544-summary-HW2

xingyu xie

February 2024

1 Word Embedding

1.1 a

pretrained Google W2V model:
Similariy between water and rock : 0.1448098
Similariy between ice and rock : 0.26914403
Similariy between apple and iphone : 0.3199041
Similariy between apple and ipad : 0.35862675

1.2 b

self-trained W2V model:
Similariy between water and rock : 0.16710913
Similariy between ice and rock : 0.29178178
Similariy between apple and iphone : 0.59669954
Similariy between apple and ipad : 0.55011463
On common words, google's pretrained model seems work better because they use larger corpus to train W2V model, which could result in better similarity results. But among some specific e-commerce words, my model seems to have better performance.

2 Simple models

Noted: In the rest of this report, I present the the best recorded score which might be different from the jupyter report due to the dataset(250k raw dataset and train-test sampled dataset) which have a influence on W2V weights and model performance.
TF-IDF result is: 0.855925 (Perceptron) and 0.895625 (LinearSVC) and 0.9035 (SVC).
Google:
Perceptron: 0.7809
SVM: 0.8324

Self:
Perceptron: 0.7348
SVM: 0.8644
My conclusion: IF-IDF model could still provide good support for some simple NLP tasks, which could even better than the Word2Vec model. My trained Word2Vec could perform better compared with google's model because it trained on my dataset (250k). And I believe if I trained my Word2Vec on the whole amazon review dataset, it could provide even more better results. By the way, I download the google pretrained model weight file. I add amazon review into its vocabulary and retrained the model (google and self), it gives the best results among these three Word2Vec model.

3 Feedforward Neural Networks

3.1 a-binary

Google: 0.8428
Self: 0.8617

3.2 a-ternary

Google: 0.6847
Self: 0.7017

3.3 b-binary

Google: 0.7536
Self: 0.7865

3.4 b-ternary

Google: 0.6222
Self: 0.6398

3.5 binary-conclusion

My conclusion:
(1) Using the average vertor of sentence is better than concatenating the first 10 Word2Vec vectors of sentence because the former one could provide more information of this sentence, but the later one have the chance to only keep useless words which have nothing to do with sentiment classification.

(2) Neural Network have the potential ability to perform better than traditional machine learning model (Perceptron), but SVM is still powerful in NLP tasks.

4 Convolutional Neural Networks

4.1 binary

Google:0.8532
Self:0.8585

4.2 ternary

Google:0.6930
Self:0.6900

In [13]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

prepare dataset

In [15]:
```
raw_data = pd.read_csv('./amazon_reviews_us_Office_Products_v1_00.tsv', sep='\t'
```

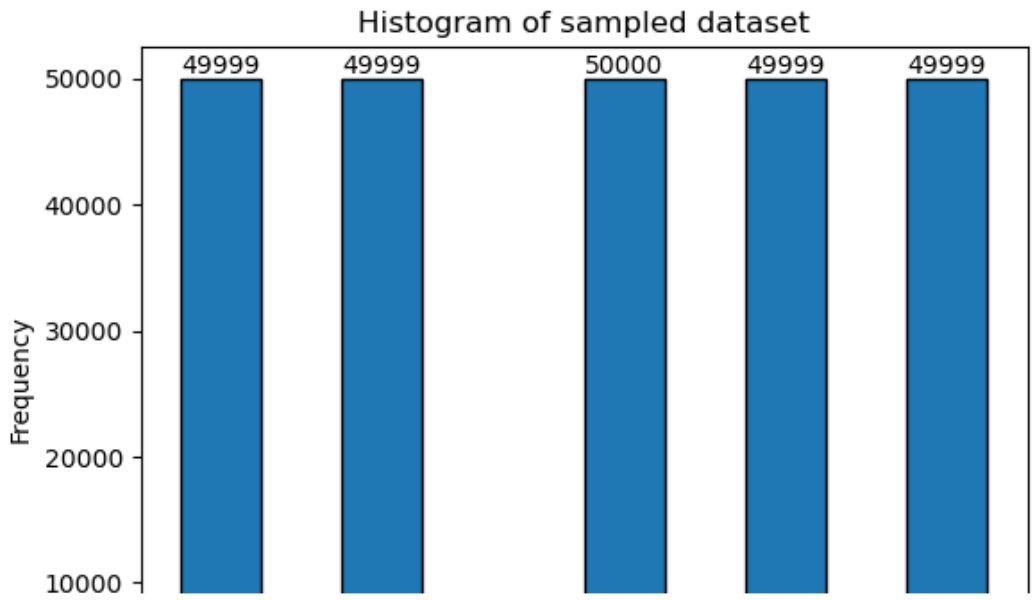
```
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\IPython\core\interactiv
eshell.py:3457: FutureWarning: The error_bad_lines argument has been deprec
ated and will be removed in a future version.


  exec(code_obj, self.user_global_ns, self.user_ns)
b'Skipping line 20773: expected 15 fields, saw 22\nSkipping line 39834: exp
ected 15 fields, saw 22\nSkipping line 52957: expected 15 fields, saw 22\nS
kipping line 54540: expected 15 fields, saw 22\n'
b'Skipping line 80276: expected 15 fields, saw 22\nSkipping line 96168: exp
ected 15 fields, saw 22\nSkipping line 96866: expected 15 fields, saw 22\nS
kipping line 98175: expected 15 fields, saw 22\nSkipping line 112539: expec
ted 15 fields, saw 22\nSkipping line 119377: expected 15 fields, saw 22\nSk
ipping line 120065: expected 15 fields, saw 22\nSkipping line 124703: expec
ted 15 fields, saw 22\n'
b'Skipping line 134024: expected 15 fields, saw 22\nSkipping line 153938: e
xpected 15 fields, saw 22\nSkipping line 156225: expected 15 fields, saw 22
\nSkipping line 168603: expected 15 fields, saw 22\nSkipping line 187002: e
xpected 15 fields, saw 22\n'
```

In [16]:
```
columns = ['review_body', 'star_rating']
raw_review_rating =  raw_data[columns]
```

In [26]:
```
values_to_stratify = [1,2,3,4,5]

def stratified_sample(group, n=50001):
    return group.sample(n=n, random_state=1)



sampled_data = raw_review_rating[raw_review_rating['star_rating'].isin(values_to

sampled_data = sampled_data[sampled_data['review_body'].apply(lambda x: isinsta
```

In [25]:
```
ax = sampled_data['star_rating'].plot(kind='hist', bins=10, edgecolor='black')
for p in ax.patches:
    ax.annotate(str(int(p.get_height())), (p.get_x() + p.get_width() / 2., p.ge
                ha='center', va='center', xytext=(0, 5), textcoords='offset poi
plt.title('Histogram of sampled dataset')
plt.xlabel('Values')
plt.ylabel('Frequency')
```

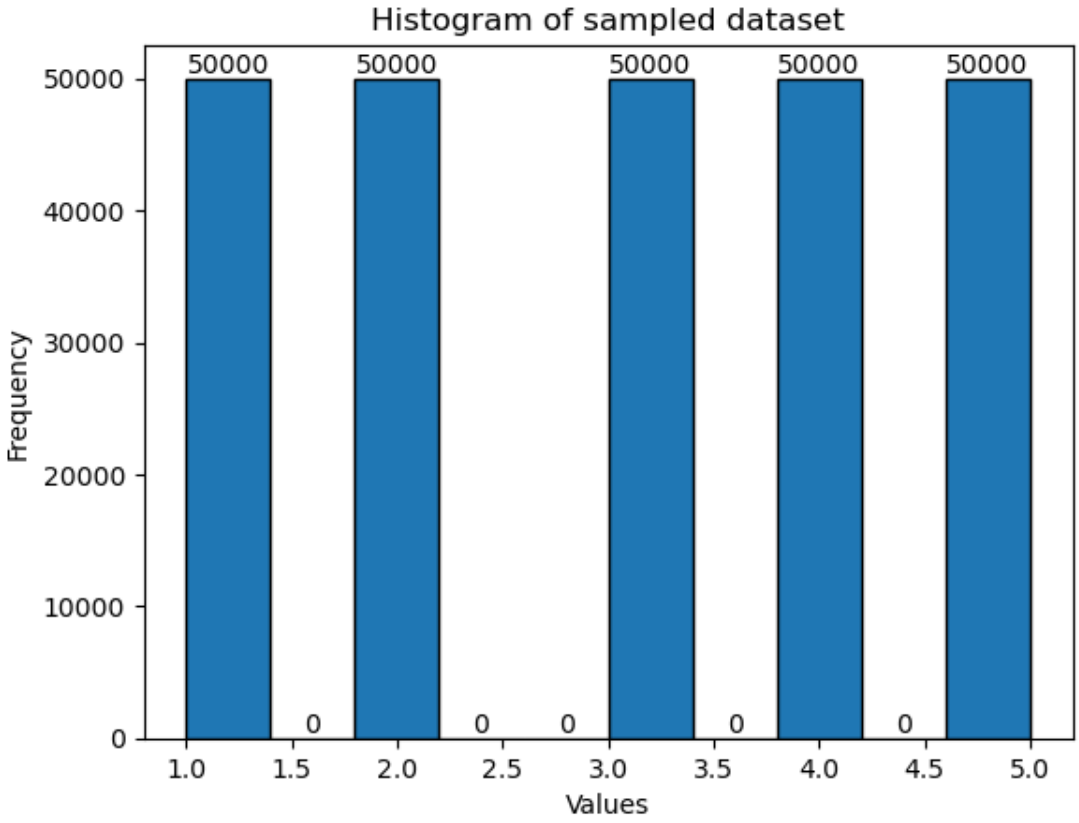
Out[25]: Text(0, 0.5, 'Frequency')



In [27]:
```
values_to_stratify = [1,2,3,4,5]

def stratified_sample(group, n=50000):
    return group.sample(n=n, random_state=1)



sampled_data = sampled_data[sampled_data['star_rating'].isin(values_to_stratify)

sampled_data = sampled_data[sampled_data['review_body'].apply(lambda x: isinsta
```

In [28]:
```
ax = sampled_data['star_rating'].plot(kind='hist', bins=10, edgecolor='black')
for p in ax.patches:
    ax.annotate(str(int(p.get_height())), (p.get_x() + p.get_width() / 2., p.ge
                ha='center', va='center', xytext=(0, 5), textcoords='offset poi
plt.title('Histogram of sampled dataset')
plt.xlabel('Values')
plt.ylabel('Frequency')
```

Out[28]: Text(0, 0.5, 'Frequency')



In [29]:
```
sampled_data.to_csv('dataset.csv', index=False)
```

In [ ]:
```

```

In [1]:
```
import pandas as pd
import numpy as np
import nltk
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[1]: True

In [2]:
```
raw_data = pd.read_csv('./dataset.csv', sep=',')
```

Rerating review AND data preparing

In [3]:
```
for index in range(len(raw_data)):
    if raw_data.at[index, 'star_rating'] > 3:
        raw_data.at[index, 'star_rating'] = 1
    elif raw_data.at[index, 'star_rating'] == 3:
        raw_data.at[index, 'star_rating'] = 2
    else:
        raw_data.at[index, 'star_rating'] = 0
```

In [4]:
```
from bs4 import BeautifulSoup
import re
# data cleaning
raw_data['review_body'] = raw_data['review_body'].str.lower()
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: re.sub(r'http\S+', '', Beauti
def remove_non_alphabetical(text):
    return re.sub(r'[^a-zA-Z\s]', '', text)

raw_data['review_body'] = raw_data['review_body'].apply(remove_non_alphabetical)
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: ' '.join(x.split()))
contraction_dict = contraction_dict = {
    "don't": "do not",
    "doesn't": "does not",
    "didn't": "did not",
    "can't": "cannot",
    "won't": "will not",
    "isn't": "is not",
    "haven't": "have not",
    "hasn't": "has not",
    "hadn't": "had not",
    "you're": "you are",
    "you've": "you have",
    "you'll": "you will",
    "when's": "when is",
    "let's": "let us",
    "'cause": "because",
    "shouldn't": "should not",
    "wouldn't": "would not",
    "couldn't": "could not",
    "wasn't": "was not",
    "weren't": "were not",
    "I'm": "I am",
    "I've": "I have",
    "I'll": "I will",
    "it's": "it is",
    "that's": "that is",
    "who's": "who is",
    "what's": "what is",
    "where's": "where is",
    "we're": "we are",
    "we've": "we have",
    "we'll": "we will",
    "they're": "they are",
    "they've": "they have",
    "they'll": "they will",
    "she's": "she is",
    "he's": "he is",
    "how's": "how is",
    "you'd": "you would",
    "we'd": "we would",
    "they'd": "they would",
}
raw_data['review_body'] = raw_data['review_body'].replace(contraction_dict, regex=True)
```

```
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\ipykernel_launcher.py:5: MarkupResemblesLoc
atorWarning: The input looks more like a filename than markup. You may want to open this file a
nd pass the filehandle into Beautiful Soup.
  """

C:\Users\monkeydc\.conda\envs\561\lib\site-packages\ipykernel_launcher.py:5: MarkupResemblesLoc
atorWarning: The input looks more like a URL than markup. You may want to use an HTTP client li
ke requests to get the document behind the URL, and feed that document to Beautiful Soup.
  """
```

In [5]:
```
from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
def remove_stop_words(text):
    words = text.split(" ")
    filtered_words = [word for word in words if word.lower() not in stop_words]
    return ' '.join(filtered_words)
raw_data['review_body'] = raw_data['review_body'].apply(remove_stop_words)
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
nltk.download('omw-1.4') # I have to download this dataset
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: ' '.join([lemmatizer.lemmatiz
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

In [6]:
```
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
import nltk
```

Need to import String

In [7]:
```
# train on personal dataset
import string
def preprocess_text(text):
    tokens = word_tokenize(text)
    stop_words = set(stopwords.words('english') + list(string.punctuation))
    tokens = [word.lower() for word in tokens if word.isalpha() and word.lower() not in stop_w
    return tokens
raw_data['clean_reviw'] = raw_data['review_body'].apply(preprocess_text)
# better word2vec
# model = Word2Vec(sentences=raw_data['clean_reviw'], vector_size=300, window=5, min_count=1, wo
```

train self W2V and Save model(named word2vec_model_b.txt)

In [8]:
```
model = Word2Vec(sentences=raw_data['clean_reviw'], vector_size=300, window=11, min_count=10, wo
```

In [9]:
```
model.wv.save_word2vec_format('word2vec_model_b.txt', binary=False)
```

In [10]:
```
# every individual jupyter notebook need to use this .txt to reload self W2V
```

Compare google pretrain and self-train W2V in word similarity

In [11]:
```
import gensim
model = gensim.models.KeyedVectors.load_word2vec_format('word2vec_model_b.txt', binary=False)
```

In [12]:
```
import numpy as np
def simple_similarity(vector1, vector2):
    dot_product = np.dot(vector1, vector2)
    norm_vector1 = np.linalg.norm(vector1)
    norm_vector2 = np.linalg.norm(vector2)
    similarity = dot_product / (norm_vector1 * norm_vector2)
    return similarity
```

In [18]:
```
print("Similariy between water and rock : ", simple_similarity(model['water'],model['rock']))
print("Similariy between ice and rock : ", simple_similarity(model['ice'],model['rock']))
print("Similariy between apple and iphone : ", simple_similarity(model['apple'],model['iphone']))
print("Similariy between apple and ipad : ", simple_similarity(model['apple'],model['ipad']))
```

```
Similariy between water and rock :  0.10122213
Similariy between ice and rock :  0.234708
Similariy between apple and iphone :  0.59669954
Similariy between apple and ipad :  0.55011463
```

In [14]:
```
import gensim.downloader as api
wv = api.load('word2vec-google-news-300')
```

In [15]:
```
# Word Embedding Similarity
```

In [17]:
```
# Example on word2vec-google-news-300
print("Similariy between water and rock : ", wv.similarity('water', 'rock'))
print("Similariy between ice and rock : ", wv.similarity('ice', 'rock'))
print("Similariy between apple and iphone : ", wv.similarity('apple','iphone'))
print("Similariy between apple and ipad : ", wv.similarity('apple','ipad'))
```

```
Similariy between water and rock :  0.1448098
Similariy between ice and rock :  0.26914403
Similariy between apple and iphone :  0.3199041
Similariy between apple and ipad :  0.35862675
```

SVM & percepton

In [18]:
```
# binary dataset
values_to_stratify = [0,1]
def stratified_sample(group, n=100000):
    return group.sample(n=n, random_state=1)

binary_data= raw_data[raw_data['star_rating'].isin(values_to_stratify)].groupby('star_rating', g
```

In [19]:
```
def average_word2vec(text, model=None, vw=None):
    if model:
        word_vectors = [model[word] for word in text if word in model.key_to_index]
    if vw:
        word_vectors = [wv[word] for word in text if word in wv.key_to_index]
    if word_vectors:
        return np.mean(word_vectors, axis=0)
    else:
        return np.zeros(300)
```

In [20]:
```
binary_data['word2vec_features_google'] = binary_data['clean_reviw'].apply(lambda x: average_wo
```

In [22]:
```
binary_data['word2vec_features_self'] = binary_data['clean_reviw'].apply(lambda x: average_word
```

In [24]:
```
from sklearn.utils import shuffle
shuffled_dataset = shuffle(binary_data, random_state=42)
```

In [25]:
```
from sklearn.model_selection import train_test_split
X_train_s, X_test_s, y_train_s, y_test_s = train_test_split(shuffled_dataset['word2vec_features_
X_train_g, X_test_g, y_train_g, y_test_g = train_test_split(shuffled_dataset['word2vec_features_
```

google + SVM/percepton

In [26]:
```
from sklearn.linear_model import Perceptron
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
perceptron_model = Perceptron()
perceptron_model.fit(X_train_g, y_train_g)
y_pred_perceptron = perceptron_model.predict(X_test_g)
accuracy_perceptron_g = accuracy_score(y_test_g, y_pred_perceptron)

# Train SVM model
svm_model = LinearSVC()
svm_model.fit(X_train_g, y_train_g)
y_pred_svm = svm_model.predict(X_test_g)
accuracy_svm_g = accuracy_score(y_test_g, y_pred_svm)
```

In [27]:
```
print(accuracy_perceptron_g)
print(accuracy_svm_g)
```

```
0.7609
0.816
```

self + SVM/percepton

In [28]:
```
from sklearn.linear_model import Perceptron
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
perceptron_model = Perceptron()
perceptron_model.fit(X_train_s, y_train_s)
y_pred_perceptron = perceptron_model.predict(X_test_s)
accuracy_perceptron = accuracy_score(y_test_s, y_pred_perceptron)

# Train SVM model
svm_model = LinearSVC()
svm_model.fit(X_train_s, y_train_s)
y_pred_svm = svm_model.predict(X_test_s)
accuracy_svm = accuracy_score(y_test_s, y_pred_svm)
print(accuracy_perceptron)
print(accuracy_svm)
```

```
0.724825
0.84435

C:\Users\monkeydc\.conda\envs\561\lib\site-packages\sklearn\svm\_base.py:1208: ConvergenceWarni
ng: Liblinear failed to converge, increase the number of iterations.
  ConvergenceWarning,
```

Useless code (remove to different .in)

In [29]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.model_selection import train_test_split
import numpy as np
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import string
```

```
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\tqdm\auto.py:22: TqdmWarning: IProgress not
found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/us
er_install.html (https://ipywidgets.readthedocs.io/en/stable/user_install.html)
  from .autonotebook import tqdm as notebook_tqdm
```

In [30]:
```
X_train_tensor_b_s = torch.FloatTensor(X_train_s)
y_train_tensor_b_s = torch.LongTensor(y_train_s.values)
X_test_tensor_b_s = torch.FloatTensor(X_test_s)
y_test_tensor_b_s = torch.LongTensor(y_test_s.values)
```

```
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\ipykernel_launcher.py:1: UserWarning: Creat
ing a tensor from a list of numpy.ndarrays is extremely slow. Please consider converting the li
st to a single numpy.ndarray with numpy.array() before converting to a tensor. (Triggered inter
nally at C:\actions-runner\_work\pytorch\pytorch\builder\windows\pytorch\torch\csrc\utils\tenso
r_new.cpp:233.)
  """Entry point for launching an IPython kernel.
```

In [31]:
```
X_train_tensor_b_g = torch.FloatTensor(X_train_g)
y_train_tensor_b_g = torch.LongTensor(y_train_g.values)
X_test_tensor_b_g = torch.FloatTensor(X_test_g)
y_test_tensor_b_g = torch.LongTensor(y_test_g.values)
```

In [32]:
```
input_size = 300
hidden_size1 = 50
hidden_size2 = 10
output_size_binary = 2
output_size_ternary = 3
```

In [33]:
```
# model
class MyMLP(nn.Module):
    def __init__(self, input_size, hidden_size1, hidden_size2, output_size):
        super(MyMLP, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size1)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size1, hidden_size2)
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(hidden_size2, output_size)

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu1(x)
        x = self.fc2(x)
        x = self.relu2(x)
        x = self.fc3(x)
        return x
```

In [34]:
```
# binary
```

In [36]:
```
# google + FNN
mlp_binary = MyMLP(input_size, hidden_size1, hidden_size2, output_size_binary)
criterion = nn.CrossEntropyLoss()
optimizer_binary = optim.Adam(mlp_binary.parameters(), lr=0.001)
for epoch in range(1500):
    outputs = mlp_binary(X_train_tensor_b_s)
    loss = criterion(outputs, y_train_tensor_b_s)
    optimizer_binary.zero_grad()
    loss.backward()
    optimizer_binary.step()
```

In [37]:
```
with torch.no_grad():
    mlp_binary.eval()
    y_pred_binary_s = torch.argmax(mlp_binary(X_test_tensor_b_s), dim=1)
    accuracy_binary_s = torch.sum(y_pred_binary_s == y_test_tensor_b_s).item() / len(y_test_tens
```

In [38]:
```
print(accuracy_binary_s)
```

```
0.861575
```

In [39]:
```
# self + FNN
mlp_binary = MyMLP(input_size, hidden_size1, hidden_size2, output_size_binary)
criterion = nn.CrossEntropyLoss()
optimizer_binary = optim.Adam(mlp_binary.parameters(), lr=0.001)
for epoch in range(1500):
    outputs = mlp_binary(X_train_tensor_b_g)
    loss = criterion(outputs, y_train_tensor_b_g)
    optimizer_binary.zero_grad()
    loss.backward()
    optimizer_binary.step()
with torch.no_grad():
    mlp_binary.eval()
    y_pred_binary_g = torch.argmax(mlp_binary(X_test_tensor_b_g), dim=1)
    accuracy_binary_g = torch.sum(y_pred_binary_g == y_test_tensor_b_g).item() / len(y_test_tens
print(accuracy_binary_g)
```

0.83945

In [40]:
```
# self+CNN # different dimension with FNN
```

In [1]:

```
import pandas as pd
import numpy as np
import nltk
nltk.download('wordnet')
raw_data = pd.read_csv('./dataset.csv', sep=',')
for index in range(len(raw_data)):
    if raw_data.at[index, 'star_rating'] > 3:
        raw_data.at[index, 'star_rating'] = 1
    elif raw_data.at[index, 'star_rating'] == 3:
        raw_data.at[index, 'star_rating'] = 2
    else:
        raw_data.at[index, 'star_rating'] = 0
from bs4 import BeautifulSoup
import re
# data cleaning
raw_data['review_body'] = raw_data['review_body'].str.lower()
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: re.sub(r'http
def remove_non_alphabetical(text):
    return re.sub(r'[^a-zA-Z\s]', '', text)

raw_data['review_body'] = raw_data['review_body'].apply(remove_non_alphabetical)
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: ' '.join(x.sp
contraction_dict = contraction_dict = {
    "don't": "do not",
    "doesn't": "does not",
    "didn't": "did not",
    "can't": "cannot",
    "won't": "will not",
    "isn't": "is not",
    "haven't": "have not",
    "hasn't": "has not",
    "hadn't": "had not",
    "you're": "you are",
    "you've": "you have",
    "you'll": "you will",
    "when's": "when is",
    "let's": "let us",
    "'cause": "because",
    "shouldn't": "should not",
    "wouldn't": "would not",
    "couldn't": "could not",
    "wasn't": "was not",
    "weren't": "were not",
    "I'm": "I am",
    "I've": "I have",
    "I'll": "I will",
    "it's": "it is",
    "that's": "that is",
    "who's": "who is",
    "what's": "what is",
    "where's": "where is",
    "we're": "we are",
    "we've": "we have",
    "we'll": "we will",
    "they're": "they are",
    "they've": "they have",
    "they'll": "they will",
    "she's": "she is",
    "he's": "he is",
    "how's": "how is",
    "you'd": "you would",
```

```
    "we'd": "we would",
    "they'd": "they would",
}
raw_data['review_body'] = raw_data['review_body'].replace(contraction_dict, rege
from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
def remove_stop_words(text):
    words = text.split(" ")
    filtered_words = [word for word in words if word.lower() not in stop_words
    return ' '.join(filtered_words)
raw_data['review_body'] = raw_data['review_body'].apply(remove_stop_words)
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
nltk.download('omw-1.4') # I have to download this dataset
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: ' '.join([lem
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
import nltk
import string
def preprocess_text(text):
    tokens = word_tokenize(text)
    stop_words = set(stopwords.words('english') + list(string.punctuation))
    tokens = [word.lower() for word in tokens if word.isalpha() and word.lower
    return tokens
raw_data['clean_reviw'] = raw_data['review_body'].apply(preprocess_text)
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\ipykernel_launcher.py:17:
MarkupResemblesLocatorWarning: The input looks more like a filename than marku
p.  You may want to open this file and pass the filehandle into Beautiful Soup.
  app.launch_new_instance()
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\ipykernel_launcher.py:17:
MarkupResemblesLocatorWarning: The input looks more like a URL than markup.  Yo
u may want to use an HTTP client like requests to get the document behind the
URL, and feed that document to Beautiful Soup.
  app.launch_new_instance()
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

In [2]:

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

In [3]:
```
# binary dataset
values_to_stratify = [0,1]
def stratified_sample(group, n=100000):
    return group.sample(n=n, random_state=1)

binary_data= raw_data[raw_data['star_rating'].isin(values_to_stratify)].groupby

def average_word2vec(text, model=None, vw=None):
    if model:
        word_vectors = [model[word] for word in text if word in model.key_to_i
    if vw:
        word_vectors = [wv[word] for word in text if word in wv.key_to_index]
    if word_vectors:
        return np.mean(word_vectors, axis=0)
    else:
        return np.zeros(300)

binary_data['word2vec_features_google'] = binary_data['clean_reviw'].apply(lamb
```

In [4]:
```
from sklearn.utils import shuffle
shuffled_dataset = shuffle(binary_data, random_state=42)
from sklearn.model_selection import train_test_split
X_train_g, X_test_g, y_train_g, y_test_g = train_test_split(shuffled_dataset['wo
```

Binary

In [5]:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.model_selection import train_test_split
import numpy as np
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import string

X_train_tensor_b_g = torch.FloatTensor(X_train_g)
y_train_tensor_b_g = torch.LongTensor(y_train_g.values)
X_test_tensor_b_g = torch.FloatTensor(X_test_g)
y_test_tensor_b_g = torch.LongTensor(y_test_g.values)

input_size = 300
hidden_size1 = 50
hidden_size2 = 10
output_size_binary = 2
output_size_ternary = 3

# model
class MyMLP(nn.Module):
    def __init__(self, input_size, hidden_size1, hidden_size2, output_size):
        super(MyMLP, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size1)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size1, hidden_size2)
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(hidden_size2, output_size)

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu1(x)
        x = self.fc2(x)
        x = self.relu2(x)
        x = self.fc3(x)
        return x

mlp_binary = MyMLP(input_size, hidden_size1, hidden_size2, output_size_binary)
criterion = nn.CrossEntropyLoss()
optimizer_binary = optim.Adam(mlp_binary.parameters(), lr=0.001)
for epoch in range(1500):
    outputs = mlp_binary(X_train_tensor_b_g)
    loss = criterion(outputs, y_train_tensor_b_g)
    optimizer_binary.zero_grad()
    loss.backward()
    optimizer_binary.step()
with torch.no_grad():
    mlp_binary.eval()
    y_pred_binary_g = torch.argmax(mlp_binary(X_test_tensor_b_g), dim=1)
    accuracy_binary_g = torch.sum(y_pred_binary_g == y_test_tensor_b_g).item()
print(accuracy_binary_g)
```

```
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\tqdm\auto.py:22: TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html (https://ipywidgets.readthedocs.io/en/stable/user_install.html)
  from .autonotebook import tqdm as notebook_tqdm
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\ipykernel_launcher.py:12: UserWarning: Creating a tensor from a list of numpy.ndarrays is extremely slow. Please consider converting the list to a single numpy.ndarray with numpy.array() before converting to a tensor. (Triggered internally at C:\actions-runner\_work\pytorch\pytorch\builder\windows\pytorch\torch\csrc\utils\tensor_new.cpp:233.)
  if sys.path[0] == "":
```

```
0.84285
```

Ternary

In [6]:

```
# ternary dataset

def average_word2vec(text, model=None, vw=None):
    if model:
        word_vectors = [model[word] for word in text if word in model.key_to_i
    if vw:
        word_vectors = [wv[word] for word in text if word in wv.key_to_index]
    if word_vectors:
        return np.mean(word_vectors, axis=0)
    else:
        return np.zeros(300)

raw_data['word2vec_features_google'] = raw_data['clean_reviw'].apply(lambda x:

from sklearn.utils import shuffle
shuffled_dataset = shuffle(raw_data, random_state=42)
from sklearn.model_selection import train_test_split
X_train_g, X_test_g, y_train_g, y_test_g = train_test_split(shuffled_dataset['wo

X_train_tensor_t_g = torch.FloatTensor(X_train_g)
y_train_tensor_t_g = torch.LongTensor(y_train_g.values)
X_test_tensor_t_g = torch.FloatTensor(X_test_g)
y_test_tensor_t_g = torch.LongTensor(y_test_g.values)

mlp_ternary = MyMLP(input_size, hidden_size1, hidden_size2, output_size_ternary)
criterion = nn.CrossEntropyLoss()
optimizer_ternary = optim.Adam(mlp_ternary.parameters(), lr=0.001)
for epoch in range(1500):
    outputs = mlp_ternary(X_train_tensor_t_g)
    loss = criterion(outputs, y_train_tensor_t_g)
    optimizer_ternary.zero_grad()
    loss.backward()
    optimizer_ternary.step()
with torch.no_grad():
    mlp_ternary.eval()
    y_pred_ternary_g = torch.argmax(mlp_ternary(X_test_tensor_t_g), dim=1)
    accuracy_ternary_g = torch.sum(y_pred_ternary_g == y_test_tensor_t_g).item(
print(accuracy_ternary_g)
```

0.6847

In [1]:

```
import pandas as pd
import numpy as np
import nltk
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.model_selection import train_test_split
import numpy as np
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
nltk.download('wordnet')
raw_data = pd.read_csv('./dataset.csv', sep=',')
for index in range(len(raw_data)):
    if raw_data.at[index, 'star_rating'] > 3:
        raw_data.at[index, 'star_rating'] = 1
    elif raw_data.at[index, 'star_rating'] == 3:
        raw_data.at[index, 'star_rating'] = 2
    else:
        raw_data.at[index, 'star_rating'] = 0
from bs4 import BeautifulSoup
import re
# data cleaning
raw_data['review_body'] = raw_data['review_body'].str.lower()
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: re.sub(r'http
def remove_non_alphabetical(text):
    return re.sub(r'[^a-zA-Z\s]', '', text)

raw_data['review_body'] = raw_data['review_body'].apply(remove_non_alphabetical)
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: ' '.join(x.sp
contraction_dict = contraction_dict = {
    "don't": "do not",
    "doesn't": "does not",
    "didn't": "did not",
    "can't": "cannot",
    "won't": "will not",
    "isn't": "is not",
    "haven't": "have not",
    "hasn't": "has not",
    "hadn't": "had not",
    "you're": "you are",
    "you've": "you have",
    "you'll": "you will",
    "when's": "when is",
    "let's": "let us",
    "'cause": "because",
    "shouldn't": "should not",
    "wouldn't": "would not",
    "couldn't": "could not",
    "wasn't": "was not",
    "weren't": "were not",
    "I'm": "I am",
    "I've": "I have",
    "I'll": "I will",
    "it's": "it is",
    "that's": "that is",
    "who's": "who is",
    "what's": "what is",
    "where's": "where is",
    "we're": "we are",
```

```
    "we've": "we have",
    "we'll": "we will",
    "they're": "they are",
    "they've": "they have",
    "they'll": "they will",
    "she's": "she is",
    "he's": "he is",
    "how's": "how is",
    "you'd": "you would",
    "we'd": "we would",
    "they'd": "they would",
}
raw_data['review_body'] = raw_data['review_body'].replace(contraction_dict, rege
from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
def remove_stop_words(text):
    words = text.split(" ")
    filtered_words = [word for word in words if word.lower() not in stop_words
    return ' '.join(filtered_words)
raw_data['review_body'] = raw_data['review_body'].apply(remove_stop_words)
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
nltk.download('omw-1.4') # I have to download this dataset
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: ' '.join([lem
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
import nltk
import string
def preprocess_text(text):
    tokens = word_tokenize(text)
    stop_words = set(stopwords.words('english') + list(string.punctuation))
    tokens = [word.lower() for word in tokens if word.isalpha() and word.lower
    return tokens
raw_data['clean_reviw'] = raw_data['review_body'].apply(preprocess_text)
```

```
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\tqdm\auto.py:22: TqdmWarni
ng: IProgress not found. Please update jupyter and ipywidgets. See https://ipy
widgets.readthedocs.io/en/stable/user_install.html (https://ipywidgets.readthe
docs.io/en/stable/user_install.html)
  from .autonotebook import tqdm as notebook_tqdm
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\ipykernel_launcher.py:26:
MarkupResemblesLocatorWarning: The input looks more like a filename than marku
p. You may want to open this file and pass the filehandle into Beautiful Soup.
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\ipykernel_launcher.py:26:
MarkupResemblesLocatorWarning: The input looks more like a URL than markup. Yo
u may want to use an HTTP client like requests to get the document behind the
URL, and feed that document to Beautiful Soup.
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

In [2]:
```
import gensim.downloader as api
wv = api.load('word2vec-google-news-300')
```

In [3]:
```
from sklearn.utils import shuffle
values_to_stratify = [0,1]
def stratified_sample(group, n=100000):
    return group.sample(n=n, random_state=1)

binary_data = raw_data[raw_data['star_rating'].isin(values_to_stratify)].groupby
shuffled_dataset = shuffle(binary_data, random_state=42)
from sklearn.model_selection import train_test_split
X_train_g = shuffled_dataset[:160000]
X_test_g = shuffled_dataset[160000:]
```

In [4]:
```
# def average_word2vec(text, model=None, vw=None):
#     if model:
#         word_vectors = [model[word] for word in text if word in model.key_to_i
#     if vw:
#         word_vectors = [wv[word] for word in text if word in wv.key_to_index]
#     if word_vectors:
#         return np.mean(word_vectors, axis=0)
#     else:
#         return np.zeros(300)

def concatenate_word_vectors(sentence, max_length=10, vw=None,  model=None):
    concatenated_vectors = []
    if vw:
        for i in range(max_length):
            if i < len(sentence):
                if sentence[i] in vw.key_to_index:
                    tmp_vectors = vw[sentence[i]]
                else:
                    tmp_vectors = np.zeros(300,dtype=np.float16)
                concatenated_vectors.extend(tmp_vectors)
            else:
                concatenated_vectors.extend(np.zeros(300,dtype=np.float16))
    else:
        for i in range(max_length):
            if i < len(sentence):
                if sentence[i] in model.key_to_index:
                    tmp_vectors = model[sentence[i]]
                else:
                    tmp_vectors = np.zeros(300,dtype=np.float16)
                concatenated_vectors.extend(tmp_vectors)
            else:
                concatenated_vectors.extend(np.zeros(300,dtype=np.float16))
    return concatenated_vectors
```

In [8]:
```
wv.vectors = wv.vectors.astype(np.float16)
```

In [9]:
```
from torch.utils.data import Dataset
class SentimentDataset(Dataset):
    def __init__(self, X, y):
        self.X = X
        self.y = y

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

    def __getitem__(self, index):
        features = torch.tensor(self.X[index], dtype=torch.float16)
        label = torch.tensor(self.y[index], dtype=torch.long)
        return features, label
```

In [10]:
```
X_train_tensor_b_g = X_train_g['clean_reviw'].apply(lambda x: concatenate_word_
y_train_binary_list = X_train_g['star_rating'].tolist()
train_dataset = SentimentDataset(X_train_tensor_b_g, y_train_binary_list)
```

In [11]:
```
X_test_tensor_b_g = X_test_g['clean_reviw'].apply(lambda x: concatenate_word_ve
y_test_binary_list = X_test_g['star_rating'].tolist()
test_dataset = SentimentDataset(X_test_tensor_b_g, y_test_binary_list)
```

In [22]:
```
# Define the MLP model
class SentimentClassifier(nn.Module):
    def __init__(self, input_size, hidden_size1, hidden_size2, output_size):
        super(SentimentClassifier, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size1).float()
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size1, hidden_size2).float()
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(hidden_size2, output_size).float()

    def forward(self, x):
        x = self.relu1(self.fc1(x))
        x = self.relu2(self.fc2(x))
        x = self.fc3(x)
        return x


input_size = 3000
hidden_size1 = 50
hidden_size2 = 10
output_size_binary = 2

batch_size = 64

model_binary = SentimentClassifier(input_size, hidden_size1, hidden_size2, outpu

criterion = nn.CrossEntropyLoss()
optimizer_binary = optim.Adam(model_binary.parameters(), lr=0.001)
```

In [13]:
```
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

In [24]:
```
print(model_binary.fc1.weight.dtype)
```

```
torch.float32
```

In [25]:
```
# Training loop
num_epochs = 5
for epoch in range(num_epochs):
    model_binary.train()
    for inputs, labels in train_loader:
        inputs = inputs.view(inputs.size(0), -1).float()
        outputs_binary = model_binary(inputs)
        loss_binary = criterion(outputs_binary, labels)

        # Backward and optimize
        optimizer_binary.zero_grad()
        loss_binary.backward()
        optimizer_binary.step()
```

In [27]:
```
from sklearn.metrics import accuracy_score
model_binary.eval()
predictions_binary = []
true_labels_binary = []
with torch.no_grad():
    for inputs, labels in test_loader:
        inputs = inputs.view(inputs.size(0), -1).float()
        outputs_binary = model_binary(inputs)
        _, predicted_binary = torch.max(outputs_binary, 1)
        predictions_binary.extend(predicted_binary.numpy())
        true_labels_binary.extend(labels.numpy())
accuracy_binary = accuracy_score(true_labels_binary, predictions_binary)
print(accuracy_binary)
```

```
0.753675
```

In [ ]:

In [ ]:

In [ ]:

In [1]:

```
import pandas as pd
import numpy as np
import nltk
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.model_selection import train_test_split
import numpy as np
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
nltk.download('wordnet')
raw_data = pd.read_csv('./dataset.csv', sep=',')
for index in range(len(raw_data)):
    if raw_data.at[index, 'star_rating'] > 3:
        raw_data.at[index, 'star_rating'] = 1
    elif raw_data.at[index, 'star_rating'] == 3:
        raw_data.at[index, 'star_rating'] = 2
    else:
        raw_data.at[index, 'star_rating'] = 0
from bs4 import BeautifulSoup
import re
# data cleaning
raw_data['review_body'] = raw_data['review_body'].str.lower()
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: re.sub(r'http
def remove_non_alphabetical(text):
    return re.sub(r'[^a-zA-Z\s]', '', text)

raw_data['review_body'] = raw_data['review_body'].apply(remove_non_alphabetical)
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: ' '.join(x.sp
contraction_dict = contraction_dict = {
    "don't": "do not",
    "doesn't": "does not",
    "didn't": "did not",
    "can't": "cannot",
    "won't": "will not",
    "isn't": "is not",
    "haven't": "have not",
    "hasn't": "has not",
    "hadn't": "had not",
    "you're": "you are",
    "you've": "you have",
    "you'll": "you will",
    "when's": "when is",
    "let's": "let us",
    "'cause": "because",
    "shouldn't": "should not",
    "wouldn't": "would not",
    "couldn't": "could not",
    "wasn't": "was not",
    "weren't": "were not",
    "I'm": "I am",
    "I've": "I have",
    "I'll": "I will",
    "it's": "it is",
    "that's": "that is",
    "who's": "who is",
    "what's": "what is",
    "where's": "where is",
    "we're": "we are",
```

```
    "we've": "we have",
    "we'll": "we will",
    "they're": "they are",
    "they've": "they have",
    "they'll": "they will",
    "she's": "she is",
    "he's": "he is",
    "how's": "how is",
    "you'd": "you would",
    "we'd": "we would",
    "they'd": "they would",
}
raw_data['review_body'] = raw_data['review_body'].replace(contraction_dict, rege
from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
def remove_stop_words(text):
    words = text.split(" ")
    filtered_words = [word for word in words if word.lower() not in stop_words
    return ' '.join(filtered_words)
raw_data['review_body'] = raw_data['review_body'].apply(remove_stop_words)
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
nltk.download('omw-1.4') # I have to download this dataset
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: ' '.join([lem
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
import nltk
import string
def preprocess_text(text):
    tokens = word_tokenize(text)
    stop_words = set(stopwords.words('english') + list(string.punctuation))
    tokens = [word.lower() for word in tokens if word.isalpha() and word.lower
    return tokens
raw_data['clean_reviw'] = raw_data['review_body'].apply(preprocess_text)
```

```
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\tqdm\auto.py:22: TqdmWarni
ng: IProgress not found. Please update jupyter and ipywidgets. See https://ipy
widgets.readthedocs.io/en/stable/user_install.html (https://ipywidgets.readthe
docs.io/en/stable/user_install.html)
  from .autonotebook import tqdm as notebook_tqdm
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\ipykernel_launcher.py:26:
MarkupResemblesLocatorWarning: The input looks more like a filename than marku
p. You may want to open this file and pass the filehandle into Beautiful Soup.
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\ipykernel_launcher.py:26:
MarkupResemblesLocatorWarning: The input looks more like a URL than markup. Yo
u may want to use an HTTP client like requests to get the document behind the
URL, and feed that document to Beautiful Soup.
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

In [2]:
```
import gensim.downloader as api
wv = api.load('word2vec-google-news-300')
```

In [3]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

In [4]:
```
from torch.utils.data import Dataset
class BetterDataset(Dataset):
    def __init__(self, reviews, labels, word2vec_model):
        self.reviews = reviews
        self.labels = labels
        self.word2vec_model = word2vec_model

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

    def __getitem__(self, idx):
        review = self.reviews[idx]
        # word_vectors = []

        concatenated_vectors = np.zeros((10, 300), dtype=np.float16)
        for i in range(min(10, len(review))):
            word = review[i]
            if word in self.word2vec_model.key_to_index:
                concatenated_vectors[i] = self.word2vec_model[word]

#         for i in range(10):
#             word = review[i]
#             if word in self.word2vec_model:
#                 word_vector = self.word2vec_model[word]
#             else:
#                 word_vector = torch.zeros(300, dtype=np.float16)  # Handle out
#             word_vectors.append(word_vector)

        label = torch.tensor(self.labels[idx], dtype=torch.long)
        return torch.tensor(concatenated_vectors.flatten().tolist()), label
```

In [5]:
```
from sklearn.utils import shuffle
shuffled_dataset = shuffle(raw_data, random_state=42)
from sklearn.model_selection import train_test_split
X_train_g = shuffled_dataset[:200000]
X_test_g = shuffled_dataset[200000:]
print(len(X_train_g))
print(len(X_test_g))
```

```
200000
50000
```

In [6]:
```
X_train_g = X_train_g.reset_index(drop=True)
```

In [7]:
```
X_test_g = X_test_g.reset_index(drop=True)
```

In [8]:
```
train_dataset = BetterDataset(X_train_g['clean_reviw'], X_train_g['star_rating']
```

In [9]:
```
test_dataset = BetterDataset(X_test_g['clean_reviw'], X_test_g['star_rating'], v
```

In [10]:
```
class SentimentClassifier(nn.Module):
    def __init__(self, input_size, hidden_size1, hidden_size2, output_size):
        super(SentimentClassifier, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size1).float()
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size1, hidden_size2).float()
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(hidden_size2, output_size).float()

    def forward(self, x):
        x = self.relu1(self.fc1(x))
        x = self.relu2(self.fc2(x))
        x = self.fc3(x)
        return x


input_size = 3000
hidden_size1 = 50
hidden_size2 = 10
output_size_ternary = 3

model_binary = SentimentClassifier(input_size, hidden_size1, hidden_size2, outpu

criterion = nn.CrossEntropyLoss()
optimizer_binary = optim.Adam(model_binary.parameters(), lr=0.001)
```

In [11]:
```
batch_size = 16
```

In [12]:
```
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
```

In [13]:
```
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

In [14]:
```
num_epochs = 5
for epoch in range(num_epochs):
    model_binary.train()
    for inputs, labels in train_loader:
        inputs = inputs.view(inputs.size(0), -1).float()
        outputs_binary = model_binary(inputs.to(device))
        loss_binary = criterion(outputs_binary, labels.to(device))

        # Backward and optimize
        optimizer_binary.zero_grad()
        loss_binary.backward()
        optimizer_binary.step()
```

In [15]:
```
from sklearn.metrics import accuracy_score
model_binary.to('cpu')
model_binary.eval()
predictions_binary = []
true_labels_binary = []
with torch.no_grad():
    for inputs, labels in test_loader:
        inputs = inputs.view(inputs.size(0), -1).float()
        outputs_binary = model_binary(inputs)
        _, predicted_binary = torch.max(outputs_binary, 1)
        predictions_binary.extend(predicted_binary.numpy())
        true_labels_binary.extend(labels.numpy())
accuracy_ternary = accuracy_score(true_labels_binary, predictions_binary)
print(accuracy_ternary)
```

```
0.62222
```

In [ ]:
```

```

In [1]:

```
import pandas as pd
import numpy as np
import nltk
nltk.download('wordnet')
raw_data = pd.read_csv('./dataset.csv', sep=',')
for index in range(len(raw_data)):
    if raw_data.at[index, 'star_rating'] > 3:
        raw_data.at[index, 'star_rating'] = 1
    elif raw_data.at[index, 'star_rating'] == 3:
        raw_data.at[index, 'star_rating'] = 2
    else:
        raw_data.at[index, 'star_rating'] = 0
from bs4 import BeautifulSoup
import re
# data cleaning
raw_data['review_body'] = raw_data['review_body'].str.lower()
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: re.sub(r'http
def remove_non_alphabetical(text):
    return re.sub(r'[^a-zA-Z\s]', '', text)

raw_data['review_body'] = raw_data['review_body'].apply(remove_non_alphabetical)
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: ' '.join(x.sp
contraction_dict = contraction_dict = {
    "don't": "do not",
    "doesn't": "does not",
    "didn't": "did not",
    "can't": "cannot",
    "won't": "will not",
    "isn't": "is not",
    "haven't": "have not",
    "hasn't": "has not",
    "hadn't": "had not",
    "you're": "you are",
    "you've": "you have",
    "you'll": "you will",
    "when's": "when is",
    "let's": "let us",
    "'cause": "because",
    "shouldn't": "should not",
    "wouldn't": "would not",
    "couldn't": "could not",
    "wasn't": "was not",
    "weren't": "were not",
    "I'm": "I am",
    "I've": "I have",
    "I'll": "I will",
    "it's": "it is",
    "that's": "that is",
    "who's": "who is",
    "what's": "what is",
    "where's": "where is",
    "we're": "we are",
    "we've": "we have",
    "we'll": "we will",
    "they're": "they are",
    "they've": "they have",
    "they'll": "they will",
    "she's": "she is",
    "he's": "he is",
    "how's": "how is",
    "you'd": "you would",
```

```
    "we'd": "we would",
    "they'd": "they would",
}
raw_data['review_body'] = raw_data['review_body'].replace(contraction_dict, rege
from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
def remove_stop_words(text):
    words = text.split(" ")
    filtered_words = [word for word in words if word.lower() not in stop_words
    return ' '.join(filtered_words)
raw_data['review_body'] = raw_data['review_body'].apply(remove_stop_words)
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
nltk.download('omw-1.4') # I have to download this dataset
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: ' '.join([lem
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
import nltk
import string
def preprocess_text(text):
    tokens = word_tokenize(text)
    stop_words = set(stopwords.words('english') + list(string.punctuation))
    tokens = [word.lower() for word in tokens if word.isalpha() and word.lower
    return tokens
raw_data['clean_reviw'] = raw_data['review_body'].apply(preprocess_text)
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\ipykernel_launcher.py:17:
MarkupResemblesLocatorWarning: The input looks more like a filename than marku
p.  You may want to open this file and pass the filehandle into Beautiful Soup.
  app.launch_new_instance()
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\ipykernel_launcher.py:17:
MarkupResemblesLocatorWarning: The input looks more like a URL than markup.  Yo
u may want to use an HTTP client like requests to get the document behind the
URL, and feed that document to Beautiful Soup.
  app.launch_new_instance()
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

In [2]:

```
import gensim
model = gensim.models.KeyedVectors.load_word2vec_format('word2vec_model_b.txt',
```

In [3]:
```
# binary dataset
values_to_stratify = [0,1]
def stratified_sample(group, n=100000):
    return group.sample(n=n, random_state=1)

binary_data= raw_data[raw_data['star_rating'].isin(values_to_stratify)].groupby

def average_word2vec(text, model=None, vw=None):
    if model:
        word_vectors = [model[word] for word in text if word in model.key_to_i
    if vw:
        word_vectors = [wv[word] for word in text if word in wv.key_to_index]
    if word_vectors:
        return np.mean(word_vectors, axis=0)
    else:
        return np.zeros(300)
binary_data['word2vec_features_self'] = binary_data['clean_reviw'].apply(lambda
```

In [4]:
```
from sklearn.utils import shuffle
shuffled_dataset = shuffle(binary_data, random_state=42)
from sklearn.model_selection import train_test_split
X_train_s, X_test_s, y_train_s, y_test_s = train_test_split(shuffled_dataset['wo
```

Binary

In [5]:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.model_selection import train_test_split
import numpy as np
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import string

X_train_tensor_b_s = torch.FloatTensor(X_train_s)
y_train_tensor_b_s = torch.LongTensor(y_train_s.values)
X_test_tensor_b_s = torch.FloatTensor(X_test_s)
y_test_tensor_b_s = torch.LongTensor(y_test_s.values)

input_size = 300
hidden_size1 = 50
hidden_size2 = 10
output_size_binary = 2
output_size_ternary = 3

# model
class MyMLP(nn.Module):
    def __init__(self, input_size, hidden_size1, hidden_size2, output_size):
        super(MyMLP, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size1)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size1, hidden_size2)
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(hidden_size2, output_size)

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu1(x)
        x = self.fc2(x)
        x = self.relu2(x)
        x = self.fc3(x)
        return x

mlp_binary = MyMLP(input_size, hidden_size1, hidden_size2, output_size_binary)
criterion = nn.CrossEntropyLoss()
optimizer_binary = optim.Adam(mlp_binary.parameters(), lr=0.001)
for epoch in range(1500):
    outputs = mlp_binary(X_train_tensor_b_s)
    loss = criterion(outputs, y_train_tensor_b_s)
    optimizer_binary.zero_grad()
    loss.backward()
    optimizer_binary.step()

with torch.no_grad():
    mlp_binary.eval()
    y_pred_binary_s = torch.argmax(mlp_binary(X_test_tensor_b_s), dim=1)
    accuracy_binary_s = torch.sum(y_pred_binary_s == y_test_tensor_b_s).item()

print(accuracy_binary_s)
```

```
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\tqdm\auto.py:22: TqdmWarni
ng: IProgress not found. Please update jupyter and ipywidgets. See https://ipy
widgets.readthedocs.io/en/stable/user_install.html (https://ipywidgets.readthe
docs.io/en/stable/user_install.html)
  from .autonotebook import tqdm as notebook_tqdm
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\ipykernel_launcher.py:12:
UserWarning: Creating a tensor from a list of numpy.ndarrays is extremely slo
w. Please consider converting the list to a single numpy.ndarray with numpy.ar
ray() before converting to a tensor. (Triggered internally at C:\actions-runne
r\_work\pytorch\pytorch\builder\windows\pytorch\torch\csrc\utils\tensor_new.cp
p:233.)
  if sys.path[0] == "":
```

0.8617

Ternary

In [6]:

```
def average_word2vec(text, model=None, vw=None):
    if model:
        word_vectors = [model[word] for word in text if word in model.key_to_i
    if vw:
        word_vectors = [wv[word] for word in text if word in wv.key_to_index]
    if word_vectors:
        return np.mean(word_vectors, axis=0)
    else:
        return np.zeros(300)

raw_data['word2vec_features_self'] = raw_data['clean_reviw'].apply(lambda x: av

from sklearn.utils import shuffle
shuffled_dataset = shuffle(raw_data, random_state=42)
from sklearn.model_selection import train_test_split
X_train_g, X_test_g, y_train_g, y_test_g = train_test_split(shuffled_dataset['wo

X_train_tensor_t_g = torch.FloatTensor(X_train_g)
y_train_tensor_t_g = torch.LongTensor(y_train_g.values)
X_test_tensor_t_g = torch.FloatTensor(X_test_g)
y_test_tensor_t_g = torch.LongTensor(y_test_g.values)

mlp_ternary = MyMLP(input_size, hidden_size1, hidden_size2, output_size_ternary)
criterion = nn.CrossEntropyLoss()
optimizer_ternary = optim.Adam(mlp_ternary.parameters(), lr=0.001)
for epoch in range(1500):
    outputs = mlp_ternary(X_train_tensor_t_g)
    loss = criterion(outputs, y_train_tensor_t_g)
    optimizer_ternary.zero_grad()
    loss.backward()
    optimizer_ternary.step()
with torch.no_grad():
    mlp_ternary.eval()
    y_pred_ternary_g = torch.argmax(mlp_ternary(X_test_tensor_t_g), dim=1)
    accuracy_ternary_g = torch.sum(y_pred_ternary_g == y_test_tensor_t_g).item(

print(accuracy_ternary_g)
```

0.70174

In [1]:

```
import pandas as pd
import numpy as np
import nltk
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.model_selection import train_test_split
import numpy as np
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
nltk.download('wordnet')
raw_data = pd.read_csv('./dataset.csv', sep=',')
for index in range(len(raw_data)):
    if raw_data.at[index, 'star_rating'] > 3:
        raw_data.at[index, 'star_rating'] = 1
    elif raw_data.at[index, 'star_rating'] == 3:
        raw_data.at[index, 'star_rating'] = 2
    else:
        raw_data.at[index, 'star_rating'] = 0
from bs4 import BeautifulSoup
import re
# data cleaning
raw_data['review_body'] = raw_data['review_body'].str.lower()
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: re.sub(r'http
def remove_non_alphabetical(text):
    return re.sub(r'[^a-zA-Z\s]', '', text)

raw_data['review_body'] = raw_data['review_body'].apply(remove_non_alphabetical)
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: ' '.join(x.sp
contraction_dict = contraction_dict = {
    "don't": "do not",
    "doesn't": "does not",
    "didn't": "did not",
    "can't": "cannot",
    "won't": "will not",
    "isn't": "is not",
    "haven't": "have not",
    "hasn't": "has not",
    "hadn't": "had not",
    "you're": "you are",
    "you've": "you have",
    "you'll": "you will",
    "when's": "when is",
    "let's": "let us",
    "'cause": "because",
    "shouldn't": "should not",
    "wouldn't": "would not",
    "couldn't": "could not",
    "wasn't": "was not",
    "weren't": "were not",
    "I'm": "I am",
    "I've": "I have",
    "I'll": "I will",
    "it's": "it is",
    "that's": "that is",
    "who's": "who is",
    "what's": "what is",
    "where's": "where is",
    "we're": "we are",
```

```
    "we've": "we have",
    "we'll": "we will",
    "they're": "they are",
    "they've": "they have",
    "they'll": "they will",
    "she's": "she is",
    "he's": "he is",
    "how's": "how is",
    "you'd": "you would",
    "we'd": "we would",
    "they'd": "they would",
}
raw_data['review_body'] = raw_data['review_body'].replace(contraction_dict, rege
from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
def remove_stop_words(text):
    words = text.split(" ")
    filtered_words = [word for word in words if word.lower() not in stop_words
    return ' '.join(filtered_words)
raw_data['review_body'] = raw_data['review_body'].apply(remove_stop_words)
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
nltk.download('omw-1.4') # I have to download this dataset
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: ' '.join([lem
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
import nltk
import string
def preprocess_text(text):
    tokens = word_tokenize(text)
    stop_words = set(stopwords.words('english') + list(string.punctuation))
    tokens = [word.lower() for word in tokens if word.isalpha() and word.lower
    return tokens
raw_data['clean_reviw'] = raw_data['review_body'].apply(preprocess_text)
```

```
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\tqdm\auto.py:22: TqdmWarni
ng: IProgress not found. Please update jupyter and ipywidgets. See https://ipy
widgets.readthedocs.io/en/stable/user_install.html (https://ipywidgets.readthe
docs.io/en/stable/user_install.html)
  from .autonotebook import tqdm as notebook_tqdm
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\ipykernel_launcher.py:26:
MarkupResemblesLocatorWarning: The input looks more like a filename than marku
p. You may want to open this file and pass the filehandle into Beautiful Soup.
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\ipykernel_launcher.py:26:
MarkupResemblesLocatorWarning: The input looks more like a URL than markup. Yo
u may want to use an HTTP client like requests to get the document behind the
URL, and feed that document to Beautiful Soup.
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

In [2]:
```
import gensim
model = gensim.models.KeyedVectors.load_word2vec_format('word2vec_model_b.txt',
```

In [3]:
```
from sklearn.utils import shuffle
values_to_stratify = [0,1]
def stratified_sample(group, n=100000):
    return group.sample(n=n, random_state=1)

binary_data = raw_data[raw_data['star_rating'].isin(values_to_stratify)].groupby
shuffled_dataset = shuffle(binary_data, random_state=42)
from sklearn.model_selection import train_test_split
X_train_g = shuffled_dataset[:160000]
X_test_g = shuffled_dataset[160000:]
```

In [4]:
```
# def average_word2vec(text, model=None, vw=None):
#     if model:
#         word_vectors = [model[word] for word in text if word in model.key_to_i
#     if vw:
#         word_vectors = [wv[word] for word in text if word in wv.key_to_index]
#     if word_vectors:
#         return np.mean(word_vectors, axis=0)
#     else:
#         return np.zeros(300)

def concatenate_word_vectors(sentence, max_length=10, vw=None,  model=None):
    concatenated_vectors = []
    if vw:
        for i in range(max_length):
            if i < len(sentence):
                if sentence[i] in vw.key_to_index:
                    tmp_vectors = vw[sentence[i]]
                else:
                    tmp_vectors = np.zeros(300,dtype=np.float16)
                concatenated_vectors.extend(tmp_vectors)
            else:
                concatenated_vectors.extend(np.zeros(300,dtype=np.float16))
    else:
        for i in range(max_length):
            if i < len(sentence):
                if sentence[i] in model.key_to_index:
                    tmp_vectors = model[sentence[i]]
                else:
                    tmp_vectors = np.zeros(300,dtype=np.float16)
                concatenated_vectors.extend(tmp_vectors)
            else:
                concatenated_vectors.extend(np.zeros(300,dtype=np.float16))
    return concatenated_vectors
```

In [5]:
```
model.vectors = model.vectors.astype(np.float16)
```

In [6]:
```
from torch.utils.data import Dataset
class SentimentDataset(Dataset):
    def __init__(self, X, y):
        self.X = X
        self.y = y

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

    def __getitem__(self, index):
        features = torch.tensor(self.X[index], dtype=torch.float16)
        label = torch.tensor(self.y[index], dtype=torch.long)
        return features, label
```

Without BetterDataset (need at least 16gb memory)

In [7]:
```
X_train_tensor_b_g = X_train_g['clean_reviw'].apply(lambda x: concatenate_word_
y_train_binary_list = X_train_g['star_rating'].tolist()
train_dataset = SentimentDataset(X_train_tensor_b_g, y_train_binary_list)
```

In [8]:
```
X_test_tensor_b_g = X_test_g['clean_reviw'].apply(lambda x: concatenate_word_ve
y_test_binary_list = X_test_g['star_rating'].tolist()
test_dataset = SentimentDataset(X_test_tensor_b_g, y_test_binary_list)
```

In [9]:
```
# Define the MLP model
class SentimentClassifier(nn.Module):
    def __init__(self, input_size, hidden_size1, hidden_size2, output_size):
        super(SentimentClassifier, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size1).float()
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size1, hidden_size2).float()
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(hidden_size2, output_size).float()

    def forward(self, x):
        x = self.relu1(self.fc1(x))
        x = self.relu2(self.fc2(x))
        x = self.fc3(x)
        return x


input_size = 3000
hidden_size1 = 50
hidden_size2 = 10
output_size_binary = 2

batch_size = 64

model_binary = SentimentClassifier(input_size, hidden_size1, hidden_size2, outpu

criterion = nn.CrossEntropyLoss()
optimizer_binary = optim.Adam(model_binary.parameters(), lr=0.001)
```

In [10]:
```
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

In [11]:
```
print(model_binary.fc1.weight.dtype)
```

```
torch.float32
```

In [12]:
```
# Training loop
num_epochs = 5
for epoch in range(num_epochs):
    model_binary.train()
    for inputs, labels in train_loader:
        inputs = inputs.view(inputs.size(0), -1).float()
        outputs_binary = model_binary(inputs)
        loss_binary = criterion(outputs_binary, labels)

        # Backward and optimize
        optimizer_binary.zero_grad()
        loss_binary.backward()
        optimizer_binary.step()
```

In [13]:
```
from sklearn.metrics import accuracy_score
model_binary.eval()
predictions_binary = []
true_labels_binary = []
with torch.no_grad():
    for inputs, labels in test_loader:
        inputs = inputs.view(inputs.size(0), -1).float()
        outputs_binary = model_binary(inputs)
        _, predicted_binary = torch.max(outputs_binary, 1)
        predictions_binary.extend(predicted_binary.numpy())
        true_labels_binary.extend(labels.numpy())
accuracy_binary = accuracy_score(true_labels_binary, predictions_binary)
print(accuracy_binary)
```

```
0.78645
```

In [ ]:
```

```

In [1]:

```
import pandas as pd
import numpy as np
import nltk
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.model_selection import train_test_split
import numpy as np
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
nltk.download('wordnet')
raw_data = pd.read_csv('./dataset.csv', sep=',')
for index in range(len(raw_data)):
    if raw_data.at[index, 'star_rating'] > 3:
        raw_data.at[index, 'star_rating'] = 1
    elif raw_data.at[index, 'star_rating'] == 3:
        raw_data.at[index, 'star_rating'] = 2
    else:
        raw_data.at[index, 'star_rating'] = 0
from bs4 import BeautifulSoup
import re
# data cleaning
raw_data['review_body'] = raw_data['review_body'].str.lower()
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: re.sub(r'http
def remove_non_alphabetical(text):
    return re.sub(r'[^a-zA-Z\s]', '', text)

raw_data['review_body'] = raw_data['review_body'].apply(remove_non_alphabetical)
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: ' '.join(x.sp
contraction_dict = contraction_dict = {
    "don't": "do not",
    "doesn't": "does not",
    "didn't": "did not",
    "can't": "cannot",
    "won't": "will not",
    "isn't": "is not",
    "haven't": "have not",
    "hasn't": "has not",
    "hadn't": "had not",
    "you're": "you are",
    "you've": "you have",
    "you'll": "you will",
    "when's": "when is",
    "let's": "let us",
    "'cause": "because",
    "shouldn't": "should not",
    "wouldn't": "would not",
    "couldn't": "could not",
    "wasn't": "was not",
    "weren't": "were not",
    "I'm": "I am",
    "I've": "I have",
    "I'll": "I will",
    "it's": "it is",
    "that's": "that is",
    "who's": "who is",
    "what's": "what is",
    "where's": "where is",
    "we're": "we are",
```

```
    "we've": "we have",
    "we'll": "we will",
    "they're": "they are",
    "they've": "they have",
    "they'll": "they will",
    "she's": "she is",
    "he's": "he is",
    "how's": "how is",
    "you'd": "you would",
    "we'd": "we would",
    "they'd": "they would",
}
raw_data['review_body'] = raw_data['review_body'].replace(contraction_dict, rege
from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
def remove_stop_words(text):
    words = text.split(" ")
    filtered_words = [word for word in words if word.lower() not in stop_words
    return ' '.join(filtered_words)
raw_data['review_body'] = raw_data['review_body'].apply(remove_stop_words)
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
nltk.download('omw-1.4') # I have to download this dataset
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: ' '.join([lem
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
import nltk
import string
def preprocess_text(text):
    tokens = word_tokenize(text)
    stop_words = set(stopwords.words('english') + list(string.punctuation))
    tokens = [word.lower() for word in tokens if word.isalpha() and word.lower
    return tokens
raw_data['clean_reviw'] = raw_data['review_body'].apply(preprocess_text)
```

```
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\tqdm\auto.py:22: TqdmWarni
ng: IProgress not found. Please update jupyter and ipywidgets. See https://ipy
widgets.readthedocs.io/en/stable/user_install.html (https://ipywidgets.readthe
docs.io/en/stable/user_install.html)
  from .autonotebook import tqdm as notebook_tqdm
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\ipykernel_launcher.py:26:
MarkupResemblesLocatorWarning: The input looks more like a filename than marku
p. You may want to open this file and pass the filehandle into Beautiful Soup.
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\ipykernel_launcher.py:26:
MarkupResemblesLocatorWarning: The input looks more like a URL than markup. Yo
u may want to use an HTTP client like requests to get the document behind the
URL, and feed that document to Beautiful Soup.
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

In [2]:
```
import gensim
model = gensim.models.KeyedVectors.load_word2vec_format('word2vec_model_b.txt',
```

In [3]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

With BetterDataset (need at least 10gb memory)

In [4]:
```
from torch.utils.data import Dataset
class BetterDataset(Dataset):
    def __init__(self, reviews, labels, word2vec_model):
        self.reviews = reviews
        self.labels = labels
        self.word2vec_model = word2vec_model

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

    def __getitem__(self, idx):
        review = self.reviews[idx]
        # word_vectors = []

        concatenated_vectors = np.zeros((10, 300), dtype=np.float16)
        for i in range(min(10, len(review))):
            word = review[i]
            if word in self.word2vec_model.key_to_index:
                concatenated_vectors[i] = self.word2vec_model[word]

#         for i in range(10):
#             word = review[i]
#             if word in self.word2vec_model:
#                 word_vector = self.word2vec_model[word]
#             else:
#                 word_vector = torch.zeros(300, dtype=np.float16)  # Handle out
#             word_vectors.append(word_vector)

        label = torch.tensor(self.labels[idx], dtype=torch.long)
        return torch.tensor(concatenated_vectors.flatten().tolist()), label
```

In [5]:
```
from sklearn.utils import shuffle
shuffled_dataset = shuffle(raw_data, random_state=42)
from sklearn.model_selection import train_test_split
X_train_g = shuffled_dataset[:200000]
X_test_g = shuffled_dataset[200000:]
print(len(X_train_g))
print(len(X_test_g))
```

```
200000
50000
```

In [6]:
```
X_train_g = X_train_g.reset_index(drop=True)
```

In [7]:
```
X_test_g = X_test_g.reset_index(drop=True)
```

In [8]:
```
train_dataset = BetterDataset(X_train_g['clean_reviw'], X_train_g['star_rating']
```

In [9]:
```
test_dataset = BetterDataset(X_test_g['clean_reviw'], X_test_g['star_rating'], r
```

In [10]:
```
class SentimentClassifier(nn.Module):
    def __init__(self, input_size, hidden_size1, hidden_size2, output_size):
        super(SentimentClassifier, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size1).float()
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size1, hidden_size2).float()
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(hidden_size2, output_size).float()

    def forward(self, x):
        x = self.relu1(self.fc1(x))
        x = self.relu2(self.fc2(x))
        x = self.fc3(x)
        return x


input_size = 3000
hidden_size1 = 50
hidden_size2 = 10
output_size_ternary = 3

model_binary = SentimentClassifier(input_size, hidden_size1, hidden_size2, outpu

criterion = nn.CrossEntropyLoss()
optimizer_binary = optim.Adam(model_binary.parameters(), lr=0.001)
```

In [11]:
```
batch_size = 16
```

In [12]:
```
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
```

In [13]:
```
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

In [14]:
```
num_epochs = 5
for epoch in range(num_epochs):
    model_binary.train()
    for inputs, labels in train_loader:
        inputs = inputs.view(inputs.size(0), -1).float()
        outputs_binary = model_binary(inputs.to(device))
        loss_binary = criterion(outputs_binary, labels.to(device))

        # Backward and optimize
        optimizer_binary.zero_grad()
        loss_binary.backward()
        optimizer_binary.step()
```

In [15]:
```
from sklearn.metrics import accuracy_score
model_binary.to('cpu')
model_binary.eval()
predictions_binary = []
true_labels_binary = []
with torch.no_grad():
    for inputs, labels in test_loader:
        inputs = inputs.view(inputs.size(0), -1).float()
        outputs_binary = model_binary(inputs)
        _, predicted_binary = torch.max(outputs_binary, 1)
        predictions_binary.extend(predicted_binary.numpy())
        true_labels_binary.extend(labels.numpy())
accuracy_ternary = accuracy_score(true_labels_binary, predictions_binary)
print(accuracy_ternary)
```

```
0.63976
```

In [ ]:
```

```

In [1]:

```
import pandas as pd
import numpy as np
import nltk
nltk.download('wordnet')
raw_data = pd.read_csv('./dataset.csv', sep=',')
for index in range(len(raw_data)):
    if raw_data.at[index, 'star_rating'] > 3:
        raw_data.at[index, 'star_rating'] = 1
    elif raw_data.at[index, 'star_rating'] == 3:
        raw_data.at[index, 'star_rating'] = 2
    else:
        raw_data.at[index, 'star_rating'] = 0
from bs4 import BeautifulSoup
import re
# data cleaning
raw_data['review_body'] = raw_data['review_body'].str.lower()
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: re.sub(r'http
def remove_non_alphabetical(text):
    return re.sub(r'[^a-zA-Z\s]', '', text)

raw_data['review_body'] = raw_data['review_body'].apply(remove_non_alphabetical)
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: ' '.join(x.sp
contraction_dict = contraction_dict = {
    "don't": "do not",
    "doesn't": "does not",
    "didn't": "did not",
    "can't": "cannot",
    "won't": "will not",
    "isn't": "is not",
    "haven't": "have not",
    "hasn't": "has not",
    "hadn't": "had not",
    "you're": "you are",
    "you've": "you have",
    "you'll": "you will",
    "when's": "when is",
    "let's": "let us",
    "'cause": "because",
    "shouldn't": "should not",
    "wouldn't": "would not",
    "couldn't": "could not",
    "wasn't": "was not",
    "weren't": "were not",
    "I'm": "I am",
    "I've": "I have",
    "I'll": "I will",
    "it's": "it is",
    "that's": "that is",
    "who's": "who is",
    "what's": "what is",
    "where's": "where is",
    "we're": "we are",
    "we've": "we have",
    "we'll": "we will",
    "they're": "they are",
    "they've": "they have",
    "they'll": "they will",
    "she's": "she is",
    "he's": "he is",
    "how's": "how is",
    "you'd": "you would",
```

```
    "we'd": "we would",
    "they'd": "they would",
}
raw_data['review_body'] = raw_data['review_body'].replace(contraction_dict, rege
from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
def remove_stop_words(text):
    words = text.split(" ")
    filtered_words = [word for word in words if word.lower() not in stop_words
    return ' '.join(filtered_words)
raw_data['review_body'] = raw_data['review_body'].apply(remove_stop_words)
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
nltk.download('omw-1.4') # I have to download this dataset
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: ' '.join([lem
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
import nltk
import string
def preprocess_text(text):
    tokens = word_tokenize(text)
    stop_words = set(stopwords.words('english') + list(string.punctuation))
    tokens = [word.lower() for word in tokens if word.isalpha() and word.lower
    return tokens
raw_data['clean_reviw'] = raw_data['review_body'].apply(preprocess_text)
# better word2vec
# model = Word2Vec(sentences=raw_data['clean_reviw'], vector_size=300, window=5,
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\ipykernel_launcher.py:17:
MarkupResemblesLocatorWarning: The input looks more like a filename than marku
p. You may want to open this file and pass the filehandle into Beautiful Soup.
  app.launch_new_instance()
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\ipykernel_launcher.py:17:
MarkupResemblesLocatorWarning: The input looks more like a URL than markup. Yo
u may want to use an HTTP client like requests to get the document behind the
URL, and feed that document to Beautiful Soup.
  app.launch_new_instance()
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

HW2-CNN-google-binary

In [2]:

```
import gensim.downloader as api
wv = api.load('word2vec-google-news-300')
# import gensim
# model = gensim.models.KeyedVectors.load_word2vec_format('word2vec_model_b.txt'
```

In [3]:
```
values_to_stratify = [0,1]
def stratified_sample(group, n=100000):
    return group.sample(n=n, random_state=1)

binary_data= raw_data[raw_data['star_rating'].isin(values_to_stratify)].groupby
```

In [4]:
```
from sklearn.utils import shuffle
shuffled_dataset = shuffle(binary_data, random_state=42)
```

In [5]:
```
max_len = 50
def tokenize_and_pad(text, max_len):
    text = text[:max_len]  # Truncate longer reviews
    padding = max(0, max_len - len(text))
    return text + ['<PAD>'] * padding

shuffled_dataset['padded_text_column'] = shuffled_dataset['clean_reviw'].apply(
```

In [6]:
```
X_self_wv = np.zeros((len(shuffled_dataset), max_len, 300), dtype=np.float32)

for i, review in enumerate(shuffled_dataset['padded_text_column']):
    for j, word in enumerate(review):
        if word in wv:
            X_self_wv[i, j, :] = wv[word]
        else:
            X_self_wv[i, j, :] = np.zeros(300, dtype=np.float32)

# X_self_wv = np.array([np.array([model[word] if word in model else np.zeros(300
```

In [7]:
```
print(X_self_wv.shape)
```

```
(200000, 50, 300)
```

In [8]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
import numpy as np
```

```
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\tqdm\auto.py:22: TqdmWarni
ng: IProgress not found. Please update jupyter and ipywidgets. See https://ipy
widgets.readthedocs.io/en/stable/user_install.html (https://ipywidgets.readthe
docs.io/en/stable/user_install.html)
  from .autonotebook import tqdm as notebook_tqdm
```

In [9]:
```
X_padded_tensor = torch.FloatTensor(X_self_wv)
X = X_padded_tensor
y = shuffled_dataset['star_rating'].astype(int)
```

In [10]:
```
print(X.shape)
```

```
torch.Size([200000, 50, 300])
```

In [11]:
```
print(len(X))
```

```
200000
```

In [12]:
```
X_train_s = X[0:160000]
X_test_s = X[160000:]
y_train_s = y[0:160000]
y_test_s = y[160000:]
```

In [13]:
```
class MyCNN(nn.Module):
    def __init__(self, input_size, output_channels):
        super(MyCNN, self).__init__()
        self.conv1 = nn.Conv1d(in_channels=300, out_channels=50, kernel_size=3)
        self.conv2 = nn.Conv1d(in_channels=50, out_channels=10, kernel_size=3)
        self.global_pooling = nn.AdaptiveMaxPool1d(1)
        self.fc = nn.Linear(10, 2)

    def forward(self, x):
        x = nn.functional.relu(self.conv1(x))
        x = nn.functional.relu(self.conv2(x))
        x = self.global_pooling(x).squeeze(2)
        x = self.fc(x)
        return x
input_size = 300  # Size of Word2Vec vectors
output_channels = [50, 10]  # Output channel sizes

cnn_model = MyCNN(input_size, output_channels)

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

X_train_tensor_s_cnn = X_train_s.permute(0,2,1)  # Permute dimensions for CNN ir
y_train_tensor_s_cnn = torch.LongTensor(y_train_s.values)  # Assuming your targe
X_test_tensor_s_cnn = X_test_s.permute(0,2,1)  # Permute dimensions for CNN inpu
y_test_tensor_s_cnn = torch.LongTensor(y_test_s.values)
```

In [14]:
```
epochs = 5
batch_size = 32

for epoch in range(epochs):
    for i in range(0, len(X_train_tensor_s_cnn), batch_size):
        batch_X = X_train_tensor_s_cnn[i:i + batch_size]
        batch_y = y_train_tensor_s_cnn[i:i + batch_size]

        optimizer.zero_grad()
        outputs = cnn_model(batch_X)
        loss = criterion(outputs.squeeze(), batch_y)
        loss.backward()
        optimizer.step()
```

In [15]:
```
from sklearn.metrics import accuracy_score
with torch.no_grad():
    cnn_model.eval()
    outputs = cnn_model(X_test_tensor_s_cnn)
    predictions = torch.argmax(outputs, dim=1)
    accuracy = accuracy_score(y_test_tensor_s_cnn.numpy(), predictions.numpy())
print(accuracy)
```

```
0.853225
```

In [ ]:

In [ ]:

In [1]:

```
import pandas as pd
import numpy as np
import nltk
nltk.download('wordnet')
raw_data = pd.read_csv('./dataset.csv', sep=',')
for index in range(len(raw_data)):
    if raw_data.at[index, 'star_rating'] > 3:
        raw_data.at[index, 'star_rating'] = 1
    elif raw_data.at[index, 'star_rating'] == 3:
        raw_data.at[index, 'star_rating'] = 2
    else:
        raw_data.at[index, 'star_rating'] = 0
from bs4 import BeautifulSoup
import re
# data cleaning
raw_data['review_body'] = raw_data['review_body'].str.lower()
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: re.sub(r'http
def remove_non_alphabetical(text):
    return re.sub(r'[^a-zA-Z\s]', '', text)

raw_data['review_body'] = raw_data['review_body'].apply(remove_non_alphabetical)
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: ' '.join(x.sp
contraction_dict = contraction_dict = {
    "don't": "do not",
    "doesn't": "does not",
    "didn't": "did not",
    "can't": "cannot",
    "won't": "will not",
    "isn't": "is not",
    "haven't": "have not",
    "hasn't": "has not",
    "hadn't": "had not",
    "you're": "you are",
    "you've": "you have",
    "you'll": "you will",
    "when's": "when is",
    "let's": "let us",
    "'cause": "because",
    "shouldn't": "should not",
    "wouldn't": "would not",
    "couldn't": "could not",
    "wasn't": "was not",
    "weren't": "were not",
    "I'm": "I am",
    "I've": "I have",
    "I'll": "I will",
    "it's": "it is",
    "that's": "that is",
    "who's": "who is",
    "what's": "what is",
    "where's": "where is",
    "we're": "we are",
    "we've": "we have",
    "we'll": "we will",
    "they're": "they are",
    "they've": "they have",
    "they'll": "they will",
    "she's": "she is",
    "he's": "he is",
    "how's": "how is",
    "you'd": "you would",
```

```
    "we'd": "we would",
    "they'd": "they would",
}
raw_data['review_body'] = raw_data['review_body'].replace(contraction_dict, rege
from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
def remove_stop_words(text):
    words = text.split(" ")
    filtered_words = [word for word in words if word.lower() not in stop_words
    return ' '.join(filtered_words)
raw_data['review_body'] = raw_data['review_body'].apply(remove_stop_words)
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
nltk.download('omw-1.4') # I have to download this dataset
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: ' '.join([lem
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
import nltk
import string
def preprocess_text(text):
    tokens = word_tokenize(text)
    stop_words = set(stopwords.words('english') + list(string.punctuation))
    tokens = [word.lower() for word in tokens if word.isalpha() and word.lower
    return tokens
raw_data['clean_reviw'] = raw_data['review_body'].apply(preprocess_text)
# better word2vec
# model = Word2Vec(sentences=raw_data['clean_reviw'], vector_size=300, window=5,
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\ipykernel_launcher.py:17:
MarkupResemblesLocatorWarning: The input looks more like a filename than marku
p. You may want to open this file and pass the filehandle into Beautiful Soup.
  app.launch_new_instance()
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\ipykernel_launcher.py:17:
MarkupResemblesLocatorWarning: The input looks more like a URL than markup. Yo
u may want to use an HTTP client like requests to get the document behind the
URL, and feed that document to Beautiful Soup.
  app.launch_new_instance()
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

HW2-CNN-google-ternary

In [2]:

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

In [3]:
```
from sklearn.utils import shuffle
shuffled_dataset = shuffle(raw_data, random_state=42)
```

In [4]:
```
max_len = 50
def tokenize_and_pad(text, max_len):
    text = text[:max_len]  # Truncate longer reviews
    padding = max(0, max_len - len(text))
    return text + ['<PAD>'] * padding

shuffled_dataset['padded_text_column'] = shuffled_dataset['clean_reviw'].apply(
```

In [5]:
```
X_self_wv = np.zeros((len(shuffled_dataset), max_len, 300), dtype=np.float32)

for i, review in enumerate(shuffled_dataset['padded_text_column']):
    for j, word in enumerate(review):
        if word in wv:
            X_self_wv[i, j, :] = wv[word]
        else:
            X_self_wv[i, j, :] = np.zeros(300, dtype=np.float32)

# X_self_wv = np.array([np.array([model[word] if word in model else np.zeros(300
```

In [6]:
```
print(X_self_wv.shape)
```

```
(250000, 50, 300)
```

In [7]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
import numpy as np
```

```
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\tqdm\auto.py:22: TqdmWarni
ng: IProgress not found. Please update jupyter and ipywidgets. See https://ipy
widgets.readthedocs.io/en/stable/user_install.html (https://ipywidgets.readthe
docs.io/en/stable/user_install.html)
  from .autonotebook import tqdm as notebook_tqdm
```

In [8]:
```
X_padded_tensor = torch.FloatTensor(X_self_wv)
X = X_padded_tensor
y = shuffled_dataset['star_rating'].astype(int)
```

In [9]:
```
print(X.shape)
```

```
torch.Size([250000, 50, 300])
```

In [10]:
```
print(len(X))
```

250000

In [11]:
```
X_train_s = X[0:200000]
X_test_s = X[200000:]
y_train_s = y[0:200000]
y_test_s = y[200000:]
```

In [12]:
```
class MyCNN(nn.Module):
    def __init__(self, input_size, output_channels):
        super(MyCNN, self).__init__()
        self.conv1 = nn.Conv1d(in_channels=300, out_channels=50, kernel_size=3)
        self.conv2 = nn.Conv1d(in_channels=50, out_channels=10, kernel_size=3)
        self.global_pooling = nn.AdaptiveMaxPool1d(1)
        self.fc = nn.Linear(10, 3)

    def forward(self, x):
        x = nn.functional.relu(self.conv1(x))
        x = nn.functional.relu(self.conv2(x))
        x = self.global_pooling(x).squeeze(2)
        x = self.fc(x)
        return x
input_size = 300  # Size of Word2Vec vectors
output_channels = [50, 10]  # Output channel sizes

cnn_model = MyCNN(input_size, output_channels)

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

X_train_tensor_s_cnn = X_train_s.permute(0,2,1)  # Permute dimensions for CNN ir
y_train_tensor_s_cnn = torch.LongTensor(y_train_s.values)  # Assuming your targe
X_test_tensor_s_cnn = X_test_s.permute(0,2,1)  # Permute dimensions for CNN inpu
y_test_tensor_s_cnn = torch.LongTensor(y_test_s.values)
```

In [13]:
```
epochs = 5
batch_size = 32

for epoch in range(epochs):
    for i in range(0, len(X_train_tensor_s_cnn), batch_size):
        batch_X = X_train_tensor_s_cnn[i:i + batch_size]
        batch_y = y_train_tensor_s_cnn[i:i + batch_size]

        optimizer.zero_grad()
        outputs = cnn_model(batch_X)
        loss = criterion(outputs.squeeze(), batch_y)
        loss.backward()
        optimizer.step()
```

In [14]:
```
correct = 0
total = 0
with torch.no_grad():ssssss
    for i in range(0, len(X_test_tensor_s_cnn), batch_size):
        inputs = X_test_tensor_s_cnn[i:i+batch_size]
        labels = y_test_tensor_s_cnn[i:i+batch_size]

        outputs =cnn_model(inputs)
        _, predicted = torch.max(outputs.data, 1)

        total += labels.size(0)
        correct += (predicted == labels).sum().item()

accuracy = correct / total
print(f'Accuracy: {accuracy}')
```

```
Accuracy: 0.69298
```

In [ ]:

In [ ]:

In [1]:

```
import pandas as pd
import numpy as np
import nltk
nltk.download('wordnet')
raw_data = pd.read_csv('./dataset.csv', sep=',')
for index in range(len(raw_data)):
    if raw_data.at[index, 'star_rating'] > 3:
        raw_data.at[index, 'star_rating'] = 1
    elif raw_data.at[index, 'star_rating'] == 3:
        raw_data.at[index, 'star_rating'] = 2
    else:
        raw_data.at[index, 'star_rating'] = 0
from bs4 import BeautifulSoup
import re
# data cleaning
raw_data['review_body'] = raw_data['review_body'].str.lower()
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: re.sub(r'http
def remove_non_alphabetical(text):
    return re.sub(r'[^a-zA-Z\s]', '', text)

raw_data['review_body'] = raw_data['review_body'].apply(remove_non_alphabetical)
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: ' '.join(x.sp
contraction_dict = contraction_dict = {
    "don't": "do not",
    "doesn't": "does not",
    "didn't": "did not",
    "can't": "cannot",
    "won't": "will not",
    "isn't": "is not",
    "haven't": "have not",
    "hasn't": "has not",
    "hadn't": "had not",
    "you're": "you are",
    "you've": "you have",
    "you'll": "you will",
    "when's": "when is",
    "let's": "let us",
    "'cause": "because",
    "shouldn't": "should not",
    "wouldn't": "would not",
    "couldn't": "could not",
    "wasn't": "was not",
    "weren't": "were not",
    "I'm": "I am",
    "I've": "I have",
    "I'll": "I will",
    "it's": "it is",
    "that's": "that is",
    "who's": "who is",
    "what's": "what is",
    "where's": "where is",
    "we're": "we are",
    "we've": "we have",
    "we'll": "we will",
    "they're": "they are",
    "they've": "they have",
    "they'll": "they will",
    "she's": "she is",
    "he's": "he is",
    "how's": "how is",
    "you'd": "you would",
```

```
    "we'd": "we would",
    "they'd": "they would",
}
raw_data['review_body'] = raw_data['review_body'].replace(contraction_dict, rege
from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
def remove_stop_words(text):
    words = text.split(" ")
    filtered_words = [word for word in words if word.lower() not in stop_words
    return ' '.join(filtered_words)
raw_data['review_body'] = raw_data['review_body'].apply(remove_stop_words)
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
nltk.download('omw-1.4') # I have to download this dataset
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: ' '.join([lem
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
import nltk
import string
def preprocess_text(text):
    tokens = word_tokenize(text)
    stop_words = set(stopwords.words('english') + list(string.punctuation))
    tokens = [word.lower() for word in tokens if word.isalpha() and word.lower
    return tokens
raw_data['clean_reviw'] = raw_data['review_body'].apply(preprocess_text)
# better word2vec
# model = Word2Vec(sentences=raw_data['clean_reviw'], vector_size=300, window=5,
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\ipykernel_launcher.py:17:
MarkupResemblesLocatorWarning: The input looks more like a filename than marku
p. You may want to open this file and pass the filehandle into Beautiful Soup.
  app.launch_new_instance()
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\ipykernel_launcher.py:17:
MarkupResemblesLocatorWarning: The input looks more like a URL than markup. Yo
u may want to use an HTTP client like requests to get the document behind the
URL, and feed that document to Beautiful Soup.
  app.launch_new_instance()
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

HW2-CNN-self-binary

In [2]:

```
import gensim.downloader as api
wv = api.load('word2vec-google-news-300')
import gensim
model = gensim.models.KeyedVectors.load_word2vec_format('word2vec_model_b.txt',
```

In [3]:
```
values_to_stratify = [0,1]
def stratified_sample(group, n=100000):
    return group.sample(n=n, random_state=1)

binary_data= raw_data[raw_data['star_rating'].isin(values_to_stratify)].groupby
```

In [4]:
```
from sklearn.utils import shuffle
shuffled_dataset = shuffle(binary_data, random_state=42)
```

In [5]:
```
max_len = 50
def tokenize_and_pad(text, max_len):
    text = text[:max_len]  # Truncate longer reviews
    padding = max(0, max_len - len(text))
    return text + ['<PAD>'] * padding

shuffled_dataset['padded_text_column'] = shuffled_dataset['clean_reviw'].apply(
```

In [6]:
```
X_self_wv = np.zeros((len(shuffled_dataset), max_len, 300), dtype=np.float32)

for i, review in enumerate(shuffled_dataset['padded_text_column']):
    for j, word in enumerate(review):
        if word in model:
            X_self_wv[i, j, :] = model[word]
        else:
            X_self_wv[i, j, :] = np.zeros(300, dtype=np.float32)

# X_self_wv = np.array([np.array([model[word] if word in model else np.zeros(300
```

In [7]:
```
print(X_self_wv.shape)
```

```
(200000, 50, 300)
```

In [8]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
import numpy as np
```

```
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\tqdm\auto.py:22: TqdmWarni
ng: IProgress not found. Please update jupyter and ipywidgets. See https://ipy
widgets.readthedocs.io/en/stable/user_install.html (https://ipywidgets.readthe
docs.io/en/stable/user_install.html)
  from .autonotebook import tqdm as notebook_tqdm
```

In [9]:
```
X_padded_tensor = torch.FloatTensor(X_self_wv)
X = X_padded_tensor
y = shuffled_dataset['star_rating'].astype(int)
```

In [10]:
```
print(X.shape)
```

```
torch.Size([200000, 50, 300])
```

In [11]:
```
print(len(X))
```

```
200000
```

In [12]:
```
X_train_s = X[0:160000]
X_test_s = X[160000:]
y_train_s = y[0:160000]
y_test_s = y[160000:]
```

In [13]:
```
class MyCNN(nn.Module):
    def __init__(self, input_size, output_channels):
        super(MyCNN, self).__init__()
        self.conv1 = nn.Conv1d(in_channels=300, out_channels=50, kernel_size=3)
        self.conv2 = nn.Conv1d(in_channels=50, out_channels=10, kernel_size=3)
        self.global_pooling = nn.AdaptiveMaxPool1d(1)
        self.fc = nn.Linear(10, 2)

    def forward(self, x):
        x = nn.functional.relu(self.conv1(x))
        x = nn.functional.relu(self.conv2(x))
        x = self.global_pooling(x).squeeze(2)
        x = self.fc(x)
        return x
input_size = 300  # Size of Word2Vec vectors
output_channels = [50, 10]  # Output channel sizes

cnn_model = MyCNN(input_size, output_channels)

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

X_train_tensor_s_cnn = X_train_s.permute(0,2,1)  # Permute dimensions for CNN ir
y_train_tensor_s_cnn = torch.LongTensor(y_train_s.values)  # Assuming your targe
X_test_tensor_s_cnn = X_test_s.permute(0,2,1)  # Permute dimensions for CNN inpu
y_test_tensor_s_cnn = torch.LongTensor(y_test_s.values)
```

In [14]:
```
epochs = 5
batch_size = 32

for epoch in range(epochs):
    for i in range(0, len(X_train_tensor_s_cnn), batch_size):
        batch_X = X_train_tensor_s_cnn[i:i + batch_size]
        batch_y = y_train_tensor_s_cnn[i:i + batch_size]

        optimizer.zero_grad()
        outputs = cnn_model(batch_X)
        loss = criterion(outputs.squeeze(), batch_y)
        loss.backward()
        optimizer.step()
```

In [15]:
```
from sklearn.metrics import accuracy_score
with torch.no_grad():
    cnn_model.eval()
    outputs = cnn_model(X_test_tensor_s_cnn)
    predictions = torch.argmax(outputs, dim=1)
    accuracy = accuracy_score(y_test_tensor_s_cnn.numpy(), predictions.numpy())
print(accuracy)
```

```
0.858525
```

In [ ]:
```

```

In [1]:

```
import pandas as pd
import numpy as np
import nltk
nltk.download('wordnet')
raw_data = pd.read_csv('./dataset.csv', sep=',')
for index in range(len(raw_data)):
    if raw_data.at[index, 'star_rating'] > 3:
        raw_data.at[index, 'star_rating'] = 1
    elif raw_data.at[index, 'star_rating'] == 3:
        raw_data.at[index, 'star_rating'] = 2
    else:
        raw_data.at[index, 'star_rating'] = 0
from bs4 import BeautifulSoup
import re
# data cleaning
raw_data['review_body'] = raw_data['review_body'].str.lower()
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: re.sub(r'http
def remove_non_alphabetical(text):
    return re.sub(r'[^a-zA-Z\s]', '', text)

raw_data['review_body'] = raw_data['review_body'].apply(remove_non_alphabetical)
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: ' '.join(x.sp
contraction_dict = contraction_dict = {
    "don't": "do not",
    "doesn't": "does not",
    "didn't": "did not",
    "can't": "cannot",
    "won't": "will not",
    "isn't": "is not",
    "haven't": "have not",
    "hasn't": "has not",
    "hadn't": "had not",
    "you're": "you are",
    "you've": "you have",
    "you'll": "you will",
    "when's": "when is",
    "let's": "let us",
    "'cause": "because",
    "shouldn't": "should not",
    "wouldn't": "would not",
    "couldn't": "could not",
    "wasn't": "was not",
    "weren't": "were not",
    "I'm": "I am",
    "I've": "I have",
    "I'll": "I will",
    "it's": "it is",
    "that's": "that is",
    "who's": "who is",
    "what's": "what is",
    "where's": "where is",
    "we're": "we are",
    "we've": "we have",
    "we'll": "we will",
    "they're": "they are",
    "they've": "they have",
    "they'll": "they will",
    "she's": "she is",
    "he's": "he is",
    "how's": "how is",
    "you'd": "you would",
```

```
    "we'd": "we would",
    "they'd": "they would",
}
raw_data['review_body'] = raw_data['review_body'].replace(contraction_dict, rege
from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
def remove_stop_words(text):
    words = text.split(" ")
    filtered_words = [word for word in words if word.lower() not in stop_words
    return ' '.join(filtered_words)
raw_data['review_body'] = raw_data['review_body'].apply(remove_stop_words)
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
nltk.download('omw-1.4') # I have to download this dataset
raw_data['review_body'] = raw_data['review_body'].apply(lambda x: ' '.join([lem
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
import nltk
import string
def preprocess_text(text):
    tokens = word_tokenize(text)
    stop_words = set(stopwords.words('english') + list(string.punctuation))
    tokens = [word.lower() for word in tokens if word.isalpha() and word.lower
    return tokens
raw_data['clean_reviw'] = raw_data['review_body'].apply(preprocess_text)
# better word2vec
# model = Word2Vec(sentences=raw_data['clean_reviw'], vector_size=300, window=5,
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\ipykernel_launcher.py:17:
MarkupResemblesLocatorWarning: The input looks more like a filename than marku
p. You may want to open this file and pass the filehandle into Beautiful Soup.
  app.launch_new_instance()
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\ipykernel_launcher.py:17:
MarkupResemblesLocatorWarning: The input looks more like a URL than markup. Yo
u may want to use an HTTP client like requests to get the document behind the
URL, and feed that document to Beautiful Soup.
  app.launch_new_instance()
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\monkeydc\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

HW2-CNN-self-ternary

In [2]:

```
import gensim
model = gensim.models.KeyedVectors.load_word2vec_format('word2vec_model_b.txt',
```

In [3]:
```
from sklearn.utils import shuffle
shuffled_dataset = shuffle(raw_data, random_state=42)
```

In [4]:
```
max_len = 50
def tokenize_and_pad(text, max_len):
    text = text[:max_len]  # Truncate longer reviews
    padding = max(0, max_len - len(text))
    return text + ['<PAD>'] * padding

shuffled_dataset['padded_text_column'] = shuffled_dataset['clean_reviw'].apply(
```

In [5]:
```
X_self_wv = np.zeros((len(shuffled_dataset), max_len, 300), dtype=np.float32)

for i, review in enumerate(shuffled_dataset['padded_text_column']):
    for j, word in enumerate(review):
        if word in model:
            X_self_wv[i, j, :] = model[word]
        else:
            X_self_wv[i, j, :] = np.zeros(300, dtype=np.float32)

# X_self_wv = np.array([np.array([model[word] if word in model else np.zeros(300
```

In [6]:
```
print(X_self_wv.shape)
```

```
(250000, 50, 300)
```

In [7]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
import numpy as np
```

```
C:\Users\monkeydc\.conda\envs\561\lib\site-packages\tqdm\auto.py:22: TqdmWarni
ng: IProgress not found. Please update jupyter and ipywidgets. See https://ipy
widgets.readthedocs.io/en/stable/user_install.html (https://ipywidgets.readthe
docs.io/en/stable/user_install.html)
  from .autonotebook import tqdm as notebook_tqdm
```

In [8]:
```
X_padded_tensor = torch.FloatTensor(X_self_wv)
X = X_padded_tensor
y = shuffled_dataset['star_rating'].astype(int)
```

In [9]:
```
print(X.shape)
```

```
torch.Size([250000, 50, 300])
```

In [10]:
```
print(len(X))
```

250000

In [11]:
```
X_train_s = X[0:200000]
X_test_s = X[200000:]
y_train_s = y[0:200000]
y_test_s = y[200000:]
```

In [12]:
```
class MyCNN(nn.Module):
    def __init__(self, input_size, output_channels):
        super(MyCNN, self).__init__()
        self.conv1 = nn.Conv1d(in_channels=300, out_channels=50, kernel_size=3)
        self.conv2 = nn.Conv1d(in_channels=50, out_channels=10, kernel_size=3)
        self.global_pooling = nn.AdaptiveMaxPool1d(1)
        self.fc = nn.Linear(10, 3)

    def forward(self, x):
        x = nn.functional.relu(self.conv1(x))
        x = nn.functional.relu(self.conv2(x))
        x = self.global_pooling(x).squeeze(2)
        x = self.fc(x)
        return x
input_size = 300  # Size of Word2Vec vectors
output_channels = [50, 10]  # Output channel sizes

cnn_model = MyCNN(input_size, output_channels)

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

X_train_tensor_s_cnn = X_train_s.permute(0,2,1)  # Permute dimensions for CNN ir
y_train_tensor_s_cnn = torch.LongTensor(y_train_s.values)  # Assuming your targe
X_test_tensor_s_cnn = X_test_s.permute(0,2,1)  # Permute dimensions for CNN inpu
y_test_tensor_s_cnn = torch.LongTensor(y_test_s.values)
```

In [13]:
```
epochs = 5
batch_size = 32

for epoch in range(epochs):
    for i in range(0, len(X_train_tensor_s_cnn), batch_size):
        batch_X = X_train_tensor_s_cnn[i:i + batch_size]
        batch_y = y_train_tensor_s_cnn[i:i + batch_size]

        optimizer.zero_grad()
        outputs = cnn_model(batch_X)
        loss = criterion(outputs.squeeze(), batch_y)
        loss.backward()
        optimizer.step()
```

In [15]:

```
correct = 0
total = 0
with torch.no_grad():
    for i in range(0, len(X_test_tensor_s_cnn), batch_size):
        inputs = X_test_tensor_s_cnn[i:i+batch_size]
        labels = y_test_tensor_s_cnn[i:i+batch_size]

        outputs =cnn_model(inputs)
        _, predicted = torch.max(outputs.data, 1)

        total += labels.size(0)
        correct += (predicted == labels).sum().item()

accuracy = correct / total
print(f'Accuracy: {accuracy}')
```

```
Accuracy: 0.68998
```

In [ ]: