HW2-CSCI544-FNN-Part 1,2,3,4

October 5, 2021

[1]:
```python
# import required libraries and methods from them

from platform import python_version

import pandas as pd
import numpy as np

import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')

import re

from bs4 import BeautifulSoup

import contractions

import gensim
import gensim.downloader as api

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Perceptron
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset,DataLoader
```

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

```
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/mrinalkadam/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /Users/mrinalkadam/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

[3]:
```python
# check the python version being used by the jupyter notebook

python_version()
```

[3]: '3.8.5'

[4]:
```python
# read the input dataset into a dataframe

df = pd.read_csv("data.tsv", sep='\t', quoting=3)
df
```

[4]:
```
        marketplace  customer_id       review_id  product_id  product_parent  \
0                US     37000337  R3DT59XH7HXR9K  B00303FI0G       529320574
1                US     15272914  R1LFS11BNASSU8  B00JCZKZN6       274237558
2                US     36137863  R296RT05AG0AF6  B00JLIKA5C       544675303
3                US     43311049  R3V37XDZ7ZCI3L  B000GBNB8G       491599489
4                US     13763148  R14GU232NQFYX2  B00VJ5KX9S       353790155
...             ...          ...             ...         ...             ...
4880461          US     51094108  R22DLC2P26MUMR  B00004SBGS       732420532
4880462          US     50562512  R1N6KLTENLQOMT  B00004SBIA       261705371
4880463          US     52469742  R10TW4QXDV8KJC  B00004SPEF       191184892
4880464          US     51865238   R41RL2U1FSQ4V  B00004RHR6       912491903
4880465          US     52900320  R1NHMPKSJG2E37  B0000021VO        41913389

                                             product_title product_category  \
0                                Arthur Court Paper Towel Holder          Kitchen
1        Olde Thompson Bavaria Glass Salt and Pepper Mi...          Kitchen
2        Progressive International PL8 Professional Man...          Kitchen
3                                    Zyliss Jumbo Garlic Press          Kitchen
4        1 X Premier Pizza Cutter - Stainless Steel 14"...          Kitchen
...                                                    ...              ...
4880461  Le Creuset Enameled Cast-Iron 6-3/4-Quart Oval...          Kitchen
4880462  Le Creuset Enameled Cast-Iron 2-Quart Heart Ca...          Kitchen
4880463           Krups 358-70 La Glaciere Ice Cream Maker          Kitchen
4880464         Hoffritz Stainless-Steel Manual Can Opener          Kitchen
4880465                                       Tammy Rogers          Kitchen

         star_rating  helpful_votes  total_votes vine verified_purchase  \
0                  5              0            0    N                 Y
```

```
1                    5              0             1    N                  Y
2                    5              0             0    N                  Y
3                    5              0             1    N                  Y
4                    5              0             0    N                  Y
...                ...            ...           ...  ...                ...
4880461              4             30            41    N                  N
4880462              5             84            92    N                  N
4880463              4             55            60    N                  N
4880464              4             30            42    N                  N
4880465              5              5             5    N                  N

                                  review_headline  \
0                 Beautiful. Looks great on counter
1                                Awesome & Self-ness
2                     Fabulous and worth every penny
3                                         Five Stars
4                                    Better than sex
...                                              ...
4880461                  Not as sturdy as you'd think.
4880462                      A Sweetheart of A Pan
4880463                       Ice Cream Like a Dream
4880464              Opens anything and everything
4880465  The more you listen, the more you hear...

                                                review_body review_date
0                      Beautiful.  Looks great on counter.  2015-08-31
1         I personally have 5 days sets and have also bo...  2015-08-31
2         Fabulous and worth every penny. Used for clean...  2015-08-31
3         A must if you love garlic on tomato marinara s...  2015-08-31
4         Worth every penny! Buy one now and be a pizza ...  2015-08-31
...                                                     ...         ...
4880461  After a month of heavy use, primarily as a chi...  2000-04-28
4880462  I've used my Le Creuset enameled cast iron coo...  2000-04-28
4880463  According to my wife, this is \\"the best birt...  2000-04-28
4880464  Hoffritz has a name of producing a trendy and ...  2000-04-24
4880465  OK. I was late to snap to the Dead Reckoners. ...  2000-01-20

[4880466 rows x 15 columns]
```

1 1. Dataset Generation

[5]:
```
# keep only reviews and ratings columns

df = df[["review_body","star_rating"]]
df
```

[5]:
```
                                               review_body  star_rating
0                        Beautiful.  Looks great on counter.            5
1          I personally have 5 days sets and have also bo...            5
2          Fabulous and worth every penny. Used for clean...            5
3          A must if you love garlic on tomato marinara s...            5
4          Worth every penny! Buy one now and be a pizza ...            5
...                                                      ...          ...
4880461  After a month of heavy use, primarily as a chi...            4
4880462  I've used my Le Creuset enameled cast iron coo...            5
4880463  According to my wife, this is \\"the best birt...            4
4880464  Hoffritz has a name of producing a trendy and ...            4
4880465  OK. I was late to snap to the Dead Reckoners. ...            5

[4880466 rows x 2 columns]
```

[6]:
```python
# find out the number of reviews falling under each distinct rating

df['star_rating'].value_counts()
```

[6]:
```
5    3128564
4     732471
1     427306
3     349929
2     242196
Name: star_rating, dtype: int64
```

[7]:
```python
# check for null values in the reviews column

df['review_body'].isnull().sum()
```

[7]:
```
243
```

[8]:
```python
# check for null values in the ratings column

df['star_rating'].isnull().sum()
```

[8]:
```
0
```

[9]:
```python
# drop null value records from the dataframe

df.dropna(inplace=True)
```

```
<ipython-input-9-ba0c96652bb5>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df.dropna(inplace=True)
```

[10]:
```python
# find out records with star ratings 1,2,3,4 and 5 and select 50000 records randomly per each rating score

df_1 = df[df['star_rating']==1].sample(n=50000, random_state=100)
df_2 = df[df['star_rating']==2].sample(n=50000, random_state=100)
df_3 = df[df['star_rating']==3].sample(n=50000, random_state=100)
df_4 = df[df['star_rating']==4].sample(n=50000, random_state=100)
df_5 = df[df['star_rating']==5].sample(n=50000, random_state=100)

# concat the above records together to get a sample of 250000 reviews

df = pd.concat([df_1,df_2,df_3,df_4,df_5]).reset_index()

# shuffle the dataset

df = df.sample(frac=1).reset_index()
df.drop(['index','level_0'],axis=1,inplace=True)
df
```

[10]:
```
                                              review_body  star_rating
0       Not pleased with the &#34;threads&#34; it crea...            2
1       fills one  16.9 ounce water bottle with two tr...            4
2       We got a similar waffle iron from Betty Crocke...            3
3       I have Lock N Locks and I hate keeping up with...            2
4       I bought this at the beginning of 9/13 even th...            1
...                                                   ...          ...
249995  As with all coffee 'grinders' that are actuall...            2
249996  Love the design of this shaker and have ordere...            2
249997  This mold is made of plastic and is a bit flim...            4
249998  I really love this yogurt maker. I made very t...            5
249999                                Don't waste your money            1

[250000 rows x 2 columns]
```

[11]:
```python
# find out the number of reviews falling under distinct ratings now

print("Positive, Negative, Neutral Reviews Count:")
print(df[((df['star_rating']==4.0) | (df['star_rating']==5.0))]['star_rating'].count(),",",df[((df['star_rating']==1.0) | (df['star_rating']==2.0))]['star_rating'].count(),",",df[df['star_rating']==3.0]['star_rating'].count())
```

```
Positive, Negative, Neutral Reviews Count:
100000 , 100000 , 50000
```

[12]:
```python
# label reviews falling under ratings 4 and 5 as 1(positive class), under ratings 1 and 2 as 2(negative class), and under rating 3 as 3(neutral class)
```

```
df['class'] = np.where(((df['star_rating']==4) | (df['star_rating']==5)),1,0)
df['class'] = np.where(((df['star_rating']==1) |␣
↪(df['star_rating']==2)),2,df['class'])
df['class'] = np.where((df['star_rating']==3),3,df['class'])
df
```

[12]:

```
                                              review_body  star_rating  class
0       Not pleased with the &#34;threads&#34; it crea...            2      2
1       fills one  16.9 ounce water bottle with two tr...            4      1
2       We got a similar waffle iron from Betty Crocke...            3      3
3       I have Lock N Locks and I hate keeping up with...            2      2
4       I bought this at the beginning of 9/13 even th...            1      2
...                                                   ...          ...    ...
249995  As with all coffee 'grinders' that are actuall...            2      2
249996  Love the design of this shaker and have ordere...            2      2
249997  This mold is made of plastic and is a bit flim...            4      1
249998  I really love this yogurt maker. I made very t...            5      1
249999                                Don't waste your money            1      2

[250000 rows x 3 columns]
```

[13]:

```
# drop the rating column once you have the label('class') column

df.drop(['star_rating'],axis=1,inplace=True)
df
```

[13]:

```
                                              review_body  class
0       Not pleased with the &#34;threads&#34; it crea...      2
1       fills one  16.9 ounce water bottle with two tr...      1
2       We got a similar waffle iron from Betty Crocke...      3
3       I have Lock N Locks and I hate keeping up with...      2
4       I bought this at the beginning of 9/13 even th...      2
...                                                   ...    ...
249995  As with all coffee 'grinders' that are actuall...      2
249996  Love the design of this shaker and have ordere...      2
249997  This mold is made of plastic and is a bit flim...      1
249998  I really love this yogurt maker. I made very t...      1
249999                                Don't waste your money      2

[250000 rows x 2 columns]
```

[14]:

```
# make a copy of the original data frame(without any data cleaning)

df_uncleaned = df.copy(deep = True)
df_uncleaned
```

[14]:
```
                                             review_body  class
0       Not pleased with the &#34;threads&#34; it crea...      2
1       fills one  16.9 ounce water bottle with two tr...      1
2       We got a similar waffle iron from Betty Crocke...      3
3       I have Lock N Locks and I hate keeping up with...      2
4       I bought this at the beginning of 9/13 even th...      2
...                                                   ...    ...
249995  As with all coffee 'grinders' that are actuall...      2
249996  Love the design of this shaker and have ordere...      2
249997  This mold is made of plastic and is a bit flim...      1
249998  I really love this yogurt maker. I made very t...      1
249999                               Don't waste your money      2

[250000 rows x 2 columns]
```

2 2. Word Embedding

3 (a)

[23]:
```python
# load the google news word2vec model

wv = api.load('word2vec-google-news-300')
```

[24]:
```python
# find out the vectors for different words using the above model

vec_King = wv['King']
vec_Man = wv['Man']
vec_Woman = wv['Woman']
vec_Queen = wv['Queen']

vec_1 = vec_King-vec_Man+vec_Woman
vec_2 = vec_Queen

# find out the similarity of the vectors using 'most_similar' function

print(wv.most_similar(positive=['King','Woman'], negative=['Man'], topn=1))
print('\n')

# find out the similarity of the vectors using cosine similarity

cosine_similarity = np.dot(vec_1,vec_2)/(np.linalg.norm(vec_1)* np.linalg.
 ↪norm(vec_2))
print("Semantic(Cosine) similarity between the two vectors is:
 ↪",cosine_similarity)
```

```
[('Queen', 0.4929388165473938)]
```

```
Semantic(Cosine) similarity between the two vectors is: 0.44240144
```

[25]:

```
# find out the similarity of the words

print('%r\t%r\t%.2f' % (w1, w2, wv.similarity('excellent', 'outstanding')))
```

```
'excellent'     'outstanding'   0.56
```

4 (b)

[26]:

```
##### REMOVE FROM COMMENT LATER

words = [row.split(' ') for row in df['review_body']]

# train your own word2vec model

model = gensim.models.Word2Vec(words, min_count=10,size=300,workers=3,␣
 ↪window=11, sg=1)

# summarize the loaded model

print(model)
```

[27]:

```
# save model

model.save('model.bin')

# load saved model

final_model = gensim.models.Word2Vec.load('model.bin')
print(final_model)
```

```
Word2Vec(vocab=34607, size=300, alpha=0.025)
```

[28]:

```
vec_King = final_model['King']
vec_Man = final_model['Man']
vec_Woman = final_model['Woman']
vec_Queen = final_model['Queen']

vec_1 = vec_King-vec_Man+vec_Woman
vec_2 = vec_Queen

# find out the similarity of the vectors using 'most_similar' function
```

```
print(final_model.most_similar(positive=['King','Woman'], negative=['Man'],␣
 ↪topn=1))
print('\n')

# find out the similarity of the vectors using cosine similarity

cosine_similarity = np.dot(vec_1,vec_2)/(np.linalg.norm(vec_1)* np.linalg.
 ↪norm(vec_2))
print("Semantic(Cosine) similarity between the two vectors is:
 ↪",cosine_similarity)
```

```
[('Arthur', 0.5806456208229065)]



Semantic(Cosine) similarity between the two vectors is: 0.4015107
```

```
<ipython-input-28-1cf12555a4bb>:1: DeprecationWarning: Call to deprecated
`__getitem__` (Method will be removed in 4.0.0, use self.wv.__getitem__()
instead).
  vec_King = final_model['King']
<ipython-input-28-1cf12555a4bb>:2: DeprecationWarning: Call to deprecated
`__getitem__` (Method will be removed in 4.0.0, use self.wv.__getitem__()
instead).
  vec_Man = final_model['Man']
<ipython-input-28-1cf12555a4bb>:3: DeprecationWarning: Call to deprecated
`__getitem__` (Method will be removed in 4.0.0, use self.wv.__getitem__()
instead).
  vec_Woman = final_model['Woman']
<ipython-input-28-1cf12555a4bb>:4: DeprecationWarning: Call to deprecated
`__getitem__` (Method will be removed in 4.0.0, use self.wv.__getitem__()
instead).
  vec_Queen = final_model['Queen']
<ipython-input-28-1cf12555a4bb>:11: DeprecationWarning: Call to deprecated
`most_similar` (Method will be removed in 4.0.0, use self.wv.most_similar()
instead).
  print(final_model.most_similar(positive=['King','Woman'], negative=['Man'],
topn=1))
```

[29]:

```
# find out the similarity of the words

print('%r\t%r\t%.2f' % (w1, w2, final_model.similarity('excellent',␣
 ↪'outstanding')))
```

```
'excellent'     'outstanding'   0.67
```

```
<ipython-input-29-2c93295bc140>:3: DeprecationWarning: Call to deprecated
`similarity` (Method will be removed in 4.0.0, use self.wv.similarity()
instead).
```

```
  print('%r\t%r\t%.2f' % (w1, w2, final_model.similarity('excellent',
'outstanding')))
```

5 Comments about this question

As seen from above,the vectors generated by our word2vec model are able to encode semantic similarities better between the words 'excellent' and 'outstanding'(Word2Vec-0.67, Google-0.56). However the google model does slighlty better when it comes to the case of 'King - Man + Woman' and 'Queen'(Google-0.44, Word2Vec-0.40). Also it can be noted that the most similar word predicted to 'King - Man + Woman' is 'Queen' by the Google model but 'Arthur' by our model. This might likely be because the Google model has a larger word vocabulary and contains more common words. Also, since we have taken these parameters for our Word2Vec model -(min_count=10,size=300,workers=3, window=11) , it isn't as refined as it could be potentially, thus leading to slightly low results in some cases.

6 3. Simple models

7 Data Cleaning

[18]:
```
# convert the reviews column to string type

df['review_body'] = df['review_body'].astype(str)

# convert the reviews column to lower case

df['review_body'] = df['review_body'].str.lower()

# using BeautifulSoup, remove HTML tags from the reviews column

# function to remove HTML tags
def remove_html(string):

    # parse through html content
    bs = BeautifulSoup(string, "html.parser")

    for text in bs(['style', 'script']):
        # remove the tags
        text.decompose()

    # return data by retrieving the tag content
    return ' '.join(bs.stripped_strings)

# apply the remove_html function to the reviews column

df['review_body']=df['review_body'].apply(lambda x : remove_html(x))
```

```python
# using RegEx, remove URLs from the reviews column

# function to remove URLS
def remove_url(string):
    result = re.sub(r'^https?:\/\/.*[\r\n]*',r' ', string, flags=re.MULTILINE)
    return result

# apply the remove_url function to the reviews column

df['review_body']=df['review_body'].apply(lambda x : remove_url(x))

# using RegEx, remove the characters apart from alphabets and single␣
↪apostrophe(required for contractions later) from the reviews column and␣
↪replace them with a single space

df['review_body'] = df['review_body'].replace(r"[^a-zA-Z' ]\s?"," ",regex=True)

# replace the single apostrophe with no space

df['review_body'] = df['review_body'].replace("'","",regex=True)

# using RegEx, remove the extra spaces between words from the reviews column

df['review_body'] = df['review_body'].replace('\s+', ' ', regex=True)

# using the contractions library, perform contractions on the reviews

df['review_body'] = df['review_body'].apply(lambda x: [contractions.fix(word)␣
↪for word in x.split()])
df['review_body'] = [' '.join(map(str, d)) for d in df['review_body']]

df
```

```
/opt/anaconda3/lib/python3.8/site-packages/bs4/__init__.py:417:
MarkupResemblesLocatorWarning: "http://www.amazon.com/review/create-
review/ref=cm_cr_dp_wrt_summary?ie=utf8&asin=b00xp0d9p0" looks like a URL.
Beautiful Soup is not an HTTP client. You should probably use an HTTP client
like requests to get the document behind the URL, and feed that document to
Beautiful Soup.
  warnings.warn(
/opt/anaconda3/lib/python3.8/site-packages/bs4/__init__.py:332:
MarkupResemblesLocatorWarning: "." looks like a filename, not markup. You should
probably open this file and pass the filehandle into Beautiful Soup.
  warnings.warn(
```

[18]:

	review_body	class
0	i assumed there were four chargers when i boug...	2

```
1       my son likes to cook hes especially good with ...      1
2       shipped fast good price they were way huger th...      1
3       containers are great but the lids are very thi...      3
4       item was received broken i returned it and ask...      2
...                                                   ...    ...
249995  the locks come off easily and they are hard to...      3
249996  i was bummed the carafe is slightly too wide a...      2
249997  I have had this kettle for just over one month...      2
249998  the idea and color of the balloons is enticing...      2
249999  product failed almost immediately digits garbl...      2

[250000 rows x 2 columns]
```

8 Pre-processing

[19]:
```python
# remove all general stop words from the reviews column

stop_words = stopwords.words('english')
df['review_body'] = df['review_body'].apply(lambda x: ' '.join([word for word in␣
↪x.split() if word not in (stop_words)]))

# perform lemmatization with POS tagging

whitespace_tokenizer = nltk.tokenize.WhitespaceTokenizer()
wordnet_lemmatizer = nltk.stem.WordNetLemmatizer()

# funtion to return a POS form of a word
def pos(word):
    """Map POS tag to first character lemmatize() accepts"""
    pos_tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dictionary = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}

    return tag_dictionary.get(pos_tag, wordnet.NOUN)

# function to lemmatize the text
def lemmatize_text(string):
    return [wordnet_lemmatizer.lemmatize(w,pos(w)) for w in whitespace_tokenizer.
↪tokenize(string)]

df['review_body'] = df['review_body'].apply(lemmatize_text)
df['review_body'] = [' '.join(map(str, l)) for l in df['review_body']]

df
```

[19]:
```
                                             review_body  class
0       assume four charger bought item pretty bought ...      2
1       son like cook he especially good grill burger ...      1
2                    ship fast good price way huger expect      1
3           container great lid thin break easily one use      3
4       item receive broken return ask replacement shi...      2
...                                                  ...    ...
249995                      lock come easily hard clean top      3
249996  bum carafe slightly wide bit short metal struc...      2
249997  I kettle one month leak water leak seal bottom...      2
249998  idea color balloon entice order package child ...      2
249999  product fail almost immediately digit garble s...      2

[250000 rows x 2 columns]
```

[20]:
```python
# Subtract target class values by 1 so that it becomes easier later on while
↪comparison

df['class'] = df['class']-1
df
```

[20]:
```
                                              review_body  class
0       assume four charger bought item pretty bought ...      1
1       son like cook he especially good grill burger ...      0
2                     ship fast good price way huger expect      0
3                 container great lid thin break easily one use      2
4       item receive broken return ask replacement shi...      1
...                                                   ...    ...
249995                     lock come easily hard clean top      2
249996  bum carafe slightly wide bit short metal struc...      1
249997  I kettle one month leak water leak seal bottom...      1
249998  idea color balloon entice order package child ...      1
249999  product fail almost immediately digit garble s...      1

[250000 rows x 2 columns]
```

[21]:
```python
# function to find the average of vectors as your input feature

def find_average_of_vectors(review,model_used):

    sentence_words = review.split(" ")

    sentence_vectors = []
    for word in sentence_words:
        try:
            sentence_vectors.append(model_used[word])
        except:
            continue

    if len(sentence_vectors)!=0:
        return (np.mean(sentence_vectors,axis=0)).flatten()
    else:
        return np.zeros((300,))
```

9 Binary

[22]:
```python
# make a copy of the original data frame(with data cleaning)

df_org_3 = df.copy(deep=True)
df_org_3
```

[22]:

```
                                              review_body  class
0       assume four charger bought item pretty bought ...      1
1       son like cook he especially good grill burger ...      0
2                     ship fast good price way huger expect      0
3           container great lid thin break easily one use      2
4       item receive broken return ask replacement shi...      1
...                                                   ...    ...
249995                      lock come easily hard clean top      2
249996  bum carafe slightly wide bit short metal struc...      1
249997  I kettle one month leak water leak seal bottom...      1
249998  idea color balloon entice order package child ...      1
249999  product fail almost immediately digit garble s...      1

[250000 rows x 2 columns]
```

[23]:

```python
# find input feature for google model

df_org_3['avg_input_features_1'] = df_org_3['review_body'].apply(lambda x:␣
 ↪find_average_of_vectors(x,wv))
df_org_3
```

[23]:

```
                                              review_body  class  \
0       assume four charger bought item pretty bought ...      1
1       son like cook he especially good grill burger ...      0
2                     ship fast good price way huger expect      0
3           container great lid thin break easily one use      2
4       item receive broken return ask replacement shi...      1
...                                                   ...    ...
249995                      lock come easily hard clean top      2
249996  bum carafe slightly wide bit short metal struc...      1
249997  I kettle one month leak water leak seal bottom...      1
249998  idea color balloon entice order package child ...      1
249999  product fail almost immediately digit garble s...      1

                                     avg_input_features_1
0       [0.04277208, -0.03597005, -0.062435575, 0.1046...
1       [-0.004893621, 0.029286703, -0.01199023, 0.162...
2       [0.1432408, 0.08569336, -0.048673358, 0.078264...
3       [0.056274414, 0.10064697, -0.0005340576, 0.056...
4       [0.043584187, -0.013412476, -0.116475426, 0.06...
...                                                   ...
249995  [0.03120931, 0.07987467, 0.03741455, 0.0357869...
249996  [-0.001551011, 0.026309744, -0.06418026, 0.125...
249997  [0.0027923584, 0.092679344, -0.03684489, 0.028...
249998  [0.047094908, 0.011726828, 0.00012925093, 0.09...
249999  [0.085134655, -0.011324369, 0.06199294, 0.0255...
```

```
[250000 rows x 3 columns]
```

[24]:
```python
# find input feature for our model

df_org_3['avg_input_features_2'] = df_org_3['review_body'].apply(lambda x:␣
↪find_average_of_vectors(x,final_model))
df_org_3
```

```
<ipython-input-21-6192696cc0bb>:10: DeprecationWarning: Call to deprecated
`__getitem__` (Method will be removed in 4.0.0, use self.wv.__getitem__()
instead).
  sentence_vectors.append(model_used[word])
```

[24]:
```
                                              review_body  class  \
0       assume four charger bought item pretty bought ...      1
1       son like cook he especially good grill burger ...      0
2                    ship fast good price way huger expect      0
3                container great lid thin break easily one use      2
4       item receive broken return ask replacement shi...      1
...                                                   ...    ...
249995                     lock come easily hard clean top      2
249996  bum carafe slightly wide bit short metal struc...      1
249997  I kettle one month leak water leak seal bottom...      1
249998  idea color balloon entice order package child ...      1
249999  product fail almost immediately digit garble s...      1

                                     avg_input_features_1  \
0       [0.04277208, -0.03597005, -0.062435575, 0.1046...
1       [-0.004893621, 0.029286703, -0.01199023, 0.162...
2       [0.1432408, 0.08569336, -0.048673358, 0.078264...
3       [0.056274414, 0.10064697, -0.0005340576, 0.056...
4       [0.043584187, -0.013412476, -0.116475426, 0.06...
...                                                   ...
249995  [0.03120931, 0.07987467, 0.03741455, 0.0357869...
249996  [-0.001551011, 0.026309744, -0.06418026, 0.125...
249997  [0.0027923584, 0.092679344, -0.03684489, 0.028...
249998  [0.047094908, 0.011726828, 0.00012925093, 0.09...
249999  [0.085134655, -0.011324369, 0.06199294, 0.0255...

                                     avg_input_features_2
0       [0.017703589, -0.11186184, -0.0030522645, -0.0...
1       [0.120273024, -0.14361034, 0.046780374, -0.138...
2       [-0.049596105, -0.018341891, 0.13302507, -0.17...
3       [0.030435072, -0.15327847, 0.11309578, -0.1425...
4       [0.08915458, -0.22801971, -0.028520422, -0.263...
...                                                   ...
249995  [0.015699785, -0.12990652, 0.21889718, -0.1027...
```

```
249996   [0.015504825, -0.031771064, 0.1092756, -0.0557...
249997   [0.020719932, -0.090553395, 0.13070571, -0.027...
249998   [0.066825956, -0.17564225, 0.05628306, -0.0763...
249999   [0.0051919767, -0.1441225, 0.13658296, -0.1857...

[250000 rows x 4 columns]
```

[25]:

```python
# binary classification dataframe

df_binary = df_org_3[((df_org_3['class'] == 0) | (df_org_3['class'] == 1))]
df_binary
```

[25]:

```
                                              review_body  class  \
0       assume four charger bought item pretty bought ...      1
1       son like cook he especially good grill burger ...      0
2                    ship fast good price way huger expect      0
4       item receive broken return ask replacement shi...      1
5       experience issue one cup fill make sure filter...      0
...                                                   ...    ...
249993  toaster oven fine especially since paid amazon...      1
249996  bum carafe slightly wide bit short metal struc...      1
249997  I kettle one month leak water leak seal bottom...      1
249998  idea color balloon entice order package child ...      1
249999  product fail almost immediately digit garble s...      1

                                     avg_input_features_1  \
0       [0.04277208, -0.03597005, -0.062435575, 0.1046...
1       [-0.004893621, 0.029286703, -0.01199023, 0.162...
2       [0.1432408, 0.08569336, -0.048673358, 0.078264...
4       [0.043584187, -0.013412476, -0.116475426, 0.06...
5       [0.0077209473, -0.015841166, -0.04876624, 0.11...
...                                                   ...
249993  [0.03401947, 0.05153087, -0.0007176717, 0.0253...
249996  [-0.001551011, 0.026309744, -0.06418026, 0.125...
249997  [0.0027923584, 0.092679344, -0.03684489, 0.028...
249998  [0.047094908, 0.011726828, 0.00012925093, 0.09...
249999  [0.085134655, -0.011324369, 0.06199294, 0.0255...

                                     avg_input_features_2
0       [0.017703589, -0.11186184, -0.0030522645, -0.0...
1       [0.120273024, -0.14361034, 0.046780374, -0.138...
2       [-0.049596105, -0.018341891, 0.13302507, -0.17...
4       [0.08915458, -0.22801971, -0.028520422, -0.263...
5       [0.0042549637, -0.026836593, 0.14918885, -0.08...
...                                                   ...
249993  [0.050901376, -0.11194899, 0.12081799, -0.0080...
249996  [0.015504825, -0.031771064, 0.1092756, -0.0557...
```

```
249997   [0.020719932, -0.090553395, 0.13070571, -0.027...
249998   [0.066825956, -0.17564225, 0.05628306, -0.0763...
249999   [0.0051919767, -0.1441225, 0.13658296, -0.1857...

[200000 rows x 4 columns]
```

10 Google Model

[30]:
```
x = df_binary['avg_input_features_1']
y = df_binary['class']

# Split the dataset into 80% training dataset and 20% testing dataset

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20,␣
↪random_state=100)

x_train = x_train.tolist()
x_test = x_test.tolist()
```

[31]:
```
# train a Perceptron model on the training dataset

perceptron = Perceptron(n_jobs=-1, random_state=100)
perceptron.fit(x_train,y_train)

# predict the labels of test values

y_test_pred = perceptron.predict(x_test)

# find the accuracy of the Perceptron model on the test set

print("-----------------------Test-----------------------")
print('\n')
print("Accuracy of Perceptron Model:",accuracy_score(y_test, y_test_pred))
```

```
-----------------------Test-----------------------


Accuracy of Perceptron Model: 0.710925
```

[32]:
```
# standardize the features using StandardScaler

scalar = StandardScaler()
x_train_std = scalar.fit_transform(x_train)
x_test_std = scalar.transform(x_test)

# train an SVM model on the training dataset
```

```
lin_svc = LinearSVC(random_state=100)
lin_svc.fit(x_train_std,y_train)

# predict the labels of test values

y_test_pred = lin_svc.predict(x_test_std)

# find the accuracy of the SVM model on the test set

print("-----------------------Test-----------------------")
print('\n')
print("Accuracy of SVM Model:",accuracy_score(y_test, y_test_pred))
```

```
-----------------------Test-----------------------


Accuracy of SVM Model: 0.819275
```

```
/opt/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.py:976:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  warnings.warn("Liblinear failed to converge, increase "
```

11 Our model

[33]:
```
x = df_binary['avg_input_features_2']
y = df_binary['class']

# Split the dataset into 80% training dataset and 20% testing dataset

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20,␣
↪random_state=100)

x_train = x_train.tolist()
x_test = x_test.tolist()
```

[34]:
```
# train a Perceptron model on the training dataset

perceptron = Perceptron(n_jobs=-1, random_state=100)
perceptron.fit(x_train,y_train)

# predict the labels of test values

y_test_pred = perceptron.predict(x_test)

# find the accuracy of the Perceptron model on the test set
```

```
print("-----------------------Test-----------------------")
print('\n')
print("Accuracy of Perceptron Model:",accuracy_score(y_test, y_test_pred))
```

```
-----------------------Test-----------------------


Accuracy of Perceptron Model: 0.811125
```

[35]:
```
# standardize the features using StandardScaler

scalar = StandardScaler()
x_train_std = scalar.fit_transform(x_train)
x_test_std = scalar.transform(x_test)

# train an SVM model on the training dataset

lin_svc = LinearSVC(random_state=100)
lin_svc.fit(x_train_std,y_train)

# predict the labels of test values

y_test_pred = lin_svc.predict(x_test_std)

# find the accuracy of the SVM model on the test set

print("-----------------------Test-----------------------")
print('\n')
print("Accuracy of SVM Model:",accuracy_score(y_test, y_test_pred))
```

```
-----------------------Test-----------------------


Accuracy of SVM Model: 0.85065
```

```
/opt/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.py:976:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  warnings.warn("Liblinear failed to converge, increase "
```

12 Comments about this question

[33]:
```
d = {'Model': ['Perceptron', 'SVM', 'Perceptron', 'SVM', 'Perceptron', 'SVM'],
     'Word2Vec Features/Other Features': ['Google News', 'Google News', 'Amazon␣
 ↪Reviews(Our)', 'Amazon Reviews(Our)', 'TF-IDF', 'TF-IDF'],
     'Accuracy': ['0.71', '0.82', '0.81', '0.85', '0.85', '0.81']}
```

```
df_results_part_3 = pd.DataFrame(data=d)
df_results_part_3
```

[33]:

```
        Model  Word2Vec Features/Other Features  Accuracy
0  Perceptron                       Google News      0.71
1         SVM                       Google News      0.82
2  Perceptron               Amazon Reviews(Our)      0.81
3         SVM               Amazon Reviews(Our)      0.85
4  Perceptron                            TF-IDF      0.85
5         SVM                            TF-IDF      0.81
```

It can be seen from the above table that the TF-IDF feature types give us the best accuracy for the perceptron model, followed by the our trained Word2Vec and then Google Word2Vec. However for the SVM model, the best accuracy is given by our trained Word2Vec, follwed by Google Word2Vec and then TF-IDF. This unstable order shows us that different features work for different models the best and there is no 'one glove fits all' / 'free lunch theorem' concept in the real world. Trying out different features and then choosing what works the best for that model(good feature selection) should be our optimal solution.

13 4. Feedforward Neural Networks

[21]:

```
# find out if GPU available

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cpu
```

[33]:

```
# set hyperparameters for all the models

EPOCHS = 100
BATCH_SIZE = 20
LEARNING_RATE = 0.001
```

[34]:

```
## train data
class trainData(Dataset):

    def __init__(self, x_data, y_data):
        self.x_data = x_data
        self.y_data = y_data

    def __getitem__(self, index):
        return self.x_data[index], self.y_data[index]

    def __len__ (self):
        return len(self.x_data)
```

```
## test data
class testData(Dataset):

    def __init__(self, X_data):
        self.X_data = X_data

    def __getitem__(self, index):
        return self.X_data[index]

    def __len__ (self):
        return len(self.X_data)
```

14 (a)

15 Binary

[40]:
```
# set parameters

input_size = 300
hidden_1_size = 50
hidden_2_size = 10
output_size = 1
```

[41]:
```
# model for binary classification

class binary_classification(nn.Module):
    def __init__(self):
        super(binary_classification, self).__init__()

        self.layer_1 = nn.Linear(input_size, hidden_1_size)
        self.layer_2 = nn.Linear(hidden_1_size, hidden_2_size)
        self.layer_out = nn.Linear(hidden_2_size, output_size)

        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(p=0.1)
        self.batchnorm1 = nn.BatchNorm1d(hidden_1_size)
        self.batchnorm2 = nn.BatchNorm1d(hidden_2_size)

    def forward(self, x):
        x = self.relu(self.layer_1(x))
        x = self.batchnorm1(x)
        x = self.relu(self.layer_2(x))
        x = self.batchnorm2(x)
        x = self.dropout(x)
```

```
        x = self.layer_out(x)

        return x
```

[42]:
```
# print model

model = binary_classification()
print(model)
```

```
binary_classification(
  (layer_1): Linear(in_features=300, out_features=50, bias=True)
  (layer_2): Linear(in_features=50, out_features=10, bias=True)
  (layer_out): Linear(in_features=10, out_features=1, bias=True)
  (relu): ReLU()
  (dropout): Dropout(p=0.1, inplace=False)
  (batchnorm1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (batchnorm2): BatchNorm1d(10, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
```

[43]:
```
# define loss function and optimizer

criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)
```

[44]:
```
# function to find the accuracy of the binary model

def binary_acc(y_pred, y_test):
    y_pred_tag = torch.round(torch.sigmoid(y_pred))

    correct_results_sum = (y_pred_tag == y_test).sum().float()
    acc = correct_results_sum/y_test.shape[0]
    acc = torch.round(acc * 100)

    return acc
```

[45]:
```
# function to train the binary model and print results(loss & accuracy per epoch)

def train_model_binary():
    model.train()
    for e in range(1, EPOCHS+1):
        epoch_loss = 0
        epoch_acc = 0

        for x_batch, y_batch in train_loader:
            x_batch, y_batch = x_batch, y_batch
```

```
            optimizer.zero_grad()

            y_pred = model(x_batch)

            loss = criterion(y_pred, y_batch.unsqueeze(1))
            acc = binary_acc(y_pred, y_batch.unsqueeze(1))

            loss.backward()
            optimizer.step()

            epoch_loss += loss.item()
            epoch_acc += acc.item()


        print(f'Epoch {e+0:03}: | Loss: {epoch_loss/len(train_loader):.5f} | Acc:
 ↪ {epoch_acc/len(train_loader):.3f}')
```

[46]:
```
def test_model_binary(y_test):
    model.eval()

    y_pred_list = []

    with torch.no_grad():
        for x_batch in test_loader:
            y_test_pred = model(x_batch)
            y_pred_list.append(y_test_pred)

    y_pred_list = torch.FloatTensor(y_pred_list)
    y_test = torch.FloatTensor(y_test.tolist())

    accuracy = binary_acc(y_pred_list, y_test)
    print("Accuracy:",accuracy.item())
```

16 Google model

[36]:
```
x = df_binary['avg_input_features_1']
y = df_binary['class']

# Split the dataset into 80% training dataset and 20% testing dataset

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20,␣
 ↪random_state=100)
```

[37]:
```
## train data
train_data = trainData(torch.FloatTensor(x_train.tolist()),
                       torch.FloatTensor(y_train))
```

```
## test data
test_data = testData(torch.FloatTensor(x_test.tolist()))
```

[38]:
```
train_loader = DataLoader(dataset=train_data, batch_size=BATCH_SIZE,␣
↪shuffle=True)
test_loader = DataLoader(dataset=test_data, batch_size=1)
```

[48]:
```
train_model_binary()
```

```
Epoch 001: | Loss: 0.42925 | Acc: 80.427
Epoch 002: | Loss: 0.41289 | Acc: 81.357
Epoch 003: | Loss: 0.40518 | Acc: 81.923
Epoch 004: | Loss: 0.40056 | Acc: 82.179
Epoch 005: | Loss: 0.39944 | Acc: 82.272
Epoch 006: | Loss: 0.39628 | Acc: 82.422
Epoch 007: | Loss: 0.39473 | Acc: 82.412
Epoch 008: | Loss: 0.39098 | Acc: 82.772
Epoch 009: | Loss: 0.38844 | Acc: 82.825
Epoch 010: | Loss: 0.38683 | Acc: 82.814
Epoch 011: | Loss: 0.38560 | Acc: 83.001
Epoch 012: | Loss: 0.38540 | Acc: 83.044
Epoch 013: | Loss: 0.38458 | Acc: 83.044
Epoch 014: | Loss: 0.38339 | Acc: 83.141
Epoch 015: | Loss: 0.38325 | Acc: 83.186
Epoch 016: | Loss: 0.38196 | Acc: 83.244
Epoch 017: | Loss: 0.38092 | Acc: 83.311
Epoch 018: | Loss: 0.37976 | Acc: 83.289
Epoch 019: | Loss: 0.37805 | Acc: 83.453
Epoch 020: | Loss: 0.37860 | Acc: 83.415
Epoch 021: | Loss: 0.37704 | Acc: 83.559
Epoch 022: | Loss: 0.37740 | Acc: 83.449
Epoch 023: | Loss: 0.37587 | Acc: 83.499
Epoch 024: | Loss: 0.37558 | Acc: 83.574
Epoch 025: | Loss: 0.37523 | Acc: 83.603
Epoch 026: | Loss: 0.37312 | Acc: 83.656
Epoch 027: | Loss: 0.37383 | Acc: 83.618
Epoch 028: | Loss: 0.37254 | Acc: 83.697
Epoch 029: | Loss: 0.37153 | Acc: 83.736
Epoch 030: | Loss: 0.37065 | Acc: 83.766
Epoch 031: | Loss: 0.37226 | Acc: 83.739
Epoch 032: | Loss: 0.37153 | Acc: 83.741
Epoch 033: | Loss: 0.37182 | Acc: 83.778
Epoch 034: | Loss: 0.37105 | Acc: 83.753
Epoch 035: | Loss: 0.36988 | Acc: 83.822
Epoch 036: | Loss: 0.36996 | Acc: 83.828
Epoch 037: | Loss: 0.36943 | Acc: 83.866
Epoch 038: | Loss: 0.36915 | Acc: 83.983
```

```
Epoch 039: | Loss: 0.36818 | Acc: 83.968
Epoch 040: | Loss: 0.36761 | Acc: 84.003
Epoch 041: | Loss: 0.36854 | Acc: 83.972
Epoch 042: | Loss: 0.36567 | Acc: 83.989
Epoch 043: | Loss: 0.37031 | Acc: 83.805
Epoch 044: | Loss: 0.36810 | Acc: 84.016
Epoch 045: | Loss: 0.36753 | Acc: 83.921
Epoch 046: | Loss: 0.36824 | Acc: 83.954
Epoch 047: | Loss: 0.36747 | Acc: 83.938
Epoch 048: | Loss: 0.36570 | Acc: 84.142
Epoch 049: | Loss: 0.36674 | Acc: 84.014
Epoch 050: | Loss: 0.36656 | Acc: 84.003
Epoch 051: | Loss: 0.36587 | Acc: 83.988
Epoch 052: | Loss: 0.36398 | Acc: 84.125
Epoch 053: | Loss: 0.36465 | Acc: 84.112
Epoch 054: | Loss: 0.36368 | Acc: 84.137
Epoch 055: | Loss: 0.36503 | Acc: 84.052
Epoch 056: | Loss: 0.36493 | Acc: 84.104
Epoch 057: | Loss: 0.36615 | Acc: 84.062
Epoch 058: | Loss: 0.36452 | Acc: 84.069
Epoch 059: | Loss: 0.36342 | Acc: 84.144
Epoch 060: | Loss: 0.36419 | Acc: 84.156
Epoch 061: | Loss: 0.36340 | Acc: 84.156
Epoch 062: | Loss: 0.36301 | Acc: 84.171
Epoch 063: | Loss: 0.36327 | Acc: 84.276
Epoch 064: | Loss: 0.36113 | Acc: 84.282
Epoch 065: | Loss: 0.36300 | Acc: 84.207
Epoch 066: | Loss: 0.36089 | Acc: 84.371
Epoch 067: | Loss: 0.36273 | Acc: 84.201
Epoch 068: | Loss: 0.36160 | Acc: 84.259
Epoch 069: | Loss: 0.36293 | Acc: 84.179
Epoch 070: | Loss: 0.36180 | Acc: 84.256
Epoch 071: | Loss: 0.36075 | Acc: 84.329
Epoch 072: | Loss: 0.35986 | Acc: 84.395
Epoch 073: | Loss: 0.36016 | Acc: 84.276
Epoch 074: | Loss: 0.36162 | Acc: 84.318
Epoch 075: | Loss: 0.36066 | Acc: 84.388
Epoch 076: | Loss: 0.36014 | Acc: 84.353
Epoch 077: | Loss: 0.36037 | Acc: 84.293
Epoch 078: | Loss: 0.35928 | Acc: 84.394
Epoch 079: | Loss: 0.36096 | Acc: 84.328
Epoch 080: | Loss: 0.36057 | Acc: 84.340
Epoch 081: | Loss: 0.35967 | Acc: 84.464
Epoch 082: | Loss: 0.35982 | Acc: 84.394
Epoch 083: | Loss: 0.36015 | Acc: 84.395
Epoch 084: | Loss: 0.36111 | Acc: 84.306
Epoch 085: | Loss: 0.35828 | Acc: 84.454
Epoch 086: | Loss: 0.35949 | Acc: 84.453
```

```
Epoch 087: | Loss: 0.35876 | Acc: 84.383
Epoch 088: | Loss: 0.36024 | Acc: 84.416
Epoch 089: | Loss: 0.35979 | Acc: 84.473
Epoch 090: | Loss: 0.35988 | Acc: 84.357
Epoch 091: | Loss: 0.35880 | Acc: 84.451
Epoch 092: | Loss: 0.35806 | Acc: 84.474
Epoch 093: | Loss: 0.35785 | Acc: 84.506
Epoch 094: | Loss: 0.35827 | Acc: 84.407
Epoch 095: | Loss: 0.35978 | Acc: 84.424
Epoch 096: | Loss: 0.35890 | Acc: 84.477
Epoch 097: | Loss: 0.35662 | Acc: 84.542
Epoch 098: | Loss: 0.35742 | Acc: 84.383
Epoch 099: | Loss: 0.35740 | Acc: 84.368
Epoch 100: | Loss: 0.35755 | Acc: 84.498
```

[40]:
```
test_model_binary(y_test)
```

```
Accuracy: 85.0
```

17 Our model

[41]:
```
x = df_binary['avg_input_features_2']
y = df_binary['class']

# Split the dataset into 80% training dataset and 20% testing dataset

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20,
 ↪random_state=100)
```

[42]:
```
## train data
train_data = trainData(torch.FloatTensor(x_train.tolist()),
                       torch.FloatTensor(y_train))
## test data
test_data = testData(torch.FloatTensor(x_test.tolist()))
```

[43]:
```
train_loader = DataLoader(dataset=train_data, batch_size=BATCH_SIZE,
 ↪shuffle=True)
test_loader = DataLoader(dataset=test_data, batch_size=1)
```

[52]:
```
train_model_binary()
```

```
Epoch 001: | Loss: 0.39232 | Acc: 82.813
Epoch 002: | Loss: 0.36632 | Acc: 84.182
Epoch 003: | Loss: 0.35946 | Acc: 84.547
Epoch 004: | Loss: 0.35653 | Acc: 84.755
Epoch 005: | Loss: 0.35471 | Acc: 84.800
Epoch 006: | Loss: 0.35266 | Acc: 84.886
```

```
Epoch 007: | Loss: 0.34956 | Acc: 85.044
Epoch 008: | Loss: 0.34865 | Acc: 85.131
Epoch 009: | Loss: 0.34677 | Acc: 85.244
Epoch 010: | Loss: 0.34363 | Acc: 85.446
Epoch 011: | Loss: 0.34300 | Acc: 85.409
Epoch 012: | Loss: 0.34189 | Acc: 85.467
Epoch 013: | Loss: 0.34211 | Acc: 85.509
Epoch 014: | Loss: 0.34012 | Acc: 85.634
Epoch 015: | Loss: 0.33893 | Acc: 85.728
Epoch 016: | Loss: 0.33678 | Acc: 85.840
Epoch 017: | Loss: 0.33812 | Acc: 85.809
Epoch 018: | Loss: 0.33775 | Acc: 85.824
Epoch 019: | Loss: 0.33789 | Acc: 85.814
Epoch 020: | Loss: 0.33505 | Acc: 85.876
Epoch 021: | Loss: 0.33324 | Acc: 85.933
Epoch 022: | Loss: 0.33596 | Acc: 85.903
Epoch 023: | Loss: 0.33278 | Acc: 86.058
Epoch 024: | Loss: 0.33759 | Acc: 85.791
Epoch 025: | Loss: 0.33218 | Acc: 86.116
Epoch 026: | Loss: 0.33294 | Acc: 86.046
Epoch 027: | Loss: 0.33325 | Acc: 86.049
Epoch 028: | Loss: 0.33159 | Acc: 86.079
Epoch 029: | Loss: 0.33284 | Acc: 86.114
Epoch 030: | Loss: 0.33073 | Acc: 86.104
Epoch 031: | Loss: 0.33017 | Acc: 86.072
Epoch 032: | Loss: 0.32992 | Acc: 86.121
Epoch 033: | Loss: 0.33085 | Acc: 86.138
Epoch 034: | Loss: 0.32784 | Acc: 86.251
Epoch 035: | Loss: 0.32973 | Acc: 86.232
Epoch 036: | Loss: 0.32661 | Acc: 86.349
Epoch 037: | Loss: 0.32801 | Acc: 86.297
Epoch 038: | Loss: 0.32906 | Acc: 86.326
Epoch 039: | Loss: 0.32786 | Acc: 86.201
Epoch 040: | Loss: 0.32752 | Acc: 86.188
Epoch 041: | Loss: 0.32987 | Acc: 86.116
Epoch 042: | Loss: 0.32918 | Acc: 86.251
Epoch 043: | Loss: 0.32669 | Acc: 86.330
Epoch 044: | Loss: 0.32733 | Acc: 86.331
Epoch 045: | Loss: 0.32391 | Acc: 86.441
Epoch 046: | Loss: 0.32584 | Acc: 86.354
Epoch 047: | Loss: 0.32441 | Acc: 86.426
Epoch 048: | Loss: 0.32511 | Acc: 86.450
Epoch 049: | Loss: 0.32569 | Acc: 86.408
Epoch 050: | Loss: 0.32575 | Acc: 86.345
Epoch 051: | Loss: 0.32466 | Acc: 86.412
Epoch 052: | Loss: 0.32621 | Acc: 86.321
Epoch 053: | Loss: 0.32728 | Acc: 86.236
Epoch 054: | Loss: 0.32691 | Acc: 86.368
```

```
Epoch 055: | Loss: 0.32392 | Acc: 86.382
Epoch 056: | Loss: 0.32668 | Acc: 86.389
Epoch 057: | Loss: 0.32294 | Acc: 86.537
Epoch 058: | Loss: 0.32282 | Acc: 86.406
Epoch 059: | Loss: 0.32060 | Acc: 86.579
Epoch 060: | Loss: 0.32326 | Acc: 86.479
Epoch 061: | Loss: 0.32210 | Acc: 86.441
Epoch 062: | Loss: 0.32237 | Acc: 86.454
Epoch 063: | Loss: 0.32480 | Acc: 86.324
Epoch 064: | Loss: 0.32176 | Acc: 86.525
Epoch 065: | Loss: 0.32265 | Acc: 86.436
Epoch 066: | Loss: 0.32066 | Acc: 86.576
Epoch 067: | Loss: 0.32243 | Acc: 86.452
Epoch 068: | Loss: 0.32156 | Acc: 86.548
Epoch 069: | Loss: 0.32088 | Acc: 86.634
Epoch 070: | Loss: 0.31985 | Acc: 86.634
Epoch 071: | Loss: 0.31822 | Acc: 86.700
Epoch 072: | Loss: 0.31938 | Acc: 86.631
Epoch 073: | Loss: 0.32076 | Acc: 86.593
Epoch 074: | Loss: 0.31971 | Acc: 86.582
Epoch 075: | Loss: 0.32273 | Acc: 86.504
Epoch 076: | Loss: 0.32132 | Acc: 86.588
Epoch 077: | Loss: 0.32099 | Acc: 86.567
Epoch 078: | Loss: 0.31815 | Acc: 86.623
Epoch 079: | Loss: 0.31833 | Acc: 86.688
Epoch 080: | Loss: 0.32235 | Acc: 86.564
Epoch 081: | Loss: 0.32186 | Acc: 86.558
Epoch 082: | Loss: 0.32028 | Acc: 86.562
Epoch 083: | Loss: 0.31744 | Acc: 86.709
Epoch 084: | Loss: 0.31719 | Acc: 86.753
Epoch 085: | Loss: 0.31888 | Acc: 86.501
Epoch 086: | Loss: 0.31788 | Acc: 86.670
Epoch 087: | Loss: 0.31918 | Acc: 86.711
Epoch 088: | Loss: 0.31895 | Acc: 86.687
Epoch 089: | Loss: 0.31959 | Acc: 86.677
Epoch 090: | Loss: 0.31753 | Acc: 86.796
Epoch 091: | Loss: 0.31670 | Acc: 86.814
Epoch 092: | Loss: 0.31715 | Acc: 86.686
Epoch 093: | Loss: 0.31750 | Acc: 86.789
Epoch 094: | Loss: 0.31823 | Acc: 86.680
Epoch 095: | Loss: 0.31642 | Acc: 86.716
Epoch 096: | Loss: 0.31713 | Acc: 86.763
Epoch 097: | Loss: 0.31738 | Acc: 86.708
Epoch 098: | Loss: 0.31473 | Acc: 86.873
Epoch 099: | Loss: 0.31520 | Acc: 86.819
Epoch 100: | Loss: 0.31742 | Acc: 86.736
```

[45]:
```
test_model_binary(y_test)
```

```
Accuracy: 87.0
```

18 Ternary

[46]:
```
# ternary classification dataframe

df_ternary = df_org_3.copy(deep=True)
df_ternary
```

[46]:
```
                                              review_body  class  \
0       send back unhappy wth quality guage s sheet us...      1
1       bought bottle week lid crack right rim boght p...      1
2       good overall instruction could use improvement...      2
3                      beautiful color unexpectedly large      0
4       puzzle review look fine bought mug use clean t...      2
...                                                   ...    ...
249995  love little skinny spatula use stovetop cookin...      0
249996  cheap leaky creaky sure pump handle break soon...      1
249997  good price awesome product buy constantly rest...      0
249998          machine little loud make great cup coffee      0
249999  portable go anywhere wine cup would probably g...      1

                                     avg_input_features_1  \
0       [-0.028214889, 0.054062814, 0.022171944, 0.065...
1       [0.009401504, 0.04494009, -0.01879862, 0.04671...
2       [-0.025609551, 0.035386518, -0.03870993, 0.123...
3       [0.051719666, 0.07980347, -0.05140686, 0.07983...
4       [0.0021718915, 0.036595784, -0.015984524, 0.04...
...                                                   ...
249995  [0.032534514, 0.028369326, 0.016048547, 0.0922...
249996  [0.019851685, 0.074625395, -0.054214478, 0.047...
249997  [0.027029855, -0.028424945, -0.025542123, 0.14...
249998  [0.0011160715, -0.0034005302, -0.033098493, 0...
249999  [-0.020776367, 0.000773112, -0.021533202, 0.12...

                                     avg_input_features_2
0       [0.016383082, -0.11017842, 0.07979045, -0.1103...
1       [0.0027814035, -0.1517246, 0.039110575, -0.084...
2       [-0.019544542, -0.09348309, 0.091074795, -0.08...
3       [-0.0066354196, -0.07669535, 0.14650348, 0.051...
4       [0.049097426, -0.15116577, 0.034840178, -0.083...
...                                                   ...
249995  [0.07504659, -0.11023586, 0.102279335, -0.0310...
249996  [-0.007991508, -0.23522964, 0.17086153, -0.027...
```

```
249997  [-0.01619439, -0.05732434, 0.008259937, -0.110...
249998  [0.0053175413, -0.09154149, 0.12706958, -0.087...
249999  [0.016355243, -0.13739465, -0.052232314, -0.01...

[250000 rows x 4 columns]
```

[83]:
```python
# set parameters

input_size = 300
hidden_1_size = 50
hidden_2_size = 10
output_size = 3
```

[58]:
```python
# model for ternary classification

class ternary_classification(nn.Module):
    def __init__(self):
        super(ternary_classification, self).__init__()
        # Number of input features is 300.
        self.layer_1 = nn.Linear(input_size, hidden_1_size)
        self.layer_2 = nn.Linear(hidden_1_size, hidden_2_size)
        self.layer_out = nn.Linear(hidden_2_size, output_size)

        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(p=0.1)
        self.batchnorm1 = nn.BatchNorm1d(hidden_1_size)
        self.batchnorm2 = nn.BatchNorm1d(hidden_2_size)

    def forward(self, x):
        x = self.relu(self.layer_1(x))
        x = self.batchnorm1(x)
        x = self.relu(self.layer_2(x))
        x = self.batchnorm2(x)
        x = self.dropout(x)
        x = self.layer_out(x)

        return x
```

[84]:
```python
# print model

model = ternary_classification()
print(model)
```

```
ternary_classification(
  (layer_1): Linear(in_features=300, out_features=50, bias=True)
  (layer_2): Linear(in_features=50, out_features=10, bias=True)
  (layer_out): Linear(in_features=10, out_features=3, bias=True)
```

```
  (relu): ReLU()
  (dropout): Dropout(p=0.1, inplace=False)
  (batchnorm1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (batchnorm2): BatchNorm1d(10, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
```

[60]:
```
# define loss function and optimizer

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)
```

[61]:
```
# function to find the accuracy of the ternary model

def ternary_acc(y_pred, y_test):
    y_pred_softmax = torch.log_softmax(y_pred, dim = 1)
    y_pred_tags = torch.argmax(y_pred_softmax, dim = 1)

    correct_pred = (y_pred_tags == y_test).float()
    acc = correct_pred.sum() / len(correct_pred)

    acc = torch.round(acc * 100)

    return acc
```

[62]:
```
# function to train the ternary model and print results(loss & accuracy per␣
 ↪epoch)

def train_model_ternary():
    model.train()
    for e in range(1, EPOCHS+1):
        epoch_loss = 0
        epoch_acc = 0

        for x_batch, y_batch in train_loader:
            x_batch, y_batch = x_batch, y_batch
            optimizer.zero_grad()

            y_pred = model(x_batch)

            loss = criterion(y_pred, y_batch.type(torch.LongTensor))
            acc = ternary_acc(y_pred, y_batch.type(torch.LongTensor))

            loss.backward()
            optimizer.step()
```

```
            epoch_loss += loss.item()
            epoch_acc += acc.item()


        print(f'Epoch {e+0:03}: | Loss: {epoch_loss/len(train_loader):.5f} | Acc:
 ↪ {epoch_acc/len(train_loader):.3f}')
```

[63]:
```
def test_model_ternary(y_test):
    model.eval()

    y_pred_list = []

    with torch.no_grad():
        for x_batch in test_loader:
            y_test_pred = model(x_batch)
            y_pred_list.extend(y_test_pred.tolist())

    y_pred_list = torch.FloatTensor(y_pred_list)
    y_test = torch.FloatTensor(y_test.tolist())

    accuracy = ternary_acc(y_pred_list, y_test)
    print("Accuracy:",accuracy.item())
```

19 Google model

[54]:
```
x = df_ternary['avg_input_features_1']
y = df_ternary['class']

# Split the dataset into 80% training dataset and 20% testing dataset

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20,␣
 ↪random_state=100)
```

[55]:
```
## train data
train_data = trainData(torch.FloatTensor(x_train.tolist()),
                       torch.FloatTensor(y_train))
## test data
test_data = testData(torch.FloatTensor(x_test.tolist()))
```

[56]:
```
train_loader = DataLoader(dataset=train_data, batch_size=BATCH_SIZE,␣
 ↪shuffle=True)
test_loader = DataLoader(dataset=test_data, batch_size=1)
```

[63]:
```
train_model_ternary()
```

```
Epoch 001: | Loss: 0.81074 | Acc: 64.684
```

```
Epoch 002: | Loss: 0.79038 | Acc: 65.774
Epoch 003: | Loss: 0.78350 | Acc: 66.063
Epoch 004: | Loss: 0.77975 | Acc: 66.197
Epoch 005: | Loss: 0.77602 | Acc: 66.421
Epoch 006: | Loss: 0.77350 | Acc: 66.501
Epoch 007: | Loss: 0.77049 | Acc: 66.802
Epoch 008: | Loss: 0.76746 | Acc: 66.855
Epoch 009: | Loss: 0.76838 | Acc: 66.906
Epoch 010: | Loss: 0.76734 | Acc: 66.832
Epoch 011: | Loss: 0.76561 | Acc: 66.938
Epoch 012: | Loss: 0.76345 | Acc: 67.022
Epoch 013: | Loss: 0.76225 | Acc: 67.181
Epoch 014: | Loss: 0.76194 | Acc: 67.082
Epoch 015: | Loss: 0.76024 | Acc: 67.168
Epoch 016: | Loss: 0.76047 | Acc: 67.124
Epoch 017: | Loss: 0.75881 | Acc: 67.260
Epoch 018: | Loss: 0.75817 | Acc: 67.221
Epoch 019: | Loss: 0.75690 | Acc: 67.308
Epoch 020: | Loss: 0.75645 | Acc: 67.327
Epoch 021: | Loss: 0.75610 | Acc: 67.394
Epoch 022: | Loss: 0.75603 | Acc: 67.302
Epoch 023: | Loss: 0.75522 | Acc: 67.371
Epoch 024: | Loss: 0.75484 | Acc: 67.377
Epoch 025: | Loss: 0.75271 | Acc: 67.562
Epoch 026: | Loss: 0.75396 | Acc: 67.415
Epoch 027: | Loss: 0.75273 | Acc: 67.512
Epoch 028: | Loss: 0.75243 | Acc: 67.593
Epoch 029: | Loss: 0.75331 | Acc: 67.391
Epoch 030: | Loss: 0.75280 | Acc: 67.484
Epoch 031: | Loss: 0.75049 | Acc: 67.660
Epoch 032: | Loss: 0.75193 | Acc: 67.531
Epoch 033: | Loss: 0.75154 | Acc: 67.484
Epoch 034: | Loss: 0.75181 | Acc: 67.530
Epoch 035: | Loss: 0.75151 | Acc: 67.629
Epoch 036: | Loss: 0.75154 | Acc: 67.565
Epoch 037: | Loss: 0.75164 | Acc: 67.579
Epoch 038: | Loss: 0.75018 | Acc: 67.640
Epoch 039: | Loss: 0.74989 | Acc: 67.638
Epoch 040: | Loss: 0.75043 | Acc: 67.663
Epoch 041: | Loss: 0.74842 | Acc: 67.745
Epoch 042: | Loss: 0.74850 | Acc: 67.700
Epoch 043: | Loss: 0.74778 | Acc: 67.737
Epoch 044: | Loss: 0.74926 | Acc: 67.620
Epoch 045: | Loss: 0.74831 | Acc: 67.674
Epoch 046: | Loss: 0.74874 | Acc: 67.719
Epoch 047: | Loss: 0.74880 | Acc: 67.709
Epoch 048: | Loss: 0.75052 | Acc: 67.526
Epoch 049: | Loss: 0.74793 | Acc: 67.635
```

```
Epoch 050: | Loss: 0.74792 | Acc: 67.728
Epoch 051: | Loss: 0.74723 | Acc: 67.850
Epoch 052: | Loss: 0.74648 | Acc: 67.820
Epoch 053: | Loss: 0.74730 | Acc: 67.649
Epoch 054: | Loss: 0.74753 | Acc: 67.760
Epoch 055: | Loss: 0.74507 | Acc: 67.840
Epoch 056: | Loss: 0.74550 | Acc: 67.885
Epoch 057: | Loss: 0.74431 | Acc: 67.954
Epoch 058: | Loss: 0.74508 | Acc: 67.832
Epoch 059: | Loss: 0.74524 | Acc: 67.882
Epoch 060: | Loss: 0.74335 | Acc: 67.927
Epoch 061: | Loss: 0.74466 | Acc: 67.864
Epoch 062: | Loss: 0.74495 | Acc: 67.856
Epoch 063: | Loss: 0.74414 | Acc: 67.969
Epoch 064: | Loss: 0.74516 | Acc: 67.915
Epoch 065: | Loss: 0.74569 | Acc: 67.849
Epoch 066: | Loss: 0.74464 | Acc: 68.021
Epoch 067: | Loss: 0.74411 | Acc: 67.951
Epoch 068: | Loss: 0.74281 | Acc: 67.945
Epoch 069: | Loss: 0.74328 | Acc: 68.031
Epoch 070: | Loss: 0.74251 | Acc: 68.001
Epoch 071: | Loss: 0.74331 | Acc: 67.944
Epoch 072: | Loss: 0.74264 | Acc: 68.025
Epoch 073: | Loss: 0.74215 | Acc: 67.987
Epoch 074: | Loss: 0.74163 | Acc: 68.017
Epoch 075: | Loss: 0.74275 | Acc: 67.941
Epoch 076: | Loss: 0.74107 | Acc: 68.001
Epoch 077: | Loss: 0.74220 | Acc: 68.114
Epoch 078: | Loss: 0.74089 | Acc: 68.112
Epoch 079: | Loss: 0.74065 | Acc: 68.055
Epoch 080: | Loss: 0.74118 | Acc: 68.038
Epoch 081: | Loss: 0.74213 | Acc: 68.025
Epoch 082: | Loss: 0.74016 | Acc: 68.120
Epoch 083: | Loss: 0.74056 | Acc: 68.160
Epoch 084: | Loss: 0.74101 | Acc: 68.103
Epoch 085: | Loss: 0.74093 | Acc: 68.191
Epoch 086: | Loss: 0.74030 | Acc: 68.171
Epoch 087: | Loss: 0.74051 | Acc: 68.052
Epoch 088: | Loss: 0.74108 | Acc: 68.122
Epoch 089: | Loss: 0.74125 | Acc: 68.064
Epoch 090: | Loss: 0.74013 | Acc: 68.115
Epoch 091: | Loss: 0.73998 | Acc: 68.145
Epoch 092: | Loss: 0.74028 | Acc: 68.075
Epoch 093: | Loss: 0.74015 | Acc: 68.190
Epoch 094: | Loss: 0.74054 | Acc: 68.091
Epoch 095: | Loss: 0.73964 | Acc: 68.181
Epoch 096: | Loss: 0.74093 | Acc: 68.058
Epoch 097: | Loss: 0.74063 | Acc: 68.108
```

```
Epoch 098: | Loss: 0.74083 | Acc: 68.090
Epoch 099: | Loss: 0.74134 | Acc: 68.052
Epoch 100: | Loss: 0.74173 | Acc: 67.995
```

[73]:
```
test_model_ternary(y_test)
```

```
Accuracy: 68.0
```

20 Our model

[74]:
```
x = df_ternary['avg_input_features_2']
y = df_ternary['class']

# Split the dataset into 80% training dataset and 20% testing dataset

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20,␣
 ↪random_state=100)
```

[75]:
```
## train data
train_data = trainData(torch.FloatTensor(x_train.tolist()),
                       torch.FloatTensor(y_train))
## test data
test_data = testData(torch.FloatTensor(x_test.tolist()))
```

[76]:
```
train_loader = DataLoader(dataset=train_data, batch_size=BATCH_SIZE,␣
 ↪shuffle=True)
test_loader = DataLoader(dataset=test_data, batch_size=1)
```

[67]:
```
train_model_ternary()
```

```
Epoch 001: | Loss: 0.77657 | Acc: 66.731
Epoch 002: | Loss: 0.74225 | Acc: 68.403
Epoch 003: | Loss: 0.73529 | Acc: 68.714
Epoch 004: | Loss: 0.73158 | Acc: 68.862
Epoch 005: | Loss: 0.72965 | Acc: 69.052
Epoch 006: | Loss: 0.72626 | Acc: 69.173
Epoch 007: | Loss: 0.72531 | Acc: 69.286
Epoch 008: | Loss: 0.72245 | Acc: 69.313
Epoch 009: | Loss: 0.72162 | Acc: 69.436
Epoch 010: | Loss: 0.72103 | Acc: 69.347
Epoch 011: | Loss: 0.71931 | Acc: 69.397
Epoch 012: | Loss: 0.71833 | Acc: 69.371
Epoch 013: | Loss: 0.71758 | Acc: 69.464
Epoch 014: | Loss: 0.71764 | Acc: 69.540
Epoch 015: | Loss: 0.71575 | Acc: 69.572
Epoch 016: | Loss: 0.71370 | Acc: 69.653
Epoch 017: | Loss: 0.71389 | Acc: 69.635
```

```
Epoch 018: | Loss: 0.71227 | Acc: 69.692
Epoch 019: | Loss: 0.71149 | Acc: 69.790
Epoch 020: | Loss: 0.71154 | Acc: 69.733
Epoch 021: | Loss: 0.71211 | Acc: 69.715
Epoch 022: | Loss: 0.70957 | Acc: 69.901
Epoch 023: | Loss: 0.70945 | Acc: 69.834
Epoch 024: | Loss: 0.70905 | Acc: 69.852
Epoch 025: | Loss: 0.70729 | Acc: 69.925
Epoch 026: | Loss: 0.70827 | Acc: 69.939
Epoch 027: | Loss: 0.70631 | Acc: 70.046
Epoch 028: | Loss: 0.70669 | Acc: 69.980
Epoch 029: | Loss: 0.70660 | Acc: 69.948
Epoch 030: | Loss: 0.70578 | Acc: 70.033
Epoch 031: | Loss: 0.70609 | Acc: 69.958
Epoch 032: | Loss: 0.70356 | Acc: 70.089
Epoch 033: | Loss: 0.70564 | Acc: 69.979
Epoch 034: | Loss: 0.70582 | Acc: 69.979
Epoch 035: | Loss: 0.70407 | Acc: 70.093
Epoch 036: | Loss: 0.70462 | Acc: 70.029
Epoch 037: | Loss: 0.70317 | Acc: 70.019
Epoch 038: | Loss: 0.70324 | Acc: 70.121
Epoch 039: | Loss: 0.70374 | Acc: 70.100
Epoch 040: | Loss: 0.70331 | Acc: 70.113
Epoch 041: | Loss: 0.70262 | Acc: 70.254
Epoch 042: | Loss: 0.70243 | Acc: 70.230
Epoch 043: | Loss: 0.70184 | Acc: 70.207
Epoch 044: | Loss: 0.70246 | Acc: 70.137
Epoch 045: | Loss: 0.70267 | Acc: 70.207
Epoch 046: | Loss: 0.70093 | Acc: 70.145
Epoch 047: | Loss: 0.70168 | Acc: 70.052
Epoch 048: | Loss: 0.70126 | Acc: 70.165
Epoch 049: | Loss: 0.70104 | Acc: 70.173
Epoch 050: | Loss: 0.70155 | Acc: 70.088
Epoch 051: | Loss: 0.70091 | Acc: 70.188
Epoch 052: | Loss: 0.70002 | Acc: 70.186
Epoch 053: | Loss: 0.69958 | Acc: 70.287
Epoch 054: | Loss: 0.69960 | Acc: 70.278
Epoch 055: | Loss: 0.69968 | Acc: 70.224
Epoch 056: | Loss: 0.69903 | Acc: 70.225
Epoch 057: | Loss: 0.70024 | Acc: 70.218
Epoch 058: | Loss: 0.69947 | Acc: 70.207
Epoch 059: | Loss: 0.70004 | Acc: 70.185
Epoch 060: | Loss: 0.69910 | Acc: 70.335
Epoch 061: | Loss: 0.69808 | Acc: 70.245
Epoch 062: | Loss: 0.69858 | Acc: 70.308
Epoch 063: | Loss: 0.69851 | Acc: 70.347
Epoch 064: | Loss: 0.69925 | Acc: 70.296
Epoch 065: | Loss: 0.69895 | Acc: 70.294
```

```
Epoch 066: | Loss: 0.69730 | Acc: 70.263
Epoch 067: | Loss: 0.69818 | Acc: 70.249
Epoch 068: | Loss: 0.69684 | Acc: 70.317
Epoch 069: | Loss: 0.69691 | Acc: 70.312
Epoch 070: | Loss: 0.69692 | Acc: 70.266
Epoch 071: | Loss: 0.69770 | Acc: 70.221
Epoch 072: | Loss: 0.69637 | Acc: 70.400
Epoch 073: | Loss: 0.69646 | Acc: 70.243
Epoch 074: | Loss: 0.69551 | Acc: 70.474
Epoch 075: | Loss: 0.69519 | Acc: 70.481
Epoch 076: | Loss: 0.69502 | Acc: 70.486
Epoch 077: | Loss: 0.69615 | Acc: 70.392
Epoch 078: | Loss: 0.69638 | Acc: 70.362
Epoch 079: | Loss: 0.69513 | Acc: 70.421
Epoch 080: | Loss: 0.69545 | Acc: 70.377
Epoch 081: | Loss: 0.69569 | Acc: 70.394
Epoch 082: | Loss: 0.69482 | Acc: 70.425
Epoch 083: | Loss: 0.69612 | Acc: 70.404
Epoch 084: | Loss: 0.69444 | Acc: 70.455
Epoch 085: | Loss: 0.69469 | Acc: 70.415
Epoch 086: | Loss: 0.69517 | Acc: 70.375
Epoch 087: | Loss: 0.69511 | Acc: 70.459
Epoch 088: | Loss: 0.69475 | Acc: 70.472
Epoch 089: | Loss: 0.69380 | Acc: 70.490
Epoch 090: | Loss: 0.69439 | Acc: 70.445
Epoch 091: | Loss: 0.69405 | Acc: 70.450
Epoch 092: | Loss: 0.69421 | Acc: 70.433
Epoch 093: | Loss: 0.69460 | Acc: 70.404
Epoch 094: | Loss: 0.69319 | Acc: 70.463
Epoch 095: | Loss: 0.69391 | Acc: 70.307
Epoch 096: | Loss: 0.69338 | Acc: 70.493
Epoch 097: | Loss: 0.69322 | Acc: 70.549
Epoch 098: | Loss: 0.69366 | Acc: 70.470
Epoch 099: | Loss: 0.69368 | Acc: 70.468
Epoch 100: | Loss: 0.69360 | Acc: 70.412
```

[78]:
```
test_model_ternary(y_test)
```

```
Accuracy: 71.0
```

21 Comments about this question

[34]:
```
d = {'Model': ['FNN', 'FNN', 'FNN', 'FNN'],
     'Word2Vec Model': ['Google News', 'Amazon Reviews(Our)', 'Google News',␣
 ↪'Amazon Reviews(Our)'],
     'Classification Type': ['Binary', 'Binary', 'Ternary', 'Ternary',],
```

```
    'Input Features Type': ['Average' , 'Average', 'Average', 'Average'],
    'Accuracy': ['0.85', '0.87', '0.68', '0.71']}

df_results_part_4_a = pd.DataFrame(data=d)
df_results_part_4_a
```

[34]:

	Model	Word2Vec Model	Classification Type	Input Features Type	Accuracy
0	FNN	Google News	Binary	Average	0.85
1	FNN	Amazon Reviews(Our)	Binary	Average	0.87
2	FNN	Google News	Ternary	Average	0.68
3	FNN	Amazon Reviews(Our)	Ternary	Average	0.71

22 (b)

[26]:
```
# function to pad a list with a specific number of zeroes

def pad_or_truncate(some_list, target_len):
    return some_list[:target_len] + [0]*(target_len - len(some_list))
```

[27]:
```
# function to concatenate vectors of first ten words as your input feature

def concatenate_vectors(review,model_used):

    sentence_words = review.split(" ")

    sentence_vectors = []

    for i,word in enumerate(sentence_words):
        if i < 10:
            try:
                sentence_vectors.append(model_used[word])
            except:
                continue

    flattened_sentence_vector = np.array(sentence_vectors).flatten()

    if len(sentence_vectors)!=0:
        if len(flattened_sentence_vector) != 3000:
            flattened_sentence_vector =␣
↪pad_or_truncate(list(flattened_sentence_vector),3000)

        return flattened_sentence_vector

    else:
        return np.zeros(3000,)
```

[28]:
```
# find input feature for google model

df_org_3['concat_input_features_1'] = df_org_3['review_body'].apply(lambda x:
 ↪concatenate_vectors(x,wv))
df_org_3
```

[28]:
```
                                             review_body  class  \
0       assume four charger bought item pretty bought ...      1
1       son like cook he especially good grill burger ...      0
2                   ship fast good price way huger expect      0
3               container great lid thin break easily one use      2
4       item receive broken return ask replacement shi...      1
...                                                  ...    ...
249995                      lock come easily hard clean top      2
249996  bum carafe slightly wide bit short metal struc...      1
249997  I kettle one month leak water leak seal bottom...      1
249998  idea color balloon entice order package child ...      1
249999  product fail almost immediately digit garble s...      1

                                    avg_input_features_1  \
0       [0.04277208, -0.03597005, -0.062435575, 0.1046...
1       [-0.004893621, 0.029286703, -0.01199023, 0.162...
2       [0.1432408, 0.08569336, -0.048673358, 0.078264...
3       [0.056274414, 0.10064697, -0.0005340576, 0.056...
4       [0.043584187, -0.013412476, -0.116475426, 0.06...
...                                                  ...
249995  [0.03120931, 0.07987467, 0.03741455, 0.0357869...
249996  [-0.001551011, 0.026309744, -0.06418026, 0.125...
249997  [0.0027923584, 0.092679344, -0.03684489, 0.028...
249998  [0.047094908, 0.011726828, 0.00012925093, 0.09...
249999  [0.085134655, -0.011324369, 0.06199294, 0.0255...

                                    avg_input_features_2  \
0       [0.017703589, -0.11186184, -0.0030522645, -0.0...
1       [0.120273024, -0.14361034, 0.046780374, -0.138...
2       [-0.049596105, -0.018341891, 0.13302507, -0.17...
3       [0.030435072, -0.15327847, 0.11309578, -0.1425...
4       [0.08915458, -0.22801971, -0.028520422, -0.263...
...                                                  ...
249995  [0.015699785, -0.12990652, 0.21889718, -0.1027...
249996  [0.015504825, -0.031771064, 0.1092756, -0.0557...
249997  [0.020719932, -0.090553395, 0.13070571, -0.027...
249998  [0.066825956, -0.17564225, 0.05628306, -0.0763...
249999  [0.0051919767, -0.1441225, 0.13658296, -0.1857...

                                 concat_input_features_1
0       [0.06640625, -0.103027344, -0.08251953, 0.1079...
```

```
1       [0.107910156, -0.030029297, 0.033203125, -0.16...
2       [0.27929688, 0.29101562, -0.21386719, -0.14648...
3       [0.048095703, 0.31640625, 0.17773438, -0.06982...
4       [0.024291992, 0.010803223, -0.107421875, 0.302...
...                                                  ...
249995  [0.017944336, 0.19335938, -0.06298828, 0.02429...
249996  [0.10546875, -0.20117188, -0.13964844, 0.32226...
249997  [0.07910156, -0.0050354004, 0.111816406, 0.212...
249998  [0.067871094, 0.011657715, 0.033691406, 0.2207...
249999  [-0.061523438, 0.095214844, 0.13378906, 0.0649...

[250000 rows x 5 columns]
```

[29]:

```
# find input feature for our model

df_org_3['concat_input_features_2'] = df_org_3['review_body'].apply(lambda x:
 ↪concatenate_vectors(x,final_model))
df_org_3
```

```
<ipython-input-27-7c3efbd4463c>:12: DeprecationWarning: Call to deprecated
`__getitem__` (Method will be removed in 4.0.0, use self.wv.__getitem__()
instead).
  sentence_vectors.append(model_used[word])
```

[29]:

```
                                                review_body  class  \
0       assume four charger bought item pretty bought ...      1
1       son like cook he especially good grill burger ...      0
2                     ship fast good price way huger expect      0
3           container great lid thin break easily one use      2
4       item receive broken return ask replacement shi...      1
...                                                   ...    ...
249995                       lock come easily hard clean top      2
249996  bum carafe slightly wide bit short metal struc...      1
249997  I kettle one month leak water leak seal bottom...      1
249998  idea color balloon entice order package child ...      1
249999  product fail almost immediately digit garble s...      1

                                    avg_input_features_1  \
0       [0.04277208, -0.03597005, -0.062435575, 0.1046...
1       [-0.004893621, 0.029286703, -0.01199023, 0.162...
2       [0.1432408, 0.08569336, -0.048673358, 0.078264...
3       [0.056274414, 0.10064697, -0.0005340576, 0.056...
4       [0.043584187, -0.013412476, -0.116475426, 0.06...
...                                                  ...
249995  [0.03120931, 0.07987467, 0.03741455, 0.0357869...
249996  [-0.001551011, 0.026309744, -0.06418026, 0.125...
249997  [0.0027923584, 0.092679344, -0.03684489, 0.028...
```

```
249998  [0.047094908, 0.011726828, 0.00012925093, 0.09...
249999  [0.085134655, -0.011324369, 0.06199294, 0.0255...

                                        avg_input_features_2  \
0        [0.017703589, -0.11186184, -0.0030522645, -0.0...
1        [0.120273024, -0.14361034, 0.046780374, -0.138...
2        [-0.049596105, -0.018341891, 0.13302507, -0.17...
3        [0.030435072, -0.15327847, 0.11309578, -0.1425...
4        [0.08915458, -0.22801971, -0.028520422, -0.263...
...                                                    ...
249995   [0.015699785, -0.12990652, 0.21889718, -0.1027...
249996   [0.015504825, -0.031771064, 0.1092756, -0.0557...
249997   [0.020719932, -0.090553395, 0.13070571, -0.027...
249998   [0.066825956, -0.17564225, 0.05628306, -0.0763...
249999   [0.0051919767, -0.1441225, 0.13658296, -0.1857...

                                     concat_input_features_1  \
0        [0.06640625, -0.103027344, -0.08251953, 0.1079...
1        [0.107910156, -0.030029297, 0.033203125, -0.16...
2        [0.27929688, 0.29101562, -0.21386719, -0.14648...
3        [0.048095703, 0.31640625, 0.17773438, -0.06982...
4        [0.024291992, 0.010803223, -0.107421875, 0.302...
...                                                    ...
249995   [0.017944336, 0.19335938, -0.06298828, 0.02429...
249996   [0.10546875, -0.20117188, -0.13964844, 0.32226...
249997   [0.07910156, -0.0050354004, 0.111816406, 0.212...
249998   [0.067871094, 0.011657715, 0.033691406, 0.2207...
249999   [-0.061523438, 0.095214844, 0.13378906, 0.0649...

                                     concat_input_features_2
0        [0.18149155, -0.23886244, -0.0827184, 0.060127...
1        [0.39106262, -0.43970776, -0.014117015, 0.1198...
2        [-0.07464662, -0.21261097, -0.26036084, -0.465...
3        [-0.044359308, -0.092595585, 0.07619203, -0.14...
4        [0.102800496, -0.12086469, -0.14640297, 0.0537...
...                                                    ...
249995   [0.24208477, -0.24096622, 0.30787, -0.2916415,...
249996   [0.14765103, -0.15398727, 0.014575721, -0.1541...
249997   [0.2060052, -0.18501587, -0.0031185225, -0.029...
249998   [0.07186723, -0.11819719, -0.024285497, -0.130...
249999   [0.17383887, 0.03144031, -0.15070951, -0.04374...

[250000 rows x 6 columns]
```

23 Binary

[30]:
```
# binary classification dataframe

df_binary = df_org_3[((df_org_3['class'] == 0) | (df_org_3['class'] == 1))]
df_binary
```

[30]:
```
                                               review_body  class  \
0       assume four charger bought item pretty bought ...      1
1       son like cook he especially good grill burger ...      0
2                    ship fast good price way huger expect      0
4       item receive broken return ask replacement shi...      1
5       experience issue one cup fill make sure filter...      0
...                                                    ...    ...
249993  toaster oven fine especially since paid amazon...      1
249996  bum carafe slightly wide bit short metal struc...      1
249997  I kettle one month leak water leak seal bottom...      1
249998  idea color balloon entice order package child ...      1
249999  product fail almost immediately digit garble s...      1

                                      avg_input_features_1  \
0       [0.04277208, -0.03597005, -0.062435575, 0.1046...
1       [-0.004893621, 0.029286703, -0.01199023, 0.162...
2       [0.1432408, 0.08569336, -0.048673358, 0.078264...
4       [0.043584187, -0.013412476, -0.116475426, 0.06...
5       [0.0077209473, -0.015841166, -0.04876624, 0.11...
...                                                    ...
249993  [0.03401947, 0.05153087, -0.0007176717, 0.0253...
249996  [-0.001551011, 0.026309744, -0.06418026, 0.125...
249997  [0.0027923584, 0.092679344, -0.03684489, 0.028...
249998  [0.047094908, 0.011726828, 0.00012925093, 0.09...
249999  [0.085134655, -0.011324369, 0.06199294, 0.0255...

                                      avg_input_features_2  \
0       [0.017703589, -0.11186184, -0.0030522645, -0.0...
1       [0.120273024, -0.14361034, 0.046780374, -0.138...
2       [-0.049596105, -0.018341891, 0.13302507, -0.17...
4       [0.08915458, -0.22801971, -0.028520422, -0.263...
5       [0.0042549637, -0.026836593, 0.14918885, -0.08...
...                                                    ...
249993  [0.050901376, -0.11194899, 0.12081799, -0.0080...
249996  [0.015504825, -0.031771064, 0.1092756, -0.0557...
249997  [0.020719932, -0.090553395, 0.13070571, -0.027...
249998  [0.066825956, -0.17564225, 0.05628306, -0.0763...
249999  [0.0051919767, -0.1441225, 0.13658296, -0.1857...

                                   concat_input_features_1  \
```

```
0         [0.06640625, -0.103027344, -0.08251953, 0.1079...
1         [0.107910156, -0.030029297, 0.033203125, -0.16...
2         [0.27929688, 0.29101562, -0.21386719, -0.14648...
4         [0.024291992, 0.010803223, -0.107421875, 0.302...
5         [0.037841797, -0.060058594, -0.05810547, -0.15...
...                                                    ...
249993    [0.14453125, -0.07421875, -0.043945312, 0.2382...
249996    [0.10546875, -0.20117188, -0.13964844, 0.32226...
249997    [0.07910156, -0.0050354004, 0.111816406, 0.212...
249998    [0.067871094, 0.011657715, 0.033691406, 0.2207...
249999    [-0.061523438, 0.095214844, 0.13378906, 0.0649...

                                    concat_input_features_2
0         [0.18149155, -0.23886244, -0.0827184, 0.060127...
1         [0.39106262, -0.43970776, -0.014117015, 0.1198...
2         [-0.07464662, -0.21261097, -0.26036084, -0.465...
4         [0.102800496, -0.12086469, -0.14640297, 0.0537...
5         [0.15096039, 0.03984432, 0.08405365, -0.053545...
...                                                    ...
249993    [0.28356823, 0.13480736, -0.103595145, 0.34340...
249996    [0.14765103, -0.15398727, 0.014575721, -0.1541...
249997    [0.2060052, -0.18501587, -0.0031185225, -0.029...
249998    [0.07186723, -0.11819719, -0.024285497, -0.130...
249999    [0.17383887, 0.03144031, -0.15070951, -0.04374...

[200000 rows x 6 columns]
```

[47]:
```
# set parameters

input_size = 3000
hidden_1_size = 50
hidden_2_size = 10
output_size = 1
```

24 Google model

[85]:
```
x = df_binary['concat_input_features_1']
y = df_binary['class']

# Split the dataset into 80% training dataset and 20% testing dataset

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20,␣
 ↪random_state=100)
```

[86]:
```
## train data
train_data = trainData(torch.FloatTensor(x_train.tolist()),
                       torch.FloatTensor(y_train))
## test data
test_data = testData(torch.FloatTensor(x_test.tolist()))
```

[87]:
```
train_loader = DataLoader(dataset=train_data, batch_size=BATCH_SIZE,␣
 ↪shuffle=True)
test_loader = DataLoader(dataset=test_data, batch_size=1)
```

[48]:
```
# print model

model = binary_classification()
print(model)
```

```
binary_classification(
  (layer_1): Linear(in_features=3000, out_features=50, bias=True)
  (layer_2): Linear(in_features=50, out_features=10, bias=True)
  (layer_out): Linear(in_features=10, out_features=1, bias=True)
  (relu): ReLU()
  (dropout): Dropout(p=0.1, inplace=False)
  (batchnorm1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (batchnorm2): BatchNorm1d(10, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
```

[49]:
```
# define loss function and optimizer

criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)
```

[79]:
```
train_model_binary()
```

```
Epoch 001: | Loss: 0.51775 | Acc: 74.389
Epoch 002: | Loss: 0.48779 | Acc: 76.416
Epoch 003: | Loss: 0.46733 | Acc: 77.754
Epoch 004: | Loss: 0.45035 | Acc: 78.726
Epoch 005: | Loss: 0.43386 | Acc: 79.877
Epoch 006: | Loss: 0.41914 | Acc: 80.656
Epoch 007: | Loss: 0.40263 | Acc: 81.550
Epoch 008: | Loss: 0.39206 | Acc: 82.135
Epoch 009: | Loss: 0.37960 | Acc: 82.766
Epoch 010: | Loss: 0.36853 | Acc: 83.453
Epoch 011: | Loss: 0.35869 | Acc: 83.838
Epoch 012: | Loss: 0.34956 | Acc: 84.293
Epoch 013: | Loss: 0.34203 | Acc: 84.739
```

```
Epoch 014: | Loss: 0.33421 | Acc: 85.152
Epoch 015: | Loss: 0.32706 | Acc: 85.552
Epoch 016: | Loss: 0.31937 | Acc: 85.869
Epoch 017: | Loss: 0.31509 | Acc: 86.163
Epoch 018: | Loss: 0.30826 | Acc: 86.475
Epoch 019: | Loss: 0.30178 | Acc: 86.912
Epoch 020: | Loss: 0.29440 | Acc: 87.188
Epoch 021: | Loss: 0.29068 | Acc: 87.469
Epoch 022: | Loss: 0.28397 | Acc: 87.698
Epoch 023: | Loss: 0.27952 | Acc: 88.007
Epoch 024: | Loss: 0.27510 | Acc: 88.255
Epoch 025: | Loss: 0.27316 | Acc: 88.382
Epoch 026: | Loss: 0.26708 | Acc: 88.584
Epoch 027: | Loss: 0.26345 | Acc: 88.776
Epoch 028: | Loss: 0.26083 | Acc: 88.868
Epoch 029: | Loss: 0.25491 | Acc: 89.221
Epoch 030: | Loss: 0.25360 | Acc: 89.345
Epoch 031: | Loss: 0.24915 | Acc: 89.467
Epoch 032: | Loss: 0.24396 | Acc: 89.727
Epoch 033: | Loss: 0.24337 | Acc: 89.830
Epoch 034: | Loss: 0.24063 | Acc: 89.904
Epoch 035: | Loss: 0.23791 | Acc: 90.053
Epoch 036: | Loss: 0.23743 | Acc: 90.127
Epoch 037: | Loss: 0.23167 | Acc: 90.332
Epoch 038: | Loss: 0.23147 | Acc: 90.436
Epoch 039: | Loss: 0.22834 | Acc: 90.511
Epoch 040: | Loss: 0.22583 | Acc: 90.581
Epoch 041: | Loss: 0.22392 | Acc: 90.703
Epoch 042: | Loss: 0.22017 | Acc: 90.826
Epoch 043: | Loss: 0.21828 | Acc: 90.966
Epoch 044: | Loss: 0.21565 | Acc: 91.126
Epoch 045: | Loss: 0.21212 | Acc: 91.270
Epoch 046: | Loss: 0.21202 | Acc: 91.290
Epoch 047: | Loss: 0.21045 | Acc: 91.366
Epoch 048: | Loss: 0.20793 | Acc: 91.474
Epoch 049: | Loss: 0.20676 | Acc: 91.513
Epoch 050: | Loss: 0.20067 | Acc: 91.831
Epoch 051: | Loss: 0.20422 | Acc: 91.692
Epoch 052: | Loss: 0.20268 | Acc: 91.831
Epoch 053: | Loss: 0.19838 | Acc: 91.938
Epoch 054: | Loss: 0.19680 | Acc: 91.986
Epoch 055: | Loss: 0.19702 | Acc: 92.025
Epoch 056: | Loss: 0.19391 | Acc: 92.185
Epoch 057: | Loss: 0.19374 | Acc: 92.161
Epoch 058: | Loss: 0.19349 | Acc: 92.219
Epoch 059: | Loss: 0.19197 | Acc: 92.248
Epoch 060: | Loss: 0.19062 | Acc: 92.291
Epoch 061: | Loss: 0.18636 | Acc: 92.478
```

```
Epoch 062: | Loss: 0.18559 | Acc: 92.550
Epoch 063: | Loss: 0.18263 | Acc: 92.637
Epoch 064: | Loss: 0.18395 | Acc: 92.572
Epoch 065: | Loss: 0.18240 | Acc: 92.689
Epoch 066: | Loss: 0.18033 | Acc: 92.797
Epoch 067: | Loss: 0.17867 | Acc: 92.818
Epoch 068: | Loss: 0.17821 | Acc: 92.855
Epoch 069: | Loss: 0.17768 | Acc: 92.845
Epoch 070: | Loss: 0.17720 | Acc: 92.916
Epoch 071: | Loss: 0.17591 | Acc: 93.016
Epoch 072: | Loss: 0.17262 | Acc: 93.171
Epoch 073: | Loss: 0.17319 | Acc: 93.066
Epoch 074: | Loss: 0.17144 | Acc: 93.142
Epoch 075: | Loss: 0.17292 | Acc: 93.126
Epoch 076: | Loss: 0.17108 | Acc: 93.180
Epoch 077: | Loss: 0.17008 | Acc: 93.266
Epoch 078: | Loss: 0.16747 | Acc: 93.394
Epoch 079: | Loss: 0.16930 | Acc: 93.291
Epoch 080: | Loss: 0.17138 | Acc: 93.216
Epoch 081: | Loss: 0.16733 | Acc: 93.356
Epoch 082: | Loss: 0.16468 | Acc: 93.520
Epoch 083: | Loss: 0.16301 | Acc: 93.583
Epoch 084: | Loss: 0.16332 | Acc: 93.579
Epoch 085: | Loss: 0.16133 | Acc: 93.643
Epoch 086: | Loss: 0.16073 | Acc: 93.674
Epoch 087: | Loss: 0.16171 | Acc: 93.589
Epoch 088: | Loss: 0.15998 | Acc: 93.707
Epoch 089: | Loss: 0.15461 | Acc: 93.942
Epoch 090: | Loss: 0.15761 | Acc: 93.778
Epoch 091: | Loss: 0.15520 | Acc: 93.917
Epoch 092: | Loss: 0.15874 | Acc: 93.737
Epoch 093: | Loss: 0.15490 | Acc: 93.859
Epoch 094: | Loss: 0.15726 | Acc: 93.867
Epoch 095: | Loss: 0.15412 | Acc: 93.957
Epoch 096: | Loss: 0.15527 | Acc: 93.920
Epoch 097: | Loss: 0.15266 | Acc: 94.091
Epoch 098: | Loss: 0.15011 | Acc: 94.139
Epoch 099: | Loss: 0.15298 | Acc: 94.015
Epoch 100: | Loss: 0.15289 | Acc: 94.037
```

[91]:
```
test_model_binary(y_test)
```

```
Accuracy: 73.0
```

25 Our model

[31]:
```
x = df_binary['concat_input_features_2']
y = df_binary['class']

# Split the dataset into 80% training dataset and 20% testing dataset

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20,␣
 ↪random_state=100)
```

[35]:
```
## train data
train_data = trainData(torch.FloatTensor(x_train.tolist()),
                       torch.FloatTensor(y_train))
## test data
test_data = testData(torch.FloatTensor(x_test.tolist()))
```

[36]:
```
train_loader = DataLoader(dataset=train_data, batch_size=BATCH_SIZE,␣
 ↪shuffle=True)
test_loader = DataLoader(dataset=test_data, batch_size=1)
```

[83]:
```
train_model_binary()
```

```
Epoch 001: | Loss: 0.52392 | Acc: 74.279
Epoch 002: | Loss: 0.46477 | Acc: 77.858
Epoch 003: | Loss: 0.45286 | Acc: 78.440
Epoch 004: | Loss: 0.44242 | Acc: 79.154
Epoch 005: | Loss: 0.42896 | Acc: 79.806
Epoch 006: | Loss: 0.41631 | Acc: 80.604
Epoch 007: | Loss: 0.40414 | Acc: 81.266
Epoch 008: | Loss: 0.39191 | Acc: 81.942
Epoch 009: | Loss: 0.37769 | Acc: 82.774
Epoch 010: | Loss: 0.36850 | Acc: 83.203
Epoch 011: | Loss: 0.35949 | Acc: 83.771
Epoch 012: | Loss: 0.34855 | Acc: 84.371
Epoch 013: | Loss: 0.33820 | Acc: 84.924
Epoch 014: | Loss: 0.33224 | Acc: 85.205
Epoch 015: | Loss: 0.32233 | Acc: 85.711
Epoch 016: | Loss: 0.31424 | Acc: 86.075
Epoch 017: | Loss: 0.30655 | Acc: 86.567
Epoch 018: | Loss: 0.30080 | Acc: 86.804
Epoch 019: | Loss: 0.29432 | Acc: 87.156
Epoch 020: | Loss: 0.28855 | Acc: 87.446
Epoch 021: | Loss: 0.28392 | Acc: 87.642
Epoch 022: | Loss: 0.27764 | Acc: 87.976
Epoch 023: | Loss: 0.27157 | Acc: 88.361
Epoch 024: | Loss: 0.26821 | Acc: 88.508
Epoch 025: | Loss: 0.26392 | Acc: 88.649
```

```
Epoch 026: | Loss: 0.25885 | Acc: 88.853
Epoch 027: | Loss: 0.25466 | Acc: 89.076
Epoch 028: | Loss: 0.25365 | Acc: 89.181
Epoch 029: | Loss: 0.24837 | Acc: 89.466
Epoch 030: | Loss: 0.24390 | Acc: 89.680
Epoch 031: | Loss: 0.23868 | Acc: 89.868
Epoch 032: | Loss: 0.23578 | Acc: 90.108
Epoch 033: | Loss: 0.23380 | Acc: 90.110
Epoch 034: | Loss: 0.22838 | Acc: 90.402
Epoch 035: | Loss: 0.22560 | Acc: 90.534
Epoch 036: | Loss: 0.22524 | Acc: 90.596
Epoch 037: | Loss: 0.22083 | Acc: 90.766
Epoch 038: | Loss: 0.21918 | Acc: 90.830
Epoch 039: | Loss: 0.21701 | Acc: 90.969
Epoch 040: | Loss: 0.21415 | Acc: 91.147
Epoch 041: | Loss: 0.21191 | Acc: 91.179
Epoch 042: | Loss: 0.20940 | Acc: 91.334
Epoch 043: | Loss: 0.20898 | Acc: 91.418
Epoch 044: | Loss: 0.20536 | Acc: 91.519
Epoch 045: | Loss: 0.20234 | Acc: 91.659
Epoch 046: | Loss: 0.20109 | Acc: 91.770
Epoch 047: | Loss: 0.20039 | Acc: 91.786
Epoch 048: | Loss: 0.19649 | Acc: 91.987
Epoch 049: | Loss: 0.19486 | Acc: 92.126
Epoch 050: | Loss: 0.19592 | Acc: 91.933
Epoch 051: | Loss: 0.19143 | Acc: 92.219
Epoch 052: | Loss: 0.18970 | Acc: 92.348
Epoch 053: | Loss: 0.18813 | Acc: 92.352
Epoch 054: | Loss: 0.18709 | Acc: 92.403
Epoch 055: | Loss: 0.18553 | Acc: 92.527
Epoch 056: | Loss: 0.18574 | Acc: 92.422
Epoch 057: | Loss: 0.18404 | Acc: 92.596
Epoch 058: | Loss: 0.18102 | Acc: 92.691
Epoch 059: | Loss: 0.17956 | Acc: 92.779
Epoch 060: | Loss: 0.17721 | Acc: 92.861
Epoch 061: | Loss: 0.17563 | Acc: 92.989
Epoch 062: | Loss: 0.17797 | Acc: 92.803
Epoch 063: | Loss: 0.17562 | Acc: 92.955
Epoch 064: | Loss: 0.16973 | Acc: 93.213
Epoch 065: | Loss: 0.17218 | Acc: 93.116
Epoch 066: | Loss: 0.17179 | Acc: 93.054
Epoch 067: | Loss: 0.16945 | Acc: 93.289
Epoch 068: | Loss: 0.16897 | Acc: 93.194
Epoch 069: | Loss: 0.16599 | Acc: 93.335
Epoch 070: | Loss: 0.16764 | Acc: 93.335
Epoch 071: | Loss: 0.16726 | Acc: 93.421
Epoch 072: | Loss: 0.16422 | Acc: 93.443
Epoch 073: | Loss: 0.16392 | Acc: 93.451
```

```
Epoch 074: | Loss: 0.16250 | Acc: 93.578
Epoch 075: | Loss: 0.16122 | Acc: 93.566
Epoch 076: | Loss: 0.15902 | Acc: 93.722
Epoch 077: | Loss: 0.15962 | Acc: 93.664
Epoch 078: | Loss: 0.15592 | Acc: 93.852
Epoch 079: | Loss: 0.15888 | Acc: 93.730
Epoch 080: | Loss: 0.15686 | Acc: 93.798
Epoch 081: | Loss: 0.15381 | Acc: 93.912
Epoch 082: | Loss: 0.15526 | Acc: 93.907
Epoch 083: | Loss: 0.15428 | Acc: 93.912
Epoch 084: | Loss: 0.15345 | Acc: 93.929
Epoch 085: | Loss: 0.14975 | Acc: 94.069
Epoch 086: | Loss: 0.15242 | Acc: 94.016
Epoch 087: | Loss: 0.15183 | Acc: 94.072
Epoch 088: | Loss: 0.15024 | Acc: 94.066
Epoch 089: | Loss: 0.15098 | Acc: 94.114
Epoch 090: | Loss: 0.14866 | Acc: 94.149
Epoch 091: | Loss: 0.15012 | Acc: 94.067
Epoch 092: | Loss: 0.14592 | Acc: 94.261
Epoch 093: | Loss: 0.14947 | Acc: 94.131
Epoch 094: | Loss: 0.14673 | Acc: 94.283
Epoch 095: | Loss: 0.14571 | Acc: 94.312
Epoch 096: | Loss: 0.14395 | Acc: 94.397
Epoch 097: | Loss: 0.14283 | Acc: 94.391
Epoch 098: | Loss: 0.14514 | Acc: 94.362
Epoch 099: | Loss: 0.14141 | Acc: 94.493
Epoch 100: | Loss: 0.14172 | Acc: 94.451
```

[51]:
```
test_model_binary(y_test)
```

```
Accuracy: 75.0
```

26 Ternary

[52]:
```
# ternary classification dataframe

df_ternary = df_org_3.copy(deep=True)
df_ternary
```

[52]:
```
                                              review_body  class  \
0       assume four charger bought item pretty bought ...      1
1       son like cook he especially good grill burger ...      0
2                     ship fast good price way huger expect      0
3           container great lid thin break easily one use      2
4       item receive broken return ask replacement shi...      1
...                                                   ...    ...
```

```
249995                  lock come easily hard clean top      2
249996  bum carafe slightly wide bit short metal struc...      1
249997  I kettle one month leak water leak seal bottom...      1
249998  idea color balloon entice order package child ...      1
249999  product fail almost immediately digit garble s...      1

                                      avg_input_features_1  \
0       [0.04277208, -0.03597005, -0.062435575, 0.1046...
1       [-0.004893621, 0.029286703, -0.01199023, 0.162...
2       [0.1432408, 0.08569336, -0.048673358, 0.078264...
3       [0.056274414, 0.10064697, -0.0005340576, 0.056...
4       [0.043584187, -0.013412476, -0.116475426, 0.06...
...                                                   ...
249995  [0.03120931, 0.07987467, 0.03741455, 0.0357869...
249996  [-0.001551011, 0.026309744, -0.06418026, 0.125...
249997  [0.0027923584, 0.092679344, -0.03684489, 0.028...
249998  [0.047094908, 0.011726828, 0.00012925093, 0.09...
249999  [0.085134655, -0.011324369, 0.06199294, 0.0255...

                                      avg_input_features_2  \
0       [0.017703589, -0.11186184, -0.0030522645, -0.0...
1       [0.120273024, -0.14361034, 0.046780374, -0.138...
2       [-0.049596105, -0.018341891, 0.13302507, -0.17...
3       [0.030435072, -0.15327847, 0.11309578, -0.1425...
4       [0.08915458, -0.22801971, -0.028520422, -0.263...
...                                                   ...
249995  [0.015699785, -0.12990652, 0.21889718, -0.1027...
249996  [0.015504825, -0.031771064, 0.1092756, -0.0557...
249997  [0.020719932, -0.090553395, 0.13070571, -0.027...
249998  [0.066825956, -0.17564225, 0.05628306, -0.0763...
249999  [0.0051919767, -0.1441225, 0.13658296, -0.1857...

                                   concat_input_features_1  \
0       [0.06640625, -0.103027344, -0.08251953, 0.1079...
1       [0.107910156, -0.030029297, 0.033203125, -0.16...
2       [0.27929688, 0.29101562, -0.21386719, -0.14648...
3       [0.048095703, 0.31640625, 0.17773438, -0.06982...
4       [0.024291992, 0.010803223, -0.107421875, 0.302...
...                                                   ...
249995  [0.017944336, 0.19335938, -0.06298828, 0.02429...
249996  [0.10546875, -0.20117188, -0.13964844, 0.32226...
249997  [0.07910156, -0.0050354004, 0.111816406, 0.212...
249998  [0.067871094, 0.011657715, 0.033691406, 0.2207...
249999  [-0.061523438, 0.095214844, 0.13378906, 0.0649...

                                   concat_input_features_2
0       [0.18149155, -0.23886244, -0.0827184, 0.060127...
```

```
1         [0.39106262, -0.43970776, -0.014117015, 0.1198...
2         [-0.07464662, -0.21261097, -0.26036084, -0.465...
3         [-0.044359308, -0.092595585, 0.07619203, -0.14...
4         [0.102800496, -0.12086469, -0.14640297, 0.0537...
...                                                   ...
249995   [0.24208477, -0.24096622, 0.30787, -0.2916415,...
249996   [0.14765103, -0.15398727, 0.014575721, -0.1541...
249997   [0.2060052, -0.18501587, -0.0031185225, -0.029...
249998   [0.07186723, -0.11819719, -0.024285497, -0.130...
249999   [0.17383887, 0.03144031, -0.15070951, -0.04374...

[250000 rows x 6 columns]
```

[68]:
```
# set parameters

input_size = 3000
hidden_1_size = 50
hidden_2_size = 10
output_size = 3
```

27 Google model

[69]:
```
x = df_ternary['concat_input_features_1']
y = df_ternary['class']

# Split the dataset into 80% training dataset and 20% testing dataset

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20,
 ↪random_state=100)
```

[70]:
```
## train data
train_data = trainData(torch.FloatTensor(x_train.tolist()),
                       torch.FloatTensor(y_train))
## test data
test_data = testData(torch.FloatTensor(x_test.tolist()))
```

[71]:
```
train_loader = DataLoader(dataset=train_data, batch_size=BATCH_SIZE,
 ↪shuffle=True)
test_loader = DataLoader(dataset=test_data, batch_size=1)
```

[72]:
```
# print model

model = ternary_classification()
print(model)
```

```
ternary_classification(
```

```
  (layer_1): Linear(in_features=3000, out_features=50, bias=True)
  (layer_2): Linear(in_features=50, out_features=10, bias=True)
  (layer_out): Linear(in_features=10, out_features=3, bias=True)
  (relu): ReLU()
  (dropout): Dropout(p=0.1, inplace=False)
  (batchnorm1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (batchnorm2): BatchNorm1d(10, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
```

[73]:
```
# define loss function and optimizer

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)
```

[91]:
```
train_model_ternary()
```

```
Epoch 001: | Loss: 0.89409 | Acc: 59.612
Epoch 002: | Loss: 0.86054 | Acc: 61.697
Epoch 003: | Loss: 0.84039 | Acc: 62.864
Epoch 004: | Loss: 0.82043 | Acc: 64.089
Epoch 005: | Loss: 0.80407 | Acc: 64.951
Epoch 006: | Loss: 0.78953 | Acc: 65.698
Epoch 007: | Loss: 0.77770 | Acc: 66.365
Epoch 008: | Loss: 0.76451 | Acc: 67.052
Epoch 009: | Loss: 0.75307 | Acc: 67.719
Epoch 010: | Loss: 0.73976 | Acc: 68.380
Epoch 011: | Loss: 0.73238 | Acc: 68.749
Epoch 012: | Loss: 0.72255 | Acc: 69.247
Epoch 013: | Loss: 0.71186 | Acc: 69.825
Epoch 014: | Loss: 0.70388 | Acc: 70.320
Epoch 015: | Loss: 0.69564 | Acc: 70.564
Epoch 016: | Loss: 0.68788 | Acc: 71.011
Epoch 017: | Loss: 0.68257 | Acc: 71.261
Epoch 018: | Loss: 0.67523 | Acc: 71.591
Epoch 019: | Loss: 0.66889 | Acc: 71.825
Epoch 020: | Loss: 0.66310 | Acc: 72.161
Epoch 021: | Loss: 0.65745 | Acc: 72.427
Epoch 022: | Loss: 0.65151 | Acc: 72.799
Epoch 023: | Loss: 0.64737 | Acc: 72.997
Epoch 024: | Loss: 0.64228 | Acc: 73.146
Epoch 025: | Loss: 0.63782 | Acc: 73.400
Epoch 026: | Loss: 0.63328 | Acc: 73.663
Epoch 027: | Loss: 0.62768 | Acc: 73.875
Epoch 028: | Loss: 0.62589 | Acc: 74.035
Epoch 029: | Loss: 0.62183 | Acc: 74.243
Epoch 030: | Loss: 0.61579 | Acc: 74.468
```

```
Epoch 031: | Loss: 0.61155 | Acc: 74.739
Epoch 032: | Loss: 0.60932 | Acc: 74.796
Epoch 033: | Loss: 0.60549 | Acc: 75.019
Epoch 034: | Loss: 0.60378 | Acc: 75.052
Epoch 035: | Loss: 0.60098 | Acc: 75.250
Epoch 036: | Loss: 0.59713 | Acc: 75.217
Epoch 037: | Loss: 0.59433 | Acc: 75.460
Epoch 038: | Loss: 0.58976 | Acc: 75.642
Epoch 039: | Loss: 0.58772 | Acc: 75.767
Epoch 040: | Loss: 0.58420 | Acc: 76.007
Epoch 041: | Loss: 0.58042 | Acc: 76.040
Epoch 042: | Loss: 0.57987 | Acc: 76.147
Epoch 043: | Loss: 0.57679 | Acc: 76.308
Epoch 044: | Loss: 0.57488 | Acc: 76.290
Epoch 045: | Loss: 0.57005 | Acc: 76.612
Epoch 046: | Loss: 0.56813 | Acc: 76.662
Epoch 047: | Loss: 0.56902 | Acc: 76.677
Epoch 048: | Loss: 0.56461 | Acc: 76.789
Epoch 049: | Loss: 0.56170 | Acc: 77.024
Epoch 050: | Loss: 0.56064 | Acc: 77.014
Epoch 051: | Loss: 0.55872 | Acc: 77.148
Epoch 052: | Loss: 0.55630 | Acc: 77.137
Epoch 053: | Loss: 0.55580 | Acc: 77.317
Epoch 054: | Loss: 0.55231 | Acc: 77.439
Epoch 055: | Loss: 0.55088 | Acc: 77.516
Epoch 056: | Loss: 0.54795 | Acc: 77.626
Epoch 057: | Loss: 0.54627 | Acc: 77.739
Epoch 058: | Loss: 0.54487 | Acc: 77.800
Epoch 059: | Loss: 0.54224 | Acc: 77.866
Epoch 060: | Loss: 0.54276 | Acc: 77.838
Epoch 061: | Loss: 0.53925 | Acc: 78.082
Epoch 062: | Loss: 0.53904 | Acc: 78.014
Epoch 063: | Loss: 0.53401 | Acc: 78.223
Epoch 064: | Loss: 0.53403 | Acc: 78.257
Epoch 065: | Loss: 0.53371 | Acc: 78.263
Epoch 066: | Loss: 0.53195 | Acc: 78.272
Epoch 067: | Loss: 0.52904 | Acc: 78.425
Epoch 068: | Loss: 0.52977 | Acc: 78.388
Epoch 069: | Loss: 0.53012 | Acc: 78.323
Epoch 070: | Loss: 0.52463 | Acc: 78.707
Epoch 071: | Loss: 0.52668 | Acc: 78.537
Epoch 072: | Loss: 0.52308 | Acc: 78.681
Epoch 073: | Loss: 0.52253 | Acc: 78.739
Epoch 074: | Loss: 0.51869 | Acc: 78.921
Epoch 075: | Loss: 0.51956 | Acc: 78.861
Epoch 076: | Loss: 0.51936 | Acc: 78.896
Epoch 077: | Loss: 0.51554 | Acc: 79.059
Epoch 078: | Loss: 0.51546 | Acc: 79.102
```

```
Epoch 079: | Loss: 0.51374 | Acc: 79.186
Epoch 080: | Loss: 0.51427 | Acc: 79.210
Epoch 081: | Loss: 0.51180 | Acc: 79.234
Epoch 082: | Loss: 0.50894 | Acc: 79.388
Epoch 083: | Loss: 0.50955 | Acc: 79.352
Epoch 084: | Loss: 0.50889 | Acc: 79.410
Epoch 085: | Loss: 0.50590 | Acc: 79.572
Epoch 086: | Loss: 0.50508 | Acc: 79.623
Epoch 087: | Loss: 0.50309 | Acc: 79.659
Epoch 088: | Loss: 0.50271 | Acc: 79.645
Epoch 089: | Loss: 0.50091 | Acc: 79.829
Epoch 090: | Loss: 0.49891 | Acc: 79.856
Epoch 091: | Loss: 0.50053 | Acc: 79.832
Epoch 092: | Loss: 0.49985 | Acc: 79.796
Epoch 093: | Loss: 0.49919 | Acc: 79.853
Epoch 094: | Loss: 0.49757 | Acc: 79.870
Epoch 095: | Loss: 0.49481 | Acc: 80.024
Epoch 096: | Loss: 0.49288 | Acc: 80.033
Epoch 097: | Loss: 0.49335 | Acc: 80.058
Epoch 098: | Loss: 0.49105 | Acc: 80.138
Epoch 099: | Loss: 0.49011 | Acc: 80.081
Epoch 100: | Loss: 0.49301 | Acc: 80.119
```

[75]:
```
test_model_ternary(y_test)
```

```
Accuracy: 57.0
```

28 Our model

[76]:
```
x = df_ternary['concat_input_features_2']
y = df_ternary['class']

# Split the dataset into 80% training dataset and 20% testing dataset

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20,␣
 ↪random_state=100)
```

[77]:
```
## train data
train_data = trainData(torch.FloatTensor(x_train.tolist()),
                       torch.FloatTensor(y_train))
## test data
test_data = testData(torch.FloatTensor(x_test.tolist()))
```

[78]:
```
train_loader = DataLoader(dataset=train_data, batch_size=BATCH_SIZE,␣
 ↪shuffle=True)
test_loader = DataLoader(dataset=test_data, batch_size=1)
```

[95]:
```
train_model_ternary()
```

```
Epoch 001: | Loss: 0.90813 | Acc: 59.358
Epoch 002: | Loss: 0.84278 | Acc: 62.880
Epoch 003: | Loss: 0.82552 | Acc: 63.622
Epoch 004: | Loss: 0.81350 | Acc: 64.262
Epoch 005: | Loss: 0.80305 | Acc: 64.778
Epoch 006: | Loss: 0.79276 | Acc: 65.254
Epoch 007: | Loss: 0.78232 | Acc: 65.812
Epoch 008: | Loss: 0.77178 | Acc: 66.382
Epoch 009: | Loss: 0.76321 | Acc: 66.742
Epoch 010: | Loss: 0.75476 | Acc: 67.319
Epoch 011: | Loss: 0.74597 | Acc: 67.665
Epoch 012: | Loss: 0.73657 | Acc: 68.159
Epoch 013: | Loss: 0.72833 | Acc: 68.512
Epoch 014: | Loss: 0.72178 | Acc: 68.912
Epoch 015: | Loss: 0.71326 | Acc: 69.347
Epoch 016: | Loss: 0.70599 | Acc: 69.620
Epoch 017: | Loss: 0.69941 | Acc: 70.033
Epoch 018: | Loss: 0.69192 | Acc: 70.278
Epoch 019: | Loss: 0.68419 | Acc: 70.690
Epoch 020: | Loss: 0.67823 | Acc: 71.061
Epoch 021: | Loss: 0.67399 | Acc: 71.223
Epoch 022: | Loss: 0.66797 | Acc: 71.364
Epoch 023: | Loss: 0.66216 | Acc: 71.787
Epoch 024: | Loss: 0.65866 | Acc: 71.964
Epoch 025: | Loss: 0.65156 | Acc: 72.379
Epoch 026: | Loss: 0.64853 | Acc: 72.467
Epoch 027: | Loss: 0.64281 | Acc: 72.623
Epoch 028: | Loss: 0.63850 | Acc: 72.835
Epoch 029: | Loss: 0.63807 | Acc: 72.868
Epoch 030: | Loss: 0.63089 | Acc: 73.273
Epoch 031: | Loss: 0.62798 | Acc: 73.379
Epoch 032: | Loss: 0.62395 | Acc: 73.424
Epoch 033: | Loss: 0.62227 | Acc: 73.554
Epoch 034: | Loss: 0.61642 | Acc: 73.916
Epoch 035: | Loss: 0.61378 | Acc: 73.963
Epoch 036: | Loss: 0.61180 | Acc: 74.090
Epoch 037: | Loss: 0.60802 | Acc: 74.188
Epoch 038: | Loss: 0.60427 | Acc: 74.467
Epoch 039: | Loss: 0.60194 | Acc: 74.472
Epoch 040: | Loss: 0.59804 | Acc: 74.652
Epoch 041: | Loss: 0.59637 | Acc: 74.876
Epoch 042: | Loss: 0.59132 | Acc: 75.026
Epoch 043: | Loss: 0.58989 | Acc: 75.079
Epoch 044: | Loss: 0.58785 | Acc: 75.195
Epoch 045: | Loss: 0.58617 | Acc: 75.153
```

```
Epoch 046: | Loss: 0.58237 | Acc: 75.389
Epoch 047: | Loss: 0.57918 | Acc: 75.473
Epoch 048: | Loss: 0.57743 | Acc: 75.582
Epoch 049: | Loss: 0.57674 | Acc: 75.671
Epoch 050: | Loss: 0.57433 | Acc: 75.673
Epoch 051: | Loss: 0.57177 | Acc: 75.849
Epoch 052: | Loss: 0.56950 | Acc: 76.007
Epoch 053: | Loss: 0.56736 | Acc: 75.990
Epoch 054: | Loss: 0.56798 | Acc: 76.041
Epoch 055: | Loss: 0.56377 | Acc: 76.275
Epoch 056: | Loss: 0.56241 | Acc: 76.281
Epoch 057: | Loss: 0.55912 | Acc: 76.452
Epoch 058: | Loss: 0.55898 | Acc: 76.465
Epoch 059: | Loss: 0.55742 | Acc: 76.492
Epoch 060: | Loss: 0.55659 | Acc: 76.529
Epoch 061: | Loss: 0.55401 | Acc: 76.680
Epoch 062: | Loss: 0.55334 | Acc: 76.715
Epoch 063: | Loss: 0.54936 | Acc: 76.879
Epoch 064: | Loss: 0.54824 | Acc: 76.985
Epoch 065: | Loss: 0.54806 | Acc: 76.932
Epoch 066: | Loss: 0.54471 | Acc: 77.099
Epoch 067: | Loss: 0.54541 | Acc: 77.037
Epoch 068: | Loss: 0.54278 | Acc: 77.109
Epoch 069: | Loss: 0.54018 | Acc: 77.318
Epoch 070: | Loss: 0.54088 | Acc: 77.314
Epoch 071: | Loss: 0.53776 | Acc: 77.374
Epoch 072: | Loss: 0.53525 | Acc: 77.538
Epoch 073: | Loss: 0.53533 | Acc: 77.542
Epoch 074: | Loss: 0.53461 | Acc: 77.618
Epoch 075: | Loss: 0.53288 | Acc: 77.670
Epoch 076: | Loss: 0.53306 | Acc: 77.633
Epoch 077: | Loss: 0.53029 | Acc: 77.811
Epoch 078: | Loss: 0.52832 | Acc: 77.823
Epoch 079: | Loss: 0.52611 | Acc: 77.954
Epoch 080: | Loss: 0.52590 | Acc: 78.003
Epoch 081: | Loss: 0.52463 | Acc: 77.964
Epoch 082: | Loss: 0.52400 | Acc: 78.034
Epoch 083: | Loss: 0.52396 | Acc: 78.005
Epoch 084: | Loss: 0.52227 | Acc: 78.138
Epoch 085: | Loss: 0.52036 | Acc: 78.137
Epoch 086: | Loss: 0.51909 | Acc: 78.262
Epoch 087: | Loss: 0.51850 | Acc: 78.376
Epoch 088: | Loss: 0.51968 | Acc: 78.184
Epoch 089: | Loss: 0.51669 | Acc: 78.319
Epoch 090: | Loss: 0.51599 | Acc: 78.419
Epoch 091: | Loss: 0.51438 | Acc: 78.424
Epoch 092: | Loss: 0.51439 | Acc: 78.459
Epoch 093: | Loss: 0.51389 | Acc: 78.460
```

```
Epoch 094: | Loss: 0.51253 | Acc: 78.600
Epoch 095: | Loss: 0.51093 | Acc: 78.607
Epoch 096: | Loss: 0.51090 | Acc: 78.544
Epoch 097: | Loss: 0.50919 | Acc: 78.671
Epoch 098: | Loss: 0.51058 | Acc: 78.664
Epoch 099: | Loss: 0.50687 | Acc: 78.781
Epoch 100: | Loss: 0.50572 | Acc: 78.916
```

[80]:
```
test_model_ternary(y_test)
```

```
Accuracy: 59.0
```

29 Comments about this question

[35]:
```
d = {'Model': ['FNN', 'FNN', 'FNN', 'FNN'],
     'Word2Vec Model': ['Google News', 'Amazon Reviews(Our)', 'Google News',␣
 ↪'Amazon Reviews(Our)'],
     'Classification Type': ['Binary', 'Binary', 'Ternary', 'Ternary',],
     'Input Features Type': ['Concat_first_10' , 'Concat_first_10',␣
 ↪'Concat_first_10', 'Concat_first_10'],
     'Accuracy': ['0.73', '0.75', '0.57', '0.59']}

df_results_part_4_b = pd.DataFrame(data=d)
df_results_part_4_b
```

[35]:

	Model	Word2Vec Model	Classification Type	Input Features Type	Accuracy
0	FNN	Google News	Binary	Concat_first_10	0.73
1	FNN	Amazon Reviews(Our)	Binary	Concat_first_10	0.75
2	FNN	Google News	Ternary	Concat_first_10	0.57
3	FNN	Amazon Reviews(Our)	Ternary	Concat_first_10	0.59

30 Comments

[38]:
```
df_results_part_3
```

[38]:

	Model	Word2Vec Features/Other Features	Accuracy
0	Perceptron	Google News	0.71
1	SVM	Google News	0.82
2	Perceptron	Amazon Reviews(Our)	0.81
3	SVM	Amazon Reviews(Our)	0.85
4	Perceptron	TF-IDF	0.85
5	SVM	TF-IDF	0.81

[36]:
```
df_results_part_4_a
```

[36]:

	Model	Word2Vec Model	Classification Type	Input Features Type	Accuracy
0	FNN	Google News	Binary	Average	0.85
1	FNN	Amazon Reviews(Our)	Binary	Average	0.87
2	FNN	Google News	Ternary	Average	0.68
3	FNN	Amazon Reviews(Our)	Ternary	Average	0.71

[37]:
```
df_results_part_4_b
```

[37]:

	Model	Word2Vec Model	Classification Type	Input Features Type	Accuracy
0	FNN	Google News	Binary	Concat_first_10	0.73
1	FNN	Amazon Reviews(Our)	Binary	Concat_first_10	0.75
2	FNN	Google News	Ternary	Concat_first_10	0.57
3	FNN	Amazon Reviews(Our)	Ternary	Concat_first_10	0.59

It can be seen from the above tables that for binary classification(as mentioned in the question pdf note), the FNN model(input features - Average Word2Vec vectors) works better or comparable(in some cases) than both the Perceptron and the SVM model for Google News/Amazon Reviews(Our)/TF-IDF Word2Vec features. However the FNN model(input features - Concat(first 10) vectors) performs poorly than both the Perceptron and the SVM model for Google News/Amaxon Reviews(Our)/TF-IDF Word2Vec features. This shows that the average vectors is a better input feature type selection here than concatenating the first 10 vectors. Also the feedforward MLP model is stronger and slightly more accurate here at binary classification if average vectors are considered. This is so since we get a lot of hyperparameter and design paramter tuning flexibility in Neural Network models(epochs,batch_size,learning_rate,activation functions(linear/non-linear:relu),loss,optimizer,etc.) that can help us achieve possibly a higher accuracy.

[ ]:
```

```