.12,>=2.11 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (2.11.2)
Requirement already satisfied: keras<2.12,>=2.11.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (2.11.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (1.22.4)
Requirement already satisfied: packaging in /usr/local/lib/python3.8/dist-packages (from tensorflow) (23.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (2.2.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (4.5.0)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (0.4.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (1.15.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (15.0.6.1)
Requirement already satisfied: tensorflow-estimator<2.12,>=2.11.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (1.14.1)
Requirement already satisfied: setuptools in /usr/local/lib/python3.8/dist-packages (from tensorflow) (57.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.8/dist-packages (from tensorfl
Requirement already satisfied: protobuf<3.20,>=3.9.2 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (3.19.6)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (1.51.1)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (3.1.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.8/dist-packages (from astunparse>=1.6.0->tensorf
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.8/dist-packages (from tensorboard<2.12,>=2.11->tens
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/python3.8/dist-packages (from tensorboard<
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in /usr/local/lib/python3.8/dist-packages (from tensorb
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.8/dist-packages (from tensorboard<2.12,>=2.11->tens
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.8/dist-packages (from tensorboard<2.1
```

```
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.8/dist-packages (from tensorboard<2.12,>=2.11
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.8/dist-packages (from tensorboard<2.12,>=2.11->
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.8/dist-packages (from google-auth<3,>=1.6.3-
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.8/dist-packages (from google-auth<3,>=1.6.3->tensorbo
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.8/dist-packages (from google-auth<3,>=1.6.3->
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.8/dist-packages (from google-auth-oauthlib
Requirement already satisfied: importlib-metadata>=4.4 in /usr/local/lib/python3.8/dist-packages (from markdown>=2.6.8->tens
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/dist-packages (from requests<3,>=2.21.0->tenso
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-packages (from requests<3,>=2.21.0->tensorboard
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.8/dist-packages (from requests<3,>=2.21.0->te
Requirement already satisfied: chardet<5,>=3.0.2 in /usr/local/lib/python3.8/dist-packages (from requests<3,>=2.21.0->tensor
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.8/dist-packages (from importlib-metadata>=4.4->markdown>=
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.8/dist-packages (from pyasn1-modules>=0.2.1->g
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.8/dist-packages (from requests-oauthlib>=0.7.0->goo
```

```
pip install -U gensim
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: gensim in /usr/local/lib/python3.8/dist-packages (4.3.0)
Requirement already satisfied: FuzzyTM>=0.4.0 in /usr/local/lib/python3.8/dist-packages (from gensim) (2.0.5)
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.8/dist-packages (from gensim) (6.3.0)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.8/dist-packages (from gensim) (1.22.4)
Requirement already satisfied: scipy>=1.7.0 in /usr/local/lib/python3.8/dist-packages (from gensim) (1.7.3)
Requirement already satisfied: pandas in /usr/local/lib/python3.8/dist-packages (from FuzzyTM>=0.4.0->gensim) (1.3.5)
Requirement already satisfied: pyfume in /usr/local/lib/python3.8/dist-packages (from FuzzyTM>=0.4.0->gensim) (0.2.25)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.8/dist-packages (from pandas->FuzzyTM>=0.4.0
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-packages (from pandas->FuzzyTM>=0.4.0->gensim)
Requirement already satisfied: fst-pso in /usr/local/lib/python3.8/dist-packages (from pyfume->FuzzyTM>=0.4.0->gensim) (1.8.
Requirement already satisfied: simpful in /usr/local/lib/python3.8/dist-packages (from pyfume->FuzzyTM>=0.4.0->gensim) (2.10
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-packages (from python-dateutil>=2.7.3->pandas->Fuzz
Requirement already satisfied: miniful in /usr/local/lib/python3.8/dist-packages (from fst-pso->pyfume->FuzzyTM>=0.4.0->gens
Requirement already satisfied: requests in /usr/local/lib/python3.8/dist-packages (from simpful->pyfume->FuzzyTM>=0.4.0->gen
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-packages (from requests->simpful->pyfume->Fuzzy
Requirement already satisfied: chardet<5,>=3.0.2 in /usr/local/lib/python3.8/dist-packages (from requests->simpful->pyfume->
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.8/dist-packages (from requests->simpful->pyfu
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/dist-packages (from requests->simpful->pyfume-
```

1. Dataset Generation

```
# input_f = "gdrive/MyDrive/Colab Notebooks/amazon_reviews_us_Beauty_v1_00.tsv"
input_f = "amazon_reviews_us_Beauty_v1_00.tsv"
df = pd.read_csv(input_f,sep='\t', error_bad_lines=False, warn_bad_lines=False)
df
```

	marketplace	customer_id	review_id	product_id	product_parent	product_title	product_category	star_rating
0	US	1797882	R3I2DHQBR577SS	B001ANOOOE	2102612	The Naked Bee Vitmin C Moisturizing Sunscreen ...	Beauty	5
1	US	18381298	R1QNE9NQFJC2Y4	B0016J22EQ	106393691	Alba Botanica Sunless Tanning Lotion, 4 Ounce	Beauty	5
2	US	19242472	R3LIDG2Q4LJBAO	B00HU6UQAG	375449471	Elysee Infusion Skin Therapy Elixir, 2oz.	Beauty	5

```
## keep the reviews and rating fields in the input data frame
reviews = df[['star_rating', 'review_body']]
reviews
```

	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 ...
...	...	...
5087968	1	I tried this stuff and it made my hair feel oi...
5087969	4	This is an extremely good razor. It definitel...
5087970	5	I bought this because I have extremely dry sen...
5087971	5	Buy it for yourself. It's less than $15, and *...
5087972	2	I've been using the Norelco series for about 5...

5087973 rows × 2 columns

5087971	US	35142523	R1L6C2F1ZB6YKT	B000063XHQ	442263387	and For Hair	Beauty	5

```
## balance dataset with 60k reviews
## finding class 1
class_1_reviews = reviews.loc[(reviews['star_rating'] == 1) | (reviews['star_rating'] == 2)].reset_index(drop = True)
class_1_reviews = class_1_reviews.dropna()
## finding class 2
class_2_reviews = reviews.loc[reviews['star_rating'] == 3].reset_index(drop = True)
class_2_reviews = class_2_reviews.dropna()
## finding class 3
class_3_reviews = reviews.loc[(reviews['star_rating'] == 4) | (reviews['star_rating'] == 5)].reset_index(drop = True)
class_3_reviews = class_3_reviews.dropna()

## randomly select 20,000 for each class
n = 20000
class_1_select = class_1_reviews.groupby('star_rating', group_keys=False).apply(lambda x: x.sample(10000, random_state = 42))
class_2_select = class_2_reviews.sample(n, replace = False, random_state = 42)
class_3_select = class_3_reviews.groupby('star_rating', group_keys=False).apply(lambda x: x.sample(10000, random_state = 42))

df_all = pd.concat([class_1_select, class_2_select, class_3_select]).reset_index(drop=True)
df_all
```

	star_rating	review_body
0	1	The product was going to be a gift for my wife...
1	1	Most of the eyelashes werent good to work with...
2	1	I own a Jessy wig and love it, so I thought I ...
3	1	Was not good for my hair did not work the way ...

```
df_all['star_rating'].unique()
```

```
array([1, 2, 3, 4, 5.0], dtype=object)
```

```
## Preprocessing
rev = df_all['review_body']

## convert all reviews into lowercase
text_lowercase = rev.str.lower()

## remove the HTML and URLs from the reviews
text_notag = []
for item in text_lowercase:
    soup = BeautifulSoup(item, 'html.parser')
    text = soup.get_text()
    text = re.sub(r'https?://\S+', '', text)
    text_notag.append(text)

## remove non-alphabetical characters
text_cha = []
for item in text_notag:
    text = re.sub(r"[^a-zA-Z |']", ' ', item)
    text_cha.append(text)

## remove extra spaces
text_sp = []
for item in text_cha:
    text = re.sub(' +', ' ', item)
    text_sp.append(text)

## perform contractions
text_con = []
for item in text_sp:
    text = contractions.fix(item)
    text_con.append(text)

# ## remove stopwords
# stop_words = set(stopwords.words('english'))
# text_stop = []
# for item in text_con:
#     item_split = item.split()
#     i = ' '.join([word for word in item_split if word not in stop_words])
#     text_stop.append(i)
```

```
df_all['new'] = text_sp
```

```
df_all
```

	star_rating	review_body	new	class
0	1	The product was going to be a gift for my wife...	the product was going to be a gift for my wife...	1

```
class_spe = df_all['star_rating'].astype('int')
for i in range (len(class_spe)):
    if class_spe[i] == 1 or class_spe[i] == 2:
        class_spe[i] = 1
    if class_spe[i] == 3:
        class_spe[i] = 2
    if class_spe[i] == 4 or class_spe[i] == 5:
        class_spe[i] = 3
df_all['class'] = class_spe
df_all
```

	star_rating	review_body	new	class
0	1	The product was going to be a gift for my wife...	the product was going to be a gift for my wife...	1
1	1	Most of the eyelashes werent good to work with...	most of the eyelashes werent good to work with...	1
2	1	I own a Jessy wig and love it, so I thought I ...	i own a jessy wig and love it so i thought i w...	1
3	1	Was not good for my hair did not work the way ...	was not good for my hair did not work the way ...	1
4	1	Don't get this it is not worth the money I got...	don't get this it is not worth the money i got...	1
...	...	...	...	...
59995	5	Great product, makes hair cutting a breeze and...	great product makes hair cutting a breeze and ...	3
59996	5	With age comes with sagging in my face. This m...	with age comes with sagging in my face this mo...	3
59997	5	I love the glass bottle and the scent. Those w...	i love the glass bottle and the scent those wh...	3
59998	5	Bought it for my 12 year-old daughter loved th...	bought it for my year old daughter loved the c...	3
59999	5	good	good	3

60000 rows × 4 columns

2. Word Embedding

(a) Load the pretrained "word2vec-google-news-300"

```
import gensim.downloader as api
wv = api.load('word2vec-google-news-300')
```

```
[==================================================] 100.0% 1662.8/1662.8MB downloaded
```

```
## check semantic simiarity

##King -Man + woman = queen
a = wv.most_similar(positive=['King', 'Woman'], negative=['Man'], topn = 1)

## excellent ~ outstanding
b = wv.similarity('excellent', 'outstanding')

## bad ~ terrible
c = wv.similarity('bad', 'terrible')

print('check semantic similarity(part a)')
print('King - Man + woman = queen : ', a)
print('excellent ~ outstanding : ', b)
print('bad ~ terrible : ', c)
```

```
check semantic similarity(part a)
King - Man + woman = queen :  [('Queen', 0.4929388165473938)]
excellent ~ outstanding :  0.55674857
bad ~ terrible :  0.68286115
```

(b) Train a Word2Vec model using your own dataset.

```
df_all
```

	star_rating	review_body	new	class	vec
0	1	The product was going to be a gift for my wife...	the product was going to be a gift for my wife...	1	[-0.00430329, -0.010766511, 0.01682123, 0.0084...
1	1	Most of the eyelashes werent good to work with...	most of the eyelashes were not good to work wi...	1	[-0.005953749, -0.00430329, 0.007137782, -0.00...
2	1	I own a Jessy wig and love it, so I thought I ...	i own a jessy wig and love it so i thought i w...	1	[-0.008428434, -0.001132857, -0.0017439779, -0...
3	1	Was not good for my hair did not work the way ...	was not good for my hair did not work the way ...	1	[0.01682123, -0.0016720962, -0.0027225495, 0.0...
4	1	Don't get this it is not worth the money I got...	do not get this it is not worth the money i go...	1	[-0.0052448274, -0.016125174, 0.003165385, 0.0...
...	...	...	...	...	...
59995	5	Great product, makes hair cutting a breeze and...	great product makes hair cutting a breeze and ...	3	[-0.00023369631, -0.010766511, 0.004071655, 0....
59996	5	With age comes with sagging in my face. This m...	with age comes with sagging in my face this mo...	3	[0.0029325103, -0.015900223, 0.005987854, 0.00...

```
sentences = [word_tokenize(x) for x in df_all['review_body']]
```

```
model = Word2Vec(sentences=sentences, vector_size = 300, window = 13, min_count = 9)
```

```
## check semantic simiarity

##King -Man + woman = queen
# a = model.wv.most_similar(positive=['King', 'Woman'], negative=['Man'], topn = 1)

## excellent ~ outstanding
b = model.wv.similarity('excellent', 'outstanding')

## bad ~ terrible
c = model.wv.similarity('bad', 'terrible')


## vector('Paris') - vector('France') + vector('Italy') results in a vector that is very close to vector('Rome')

print('check semantic similarity(part b)')
print('King - Man + woman = queen : ', 'not included in the vocabulary')
print('excellent ~ outstanding : ', b)
print('bad ~ terrible : ', c)
```

```
check semantic similarity(part b)
King - Man + woman = queen :  not included in the vocabulary
excellent ~ outstanding :  0.62379426
bad ~ terrible :  0.61245257
```

Ans: After comparing vectors generated by yourself and the pretrained model, I think that the pretrained word2vec model is better because it has more vocabularies and have more accurate semantic similarities.

3. Simple models

```
X = df_all['new']
y = df_all['class'].astype('int')
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
def w2v(text):
    text = text.split(" ")
    t = np.zeros(300)
    n = len(text)
    for i in text:
        if i in wv:
            t += wv[i]
    return t/n
```

```
X_train_w2v = np.array([w2v(text) for text in X_train])
X_test_w2v = np.array([w2v(text) for text in X_test])
```

```
clf = Perceptron()
clf.fit(X_train_w2v, y_train)
```

```
Perceptron()
```

```
y_predict = clf.predict(X_test_w2v)
```

```
accuracy_score(y_test, y_predict)
```

```
0.5750833333333333
```

```
## using tfidf for comparision
from sklearn.feature_extraction.text import TfidfVectorizer
X = np.array(df_all['new'])
Y = np.array(df_all['star_rating'].astype('int'))
vectorizer = TfidfVectorizer(ngram_range=(1,3))
X= vectorizer.fit_transform(X)
X_train_tfidf, X_test_tfidf, y_train_tfidf, y_test_tfidf = train_test_split(X, Y, test_size=0.2)
```

```
clf_tfidf = Perceptron()
clf_tfidf.fit(X_train_tfidf, y_train_tfidf)
```

```
Perceptron()
```

```
y_predict_tfidf = clf_tfidf.predict(X_test_tfidf)
accuracy_score(y_test_tfidf, y_predict_tfidf)
```

```
0.51075
```

```
### SVM
Linear_SVC = LinearSVC()
Linear_SVC.fit(X_train_w2v, y_train)
```

```
LinearSVC()
```

```
y_predict_SVC = Linear_SVC.predict(X_test_w2v)
```

```
accuracy_score(y_test, y_predict_SVC)
```

```
0.6169166666666667
```

```
#tfidf comparision
Linear_SVC_tfidf = LinearSVC()
Linear_SVC_tfidf.fit(X_train_tfidf, y_train_tfidf)
```

```
LinearSVC()
```

```
y_predict_SVC_tfidf = Linear_SVC_tfidf.predict(X_test_tfidf)
```

```
print('Accuracy values with features Word2Vec for Perceptron is ', accuracy_score(y_test, y_predict))
print('Accuracy values with features TF-IDF for Perceptron is ', accuracy_score(y_test_tfidf, y_predict_tfidf))
print('Accuracy values with features Word2Vec for Perceptron is ', accuracy_score(y_test, y_predict_SVC))
print('Accuracy values with features TF-IDF for Perceptron is ', accuracy_score(y_test_tfidf, y_predict_SVC_tfidf))
```

```
Accuracy values with features Word2Vec for Perceptron is  0.5750833333333333
Accuracy values with features TF-IDF for Perceptron is  0.51075
Accuracy values with features Word2Vec for Perceptron is  0.6169166666666667
Accuracy values with features TF-IDF for Perceptron is  0.5506666666666666
```

Ans: Comparing the accuracy values, I found out that word2vec features has better performance on models compare to TF-IDF features with same preprocessing. SVM models have better performance than perceptron models for both types of features.

4. Feedforward Neural Networks

```
from tensorflow.keras import layers
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.datasets import mnist
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.utils import to_categorical
```

(a) use the average Word2Vec vectors similar to the “Simple models” section and train the neural network.

```
from keras.layers import Dense, Embedding, LSTM, Flatten
from keras import Input
# Build the model.
model = Sequential()

model.add(Input(shape=(X_train_w2v.shape[1],)))
## Embedding(vocab_size, 300, input_length=X_train_w2v.shape[1])
## Input(shape=(X_train_w2v.shape[1],))
# model.add(Dense(128, input_dim=300, activation='relu'))
## Dense(64, activation='relu', input_shape = X_train_w2v[0].shape)
model.add(Flatten())

model.add(Dense(100, activation='relu'))
# model.add(Dropout(0.7))
model.add(Dense(10, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(3, activation='softmax'))

# Display the model summary.
model.summary()

model.compile(optimizer='Adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_2 (Flatten)         (None, 300)               0

 dense_27 (Dense)            (None, 100)               30100

 dense_28 (Dense)            (None, 10)                1010

 dropout_8 (Dropout)         (None, 10)                0

 dense_29 (Dense)            (None, 3)                 33

=================================================================
Total params: 31,143
Trainable params: 31,143
Non-trainable params: 0
_________________________________________________________________
```

```
y_train_onehot=to_categorical(y_train -1)
y_test_onehot=to_categorical(y_test - 1)
```

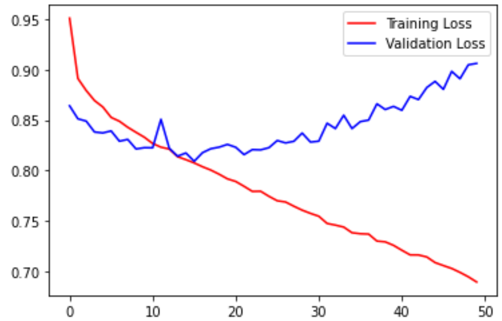
```
from tensorflow.keras.callbacks import EarlyStopping
# callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)
training_results = model.fit(X_train_w2v,
                             y_train_onehot,
                             epochs=50,
                             batch_size=50,
                             validation_data=(X_test_w2v, y_test_onehot))
```

```
Epoch 1/50
960/960 [==============================] - 4s 3ms/step - loss: 0.9511 - accuracy: 0.5294 - val_loss: 0.8642 - val_accurac
Epoch 2/50
960/960 [==============================] - 3s 3ms/step - loss: 0.8913 - accuracy: 0.5798 - val_loss: 0.8513 - val_accurac
Epoch 3/50
960/960 [==============================] - 3s 3ms/step - loss: 0.8794 - accuracy: 0.5883 - val_loss: 0.8490 - val_accurac
Epoch 4/50
960/960 [==============================] - 3s 3ms/step - loss: 0.8694 - accuracy: 0.5971 - val_loss: 0.8382 - val_accurac
Epoch 5/50
960/960 [==============================] - 3s 3ms/step - loss: 0.8631 - accuracy: 0.5990 - val_loss: 0.8373 - val_accurac
Epoch 6/50
960/960 [==============================] - 3s 3ms/step - loss: 0.8527 - accuracy: 0.6043 - val_loss: 0.8393 - val_accurac
Epoch 7/50
960/960 [==============================] - 3s 3ms/step - loss: 0.8489 - accuracy: 0.6067 - val_loss: 0.8291 - val_accurac
Epoch 8/50
960/960 [==============================] - 3s 3ms/step - loss: 0.8429 - accuracy: 0.6089 - val_loss: 0.8309 - val_accurac
Epoch 9/50
960/960 [==============================] - 3s 3ms/step - loss: 0.8379 - accuracy: 0.6064 - val_loss: 0.8214 - val_accurac
Epoch 10/50
960/960 [==============================] - 3s 3ms/step - loss: 0.8331 - accuracy: 0.6127 - val_loss: 0.8227 - val_accurac
Epoch 11/50
960/960 [==============================] - 3s 3ms/step - loss: 0.8268 - accuracy: 0.6131 - val_loss: 0.8226 - val_accurac
Epoch 12/50
960/960 [==============================] - 3s 3ms/step - loss: 0.8232 - accuracy: 0.6173 - val_loss: 0.8509 - val_accurac
Epoch 13/50
960/960 [==============================] - 3s 3ms/step - loss: 0.8213 - accuracy: 0.6164 - val_loss: 0.8221 - val_accurac
Epoch 14/50
960/960 [==============================] - 3s 3ms/step - loss: 0.8138 - accuracy: 0.6198 - val_loss: 0.8143 - val_accurac
Epoch 15/50
960/960 [==============================] - 3s 3ms/step - loss: 0.8108 - accuracy: 0.6219 - val_loss: 0.8174 - val_accurac
Epoch 16/50
960/960 [==============================] - 3s 3ms/step - loss: 0.8076 - accuracy: 0.6209 - val_loss: 0.8092 - val_accurac
Epoch 17/50
960/960 [==============================] - 3s 3ms/step - loss: 0.8038 - accuracy: 0.6253 - val_loss: 0.8177 - val_accurac
Epoch 18/50
960/960 [==============================] - 3s 3ms/step - loss: 0.8005 - accuracy: 0.6262 - val_loss: 0.8217 - val_accurac
Epoch 19/50
960/960 [==============================] - 3s 3ms/step - loss: 0.7964 - accuracy: 0.6308 - val_loss: 0.8232 - val_accurac
Epoch 20/50
960/960 [==============================] - 3s 3ms/step - loss: 0.7918 - accuracy: 0.6306 - val_loss: 0.8259 - val_accurac
Epoch 21/50
960/960 [==============================] - 3s 3ms/step - loss: 0.7892 - accuracy: 0.6328 - val_loss: 0.8231 - val_accurac
Epoch 22/50
960/960 [==============================] - 3s 3ms/step - loss: 0.7844 - accuracy: 0.6315 - val_loss: 0.8158 - val_accurac
Epoch 23/50
960/960 [==============================] - 3s 3ms/step - loss: 0.7793 - accuracy: 0.6379 - val_loss: 0.8208 - val_accurac
Epoch 24/50
960/960 [==============================] - 3s 3ms/step - loss: 0.7795 - accuracy: 0.6361 - val_loss: 0.8205 - val_accurac
Epoch 25/50
960/960 [==============================] - 3s 3ms/step - loss: 0.7745 - accuracy: 0.6414 - val_loss: 0.8227 - val_accurac
Epoch 26/50
960/960 [==============================] - 3s 3ms/step - loss: 0.7701 - accuracy: 0.6432 - val_loss: 0.8298 - val_accurac
Epoch 27/50
960/960 [==============================] - 3s 3ms/step - loss: 0.7688 - accuracy: 0.6391 - val_loss: 0.8274 - val_accurac
Epoch 28/50
960/960 [==============================] - 3s 3ms/step - loss: 0.7646 - accuracy: 0.6450 - val_loss: 0.8290 - val_accurac
```

```
df_fnn = pd.DataFrame(training_results.history)
df_fnn
```

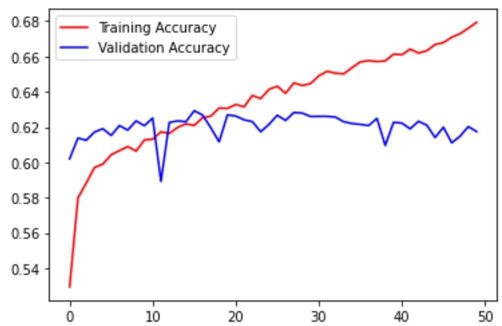
	loss	accuracy	val_loss	val_accuracy
0	0.951131	0.529396	0.864207	0.602000
1	0.891318	0.579833	0.851349	0.613750
2	0.879392	0.588271	0.849039	0.612500
3	0.869433	0.597125	0.838249	0.617167
4	0.863080	0.599000	0.837328	0.619083
5	0.852739	0.604312	0.839308	0.615250
6	0.848888	0.606667	0.829117	0.620833
7	0.842946	0.608938	0.830892	0.618250
8	0.837919	0.606375	0.821400	0.623500
9	0.833050	0.612688	0.822705	0.620750
10	0.826841	0.613146	0.822590	0.625083
11	0.823201	0.617312	0.850856	0.589167
12	0.821338	0.616354	0.822150	0.622583
13	0.813778	0.619812	0.814250	0.623500
14	0.810809	0.621854	0.817435	0.623000
15	0.807580	0.620917	0.809215	0.629250
16	0.803810	0.625313	0.817722	0.626833
17	0.800483	0.626188	0.821650	0.619583
18	0.796390	0.630813	0.823213	0.611583
19	0.791797	0.630562	0.825941	0.626917
20	0.789184	0.632833	0.823112	0.626250
21	0.784402	0.631479	0.815803	0.624083
22	0.779334	0.637917	0.820768	0.623083
23	0.779481	0.636146	0.820518	0.617333
24	0.774464	0.641354	0.822700	0.621500
25	0.770052	0.643167	0.829785	0.626667
26	0.768840	0.639146	0.827414	0.623750
27	0.764634	0.645042	0.829031	0.628250
28	0.760727	0.643583	0.837176	0.627917
29	0.757524	0.644604	0.828259	0.626000
30	0.754705	0.649042	0.829139	0.626083
31	0.747561	0.651625	0.846958	0.626083
32	0.745893	0.650563	0.841632	0.625667
33	0.743960	0.650125	0.854901	0.623000
34	0.738340	0.653708	0.841622	0.622000
35	0.737350	0.656958	0.848527	0.621500
36	0.737024	0.657646	0.850061	0.620833
37	0.730072	0.657146	0.866195	0.624917
38	0.729309	0.657500	0.860540	0.609583

```
loss=df_fnn['loss']
val_loss=df_fnn['val_loss']
epochs=range(len(loss)) # Get number of epochs
import matplotlib.pyplot as plt
# Plot training and validation loss per epoch
plt.plot(epochs, loss, 'r', label="Training Loss")
plt.plot(epochs, val_loss, 'b', label="Validation Loss")
plt.legend()
plt.show()
```




```
acc=df_fnn['accuracy']
val_acc=df_fnn['val_accuracy']
epochs=range(len(acc)) # Get number of epochs

# Plot training and validation loss per epoch
plt.plot(epochs, acc, 'r', label="Training Accuracy")
plt.plot(epochs, val_acc, 'b', label="Validation Accuracy")
plt.legend()
plt.show()
```




```
re = model.evaluate(X_test_w2v, y_test_onehot)
print('Accuracy for 4a FNN is ' , re[1])
```

```
375/375 [==============================] - 1s 2ms/step - loss: 0.9063 - accuracy: 0.6174
Accuracy for 4a FNN is  0.6174166798591614
```

b) concatenate the first 10 Word2Vec vectors for each review as the input feature (x = [WT 1,...,WT 10]) and train the neural network.

```
def w2v_first10(text):
    text = text.split(" ")
    v = np.zeros((300*10))
    j = 0
    for i in text:
      if j == 10:
        break
      if i in wv:
        v[j*300:(j+1)*300] = wv[i]
        j += 1
    return np.array(v)
```

```
w2v_first10('the product was going to be').shape
```

```
the
product
was
going
```

```
be
(10, 300)
```

```
X_train_w2v_first10 = np.array([w2v_first10(text) for text in X_train])
X_test_w2v_first10 = np.array([w2v_first10(text) for text in X_test])
```

```
X_train_w2v_first10.shape
```

```
(48000, 3000)
```

```
from keras.layers import Dense, Embedding, LSTM, Flatten, Dropout, Conv2D, MaxPooling2D
from keras import Input
from tensorflow.keras.optimizers import Adam
# Build the model.
model = Sequential()

model.add(Input(shape=(X_train_w2v_first10.shape[1],)))
# model.add(Input(shape=(X_train_w2v.shape[1],)))
## Embedding(vocab_size, 300, input_length=X_train_w2v.shape[1])
## Input(shape=(X_train_w2v.shape[1],))
# model.add(Dense(128, input_dim=300, activation='relu'))
# model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.7))
model.add(Dense(10, activation='relu'))
# model.add(Dropout(0.7))

model.add(Dense(3, activation='softmax'))
## Display the model summary.
model.summary()
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
model.compile(optimizer=Adam(learning_rate=0.01),
              loss='categorical_crossentropy',

              metrics=['accuracy'])
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_18 (Dense)            (None, 100)               300100

 dropout_6 (Dropout)         (None, 100)               0

 dense_19 (Dense)            (None, 10)                1010

 dense_20 (Dense)            (None, 3)                 33

=================================================================
Total params: 301,143
Trainable params: 301,143
Non-trainable params: 0
_________________________________________________________________
```

```
training_results_5b = model.fit(X_train_w2v_first10,
                             y_train_onehot,
                             epochs=50,
                             batch_size=64,
                             callbacks = [callback],
                             validation_data=(X_test_w2v_first10, y_test_onehot))
```

```
Epoch 1/50
750/750 [==============================] - 4s 4ms/step - loss: 1.0420 - accuracy: 0.4359 - val_loss: 0.9787 - val_accuracy:
Epoch 2/50
750/750 [==============================] - 3s 3ms/step - loss: 1.0004 - accuracy: 0.4762 - val_loss: 0.9588 - val_accuracy:
Epoch 3/50
750/750 [==============================] - 2s 3ms/step - loss: 0.9833 - accuracy: 0.4888 - val_loss: 0.9610 - val_accuracy:
Epoch 4/50
750/750 [==============================] - 3s 4ms/step - loss: 0.9752 - accuracy: 0.4984 - val_loss: 0.9552 - val_accuracy:
Epoch 5/50
750/750 [==============================] - 2s 3ms/step - loss: 0.9635 - accuracy: 0.5099 - val_loss: 0.9474 - val_accuracy:
Epoch 6/50
750/750 [==============================] - 2s 3ms/step - loss: 0.9598 - accuracy: 0.5110 - val_loss: 0.9433 - val_accuracy:
Epoch 7/50
750/750 [==============================] - 3s 3ms/step - loss: 0.9511 - accuracy: 0.5179 - val_loss: 0.9382 - val_accuracy:
Epoch 8/50
750/750 [==============================] - 3s 4ms/step - loss: 0.9443 - accuracy: 0.5242 - val_loss: 0.9339 - val_accuracy:
```

```
Epoch 9/50
750/750 [==============================] - 3s 4ms/step - loss: 0.9384 - accuracy: 0.5267 - val_loss: 0.9475 - val_accuracy:
Epoch 10/50
750/750 [==============================] - 2s 3ms/step - loss: 0.9321 - accuracy: 0.5282 - val_loss: 0.9374 - val_accuracy:
Epoch 11/50
750/750 [==============================] - 2s 3ms/step - loss: 0.9309 - accuracy: 0.5321 - val_loss: 0.9394 - val_accuracy:
Epoch 12/50
750/750 [==============================] - 3s 4ms/step - loss: 0.9238 - accuracy: 0.5359 - val_loss: 0.9423 - val_accuracy:
Epoch 13/50
750/750 [==============================] - 2s 3ms/step - loss: 0.9186 - accuracy: 0.5390 - val_loss: 0.9513 - val_accuracy:
```

```
re2 = model.evaluate(X_test_w2v_first10, y_test_onehot)
print('Accuracy for 4b FNN is ' , re2[1])
```

```
375/375 [==============================] - 1s 2ms/step - loss: 0.9513 - accuracy: 0.5314
Accuracy for 4b FNN is  0.531416654586792
```

Ans: The model from 4a has better accuracy about 61%. The seconde is the model from simple model perceptron. 4b has the least accuracy model about 53%.

5. Recurrent Neural Networks

(a) Train a simple RNN for sentiment analysis

```
def w2v_first10(text):
    text = text.split(" ")
    n = len(text)
    d = []
    j = 0
    # t = np.zeros(300)
    for i in text:
        if j == 10:
            break
        if i in wv:
            print(i)
            j += 1
            d.append(wv[i])
    for k in range(j, 10):
        d.append(np.zeros(300))
    return np.array(d)
```

```
def w2v_first20(text):
    text = text.split(" ")
    d = []
    j = 0
    # t = np.zeros(300)
    for i in text:
        if j == 20:
            break
        if i in wv:
            j += 1
            d.append(wv[i])
    for k in range(j, 20):
        d.append(np.zeros(300))
    return np.array(d)
```

```
X_train_w2v_first20 = np.array([w2v_first20(text) for text in X_train])
X_test_w2v_first20 = np.array([w2v_first20(text) for text in X_test])
```

```
import tensorflow as tf
X_train_w2v_first20 = np.asarray(X_train_w2v_first20).astype('float32')
train_tensor = tf.convert_to_tensor(X_train_w2v_first20)

X_test_w2v_first20 = np.asarray(X_test_w2v_first20).astype('float32')
test_tensor = tf.convert_to_tensor(X_test_w2v_first20)
```

```
y_tensor1 = np.asarray(y_train).astype('float32')
y_tensor_train = tf.convert_to_tensor(y_tensor1)

y_tensor2 = np.asarray(y_test).astype('float32')
y_tensor_test = tf.convert_to_tensor(y_tensor2)
```

```
test_tensor.shape
```

```
TensorShape([12000, 20, 300])
```

```
from keras.models import Sequential
from keras.layers import Embedding, SimpleRNN

model = Sequential()
model.add(Input(shape=(train_tensor.shape[1],train_tensor.shape[2])))
# model.add(Flatten())
# model.add(Embedding(train_tensor.shape[0], 300, input_length = 20))
model.add(SimpleRNN(20, return_sequences=True))
model.add(Flatten())
# model.add(Dense(10, activation = 'relu'))
model.add(Dense(3, activation='softmax'))
model.add(Dropout(0.4))
# model.build(input_shape=X_train_w2v_first20.shape)
model.summary()

model.compile(optimizer=Adam(learning_rate=0.0001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
Model: "sequential_31"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_30 (SimpleRNN)   (None, 20, 20)            6420

 flatten_24 (Flatten)        (None, 400)               0

 dense_33 (Dense)            (None, 3)                 1203

 dropout_21 (Dropout)        (None, 3)                 0

=================================================================
Total params: 7,623
Trainable params: 7,623
Non-trainable params: 0
_________________________________________________________________
```

```
training_results_5a = model.fit(train_tensor,
                             y_train_onehot,
                             epochs=50,
                             batch_size=50,

                             validation_data=(test_tensor, y_test_onehot))
```

```
Epoch 1/50
960/960 [==============================] - 12s 11ms/step - loss: nan - accuracy: 0.3516 - val_loss: 1.0424 - val_accuracy
Epoch 2/50
960/960 [==============================] - 10s 10ms/step - loss: nan - accuracy: 0.4027 - val_loss: 0.9831 - val_accuracy
Epoch 3/50
960/960 [==============================] - 9s 10ms/step - loss: nan - accuracy: 0.4216 - val_loss: 0.9618 - val_accuracy:
Epoch 4/50
960/960 [==============================] - 10s 10ms/step - loss: nan - accuracy: 0.4260 - val_loss: 0.9519 - val_accuracy
Epoch 5/50
960/960 [==============================] - 10s 10ms/step - loss: nan - accuracy: 0.4302 - val_loss: 0.9422 - val_accuracy
Epoch 6/50
960/960 [==============================] - 9s 10ms/step - loss: nan - accuracy: 0.4337 - val_loss: 0.9350 - val_accuracy:
Epoch 7/50
960/960 [==============================] - 10s 10ms/step - loss: nan - accuracy: 0.4386 - val_loss: 0.9292 - val_accuracy
Epoch 8/50
960/960 [==============================] - 10s 10ms/step - loss: nan - accuracy: 0.4389 - val_loss: 0.9241 - val_accuracy
Epoch 9/50
960/960 [==============================] - 10s 11ms/step - loss: nan - accuracy: 0.4392 - val_loss: 0.9203 - val_accuracy
Epoch 10/50
960/960 [==============================] - 9s 9ms/step - loss: nan - accuracy: 0.4449 - val_loss: 0.9132 - val_accuracy:
Epoch 11/50
```

```
960/960 [==============================] - 10s 10ms/step - loss: nan - accuracy: 0.4496 - val_loss: 0.9116 - val_accuracy
Epoch 12/50
960/960 [==============================] - 10s 10ms/step - loss: nan - accuracy: 0.4505 - val_loss: 0.9071 - val_accuracy
Epoch 13/50
960/960 [==============================] - 9s 10ms/step - loss: nan - accuracy: 0.4446 - val_loss: 0.9043 - val_accuracy:
Epoch 14/50
960/960 [==============================] - 10s 10ms/step - loss: nan - accuracy: 0.4484 - val_loss: 0.9016 - val_accuracy
Epoch 15/50
960/960 [==============================] - 10s 10ms/step - loss: nan - accuracy: 0.4515 - val_loss: 0.9030 - val_accuracy
Epoch 16/50
960/960 [==============================] - 10s 10ms/step - loss: nan - accuracy: 0.4514 - val_loss: 0.9000 - val_accuracy
Epoch 17/50
960/960 [==============================] - 9s 9ms/step - loss: nan - accuracy: 0.4531 - val_loss: 0.8963 - val_accuracy:
Epoch 18/50
960/960 [==============================] - 10s 10ms/step - loss: nan - accuracy: 0.4532 - val_loss: 0.8972 - val_accuracy
Epoch 19/50
960/960 [==============================] - 10s 10ms/step - loss: nan - accuracy: 0.4496 - val_loss: 0.8939 - val_accuracy
Epoch 20/50
960/960 [==============================] - 9s 9ms/step - loss: nan - accuracy: 0.4553 - val_loss: 0.8924 - val_accuracy:
Epoch 21/50
960/960 [==============================] - 10s 10ms/step - loss: nan - accuracy: 0.4535 - val_loss: 0.8942 - val_accuracy
Epoch 22/50
960/960 [==============================] - 10s 10ms/step - loss: nan - accuracy: 0.4539 - val_loss: 0.8909 - val_accuracy
Epoch 23/50
960/960 [==============================] - 11s 11ms/step - loss: nan - accuracy: 0.4507 - val_loss: 0.8900 - val_accuracy
Epoch 24/50
960/960 [==============================] - 9s 9ms/step - loss: nan - accuracy: 0.4551 - val_loss: 0.8942 - val_accuracy:
Epoch 25/50
960/960 [==============================] - 10s 10ms/step - loss: nan - accuracy: 0.4539 - val_loss: 0.8917 - val_accuracy
Epoch 26/50
960/960 [==============================] - 10s 10ms/step - loss: nan - accuracy: 0.4565 - val_loss: 0.8882 - val_accuracy
Epoch 27/50
960/960 [==============================] - 10s 10ms/step - loss: nan - accuracy: 0.4564 - val_loss: 0.8885 - val_accuracy
Epoch 28/50
```

```
re3 = model.evaluate(test_tensor, y_test_onehot)
print('Accuracy for 5a simple RNN is ' , re3[1])
```

```
375/375 [==============================] - 2s 4ms/step - loss: 0.8801 - accuracy: 0.5809
Accuracy for 4b FNN is  0.5809166431427002
```

Ans: I conclude that the RNN simple model have worse performance(about 58% accuracy) than the Feedforward neural network.

5b. GRU RNN

```
from keras.layers import Embedding, SimpleRNN, Bidirectional, GRU
model = Sequential()
model.add(Input(shape=(train_tensor.shape[1],train_tensor.shape[2])))
# model.add(Flatten())
# model.add(Embedding(train_tensor.shape[0], 300, input_length = 20))
model.add(Bidirectional(GRU(20, return_sequences=True)))
model.add(Flatten())
# model.add(Dense(10, activation = 'relu'))
model.add(Dense(3, activation='softmax'))
model.add(Dropout(0.4))
# model.build(input_shape=X_train_w2v_first20.shape)
model.summary()

model.compile(optimizer=Adam(learning_rate=0.0001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
Model: "sequential_34"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 bidirectional_2 (Bidirectio  (None, 20, 40)           38640     
 nal)                                                            
                                                                 
 flatten_27 (Flatten)        (None, 800)               0         
                                                                 
 dense_36 (Dense)            (None, 3)                 2403      
                                                                 
 dropout_24 (Dropout)        (None, 3)                 0         
```

```
=================================================================
Total params: 41,043
Trainable params: 41,043
Non-trainable params: 0
_________________________________________________________________
```

```
training_results_5b = model.fit(train_tensor,
                                y_train_onehot,
                                epochs=75,
                                batch_size=50,

                                validation_data=(test_tensor, y_test_onehot))
```

```
Epoch 48/75
960/960 [==============================] - 6s 7ms/step - loss: nan - accuracy: 0.4859 - val_loss: 0.8214 - val_accuracy:
Epoch 49/75
960/960 [==============================] - 6s 6ms/step - loss: nan - accuracy: 0.4882 - val_loss: 0.8216 - val_accuracy:
Epoch 50/75
960/960 [==============================] - 6s 7ms/step - loss: nan - accuracy: 0.4850 - val_loss: 0.8344 - val_accuracy:
Epoch 51/75
960/960 [==============================] - 6s 6ms/step - loss: nan - accuracy: 0.4818 - val_loss: 0.8500 - val_accuracy:
Epoch 52/75
960/960 [==============================] - 6s 7ms/step - loss: nan - accuracy: 0.4855 - val_loss: 0.8240 - val_accuracy:
Epoch 53/75
960/960 [==============================] - 6s 6ms/step - loss: nan - accuracy: 0.4879 - val_loss: 0.8212 - val_accuracy:
Epoch 54/75
960/960 [==============================] - 6s 7ms/step - loss: nan - accuracy: 0.4849 - val_loss: 0.8217 - val_accuracy:
Epoch 55/75
960/960 [==============================] - 6s 6ms/step - loss: nan - accuracy: 0.4851 - val_loss: 0.8233 - val_accuracy:
Epoch 56/75
960/960 [==============================] - 6s 6ms/step - loss: nan - accuracy: 0.4871 - val_loss: 0.8223 - val_accuracy:
Epoch 57/75
960/960 [==============================] - 6s 6ms/step - loss: nan - accuracy: 0.4884 - val_loss: 0.8293 - val_accuracy:
Epoch 58/75
960/960 [==============================] - 6s 6ms/step - loss: nan - accuracy: 0.4864 - val_loss: 0.8208 - val_accuracy:
Epoch 59/75
960/960 [==============================] - 6s 7ms/step - loss: nan - accuracy: 0.4850 - val_loss: 0.8216 - val_accuracy:
Epoch 60/75
960/960 [==============================] - 7s 7ms/step - loss: nan - accuracy: 0.4889 - val_loss: 0.8178 - val_accuracy:
Epoch 61/75
960/960 [==============================] - 6s 7ms/step - loss: nan - accuracy: 0.4886 - val_loss: 0.8241 - val_accuracy:
Epoch 62/75
960/960 [==============================] - 6s 6ms/step - loss: nan - accuracy: 0.4844 - val_loss: 0.8299 - val_accuracy:
Epoch 63/75
960/960 [==============================] - 6s 7ms/step - loss: nan - accuracy: 0.4938 - val_loss: 0.8212 - val_accuracy:
Epoch 64/75
960/960 [==============================] - 6s 6ms/step - loss: nan - accuracy: 0.4899 - val_loss: 0.8197 - val_accuracy:
Epoch 65/75
960/960 [==============================] - 6s 7ms/step - loss: nan - accuracy: 0.4870 - val_loss: 0.8192 - val_accuracy:
Epoch 66/75
960/960 [==============================] - 6s 6ms/step - loss: nan - accuracy: 0.4936 - val_loss: 0.8332 - val_accuracy:
Epoch 67/75
960/960 [==============================] - 6s 7ms/step - loss: nan - accuracy: 0.4901 - val_loss: 0.8264 - val_accuracy:
Epoch 68/75
960/960 [==============================] - 6s 6ms/step - loss: nan - accuracy: 0.4916 - val_loss: 0.8206 - val_accuracy:
Epoch 69/75
960/960 [==============================] - 6s 7ms/step - loss: nan - accuracy: 0.4904 - val_loss: 0.8271 - val_accuracy:
Epoch 70/75
960/960 [==============================] - 6s 6ms/step - loss: nan - accuracy: 0.4916 - val_loss: 0.8208 - val_accuracy:
Epoch 71/75
960/960 [==============================] - 6s 7ms/step - loss: nan - accuracy: 0.4929 - val_loss: 0.8246 - val_accuracy:
Epoch 72/75
960/960 [==============================] - 6s 6ms/step - loss: nan - accuracy: 0.4939 - val_loss: 0.8269 - val_accuracy:
Epoch 73/75
960/960 [==============================] - 6s 6ms/step - loss: nan - accuracy: 0.4917 - val_loss: 0.8233 - val_accuracy:
Epoch 74/75
960/960 [==============================] - 6s 6ms/step - loss: nan - accuracy: 0.4978 - val_loss: 0.8686 - val_accuracy:
Epoch 75/75
960/960 [==============================] - 6s 6ms/step - loss: nan - accuracy: 0.4935 - val_loss: 0.8266 - val_accuracy:
```

```
re4 = model.evaluate(test_tensor, y_test_onehot)
print('Accuracy for 5b GRU  is ' , re4[1])
```

```
375/375 [==============================] - 2s 4ms/step - loss: 0.8266 - accuracy: 0.6243
Accuracy for 5b GRU  is  0.6243333220481873
```

5c) LSTM

```
from keras.layers import Embedding, SimpleRNN, Bidirectional, GRU, LSTM
model = Sequential()
model.add(Input(shape=(train_tensor.shape[1],train_tensor.shape[2])))
# model.add(Flatten())
# model.add(Embedding(train_tensor.shape[0], 300, input_length = 20))
model.add(Bidirectional(LSTM(20, return_sequences=True)))
model.add(Flatten())
# model.add(Dense(10, activation = 'relu'))
model.add(Dense(3, activation='softmax'))
model.add(Dropout(0.4))
# model.build(input_shape=X_train_w2v_first20.shape)
model.summary()

model.compile(optimizer=Adam(learning_rate=0.0001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
Model: "sequential_33"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 bidirectional_1 (Bidirectio  (None, 20, 40)           51360     
 nal)                                                            
                                                                 
 flatten_26 (Flatten)        (None, 800)               0         
                                                                 
 dense_35 (Dense)            (None, 3)                 2403      
                                                                 
 dropout_23 (Dropout)        (None, 3)                 0         
                                                                 
=================================================================
Total params: 53,763
Trainable params: 53,763
Non-trainable params: 0
_________________________________________________________________
```

```
training_results_5C = model.fit(train_tensor,
                             y_train_onehot,
                             epochs=75,
                             batch_size=50,

                             validation_data=(test_tensor, y_test_onehot))
```

```
Epoch 65/75
960/960 [==============================] - 6s 7ms/step - loss: nan - accuracy: 0.4923 - val_loss: 0.8346 - val_accuracy:
Epoch 66/75
960/960 [==============================] - 7s 7ms/step - loss: nan - accuracy: 0.4943 - val_loss: 0.8289 - val_accuracy:
Epoch 67/75
960/960 [==============================] - 6s 7ms/step - loss: nan - accuracy: 0.4977 - val_loss: 0.8375 - val_accuracy:
Epoch 68/75
960/960 [==============================] - 7s 7ms/step - loss: nan - accuracy: 0.4929 - val_loss: 0.8291 - val_accuracy:
Epoch 69/75
960/960 [==============================] - 6s 7ms/step - loss: nan - accuracy: 0.4931 - val_loss: 0.8396 - val_accuracy:
Epoch 70/75
960/960 [==============================] - 7s 7ms/step - loss: nan - accuracy: 0.4943 - val_loss: 0.8366 - val_accuracy:
Epoch 71/75
960/960 [==============================] - 6s 7ms/step - loss: nan - accuracy: 0.4963 - val_loss: 0.8325 - val_accuracy:
Epoch 72/75
960/960 [==============================] - 6s 6ms/step - loss: nan - accuracy: 0.4966 - val_loss: 0.8473 - val_accuracy:
Epoch 73/75
960/960 [==============================] - 6s 7ms/step - loss: nan - accuracy: 0.4965 - val_loss: 0.8333 - val_accuracy:
Epoch 74/75
960/960 [==============================] - 6s 6ms/step - loss: nan - accuracy: 0.4984 - val_loss: 0.8376 - val_accuracy:
Epoch 75/75
```

```
re5 = model.evaluate(test_tensor, y_test_onehot)
print('Accuracy for 5b GRU  is ' , re5[1])
```

```
375/375 [==============================] - 2s 4ms/step - loss: 0.8312 - accuracy: 0.6234
Accuracy for 5b GRU  is  0.6234166622161865
```

Ans: I conclude that GRU and LSTM layer has similar and high accuracy(62%) than any other models. Simple RNN modle has the least performance(58%)

✓ 2s completed at 12:29 PM