Installed Packages

In [ ]:
```
import sys
!{sys.executable} -m pip install contractions
!{sys.executable} -m pip install gensim==4.2.0
!pip install scikit-learn
!pip install torch torchvision torchaudio
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) https://us-python.pkg.dev/colab-wheel
s/public/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 anyascii
  Downloading anyascii-0.3.1-py3-none-any.whl (287 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 287.5/287.5 KB 7.5 MB/s eta 0:00:00
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 8.5 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
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) https://us-python.pkg.dev/colab-wheel
s/public/simple/ (https://us-python.pkg.dev/colab-wheels/public/simple/)
Collecting gensim==4.2.0
  Downloading gensim-4.2.0-cp38-cp38-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (24.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.1/24.1 MB 28.9 MB/s eta 0:00:00
```

In [ ]:
```
## Importing and installing libraries

import numpy as np
import copy
import pandas as pd
import warnings
import re
import sys
import nltk
from gensim.models import Word2Vec
from nltk.corpus import stopwords
import string
from torch import nn
import torch.nn as nn
import torch.nn.functional as F
import torch
from torch.nn import CrossEntropyLoss, Softmax, Linear
from torch.optim import SGD, Adam
from sklearn.metrics.pairwise import cosine_similarity
from torch.optim.lr_scheduler import ReduceLROnPlateau
from nltk.stem import WordNetLemmatizer
from gensim.models import KeyedVectors
from gensim import utils
from scipy.sparse import hstack
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron
from sklearn.metrics import classification_report
from sklearn import svm
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from statistics import mean
from os import path
import os.path
import gensim
import gensim.downloader
from sklearn.svm import LinearSVC

nltk.download('punkt')

warnings.filterwarnings('ignore')

import contractions
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

1. Dataset Generation

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive/')
%cd /content/drive/My Drive/Colab Notebooks/
```

```
Mounted at /content/drive/
/content/drive/My Drive/Colab Notebooks
```

In [ ]:
```
#fields required in the balanced dataframe from the original dataset
input_column=["review_body","star_rating"]

#reading the original dataset to filter the columns that are required
input_df =pd.read_csv('./amazon_reviews_us_Beauty_v1_00.tsv',usecols=input_column,sep='\t',error_bad_lines=Fals
```

In [ ]:
```
#Creating 3 different classes to get 20000 data from each class to avoid computational burden

class_one_df =(input_df[(input_df['star_rating'] == 1) | (input_df['star_rating'] == 2) ]).sample(n=20000)
class_one_df['class']=1

class_two_df =(input_df[(input_df['star_rating'] == 3)]).sample(n=20000)
class_two_df['class']=2

class_three_df =(input_df[(input_df['star_rating'] == 4) | (input_df['star_rating'] == 5) ]).sample(n=20000)
class_three_df['class']=3

#Combining all the data received from each class into a single balanced dataframe

amazon_balanced_df = pd.concat([class_one_df, class_two_df, class_three_df])

#Resetting the index as we have retrieved different data according to the classes created.
#Therefore, we will have irregular or unsorted index keys.
#We will reset the index to the new and incremental values from 0

amazon_balanced_df = amazon_balanced_df.reset_index(drop=True)

# Created a new dataframe consisting of the two columns (star_rating and review_body)
#along with class one assigned to them on the basis of star_rating. We are also resetting the index
```

Data Cleaning

Handling null values

In [ ]:
```
#We are changing all null values to an empty string

amazon_balanced_df = amazon_balanced_df.fillna('')
```

In [ ]:
```
#Uncleaned data copy
amazon_df=amazon_balanced_df.copy()
```

Convert all reviews into lowercase

In [ ]:
```
# Converting all review body into lowercase

amazon_balanced_df['review_body'] = amazon_balanced_df['review_body'].str.lower()
```

Remove the HTML from the reviews

In [ ]:
```
# Removing all the html tags from each review body

amazon_balanced_df['review_body']=amazon_balanced_df['review_body'].apply(lambda x : re.sub('<.*?>','',str(x)))
```

Remove the URLs from the reviews

In [ ]:
```
# Removing all the URLs from each review body

amazon_balanced_df['review_body'] = amazon_balanced_df['review_body'].apply(lambda y: re.split('https:\/\/.*', s
```

Remove non-alphabetical characters

In [ ]:
```
# Removing all the non alphabetic chaarcters(symbols, numbers) from each review body

amazon_balanced_df['review_body'] = amazon_balanced_df['review_body'].apply(lambda z: " ".join([re.sub('[^A-Za-
```

Remove extra spaces

In [ ]:
```
# Will remove leading and trailing spaces
amazon_balanced_df['review_body'] = amazon_balanced_df['review_body'].str.strip()
```

Perform contractions on the review_body

In [ ]:
```
## This will elongate the short form used in sentences like (I'll ---> I will)

amazon_balanced_df['without_contraction'] = amazon_balanced_df['review_body'].apply(lambda a: [contractions.fix
amazon_balanced_df['review_body'] = [' '.join(map(str, x)) for x in amazon_balanced_df['without_contraction']]
```

Remove Punctuations

In [ ]:
```
amazon_balanced_df['review_body'] = amazon_balanced_df['review_body'].str.replace(r'[^\w\s]+', '')
```

2. Word Embedding

(a) Downloading pretrained word2vec-google-news-300

In [ ]:
```
word2vec_model = gensim.downloader.load('word2vec-google-news-300')
```

```
[==================================================-] 99.7% 1657.5/1662.8MB downloaded
```

In [ ]:
```
word2vec_model.save('gensim2.kv')
```

In [ ]:
```
print(cosine_similarity([word2vec_model['queen']], [word2vec_model['king'] - word2vec_model['man'] + word2vec_mo
print(cosine_similarity([word2vec_model['queen']], [word2vec_model['king']]))
```

```
[[0.7300518]]
[[0.6510957]]
```

In [ ]:
```
word2vec_model.most_similar("good")
```

Out[18]:
```
[('great', 0.7291510105133057),
 ('bad', 0.7190051078796387),
 ('terrific', 0.6889115571975708),
 ('decent', 0.6837348341941833),
 ('nice', 0.6836092472076416),
 ('excellent', 0.644292950630188),
 ('fantastic', 0.6407778263092041),
 ('better', 0.6120728850364685),
 ('solid', 0.5806034803390503),
 ('lousy', 0.576420247554779)]
```

In [ ]:
```
word2vec_model.similarity(w1="daughter", w2="sister")
```

Out[19]:
```
0.7814771
```

(b) Training word2vec model on our own dataset

In [ ]:
```
class dataEmbed:
    def __init__(self, data_set):
      self.data_set = data_set

    def __iter__(self):
        for x in self.data_set:
            yield utils.simple_preprocess(x)
```

In [ ]:
```
sentence_embed = dataEmbed(amazon_balanced_df.review_body)
# window=13
# vector_size=300
# min_count=9
embed_word2vec = Word2Vec(sentences=sentence_embed, vector_size=300, min_count=9, window=13)
model = embed_word2vec.wv
```

In [ ]:
```
print(cosine_similarity([model['queen']], [model['king'] - model['man'] + model['woman']]))
print(cosine_similarity([model['queen']], [model['king']]))
```

```
[[0.1676019]]
[[0.40402395]]
```

In [ ]:
```
model.most_similar("good")
```

Out[23]:
```
[('great', 0.7492009997367859),
 ('decent', 0.6979432106018066),
 ('nice', 0.6569323539733887),
 ('fantastic', 0.600950300693512),
 ('ok', 0.5753679275512695),
 ('bad', 0.5533730387687683),
 ('okay', 0.5282017588615417),
 ('alright', 0.5122868418693542),
 ('awesome', 0.5109883546829224),
 ('high', 0.4818119406700134)]
```

In [ ]:
```
model.similarity(w1="daughter", w2="sister")
```

Out[24]:
```
0.8866891
```

3. Simple Models

Split the data into 80% train and 20% test

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(amazon_balanced_df['review_body'], amazon_balanced_df['class
```

In [ ]:
```
# print("Train Size ", X_train.shape)
# print("Test Size ", X_test.shape)
```

TF-IDF

In [ ]:
```
Tf_Id_Vector = TfidfVectorizer(
ngram_range=(1, 2),
analyzer='word',
token_pattern=r'\w{1,}',
strip_accents='unicode',
max_features=10000,
stop_words='english'
)
```

In [ ]:
```
X_tf_id_train = Tf_Id_Vector.fit_transform(X_train)
X_tf_id_test = Tf_Id_Vector.transform(X_test)
print("Train feature names", Tf_Id_Vector.get_feature_names_out())
```

```
Train feature names ['aa' 'ability' 'able' ... 'zipper' 'zits' 'zone']
```

Perceptron

In [ ]:
```
perceptron = Perceptron(
penalty= 'l1', #Provided to model on wrong prediction
alpha=0.000005, #Strength of the L2 regularization term
max_iter=1500, #Maximum number of iterations over the data
tol=1e-1, #Tolerance for the optimization or the criterion for stopping
)
perceptron.fit(X_tf_id_train , y_train)
#prediction through X_test data

y_test_prediction=perceptron.predict(X_tf_id_test)
#Preparing the report by comparing actual value and predicted value
report=classification_report(y_test, y_test_prediction)
print("\n Values for Perceptron Model")
print(report)
```

```
 Values for Perceptron Model
              precision    recall  f1-score   support

           1       0.61      0.69      0.65      4029
           2       0.52      0.51      0.52      3990
           3       0.73      0.67      0.70      3981

    accuracy                           0.62     12000
   macro avg       0.62      0.62      0.62     12000
weighted avg       0.62      0.62      0.62     12000
```

SVM

In [ ]:
```
svm = LinearSVC(
C=0.10, #Regularization parameter. Default is 1.
penalty='l2', #Norm of Penalty
tol=1e-1, #tolerance of Stopping criteria. default is 1e-3
class_weight="balanced", #adjust and provides weight to each class
max_iter=1000, #Hard limit on iterations within solver, or -1 for no limit.
random_state=1, #Controls the pseudo random number generation for shuffling the data for probability estimates.
loss='squared_hinge', #Specifies the Loss Function
dual=False, #Selects the algorithm to either the dual or primal optimization
fit_intercept=False, #Weather to calculate intercept for this model
)
svm.fit(X_tf_id_train, y_train)
#prediction through X_test data
y_test_prediction_svm=svm.predict(X_tf_id_test)
#Preparing the report by comparing actual value and predicted value
report_svm=classification_report(y_test, y_test_prediction_svm)
print("\n Values for SVM Model")
print(report_svm)
```

```
 Values for SVM Model
              precision    recall  f1-score   support

           1       0.69      0.70      0.70      4029
           2       0.61      0.56      0.58      3990
           3       0.74      0.79      0.76      3981

    accuracy                           0.68     12000
   macro avg       0.68      0.68      0.68     12000
weighted avg       0.68      0.68      0.68     12000
```

Process to extract word2vec embeddings

In [ ]:
```
embedding_space = []
for i in range(60000):
    vectorWord = np.zeros((1,300))
    listword = amazon_df['review_body'][i].split(" ")
    for word in listword:
        if word in word2vec_model.key_to_index:
            np.reshape(word2vec_model[word], (1, 300))
            vectorWord += word2vec_model[word]
        else:
            vectorWord += np.zeros((1,300))
    avg_wordVec = vectorWord/len(listword)
    embedding_space.append(avg_wordVec)


embedding_dataset = np.array(embedding_space)
print(embedding_dataset.shape)
embedding_dataset = embedding_dataset.reshape(embedding_dataset.shape[0], embedding_dataset.shape[2])
```

```
(60000, 1, 300)
```

In [ ]:
```
embedding_space_concat = []
for i in range(60000):
    vectorWord = [] # change the size of the vector
    listword = amazon_df['review_body'][i].split(" ")
    for item in listword[:20]:
        if item in word2vec_model:
            x=np.reshape(word2vec_model[item], (1, 300))
            vectorWord.append(x)
    vectorWord=vectorWord[1:]
    if len(vectorWord) < 20:
        di = 20 - len(vector_word)
        vectorWord += [np.zeros((1, 300))] * di

    embedding_space_concat.append(vectorWord)
embedding_space_concat=np.array(embedding_space_concat)
embedding_dataset=embedding_space_concat.reshape(embedding_space_concat.shape[0], embedding_space_concat.shape[
```

In [ ]:
```

A_train, A_test, B_train, B_test = train_test_split(embedding_dataset, amazon_df['class'], test_size=0.20, rando

B_train = B_train.reset_index(drop=True)
B_test = B_test.reset_index(drop=True)


print(A_train.shape, A_test.shape, B_train.shape, B_test.shape)
```

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

Perceptron

In [ ]:
```
perceptron = Perceptron(
penalty= 'l1', #Provided to model on wrong prediction
alpha=0.000005, #Strength of the L2 regularization term
max_iter=1500, #Maximum number of iterations over the data
tol=1e-1, #Tolerance for the optimization or the criterion for stopping
)
perceptron.fit(A_train , B_train)
#prediction through X_test data

B_test_prediction=perceptron.predict(A_test)
#Preparing the report by comparing actual value and predicted value
report=classification_report(B_test, B_test_prediction)
print("\n Values for Perceptron Model")
print(report)
```

```
 Values for Perceptron Model
              precision    recall  f1-score   support

           1       0.67      0.50      0.58      4000
           2       0.58      0.32      0.41      4000
           3       0.51      0.87      0.65      4000

    accuracy                           0.56     12000
   macro avg       0.59      0.56      0.54     12000
weighted avg       0.59      0.56      0.54     12000
```

SVM

In [ ]:
```
svm = LinearSVC(
C=0.10, #Regularization parameter. Default is 1.
penalty='l2', #Norm of Penalty
tol=1e-1, #tolerance of Stopping criteria. default is 1e-3
class_weight="balanced", #adjust and provides weight to each class
max_iter=1000, #Hard limit on iterations within solver, or -1 for no limit.
random_state=1, #Controls the pseudo random number generation for shuffling the data for probability estimates.
loss='squared_hinge', #Specifies the Loss Function
dual=False, #Selects the algorithm to either the dual or primal optimization
fit_intercept=False, #Weather to calculate intercept for this model
)
svm.fit(A_train , B_train)
#prediction through X_test data
B_test_prediction_svm=svm.predict(A_test)
#Preparing the report by comparing actual value and predicted value
report_svm=classification_report(B_test, B_test_prediction_svm)
print("\n Values for SVM Model")
print(report_svm)
```

```
 Values for SVM Model
              precision    recall  f1-score   support

           1       0.60      0.66      0.63      4000
           2       0.56      0.49      0.53      4000
           3       0.67      0.70      0.68      4000

    accuracy                           0.62     12000
   macro avg       0.61      0.62      0.61     12000
weighted avg       0.61      0.62      0.61     12000
```

4. Feedforward Neural Networks

In [ ]:
```
from torch.utils.data import Dataset, DataLoader
```

In [ ]:
```
#Creating a dataloader using torch
class dataloader(torch.utils.data.Dataset):
    def __init__(self, dataset_record, label_record):
        self.dataset = dataset_record
        self.labels = label_record

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, index):
        dataset = self.dataset[index]
        labels  = self.labels[index]

        return dataset, labels
```

In [ ]:
```
#Creating classes to define the architecure
class feedForward(nn.Module):
    def __init__(self, output_size, input_size):
        super(feedForward, self).__init__()
        self.layer1 = nn.Linear(input_size, 100)
        self.relu1 = nn.ReLU()
        self.layer2 = nn.Linear(100, 10)
        self.relu2 = nn.ReLU()
        self.layer3 = nn.Linear(10, output_size)

    def forward(self, x):
        return  self.layer3(self.relu2(self.layer2(self.relu1(self.layer1(x)))))
```

In [ ]:
```
fnn=feedForward(3,300)
fnn
```

Out[42]:
```
feedForward(
  (layer1): Linear(in_features=300, out_features=100, bias=True)
  (relu1): ReLU()
  (layer2): Linear(in_features=100, out_features=10, bias=True)
  (relu2): ReLU()
  (layer3): Linear(in_features=10, out_features=3, bias=True)
)
```

(a) Testing split of Multi layer perceptron and fetching accuracy of FNN

In [ ]:
```
# Convert A_train and A_test to float32
A_word2vec_train = A_train.astype(np.float32)
A_word2vec_test  = A_test.astype(np.float32)

# Subtract 1 from B_train and B_test values
B_train = B_train - 1
B_test = B_test - 1

# Create PyTorch DataLoader objects for the training and testing sets
train_dataset = dataloader(A_word2vec_train, B_train)
train_set = torch.utils.data.DataLoader(train_dataset, batch_size=50)

test_dataset = dataloader(A_word2vec_test, B_test)
test_set = torch.utils.data.DataLoader(test_dataset, batch_size=50)
```

In [ ]:
```
from sklearn.metrics import accuracy_score, f1_score
```

In [ ]:
```
def train(reviews_dataloader_train, reviews_dataloader_test, model, num_epochs, concat=False, rnn=False, gru=Fal
    y_pred_label_train = []
    y_true_label_train = []
    y_pred_label_test = []
    y_true_label_test = []

    # Set the device for the model
    # device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    # model.to(device)

    # Define the loss function and optimizer
    criterion = nn.CrossEntropyLoss()
    optimizer = Adam(model.parameters(), lr=0.001)
    softmax = Softmax(dim=1)

    # Define the scheduler
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1)

    # Keep track of the best model
    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    # Keep track of the previous loss
    loss_min = prev_loss

    # Train the model
    for epoch in range(num_epochs):
        print('\n Epoch: {}'.format(epoch))

        # print(reviews_dataloader_train)
        for j, (x, y) in enumerate(reviews_dataloader_train):
            y_pred = model(x)
            y_pred_label_train.append(torch.argmax(softmax(y_pred.detach()), axis=1))
            y_true_label_train.append(y.detach())
            loss = criterion(y_pred, y)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            # if j % 100 == 0:
            #     print('Epoch {:03} Batch {:03}/{:03} Loss: {:.4f}'.format(epoch, j, len(reviews_dataloader_tra

        # Evaluate the model on the test set
        with torch.no_grad():
            for x, y in reviews_dataloader_test:
                y_pred = model(x)
                y_pred_label_test.append(torch.argmax(softmax(y_pred.detach()), axis=1))
                y_true_label_test.append(y.detach())

        # Calculate accuracy and f1-score
        y_pred_train = torch.cat(y_pred_label_train)
        y_true_train = torch.cat(y_true_label_train)
        y_pred_test = torch.cat(y_pred_label_test)
        y_true_test = torch.cat(y_true_label_test)

        train_acc = accuracy_score(y_true_train.cpu().numpy(), y_pred_train.cpu().numpy())
        test_acc = accuracy_score(y_true_test.cpu().numpy(), y_pred_test.cpu().numpy())
        train_f1 = f1_score(y_true_train.cpu().numpy(), y_pred_train.cpu().numpy(), average='macro')
        test_f1 = f1_score(y_true_test.cpu().numpy(), y_pred_test.cpu().numpy(), average='macro')

        print('Epoch: {:03}, Loss: {:.4f}, Train Acc: {:.4f}, Test Acc: {:.4f}'.format(epoch, loss.item(), trai

        # Update the learning rate
        scheduler.step()

        # Save the best model based on test accuracy
        if test_acc > best_acc:
            best_acc = test_acc
            best_model_wts = copy.deepcopy(model.state_dict())

        # Save the model checkpoint
        # if loss.item() < loss_min:
        #     print(f'Loss decreased from {loss_min:.4f} to {loss.item():.4f}. Saving model...')
        #     torch.save(model.state_dict(), 'model_checkpoint.pt')
        #     loss
```

In [ ]:
```
train(train_set, test_set, fnn, 20)
```

```
 Epoch: 0
Epoch: 000, Loss: 0.8441, Train Acc: 0.5594, Test Acc: 0.6198

 Epoch: 1
Epoch: 001, Loss: 0.8092, Train Acc: 0.5902, Test Acc: 0.6267

 Epoch: 2
Epoch: 002, Loss: 0.7992, Train Acc: 0.6043, Test Acc: 0.6308

 Epoch: 3
Epoch: 003, Loss: 0.7932, Train Acc: 0.6129, Test Acc: 0.6334

 Epoch: 4
Epoch: 004, Loss: 0.7883, Train Acc: 0.6187, Test Acc: 0.6351

 Epoch: 5
Epoch: 005, Loss: 0.7794, Train Acc: 0.6241, Test Acc: 0.6365

 Epoch: 6
Epoch: 006, Loss: 0.7783, Train Acc: 0.6281, Test Acc: 0.6376

 Epoch: 7
Epoch: 007, Loss: 0.7764, Train Acc: 0.6312, Test Acc: 0.6384

 Epoch: 8
Epoch: 008, Loss: 0.7749, Train Acc: 0.6337, Test Acc: 0.6391

 Epoch: 9
Epoch: 009, Loss: 0.7736, Train Acc: 0.6357, Test Acc: 0.6396

 Epoch: 10
Epoch: 010, Loss: 0.7743, Train Acc: 0.6374, Test Acc: 0.6399

 Epoch: 11
Epoch: 011, Loss: 0.7742, Train Acc: 0.6388, Test Acc: 0.6402

 Epoch: 12
Epoch: 012, Loss: 0.7740, Train Acc: 0.6400, Test Acc: 0.6404

 Epoch: 13
Epoch: 013, Loss: 0.7739, Train Acc: 0.6410, Test Acc: 0.6407

 Epoch: 14
Epoch: 014, Loss: 0.7737, Train Acc: 0.6419, Test Acc: 0.6408

 Epoch: 15
Epoch: 015, Loss: 0.7739, Train Acc: 0.6427, Test Acc: 0.6410

 Epoch: 16
Epoch: 016, Loss: 0.7740, Train Acc: 0.6434, Test Acc: 0.6412

 Epoch: 17
Epoch: 017, Loss: 0.7741, Train Acc: 0.6440, Test Acc: 0.6414

 Epoch: 18
Epoch: 018, Loss: 0.7742, Train Acc: 0.6445, Test Acc: 0.6415

 Epoch: 19
Epoch: 019, Loss: 0.7742, Train Acc: 0.6450, Test Acc: 0.6417
```

In [ ]:
```
### To concatenate first 10 Word2Vec vectors for each review as the input feature
embedding_space_concat = []
for i in range(60000):
    vectorWord = np.zeros((1,300*10))  # change the size of the vector
    listword = amazon_df['review_body'][i].split(" ")
    for j, word in enumerate(listword):
        if j < 10:  # only consider the first 10 words
            if word in word2vec_model.key_to_index:
                np.reshape(word2vec_model[word], (1, 300))
                vectorWord[0, j*300:(j+1)*300] = word2vec_model[word]  # concatenate the vector
            else:
                vectorWord[0, j*300:(j+1)*300] = np.zeros((1,300))
    embedding_space_concat.append(vectorWord)

embedding_dataset_concat = np.array(embedding_space_concat)
print(embedding_dataset_concat.shape)
embedding_dataset_concat = embedding_dataset_concat.reshape(embedding_dataset_concat.shape[0], embedding_dataset
```

```
(60000, 1, 3000)
```

In [ ]:
```
P_train, P_test, Q_train, Q_test = train_test_split(embedding_space_concat, amazon_df['class'], test_size=0.20,

Q_train = Q_train.reset_index(drop=True)
Q_test = Q_test.reset_index(drop=True)

P_train= np.array(P_train)
P_test= np.array(P_test)
```

In [ ]:
```
type(P_train)
```

Out[68]:
```
numpy.ndarray
```

In [ ]:
```
# Convert A_train and A_test to float32
P_word2vec_train = P_train.astype(np.float32)
P_word2vec_test  = P_test.astype(np.float32)

# Subtract 1 from B_train and B_test values
Q_train = Q_train - 1
Q_test = Q_test - 1

# Create PyTorch DataLoader objects for the training and testing sets
train_datasetC = dataloader(P_word2vec_train, Q_train)
train_setC = torch.utils.data.DataLoader(train_datasetC, batch_size=50)

test_datasetC = dataloader(P_word2vec_test, Q_test)
test_setC = torch.utils.data.DataLoader(test_datasetC, batch_size=50)
```

In [ ]:
```
fnnc=feedForward(3,3000)
fnnc
```

In [ ]:
```
train(train_setC, test_setC, fnnc, 20, concat=True)
```

Installed Packages

In [1]:
```
import sys
!{sys.executable} -m pip install contractions
!{sys.executable} -m pip install gensim==4.2.0
!pip install scikit-learn
!pip install torch torchvision torchaudio
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) https://us-python.pkg.dev/colab-wheel
s/public/simple/ (https://us-python.pkg.dev/colab-wheels/public/simple/)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.8/dist-packages (1.2.1)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.8/dist-packages (from scikit-learn) (1.
10.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.8/dist-packages (from scikit-le
arn) (3.1.0)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.8/dist-packages (from scikit-learn)
(1.2.0)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.8/dist-packages (from scikit-learn)
(1.24.2)
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) https://us-python.pkg.dev/colab-wheel
s/public/simple/ (https://us-python.pkg.dev/colab-wheels/public/simple/)
Requirement already satisfied: torch in /usr/local/lib/python3.8/dist-packages (1.13.1+cu116)
Requirement already satisfied: torchvision in /usr/local/lib/python3.8/dist-packages (0.14.1+cu116)
Requirement already satisfied: torchaudio in /usr/local/lib/python3.8/dist-packages (0.13.1+cu116)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.8/dist-packages (from torch) (4.5.
0)
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (from torchvision) (1.24.2)
Requirement already satisfied: requests in /usr/local/lib/python3.8/dist-packages (from torchvision) (2.25.1)
```

In [2]:
```
## Importing and installing libraries

import numpy as np
import copy
import pandas as pd
import warnings
import re
import sys
import nltk
from gensim.models import Word2Vec
from nltk.corpus import stopwords
import string
from torch import nn
import torch.nn as nn
import torch.nn.functional as F
import torch
from torch.nn import CrossEntropyLoss, Softmax, Linear
from torch.optim import SGD, Adam
from sklearn.metrics.pairwise import cosine_similarity
from torch.optim.lr_scheduler import ReduceLROnPlateau
from nltk.stem import WordNetLemmatizer
from gensim.models import KeyedVectors
from gensim import utils
from scipy.sparse import hstack
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron
from sklearn.metrics import classification_report
from sklearn import svm
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from statistics import mean
from os import path
import os.path
import gensim
import gensim.downloader
from sklearn.svm import LinearSVC

nltk.download('punkt')

warnings.filterwarnings('ignore')

import contractions
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

1. Dataset Generation

In [3]:
```
from google.colab import drive
drive.mount('/content/drive/')
%cd /content/drive/My Drive/Colab Notebooks/
```

```
Mounted at /content/drive/
/content/drive/My Drive/Colab Notebooks
```

In [4]:
```
#fields required in the balanced dataframe from the original dataset
input_column=["review_body","star_rating"]

#reading the original dataset to filter the columns that are required
input_df =pd.read_csv('./amazon_reviews_us_Beauty_v1_00.tsv',usecols=input_column,sep='\t',error_bad_lines=False
```

In [5]:
```
#Creating 3 different classes to get 20000 data from each class to avoid computational burden

class_one_df =(input_df[(input_df['star_rating'] == 1) | (input_df['star_rating'] == 2) ]).sample(n=20000)
class_one_df['class']=1

class_two_df =(input_df[(input_df['star_rating'] == 3)]).sample(n=20000)
class_two_df['class']=2

class_three_df =(input_df[(input_df['star_rating'] == 4) | (input_df['star_rating'] == 5) ]).sample(n=20000)
class_three_df['class']=3

#Combining all the data received from each class into a single balanced dataframe

amazon_balanced_df = pd.concat([class_one_df, class_two_df, class_three_df])

#Resetting the index as we have retrieved different data according to the classes created.
#Therefore, we will have irregular or unsorted index keys.
#We will reset the index to the new and incremental values from 0

amazon_balanced_df = amazon_balanced_df.reset_index(drop=True)

# Created a new dataframe consisting of the two columns (star_rating and review_body)
#along with class one assigned to them on the basis of star_rating. We are also resetting the index
```

Data Cleaning

Handling null values

In [6]:
```
#We are changing all null values to an empty string

amazon_balanced_df = amazon_balanced_df.fillna('')
```

In [7]:
```
#Uncleaned data copy
amazon_df=amazon_balanced_df.copy()
```

Convert all reviews into lowercase

In [8]:
```
# Converting all review body into lowercase

amazon_balanced_df['review_body'] = amazon_balanced_df['review_body'].str.lower()
```

Remove the HTML from the reviews

In [9]:
```
# Removing all the html tags from each review body

amazon_balanced_df['review_body']=amazon_balanced_df['review_body'].apply(lambda x : re.sub('<.*?>','',str(x)))
```

Remove the URLs from the reviews

In [10]:
```
# Removing all the URLs from each review body

amazon_balanced_df['review_body'] = amazon_balanced_df['review_body'].apply(lambda y: re.split('https:\/\/.*', s
```

Remove non-alphabetical characters

In [11]:
```
# Removing all the non alphabetic chaarcters(symbols, numbers) from each review body

amazon_balanced_df['review_body'] = amazon_balanced_df['review_body'].apply(lambda z: " ".join([re.sub('[^A-Za-:
```

Remove extra spaces

In [12]:
```
# Will remove leading and trailing spaces
amazon_balanced_df['review_body'] = amazon_balanced_df['review_body'].str.strip()
```

Perform contractions on the review_body

In [13]:
```
## This will elongate the short form used in sentences like (I'll ---> I will)

amazon_balanced_df['without_contraction'] = amazon_balanced_df['review_body'].apply(lambda a: [contractions.fix
amazon_balanced_df['review_body'] = [' '.join(map(str, x)) for x in amazon_balanced_df['without_contraction']]
```

Remove Punctuations

In [14]:
```
amazon_balanced_df['review_body'] = amazon_balanced_df['review_body'].str.replace(r'[^\w\s]+', '')
```

2. Word Embedding

(a) Downloading pretrained word2vec-google-news-300

In [ ]:
```
# word2vec_model = gensim.downloader.load('word2vec-google-news-300')
```

In [ ]:
```
# word2vec_model.save('Gensim_word2vec_model.kv')
```

In [15]:
```
from gensim.models import KeyedVectors
word2vec_model= KeyedVectors.load("Gensim_word2vec_model.kv")
```

(b) Training word2vec model on our own dataset

In [16]:
```
class dataEmbed:
    def __init__(self, data_set):
      self.data_set = data_set

    def __iter__(self):
        for x in self.data_set:
            yield utils.simple_preprocess(x)
```

In [ ]:
```
# sentence_embed = dataEmbed(amazon_balanced_df.review_body)
# # window=13
# # vector_size=300
# # min_count=9
# embed_word2vec = Word2Vec(sentences=sentence_embed, vector_size=300, min_count=9, window=13)
# model = embed_word2vec.wv
```

Process to extract word2vec embeddings

In [18]:
```
### To concatenate first 10 Word2Vec vectors for each review as the input feature

embedding_space_concat = []
for i in range(60000):
    vectorWord = np.zeros((1,300))  # change the size of the vector
    listword = amazon_df['review_body'][i].split(" ")
    for item in listword[:10]:
        if item in word2vec_model:
            vectorWord = np.concatenate([vectorWord, np.expand_dims(word2vec_model[item], axis=0)], axis=0)

    vectorWord = vectorWord[1:]
    if len(vectorWord)<10:
        for i in range(10 - len(vectorWord)):
            vectorWord = np.concatenate([vectorWord, np.zeros((1,300))], axis=0)
    embedding_space_concat.append(vectorWord)

embedding_dataset_concat = np.array(embedding_space_concat)
embedding_dataset_concat = embedding_dataset_concat.reshape(embedding_dataset_concat.shape[0], embedding_dataset
```

In [19]:
```
print(embedding_dataset_concat.shape)
```

```
(60000, 3000)
```

In [20]:
```
P_train, P_test, Q_train, Q_test = train_test_split(embedding_dataset_concat, amazon_df['class'], test_size=0.2(

Q_train = Q_train.reset_index(drop=True)
Q_test = Q_test.reset_index(drop=True)


print(P_train.shape, P_test.shape, Q_train.shape, Q_test.shape)
```

```
(48000, 3000) (12000, 3000) (48000,) (12000,)
```

In [ ]:
```
type(P_train)
```

Out[25]:
```
numpy.ndarray
```

4. Feedforward Neural Networks

In [21]:
```
from torch.utils.data import Dataset, DataLoader
```

In [22]:
```
#Creating a dataloader using torch
class dataloader(torch.utils.data.Dataset):
    def __init__(self, dataset_record, label_record):
        self.dataset = dataset_record
        self.labels = label_record

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, index):
        dataset = self.dataset[index]
        labels  = self.labels[index]

        return dataset, labels
```

In [23]:
```
#Creating classes to define the architecure
class feedForward(nn.Module):
    def __init__(self, output_size, input_size):
        super(feedForward, self).__init__()
        self.layer1 = nn.Linear(input_size, 300)
        self.relu1 = nn.ReLU()
        self.layer2 = nn.Linear(300, 100)
        self.relu2 = nn.ReLU()
        self.layer3 = nn.Linear(100, output_size)

    def forward(self, x):
        return  self.layer3(self.relu2(self.layer2(self.relu1(self.layer1(x)))))
```

In [24]:
```
fnn=feedForward(3,3000)
fnn
```

Out[24]:
```
feedForward(
  (layer1): Linear(in_features=3000, out_features=300, bias=True)
  (relu1): ReLU()
  (layer2): Linear(in_features=300, out_features=100, bias=True)
  (relu2): ReLU()
  (layer3): Linear(in_features=100, out_features=3, bias=True)
)
```

(b)

In [25]:
```
# Convert P_train and P_test to float32
A_word2vec_train = P_train.astype(np.float32)
A_word2vec_test  = P_test.astype(np.float32)

# Subtract 1 from B_train and B_test values
B_train = Q_train - 1
B_test = Q_test - 1

# Create PyTorch DataLoader objects for the training and testing sets
train_dataset = dataloader(A_word2vec_train, B_train)
train_set = torch.utils.data.DataLoader(train_dataset, batch_size=50)

test_dataset = dataloader(A_word2vec_test, B_test)
test_set = torch.utils.data.DataLoader(test_dataset, batch_size=50)
```

In [26]:
```
from sklearn.metrics import accuracy_score, f1_score
```

In [27]:
```
f train(reviews_dataloader_train, reviews_dataloader_test, model, num_epochs, concat=True, rnn=False, gru=False,
    y_pred_label_train = []
    y_true_label_train = []
    y_pred_label_test = []
    y_true_label_test = []

    # Set the device for the model
    # device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    # model.to(device)

    # Define the loss function and optimizer
    criterion = nn.CrossEntropyLoss()
    optimizer = Adam(model.parameters(), lr=0.001)
    softmax = Softmax(dim=1)

    # Define the scheduler
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1)

    # Keep track of the best model
    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    # Keep track of the previous loss
    loss_min = prev_loss

    # Train the model
    for epoch in range(num_epochs):
        print('\n Epoch: {}'.format(epoch))

        # print(reviews_dataloader_train)
        for j, (x, y) in enumerate(reviews_dataloader_train):
            y_pred = model(x)
            y_pred_label_train.append(torch.argmax(softmax(y_pred.detach()), axis=1))
            y_true_label_train.append(y.detach())
            loss = criterion(y_pred, y)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            # if j % 100 == 0:
            #     print('Epoch {:03} Batch {:03}/{:03} Loss: {:.4f}'.format(epoch, j, len(reviews_dataloader_trair

        # Evaluate the model on the test set
        with torch.no_grad():
            for x, y in reviews_dataloader_test:
                y_pred = model(x)
                y_pred_label_test.append(torch.argmax(softmax(y_pred.detach()), axis=1))
                y_true_label_test.append(y.detach())

        # Calculate accuracy and f1-score
        y_pred_train = torch.cat(y_pred_label_train)
        y_true_train = torch.cat(y_true_label_train)
        y_pred_test = torch.cat(y_pred_label_test)
        y_true_test = torch.cat(y_true_label_test)

        train_acc = accuracy_score(y_true_train.cpu().numpy(), y_pred_train.cpu().numpy())
        test_acc = accuracy_score(y_true_test.cpu().numpy(), y_pred_test.cpu().numpy())
        train_f1 = f1_score(y_true_train.cpu().numpy(), y_pred_train.cpu().numpy(), average='macro')
        test_f1 = f1_score(y_true_test.cpu().numpy(), y_pred_test.cpu().numpy(), average='macro')

        print('Epoch: {:03}, Loss: {:.4f}, Train Acc: {:.4f}, Test Acc: {:.4f}'.format(epoch, loss.item(), train_a

        # Update the learning rate
        scheduler.step()

        # Save the best model based on test accuracy
        if test_acc > best_acc:
            best_acc = test_acc
            best_model_wts = copy.deepcopy(model.state_dict())

        # Save the model checkpoint
        # if loss.item() < loss_min:
        #     print(f'Loss decreased from {loss_min:.4f} to {loss.item():.4f}. Saving model...')
        #     torch.save(model.state_dict(), 'model_checkpoint.pt')
        #     loss
```

In [28]:
```
train(train_set, test_set, fnn, 20)
```

```
 Epoch: 0
Epoch: 000, Loss: 0.9506, Train Acc: 0.5246, Test Acc: 0.5460

 Epoch: 1
Epoch: 001, Loss: 0.8462, Train Acc: 0.5612, Test Acc: 0.5457

 Epoch: 2
Epoch: 002, Loss: 0.7152, Train Acc: 0.6004, Test Acc: 0.5366

 Epoch: 3
Epoch: 003, Loss: 0.5106, Train Acc: 0.6424, Test Acc: 0.5287

 Epoch: 4
Epoch: 004, Loss: 0.4892, Train Acc: 0.6795, Test Acc: 0.5248

 Epoch: 5
Epoch: 005, Loss: 0.5019, Train Acc: 0.7056, Test Acc: 0.5249

 Epoch: 6
Epoch: 006, Loss: 0.4329, Train Acc: 0.7307, Test Acc: 0.5245

 Epoch: 7
Epoch: 007, Loss: 0.3555, Train Acc: 0.7528, Test Acc: 0.5241

 Epoch: 8
Epoch: 008, Loss: 0.2691, Train Acc: 0.7723, Test Acc: 0.5237

 Epoch: 9
Epoch: 009, Loss: 0.1962, Train Acc: 0.7894, Test Acc: 0.5233

 Epoch: 10
Epoch: 010, Loss: 0.1898, Train Acc: 0.8041, Test Acc: 0.5228

 Epoch: 11
Epoch: 011, Loss: 0.1857, Train Acc: 0.8167, Test Acc: 0.5224

 Epoch: 12
Epoch: 012, Loss: 0.1792, Train Acc: 0.8277, Test Acc: 0.5219

 Epoch: 13
Epoch: 013, Loss: 0.1708, Train Acc: 0.8371, Test Acc: 0.5214

 Epoch: 14
Epoch: 014, Loss: 0.1628, Train Acc: 0.8455, Test Acc: 0.5211

 Epoch: 15
Epoch: 015, Loss: 0.1601, Train Acc: 0.8529, Test Acc: 0.5207

 Epoch: 16
Epoch: 016, Loss: 0.1590, Train Acc: 0.8594, Test Acc: 0.5204

 Epoch: 17
Epoch: 017, Loss: 0.1580, Train Acc: 0.8653, Test Acc: 0.5202

 Epoch: 18
Epoch: 018, Loss: 0.1570, Train Acc: 0.8705, Test Acc: 0.5200

 Epoch: 19
Epoch: 019, Loss: 0.1561, Train Acc: 0.8752, Test Acc: 0.5198
```

In [ ]:

Installed Packages

In [1]:
```
import sys
!{sys.executable} -m pip install contractions
!{sys.executable} -m pip install gensim==4.2.0
!pip install scikit-learn
!pip install torch torchvision torchaudio
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) https://us-python.pkg.dev/colab-wheel
s/public/simple/ (https://us-python.pkg.dev/colab-wheels/public/simple/)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.8/dist-packages (1.2.1)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.8/dist-packages (from scikit-learn)
(1.22.4)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.8/dist-packages (from scikit-le
arn) (3.1.0)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.8/dist-packages (from scikit-learn) (1.
10.1)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.8/dist-packages (from scikit-learn)
(1.2.0)
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) https://us-python.pkg.dev/colab-wheel
s/public/simple/ (https://us-python.pkg.dev/colab-wheels/public/simple/)
Requirement already satisfied: torch in /usr/local/lib/python3.8/dist-packages (1.13.1+cu116)
Requirement already satisfied: torchvision in /usr/local/lib/python3.8/dist-packages (0.14.1+cu116)
Requirement already satisfied: torchaudio in /usr/local/lib/python3.8/dist-packages (0.13.1+cu116)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.8/dist-packages (from torch) (4.5.
0)
Requirement already satisfied: requests in /usr/local/lib/python3.8/dist-packages (from torchvision) (2.25.1)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.8/dist-packages (from torchvis
```

In [2]:
```
## Importing and installing libraries

import numpy as np
import copy
import pandas as pd
import warnings
import re
import sys
import nltk
from gensim.models import Word2Vec
from nltk.corpus import stopwords
import string
from torch import nn
import torch.nn as nn
import torch.nn.functional as F
import torch
from torch.nn import CrossEntropyLoss, Softmax, Linear
from torch.optim import SGD, Adam
from sklearn.metrics.pairwise import cosine_similarity
from torch.optim.lr_scheduler import ReduceLROnPlateau
from nltk.stem import WordNetLemmatizer
from gensim.models import KeyedVectors
from gensim import utils
from scipy.sparse import hstack
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron
from sklearn.metrics import classification_report
from sklearn import svm
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from statistics import mean
from os import path
import os.path
import gensim
import gensim.downloader
from sklearn.svm import LinearSVC

nltk.download('punkt')

warnings.filterwarnings('ignore')

import contractions
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

1. Dataset Generation

In [3]:
```
from google.colab import drive
drive.mount('/content/drive/')
# %cd /content/drive/My Drive/Colab Notebooks/
```

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

In [4]:
```
%cd /content/drive/My Drive/
```

```
/content/drive/My Drive
```

In [5]:
```
#fields required in the balanced dataframe from the original dataset
input_column=["review_body","star_rating"]

#reading the original dataset to filter the columns that are required
input_df =pd.read_csv('./amazon_reviews_us_Beauty_v1_00.tsv',usecols=input_column,sep='\t',error_bad_lines=Fals
```

In [6]:
```
#Creating 3 different classes to get 20000 data from each class to avoid computational burden

class_one_df =(input_df[(input_df['star_rating'] == 1) | (input_df['star_rating'] == 2) ]).sample(n=20000)
class_one_df['class']=1

class_two_df =(input_df[(input_df['star_rating'] == 3)]).sample(n=20000)
class_two_df['class']=2

class_three_df =(input_df[(input_df['star_rating'] == 4) | (input_df['star_rating'] == 5) ]).sample(n=20000)
class_three_df['class']=3

#Combining all the data received from each class into a single balanced dataframe

amazon_balanced_df = pd.concat([class_one_df, class_two_df, class_three_df])

#Resetting the index as we have retrieved different data according to the classes created.
#Therefore, we will have irregular or unsorted index keys.
#We will reset the index to the new and incremental values from 0

amazon_balanced_df = amazon_balanced_df.reset_index(drop=True)

# Created a new dataframe consisting of the two columns (star_rating and review_body)
#along with class one assigned to them on the basis of star_rating. We are also resetting the index
```

Data Cleaning

Handling null values

In [7]:
```
#We are changing all null values to an empty string

amazon_balanced_df = amazon_balanced_df.fillna('')
```

In [8]:
```
#Uncleaned data copy
amazon_df=amazon_balanced_df.copy()
```

Convert all reviews into lowercase

In [9]:
```
# Converting all review body into lowercase

amazon_balanced_df['review_body'] = amazon_balanced_df['review_body'].str.lower()
```

Remove the HTML from the reviews

In [10]:
```
# Removing all the html tags from each review body

amazon_balanced_df['review_body']=amazon_balanced_df['review_body'].apply(lambda x : re.sub('<.*?>','',str(x)))
```

Remove the URLs from the reviews

In [11]:
```
# Removing all the URLs from each review body

amazon_balanced_df['review_body'] = amazon_balanced_df['review_body'].apply(lambda y: re.split('https:\/\/.*',
```

Remove non-alphabetical characters

In [12]:
```
# Removing all the non alphabetic chaarcters(symbols, numbers) from each review body

amazon_balanced_df['review_body'] = amazon_balanced_df['review_body'].apply(lambda z: " ".join([re.sub('[^A-Za-
```

Remove extra spaces

In [13]:
```
# Will remove leading and trailing spaces
amazon_balanced_df['review_body'] = amazon_balanced_df['review_body'].str.strip()
```

Perform contractions on the review_body

In [14]:
```
## This will elongate the short form used in sentences like (I'll ---> I will)

amazon_balanced_df['without_contraction'] = amazon_balanced_df['review_body'].apply(lambda a: [contractions.fix
amazon_balanced_df['review_body'] = [' '.join(map(str, x)) for x in amazon_balanced_df['without_contraction']]
```

Remove Punctuations

In [15]:
```
amazon_balanced_df['review_body'] = amazon_balanced_df['review_body'].str.replace(r'[^\w\s]+', '')
```

2. Word Embedding

(a) Downloading pretrained word2vec-google-news-300

In [21]:
```
# word2vec_model = gensim.downloader.load('word2vec-google-news-300')
# word2vec_model.save('Gensim_word2vec_model.kv')
```

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

In [16]:
```
from gensim.models import KeyedVectors
word2vec_model= KeyedVectors.load("Gensim_word2vec_model.kv")
```

Process to extract word2vec embeddings

In [ ]:
```
# embedding_space_concat = []
# for i in range(60000):
#     vectorWord = np.zeros((1,300))  # change the size of the vector
#     listword = amazon_df['review_body'][i].split(" ")
#     for item in listword[:20]:
#         if item in word2vec_model:
#             vectorWord = np.concatenate([vectorWord, np.expand_dims(word2vec_model[item], axis=0)], axis=0)

#     vectorWord = vectorWord[1:]
#     if len(vectorWord)<20:
#         for i in range(20 - len(vectorWord)):
#             vectorWord = np.concatenate([vectorWord, np.zeros((1,300))], axis=0)
#     embedding_space_concat.append(vectorWord)

# embedding_dataset_concat = np.array(embedding_space_concat)
# embedding_dataset_concat = embedding_dataset_concat.reshape(embedding_dataset_concat.shape[0], embedding_data
# embedding_dataset_concat
```

In [17]:
```
embedding_space_concat = []
for i in range(60000):
    vectorWord = [] # change the size of the vector
    listword = amazon_df['review_body'][i].split(" ")
    for item in listword[:20]:
        if item in word2vec_model:
            x=np.reshape(word2vec_model[item], (1, 300))
            vectorWord.append(x)
    vectorWord=vectorWord[1:]
    if len(vectorWord) < 20:
        di = 20 - len(vectorWord)
        vectorWord += [np.zeros((1, 300))] * di

    embedding_space_concat.append(vectorWord)
embedding_dataset_concat=np.array(embedding_space_concat)
embedding_dataset_concat=embedding_dataset_concat.reshape(embedding_dataset_concat.shape[0], embedding_dataset_c
```

In [18]:
```
embedding_dataset_concat.shape
```

Out[18]:
```
(60000, 20, 300)
```

In [19]:
```
A_train, A_test, B_train, B_test = train_test_split(embedding_dataset_concat, amazon_df['class'], test_size=0.2(

B_train = B_train.reset_index(drop=True)
B_test = B_test.reset_index(drop=True)
```

5. Recurrent Neural Networks

In [21]:
```
from torch.utils.data import Dataset, DataLoader
```

In [22]:
```
#Creating a dataloader using torch
class dataloader(torch.utils.data.Dataset):
    def __init__(self, dataset_record, label_record):
        self.dataset = dataset_record
        self.labels = label_record

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, index):
        dataset = self.dataset[index]
        labels  = self.labels[index]

        return dataset, labels
```

In [23]:
```
class RNN(nn.Module):
    def __init__(self, classes, layer, batch_size):
        super(RNN, self).__init__()
        self.rnn = nn.RNN(300, 600, layer, batch_first=True)
        self.h1 = torch.randn(layer, batch_size, 600)
        self.linear = nn.Linear(600, classes)

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

In [24]:
```
rnn_m = RNN(3, 2, 100)
rnn_m
```

Out[24]:
```
RNN(
  (rnn): RNN(300, 600, num_layers=2, batch_first=True)
  (linear): Linear(in_features=600, out_features=3, bias=True)
)
```

5. (a) Non Gated RNN

In [25]:
```
# Convert A_train and A_test to float32
A_word2vec_train = A_train.astype(np.float32)
A_word2vec_test  = A_test.astype(np.float32)

# Subtract 1 from B_train and B_test values
B_train = B_train - 1
B_test = B_test - 1

# Create PyTorch DataLoader objects for the training and testing sets
train_dataset = dataloader(A_word2vec_train, B_train)
train_set = torch.utils.data.DataLoader(train_dataset, batch_size=100)

test_dataset = dataloader(A_word2vec_test, B_test)
test_set = torch.utils.data.DataLoader(test_dataset, batch_size=100)
```

In [26]:
```
from sklearn.metrics import accuracy_score, f1_score
```

In [27]:
```
def train(reviews_dataloader_train, reviews_dataloader_test, model, num_epochs, concat=True, rnn=True, gru=False
    y_pred_label_train = []
    y_true_label_train = []
    y_pred_label_test = []
    y_true_label_test = []

    # Set the device for the model
    # device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    # model.to(device)

    # Define the loss function and optimizer
    criterion = nn.CrossEntropyLoss()
    optimizer = Adam(model.parameters(), lr=0.001)
    # optimizer = SGD(rnn.parameters(), lr=1e-2)
    scheduler = ReduceLROnPlateau(optimizer)

    # optimizer = Adam(model.parameters(), lr=0.001)
    softmax = Softmax(dim=1)

    # Define the scheduler
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1)

    # Keep track of the best model
    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    # Keep track of the previous loss
    loss_min = prev_loss

    # Train the model
    for epoch in range(num_epochs):
        print('\n Epoch: {}'.format(epoch))

        # print(reviews_dataloader_train)
        for j, (x, y) in enumerate(reviews_dataloader_train):
            y_pred = model(x)
            y_pred_label_train.append(torch.argmax(softmax(y_pred.detach()), axis=1))
            y_true_label_train.append(y.detach())
            loss = criterion(y_pred, y)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            # if j % 100 == 0:
            #     print('Epoch {:03} Batch {:03}/{:03} Loss: {:.4f}'.format(epoch, j, len(reviews_dataloader_tra

        # Evaluate the model on the test set
        with torch.no_grad():
            for x, y in reviews_dataloader_test:
                y_pred = model(x)
                y_pred_label_test.append(torch.argmax(softmax(y_pred.detach()), axis=1))
                y_true_label_test.append(y.detach())

        # Calculate accuracy and f1-score
        y_pred_train = torch.cat(y_pred_label_train)
        y_true_train = torch.cat(y_true_label_train)
        y_pred_test = torch.cat(y_pred_label_test)
        y_true_test = torch.cat(y_true_label_test)

        train_acc = accuracy_score(y_true_train.cpu().numpy(), y_pred_train.cpu().numpy())
        test_acc = accuracy_score(y_true_test.cpu().numpy(), y_pred_test.cpu().numpy())
        train_f1 = f1_score(y_true_train.cpu().numpy(), y_pred_train.cpu().numpy(), average='macro')
        test_f1 = f1_score(y_true_test.cpu().numpy(), y_pred_test.cpu().numpy(), average='macro')

        print('Epoch: {:03}, Loss: {:.4f}, Train Acc: {:.4f}, Test Acc: {:.4f}'.format(epoch, loss.item(), trai

        # Update the learning rate
        scheduler.step()

        # Save the best model based on test accuracy
        if test_acc > best_acc:
            best_acc = test_acc
            best_model_wts = copy.deepcopy(model.state_dict())

        # Save the model checkpoint
        # if loss.item() < loss_min:
        #     print(f'Loss decreased from {loss_min:.4f} to {loss.item():.4f}. Saving model...')
        #     torch.save(model.state_dict(), 'model_checkpoint.pt')
        #     loss
```

In [28]:
```
train(train_set, test_set, rnn_m, 20)
```

```
 Epoch: 0
Epoch: 000, Loss: 1.0730, Train Acc: 0.3482, Test Acc: 0.3767

 Epoch: 1
Epoch: 001, Loss: 1.1250, Train Acc: 0.3590, Test Acc: 0.3447

 Epoch: 2
Epoch: 002, Loss: 1.1038, Train Acc: 0.3572, Test Acc: 0.3496

 Epoch: 3
Epoch: 003, Loss: 1.1070, Train Acc: 0.3566, Test Acc: 0.3521

 Epoch: 4
Epoch: 004, Loss: 1.1113, Train Acc: 0.3562, Test Acc: 0.3537

 Epoch: 5
Epoch: 005, Loss: 1.0876, Train Acc: 0.3581, Test Acc: 0.3576

 Epoch: 6
Epoch: 006, Loss: 1.0864, Train Acc: 0.3598, Test Acc: 0.3606

 Epoch: 7
Epoch: 007, Loss: 1.0828, Train Acc: 0.3612, Test Acc: 0.3629

 Epoch: 8
Epoch: 008, Loss: 1.0826, Train Acc: 0.3623, Test Acc: 0.3647

 Epoch: 9
Epoch: 009, Loss: 1.0824, Train Acc: 0.3632, Test Acc: 0.3661

 Epoch: 10
Epoch: 010, Loss: 1.0850, Train Acc: 0.3645, Test Acc: 0.3672

 Epoch: 11
Epoch: 011, Loss: 1.0849, Train Acc: 0.3657, Test Acc: 0.3682

 Epoch: 12
Epoch: 012, Loss: 1.0849, Train Acc: 0.3666, Test Acc: 0.3690

 Epoch: 13
Epoch: 013, Loss: 1.0849, Train Acc: 0.3674, Test Acc: 0.3697

 Epoch: 14
Epoch: 014, Loss: 1.0849, Train Acc: 0.3682, Test Acc: 0.3703

 Epoch: 15
Epoch: 015, Loss: 1.0849, Train Acc: 0.3689, Test Acc: 0.3709

 Epoch: 16
Epoch: 016, Loss: 1.0850, Train Acc: 0.3695, Test Acc: 0.3713

 Epoch: 17
Epoch: 017, Loss: 1.0850, Train Acc: 0.3701, Test Acc: 0.3717

 Epoch: 18
Epoch: 018, Loss: 1.0850, Train Acc: 0.3706, Test Acc: 0.3721

 Epoch: 19
Epoch: 019, Loss: 1.0850, Train Acc: 0.3710, Test Acc: 0.3725
```

5. (b) Gated RNN

In [29]:
```
class GatedRNN(nn.Module):
    def __init__(self, num_classes, layers, batch_size):
        super(GatedRNN, self).__init__()
        self.rnn = nn.GRU(300, 300, layers, batch_first=True)
        self.h1 = torch.randn(layers, batch_size, 300)
        self.linear = nn.Linear(300, num_classes)

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

In [30]:
```
gru = GatedRNN(5, 1, 100)
```

In [31]:
```
train(train_set, test_set, gru, 20,True,True, True)
```

```
 Epoch: 0
Epoch: 000, Loss: 0.9270, Train Acc: 0.5037, Test Acc: 0.5558

 Epoch: 1
Epoch: 001, Loss: 0.8748, Train Acc: 0.5415, Test Acc: 0.5694

 Epoch: 2
Epoch: 002, Loss: 0.8604, Train Acc: 0.5606, Test Acc: 0.5769

 Epoch: 3
Epoch: 003, Loss: 0.8465, Train Acc: 0.5739, Test Acc: 0.5822

 Epoch: 4
Epoch: 004, Loss: 0.8341, Train Acc: 0.5850, Test Acc: 0.5863

 Epoch: 5
Epoch: 005, Loss: 0.8319, Train Acc: 0.5962, Test Acc: 0.5909

 Epoch: 6
Epoch: 006, Loss: 0.8230, Train Acc: 0.6048, Test Acc: 0.5939

 Epoch: 7
Epoch: 007, Loss: 0.8148, Train Acc: 0.6116, Test Acc: 0.5961

 Epoch: 8
Epoch: 008, Loss: 0.8064, Train Acc: 0.6173, Test Acc: 0.5978

 Epoch: 9
Epoch: 009, Loss: 0.7969, Train Acc: 0.6221, Test Acc: 0.5992

 Epoch: 10
Epoch: 010, Loss: 0.7995, Train Acc: 0.6266, Test Acc: 0.6007

 Epoch: 11
Epoch: 011, Loss: 0.7995, Train Acc: 0.6304, Test Acc: 0.6019

 Epoch: 12
Epoch: 012, Loss: 0.7987, Train Acc: 0.6336, Test Acc: 0.6029

 Epoch: 13
Epoch: 013, Loss: 0.7977, Train Acc: 0.6364, Test Acc: 0.6037

 Epoch: 14
Epoch: 014, Loss: 0.7966, Train Acc: 0.6389, Test Acc: 0.6044

 Epoch: 15
Epoch: 015, Loss: 0.7967, Train Acc: 0.6411, Test Acc: 0.6050

 Epoch: 16
Epoch: 016, Loss: 0.7967, Train Acc: 0.6430, Test Acc: 0.6056

 Epoch: 17
Epoch: 017, Loss: 0.7967, Train Acc: 0.6447, Test Acc: 0.6061

 Epoch: 18
Epoch: 018, Loss: 0.7967, Train Acc: 0.6463, Test Acc: 0.6065

 Epoch: 19
Epoch: 019, Loss: 0.7967, Train Acc: 0.6476, Test Acc: 0.6069
```

Installed Packages

In [ ]:

```
import sys
!{sys.executable} -m pip install contractions
!{sys.executable} -m pip install gensim==4.2.0
!pip install scikit-learn
!pip install torch torchvision torchaudio
```

In [ ]:

```
## Importing and installing libraries

import numpy as np
import copy
import pandas as pd
import warnings
import re
import sys
import nltk
from gensim.models import Word2Vec
from nltk.corpus import stopwords
import string
from torch import nn
import torch.nn as nn
import torch.nn.functional as F
import torch
from torch.nn import CrossEntropyLoss, Softmax, Linear
from torch.optim import SGD, Adam
from sklearn.metrics.pairwise import cosine_similarity
from torch.optim.lr_scheduler import ReduceLROnPlateau
from nltk.stem import WordNetLemmatizer
from gensim.models import KeyedVectors
from gensim import utils
from scipy.sparse import hstack
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron
from sklearn.metrics import classification_report
from sklearn import svm
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from statistics import mean
from os import path
import os.path
import gensim
import gensim.downloader
from sklearn.svm import LinearSVC

nltk.download('punkt')

warnings.filterwarnings('ignore')

import contractions
```

1. Dataset Generation

In [6]:
```
#fields required in the balanced dataframe from the original dataset
input_column=["review_body","star_rating"]

#reading the original dataset to filter the columns that are required
input_df =pd.read_csv('https://s3.amazonaws.com/amazon-reviews-pds/tsv/ama
```

In [8]:
```
#Creating 3 different classes to get 20000 data from each class to avoid c

class_one_df =(input_df[(input_df['star_rating'] == 1) | (input_df['star_r
class_one_df['class']=1

class_two_df =(input_df[(input_df['star_rating'] == 3)]).sample(n=20000)
class_two_df['class']=2

class_three_df =(input_df[(input_df['star_rating'] == 4) | (input_df['star
class_three_df['class']=3

#Combining all the data received from each class into a single balanced da

amazon_balanced_df = pd.concat([class_one_df, class_two_df, class_three_df

#Resetting the index as we have retrieved different data according to the
#Therefore, we will have irregular or unsorted index keys.
#We will reset the index to the new and incremental values from 0

amazon_balanced_df = amazon_balanced_df.reset_index(drop=True)

# Created a new dataframe consisting of the two columns (star_rating and r
#along with class one assigned to them on the basis of star_rating. We are
```

Data Cleaning

Handling null values

In [9]:
```
#We are changing all null values to an empty string

amazon_balanced_df = amazon_balanced_df.fillna('')
```

In [10]:
```
#Uncleaned data copy
amazon_df=amazon_balanced_df.copy()
```

Convert all reviews into lowercase

In [11]:
```
# Converting all review body into lowercase

amazon_balanced_df['review_body'] = amazon_balanced_df['review_body'].str.
```

Remove the HTML from the reviews

In [12]:
```
# Removing all the html tags from each review body

amazon_balanced_df['review_body']=amazon_balanced_df['review_body'].apply(
```

Remove the URLs from the reviews

In [13]:
```
# Removing all the URLs from each review body

amazon_balanced_df['review_body'] = amazon_balanced_df['review_body'].appl
```

Remove non-alphabetical characters

In [14]:
```
# Removing all the non alphabetic chaarcters(symbols, numbers) from each r

amazon_balanced_df['review_body'] = amazon_balanced_df['review_body'].appl
```

Remove extra spaces

In [15]:
```
# Will remove leading and trailing spaces
amazon_balanced_df['review_body'] = amazon_balanced_df['review_body'].str.
```

Perform contractions on the review_body

In [16]:

```
## This will elongate the short form used in sentences like (I'll ---> I w

amazon_balanced_df['without_contraction'] = amazon_balanced_df['review_boc
amazon_balanced_df['review_body'] = [' '.join(map(str, x)) for x in amazor
```

Remove Punctuations

In [17]:

```
amazon_balanced_df['review_body'] = amazon_balanced_df['review_body'].str.
```

2. Word Embedding

(a) Downloading pretrained word2vec-google-news-300

In [20]:

```
# word2vec_model = gensim.downloader.load('word2vec-google-news-300')
# word2vec_model.save('Gensim_word2vec_model.kv')
```

In [21]:

```
from gensim.models import KeyedVectors
word2vec_model= KeyedVectors.load("Gensim_word2vec_model.kv")
```

Process to extract word2vec embeddings

In [23]:
```
embedding_space_concat = []
for i in range(60000):
    vectorWord = [] # change the size of the vector
    listword = amazon_df['review_body'][i].split(" ")
    for item in listword[:20]:
        if item in word2vec_model:
            x=np.reshape(word2vec_model[item], (1, 300))
            vectorWord.append(x)
    vectorWord=vectorWord[1:]
    if len(vectorWord) < 20:
        di = 20 - len(vectorWord)
        vectorWord += [np.zeros((1, 300))] * di

    embedding_space_concat.append(vectorWord)
embedding_dataset_concat=np.array(embedding_space_concat)
embedding_dataset_concat=embedding_dataset_concat.reshape(embedding_datase
```

In [24]:
```
embedding_dataset_concat.shape
```

Out[24]: (60000, 20, 300)

In [25]:
```
A_train, A_test, B_train, B_test = train_test_split(embedding_dataset_conc
```

In [26]:
```

B_train = B_train.reset_index(drop=True)
B_test = B_test.reset_index(drop=True)


print(A_train.shape, A_test.shape, B_train.shape, B_test.shape)
```

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

5. Recurrent Neural Networks

In [27]:
```
from torch.utils.data import Dataset, DataLoader
```

In [28]:

```
#Creating a dataloader using torch
class dataloader(torch.utils.data.Dataset):
    def __init__(self, dataset_record, label_record):
        self.dataset = dataset_record
        self.labels = label_record

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, index):
        dataset = self.dataset[index]
        labels  = self.labels[index]

        return dataset, labels
```

In [29]:

```
# Convert A_train and A_test to float32
A_word2vec_train = A_train.astype(np.float32)
A_word2vec_test  = A_test.astype(np.float32)

# Subtract 1 from B_train and B_test values
B_train = B_train - 1
B_test = B_test - 1

# Create PyTorch DataLoader objects for the training and testing sets
train_dataset = dataloader(A_word2vec_train, B_train)
train_set = torch.utils.data.DataLoader(train_dataset, batch_size=100)

test_dataset = dataloader(A_word2vec_test, B_test)
test_set = torch.utils.data.DataLoader(test_dataset, batch_size=100)
```

In [30]:

```
from sklearn.metrics import accuracy_score, f1_score
```

In [31]:

```
def train(reviews_dataloader_train, reviews_dataloader_test, model, num_ep
    y_pred_label_train = []
    y_true_label_train = []
    y_pred_label_test = []
    y_true_label_test = []

    # Set the device for the model
    # device = torch.device('cuda' if torch.cuda.is_available() else 'cpu
    # model.to(device)

    # Define the loss function and optimizer
    criterion = nn.CrossEntropyLoss()
    optimizer = Adam(model.parameters(), lr=0.001)
    # optimizer = SGD(rnn.parameters(), lr=1e-2)
    scheduler = ReduceLROnPlateau(optimizer)

    # optimizer = Adam(model.parameters(), lr=0.001)
    softmax = Softmax(dim=1)

    # Define the scheduler
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=5, ga

    # Keep track of the best model
    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    # Keep track of the previous loss
    loss_min = prev_loss

    # Train the model
    for epoch in range(num_epochs):
        print('\n Epoch: {}'.format(epoch))

        # print(reviews_dataloader_train)
        for j, (x, y) in enumerate(reviews_dataloader_train):
            y_pred = model(x)
            y_pred_label_train.append(torch.argmax(softmax(y_pred.detach()
            y_true_label_train.append(y.detach())
            loss = criterion(y_pred, y)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            # if j % 100 == 0:
            #     print('Epoch {:03} Batch {:03}/{:03} Loss: {:.4f}'.forma

        # Evaluate the model on the test set
        with torch.no_grad():
            for x, y in reviews_dataloader_test:
                y_pred = model(x)
                y_pred_label_test.append(torch.argmax(softmax(y_pred.detac
                y_true_label_test.append(y.detach())

        # Calculate accuracy and f1-score
        y_pred_train = torch.cat(y_pred_label_train)
        y_true_train = torch.cat(y_true_label_train)
```

```
        y_pred_test = torch.cat(y_pred_label_test)
        y_true_test = torch.cat(y_true_label_test)

        train_acc = accuracy_score(y_true_train.cpu().numpy(), y_pred_trai
        test_acc = accuracy_score(y_true_test.cpu().numpy(), y_pred_test.c
        train_f1 = f1_score(y_true_train.cpu().numpy(), y_pred_train.cpu()
        test_f1 = f1_score(y_true_test.cpu().numpy(), y_pred_test.cpu().nu

        print('Epoch: {:03}, Loss: {:.4f}, Train Acc: {:.4f}, Test Acc: {:

        # Update the learning rate
        scheduler.step()

        # Save the best model based on test accuracy
        if test_acc > best_acc:
            best_acc = test_acc
            best_model_wts = copy.deepcopy(model.state_dict())

        # Save the model checkpoint
        # if loss.item() < loss_min:
        #     print(f'Loss decreased from {loss_min:.4f} to {loss.item():.
        #     torch.save(model.state_dict(), 'model_checkpoint.pt')
        #     loss
```

5. (c)

In [32]:
```
class LSTM(nn.Module):
    def __init__(self, num_classes, layers, hidden_size):
        super(LSTM, self).__init__()
        self.lstm = nn.LSTM(300, hidden_size, layers, batch_first=True)
        self.linear = nn.Linear(hidden_size, num_classes)

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

In [37]:
```
lstm = LSTM(3,30,100)
```

In [50]:
```
tr = train(train_set, test_set, lstm, 5,True,True)
```

```
 Epoch: 0
Epoch: 000, Loss: 0.8173, Train Acc: 0.6202, Test Acc: 0.6027

 Epoch: 1
Epoch: 001, Loss: 0.7975, Train Acc: 0.6246, Test Acc: 0.6059

 Epoch: 2
Epoch: 002, Loss: 0.7789, Train Acc: 0.6294, Test Acc: 0.6079

 Epoch: 3
Epoch: 003, Loss: 0.7677, Train Acc: 0.6345, Test Acc: 0.6096

 Epoch: 4
Epoch: 004, Loss: 0.7480, Train Acc: 0.6393, Test Acc: 0.6114

 Epoch: 5
```

In [1]:
```
print('Accuracy for LSTM is :61.14' )
```

```
Accuracy for LSTM is :61.14
```

In [ ]:
```

```