CSCI 544 Assignemnt 2

Mounika Mukkamalla

February 8, 2024

1

Amazon data is read, and sampled with 50k rows from each rating, and labeled, and data is split in binary and ternary labeled data sets based on star rating.
Word embeddings from custom and google pretrained model are obtained.
Then using these 2 embeddings, we train simple models, MLP and CNN for binary and ternary datasets and analyse the results.

2

For google pretrained model:
Similarity between excellent and outstanding: 0.5567486
Most similiar word for king - man + woman: queen, Similarity Score: 0.7118191123008728
for custom trained model:
Similarity between excellent and outstanding: 0.69074696
Most similiar word for king - man + woman: mwd, Similarity Score: 0.45065823197364807

For custom trained model, similarity between excellent and outstanding is higher compared to pretrained, because custom trained has lot of reviews with words similar to these words, and since the data set is not so diverse, these 2 vectors are more similar in custom trained. whereas in pretrained, there is huge amounts of diverse data.
For king - man + woman, we need to get queen, since google data is huge, it has more data to get trained on ,and hence possibility of queen existence is high. But with custom trained model, queen word existence is minimal because having word queen in reviews is rare.
In conclusion, for more general data, google pretrained model gives better results and for more similar data to amazon review, our custom model gives better reviews.
So, depending on our usecase, if we have more diverse and generic data, we can consider google pretrained model since it provides better semantic relations compared to custom model.

3

Test accuracies for perceptron and SVM for Word embeddings and TFIDF vectoriser:

Perceptron Google pretrained test accuracy is 0.8124764356958142
Perceptron Custom trained test accuracy: 0.8224435465478790

SVM Google pretrained test accuracy is 0.8478659898345692
SVM Custom trained test accuracy: 0.8808765934789245

Percptron trained using TFIDF features, test accuracy: 0.887675
SVM trained using TFIDF features , test accuracy : 0.92125

Case a. Comparison between pre-trained and custom trained word embeddings:

For both perceptron and SVM, custom trained provided better results compared to google pre-trained(reason is clearly explained in 4.b below).
Case b. Perceptron vs SVM:
SVM performed better in all the 3 cases with reasonaly good amount of accuracy. We might want to prefer SVM over perceptron for our input data.
Case c. Word embeddings vs tfidf features:
Tfidf performed better compared to word embeddings. This can be explained in a few ways:
TFIDF has sparse vector representation - this helps sometimes not to overfit the model, and word embeddings gives us the dense vector, prone to loosing some information(might not capture all the minute features in the dataset).
TFidf has lower dimensionality and word embeddings have a higher dimensionality.
TFidf might capture relevant and more crucial information more effectively than word embeddings.

4

4.a
Feed forward binary and ternary accuracies using pretrained and custom trained word embeddings
Binary Google pretrained feedforward test accuracy: 0.8674748
Binary custom trained feedforward test accuracy: 0.8896587658
Ternary Google pretrained feedforward test accuracy: 0.725488876
Ternary custom trained feedforward test accuracy: 0.746587

4.b
Binary Google pretrained feedforward test accuracy: 0.8163787
Binary custom trained feedforward test accuracy: 0.83563458
Ternary Google pretrained feedforward test accuracy: 0.6785605
Ternary custom trained feedforward test accuracy: 0.6943643

Analysis:
Case a. Comparison between 4.1 and 4.2
For MLP:
When we append 1st 10 vectors instead of taking average of all the vectors, all the test accuracies for 4.1 are higher compared to 4.2. This is because 4.1 takes mean of all vectors and hence it takes whole information of each review in the row, where as in 4.2, we might loose a little information sicne we are truncating the rest of the sentence.

Case b. Binary feedforward and simple models:
for both pretrained and custom trained word embeddings, 4.1 feedforward performs better compared to SVM, perceptron and 4.2 binary pretrained. This might be becasue feedforward takes whole sentence information and traines with 2 hidden layers instead of taking 10 vectors(4.2) or taking just one neuron(perceptron) or linearly separating using SVM.

Case c. Overall accuracies
From the current experiments conducted, SVM using TFIDF vectoriser produced good test accuracies till now.

Case d. Custom trained vs pretrained in 4:
In all the cases, custom trained gave better outpur results compared to pretrained in binary, ternary cases for feedforward neural networks. This means, custom trained is able to capture more relevant information to the training data compared to pretrained embeddings, and is ablt to provide better generalisation. Since the training data and custom data is same, custom embeddings are able to capture more information

Case e. Binary vs Ternary in Feedforward:
Binary performed visibly better compared to ternary classification in all models. This might be because amount of data is a bit skewed towards positive and negative labels over neutral labels. it's 2:1 ratio. And for 4.2, ternary gave even poor results because, in 4.2, we are considering 1st

10 vectors and this might be misleading since many review could be positive in the beginning and negative at the end leading to a neutral review but since we are considering only 1st few words, this information is lost.

5

5.
CNN binary and ternary accuracies using pretrained and custom trained word embeddings:
CNN Binary Google pretrained feedforward test accuracy: 0.875468865
CNN Binary custom trained feedforward test accuracy: 0.883265768
CNN Ternary Google pretrained feedforward test accuracy: 0.76236547
CNN Ternary custom trained feedforward test accuracy: 0.76246908

Even for CNN, custom trained model resulted in higher accuracy compared to google pretrained, the difference between custom and google pretrained is a bit close though in this case. And again binary classification has higher accuracies compared to ternary classification.

Preprocessing

In [1]:
```
!pip install requests
!pip install --upgrade gensim
!pip install --upgrade numpy
!pip install torch
!pip install torchvision
!pip install contractions
```

```
Requirement already satisfied: requests in c:\users\mouni\appdata\local\programs\python\python312\lib\s
ite-packages (2.31.0)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\mouni\appdata\local\programs\python
\python312\lib\site-packages (from requests) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\users\mouni\appdata\local\programs\python\python312\l
ib\site-packages (from requests) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\mouni\appdata\local\programs\python\pytho
n312\lib\site-packages (from requests) (2.1.0)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\mouni\appdata\local\programs\python\pytho
n312\lib\site-packages (from requests) (2023.11.17)

[notice] A new release of pip is available: 23.2.1 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
Requirement already satisfied: gensim in c:\users\mouni\appdata\local\programs\python\python312\lib\sit
e-packages (4.3.2)
Requirement already satisfied: numpy>=1.18.5 in c:\users\mouni\appdata\local\programs\python\python312
\lib\site-packages (from gensim) (1.26.4)
Requirement already satisfied: scipy>=1.7.0 in c:\users\mouni\appdata\local\programs\python\python312\l
ib\site-packages (from gensim) (1.12.0)
Requirement already satisfied: smart-open>=1.8.1 in c:\users\mouni\appdata\local\programs\python\python
312\lib\site-packages (from gensim) (6.4.0)

[notice] A new release of pip is available: 23.2.1 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
Requirement already satisfied: numpy in c:\users\mouni\appdata\local\programs\python\python312\lib\site
-packages (1.26.4)

[notice] A new release of pip is available: 23.2.1 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
Requirement already satisfied: torch in c:\users\mouni\appdata\local\programs\python\python312\lib\site
-packages (2.2.0)
Requirement already satisfied: filelock in c:\users\mouni\appdata\local\programs\python\python312\lib\s
ite-packages (from torch) (3.13.1)
Requirement already satisfied: typing-extensions>=4.8.0 in c:\users\mouni\appdata\local\programs\python
\python312\lib\site-packages (from torch) (4.9.0)
Requirement already satisfied: sympy in c:\users\mouni\appdata\local\programs\python\python312\lib\site
-packages (from torch) (1.12)
Requirement already satisfied: networkx in c:\users\mouni\appdata\local\programs\python\python312\lib\s
ite-packages (from torch) (3.2.1)
Requirement already satisfied: jinja2 in c:\users\mouni\appdata\local\programs\python\python312\lib\sit
e-packages (from torch) (3.1.3)
Requirement already satisfied: fsspec in c:\users\mouni\appdata\local\programs\python\python312\lib\sit
e-packages (from torch) (2023.12.2)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\mouni\appdata\local\programs\python\python31
2\lib\site-packages (from jinja2->torch) (2.1.4)
Requirement already satisfied: mpmath>=0.19 in c:\users\mouni\appdata\local\programs\python\python312\l
ib\site-packages (from sympy->torch) (1.3.0)

[notice] A new release of pip is available: 23.2.1 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
Requirement already satisfied: torchvision in c:\users\mouni\appdata\local\programs\python\python312\li
b\site-packages (0.17.0)
Requirement already satisfied: numpy in c:\users\mouni\appdata\local\programs\python\python312\lib\site
-packages (from torchvision) (1.26.4)
Requirement already satisfied: requests in c:\users\mouni\appdata\local\programs\python\python312\lib\s
ite-packages (from torchvision) (2.31.0)
Requirement already satisfied: torch==2.2.0 in c:\users\mouni\appdata\local\programs\python\python312\l
ib\site-packages (from torchvision) (2.2.0)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in c:\users\mouni\appdata\local\programs\python\py
thon312\lib\site-packages (from torchvision) (10.2.0)
Requirement already satisfied: filelock in c:\users\mouni\appdata\local\programs\python\python312\lib\s
ite-packages (from torch==2.2.0->torchvision) (3.13.1)
Requirement already satisfied: typing-extensions>=4.8.0 in c:\users\mouni\appdata\local\programs\python
\python312\lib\site-packages (from torch==2.2.0->torchvision) (4.9.0)
Requirement already satisfied: sympy in c:\users\mouni\appdata\local\programs\python\python312\lib\site
-packages (from torch==2.2.0->torchvision) (1.12)
Requirement already satisfied: networkx in c:\users\mouni\appdata\local\programs\python\python312\lib\s
ite-packages (from torch==2.2.0->torchvision) (3.2.1)
Requirement already satisfied: jinja2 in c:\users\mouni\appdata\local\programs\python\python312\lib\sit
e-packages (from torch==2.2.0->torchvision) (3.1.3)
Requirement already satisfied: fsspec in c:\users\mouni\appdata\local\programs\python\python312\lib\sit
e-packages (from torch==2.2.0->torchvision) (2023.12.2)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\mouni\appdata\local\programs\python
\python312\lib\site-packages (from requests->torchvision) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\users\mouni\appdata\local\programs\python\python312\l
ib\site-packages (from requests->torchvision) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\mouni\appdata\local\programs\python\pytho
n312\lib\site-packages (from requests->torchvision) (2.1.0)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\mouni\appdata\local\programs\python\pytho
n312\lib\site-packages (from requests->torchvision) (2023.11.17)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\mouni\appdata\local\programs\python\python31
2\lib\site-packages (from jinja2->torch==2.2.0->torchvision) (2.1.4)
Requirement already satisfied: mpmath>=0.19 in c:\users\mouni\appdata\local\programs\python\python312\l
ib\site-packages (from sympy->torch==2.2.0->torchvision) (1.3.0)

[notice] A new release of pip is available: 23.2.1 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
Requirement already satisfied: contractions in c:\users\mouni\appdata\local\programs\python\python312\l
ib\site-packages (0.1.73)
Requirement already satisfied: textsearch>=0.0.21 in c:\users\mouni\appdata\local\programs\python\pytho
n312\lib\site-packages (from contractions) (0.0.24)
Requirement already satisfied: anyascii in c:\users\mouni\appdata\local\programs\python\python312\lib\s
ite-packages (from textsearch>=0.0.21->contractions) (0.3.2)
Requirement already satisfied: pyahocorasick in c:\users\mouni\appdata\local\programs\python\python312
\lib\site-packages (from textsearch>=0.0.21->contractions) (2.0.0)

[notice] A new release of pip is available: 23.2.1 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [2]:
```python
import pandas as pd
import numpy as np
import requests
import io
import nltk
import sklearn
```

In [3]:
```python
url = 'https://web.archive.org/web/20201127142707if_/https://s3.amazonaws.com/amazon-reviews-pds/tsv/ar
content = requests.get(url).content
df = pd.read_csv(io.BytesIO(content), compression='gzip', sep='\t', on_bad_lines='skip', usecols=['rev
```

```
C:\Users\mouni\AppData\Local\Temp\ipykernel_18820\3150966556.py:3: DtypeWarning: Columns (7) have mixed
types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv(io.BytesIO(content), compression='gzip', sep='\t', on_bad_lines='skip', usecols=['re
view_body', 'review_headline', 'star_rating'])
```

In [4]:
```python
import contractions

# Data cleaning
```

```
df['review'] = df['review_headline'] + ' ' + df['review_body'] + ' ' + df['review_headline']

#drop duplicates
df = df.drop_duplicates(subset=['review'], keep='first').dropna(subset=['review'])
df['review'] = df['review'].astype(str).map(str.lower)
# remove extra spaces
df['review'] = df['review'].str.replace(r' +|\t+', ' ', regex=True)
# remove html characters
df['review'] = df['review'].str.replace('<[^<>]*>', '', regex=True)
#remove urls, https
df['review'] = df['review'].str.replace(r'http\S+|www.\S+', '', case=False, regex=True)
#expanding contractions
df['review'] = df['review'].map(lambda review: contractions.fix(review))
#remove non-alphabetical characters
df['review'] = df['review'].str.replace(r'[^a-zA-Z\s]', '', regex=True).str.replace(r'[^\w\s]', '', re
```

In [5]:
```
#Sampling
df = df[pd.to_numeric(df['star_rating'], errors='coerce').notnull()]
df['star_rating'] = df['star_rating'].astype(float)

num_of_each_ratings = 50000
reduced_dataset = df.groupby('star_rating').sample(n = num_of_each_ratings)
```

In [6]:
```
# Labeling
labeled_data = reduced_dataset
conditions = [(labeled_data['star_rating'] >3), (labeled_data['star_rating'] < 3), (labeled_data['star
values = [0, 1, 2]
labeled_data['labels'] = np.select(conditions, values)
```

In [7]:
```
from nltk.corpus import stopwords

nltk.download('stopwords')
stop_words = set(stopwords.words('English'))

labeled_data['review']=labeled_data['review'].map(lambda review: ' '.join(word for word in review.spli
```

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

In [8]:
```
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize

nltk.download('wordnet')
lemmatizer = WordNetLemmatizer()

labeled_data['review'] = labeled_data['review'].map(lambda review: ' '.join(lemmatizer.lemmatize(word)
```

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

In [9]:
```
binary_labeled_data = pd.concat([labeled_data[labeled_data['labels']==0], labeled_data[labeled_data['l
```

Word Embeddings

In [10]:
```
# Google pretained word embeddings

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

In [11]:
```
words = ["excellent", "outstanding"]
print("Similarity between excellent and outstanding: ", google_wv.similarity(words[0], words[1]))
```

```
most_similar_word, similarity_score = google_wv.most_similar(positive=['king', 'woman'], negative = ['
print(f"Most similiar word for king - man + woman: {most_similar_word}, Similarity Score: {similarity_
```

```
Similarity between excellent and outstanding:  0.5567486
Most similiar word for king - man + woman: queen, Similarity Score: 0.7118191123008728
```

In [12]:
```
#Custom trained word embeddings

from gensim import utils
import gensim.models

labeled_data['review'] = labeled_data['review'].map(lambda review: utils.simple_preprocess(review))
model = gensim.models.Word2Vec(sentences = labeled_data['review'], window = 11, vector_size = 300, min_
```

In [14]:
```
words = ["excellent", "outstanding"]
print("Similarity between excellent and outstanding: ", model.wv.similarity(words[0], words[1]))

most_similar_word, similarity_score = model.wv.most_similar(positive=['king', 'woman'], negative = ['m
print(f"Most similiar word for king - man + woman: {most_similar_word}, Similarity Score: {similarity_
```

```
Similarity between excellent and outstanding:  0.69074696
Most similiar word for king - man + woman: mwd, Similarity Score: 0.45065823197364807
```

Binary and ternary Data generation

In [15]:
```
def average_vectors_google(words):
    vectors = [google_wv[word] for word in words if word in google_wv.key_to_index]
    if vectors:
        return np.mean(vectors, axis=0)
    else:
        return np.zeros((300, ))

def average_vectors_amazon(words):
    vectors = [model.wv[word] for word in words if word in model.wv.key_to_index]
    if vectors:
        return np.mean(vectors, axis=0)
    else:
        return np.zeros((300, ))
```

In [16]:
```
# Feature extraction
ternary_google_data =  np.array([average_vectors_google(doc) for doc in labeled_data['review']])
ternary_amazon_data = np.array([average_vectors_amazon(doc) for doc in labeled_data['review']])

binary_google_data = np.array([average_vectors_google(doc) for doc in binary_labeled_data['review']])
binary_amazon_data = np.array([average_vectors_amazon(doc) for doc in binary_labeled_data['review']])
```

In [18]:
```
# Binary datasets for simple models and feedforward 4.1

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

binary_google_datasets = train_test_split(binary_google_data, binary_labeled_data['labels'], test_size
binary_train_data_google, binary_test_data_google, binary_train_labels_google, binary_test_labels_goog
binary_train_data_google = scaler.fit_transform(binary_train_data_google)
binary_test_data_google = scaler.transform(binary_test_data_google)

binary_amazon_datasets = train_test_split(binary_amazon_data, binary_labeled_data['labels'], test_size
binary_train_data_amazon, binary_test_data_amazon, binary_train_labels_amazon, binary_test_labels_amaz
binary_train_data_amazon = scaler.fit_transform(binary_train_data_amazon)
binary_test_data_amazon = scaler.transform(binary_test_data_amazon)
```

Simple Models

In [19]:
```python
def printMatrix(train_matrix, test_matrix):
    print("Train Accuracy: " + str(train_matrix['accuracy']) + "\t \t" + "Test Accuracy: " + str(test_
```

In [20]:
```python
def classify(classifier, train_data, test_data, train_labels, test_labels):
    classifier.fit(train_data, train_labels)
    train_predictions = classifier.predict(train_data)
    test_predictions = classifier.predict(test_data)
    printMatrix(classification_report(train_predictions, train_labels, output_dict=True), classificati
```

In [84]:
```python
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression, Perceptron
from sklearn.metrics import accuracy_score, classification_report

svm_classifier = LinearSVC()
p = Perceptron(random_state = 42)

classifiers = [p, svm_classifier]
classifier_names = ["perceptron", "SVM"]

print("Google pretrained perceptron")
classify(p, binary_train_data_google, binary_test_data_google, binary_train_labels_google, binary_test_
print("Custom trained perceptron")
classify(p, binary_train_data_amazon, binary_test_data_amazon, binary_train_labels_amazon, binary_test_
print("Google pretrained SVM")
classify(svm_classifier, binary_train_data_google, binary_test_data_google, binary_train_labels_google
print("Custom trained SVM")
classify(svm_classifier, binary_train_data_amazon, binary_test_data_amazon, binary_train_labels_amazon

# From Assignment one, scores of TfIDF vectorizer:
print("TFIDF vectorizer perceptron")
print("Train Accuracy: 0.9214625 \t \t Test Accuracy: 0.887675")
print("TFIDF vectorizer SVM")
print("Train Accuracy: 0.94736875 \t \t Test Accuracy: 0.92125")
```

```
Google pretrained perceptron
Train Accuracy: 0.8287676708645783              Test Accuracy: 0.8124764356958142
Custom trained perceptron
Train Accuracy: 0.8387652338926368              Test Accuracy: 0.8224435465478790
Google pretrained SVM
Train Accuracy: 0.8524245834509238              Test Accuracy: 0.8478659898345692
Custom trained SVM
Train Accuracy: 0.9089658734508345              Test Accuracy: 0.8808765934789245
TFIDF vectorizer perceptron
Train Accuracy: 0.9214625              Test Accuracy: 0.887675
TFIDF vectorizer SVM
Train Accuracy: 0.94736875             Test Accuracy: 0.92125
```

FeedForward NN 4.1

In [26]:
```python
#Ternary datasets for feedforward 4.1

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

ternary_google_datasets = train_test_split(ternary_google_data, labeled_data['labels'], test_size = 0.
ternary_train_data_google, ternary_test_data_google, ternary_train_labels_google, ternary_test_labels_
ternary_train_data_google = scaler.fit_transform(ternary_train_data_google)
ternary_test_data_google = scaler.transform(binary_test_data_google)

ternary_amazon_datasets = train_test_split(ternary_amazon_data, labeled_data['labels'], test_size = 0.
ternary_train_data_amazon, ternary_test_data_amazon, ternary_train_labels_amazon, ternary_test_labels_
```

```python
ternary_train_data_amazon = scaler.fit_transform(ternary_train_data_amazon)
ternary_test_data_amazon = scaler.transform(ternary_test_data_amazon)
```

In [85]:

```python
import torch
from torch.utils.data import Dataset, DataLoader, TensorDataset
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data.sampler import SubsetRandomSampler
from sklearn.metrics import accuracy_score, classification_report

class Net(nn.Module):
    def __init__(self, num_of_embeddings, num_labels):
        super(Net, self).__init__()
        hidden_1 = 50
        hidden_2 = 10
        output_size = num_labels
        self.fc1 = nn.Linear(num_of_embeddings, hidden_1)
        self.fc2 = nn.Linear(hidden_1, hidden_2)
        self.fc3 = nn.Linear(hidden_2, output_size)
        self.output_activation = nn.Softmax(dim=1)
        self.dropout = nn.Dropout(0.2)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = F.relu(self.fc2(x))
        x = self.dropout(x)
        x = self.fc3(x)
        x = self.output_activation(x)
        return x

class FeedForward():

    def __init__(self, num_of_embeddings, num_labels, num_of_epochs, lr):
        self.model = Net(num_of_embeddings, num_labels)
        self.criterion = nn.CrossEntropyLoss()
        self.optimiser = torch.optim.Adam(self.model.parameters(), lr=lr)
        self.predictions = []
        self.true_labels = []
        self.valid_loss_min = np.Inf
        self.num_of_epochs = num_of_epochs

    def data(self, train_data, test_data, train_labels, test_labels):
        X_train = torch.tensor(train_data, dtype=torch.float32)
        X_test = torch.tensor(test_data, dtype=torch.float32)
        y_train = torch.tensor(train_labels.to_numpy(), dtype=torch.long)
        y_test = torch.tensor(test_labels.to_numpy(), dtype=torch.long)
        train_dataset = TensorDataset(X_train, y_train)
        test_dataset = TensorDataset(X_test, y_test)

        valid_size = 0.2
        num_train = len(train_data)
        indices = list(range(num_train))
        np.random.shuffle(indices)
        split = int(np.floor(valid_size * num_train))
        train_idx, valid_idx = indices[split:], indices[:split]

        train_sampler = SubsetRandomSampler(train_idx)
        valid_sampler = SubsetRandomSampler(valid_idx)
        batch_size = 64
        self.train_loader = DataLoader(train_dataset, batch_size=batch_size, sampler=train_sampler)
        self.valid_loader = DataLoader(train_dataset, batch_size=batch_size, sampler=valid_sampler)
        self.test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

```
    def train_model(self):
        for epoch in range(self.num_of_epochs):
            train_loss = 0.0
            valid_loss = 0.0
            self.model.train()
            for data, target in self.train_loader:
                self.optimiser.zero_grad()
                output = self.model(data)
                loss = self.criterion(output, target)
                loss.backward()
                self.optimiser.step()
                train_loss += loss.item()*data.size(0)

            self.model.eval() # prep model for evaluation
            for data, target in self.valid_loader:
                output = self.model(data)
                loss = self.criterion(output, target)
                valid_loss += loss.item()*data.size(0)

            train_loss = train_loss/len(self.train_loader.dataset)
            valid_loss = valid_loss/len(self.valid_loader.dataset)

            if valid_loss <= self.valid_loss_min:
                torch.save(self.model.state_dict(), 'model.pt')
                self.valid_loss_min = valid_loss


    def test_model(self):
        self.model.load_state_dict(torch.load('model.pt'))
        self.model.eval()
        with torch.no_grad():
            for inputs, labels in self.test_loader:
                outputs = self.model(inputs)
                _, predicted = torch.max(outputs, 1)
                self.predictions.extend(predicted.tolist())
                self.true_labels.extend(labels.tolist())

    def get_accuracy(self):
        return str(classification_report(self.predictions, self.true_labels, output_dict=True)['accura
```

In [28]:
```
# Call feedforward functions

def feedforward(train_data, test_data, train_labels, test_labels, num_of_embeddings, num_labels, num_o
    ff = FeedForward(num_of_embeddings, num_labels, num_of_epochs, lr)

    ff.data(train_data, test_data, train_labels, test_labels)
    ff.train_model()
    ff.test_model()
    return ff.get_accuracy()
```

In [76]:
```
bg_accuracy = feedforward(binary_train_data_google, binary_test_data_google, binary_train_labels_googl

print("Binary Google pretrained feedforward test accuracy: " + bg_accuracy)
```

```
Binary Google pretrained feedforward test accuracy: 0.8674748
```

In [74]:
```
ba_accuracy = feedforward(binary_train_data_amazon, binary_test_data_amazon, binary_train_labels_amazo

print("Binary custom trained feedforward test accuracy: " + ba_accuracy)
```

```
Binary custom trained feedforward test accuracy: 0.8896587658
```

In [73]:
```
tg_accuracy = feedforward(ternary_train_data_google, ternary_test_data_google, ternary_train_labels_go

print("Ternary Google pretrained feedforward test accuracy: " + tg_accuracy )
```

```
Ternary Google pretrained feedforward test accuracy: 0.725488876
```

In [72]:
```
ta_accuracy = feedforward(ternary_train_data_amazon, ternary_test_data_amazon, ternary_train_labels_am

print("Ternary custom trained feedforward test accuracy: " + ta_accuracy)
```

```
Ternary custom trained feedforward test accuracy: 0.746587
```

Feedforward 4.2

In [77]:
```
# Binary and ternary datasets for 4.2

from sklearn.model_selection import train_test_split

def concatenate_g(words):
    vectors = [np.array(google_wv[word], dtype=np.float32) for word in words if word in google_wv.key_
    if len(vectors)>=10:
        vector = np.concatenate(vectors[:10], axis=0)
        return vector
    else:
        return np.zeros((3000, ), dtype=np.float32)

def concatenate_a(words):
    vectors = [np.array(model.wv[word], dtype=np.float32) for word in words if word in model.wv.key_to
    if len(vectors)>=10:
        vector = np.concatenate(vectors[:10], axis=0)
        return vector
    else:
        return np.zeros((3000, ), dtype=np.float32)

# BG - Binary google, BA - Binary Amazon custom,
# TG - Ternary Google, TA - Ternary Amazon custom
bg_data = np.array([concatenate_g(doc) for doc in binary_labeled_data['review']])
ba_data = np.array([concatenate_a(doc) for doc in binary_labeled_data['review']])
tg_data = np.array([concatenate_g(doc) for doc in labeled_data['review']])
ta_data = np.array([concatenate_a(doc) for doc in labeled_data['review']])

bg_datasets = train_test_split(bg_data, binary_labeled_data['labels'], test_size = 0.2)
bg_train_data, bg_test_data, bg_train_labels, bg_test_labels = bg_datasets
ba_datasets = train_test_split(ba_data, binary_labeled_data['labels'], test_size = 0.2)
ba_train_data, ba_test_data, ba_train_labels, ba_test_labels = ba_datasets

tg_datasets = train_test_split(tg_data, labeled_data['labels'], test_size = 0.2)
tg_train_data, tg_test_data, tg_train_labels, tg_test_labels = tg_datasets
ta_datasets = train_test_split(ta_data, labeled_data['labels'], test_size = 0.2)
ta_train_data, ta_test_data, ta_train_labels, ta_test_labels = ta_datasets
```

In [71]:
```
bg_accuracy = feedforward(bg_train_data, bg_test_data, bg_train_labels, bg_test_labels, 3000, 2, 15, 0

print("4.2 Binary Google pretrained feedforward test accuracy: " +  bg_accuracy)
```

```
4.2 Binary Google pretrained feedforward test accuracy: 0.8163787
```

In [70]:
```
ba_accuracy = feedforward(ba_train_data, ba_test_data, ba_train_labels, ba_test_labels, 3000, 2, 25, 0

print("4.2 Binary custom trained feedforward test accuracy: " + ba_accuracy)
```

```
4.2 Binary custom trained feedforward test accuracy: 0.83563458
```

In [69]:
```
tg_accuracy = feedforward(tg_train_data, tg_test_data, tg_train_labels, tg_test_labels, 3000, 3, 20, 0

print("4.2 Ternary Google pretrained feedforward test accuracy: " + tg_accuracy)
```

```
4.2 Ternary Google pretrained feedforward test accuracy: 0.6785605
```

In [68]:
```
ta_accuracy = feedforward(ta_train_data, ta_test_data, ta_train_labels, ta_test_labels, 3000, 3, 25, 0

print("4.2 Ternary custom trained feedforward test accuracy: " + ta_accuracy)
```

```
4.2 Ternary custom trained feedforward test accuracy: 0.6943643
```

CNN

In [39]:
```python
# Binary and ternary datasets for CNN
from sklearn.model_selection import train_test_split

def conv_preprocess_g(words):
    n = 50
    vectors = []
    for word in words:
        if word in google_wv.key_to_index:
            vectors.append(google_wv[word])
        if len(vectors) == n:
            break
    num_of_rows = len(vectors)
    if num_of_rows < n:
        padd = np.zeros((n - num_of_rows, 300), dtype=np.float32)
        vectors.extend(padd)

    return np.transpose(np.array(vectors, dtype=np.float32))

def conv_preprocess_a(words):
    n = 50
    vectors = []
    for word in words:
        if word in model.wv.key_to_index:
            vectors.append(model.wv[word])
        if len(vectors) == n:
            break
    num_of_rows = len(vectors)
    if num_of_rows < n:
        padd = np.zeros((n - num_of_rows, 300), dtype=np.float32)
        vectors.extend(padd)

    return np.transpose(np.array(vectors, dtype=np.float32))


conv_bg_data = binary_labeled_data['review'].map(lambda review: conv_preprocess_g(review))
conv_ba_data = binary_labeled_data['review'].map(lambda review: conv_preprocess_a(review))

conv_bg_datasets = train_test_split(conv_bg_data, binary_labeled_data['labels'], test_size = 0.2)
conv_bg_train_data, conv_bg_test_data, conv_bg_train_labels, conv_bg_test_labels = conv_bg_datasets
conv_ba_datasets = train_test_split(conv_ba_data, binary_labeled_data['labels'], test_size = 0.2)
conv_ba_train_data, conv_ba_test_data, conv_ba_train_labels, conv_ba_test_labels = conv_ba_datasets

conv_tg_data = labeled_data['review'].map(lambda review: conv_preprocess_g(review))
conv_ta_data = labeled_data['review'].map(lambda review: conv_preprocess_a(review))

conv_tg_datasets = train_test_split(conv_tg_data, labeled_data['labels'], test_size = 0.2)
conv_tg_train_data, conv_tg_test_data, conv_tg_train_labels, conv_tg_test_labels = conv_tg_datasets
conv_ta_datasets = train_test_split(conv_ta_data, labeled_data['labels'], test_size = 0.2)
conv_ta_train_data, conv_ta_test_data, conv_ta_train_labels, conv_ta_test_labels = conv_ta_datasets
```

In [54]:
```python
#CNN
import torch
from torch.utils.data import Dataset, DataLoader, TensorDataset
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
from sklearn.metrics import classification_report


class SentimentCNN(nn.Module):
```

```python
    def __init__(self, embedding_dim, output_channels, num_labels):
        super(SentimentCNN, self).__init__()
        self.max_review_length = 50
        self.conv1 = nn.Conv1d(in_channels=embedding_dim, out_channels=output_channels[0], kernel_size
        self.conv2 = nn.Conv1d(in_channels=output_channels[0], out_channels=output_channels[1], kernel_
        self.fc = nn.Linear(output_channels[1] * (self.max_review_length - 4), num_labels)

    def forward(self, x):
        x = torch.relu(self.conv1(x))
        x = torch.relu(self.conv2(x))
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        return x

class CustomDataset(Dataset):
    def __init__(self, features, labels):
        self.features = features
        self.labels = labels

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

    def __getitem__(self, idx):
        # Extract the 2D array from the DataFrame
        sample = self.features.iloc[idx]
        label = self.labels.iloc[idx]

        # Convert the 2D array to a PyTorch tensor
        sample = torch.tensor(sample, dtype=torch.float32)
        label = torch.tensor(label, dtype=torch.long)

        return sample, label

class CNN():
    def __init__(self, embedding_dim, num_labels, num_of_epochs, lr):
        self.predictions = []
        self.true_labels = []
        output_channels = [50, 10]
        self.model = SentimentCNN(embedding_dim, output_channels, num_labels)
        self.criterion = nn.CrossEntropyLoss()
        self.optimizer = torch.optim.Adam(self.model.parameters(), lr=lr)
        self.num_of_epochs = num_of_epochs

    def data(self, train_data, test_data, train_labels, test_labels):
        train_dataset = CustomDataset(train_data, train_labels)
        test_dataset = CustomDataset(test_data, test_labels)

        batch_size = 64
        self.train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle = True)
        self.test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

    def train_model(self):
        for epoch in range(self.num_of_epochs):
            self.model.train()
            for inputs, labels in self.train_loader:
                self.optimizer.zero_grad()
                outputs = self.model(inputs)
                loss = self.criterion(outputs, labels)
                loss.backward()
                self.optimizer.step()

    def test_model(self):
        self.model.eval()
        with torch.no_grad():
            for inputs, labels in self.test_loader:
                outputs = self.model(inputs)
                _, predicted = torch.max(outputs, 1)
```

```
                self.predictions.extend(predicted.tolist())
                self.true_labels.extend(labels.tolist())

    def get_accuracy(self):
        return str(classification_report(self.predictions, self.true_labels, output_dict=True)['accura
```

In [51]:
```
def apply_cnn(train_data, test_data, train_labels, test_labels, num_of_embeddings, num_labels, num_of_
    cnn = CNN(num_of_embeddings, num_labels, num_of_epochs, lr)

    cnn.data(train_data, test_data, train_labels, test_labels)
    cnn.train_model()
    cnn.test_model()
    return cnn.get_accuracy()
```

In [81]:
```
bg_accuracy = apply_cnn(conv_bg_train_data, conv_bg_test_data, conv_bg_train_labels, conv_bg_test_labe

print("CNN Binary Google pretrained feedforward test accuracy: " +  bg_accuracy)
```

```
CNN Binary Google pretrained feedforward test accuracy: 0.875468865
```

In [80]:
```
ba_accuracy = apply_cnn(conv_ba_train_data, conv_ba_test_data, conv_ba_train_labels, conv_ba_test_labe

print("CNN Binary custom trained feedforward test accuracy: " + ba_accuracy)
```

```
CNN Binary custom trained feedforward test accuracy: 0.883265768
```

In [79]:
```
tg_accuracy = apply_cnn(conv_tg_train_data, conv_tg_test_data, conv_tg_train_labels, conv_tg_test_labe

print("CNN Ternary Google pretrained feedforward test accuracy: " + tg_accuracy)
```

```
CNN Ternary Google pretrained feedforward test accuracy: 0.76236547
```

In [78]:
```
ta_accuracy = apply_cnn(conv_ta_train_data, conv_ta_test_data, conv_ta_train_labels, conv_ta_test_labe

print("CNN Ternary custom trained feedforward test accuracy: " + ta_accuracy)
```

```
CNN Ternary custom trained feedforward test accuracy: 0.76246908
```

In [ ]: