Assignment 3 - CSCI544 - Krish Sukhani

Task 1 - Dataset Creation

Used the data.tsv file -- Location: Same folder

In [1]:

```
import warnings
import pandas as pd
import numpy as np
warnings.filterwarnings('ignore')
# Dataset: https://s3.amazonaws.com/amazon-reviews-pds/tsv/amazon_reviews_us_Beauty_
```

In [2]:

```
data = pd.read_csv('data.tsv', on_bad_lines='skip', sep='\t', low_memory=False)
```

In [3]:

```
data.head()
```

Out[3]:

	marketplace	customer_id	review_id	product_id	product_parent	product_title
0	US	1797882	R3I2DHQBR577SS	B001ANOOOE	2102612	The Naked Bee Vitmin C Moisturizing Sunscreen ...
1	US	18381298	R1QNE9NQFJC2Y4	B0016J22EQ	106393691	Alba Botanica Sunless Tanning Lotion, 4 Ounce
2	US	19242472	R3LIDG2Q4LJBAO	B00HU6UQAG	375449471	Elysee Infusion Skin Therapy Elixir, 2oz.
3	US	19551372	R3KSZHPAEVPEAL	B002HWS7RM	255651889	Diane D722 Color, Perm And Conditioner Process...
4	US	14802407	RAI2OIG50KZ43	B00SM99KWU	116158747	Biore UV Aqua Rich Watery Essence SPF50+/PA+++...

Keep Reviews and Ratings

In [4]:

```
data = data[['review_body', 'star_rating']]
data.head()
```

Out[4]:

	review_body	star_rating
0	Love this, excellent sun block!!	5
1	The great thing about this cream is that it do...	5
2	Great Product, I'm 65 years old and this is al...	5
3	I use them as shower caps & conditioning caps....	5
4	This is my go-to daily sunblock. It leaves no ...	5

In [5]:

```
data.isnull().sum()
```

Out[5]:

```
review_body    400
star_rating     10
dtype: int64
```

In [6]:

```
data = data.dropna()
```

In [7]:

```
data.isnull().sum()
data
```

Out[7]:

	review_body	star_rating
0	Love this, excellent sun block!!	5
1	The great thing about this cream is that it do...	5
2	Great Product, I'm 65 years old and this is al...	5
3	I use them as shower caps & conditioning caps....	5
4	This is my go-to daily sunblock. It leaves no ...	5
...	...	...
5094302	After watching my Dad struggle with his scisso...	5
5094303	Like most sound machines, the sounds choices a...	3
5094304	I bought this product because it indicated 30 ...	5
5094305	We have used Oral-B products for 15 years; thi...	5
5094306	I love this toothbrush. It's easy to use, and ...	5

5093907 rows × 2 columns

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

Ref: https://sparkbyexamples.com/pandas/pandas-replace-values-based-on-condition/#:~:text=You%20can%20replace%20values%20of,the%20values%20of%20pandas%20DataFra (https://sparkbyexamples.com/pandas/pandas-replace-values-based-on-condition/#:~:text=You%20can%20replace%20values%20of,the%20values%20of%20pandas%20DataFra

In [8]:

```
data['star_rating'] = np.where(data['star_rating'] == '1', 'class1', data['star_rati
data['star_rating'] = np.where(data['star_rating'] == '2', 'class1', data['star_rati
data['star_rating'] = np.where(data['star_rating'] == '3', 'class2', data['star_rati
data['star_rating'] = np.where(data['star_rating'] == '4', 'class3', data['star_rati
data['star_rating'] = np.where(data['star_rating'] == '5', 'class3', data['star_rati

data.head()
```

Out[8]:

	review_body	star_rating
0	Love this, excellent sun block!!	class3
1	The great thing about this cream is that it do...	class3
2	Great Product, I'm 65 years old and this is al...	class3
3	I use them as shower caps & conditioning caps....	class3
4	This is my go-to daily sunblock. It leaves no ...	class3

In [9]:

```
data_class1 = data[data['star_rating'] == 'class1'].sample(n=20000, replace = False)
data_class2 = data[data['star_rating'] == 'class2'].sample(n=20000, replace = False)
data_class3 = data[data['star_rating'] == 'class3'].sample(n=20000, replace = False)
```

In [10]:

```
df_new = pd.concat([data_class1,data_class2,data_class3])
```

In [11]:

```
df_new
```

Out[11]:

	review_body	star_rating
563295	Pos when you turn it on the tools vibrate out....	class1
2770879	small quantity of each color, not for longwear	class1
2974694	Since when does NEEM oil equal BRAHMI oil?? A...	class1
3662933	Unfortunately the quality was not great. The d...	class1
3885061	What a piece of junk....bought 2 and both are ...	class1
...	...	...
897100	These are the third Rubis needlenose tweezers ...	class3
2122361	My niece has been wanting this for a while, so...	class3
529500	Easy to use.	class3
1331023	I got this product as described. My coworkers ...	class3
4010270	This works great. My son has never been sunbur...	class3

60000 rows × 2 columns

Coverting the classes to 0, 1, 2

In [12]:

```
df_new['star_rating']=df_new['star_rating'].map({'class1':0,'class2':1, 'class3':2})
```

In [13]:

```
df_new
```

Out[13]:

	review_body	star_rating
563295	Pos when you turn it on the tools vibrate out....	0
2770879	small quantity of each color, not for longwear	0
2974694	Since when does NEEM oil equal BRAHMI oil?? A...	0
3662933	Unfortunately the quality was not great. The d...	0
3885061	What a piece of junk....bought 2 and both are ...	0
...	...	...
897100	These are the third Rubis needlenose tweezers ...	2
2122361	My niece has been wanting this for a while, so...	2
529500	Easy to use.	2
1331023	I got this product as described. My coworkers ...	2
4010270	This works great. My son has never been sunbur...	2

60000 rows × 2 columns

In [14]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    df_new['review_body'], df_new['star_rating'], test_size=0.2, random_state=42)
```

WORD EMBEDDINGS

[https://radimrehurek.com/gensim/auto_examples/tutorials/run_word2vec.html#sphx-glr-auto-examples-tutorials-run-word2vec-py (https://radimrehurek.com/gensim/auto_examples/tutorials/run_word2vec.html#sphx-glr-auto-examples-tutorials-run-word2vec-py)](https://radimrehurek.com/gensim/auto_examples/tutorials/run_word2vec.html#sphx-glr-auto-examples-tutorials-run-word2vec-py)

Task 2a

Loading google word2vec model and checking for sematic similarity

In [15]:

```
import gensim.downloader as api
```

In [16]:

```
wv = api.load('word2vec-google-news-300')
# loading the google word2vec dataset
```

Example 1

In [17]:

```
wv.similarity('excellent', 'outstanding')
```

Out[17]:

```
0.55674857
```

Excellent and Outstanding are 55.6% similar

Example 2

In [18]:

```
wv.most_similar(positive=['woman', 'king'], negative=['man'], topn = 5)
```

Out[18]:

```
[('queen', 0.7118193507194519),
 ('monarch', 0.6189674139022827),
 ('princess', 0.5902431011199951),
 ('crown_prince', 0.5499460697174072),
 ('prince', 0.5377321839332581)]
```

The equation King + Woman - Man gives Queen as the Most similar answer with 71.1% similarity

Example 3

In [19]:

```
wv.most_similar('awesome')
```

Out[19]:

```
[('amazing', 0.8282866477966309),
 ('unbelievable', 0.74649578332901),
 ('fantastic', 0.7453290224075317),
 ('incredible', 0.7390913367271423),
 ('unbelieveable', 0.6678116917610168),
 ('terrific', 0.654850423336029),
 ('wonderful', 0.6525596380233765),
 ('great', 0.6510506868362427),
 ('fabulous', 0.6416462659835815),
 ('nice', 0.6404187679290771)]
```

The most similar word to Awesome is Amazing

Example 4

In [20]:

```
wv.similarity('brother', 'boy')
```

Out[20]:

```
0.51953924
```

Brother and Boy are 51.9% similar

Example 5

In [21]:

```
wv.most_similar(positive=['woman', 'boy'], negative=['man'], topn = 5)
```

Out[21]:

```
[('girl', 0.8881361484527588),
 ('teenage_girl', 0.7058953642845154),
 ('mother', 0.6978276968002319),
 ('toddler', 0.6870075464248657),
 ('daughter', 0.6686559915542603)]
```

The equation Woman + Boy - Man gives Girl as the output

Task - 2b

[https://medium.com/@dilip.voleti/classification-using-word2vec-b1d79d375381 (https://medium.com/@dilip.voleti/classification-using-word2vec-b1d79d375381)](https://medium.com/@dilip.voleti/classification-using-word2vec-b1d79d375381)

[https://tedboy.github.io/nlps/generated/generated/gensim.utils.simple_preprocess.html (https://tedboy.github.io/nlps/generated/generated/gensim.utils.simple_preprocess.html)](https://tedboy.github.io/nlps/generated/generated/gensim.utils.simple_preprocess.html)

In [22]:

```
from gensim.utils import simple_preprocess
train_data = []
for i in X_train:
    val = simple_preprocess(i, deacc=True)
    train_data.append(val)
```

In [23]:

```
test_data = []
for i in X_test:
    val = simple_preprocess(i, deacc=True)
    test_data.append(val)
```

In [24]:

```
# train_data
```

Out word2vec model i.e. on our dataset based on the parameters mentioned in the homework

In [25]:

```
from gensim.models import Word2Vec
model = Word2Vec(sentences=train_data, vector_size=300, window=13, min_count=9, work
```

Example 1

In [26]:

```
model.wv.similarity('excellent', 'outstanding')
```

Out[26]:

```
0.7757139
```

Excellent and Outstanding are 77.5% Similar

Example 2

In [27]:

```
model.wv.most_similar('awesome')
```

Out[27]:

```
[('amazing', 0.8166578412055969),
 ('fantastic', 0.7310990691184998),
 ('wonderful', 0.7234351634979248),
 ('excellent', 0.672081053256985),
 ('great', 0.6694231629371643),
 ('terrific', 0.6081065535545349),
 ('delicious', 0.5964312553405762),
 ('perfect', 0.5868383049964905),
 ('fabulous', 0.5829119682312012),
 ('outstanding', 0.5827775597572327)]
```

The most similar word to Awesome is Amazing

Example 3

In [28]:

```
model.wv.similarity('brother', 'boy')
```

Out[28]:

```
0.40999338
```

Example 4

In [29]:

```
model.wv.most_similar(positive=['woman', 'boy'], negative=['man'], topn = 5)
```

Out[29]:

```
[('haired', 0.5567963719367981),
 ('hispanic', 0.5485888719558716),
 ('transitioning', 0.5455800890922546),
 ('female', 0.5215923190116882),
 ('skinned', 0.5153078436851501)]
```

The equation Woman + Boy - Man gives haired as output which is not so good

Some words not present in out training dataset makes it difficult for the model to find most similar word that is expected

The vectors generated by the pretrained google word2vec models ideally perform better as they have a large dataset on which they have been trained on whereas the self-trained model can outperform the pre trained model in the case where there is more training data related to it in the dataset.

The pretrained model seems to encode semantic similarities better as it has been trained on a large dataset

Task - 3 : Simple Models

In [30]:

```
import gensim
```

In [31]:

```
ds = set(wv.index_to_key)
rain_vect = np.array([np.array([wv[i] for i in ls if i in words])for ls in train_data
est_vect = np.array([np.array([wv[i] for i in ls if i in words])for ls in test_data])

reating the train and test vectors based on the word existing in the google word2vec
```

In [32]:

```
y_train = list(y_train)
y_test = list(y_test)
```

In [33]:

```
# calculating the average training vectors

avg_x_train = []
avg_y_train = []
for i in range(0, len(X_train_vect)):
    if len(X_train_vect[i]) > 0:
        avg_x_train.append(list(np.mean(X_train_vect[i], axis=0)))
        avg_y_train.append(y_train[i])
```

In [34]:

```
# avg_x_train
```

In [35]:

```
# avg_y_train
```

In [36]:

```
# calculating the average testing vectors

avg_x_test = []
avg_y_test = []
for i in range(0, len(X_test_vect)):
    if len(X_test_vect[i]) > 0:
        avg_x_test.append(list(np.mean(X_test_vect[i], axis=0)))
        avg_y_test.append(y_test[i])
```

In [37]:

```
# avg_x_test
```

In [38]:

```
# avg_y_test
```

In [39]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(df_new["review_body"])
y = df_new['star_rating']

#Ref: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.t
```

In [40]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

In [41]:

```
from sklearn.linear_model import Perceptron
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

#Word2Vec
model_wv = Perceptron(tol=1e-3, random_state=23)
model_wv.fit(avg_x_train, avg_y_train)

#TFIDF
model_tf = Perceptron(tol=1e-3, random_state=23)
model_tf.fit(X_train, y_train)
```

Out[41]:

```
Perceptron(random_state=23)
```

In [42]:

```
print('Perceptron Word2Vec')
print(model_wv.score(avg_x_train, avg_y_train))

print('Perceptron TFIDF')
print(model_tf.score(X_train, y_train))
```

```
Perceptron Word2Vec
0.592712331052895
Perceptron TFIDF
0.8569791666666666
```

In [43]:

```
y_pred_wv = model_wv.predict(avg_x_test)
y_pred_tf = model_tf.predict(X_test)
```

In [44]:

```
print("Word2Vec")
print(classification_report(avg_y_test, y_pred_wv))
```

```
Word2Vec
              precision    recall  f1-score   support

           0       0.49      0.87      0.63      3959
           1       0.60      0.34      0.43      4012
           2       0.82      0.56      0.67      4010

    accuracy                           0.59     11981
   macro avg       0.64      0.59      0.58     11981
weighted avg       0.64      0.59      0.58     11981
```

In [45]:

```
print("TF-IDF")
print(classification_report(y_test, y_pred_tf))
```

```
TF-IDF
              precision    recall  f1-score   support

           0       0.63      0.67      0.65      4051
           1       0.55      0.52      0.54      3935
           2       0.73      0.73      0.73      4014

    accuracy                           0.64     12000
   macro avg       0.64      0.64      0.64     12000
weighted avg       0.64      0.64      0.64     12000
```

In [46]:

```
print('Perceptron - Word2Vec (Accuracy)')
print(accuracy_score(avg_y_test, y_pred_wv))

print('Perceptron - TF-IDF (Accuracy)')
print(accuracy_score(y_test, y_pred_tf))
```

```
Perceptron - Word2Vec (Accuracy)
0.5879308905767465
Perceptron - TF-IDF (Accuracy)
0.638
```

Perceptron Accuracy (Word2Vec) - 58.79%

Perceptron Accuracy (TF-IDF) - 63.8%

In [47]:

```
from sklearn.svm import LinearSVC
```

In [48]:

```
#Word2Vec
model_wv = LinearSVC(tol=1e-3, random_state=23)
model_wv.fit(avg_x_train, avg_y_train)

#TFIDF
model_tf = LinearSVC(tol=1e-3, random_state=23)
model_tf.fit(X_train, y_train)
```

Out[48]:

```
LinearSVC(random_state=23, tol=0.001)
```

In [49]:

```
print('SVM - Word2Vec (Score)')
print(model_wv.score(avg_x_train, avg_y_train))

print('SVM - TF-IDF (Score)')
print(model_tf.score(X_train, y_train))
```

```
SVM - Word2Vec (Score)
0.6676747872517937
SVM - TF-IDF (Score)
0.8540833333333333
```

In [50]:

```
y_pred_wv = model_wv.predict(avg_x_test)
y_pred_tf = model_tf.predict(X_test)
```

In [51]:

```
print("Word2Vec")
print(classification_report(avg_y_test, y_pred_wv))
```

```
Word2Vec
              precision    recall  f1-score   support

           0       0.66      0.70      0.68      3959
           1       0.60      0.56      0.58      4012
           2       0.73      0.74      0.74      4010

    accuracy                           0.67     11981
   macro avg       0.67      0.67      0.67     11981
weighted avg       0.67      0.67      0.67     11981
```

In [52]:

```
print("TF-IDF")
print(classification_report(y_test, y_pred_tf))
```

```
TF-IDF
              precision    recall  f1-score   support

           0       0.70      0.72      0.71      4051
           1       0.61      0.60      0.61      3935
           2       0.79      0.79      0.79      4014

    accuracy                           0.70     12000
   macro avg       0.70      0.70      0.70     12000
weighted avg       0.70      0.70      0.70     12000
```

In [53]:

```
print('SVM - Word2Vec (Accuracy)')
print(accuracy_score(avg_y_test, y_pred_wv))

print('SVM - TF-IDF (Accuracy)')
print(accuracy_score(y_test, y_pred_tf))
```

```
SVM - Word2Vec (Accuracy)
0.6666388448376597
SVM - TF-IDF (Accuracy)
0.70325
```

SVM Accuracy (Word2Vec) - 66.6%

SVM Accuracy (TF-IDF) -70.32%

We can see that TFIDF performs better than Word2Vec in both perceptron as well as SVM. This is because word2vec is usually used for getting the context of the sentence and is computationally intensive whereas TFIDF works on the works and in case of reviews, only words can suffice to identify the sentiment. Further it means that, word2vec here is a general dataset whereas in TFIDF we have a targeted data.

4 - Feedforward Neural Networks

[https://www.kaggle.com/code/mishra1993/pytorch-multi-layer-perceptron-mnist/notebook (https://www.kaggle.com/code/mishra1993/pytorch-multi-layer-perceptron-mnist/notebook)](https://www.kaggle.com/code/mishra1993/pytorch-multi-layer-perceptron-mnist/notebook)

Task 4a -

In [54]:

```
import torch
import torch.nn as nn
import torch.optim as optim
```

Converting the data to tensors

In [55]:

```
Xtr = torch.tensor(avg_x_train)
```

In [56]:

```
ytr = torch.tensor(avg_y_train)
```

In [57]:

```
Xts = torch.tensor(avg_x_test)
```

In [58]:

```
yts = torch.tensor(avg_y_test)
```

[https://machinelearningmastery.com/building-multilayer-perceptron-models-in-pytorch/ (https://machinelearningmastery.com/building-multilayer-perceptron-models-in-pytorch/)](https://machinelearningmastery.com/building-multilayer-perceptron-models-in-pytorch/)

hidden layers = 2 (100 and 10 nodes)

epochs = 20

Loss - Cross entropy Loss

Non linearity - Relu and softmax

Adam Optimizer

In [63]:

```
class MLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(Xtr.shape[1], 100)
        self.fc2 = nn.Linear(100, 10)
        self.fc3 = nn.Linear(10, 3)
        self.relu = nn.ReLU()
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.fc3(x)
        x = self.softmax(x)
        return x

model = MLP()
```

In [64]:

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

In [65]:

```
for epoch in range(20):
    running_loss = 0.0
    for i in range(0, len(Xtr), 32):
        optimizer.zero_grad()
        outputs = model(Xtr[i:i+32])
        loss = criterion(outputs, ytr[i:i+32])
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

    print('Epoch %d loss: %.3f' % (epoch+1, running_loss/len(Xtr)))
```

```
Epoch 1 loss: 0.029
Epoch 2 loss: 0.028
Epoch 3 loss: 0.027
Epoch 4 loss: 0.027
Epoch 5 loss: 0.027
Epoch 6 loss: 0.027
Epoch 7 loss: 0.027
Epoch 8 loss: 0.027
Epoch 9 loss: 0.027
Epoch 10 loss: 0.027
Epoch 11 loss: 0.027
Epoch 12 loss: 0.027
Epoch 13 loss: 0.027
Epoch 14 loss: 0.027
Epoch 15 loss: 0.027
Epoch 16 loss: 0.027
Epoch 17 loss: 0.027
Epoch 18 loss: 0.026
Epoch 19 loss: 0.026
Epoch 20 loss: 0.026
```

In [66]:

```
with torch.no_grad():
    outputs = model(Xts)
    _, predicted = torch.max(outputs, 1)
    accuracy = (predicted == yts).sum().item() / len(yts)
print('Accuracy: %.2f' % (accuracy*100))
```

```
Accuracy: 66.83
```

Accuracy for MLP (testing split) - 66.83%

Task 4b

In [67]:

```
#concatenating the first 10 words to data_tr

data_tr = []
for review in df_new['review_body'].tolist():
    #print(review)
    data_temp = []
    for i, word in enumerate(review.split()):
        #print(i, word)
        if i<10 and word in words:
            data_temp.append(wv[word])
    if len(data_temp) < 10:
        while len(data_temp) != 10:
            data_temp.append(np.zeros(shape=(300,)))
    data_tr.append(data_temp)
```

In [68]:

```
X = torch.tensor(data_tr)
```

In [69]:

```
y = torch.tensor(df_new['star_rating'].tolist())
```

In [70]:

```
# Define the MLP model
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(3000, 100)
        self.fc2 = nn.Linear(100, 10)
        self.fc3 = nn.Linear(10, 3)
        self.relu = nn.ReLU()
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        out = self.relu(out)
        out = self.fc3(out)
        out = self.softmax(out)
        return out

# Create an instance of the MLP model
model = MLP()

# Define the loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

In [71]:

```
X = torch.tensor(X, dtype=torch.float32)
y = torch.tensor(y, dtype=torch.long)
```

In [72]:

```
X = X.view((60000, 3000))
```

In [73]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat
```

In [78]:

```
# Train the MLP model
num_epochs = 20
batch_size = 32
for epoch in range(num_epochs):
    for i in range(0, len(X_train), batch_size):
        # Get the batch data
        batch_X = X_train[i:i+batch_size]
        batch_y = y_train[i:i+batch_size]

        # Zero the gradients
        optimizer.zero_grad()

        # Forward pass
        outputs = model(batch_X)
        loss = criterion(outputs, batch_y)

        # Backward pass and optimization
        loss.backward()
        optimizer.step()
```

In [79]:

```
outputs.shape
```

Out[79]:

```
torch.Size([32, 3])
```

In [80]:

```
y_train.dtype
```

Out[80]:

```
torch.int64
```

In [81]:

```
with torch.no_grad():
    outputs = model(X_test)
    _, predicted = torch.max(outputs.data, 1)
    total = y_test.size(0)
    correct = (predicted == y_test).sum().item()
    accuracy = 100 * correct / total
    print(f"Epoch {epoch+1}, accuracy: {accuracy:.2f}%")
```

```
Epoch 20, accuracy: 54.74%
```

Accuracy for MLP (concatenation) - 54.74%

The accuracy is better as compared to the simple models because it trains a neural network and takes context into conideration. This helps to improve the accuracy. The one without 10 vectors gives better accuracy as it considers all the words rather than only 10

Task 5

[https://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html (https://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html)](https://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html)

epochs = 10

Loss - Cross entropy Loss

Adam Optimizer

lr = 0.001

In [82]:

```
from torch.utils.data import Dataset, TensorDataset, DataLoader
```

In [83]:

```
import torch.nn as nn

class RNNModel(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(RNNModel, self).__init__()
        self.hidden_dim = hidden_dim
        self.rnn = nn.RNN(input_dim, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        h0 = torch.zeros(1, x.size(0), self.hidden_dim).to(x.device)
        out, hidden = self.rnn(x, h0)
        out = out[:, -1, :]
        out = self.fc(out)
        return out
```

In [84]:

```
device = torch.device('cpu')
model = RNNModel(6000, 20, 3).to(device)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

In [85]:

```
#creating a list of reviews and their word vec concatenated until 20 words
data_tr_rnn = []
for review in df_new['review_body'].tolist():
    #print(review)
    data_temp_rnn = []
    for i, word in enumerate(review.split()):
        #print(i, word)
        if i<20 and word in words:
            data_temp_rnn.append(wv[word])
    if len(data_temp_rnn) < 20:
        while len(data_temp_rnn) != 20:
            data_temp_rnn.append(np.zeros(shape=(300,)))
    data_tr_rnn.append(np.array(data_temp_rnn, dtype=np.float32).flatten())
```

In [86]:

```
rain, X_test, y_train, y_test = train_test_split(data_tr_rnn, df_new['star_rating'].
```

In [87]:

```
trainX_tn = torch.from_numpy(np.array(X_train))
testX_tn = torch.from_numpy(np.array(X_test))
```

In [88]:

```
trainY_tn = torch.from_numpy(np.array(y_train))
testY_tn = torch.from_numpy(np.array(y_test))
```

In [89]:

```
train_dataset_tn = TensorDataset(trainX_tn, trainY_tn)
test_dataset_tn = TensorDataset(testX_tn, testY_tn)
```

In [90]:

```
train_loader = DataLoader(train_dataset_tn, batch_size=32)
test_loader = DataLoader(test_dataset_tn, batch_size=32)
```

In [106]:

```
num_epochs = 10
batch_size = 32
# Train the model
n_total_steps = len(train_loader)
for epoch in range(num_epochs):
    for i, (data, labels) in enumerate(train_loader):
        data = data.reshape(batch_size, -1, 6000)
        # Forward pass
        outputs = model(data)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if (i+1) % 750 == 0:
            print (f'Epoch [{epoch+1}/{num_epochs}], Step [{i+1}/{n_total_steps}], I
```

```
Epoch [1/10], Step [750/1500], Loss: 0.9000
Epoch [1/10], Step [1500/1500], Loss: 0.7904
Epoch [2/10], Step [750/1500], Loss: 0.7196
Epoch [2/10], Step [1500/1500], Loss: 0.6597
Epoch [3/10], Step [750/1500], Loss: 0.5783
Epoch [3/10], Step [1500/1500], Loss: 0.6027
Epoch [4/10], Step [750/1500], Loss: 0.4388
Epoch [4/10], Step [1500/1500], Loss: 0.5562
Epoch [5/10], Step [750/1500], Loss: 0.3315
Epoch [5/10], Step [1500/1500], Loss: 0.5119
Epoch [6/10], Step [750/1500], Loss: 0.2427
Epoch [6/10], Step [1500/1500], Loss: 0.4162
Epoch [7/10], Step [750/1500], Loss: 0.1509
Epoch [7/10], Step [1500/1500], Loss: 0.2818
Epoch [8/10], Step [750/1500], Loss: 0.1091
Epoch [8/10], Step [1500/1500], Loss: 0.1320
Epoch [9/10], Step [750/1500], Loss: 0.0860
Epoch [9/10], Step [1500/1500], Loss: 0.0889
Epoch [10/10], Step [750/1500], Loss: 0.0639
Epoch [10/10], Step [1500/1500], Loss: 0.0595
```

In [107]:

```
with torch.no_grad():
    n_correct = 0
    n_samples = 0
    for data, labels in test_loader:
        data = data.reshape(batch_size, -1, 6000)
        outputs = model(data)
        _, predicted = torch.max(outputs.data, 1)
        n_samples += labels.size(0)
        n_correct += (predicted == labels).sum().item()

    acc = 100.0 * n_correct / n_samples
    print(f'Accuracy : {acc} %')
```

```
Accuracy : 52.65 %
```

RNN Accuracy - 52.65%

FNN with avg vectors mostly performs better than simple RNN

In [108]:

```
class GRUModel(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(GRUModel, self).__init__()
        self.hidden_dim = hidden_dim
        self.gru = nn.GRU(input_dim, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        h0 = torch.zeros(1, x.size(0), self.hidden_dim).to(x.device)
        out, hidden = self.gru(x, h0)
        out = out[:, -1, :]
        out = self.fc(out)
        return out
```

In [109]:

```
model = GRUModel(6000, 20, 3).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

In [111]:

```
num_epochs = 10
batch_size = 32
# Train the model
n_total_steps = len(train_loader)
for epoch in range(num_epochs):
    for i, (data, labels) in enumerate(train_loader):
        data = data.reshape(batch_size, -1, 6000)
        # Forward pass
        outputs = model(data)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if (i+1) % 750 == 0:
            print (f'Epoch [{epoch+1}/{num_epochs}], Step [{i+1}/{n_total_steps}], I
```

```
Epoch [1/10], Step [750/1500], Loss: 0.7281
Epoch [1/10], Step [1500/1500], Loss: 0.6550
Epoch [2/10], Step [750/1500], Loss: 0.6033
Epoch [2/10], Step [1500/1500], Loss: 0.5467
Epoch [3/10], Step [750/1500], Loss: 0.5184
Epoch [3/10], Step [1500/1500], Loss: 0.4596
Epoch [4/10], Step [750/1500], Loss: 0.4197
Epoch [4/10], Step [1500/1500], Loss: 0.3891
Epoch [5/10], Step [750/1500], Loss: 0.2989
Epoch [5/10], Step [1500/1500], Loss: 0.2902
Epoch [6/10], Step [750/1500], Loss: 0.2075
Epoch [6/10], Step [1500/1500], Loss: 0.1958
Epoch [7/10], Step [750/1500], Loss: 0.1576
Epoch [7/10], Step [1500/1500], Loss: 0.1135
Epoch [8/10], Step [750/1500], Loss: 0.1366
Epoch [8/10], Step [1500/1500], Loss: 0.1021
Epoch [9/10], Step [750/1500], Loss: 0.1333
Epoch [9/10], Step [1500/1500], Loss: 0.0352
Epoch [10/10], Step [750/1500], Loss: 0.0801
Epoch [10/10], Step [1500/1500], Loss: 0.0441
```

In [112]:

```
with torch.no_grad():
    n_correct = 0
    n_samples = 0
    for data, labels in test_loader:
#         images = images.reshape(-1, sequence_length, input_size).to(device)
#         labels = labels.to(device)
        data = data.reshape(batch_size, -1, 6000)
        outputs = model(data)
        # max returns (value ,index)
        _, predicted = torch.max(outputs.data, 1)
        n_samples += labels.size(0)
        n_correct += (predicted == labels).sum().item()

    acc = 100.0 * n_correct / n_samples
    print(f'Accuracy : {acc} %')
```

```
Accuracy : 52.09166666666667 %
```

GRU accuracy == 52.092%

FNN with avg vectors mostly performs better than GRU

In [113]:

```
class LSTMModel(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(LSTMModel, self).__init__()
        self.hidden_dim = hidden_dim
        self.lstm = nn.LSTM(input_dim, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        h0 = torch.zeros(1, x.size(0), self.hidden_dim).to(x.device)
        c0 = torch.zeros(1, x.size(0), self.hidden_dim).to(device)
        out, hidden = self.lstm(x, (h0,c0))
        out = out[:, -1, :]
        out = self.fc(out)
        return out
```

In [114]:

```
model = LSTMModel(6000, 20, 3).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

In [115]:

```
num_epochs = 10
batch_size = 32
# Train the model
n_total_steps = len(train_loader)
for epoch in range(num_epochs):
    for i, (data, labels) in enumerate(train_loader):
        data = data.reshape(batch_size, -1, 6000)
        # Forward pass
        outputs = model(data)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if (i+1) % 800 == 0:
            print (f'Epoch [{epoch+1}/{num_epochs}], Step [{i+1}/{n_total_steps}], I
```

```
Epoch [1/10], Step [800/1500], Loss: 0.9832
Epoch [2/10], Step [800/1500], Loss: 0.7546
Epoch [3/10], Step [800/1500], Loss: 0.5776
Epoch [4/10], Step [800/1500], Loss: 0.4471
Epoch [5/10], Step [800/1500], Loss: 0.3370
Epoch [6/10], Step [800/1500], Loss: 0.2400
Epoch [7/10], Step [800/1500], Loss: 0.1750
Epoch [8/10], Step [800/1500], Loss: 0.1185
Epoch [9/10], Step [800/1500], Loss: 0.0954
Epoch [10/10], Step [800/1500], Loss: 0.0756
```

In [116]:

```
with torch.no_grad():
    n_correct = 0
    n_samples = 0
    for data, labels in test_loader:
#         images = images.reshape(-1, sequence_length, input_size).to(device)
#         labels = labels.to(device)
        data = data.reshape(batch_size, -1, 6000)
        outputs = model(data)
        # max returns (value ,index)
        _, predicted = torch.max(outputs.data, 1)
        n_samples += labels.size(0)
        n_correct += (predicted == labels).sum().item()

    acc = 100.0 * n_correct / n_samples
    print(f'Accuracy : {acc} %')
```

```
Accuracy : 52.94166666666667 %
```

LSTM Accuracy - 52.94%

FNN with avg vectors mostly performs better than LSTM

The accuracies of GRU and LSTM is better than simple RNN.