Samhitha Gundam

HW 2 -

Question 1,2,3,4 are present in this notebook and Q5 is present in the second Notebook

In [1]:
```
!pip install torch
```

```
Requirement already satisfied: torch in /Users/samhitha/opt/anaconda3/lib
/python3.9/site-packages (1.12.1)
Requirement already satisfied: typing-extensions in /Users/samhitha/opt/a
naconda3/lib/python3.9/site-packages (from torch) (4.1.1)
```

In [2]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
import gensim.models
import gensim.downloader as api
import nltk
import numpy as np
nltk.download('punkt')
import torch
import torch.nn as nn
from torch.utils.data.sampler import SubsetRandomSampler
```

```
[nltk_data] Downloading package punkt to /Users/samhitha/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Question 1

In [3]:
```
raw_data=pd.read_csv("amazon_reviews_us_Jewelry_v1_00.tsv",sep="\t",on_ba
data=raw_data[['star_rating','review_body']]
data=data.dropna()
data = data.reset_index(drop=True)
data['star_rating']=data['star_rating'].astype(int)
```

```
/var/folders/3c/dvx5c3n50kx5fh6qcgyy221w0000gn/T/ipykernel_3452/405122424
0.py:1: DtypeWarning: Columns (7) have mixed types. Specify dtype option
on import or set low_memory=False.
  raw_data=pd.read_csv("amazon_reviews_us_Jewelry_v1_00.tsv",sep="\t",on_
bad_lines='skip')
```

In [4]:
```
data
```

Out[4]:

	star_rating	review_body
0	5	so beautiful even tho clearly not high end .....
1	5	Great product.. I got this set for my mother, ...
2	5	Exactly as pictured and my daughter's friend l...
3	5	Love it. Fits great. Super comfortable and nea...
4	5	Got this as a Mother's Day gift for my Mom and...
...	...	...
1766743	4	It is nice looking and everything (it is sterl...
1766744	4	my boyfriend bought me this last christmas, an...
1766745	4	This is a great way to quickly start learning ...
1766746	5	the 14kt gold earrings look remarkable...would...
1766747	5	It will be a gift to my special friend. We kno...

1766748 rows × 2 columns

In [4]:
```
df=data.groupby('star_rating').sample(n=20000)
```

In [5]:
```
df['review_body']=df['review_body'].str.lower()  #convert to lower case

#remove contractions
df['review_body']=df['review_body'].str.replace("\'re"," are")
df['review_body']=df['review_body'].str.replace("br"," ")
df['review_body']=df['review_body'].str.replace("\n't"," not")
df['review_body']=df['review_body'].str.replace("\'s'"," is")
df['review_body']=df['review_body'].str.replace("i'm"," i am")
df['review_body']=df['review_body'].str.replace("\'ve'"," have")
df['review_body']=df['review_body'].str.replace("din't","did not")

df['review_body']=df['review_body'].str.replace('http\S+|www.\S+', '', ca
df['review_body']=df['review_body'].str.replace('[^a-zA-Z0-9 ]', '')
df['review_body']=df['review_body'].str.replace('/ +/', ' ')#convert mult
df['review_body']=df['review_body'].str.replace('/^ /', '') # remove spac
df['review_body']=df['review_body'].str.replace('/ $/', '') # remove unne
```

```
/var/folders/3c/dvx5c3n50kx5fh6qcgyy221w0000gn/T/ipykernel_3452/77415047.
py:12: FutureWarning: The default value of regex will change from True to
False in a future version.
  df['review_body']=df['review_body'].str.replace('http\S+|www.\S+', '',
case=False)
/var/folders/3c/dvx5c3n50kx5fh6qcgyy221w0000gn/T/ipykernel_3452/77415047.
py:13: FutureWarning: The default value of regex will change from True to
False in a future version.
  df['review_body']=df['review_body'].str.replace('[^a-zA-Z0-9 ]', '')
/var/folders/3c/dvx5c3n50kx5fh6qcgyy221w0000gn/T/ipykernel_3452/77415047.
py:14: FutureWarning: The default value of regex will change from True to
False in a future version.
  df['review_body']=df['review_body'].str.replace('/ +/', ' ')#convert mu
ltispace to space
/var/folders/3c/dvx5c3n50kx5fh6qcgyy221w0000gn/T/ipykernel_3452/77415047.
py:15: FutureWarning: The default value of regex will change from True to
False in a future version.
  df['review_body']=df['review_body'].str.replace('/^ /', '') # remove sp
aces in the start of the string
/var/folders/3c/dvx5c3n50kx5fh6qcgyy221w0000gn/T/ipykernel_3452/77415047.
py:16: FutureWarning: The default value of regex will change from True to
False in a future version.
  df['review_body']=df['review_body'].str.replace('/ $/', '') # remove un
necesary space at the end of the string
```

Question 2(a) Implementation of word2Vec using google new data set

In [6]:
```
wv = api.load('word2vec-google-news-300')
```

In [7]:
```
wv.most_similar(positive = ['king','woman'], negative =['man'],topn=3)
```

Out[7]:
```
[('queen', 0.7118193507194519),
 ('monarch', 0.6189674735069275),
 ('princess', 0.5902431011199951)]
```

In [9]:
```
wv.most_similar(positive = ['father','woman'], negative =['man'],topn=3)
```

Out[9]:
```
[('mother', 0.8462508320808411),
 ('daughter', 0.7899606227874756),
 ('husband', 0.7560456991195679)]
```

In [10]:
```
print('excellent', 'outstanding',wv.similarity('excellent','outstanding')
```

```
excellent outstanding 0.5567486
```

In [12]:
```
print('good', 'great',wv.similarity('good','great'))
print('worst','bad',wv.similarity('worst','bad'))
print('car', 'truck',wv.similarity('car','truck'))
```

```
good great 0.72915095
worst bad 0.43674564
car truck 0.67357904
```

Question 2b - Word2Vec using review dataset

In [13]:
```
X=df['review_body']
Y=df['star_rating']
```

In [14]:
```
inputs=X.to_list()
sentences=[k.split() for k in inputs]
```

In [15]:
```
model = gensim.models.Word2Vec(sentences=sentences,vector_size=300, windo
```

In [16]:
```
model.wv.most_similar(positive = ['king','woman'], negative =['man'],topn
```

Out[16]:
```
[('avenue', 0.5949714779853821),
 ('candy', 0.5309351086616516),
 ('amazoncom', 0.5016636252403259)]
```

In [19]:
```
model.wv.most_similar(positive = ['father','woman'], negative =['man'],to
```

Out[19]:
```
[('grandchildren', 0.6483133435249329),
 ('lady', 0.6390446424484253),
 ('cousin', 0.6286104321479797)]
```

In [20]:
```
print('excellent', 'outstanding',model.wv.similarity('excellent','outstan
print('good', 'great',model.wv.similarity('good','great'))
print('worst','bad',model.wv.similarity('good','great'))
```

```
excellent outstanding 0.70434016
good great 0.83079803
worst bad 0.83079803
```

Using the above values of similarity we can state that pretrained google model is better for terms like king, queen but for case of adjectives like good, bad, excellent which are greatly a part of the review dataset, our word2vec model performs better as there are greater examples with such used thus allowing the model to be trained better

Question 3

In [21]:
```
X=df['review_body']
Y=df['star_rating']
```

In [22]:
```
X=pd.DataFrame(X)
sentences=[k.split() for k in X['review_body']]
X['reviews_split']=sentences
```

In [23]:
```
X
```

Out[23]:

	review_body	reviews_split
1087812	this tarnished it is not sterling silver and t...	[this, tarnished, it, is, not, sterling, silve...
220841	got one for the most beautiful woman ive ever ...	[got, one, for, the, most, beautiful, woman, i...
1271914	when i received this ring i was so disappointe...	[when, i, received, this, ring, i, was, so, di...
387606	pretty but damage	[pretty, but, damage]
349019	the enamel on the ring scratches very easily a...	[the, enamel, on, the, ring, scratches, very, ...
...	...	...
1531270	i do not wear silver and very few 34dangle34 e...	[i, do, not, wear, silver, and, very, few, 34d...
176086	100 percent satisfied excellent product and ha...	[100, percent, satisfied, excellent, product, ...
1381040	the picture doesnt do this beautiful acelet j...	[the, picture, doesnt, do, this, beautiful, ac...
468354	this was a gift for my wife and she loved it	[this, was, a, gift, for, my, wife, and, she, ...
1008475	i ordered this ring for my daughter and she ab...	[i, ordered, this, ring, for, my, daughter, an...

100000 rows × 2 columns

In [19]:
```
features=[]
for lis in X['reviews_split']:
    vector=np.zeros(300,)
    count=len(lis)
    for j in lis:
        if j in wv.key_to_index:
            vector=vector+wv[j]
    features.append(vector/count)

X['input']=features
```

```
/var/folders/3c/dvx5c3n50kx5fh6qcgyy221w0000gn/T/ipykernel_71284/2766180496.py:8: RuntimeWarning: invalid value encountered in true_divide
  features.append(vector/count)
```

In [20]:
```
df3 = X.input.apply(pd.Series)
df3=df3.replace(np.nan,0)
X_train,X_test,Y_train, Y_test = train_test_split(df3,Y, test_size=0.2, r
```

Perceptron model - accuracy obtained with TF-IDF features - 41%; Word2Vec - 33% . In case of perceptron model, the TF-IDF features perform better than word2Vec features but not by much

In [21]:
```
from sklearn.linear_model import Perceptron
clf_perceptron = Perceptron()
clf_perceptron.fit(X_train, Y_train)
```

Out[21]:
```
Perceptron()
```

In [22]:
```
y_test_perceptron=clf_perceptron.predict(X_test)
```

In [23]:
```
from sklearn.metrics import classification_report
report=classification_report(Y_test, y_test_perceptron,output_dict=True)
```

In [24]:
```
report
```

Out[24]:
```
{'1': {'precision': 0.614221916867056,
  'recall': 0.4452191235059761,
  'f1-score': 0.5162407968817669,
  'support': 4016},
 '2': {'precision': 0.37142857142857144,
  'recall': 0.07454500124657193,
  'f1-score': 0.12416943521594685,
  'support': 4011},
 '3': {'precision': 0.31971153846153844,
  'recall': 0.09922904750062174,
  'f1-score': 0.1514518884038717,
  'support': 4021},
 '4': {'precision': 0.25334829254610475,
  'recall': 0.8992035838725734,
  'f1-score': 0.39531703047212646,
  'support': 4018},
 '5': {'precision': 0.7212903225806452,
  'recall': 0.14209456024402645,
  'f1-score': 0.23741771076661713,
  'support': 3934},
 'accuracy': 0.3329,
 'macro avg': {'precision': 0.4560001283767832,
  'recall': 0.3320582632739539,
  'f1-score': 0.2849193723480658,
  'support': 20000},
 'weighted avg': {'precision': 0.45487924413872244,
  'recall': 0.3329,
  'f1-score': 0.2851319895396592,
  'support': 20000}}
```

SVM model - accuracy obtained with TF-IDF features - 49%; Word2Vec - 48%. Similar to case of perceptron model, In case of SVM model too, the TF-IDF features perform better than word2Vec features but by a very marginal difference .

In [25]:
```
from sklearn.svm import LinearSVC
clf_SVM = LinearSVC()
clf_SVM.fit(X_train, Y_train)
y_test_SVM=clf_SVM.predict(X_test)
report_SVM=classification_report(Y_test, y_test_SVM,output_dict=True)
```

In [26]:
```
report_SVM
```

Out[26]:

```
{'1': {'precision': 0.5076977526101575,
  'recall': 0.7143924302788844,
  'f1-score': 0.5935657391124445,
  'support': 4016},
 '2': {'precision': 0.38315318673127097,
  'recall': 0.256295188232361,
  'f1-score': 0.30714072303555423,
  'support': 4011},
 '3': {'precision': 0.40065952184666115,
  'recall': 0.36259636906242226,
  'f1-score': 0.3806788511749347,
  'support': 4021},
 '4': {'precision': 0.43756558237145854,
  'recall': 0.3113489298158288,
  'f1-score': 0.36382143376472303,
  'support': 4018},
 '5': {'precision': 0.5688854489164087,
  'recall': 0.7473309608540926,
  'f1-score': 0.6460118655240605,
  'support': 3934},
 'accuracy': 0.4773,
 'macro avg': {'precision': 0.45959229849519134,
  'recall': 0.4783927756487178,
  'f1-score': 0.4582437225223434,
  'support': 20000},
 'weighted avg': {'precision': 0.45914637049063084,
  'recall': 0.4773,
  'f1-score': 0.4574828154391955,
  'support': 20000}}
```


Question 4a

In [27]:
```
#Creating tensor data
#training data
xtrain_np=X_train.to_numpy()
x=torch.from_numpy(xtrain_np)

ytrain_np=Y_train.to_numpy()
y=torch.from_numpy(ytrain_np-1)

#testing data
xtest_np=X_test.to_numpy()
test_x=torch.from_numpy(xtest_np)

ytest_np=Y_test.to_numpy()
test_y=torch.from_numpy(ytest_np-1)
```

In [37]:
```
model=torch.nn.Sequential(
    torch.nn.Linear(300,50),
    torch.nn.ReLU(),
    torch.nn.Linear(50,10),
    torch.nn.ReLU(),
    torch.nn.Linear(10,5)
)
criterion = nn.CrossEntropyLoss()
learning_rate = 0.2
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

In [38]:
```
model=model.float()
for t in range(1000):
    y_pred = model(x.float())
    loss = criterion(y_pred, y)

    optimizer.zero_grad()
    loss.backward()

    optimizer.step()

    if(t%50==0):
        pred=model(x.float())
        _,predicted = torch.max(pred.data,1)
        correct = (predicted == y).sum()
        print(" traning Accuracy for",t,"-",correct/len(y))
```

```
 traning Accuracy for 0 - tensor(0.2008)
 traning Accuracy for 50 - tensor(0.2024)
 traning Accuracy for 100 - tensor(0.2779)
 traning Accuracy for 150 - tensor(0.3130)
 traning Accuracy for 200 - tensor(0.3367)
 traning Accuracy for 250 - tensor(0.3516)
 traning Accuracy for 300 - tensor(0.3626)
 traning Accuracy for 350 - tensor(0.3648)
 traning Accuracy for 400 - tensor(0.3715)
 traning Accuracy for 450 - tensor(0.3758)
 traning Accuracy for 500 - tensor(0.3847)
 traning Accuracy for 550 - tensor(0.3401)
 traning Accuracy for 600 - tensor(0.3665)
 traning Accuracy for 650 - tensor(0.3733)
 traning Accuracy for 700 - tensor(0.3796)
 traning Accuracy for 750 - tensor(0.3848)
 traning Accuracy for 800 - tensor(0.3900)
 traning Accuracy for 850 - tensor(0.3947)
 traning Accuracy for 900 - tensor(0.3992)
 traning Accuracy for 950 - tensor(0.4046)
```

In [39]:
```
pred=model(test_x.float())
_,predicted = torch.max(pred.data,1)
correct = (predicted == test_y).sum()
print("Accuracy for question 4a:",correct/len(test_y))
```

```
Accuracy for question 4a: tensor(0.3986)
```

Question 4b

In [24]:
```
#Creating feature vector

features=[]
for lis in X['reviews_split']:
    wordvecs=[]
    vector=np.zeros(300,)
    for j in range(len(lis)):
        if(j>=10):
            break
        else:
            if lis[j] in wv.key_to_index:
                wordvecs.append(wv[lis[j]])
    while len(wordvecs)<10:
        wordvecs.append(np.zeros(300,))

    features.append(wordvecs)

X['input_2']=features
```

In [25]:
```
df4 = X.input_2.apply(pd.Series)
df=pd.DataFrame()
for column in df4.columns:
    temp=df4.iloc[:,column].apply(pd.Series)
    df=pd.concat([df,temp],axis=1)
df
```

Out[25]:

	0	1	2	3	4	5	6
1087812	0.109375	0.140625	-0.031738	0.166016	-0.071289	0.015869	-0.003113
220841	0.062012	0.108398	-0.096680	0.079102	0.033936	-0.347656	-0.069824
1271914	0.170898	0.024292	0.138672	0.022217	0.068848	-0.090820	-0.031738
387606	0.123047	-0.046143	-0.202148	0.144531	-0.027100	-0.037598	-0.103027
349019	0.080078	0.104980	0.049805	0.053467	-0.067383	-0.120605	0.035156
...	...	...	...	...	...	...	...
1531270	-0.225586	-0.019531	0.090820	0.237305	-0.029297	0.093262	-0.058838
176086	0.081055	-0.235352	-0.045898	-0.036377	-0.063477	-0.130859	0.030396
1381040	0.080078	0.104980	0.049805	0.053467	-0.067383	-0.120605	0.035156
468354	0.109375	0.140625	-0.031738	0.166016	-0.071289	0.015869	-0.003113
1008475	-0.225586	-0.019531	0.090820	0.237305	-0.029297	0.093262	-0.058838

100000 rows × 3000 columns

In [26]:
```
X_train,X_test,Y_train, Y_test = train_test_split(df,Y, test_size=0.2, ra
```

In [27]:
```
#training data
xtrain_np=X_train.to_numpy()
x=torch.from_numpy(xtrain_np)

ytrain_np=Y_train.to_numpy()
y=torch.from_numpy(ytrain_np-1)

#testing data
xtest_np=X_test.to_numpy()
test_x=torch.from_numpy(xtest_np)

ytest_np=Y_test.to_numpy()
test_y=torch.from_numpy(ytest_np-1)
```

In [29]:
```
model=torch.nn.Sequential(
    torch.nn.Linear(3000,50),
    torch.nn.ReLU(),
    torch.nn.Linear(50,10),
    torch.nn.ReLU(),
    torch.nn.Linear(10,5)
)
```

In [32]:
```
criterion = nn.CrossEntropyLoss()
learning_rate = 0.1
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

In [33]:
```
model=model.float()
for t in range(150):
    y_pred=model(x.float())
    loss = criterion(y_pred, y)

    optimizer.zero_grad()
    loss.backward()

    optimizer.step()

    if(t%10==0):
        pred=model(x.float())
        _,predicted = torch.max(pred.data,1)
        correct = (predicted == y).sum()
        print(" train Accuracy for",t,"-",correct/len(y))
```

```
train Accuracy for 0 - tensor(0.2102)
train Accuracy for 10 - tensor(0.2211)
train Accuracy for 20 - tensor(0.2332)
train Accuracy for 30 - tensor(0.2463)
train Accuracy for 40 - tensor(0.2625)
train Accuracy for 50 - tensor(0.2799)
train Accuracy for 60 - tensor(0.2923)
train Accuracy for 70 - tensor(0.3007)
train Accuracy for 80 - tensor(0.3083)
train Accuracy for 90 - tensor(0.3137)
train Accuracy for 100 - tensor(0.3202)
train Accuracy for 110 - tensor(0.3245)
train Accuracy for 120 - tensor(0.3263)
train Accuracy for 130 - tensor(0.3288)
train Accuracy for 140 - tensor(0.3307)
```

In [34]:
```
pred=model(test_x.float())
```

In [35]:
```
_,predicted = torch.max(pred.data,1)
correct = (predicted == test_y).sum()
print("Test Accuracy - 4b:",correct/len(test_y))
```

```
Test Accuracy - 4b: tensor(0.3249)
```

Samhitha Gundam

HW 2 - part 2

Question 5a, 5b present in this notebook and Q 1,2,3,4 is present in the other Notebook

In [2]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
import gensim.models
import gensim.downloader as api
import nltk
import numpy as np
nltk.download('punkt')
import torch
import torch.nn as nn
from torch.utils.data.sampler import SubsetRandomSampler
```

```
[nltk_data] Downloading package punkt to /Users/samhitha/nltk_data
...
[nltk_data]   Package punkt is already up-to-date!
```

In [3]:
```
raw_data=pd.read_csv("amazon_reviews_us_Jewelry_v1_00.tsv",sep="\t"
data=raw_data[['star_rating','review_body']]
data=data.dropna()
data = data.reset_index(drop=True)
data['star_rating']=data['star_rating'].astype(int)
```

```
/var/folders/3c/dvx5c3n50kx5fh6qcgyy221w0000gn/T/ipykernel_3250/40
51224240.py:1: DtypeWarning: Columns (7) have mixed types. Specify
dtype option on import or set low_memory=False.
  raw_data=pd.read_csv("amazon_reviews_us_Jewelry_v1_00.tsv",sep="
\t",on_bad_lines='skip')
```

In [4]:
```
df=data.groupby('star_rating').sample(n=20000)

df['review_body']=df['review_body'].str.lower()  #convert to lower

#remove contractions
df['review_body']=df['review_body'].str.replace("\'re"," are")
df['review_body']=df['review_body'].str.replace("br"," ")
df['review_body']=df['review_body'].str.replace("\n't"," not")
df['review_body']=df['review_body'].str.replace("\'s'"," is")
df['review_body']=df['review_body'].str.replace("i'm"," i am")
df['review_body']=df['review_body'].str.replace("\'ve'"," have")
df['review_body']=df['review_body'].str.replace("din't","did not")

df['review_body']=df['review_body'].str.replace('http\S+|www.\S+',
df['review_body']=df['review_body'].str.replace('[^a-zA-Z0-9 ]', ''
df['review_body']=df['review_body'].str.replace('/ +/', ' ')#conver
df['review_body']=df['review_body'].str.replace('/^ /', '') # remov
df['review_body']=df['review_body'].str.replace('/ $/', '') # remov
```

```
/var/folders/3c/dvx5c3n50kx5fh6qcgyy221w0000gn/T/ipykernel_3250/34
3968516.py:14: FutureWarning: The default value of regex will chan
ge from True to False in a future version.
  df['review_body']=df['review_body'].str.replace('http\S+|www.\S+
', '', case=False)
/var/folders/3c/dvx5c3n50kx5fh6qcgyy221w0000gn/T/ipykernel_3250/34
3968516.py:15: FutureWarning: The default value of regex will chan
ge from True to False in a future version.
  df['review_body']=df['review_body'].str.replace('[^a-zA-Z0-9 ]',
'')
/var/folders/3c/dvx5c3n50kx5fh6qcgyy221w0000gn/T/ipykernel_3250/34
3968516.py:16: FutureWarning: The default value of regex will chan
ge from True to False in a future version.
  df['review_body']=df['review_body'].str.replace('/ +/', ' ')#con
vert multispace to space
/var/folders/3c/dvx5c3n50kx5fh6qcgyy221w0000gn/T/ipykernel_3250/34
3968516.py:17: FutureWarning: The default value of regex will chan
ge from True to False in a future version.
  df['review_body']=df['review_body'].str.replace('/^ /', '') # re
move spaces in the start of the string
/var/folders/3c/dvx5c3n50kx5fh6qcgyy221w0000gn/T/ipykernel_3250/34
3968516.py:18: FutureWarning: The default value of regex will chan
ge from True to False in a future version.
  df['review_body']=df['review_body'].str.replace('/ $/', '') # re
move unnecesary space at the end of the string
```

In [5]:
```
wv = api.load('word2vec-google-news-300')
```

In [6]:
```
X=df['review_body']
Y=df['star_rating']

X=pd.DataFrame(X)
sentences=[k.split() for k in X['review_body']]
X['reviews_split']=sentences
```

In [7]:
```
features=[]
for lis in X['reviews_split']:
    wordvecs=[]
    vector=np.zeros(300,)
    for j in range(len(lis)):
        if(j>=20):
            break
        else:
            if lis[j] in wv.key_to_index:
                wordvecs.append(wv[lis[j]])
    while len(wordvecs)<20:
        wordvecs.append(np.zeros(300,))

    features.append(wordvecs)

X['input_3']=features
```

In [8]:
```
X
```

Out[8]:

	review_body	reviews_split	input_3
911218	made carelessly the second i put it on my wris...	[made, carelessly, the, second, i, put, it, on...	[[-0.055908203, 0.11767578, 0.2109375, 0.00836...
668073	the pair they gave me looks nothing like the o...	[the, pair, they, gave, me, looks, nothing, li...	[[0.080078125, 0.10498047, 0.049804688, 0.0534...
680243	color peels off i wished i would have read the...	[color, peels, off, i, wished, i, would, have,...	[[-0.0043029785, 0.14355469, 0.036376953, 0.12...
317872	elastic string doesnt work fluent afraid of e...	[elastic, string, doesnt, work, fluent, afraid...	[[0.17285156, -0.084472656, -0.29101562, 0.185...
1048382	very upset the ring when i got it has black st...	[very, upset, the, ring, when, i, got, it, has...	[[0.016601562, 0.045654297, -0.119140625, 0.06...
...	...	...	...
1197729	ive always hated wearing necklaces because the...	[ive, always, hated, wearing, necklaces, becau...	[[-0.41210938, 0.18847656, -0.234375, 0.296875...
960962	i am so pleased with this purchase i am pregn...	[i, am, so, pleased, with, this, purchase, i, ...	[[-0.22558594, -0.01953125, 0.09082031, 0.2373...
1311381	this earcuff was small simple and wellmade ser...	[this, earcuff, was, small, simple, and, wellm...	[[0.109375, 0.140625, -0.03173828, 0.16601562,...
99933	i love it it fits well	[i, love, it, it, fits, well]	[[-0.22558594, -0.01953125, 0.09082031, 0.2373...
769897	beautiful colors are so pretty	[beautiful, colors, are, so, pretty]	[[-0.018310547, 0.055664062, -0.0115356445, 0....

100000 rows × 3 columns

In [9]:
```
df = X.input_3.apply(pd.Series)
df
```

Out[9]:

	0	1	2	3	4	
911218	[-0.055908203, 0.11767578, 0.2109375, 0.008361...	[0.40429688, -0.008666992, 0.11230469, 0.09179...	[0.080078125, 0.10498047, 0.049804688, 0.05346...	[0.13378906, -0.024414062, 0.07128906, 0.07568...	[-0.22558594, -0.01953125, 0.09082031, 0.23730...	[-0 0.0
668073	[0.080078125, 0.10498047, 0.049804688, 0.05346...	[-0.09863281, -0.087402344, -0.26367188, 0.053...	[0.064453125, 0.036132812, 0.03857422, 0.09472...	[0.22265625, -0.016601562, 0.13476562, -0.1337...	[0.13867188, -0.091796875, 0.03491211, 0.15039...	[0 -0
680243	[-0.0043029785, 0.14355469, 0.036376953, 0.129...	[-0.033935547, 0.16015625, 0.12060547, 0.06738...	[-0.071777344, -0.11035156, 0.008239746, 0.151...	[-0.22558594, -0.01953125, 0.09082031, 0.23730...	[-0.03149414, 0.26367188, 0.100097656, 0.00842...	[- -
317872	[0.17285156, -0.084472656, -0.29101562, 0.1855...	[-0.042236328, 0.080078125, -0.18652344, -0.07...	[-0.075683594, 0.033691406, -0.064941406, 0.13...	[0.012512207, -0.04736328, -0.13574219, -0.015...	[0.31054688, 0.06640625, 0.21386719, 0.0625, -...	0 -0.0
1048382	[0.016601562, 0.045654297, -0.119140625, 0.069...	[0.14160156, 0.15039062, 0.28125, -0.18847656,...	[0.080078125, 0.10498047, 0.049804688, 0.05346...	[-0.17285156, -0.22851562, -0.2109375, -0.1962...	[0.17089844, 0.024291992, 0.13867188, 0.022216...	[- -
...	...	...	...	...	...	
1197729	[-0.41210938, 0.18847656, -0.234375, 0.296875,...	[0.055908203, 0.057617188, 0.015197754, 0.2519...	[0.030029297, 0.09667969, 0.296875, 0.10791015...	[0.016357422, -0.016235352, -0.060546875, 0.04...	[-0.012023926, 0.0625, -0.09375, 0.25585938, -...	[-0
960962	[-0.22558594, -0.01953125, 0.09082031, 0.23730...	[-0.16699219, -0.06640625, 0.057373047, -0.059...	[-0.057128906, -0.030517578, -0.0055236816, 0....	[-0.20703125, 0.25195312, -0.014831543, -0.129...	[-0.024902344, 0.021972656, -0.03540039, 0.136...	- 0.
1311381	[0.109375, 0.140625, -0.03173828, 0.16601562, ...	[0.026000977, -0.0018920898, 0.18554688, -0.05...	[0.15917969, 0.095703125, -0.125, 0.10253906, ...	[0.30664062, -0.07519531, -0.052490234, 0.0344...	[0.067871094, -0.041259766, 0.100097656, 0.058...	[0 -0.0
99933	[-0.22558594, -0.01953125, 0.09082031, 0.23730...	[0.103027344, -0.15234375, 0.025878906, 0.1650...	[0.084472656, -0.0003528595, 0.053222656, 0.09...	[0.084472656, -0.0003528595, 0.053222656, 0.09...	[-0.17675781, 0.17578125, -0.15820312, -0.0520...	[- 0 -
769897	[-0.018310547, 0.055664062, -0.0115356445, 0.0...	[0.10205078, 0.16796875, 0.21386719, 0.0605468...	[-0.09667969, -0.026367188, 0.09033203, 0.0322...	[-0.057128906, -0.030517578, -0.0055236816, 0....	[0.123046875, -0.046142578, -0.20214844, 0.144...	0.0

100000 rows × 20 columns

In [10]:
```
X_train,X_test,Y_train, Y_test = train_test_split(df,Y, test_size=0

x=X_train
y=Y_train
test_x=X_test
test_y=Y_test
```

In [11]:
```
class RNN(nn.Module):
    def __init__(self, input_size, hidden_size1,hidden_size2, outpu
        super(RNN, self).__init__()

        self.hidden_size1 = hidden_size1
        self.hidden_size2 = hidden_size2

        self.i2h1 = nn.Linear(input_size + hidden_size2, hidden_siz
        self.h1h2 = nn.Linear(hidden_size1, hidden_size2)
        self.h2o = nn.Linear(hidden_size2, output_size)

        #self.i2o = nn.Linear(input_size + hidden_size2, output_siz
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden):
        combined = torch.cat([input,hidden],1)
        hidden1 = self.i2h1(combined)
        hidden2 = self.h1h2(hidden1)
        output = self.h2o(hidden2)
        output = self.softmax(output)
        return output, hidden2

    def initHidden1(self):
        return torch.zeros(1,self.hidden_size1)

    def initHidden2(self):
        return torch.zeros(1, self.hidden_size2)

rnn = RNN(300, 50,10, 5)
learning_rate=0.1
optimizer = torch.optim.SGD(rnn.parameters(), lr=learning_rate)
criterion = nn.CrossEntropyLoss()
```

In [*]:
```
epochs=1

rnn =rnn.float()
for i in range(epochs):

    for j in range(len(x)):
        #print("j",j)
        hidden2 = rnn.initHidden2()
        optimizer.zero_grad()
        #pred=[]
        #Hidden state - 20. Sending word by word vectors into RNN
        for index in range(20):
            temp=x.iloc[j,index]
            temp=np.reshape(temp,(1,300))
            temp_df=torch.tensor(temp)
            output, hidden2 = rnn(temp_df.float(), hidden2.float())
        #print("output",output)
        if(j==0):
            pred=output
        else:
            pred=torch.cat([pred,output],0)

    loss = criterion(pred, torch.tensor(y.to_numpy()-1))
    loss.backward()

    optimizer.step()
    print("Epoch ",i ," done")
```

In [*]:
```
for j in range(len(test_x)):
    hidden2 = rnn.initHidden2()
    optimizer.zero_grad()
    #pred=[]
    for index in range(20):
        temp=test_x.iloc[j,index]
        temp=np.reshape(temp,(1,300))
        temp_df=torch.tensor(temp)
        output, hidden2 = rnn(temp_df.float(), hidden2.float())
    #pred.append(output)
    if(j==0):
        pred=output
    else:
        pred=torch.cat([pred,output],0)

_,predicted = torch.max(pred.data,1)
correct = (predicted == test_y).sum()
print("Accuracy RNN 5(a):",correct,"/",len(test_y))
```

Question 5b

In [ ]:

```
# define the GRU architecture
class GRUModel(nn.Module):
    def __init__(self, input_dim, hidden_dim, layer_dim, output_dim
    super().__init__()

    # Number of hidden dimensions
    self.hidden_dim = hidden_dim

    # Number of hidden layers
    self.layer_dim = layer_dim

    # GRU
    self.gru = nn.GRU(input_dim, hidden_dim, layer_dim, batch_first
    # Output layer
    self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):

    # Initialize hidden state with zeros
    h0 = torch.zeros(self.layer_dim, x.size(0), self.hidden_dim)

    # One time step
    out, hn = self.gru(x, h0)
    out = self.fc(out[:, -1, :])
    return out

gru = GRUModel(300, 20, 1, 5)

criterion = nn.CrossEntropyLoss()

optimizer= torch.optim.SGD(gru.parameters(), lr=0.01)
```

In [ ]:
```
epochs=1

gru =gru.float()
for i in range(epochs):

    for j in range(len(x)):

        optimizer.zero_grad()
        pred=[]
        for index in range(20):
            temp=x.iloc[j,index]
            temp=np.reshape(temp,(1,300))
            temp_df=torch.tensor(temp)
            output = gru(temp_df.float())
        pred.append(output)
    #print(output.shape)
    ten=pred[0]
    for t in range(1,len(pred)):
        torch.cat([ten,t],0)

    print(ten.shape())
    loss = criterion(output, torch.tensor(y))
    loss.backward()

    optimizer.step()
```

In [ ]:
```
for j in range(len(test_x)):
    hidden2 = rnn.initHidden2()
    optimizer.zero_grad()
    #pred=[]
    for index in range(20):
        temp=test_x.iloc[j,index]
        temp=np.reshape(temp,(1,300))
        temp_df=torch.tensor(temp)
        output, hidden2 = rnn(temp_df.float(), hidden2.float())
    #pred.append(output)
    if(j==0):
        pred=output
    else:
        pred=torch.cat([pred,output],0)

_,predicted = torch.max(pred.data,1)
correct = (predicted == test_y).sum()
print(correct/len(test_y))
```