NLP Project - Yelp Reviews

August 10, 2020

1 NLP Classificaion - Prediction of Yelp Reviews

For this NLP Classification project I will be using **Yelp Reviews** dataset from Kaggle.

Project goal:

Predict Movie Ratings on text data from Yelp Reviews using various Machine Learning algorythms.

1. We will use such models as Naive Bayes, Random Forest, SVM, XGBoost and Multilayer Perceptron.
2. Also we will preprocess text information using Lemmatization and Porter Stemming to find out which one is more suitable for the task.
3. We will use 2 types of converting preprocessed text data: Bag of Words Vectorization and TF-IDF.

! Scroll down to the bottom of the document to see model evaluation results.

The Yelp Reviews dataset contains the following fields:

- business_id - Unique Business ID
- date - Date of Review
- review_id - Review ID
- stars - Stars given by the user
- text - Review given by the user
- type - Type of text entered - Review
- user_id - Unique User ID
- cool - The number of cool votes the review received
- useful - The number of useful votes the review received
- funny - The number of funny votes the review received

1.1 Importing Libraries and Loading Data

[286]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import nltk
from sklearn.feature_extraction.text import TfidfTransformer
from nltk.corpus import stopwords
```

```python
from nltk import PorterStemmer,WordNetLemmatizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score
from sklearn.feature_extraction.text import TfidfTransformer, CountVectorizer
from sklearn.model_selection import GridSearchCV
from wordcloud import WordCloud, ImageColorGenerator
from PIL import Image
import urllib
import requests
import re
%matplotlib inline
```

[295]:
```python
# Loading Yelp Reviews dataset
path = r'D:\Professional\_My Projects\NLP Project - Predicting Yelp Reviews␣
 ↪Rating'
path_to_file = (path + r'\yelp.csv')
df = pd.read_csv(path_to_file)
```

[296]:
```python
df.head()
```

[296]:
```
              business_id        date               review_id  stars  \
0  9yKzy9PApeiPPOUJEtnvkg  2011-01-26  fWKvX83p0-ka4JS3dc6E5A      5
1  ZRJwVLyzEJq1VAihDhYiow  2011-07-27  IjZ33sJrzXqU-0X6U8NwyA      5
2  6oRAC4uyJCsJl1X0WZpVSA  2012-06-14  IESLBzqUCLdSzSqm0eCSxQ      4
3  _1QQZuf4zZOyFCvXc0o6Vg  2010-05-27  G-WvGaISbqqaMHlNnByodA      5
4  6ozycU1RpktNG2-1BroVtw  2012-01-05  1uJFq2r5QfJG_6ExMRCaGw      5

                                                text    type  \
0  My wife took me here on my birthday for breakf...  review
1  I have no idea why some people give bad review...  review
2  love the gyro plate. Rice is so good and I als...  review
3  Rosie, Dakota, and I LOVE Chaparral Dog Park!!...  review
4  General Manager Scott Petello is a good egg!!!...  review

                  user_id  cool  useful  funny
0  rLtl8ZkDX5vH5nAx9C3q5Q     2       5      0
1  0a2KyEL0d3Yb1V6aivbIuQ     0       0      0
2  0hT2KtfLiobPvh6cDC8JQg     0       1      0
3  uZetl9T0NcROGOyFfughhg     1       2      0
4  vYmM4KTsC8ZfQBg-j5MWkw     0       0      0
```

1.2 Exploratory Data Analysis

Create a new column called "text length" which is the number of words in the text column. Also for our classification task we don't need columns like 'business_id', 'review_id', 'user_id' and 'type'.

[29]:
```
df['text_length'] = df['text'].apply(len)
df.drop(["business_id", "review_id", "user_id", "type"], axis=1, inplace=True)
```

[30]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 7 columns):
date           10000 non-null object
stars          10000 non-null int64
text           10000 non-null object
cool           10000 non-null int64
useful         10000 non-null int64
funny          10000 non-null int64
text_length    10000 non-null int64
dtypes: int64(5), object(2)
memory usage: 547.0+ KB
```

[31]:
```
df.describe()
```

[31]:

	stars	cool	useful	funny	text_length
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	3.777500	0.876800	1.409300	0.701300	710.738700
std	1.214636	2.067861	2.336647	1.907942	617.399827
min	1.000000	0.000000	0.000000	0.000000	1.000000
25%	3.000000	0.000000	0.000000	0.000000	294.000000
50%	4.000000	0.000000	1.000000	0.000000	541.500000
75%	5.000000	1.000000	2.000000	1.000000	930.000000
max	5.000000	77.000000	76.000000	57.000000	4997.000000

First, let's check are there any Null values in the dataset.

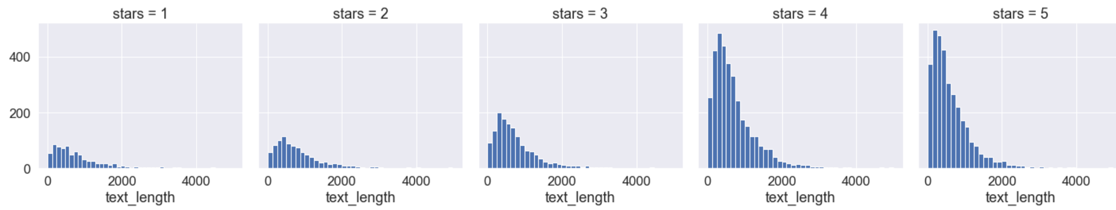
[32]:
```
round((df.isna().sum() / df.count()) * 100, 1)
```

[32]:
```
date           0.0
stars          0.0
text           0.0
cool           0.0
useful         0.0
funny          0.0
text_length    0.0
dtype: float64
```

There are **no Null values** in our dataset.

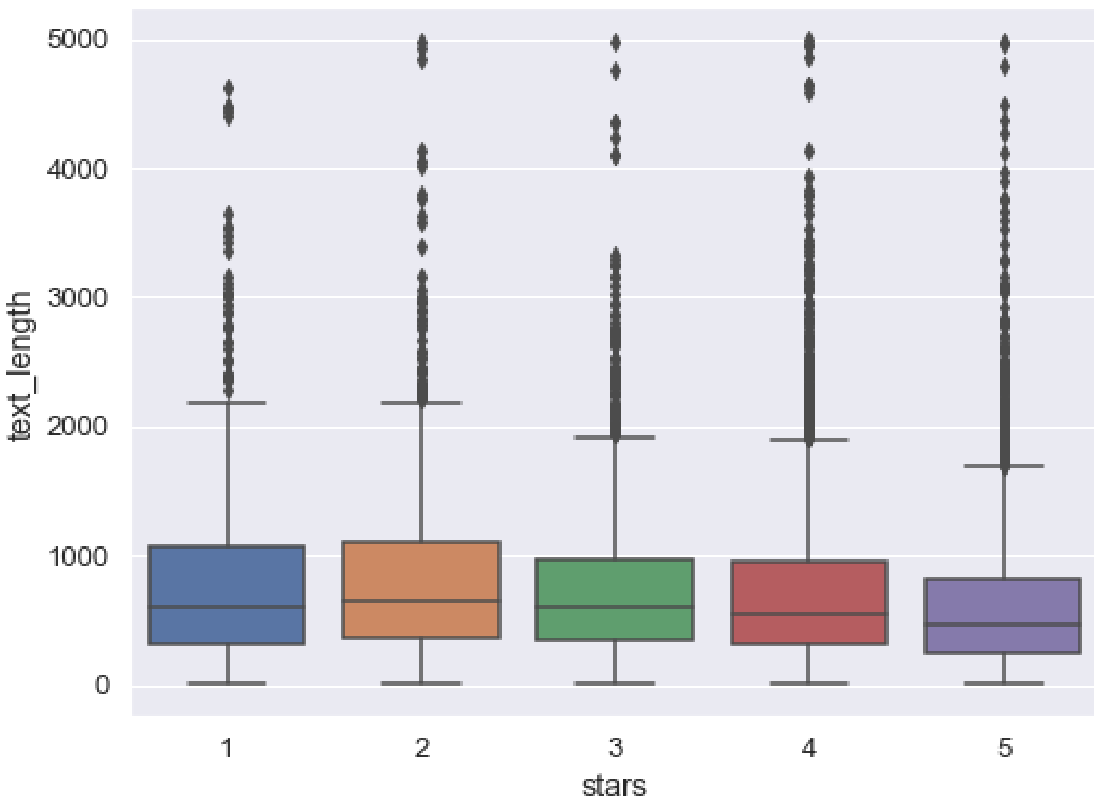
[33]:
```
sns.set(style = 'darkgrid', font_scale = 1.5)
g = sns.FacetGrid(df, col = 'stars', height = 4, aspect = 1)
g.map(plt.hist, 'text_length', bins = 40)
```

[33]:
```
<seaborn.axisgrid.FacetGrid at 0x2a8bef8a390>
```



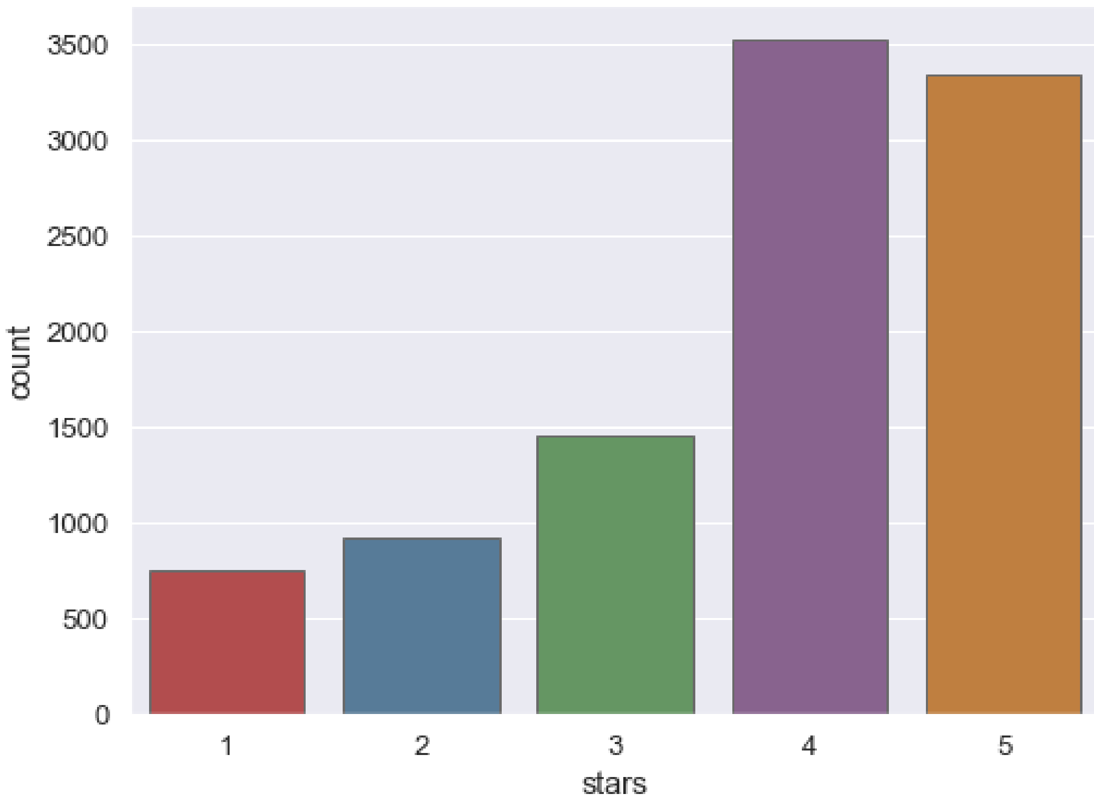
[34]:
```
sns.set(style = 'darkgrid', font_scale = 1.2, rc = {'figure.figsize':(8,6)})
sns.boxplot(data = df, x = 'stars', y = 'text_length')
```

[34]: <matplotlib.axes._subplots.AxesSubplot at 0x2a8c51add68>



[35]:
```
sns.countplot(data = df, x = 'stars', palette = 'Set1', lw = 1, edgecolor =␣
↪'dimgray', saturation = 0.5)
```

[35]: <matplotlib.axes._subplots.AxesSubplot at 0x2a8c548ab00>



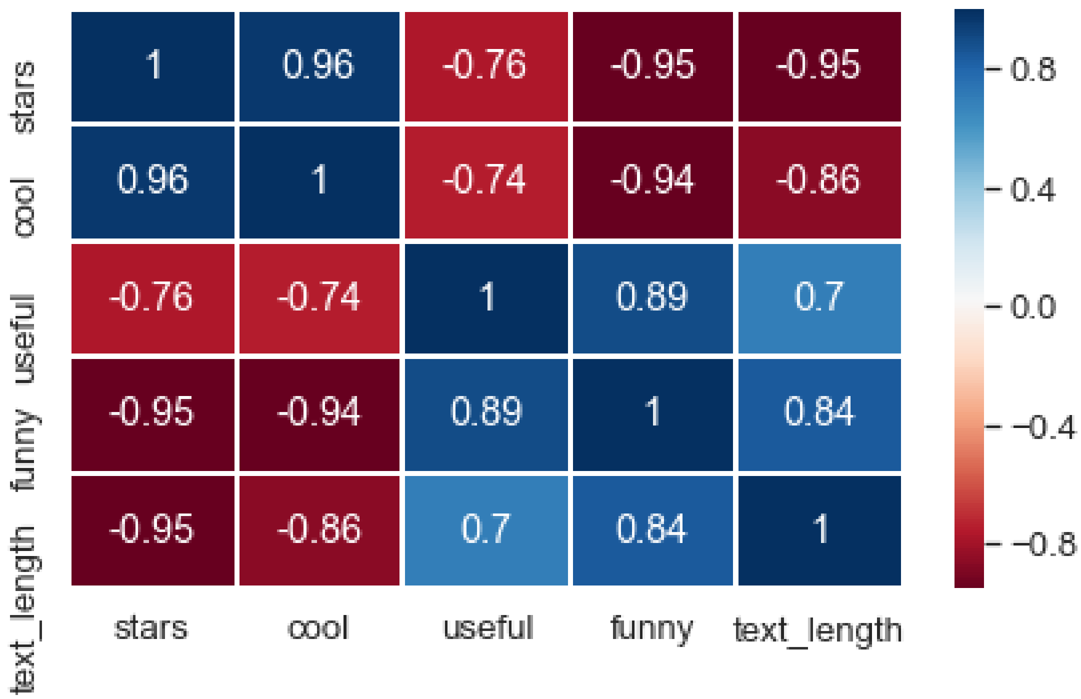
[36]:
```
df.groupby('stars').mean().reset_index()
```

[36]:

	stars	cool	useful	funny	text_length
0	1	0.576769	1.604806	1.056075	826.515354
1	2	0.719525	1.563107	0.875944	842.256742
2	3	0.788501	1.306639	0.694730	758.498289
3	4	0.954623	1.395916	0.670448	712.923142
4	5	0.944261	1.381780	0.608631	624.999101

It seems that 'stars' column is negatively correlated with 'text_length'. Also the number of cool, useful and funny votes is also correlated with stars. Let's check it.

[38]:
```
corr_mat = df.groupby('stars').mean().reset_index().corr()
fig, ax = plt.subplots(figsize = (7,4))
fig = sns.heatmap(corr_mat, linecolor = 'white', lw = 1, cmap = 'RdBu', annot =␣
 ↪corr_mat)
```



All 4 variables have strong correlation with 'stars' rating.

1.3 Text Preprocessing

First we need to preprocess reviews data to remove punctuation, non-letters, stopwords and implement Lemmatization/Stemming. For this purpose we define custom fuctions: **stem_preprocess** and **lemm_preprocess**.

[39]:
```
import re
from nltk.corpus import stopwords
from nltk import PorterStemmer,WordNetLemmatizer
```

[40]:
```
ps = PorterStemmer()
WL = WordNetLemmatizer()
```

[41]:
```
def stem_preprocess(raw_review):
    """
    Function to convert a raw review to a string of words
    The input is a single string (a raw movie review), and
    the output is a single string (a preprocessed movie review).
    The function performs the following steps:
    1. Remove non-letters
    2. Convert to lower case, split into individual words
    3. Convert stop words to a set - for speed purposes
    4. Remove stop words
    5. Implement word stemming using PorterStemmer
```

```
    6. Join the words into one string and return it
    """
    # Function to convert a raw review to a string of words
    # The input is a single string (a raw movie review), and
    # the output is a single string (a preprocessed movie review)
    #
    # 1. Remove non-letters
    letters_only = re.sub("[^a-zA-Z]", " ", raw_review)
    #
    # 2. Convert to lower case, split into individual words
    words = letters_only.lower().split()
    #
    # 3. In Python, searching a set is much faster than searching
    #   a list, so convert the stop words to a set
    stops = set(stopwords.words("english"))
    #
    # 4. Remove stop words
    meaningful_words = [w for w in words if not w in stops]
    #
    # 5. Word stemming
    stemmed_words = [ps.stem(i) for i in meaningful_words]
    #
    # 6. Join the words back into one string separated by space, and return the␣
↪result.
    return( " ".join( stemmed_words ))
```

[42]:
```
def lemm_preprocess(raw_review):
    """
    Function to convert a raw review to a string of words
    The input is a single string (a raw movie review), and
    the output is a single string (a preprocessed movie review).
    The function performs the following steps:
    1. Remove non-letters
    2. Convert to lower case, split into individual words
    3. Convert stop words to a set - for speed purposes
    4. Remove stop words
    5. Implement word Lemmatization using WordNetLemmatizer
    6. Join the words into one string and return it
    """
    # Function to convert a raw review to a string of words
    # The input is a single string (a raw movie review), and
    # the output is a single string (a preprocessed movie review)
    #
    # 1. Remove non-letters
    letters_only = re.sub("[^a-zA-Z]", " ", raw_review)
    #
    # 2. Convert to lower case, split into individual words
```

```
    words = letters_only.lower().split()
    #
    # 3. In Python, searching a set is much faster than searching
    #   a list, so convert the stop words to a set
    stops = set(stopwords.words("english"))
    #
    # 4. Remove stop words
    meaningful_words = [w for w in words if not w in stops]
    #
    # 5. Word lemmatizing
    lemmatized_words = [WL.lemmatize(i) for i in meaningful_words]
    #
    # 6. Join the words back into one string separated by space, and return the
↪result.
    return( " ".join( lemmatized_words ))
```

Let's take a look at the example of our text preprocessing.

[43]:
```
print('Normal text before preprocessing:')
print(df['text'][10])
print('\n')
print('Text after Stemming:')
print(stem_preprocess(df['text'][10]))
print('\n')
print('Text after Lemmatizing:')
print(lemm_preprocess(df['text'][10]))
```

```
Normal text before preprocessing:
The oldish man who owns the store is as sweet as can be.  Perhaps sweeter than
the cookies or ice cream.

Here's the lowdown: Giant ice cream cookie sandwiches for super cheap.  The
flavor permutations are basically endless.  I had snickerdoodle with cookies and
cream ice cream.  It was marvelous.



Text after Stemming:
oldish man own store sweet perhap sweeter cooki ice cream lowdown giant ice
cream cooki sandwich super cheap flavor permut basic endless snickerdoodl cooki
cream ice cream marvel



Text after Lemmatizing:
oldish man owns store sweet perhaps sweeter cooky ice cream lowdown giant ice
cream cookie sandwich super cheap flavor permutation basically endless
snickerdoodle cooky cream ice cream marvelous
```

Preprocess 'text' variable:

[44]:
```python
df['stemmed_text'] = df['text'].apply(stem_preprocess)
df['lemmatized_text'] = df['text'].apply(lemm_preprocess)
```

1.4 Word Clouds

Let's plot wordclouds from positive and negative reviews to look if those sets of words make sense.

[263]:
```python
df_pos = df[(df['stars'] == 5) | (df['stars'] == 4)]
df_neg = df[(df['stars'] == 1) | (df['stars'] == 2)]
```

[264]:
```python
plot_all_words = ' '.join(text for text in df['lemmatized_text'])
plot_pos_words = ' '.join(text for text in df_pos['lemmatized_text'])
plot_neg_words = ' '.join(text for text in df_neg['lemmatized_text'])
```

[195]:
```python
from wordcloud import WordCloud,ImageColorGenerator
from PIL import Image
import urllib
import requests
```

[280]:
```python
mask = np.array(Image.open('D:\\thumbs_up3.png'))
image_colors = ImageColorGenerator(mask)
wc = WordCloud(mask=mask, background_color="white",
               max_words=2000, max_font_size=150,
               random_state=42, width=mask.shape[1],
               height=mask.shape[0])
wc.generate(plot_pos_words)
plt.figure(figsize = (10,20))
plt.imshow(wc.recolor(color_func=image_colors), interpolation="bilinear")
plt.axis('off')
#plt.savefig("wordcloud_like.png", format="png")
plt.show()
```


```
[277]: mask = np.array(Image.open('D:\\thumbs_down3.png'))
image_colors = ImageColorGenerator(mask)
wc = WordCloud(mask=mask, background_color="black",
               max_words=2000, max_font_size=150,
               random_state=42, width=Mask.shape[1],
               height=Mask.shape[0])
wc.generate(plot_neg_words)
plt.figure(figsize = (10,20))
plt.imshow(wc.recolor(color_func=image_colors), interpolation="bilinear")
plt.axis('off')
plt.show()
```

1.5 Train Test Split

[127]:
```
X_stem = df['stemmed_text']
X_lemm = df['lemmatized_text']
y = df['stars']
```

[128]:
```
X_train_stem, X_test_stem, y_train, y_test = train_test_split(X_stem, y, test_size=0.3, random_state=101)
X_train_lemm, X_test_lemm, y_train, y_test = train_test_split(X_lemm, y, test_size=0.3, random_state=101)
```

[129]:
```
[[X_train_stem.shape, X_test_stem.shape],
 [X_train_lemm.shape, X_test_lemm.shape]]
```

[129]: [[(7000,), (3000,)], [(7000,), (3000,)]]

1.6 Vectorization

Initialize Bag of Words model to vectorize stemmed and lemmatized text data.

[130]:
```
# Initialize Vectorizer
bow_model_stem = CountVectorizer(max_features = 5000)
# Vectorizing stemmed text
X_train_stem = bow_model_stem.fit_transform(X_train_stem)
X_test_stem = bow_model_stem.transform(X_test_stem)
```

[131]:
```
# Initialize Vectorizer
bow_model_lemm = CountVectorizer(max_features = 5000)
# Vectorizing lemmatized text
X_train_lemm = bow_model_lemm.fit_transform(X_train_lemm)
X_test_lemm = bow_model_lemm.transform(X_test_lemm)
```

[132]:
```
[[X_train_stem.shape, X_test_stem.shape],
 [X_train_lemm.shape, X_test_lemm.shape]]
```

[132]: [[(7000, 5000), (3000, 5000)], [(7000, 5000), (3000, 5000)]]

1.7 Predicting Rating using Vectorized Data

1.7.1 Random Forest

[283]:
```
print("Training the random forest...")
from sklearn.ensemble import RandomForestClassifier

# Initialize a Random Forest classifier with 200 trees
forest_stem = RandomForestClassifier(n_estimators = 200)
forest_lemm = RandomForestClassifier(n_estimators = 200)

# Fit the forest to the training set, using the bag of words as features and␣
↪the sentiment labels as the response variable
forest_stem.fit(X_train_stem, y_train)
forest_lemm.fit(X_train_lemm, y_train)
print("Random Forest has been trained")
```

```
Training the random forest...
Random Forest has been trained
```

Now let's use our model to predict ratings for test_data_features

[284]:
```
pred_stem = forest_stem.predict(X_test_stem)
pred_lemm = forest_lemm.predict(X_test_lemm)
```

[285]:
```
print('Model Results:')
print('\n')
print('Using Stemmed Data:')
```

```python
print('\n')
print(classification_report(y_test, pred_stem))
print('\n')
print('Using Lemmatized Data:')
print('\n')
print(classification_report(y_test, pred_lemm))
```

```
Model Results:


Using Stemmed Data:


              precision    recall  f1-score   support

           1       0.75      0.26      0.39       220
           2       0.31      0.03      0.05       273
           3       0.38      0.04      0.07       443
           4       0.42      0.71      0.53      1064
           5       0.56      0.58      0.57      1000

    accuracy                           0.47      3000
   macro avg       0.48      0.32      0.32      3000
weighted avg       0.47      0.47      0.42      3000



Using Lemmatized Data:


              precision    recall  f1-score   support

           1       0.73      0.28      0.41       220
           2       0.20      0.02      0.04       273
           3       0.43      0.05      0.09       443
           4       0.42      0.71      0.53      1064
           5       0.56      0.58      0.57      1000

    accuracy                           0.48      3000
   macro avg       0.47      0.33      0.33      3000
weighted avg       0.47      0.48      0.42      3000
```

1.7.2 Naive Bayes Classifier

[133]:
```
from sklearn.naive_bayes import MultinomialNB
# Initialize lassifier
nb_model_stem = MultinomialNB()
nb_model_lemm = MultinomialNB()

# Fit the model to the training set
nb_model_stem.fit(X_train_stem, y_train)
nb_model_lemm.fit(X_train_lemm, y_train)
```

[133]:
```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

Now let's use our model to predict ratings for test_data_features

[134]:
```
pred_stem = nb_model_stem.predict(X_test_stem)
pred_lemm = nb_model_lemm.predict(X_test_lemm)
```

[135]:
```
print('Model Results:')
print('\n')
print('Using Stemmed Data:')
print('\n')
print(classification_report(y_test, pred_stem))
print('\n')
print('Using Lemmatized Data:')
print('\n')
print(classification_report(y_test, pred_lemm))
```

```
Random Forest Model Results:


Using Stemmed Data:


              precision    recall  f1-score   support

           1       0.51      0.56      0.53       220
           2       0.37      0.28      0.32       273
           3       0.36      0.32      0.34       443
           4       0.51      0.56      0.53      1064
           5       0.60      0.59      0.59      1000

    accuracy                           0.51      3000
   macro avg       0.47      0.46      0.46      3000
weighted avg       0.50      0.51      0.51      3000



Using Lemmatized Data:
```

```
              precision    recall  f1-score   support

           1       0.47      0.52      0.49       220
           2       0.35      0.27      0.31       273
           3       0.36      0.33      0.34       443
           4       0.51      0.56      0.53      1064
           5       0.60      0.59      0.59      1000

    accuracy                           0.51      3000
   macro avg       0.46      0.45      0.45      3000
weighted avg       0.50      0.51      0.50      3000
```

1.7.3 Support Vector Machine

[83]:
```
from sklearn.svm import SVC
svm = SVC(random_state=101, gamma = 'scale')
```

[86]:
```
param_grid = {'C': [0.1,1, 10], 'gamma': [1, 0.1, 0.01, 0.001], 'kernel':␣
↪['rbf']}
```

[85]:
```
# 3 folds for 12 candidates - 36 fits - 15min
grid_stem = GridSearchCV(SVC(), param_grid, refit=True, verbose=3)
grid_stem.fit(X_train_stem, y_train)
grid_stem.best_params_
```

```
D:\Personal\Anaconda\lib\site-packages\sklearn\model_selection\_split.py:1978:
FutureWarning: The default value of cv will change from 3 to 5 in version 0.22.
Specify it explicitly to silence this warning.
  warnings.warn(CV_WARNING, FutureWarning)
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

Fitting 3 folds for each of 12 candidates, totalling 36 fits
[CV] C=0.1, gamma=1, kernel=rbf ...
[CV] ... C=0.1, gamma=1, kernel=rbf, score=0.352, total=  23.9s
[CV] C=0.1, gamma=1, kernel=rbf ...

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   23.8s remaining:    0.0s

[CV] ... C=0.1, gamma=1, kernel=rbf, score=0.352, total=  23.4s
[CV] C=0.1, gamma=1, kernel=rbf ...

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:   47.2s remaining:    0.0s

[CV] ... C=0.1, gamma=1, kernel=rbf, score=0.352, total=  23.6s
[CV] C=0.1, gamma=0.1, kernel=rbf ...
[CV] ... C=0.1, gamma=0.1, kernel=rbf, score=0.362, total=  22.7s
[CV] C=0.1, gamma=0.1, kernel=rbf ...
```

```
[CV] ... C=0.1, gamma=0.1, kernel=rbf, score=0.366, total=  23.2s
[CV] C=0.1, gamma=0.1, kernel=rbf ...
[CV] ... C=0.1, gamma=0.1, kernel=rbf, score=0.364, total=  30.2s
[CV] C=0.1, gamma=0.01, kernel=rbf ...
[CV] ... C=0.1, gamma=0.01, kernel=rbf, score=0.407, total=  20.8s
[CV] C=0.1, gamma=0.01, kernel=rbf ...
[CV] ... C=0.1, gamma=0.01, kernel=rbf, score=0.399, total=  21.2s
[CV] C=0.1, gamma=0.01, kernel=rbf ...
[CV] ... C=0.1, gamma=0.01, kernel=rbf, score=0.407, total=  21.1s
[CV] C=1, gamma=1, kernel=rbf ...
[CV] ... C=1, gamma=1, kernel=rbf, score=0.356, total=  24.2s
[CV] C=1, gamma=1, kernel=rbf ...
[CV] ... C=1, gamma=1, kernel=rbf, score=0.357, total=  24.4s
[CV] C=1, gamma=1, kernel=rbf ...
[CV] ... C=1, gamma=1, kernel=rbf, score=0.355, total=  24.0s
[CV] C=1, gamma=0.1, kernel=rbf ...
[CV] ... C=1, gamma=0.1, kernel=rbf, score=0.386, total=  24.4s
[CV] C=1, gamma=0.1, kernel=rbf ...
[CV] ... C=1, gamma=0.1, kernel=rbf, score=0.386, total=  25.1s
[CV] C=1, gamma=0.1, kernel=rbf ...
[CV] ... C=1, gamma=0.1, kernel=rbf, score=0.387, total=  24.5s
[CV] C=1, gamma=0.01, kernel=rbf ...
[CV] ... C=1, gamma=0.01, kernel=rbf, score=0.462, total=  21.5s
[CV] C=1, gamma=0.01, kernel=rbf ...
[CV] ... C=1, gamma=0.01, kernel=rbf, score=0.471, total=  20.7s
[CV] C=1, gamma=0.01, kernel=rbf ...
[CV] ... C=1, gamma=0.01, kernel=rbf, score=0.456, total=  21.1s
[CV] C=10, gamma=1, kernel=rbf ...
[CV] ... C=10, gamma=1, kernel=rbf, score=0.356, total=  27.1s
[CV] C=10, gamma=1, kernel=rbf ...
[CV] ... C=10, gamma=1, kernel=rbf, score=0.356, total=  26.2s
[CV] C=10, gamma=1, kernel=rbf ...
[CV] ... C=10, gamma=1, kernel=rbf, score=0.356, total=  25.5s
[CV] C=10, gamma=0.1, kernel=rbf ...
[CV] ... C=10, gamma=0.1, kernel=rbf, score=0.398, total=  26.6s
[CV] C=10, gamma=0.1, kernel=rbf ...
[CV] ... C=10, gamma=0.1, kernel=rbf, score=0.386, total=  24.8s
[CV] C=10, gamma=0.1, kernel=rbf ...
[CV] ... C=10, gamma=0.1, kernel=rbf, score=0.387, total=  24.2s
[CV] C=10, gamma=0.01, kernel=rbf ...
[CV] ... C=10, gamma=0.01, kernel=rbf, score=0.493, total=  23.4s
[CV] C=10, gamma=0.01, kernel=rbf ...
[CV] ... C=10, gamma=0.01, kernel=rbf, score=0.473, total=  23.3s
[CV] C=10, gamma=0.01, kernel=rbf ...
[CV] ... C=10, gamma=0.01, kernel=rbf, score=0.475, total=  23.7s
[CV] C=100, gamma=1, kernel=rbf ...
[CV] ... C=100, gamma=1, kernel=rbf, score=0.356, total=  25.2s
[CV] C=100, gamma=1, kernel=rbf ...
```

```
[CV] ... C=100, gamma=1, kernel=rbf, score=0.356, total=  24.6s
[CV] C=100, gamma=1, kernel=rbf ...
[CV] ... C=100, gamma=1, kernel=rbf, score=0.356, total=  26.2s
[CV] C=100, gamma=0.1, kernel=rbf ...
[CV] ... C=100, gamma=0.1, kernel=rbf, score=0.393, total=  26.4s
[CV] C=100, gamma=0.1, kernel=rbf ...
[CV] ... C=100, gamma=0.1, kernel=rbf, score=0.382, total=  26.9s
[CV] C=100, gamma=0.1, kernel=rbf ...
[CV] ... C=100, gamma=0.1, kernel=rbf, score=0.385, total=  25.2s
[CV] C=100, gamma=0.01, kernel=rbf ...
[CV] ... C=100, gamma=0.01, kernel=rbf, score=0.461, total=  21.6s
[CV] C=100, gamma=0.01, kernel=rbf ...
[CV] ... C=100, gamma=0.01, kernel=rbf, score=0.444, total=  21.2s
[CV] C=100, gamma=0.01, kernel=rbf ...
[CV] ... C=100, gamma=0.01, kernel=rbf, score=0.455, total=  22.4s

[Parallel(n_jobs=1)]: Done  36 out of  36 | elapsed: 14.4min finished
```

[85]:
```
{'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
```

[87]:
```python
# 3 folds for 12 candidates - 36 fits - 13min
grid_lemm = GridSearchCV(SVC(), param_grid, refit=True, verbose=3)
grid_lemm.fit(X_train_lemm, y_train)
grid_lemm.best_params_
```

```
D:\Personal\Anaconda\lib\site-packages\sklearn\model_selection\_split.py:1978:
FutureWarning: The default value of cv will change from 3 to 5 in version 0.22.
Specify it explicitly to silence this warning.
  warnings.warn(CV_WARNING, FutureWarning)
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

Fitting 3 folds for each of 12 candidates, totalling 36 fits
[CV] C=0.1, gamma=1, kernel=rbf ...
[CV] ... C=0.1, gamma=1, kernel=rbf, score=0.352, total=  22.5s
[CV] C=0.1, gamma=1, kernel=rbf ...

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   22.4s remaining:    0.0s

[CV] ... C=0.1, gamma=1, kernel=rbf, score=0.352, total=  22.7s
[CV] C=0.1, gamma=1, kernel=rbf ...

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:   45.1s remaining:    0.0s

[CV] ... C=0.1, gamma=1, kernel=rbf, score=0.352, total=  22.1s
[CV] C=0.1, gamma=0.1, kernel=rbf ...
[CV] ... C=0.1, gamma=0.1, kernel=rbf, score=0.361, total=  22.1s
[CV] C=0.1, gamma=0.1, kernel=rbf ...
[CV] ... C=0.1, gamma=0.1, kernel=rbf, score=0.366, total=  22.5s
[CV] C=0.1, gamma=0.1, kernel=rbf ...
```

```
[CV] ... C=0.1, gamma=0.1, kernel=rbf, score=0.364, total=  22.4s
[CV] C=0.1, gamma=0.01, kernel=rbf ...
[CV] ... C=0.1, gamma=0.01, kernel=rbf, score=0.409, total=  20.0s
[CV] C=0.1, gamma=0.01, kernel=rbf ...
[CV] ... C=0.1, gamma=0.01, kernel=rbf, score=0.396, total=  20.1s
[CV] C=0.1, gamma=0.01, kernel=rbf ...
[CV] ... C=0.1, gamma=0.01, kernel=rbf, score=0.410, total=  19.8s
[CV] C=0.1, gamma=0.001, kernel=rbf ...
[CV] ... C=0.1, gamma=0.001, kernel=rbf, score=0.352, total=  18.8s
[CV] C=0.1, gamma=0.001, kernel=rbf ...
[CV] ... C=0.1, gamma=0.001, kernel=rbf, score=0.352, total=  18.8s
[CV] C=0.1, gamma=0.001, kernel=rbf ...
[CV] ... C=0.1, gamma=0.001, kernel=rbf, score=0.352, total=  19.2s
[CV] C=1, gamma=1, kernel=rbf ...
[CV] ... C=1, gamma=1, kernel=rbf, score=0.356, total=  23.5s
[CV] C=1, gamma=1, kernel=rbf ...
[CV] ... C=1, gamma=1, kernel=rbf, score=0.357, total=  23.2s
[CV] C=1, gamma=1, kernel=rbf ...
[CV] ... C=1, gamma=1, kernel=rbf, score=0.355, total=  23.7s
[CV] C=1, gamma=0.1, kernel=rbf ...
[CV] ... C=1, gamma=0.1, kernel=rbf, score=0.387, total=  23.2s
[CV] C=1, gamma=0.1, kernel=rbf ...
[CV] ... C=1, gamma=0.1, kernel=rbf, score=0.385, total=  24.8s
[CV] C=1, gamma=0.1, kernel=rbf ...
[CV] ... C=1, gamma=0.1, kernel=rbf, score=0.387, total=  23.5s
[CV] C=1, gamma=0.01, kernel=rbf ...
[CV] ... C=1, gamma=0.01, kernel=rbf, score=0.468, total=  20.5s
[CV] C=1, gamma=0.01, kernel=rbf ...
[CV] ... C=1, gamma=0.01, kernel=rbf, score=0.462, total=  20.1s
[CV] C=1, gamma=0.01, kernel=rbf ...
[CV] ... C=1, gamma=0.01, kernel=rbf, score=0.455, total=  19.7s
[CV] C=1, gamma=0.001, kernel=rbf ...
[CV] ... C=1, gamma=0.001, kernel=rbf, score=0.430, total=  18.2s
[CV] C=1, gamma=0.001, kernel=rbf ...
[CV] ... C=1, gamma=0.001, kernel=rbf, score=0.426, total=  18.1s
[CV] C=1, gamma=0.001, kernel=rbf ...
[CV] ... C=1, gamma=0.001, kernel=rbf, score=0.438, total=  20.9s
[CV] C=10, gamma=1, kernel=rbf ...
[CV] ... C=10, gamma=1, kernel=rbf, score=0.356, total=  23.4s
[CV] C=10, gamma=1, kernel=rbf ...
[CV] ... C=10, gamma=1, kernel=rbf, score=0.357, total=  23.7s
[CV] C=10, gamma=1, kernel=rbf ...
[CV] ... C=10, gamma=1, kernel=rbf, score=0.357, total=  23.5s
[CV] C=10, gamma=0.1, kernel=rbf ...
[CV] ... C=10, gamma=0.1, kernel=rbf, score=0.398, total=  23.8s
[CV] C=10, gamma=0.1, kernel=rbf ...
[CV] ... C=10, gamma=0.1, kernel=rbf, score=0.387, total=  23.8s
[CV] C=10, gamma=0.1, kernel=rbf ...
```

```
[CV] ... C=10, gamma=0.1, kernel=rbf, score=0.389, total=  23.4s
[CV] C=10, gamma=0.01, kernel=rbf ...
[CV] ... C=10, gamma=0.01, kernel=rbf, score=0.487, total=  21.9s
[CV] C=10, gamma=0.01, kernel=rbf ...
[CV] ... C=10, gamma=0.01, kernel=rbf, score=0.483, total=  21.7s
[CV] C=10, gamma=0.01, kernel=rbf ...
[CV] ... C=10, gamma=0.01, kernel=rbf, score=0.471, total=  21.9s
[CV] C=10, gamma=0.001, kernel=rbf ...
[CV] ... C=10, gamma=0.001, kernel=rbf, score=0.488, total=  16.3s
[CV] C=10, gamma=0.001, kernel=rbf ...
[CV] ... C=10, gamma=0.001, kernel=rbf, score=0.483, total=  20.7s
[CV] C=10, gamma=0.001, kernel=rbf ...
[CV] ... C=10, gamma=0.001, kernel=rbf, score=0.477, total=  17.8s

[Parallel(n_jobs=1)]: Done  36 out of  36 | elapsed: 12.9min finished
```

[87]:
```
{'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
```

[90]:
```
pred_stem = grid_stem.predict(X_test_stem)
pred_lemm = grid_lemm.predict(X_test_lemm)
```

[91]:
```
print('Random Forest Model Results:')
print('\n')
print('Using Stemmed Data:')
print('\n')
print(classification_report(y_test, pred_stem))
print('\n')
print('Using Lemmatized Data:')
print('\n')
print(classification_report(y_test, pred_lemm))
```

```
Random Forest Model Results:


Using Stemmed Data:


              precision    recall  f1-score   support

           1       0.52      0.37      0.43       220
           2       0.36      0.23      0.28       273
           3       0.33      0.27      0.30       443
           4       0.48      0.57      0.52      1064
           5       0.58      0.61      0.59      1000

    accuracy                           0.49      3000
   macro avg       0.45      0.41      0.43      3000
weighted avg       0.48      0.49      0.48      3000
```

Using Lemmatized Data:

```
              precision    recall  f1-score   support

           1       0.55      0.38      0.45       220
           2       0.40      0.22      0.28       273
           3       0.40      0.25      0.30       443
           4       0.48      0.55      0.51      1064
           5       0.56      0.68      0.61      1000

    accuracy                           0.50      3000
   macro avg       0.48      0.41      0.43      3000
weighted avg       0.49      0.50      0.49      3000
```

1.7.4 XGBoost Classifier

[97]:
```
#pip install xgboost
```

[98]:
```
from xgboost import XGBClassifier
```

[99]:
```
xgb_stem = XGBClassifier()
xgb_lemm = XGBClassifier()

xgb_stem.fit(X_train_stem,y_train)
xgb_lemm.fit(X_train_lemm,y_train)

print('Random Forest Model Results:')
print('\n')
print('Using Stemmed Data:')
print('\n')
print(classification_report(y_test, pred_stem))
print('\n')
print('Using Lemmatized Data:')
print('\n')
print(classification_report(y_test, pred_lemm))
```

Random Forest Model Results:

Using Stemmed Data:

```
              precision    recall  f1-score   support
```

```
             1       0.52      0.37      0.43       220
             2       0.36      0.23      0.28       273
             3       0.33      0.27      0.30       443
             4       0.48      0.57      0.52      1064
             5       0.58      0.61      0.59      1000

      accuracy                           0.49      3000
     macro avg       0.45      0.41      0.43      3000
  weighted avg       0.48      0.49      0.48      3000



Using Lemmatized Data:


              precision    recall  f1-score   support

             1       0.55      0.38      0.45       220
             2       0.40      0.22      0.28       273
             3       0.40      0.25      0.30       443
             4       0.48      0.55      0.51      1064
             5       0.56      0.68      0.61      1000

      accuracy                           0.50      3000
     macro avg       0.48      0.41      0.43      3000
  weighted avg       0.49      0.50      0.49      3000
```

1.7.5 Multilayer Perseptron Classifier

[100]:
```
# MULTILAYER PERCEPTRON CLASSIFIER
from sklearn.neural_network import MLPClassifier

mlp_stem = MLPClassifier()
mlp_lemm = MLPClassifier()

mlp_stem.fit(X_train_stem, y_train)
mlp_lemm.fit(X_train_lemm, y_train)

pred_stem = mlp_stem.predict(X_test_stem)
pred_lemm = mlp_lemm.predict(X_test_lemm)

print('Random Forest Model Results:')
print('\n')
print('Using Stemmed Data:')
print('\n')
print(classification_report(y_test, pred_stem))
```

```
print('\n')
print('Using Lemmatized Data:')
print('\n')
print(classification_report(y_test, pred_lemm))
```

```
Random Forest Model Results:


Using Stemmed Data:


              precision    recall  f1-score   support

           1       0.54      0.47      0.50       220
           2       0.31      0.28      0.30       273
           3       0.33      0.33      0.33       443
           4       0.49      0.48      0.49      1064
           5       0.55      0.59      0.57      1000

    accuracy                           0.48      3000
   macro avg       0.44      0.43      0.44      3000
weighted avg       0.47      0.48      0.47      3000



Using Lemmatized Data:


              precision    recall  f1-score   support

           1       0.49      0.40      0.44       220
           2       0.32      0.27      0.29       273
           3       0.32      0.32      0.32       443
           4       0.49      0.49      0.49      1064
           5       0.54      0.58      0.56      1000

    accuracy                           0.47      3000
   macro avg       0.43      0.41      0.42      3000
weighted avg       0.46      0.47      0.46      3000
```

1.8 Predicting Rating using TF-IDF

Transform our vectorized data into TF-IDF format

[287]:
```
# Create TF-IDF for stemmed data
tfidf_transformer_stem = TfidfTransformer()
tfidf_transformer_stem.fit(X_train_stem)
```

```
X_train_stem_tfidf = tfidf_transformer_stem.transform(X_train_stem)
X_test_stem_tfidf = tfidf_transformer_stem.transform(X_test_stem)
```

[288]:
```
# Create TF-IDF for lemmatized data
tfidf_transformer_lemm = TfidfTransformer()
tfidf_transformer_lemm.fit(X_train_lemm)
X_train_lemm_tfidf = tfidf_transformer_lemm.transform(X_train_lemm)
X_test_lemm_tfidf = tfidf_transformer_stem.transform(X_test_lemm)
```

1.8.1 Random Forest

[108]:
```
# Initialize a Random Forest classifier with 200 trees
forest_stem = RandomForestClassifier(n_estimators = 200)
forest_lemm = RandomForestClassifier(n_estimators = 200)

# Fit the forest to the training set, using the bag of words as features and␣
↪the sentiment labels as the response variable
forest_stem.fit(X_train_stem_tfidf, y_train)
forest_lemm.fit(X_train_lemm_tfidf, y_train)
```

[108]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=200,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

[109]:
```
pred_stem = forest_stem.predict(X_test_stem_tfidf)
pred_lemm = forest_lemm.predict(X_test_lemm_tfidf)
```

[110]:
```
print('Random Forest Model Results:')
print('\n')
print('Using Stemmed Data:')
print('\n')
print(classification_report(y_test, pred_stem))
print('\n')
print('Using Lemmatized Data:')
print('\n')
print(classification_report(y_test, pred_lemm))
```

```
Random Forest Model Results:


Using Stemmed Data:


              precision    recall  f1-score   support
```

```
             1       0.80      0.27      0.40       220
             2       0.40      0.01      0.03       273
             3       0.52      0.04      0.07       443
             4       0.41      0.70      0.52      1064
             5       0.55      0.61      0.58      1000

      accuracy                           0.48      3000
     macro avg       0.54      0.32      0.32      3000
  weighted avg       0.50      0.48      0.42      3000



Using Lemmatized Data:


              precision    recall  f1-score   support

             1       0.71      0.21      0.32       220
             2       0.38      0.03      0.05       273
             3       0.43      0.06      0.11       443
             4       0.42      0.73      0.54      1064
             5       0.55      0.57      0.56      1000

      accuracy                           0.47      3000
     macro avg       0.50      0.32      0.32      3000
  weighted avg       0.48      0.47      0.42      3000
```

1.8.2 Naive Bayes classifier

[289]:
```python
# Initialize a Random Forest classifier with 200 trees
nb_model_stem = MultinomialNB()
nb_model_lemm = MultinomialNB()

# Fit the forest to the training set, using the bag of words as features and␣
↪the sentiment labels as the response variable
nb_model_stem.fit(X_train_stem_tfidf, y_train)
nb_model_lemm.fit(X_train_lemm_tfidf, y_train)
```

[289]:
```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

[290]:
```python
pred_stem = nb_model_stem.predict(X_test_stem_tfidf)
pred_lemm = nb_model_lemm.predict(X_test_lemm_tfidf)
```

[291]:
```python
print('Model Results:')
print('\n')
print('Using Stemmed Data:')
print('\n')
```

```
print(classification_report(y_test, pred_stem))
print('\n')
print('Using Lemmatized Data:')
print('\n')
print(classification_report(y_test, pred_lemm))
```

```
Model Results:


Using Stemmed Data:


              precision    recall  f1-score   support

           1       1.00      0.03      0.05       220
           2       0.00      0.00      0.00       273
           3       0.45      0.01      0.02       443
           4       0.40      0.76      0.53      1064
           5       0.55      0.53      0.54      1000

    accuracy                           0.45      3000
   macro avg       0.48      0.27      0.23      3000
weighted avg       0.47      0.45      0.37      3000



Using Lemmatized Data:


              precision    recall  f1-score   support

           1       1.00      0.00      0.01       220
           2       0.00      0.00      0.00       273
           3       0.60      0.01      0.01       443
           4       0.40      0.81      0.53      1064
           5       0.60      0.50      0.55      1000

    accuracy                           0.46      3000
   macro avg       0.52      0.26      0.22      3000
weighted avg       0.50      0.46      0.37      3000
```

The best model is Naive Bayes Classifier trained on vectorized stemmed reviews. Model's accuracy is **0.51** Model's weighted average F1-score is **0.51**

1.9 Classification Accuracy Results:

Weighted Average F1-Score

Word Vectorizing *5000 words vocab*

1. Stemmed data

   - Random Forest(n=200) = **0.42**
   - Naive Bayes Classifier = **0.51**
   - SVM (gridsearch) = **0.48**
   - XGBoost = **0.48**
   - Multilayer Perceptron = **0.47**

2. Lemmatized data

   - Random Forest(n=200) = **0.42**
   - Naive Bayes Classifier = **0.50**
   - SVM (gridsearch) = **0.49**
   - XGBoost = **0.49**
   - Multilayer Perceptron = **0.46**

TF-IDF *5000 words vocab*

1. Stemmed data

   - Random Forest(n=200) = **0.42**
   - Naive Bayes Classifier = **0.37**

2. Lemmatized dataM

   - Random Forest(n=200) = **0.42**
   - Naive Bayes Classifier = **0.37**

Conclusion:
The most accurate model was **Naive Bayes Classifier** using vectorized stemmed data.

Overall, there are no significant differencies in accuracy between models based on stemmed and lemmatized data.

Converting data to TF-IDF format didn't increase the accuracy of classification models.