TextClassification

April 2, 2023

Justin Hardy | JEH180008 | Dr. Mazidi | CS 4395.001

The purpose of this assignment is to explore various machine learning algorithms, and implement them in problems related to Natural Language Processing.

1 Imports

[164]:
```
import pandas
import math
import sys
import re as regex
from nltk.corpus import stopwords, wordnet as wn
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.model_selection import train_test_split as tts
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, confusion_matrix, log_loss
```

2 About the Data Set

The data set I'll be using for this assignment is a data sed I'd found on Kaggle, which is a data set that contains 4 million amazon reviews and their corresponding sentiments (negative/positive). The review text includes both the title of the review, as well as the review itself.

You can click this link to view the data set: Amazon Reviews Data Set

3 Reading in the Data Set

We'll start by reading in both the train and test data from the files as data frames. We'll then combine the two data frames and cut down the amount of rows we'll use in the data by a substantial amount. Then do an 80/20 split (rather than the 90/10 split already done).

[63]:
```
# Read train and test
col_names = ['Type', 'Title', 'Review']
```

```
df_partial_1 = pandas.read_csv('data/train.csv', names=col_names, header=None,␣
 ↪encoding='utf-8', keep_default_na=False)
df_partial_2 = pandas.read_csv('data/test.csv', names=col_names, header=None,␣
 ↪encoding='utf-8', keep_default_na=False)

# Combine the data frames
df = pandas.concat([df_partial_1, df_partial_2], ignore_index=True)

# Print Shape of Data Frame
print("Shape before cut:", df.shape)

# Cut down Data Frame size
df_type_1 = df.loc[df['Type'] == 1]
df_type_2 = df.loc[df['Type'] == 2]
df_type_1_cut = df_type_1
df_type_2_cut = df_type_2

# Take a tenth of the Data Frame's contents
df_type_1 = df_type_1.iloc[:int(len(df_type_1)/20)] # 20 = 200,000; 25 = ␣
 ↪160,000; 40 = 100,000
df_type_2 = df_type_2.iloc[:int(len(df_type_2)/20)]

# Combine the two separate Data Frame back into the full Data Frame.
df = pandas.concat([df_type_1, df_type_2], ignore_index=True)

# Convert Type column from 1/2 notation to binary 0/1 notation
df.Type = [{1:0, 2:1}[t] for t in df.Type]

# Print Shape of Data Frame
print("Shape after cut:", df.shape)

# Print Head/Tail of the Data Frame
print("Final Data Frame (head and tail):")
print(df.head())
print()
print(df.tail())
```

```
Shape before cut: (4000000, 3)
Shape after cut: (200000, 3)
Final Data Frame (head and tail):
   Type                                    Title  \
0     0                            Buyer beware
1     0                              The Worst!
2     0                               Oh please
3     0                    Awful beyond belief!
4     0  Don't try to fool us with fake reviews.
```

```
                                                Review
0  This is a self-published book, and if you want…
1  A complete waste of time. Typographical errors…
2  I guess you have to be a romance novel lover f…
3  I feel I have to write to keep others from was…
4  It's glaringly obvious that all of the glowing…

        Type                       Title  \
199995     1                   It Works!
199996     1            Love this book!
199997     1           Good basics book
199998     1  Must read for new parents
199999     1                 great book!

                                                Review
199995  Dr. Harvey Karp has come up with a way to help…
199996  We are expecting our first child and this book…
199997  We got this book upon a recommendation from an…
199998  I enjoyed this book and suggest reading it BEF…
199999  Great book! It takes things you may already kn…
```

4 Text Preprocessing

To process the text, we'll need to specify which columns we'll use as features, and which one will be our target. Since we're vectorizing our features & labels using SKLearn's TF-IDF Vectorizer, we'll need to concatenate the contents of each feature together, so that it can be transformed together.

[64]:
```
# Initialize tfidf vars
stop_words = set(stopwords.words('english'))
vectorizer = TfidfVectorizer(stop_words=stop_words)

# Define x and y columns
x = df.Title + ' ' + df.Review # concatenate Title and Review columns (for␣
 ↪vectorizer)
y = df.Type

# Split into train/test
x_train, x_test, y_train, y_test = tts(x, y, test_size=0.2, train_size=0.8,␣
 ↪random_state=66)

# Print shapes
print('x shape:', x_train.shape)
print('y shape:', y_train.shape)

# Apply tfidf vectorizer to features
x_train = vectorizer.fit_transform(x_train)
x_test = vectorizer.transform(x_test)
```

```
# Print snapshot of the vectorized features
print("vectorized train size:", x_train.shape)
print("vectorized test size:", x_test.shape)
```

```
x shape: (160000,)
y shape: (160000,)
vectorized train size: (160000, 146029)
vectorized test size: (40000, 146029)
```

5 Training The Models

For the Machine Learning models, we'll train Naive Bayes, Logistic Regression, and Neural Network models, making two attempts at each. The first attempt will be a simple version of the model, while the second attempt will be my attempt at an improved version of the simple model. Any things I tried that didn't make it into the final version of the second attempt will be noted in my explanation of the model.

5.1 Naive Bayes (First Attempt)

In this attempt, I'll create a simple Naive Bayes model using Multinomial Naive Bayes, and examine its performance from there.

5.1.1 Training

[65]:
```
# Create Naive Bayes model & fit it to the training data
nb1 = MultinomialNB()
nb1.fit(x_train, y_train)

# Calculate priors
prior_p = sum(y_train == 1) / len(y_train)
log_prior_p = math.log(prior_p)
print('prior spam:', prior_p, '\n')
print('log of prior:', log_prior_p)

print('The prior above should match the following model prior:', nb1.
 ↪class_log_prior_[1])
```

```
prior spam: 0.5007

log of prior: -0.6917481596462379
The prior above should match the following model prior: -0.691748159646238
```

5.1.2 Evaluation

[66]:
```
# Predict off the test data
pred_nb1 = nb1.predict(x_test)

# Print accuracy report
print(confusion_matrix(y_test, pred_nb1))
print()
print('Accuracy:\t\t\t\t', accuracy_score(y_test, pred_nb1))
print()
print('Precision (positive):\t', precision_score(y_test, pred_nb1, pos_label=1))
print('Precision (negative):\t', precision_score(y_test, pred_nb1, pos_label=0))
print('Precision (average):\t', (precision_score(y_test, pred_nb1,␣
 ↪pos_label=1)+precision_score(y_test, pred_nb1, pos_label=0))/2)
print()
print('Recall (positive):\t\t', recall_score(y_test, pred_nb1, pos_label=1))
print('Recall (negative):\t\t', recall_score(y_test, pred_nb1, pos_label=0))
print('Recall (average):\t\t', (recall_score(y_test, pred_nb1,␣
 ↪pos_label=1)+recall_score(y_test, pred_nb1, pos_label=0))/2)
print()
print('F1 Score:\t\t\t\t', f1_score(y_test, pred_nb1))
print()
print('First 10 Mis-classifications (out of ' + str(len(y_test[y_test !=␣
 ↪pred_nb1])) + '):')
#print(y_test[y_test != pred_nb1].iloc[:10])
for i in y_test[y_test!= pred_nb1].iloc[:10].index:
    print(i)
    print("Title:", df.loc[i].Title)
    print('Review:', df.loc[i].Review)
    print()
```

```
[[17417  2695]
 [ 3242 16646]]

Accuracy:                               0.851575

Precision (positive):    0.8606587043069128
Precision (negative):    0.8430708165932523
Precision (average):     0.8518647604500826

Recall (positive):               0.8369871279163315
Recall (negative):               0.8660003977724742
Recall (average):                0.8514937628444028

F1 Score:                               0.8486578806495196

First 10 Mis-classifications (out of 5937):
162196
```

Title: What's Really THAT bad about this movie?
Review: I don't know why people are saying that this is such a horrible movie. It wasn't that bad, I guess it was a little far fetched, but look at some other big movies these days. It's a horror film, their usually all the same, and the killer never dies. Look at Halloween for example, will he ever die? And these killers are already dead! It was scary and when I see it, it still is sometimes scary. I thought it deserves 3 and a half stars, but the rating doesn't have that, so I gave it 4.

145969
Title: A GREAT Portable printer
Review: I realllly wanted to give it 4 1/2 stars, but truly can't. I would have liked a LCD screen, and I feel it needs a few more editing capabilities while standalone. While I wish it could be battery, or electrically operated, I won't hold those things against it.I set mine up to print pictures I needed quickly that had been loaded to my computer, and deleted from my cards. The lil workhorse printed them all beautifully. I love it.I then printed a couple more just to see how well it printed. When my husband saw them, he stated, oh these were not taken by your digital. I said oh yes they were.If it had a few more editing capabilites, and the screen it would be perfect.

140039
Title: This book introduced me to this series!
Review: I am amazed. I love this book, and I went and bought the rest of the series a week later! I am a big fan of Laurell K Hamilton, though I admit her latest books have to much sex in them. Though the Dark Hunter series is in the romance section at the book store, it has less sex and the same amount of action as LKH's books that are in the scifi/ficiton section. I love the witty humor, and the guys of the DH series. I am a dedicated fan and can not wait until her new book comes out. Definately a worth while read!

180149
Title: The Wicca Spellbook
Review: I think Gerina Dunwich is a great author of Wicca. She knows all about different love spells, luck, and other spells. I highly recommend this book and her other books.

66282
Title: What the heck was this?!!!!!!!!
Review: I have seen many Bronson movies like Chino, The Great Escape, and The Magnificent Seven, and The Death Wish Series. Death Wish 5 was the WORST movie that I have seen that starred Bronson in a minor or major role. Nothing in this movie was believable, especially the acting. The death scenes and gun shots were extremely exaggerated. If this movie was a comedy, I would have given it a 5.

184808
Title: Audio cassette to CD
Review: I bought this to convert old cassette recordings to audio format on my

computer and from there I create audio CDs. This product is simple to setup and use. There are a lot of features, I'm still discovereing. I would give this 5 stars except I have not finished learning about it. After 6 month of use, I found that the software is very good. I do connect a cassette player to my labtop via 1/8" audio wire, run the program, record the cassette recording into the hard driv then burn it on CD.

54860
Title: This is comforting escapism?
Review: I enjoy a good romance novel as much as the next person and I don't think one need feel apologetic about reading them. I've read several of Spencer's books and have enjoyed them. However, I found this one intolerable. How can I feel the sense of escape, comfort and romance that I hope for when reading this kind of book when the major plot turns upon an extramarital affair? Like some other reviewers, I thought the portrayals of Nancy and Katy were disturbing; Nancy is depicted as such a bitter cold shrew (because she likes her job and has the self-awareness to know she wouldn't be a good parent) that she "deserves" to lose her husband. Katy is a selfish, rigid and unloving brat because she believes in obscure moral tenets such as "don't sleep with married men."The novel does feature lovely descriptions of Door Country and some engaging humorous moments. Overall, though, I can't recommend this book at all.

98127
Title: Product received is as shown in the picture.
Review: Cushion has one third thickness of what shown in the picture. It is not easy to adjust height. You need your own tool unscrew the screw. Do not plan to share with yuor family member who needs seat adjustment on the fly.

18308
Title: Less than I expected.
Review: I received The Gershwin Songbook, "'Swonderful." My complaint has nothing to do with Amazon or the supplier. I'm just a traditionalist, and found the music and singing to be less than satisfying. As an arranger or singer, how could you possibly think your sense of rhythm or your lyrical ability could exceed that of the master? I'm here to tell you that neither of yours did. Just play the music and sing the lyrics as they were written so the listeners can sing along, snap their fingers and reminisce about happier times.

18512
Title: Cover Art Is Nice
Review: This book is just not very original. In fact, after reading it, I got so hungry for the dragons of Pern that I am rereading that whole series (and all the offshoots). Don't bother with Eragon. Read Anne McCaffrey's Dragons of Pern series and Tolkien's Lord of the Rings instead; both of these series are far, far more intriguing and imaginative.

As we can see, the algorithm achieved an 85% accuracy, average precision, average recall, and

(roughly) F1 score. It seems to have a relatively balanced false positive/false negative rate, as well as a true positive/true negative rate.

5.2 Naive Bayes (Second Attempt)

With this attempt, I wanted to do my best to improve the Precision/Recall scores of the model, without affecting the Accuracy too negatively.

5.2.1 Training

[67]:
```
# Create Naive Bayes model & fit it to the training data
nb2 = BernoulliNB()
nb2.fit(x_train, y_train)

# Calculate priors
prior_p = sum(y_train == 1) / len(y_train)
log_prior_p = math.log(prior_p)
print('prior spam:', prior_p, '\n')
print('log of prior:', log_prior_p)

print('The prior above should match the following model prior:', nb1.
 ↪class_log_prior_[1])
```

```
prior spam: 0.5007

log of prior: -0.6917481596462379
The prior above should match the following model prior: -0.691748159646238
```

5.2.2 Evaluation

[68]:
```
# Predict off the test data
pred_nb2 = nb2.predict(x_test)

# Print accuracy report
print(confusion_matrix(y_test, pred_nb2))
print()
print('Accuracy:\t\t\t\t', accuracy_score(y_test, pred_nb2))
print()
print('Precision (positive):\t', precision_score(y_test, pred_nb2, pos_label=1))
print('Precision (negative):\t', precision_score(y_test, pred_nb2, pos_label=0))
print('Precision (average):\t', (precision_score(y_test, pred_nb2,␣
 ↪pos_label=1)+precision_score(y_test, pred_nb2, pos_label=0))/2)
print()
print('Recall (positive):\t\t', recall_score(y_test, pred_nb2, pos_label=1))
print('Recall (negative):\t\t', recall_score(y_test, pred_nb2, pos_label=0))
print('Recall (average):\t\t', (recall_score(y_test, pred_nb2,␣
 ↪pos_label=1)+recall_score(y_test, pred_nb2, pos_label=0))/2)
print()
```

```
print('F1 Score:\t\t\t\t', f1_score(y_test, pred_nb2))
print()
print('First 10 Mis-classifications (out of ' + str(len(y_test[y_test !=␣
 ↪pred_nb2])) + '):')
for i in y_test[y_test!= pred_nb2].iloc[:10].index:
    print(i)
    print("Title:", df.loc[i].Title)
    print('Review:', df.loc[i].Review)
    print()
```

```
[[17060  3052]
 [ 2697 17191]]

Accuracy:                                0.856275

Precision (positive):    0.8492318332263005
Precision (negative):    0.8634914207622615
Precision (average):     0.856361626994281

Recall (positive):               0.8643905872888173
Recall (negative):               0.8482498011137629
Recall (average):                0.8563201942012901

F1 Score:                                0.8567441628666118

First 10 Mis-classifications (out of 5749):
162196
Title: What's Really THAT bad about this movie?
Review: I don't know why people are saying that this is such a horrible movie.
It wasn't that bad, I guess it was a little far fetched, but look at some other
big movies these days. It's a horror film, their usually all the same, and the
killer never dies. Look at Halloween for example, will he ever die? And these
killers are already dead! It was scary and when I see it, it still is sometimes
scary. I thought it deserves 3 and a half stars, but the rating doesn't have
that, so I gave it 4.

140039
Title: This book introduced me to this series!
Review: I am amazed. I love this book, and I went and bought the rest of the
series a week later! I am a big fan of Laurell K Hamilton, though I admit her
latest books have to much sex in them. Though the Dark Hunter series is in the
romance section at the book store, it has less sex and the same amount of action
as LKH's books that are in the scifi/ficiton section. I love the witty humor,
and the guys of the DH series. I am a dedicated fan and can not wait until her
new book comes out. Definately a worth while read!

20681
Title: dated examples
```

Review: This book is old and shows it age. There are very few examples, and those in the book are over a decade old! A lot of changes have happened in strategy in a decade,… Avoid this book if possible.

77948
Title: Long read, little pay-off
Review: I wouldn't be surprised if, in the next installment, Roland actually finds a motorcycle and jumps a shark with it.

21139
Title: Stick with the computer version
Review: I am a Simaddict.I know what it is like to stay up until three or four in the morning downloading the perfect skins, creating children, and attempting to get abducted.Bottom line: no Playstation version of the Sims comes remotely close to the addictive fun of the computer version.The freedom is gone.In Computer Sims, part of the fun is that you are like the SIM'S G-d but in this you are so very limited.I don't like itI don't recommend it.If it's a gift…eBay it, or regift it to someone you are not so fond of.

184808
Title: Audio cassette to CD
Review: I bought this to convert old cassette recordings to audio format on my computer and from there I create audio CDs. This product is simple to setup and use. There are a lot of features, I'm still discovereing. I would give this 5 stars except I have not finished learning about it. After 6 month of use, I found that the software is very good. I do connect a cassette player to my labtop via 1/8" audio wire, run the program, record the cassette recording into the hard driv then burn it on CD.

56124
Title: A Let Down Read
Review: When a kid is smarter than the adults fiction is in trouble. Kidnapping and child abuse are tough subjects, but a Judge who doesn't notify the police and a mother who isn't hysterical are hard to swallow. I've enjoyed Coulter's stories in the past but THE TARGET never worked for this reader.Savich and Sherlock do stop by for an encore, but without their normal punch. If you are a fan of her writing you may enjoy it, but for others pass this one by, Ms. Coulter can construct a better story.Nash Black, author of TRAVELERS and SINS OF THE FATHERS.

54860
Title: This is comforting escapism?
Review: I enjoy a good romance novel as much as the next person and I don't think one need feel apologetic about reading them. I've read several of Spencer's books and have enjoyed them. However, I found this one intolerable. How can I feel the sense of escape, comfort and romance that I hope for when reading this kind of book when the major plot turns upon an extramarital affair? Like some other reviewers, I thought the portrayals of Nancy and Katy were

```
disturbing; Nancy is depicted as such a bitter cold shrew (because she likes her
job and has the self-awareness to know she wouldn't be a good parent) that she
"deserves" to lose her husband. Katy is a selfish, rigid and unloving brat
because she believes in obscure moral tenets such as "don't sleep with married
men."The novel does feature lovely descriptions of Door Country and some
engaging humorous moments. Overall, though, I can't recommend this book at all.

39585
Title: lacking knowledge of subject matter
Review: better information on handwriting analysis is avalable from other
sources.

98127
Title: Product received is as shown in the picture.
Review: Cushion has one third thickness of what shown in the picture. It is not
easy to adjust height. You need your own tool unscrew the screw. Do not plan to
share with yuor family member who needs seat adjustment on the fly.
```

I'd at first started by having the algorithm identify all-caps words, triple (or more) dots, as well as consecutive exclamation points/question marks. This resulted in an overall worse performance in all instances that I'd included this (and with all combinations of the three). So, I scrapped the idea and instead switched the algorithm to use the Bernoulli (binomial) variant of Naive Bayes. This resulted in a slightly better overall performance than my original model, but only by a small percentage.

I believe the model performed worst after identifying those aforementioned sequences due to the fact that they aren't necessarily telling of the sentiment of the message. For instance, both positive and negative reviews may include triple dots (or more), as well as some number of capital words. However, one thing I should have tried (in reflection) was separately checking for consecutive exclamation points and consecutive question marks, rather than checking for any combination of both. Mainly because, I'd imagine, question marks would be used more often in negative reviews, and exclamation points would be used more often in positive reviews (albeit still in negative reviews). Combining both makes the algorithm ignorant of context. Or at least, that's what I speculate.

5.3 Logistic Regression (First Attempt)

In this attempt, I'll create a simple Logistic Regression model, and examine its performance from there.

5.3.1 Training

[69]:
```
# Create Logistic Regression model & fit it to the training data
lr1 = LogisticRegression(solver='lbfgs', class_weight='balanced',␣
 ↪random_state=66)
lr1.fit(x_train, y_train)
```

[69]:
```
LogisticRegression(class_weight='balanced', random_state=66)
```

5.3.2 Evaluation

[70]:
```
# Predict off of the test data
pred_lr1 = lr1.predict(x_test)

# Print accuracy report
print(confusion_matrix(y_test, pred_lr1))
print()
print('Accuracy:\t\t\t\t', accuracy_score(y_test, pred_lr1))
print()
print('Precision (positive):\t', precision_score(y_test, pred_lr1, pos_label=1))
print('Precision (negative):\t', precision_score(y_test, pred_lr1, pos_label=0))
print('Precision (average):\t', (precision_score(y_test, pred_lr1,␣
 ↪pos_label=1)+precision_score(y_test, pred_lr1, pos_label=0))/2)
print()
print('Recall (positive):\t\t', recall_score(y_test, pred_lr1, pos_label=1))
print('Recall (negative):\t\t', recall_score(y_test, pred_lr1, pos_label=0))
print('Recall (average):\t\t', (recall_score(y_test, pred_lr1,␣
 ↪pos_label=1)+recall_score(y_test, pred_lr1, pos_label=0))/2)
print()
print('F1 Score:\t\t\t\t', f1_score(y_test, pred_lr1))
```

```
[[17846  2266]
 [ 2122 17766]]

Accuracy:                               0.8903

Precision (positive):    0.8868809904153354
Precision (negative):    0.893729967948718
Precision (average):     0.8903054791820266

Recall (positive):               0.8933024939662108
Recall (negative):               0.8873309466984884
Recall (average):                0.8903167203323497

F1 Score:                               0.8900801603206413
```

As you can see, the Logistic Regression model performed notably better than both versions of the Naive Bayes model, across all metrics. I believe this is because of the gradient descent that occurs in the background when training this model, as that aims to make the weights of the model more accurate.

5.4 Logistic Regression (Second Attempt)

In this attempt, we'll change of things about the previous model, aiming for across-the-board improvements yet again. Above all else, though, we'd like to see an improvement to our F1 score without sacrificing very much accuracy. The changes I opted to make for this attempt were to use bigrams (instead of unigrams), and changing the solver to liblinear. From research I'd done online, liblinear seems best fitted for a problem such as this one, and should hopefully boost results.

5.5 Further Preprocessing

[90]:
```
# Create new vectorizer
vectorizer2 = TfidfVectorizer(stop_words=stop_words, ngram_range=(2,2))

# Copy df, recreate vars
df2 = df.copy()

# Define x and y columns
x = df2.Title + ' ' + df2.Review # concatenate Title and Review columns (for␣
 ↪vectorizer)
y = df2.Type

# Split into train/test
x_train, x_test, y_train, y_test = tts(x, y, test_size=0.2, train_size=0.8,␣
 ↪random_state=66)

# Print shapes
print('x shape:', x_train.shape)
print('y shape:', y_train.shape)

# Apply new tfidf vectorizer to features
x_train = vectorizer2.fit_transform(x_train)
x_test = vectorizer2.transform(x_test)

# Print snapshot of the vectorized features
print("vectorized train size:", x_train.shape)
print("vectorized test size:", x_test.shape)
```

```
x shape: (160000,)
y shape: (160000,)
vectorized train size: (160000, 3185687)
vectorized test size: (40000, 3185687)
```

5.5.1 Training

[91]:
```
# Create Logistic Regression model & fit it to the training data
lr2 = LogisticRegression(solver='liblinear', class_weight='balanced',␣
 ↪random_state=66)
lr2.fit(x_train, y_train)
```

[91]:
```
LogisticRegression(class_weight='balanced', random_state=66, solver='liblinear')
```

5.5.2 Evaluation

[92]:
```
# Predict off of the test data
pred_lr2 = lr2.predict(x_test)

# Print accuracy report
print(confusion_matrix(y_test, pred_lr2))
print()
print('Accuracy:\t\t\t\t', accuracy_score(y_test, pred_lr2))
print()
print('Precision (positive):\t', precision_score(y_test, pred_lr2, pos_label=1))
print('Precision (negative):\t', precision_score(y_test, pred_lr2, pos_label=0))
print('Precision (average):\t', (precision_score(y_test, pred_lr2,␣
 ↪pos_label=1)+precision_score(y_test, pred_lr2, pos_label=0))/2)
print()
print('Recall (positive):\t\t', recall_score(y_test, pred_lr2, pos_label=1))
print('Recall (negative):\t\t', recall_score(y_test, pred_lr2, pos_label=0))
print('Recall (average):\t\t', (recall_score(y_test, pred_lr2,␣
 ↪pos_label=1)+recall_score(y_test, pred_lr2, pos_label=0))/2)
print()
print('F1 Score:\t\t\t\t', f1_score(y_test, pred_lr2))
```

```
[[17159  2953]
 [ 2718 17170]]

Accuracy:                               0.858225

Precision (positive):    0.8532524971425732
Precision (negative):    0.8632590431151582
Precision (average):     0.8582557701288658

Recall (positive):              0.8633346741753821
Recall (negative):              0.8531722354813047
Recall (average):               0.8582534548283434

F1 Score:                               0.8582639774062133
```

As you can see, the results of this attempt were worse than my original attempt. I'd tried a number of things to try to improve the algorithm's results while keeping the use of bigrams the same, but results only plummeted. What is interesting to me is that, using trigrams resulted in a 12% decrease in each statistic. It seems that for classifying for the purpose of sentiment analysis, sticking to unigrams will give better results. At least, based off of my findings.

5.6 Neural Network (First Attempt)

Lastly, we'll build a simple neural network, as well as a more improved version of it. Neural networks use hidden layers to let the learning be done through multiple functions, so I anticipate this model having the best performance out of all of them.

6 Preprocessing (back to first)

[94]:
```
# Define x and y columns
x = df.Title + ' ' + df.Review # concatenate Title and Review columns (for␣
 ↪vectorizer)
y = df.Type

# Split into train/test
x_train, x_test, y_train, y_test = tts(x, y, test_size=0.2, train_size=0.8,␣
 ↪random_state=66)

# Print shapes
print('x shape:', x_train.shape)
print('y shape:', y_train.shape)

# Apply tfidf vectorizer to features
x_train = vectorizer.fit_transform(x_train)
x_test = vectorizer.transform(x_test)

# Print snapshot of the vectorized features
print("vectorized train size:", x_train.shape)
print("vectorized test size:", x_test.shape)
```

```
x shape: (160000,)
y shape: (160000,)
vectorized train size: (160000, 146029)
vectorized test size: (40000, 146029)
```

6.0.1 Training

[110]:
```
# Create Neural Network & fit it to the training data
nn1 = MLPClassifier(solver='lbfgs', activation='relu', alpha=1e-5,␣
 ↪hidden_layer_sizes=(15,2), random_state=66, max_iter=200, verbose=True)
nn1.fit(x_train, y_train)
```

```
C:\Users\Justi\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:559:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```

[110]:
```
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(30, 2), random_state=66,
              solver='lbfgs', verbose=True)
```

6.0.2 Evaluation

[111]:
```
# Predict off of the test data
pred_nn1 = nn1.predict(x_test)

# Print accuracy report
print(confusion_matrix(y_test, pred_nn1))
print()
print('Accuracy:\t\t\t\t', accuracy_score(y_test, pred_nn1))
print()
print('Precision (positive):\t', precision_score(y_test, pred_nn1, pos_label=1))
print('Precision (negative):\t', precision_score(y_test, pred_nn1, pos_label=0))
print('Precision (average):\t', (precision_score(y_test, pred_nn1,␣
 ↪pos_label=1)+precision_score(y_test, pred_nn1, pos_label=0))/2)
print()
print('Recall (positive):\t\t', recall_score(y_test, pred_nn1, pos_label=1))
print('Recall (negative):\t\t', recall_score(y_test, pred_nn1, pos_label=0))
print('Recall (average):\t\t', (recall_score(y_test, pred_nn1,␣
 ↪pos_label=1)+recall_score(y_test, pred_nn1, pos_label=0))/2)
print()
print('F1 Score:\t\t\t\t', f1_score(y_test, pred_nn1))
```

```
[[17708  2404]
 [ 2381 17507]]

Accuracy:                               0.880375

Precision (positive):    0.879262719099995
Precision (negative):    0.8814774254567176
Precision (average):     0.8803700722783563

Recall (positive):               0.8802795655671762
Recall (negative):               0.8804693715194909
Recall (average):                0.8803744685433336

F1 Score:                               0.8797708485137817
```

As we can observe from the results, this simple Neural Network performed better than the Naive Bayes models, but worse than the first Logistic Regression model.

6.1 Neural Network (Second Attempt)

In next and final attempt overall, we'll play around with hidden layer sizes, and try using the adam solver as well as the logistic activation function.

6.1.1 Training

[161]:
```
# Create Neural Network & fit it to the training data
nn2 = MLPClassifier(solver='adam', activation='logistic', alpha=1e-5,␣
 ↪hidden_layer_sizes=(5,2), random_state=66, verbose=True, max_iter=10)
nn2.fit(x_train, y_train)
```

```
Iteration 1, loss = 0.72425351
Iteration 2, loss = 0.67560853
Iteration 3, loss = 0.51121314
Iteration 4, loss = 0.32580015
Iteration 5, loss = 0.26032270
Iteration 6, loss = 0.22849924
Iteration 7, loss = 0.20786246
Iteration 8, loss = 0.19225683
Iteration 9, loss = 0.17941170
Iteration 10, loss = 0.16836061
```

```
C:\Users\Justi\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:702:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

[161]:
```
MLPClassifier(activation='logistic', alpha=1e-05, hidden_layer_sizes=(5, 2),
              max_iter=10, random_state=66, verbose=True)
```

6.1.2 Evaluation

[162]:
```
# Predict off of the test data
pred_nn2 = nn2.predict(x_test)

# Print accuracy report
print(confusion_matrix(y_test, pred_nn2))
print()
print('Accuracy:\t\t\t\t', accuracy_score(y_test, pred_nn2))
print()
print('Precision (positive):\t', precision_score(y_test, pred_nn2, pos_label=1))
print('Precision (negative):\t', precision_score(y_test, pred_nn2, pos_label=0))
print('Precision (average):\t', (precision_score(y_test, pred_nn2,␣
 ↪pos_label=1)+precision_score(y_test, pred_nn2, pos_label=0))/2)
print()
print('Recall (positive):\t\t', recall_score(y_test, pred_nn2, pos_label=1))
print('Recall (negative):\t\t', recall_score(y_test, pred_nn2, pos_label=0))
print('Recall (average):\t\t', (recall_score(y_test, pred_nn2,␣
 ↪pos_label=1)+recall_score(y_test, pred_nn2, pos_label=0))/2)
print()
print('F1 Score:\t\t\t\t', f1_score(y_test, pred_nn2))
```

```
[[17872  2240]
 [ 2102 17786]]

Accuracy:                                0.89145

Precision (positive):    0.8881454109657445
Precision (negative):    0.8947631921497947
Precision (average):     0.8914543015577696

Recall (positive):               0.8943081255028158
Recall (negative):               0.888623707239459
Recall (average):                0.8914659163711374

F1 Score:                                0.89121611464649
```

Looking at the results of this Neural Network, we see a small improvement from the previous one, of about an additive 1% to each statistic. In some instances of training this neural network with this layer size, I was able to achieve both an accuracy and F1 score above 90%.

I noticed that using a simpler layer sizing yielded the best results, although that may be the case as I only ran the training for 10 iterations (specifically for reasons related to time consumption. More trial and error could probably be done to get better results via better layering.

7 Conclusion

In conclusion, we can observe the Neural Network and Logistic Regression models performing the best. The difference between both models is rather small, but given the potential for growth that Neural Networks have, I'd say the Neural Network model would perform better on this data in the long run, especially given the large size of it data.

While Naive Bayes performed the worst, it was very quick and effective, and yielded decent results. Perhaps if the data set were smaller, the algorithm would have performed even better, though.

Upon reflection, I've realized that I could probably have improved performance for each model by performing sentence negation on each sentence in both the Title and Review. Such that a phrase like 'This is not beautiful' would become 'This is ugly'. As we'd talked about in class, the sentiment when a negating term such as 'not' or something that ends in "n't" is used, it can be easy to misinterpret a negative sentiment as a positive sentiment, and vice versa. Negating the sentences could help alleviate this issue, and possibly improve results. Of course, further preprocessing on the data would need to be done, and tokenizing would need to be done on my end before transforming it via the vectorizer. If I were to explore a similar task, I would definitely implement sentence negation to (hopefully) improve results.