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 [ ]:
```

```