Brian_Sampreeth_hw2

March 14, 2024

Sampreeth Avvari

Net ID: spa9659

TA Name: Yirong (Brett) Bian

[ ]:
```
from google.colab import files

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

import nltk
import string
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer, PorterStemmer
from nltk.corpus import stopwords
from nltk.collocations import BigramCollocationFinder
from nltk.metrics import BigramAssocMeasures
from scipy.stats import chi2, chi2_contingency
import re
from tqdm import tqdm
from functools import reduce
from operator import mul
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
nltk.download('wordnet')
nltk.download('punkt')
nltk.download('stopwords')
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
from nltk.collocations import BigramCollocationFinder
from nltk import BigramAssocMeasures
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data] Downloading package punkt to /root/nltk_data…
```

```
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
```

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

1 Q1

1.1 1a

create corpus from given sentences, separately and make them a list and make dtm

[ ]:
```
rep='immigration aliens criminals loophole country  voter economy tax growth␣
↪security healthcare cost socialism unfair help'
dem='immigration country growth help voter healthcare inequality expansion␣
↪unfair economy infrastructure opportunity expansion country security␣
↪abortion choice right women help'
```

[ ]:
```
corpus=[rep, dem]
len(corpus)
```

[ ]: 2

[ ]:
```
def q1_preprocess(doc):
  d = doc.translate(str.maketrans('', '', string.punctuation))
  tokens = word_tokenize(d)
  print(len(set(tokens)))
  return ' '.join(tokens)
```

[ ]:
```
q1_preproc=list(map(q1_preprocess, corpus))
```

```
15
17
```

[ ]:
```
q1_vectorizer=CountVectorizer()
q1_dtm = q1_vectorizer.fit_transform(q1_preproc)
```

[ ]:
```
vocab = q1_vectorizer.get_feature_names_out()
doc_labels = range(len(corpus))

df = pd.DataFrame(q1_dtm.toarray(), columns=vocab, index=doc_labels)
df.index=['Republican', 'Democrat']
df
```

[ ]:
```
            abortion  aliens  choice  cost  country  criminals  economy  \
Republican         0       1       0     1        1          1        1
```

```
Democrat            1        0        1      0         2           0         1

            expansion  growth  healthcare  …  infrastructure  loophole  \
Republican          0       1           1  …               0         1
Democrat            2       1           1  …               1         0

            opportunity  right  security  socialism  tax  unfair  voter  women
Republican            0      0         1          1    1       1      1      0
Democrat              1      1         1          0    0       1      1      1

[2 rows x 23 columns]
```

"infrastructure voter growth help economy"

[ ]:
```python
import pandas as pd

data = {
    'email': [
        'immigration aliens criminals loophole countryimmigration aliens␣
 ↪criminals loophole country',
        'voter economy tax growth security',
        'healthcare cost socialism unfair help',
        'immigration country growth help voter',
        'healthcare inequality expansion unfair economy',
        'infrastructure opportunity expansion country security',
        'abortion choice right women help'
    ],
    'y': [
        'republican',
        'republican',
        'republican',
        'democrat',
        'democrat',
        'democrat',
        'democrat'
    ]
}

q1_df = pd.DataFrame(data)
q1_df
```

[ ]:
```
                                               email           y
0  immigration aliens criminals loophole countryi…  republican
1                  voter economy tax growth security  republican
2              healthcare cost socialism unfair help  republican
3              immigration country growth help voter    democrat
4     healthcare inequality expansion unfair economy    democrat
5  infrastructure opportunity expansion country s…    democrat
```

```
6                     abortion choice right women help    democrat
```

```
[ ]: test_email="infrastructure voter growth help economy"

     email_words=word_tokenize(test_email)
     email_words
```

```
[ ]: ['infrastructure', 'voter', 'growth', 'help', 'economy']
```

```
[ ]: rep_word_count = {}
     for w in email_words:
       rep_word_count[w] = df.loc['Republican',w]
     # sum(rep_word_count.values())
     dem_word_count = {}
     for w in email_words:
       dem_word_count[w] = df.loc['Democrat',w]
```

```
[ ]: likelihood_rep = np.array([rep_word_count[word] / (rep_word_count[word] +␣
      ↪dem_word_count[word])  for word in email_words])
     likelihood_dem = np.array([dem_word_count[word] / (rep_word_count[word] +␣
      ↪dem_word_count[word])  for word in email_words])
```

```
[ ]: prior_rep = 3 / 7  # Based on the provided dataset: 3 Republican emails out of␣
      ↪7 total
     prior_dem = 4 / 7  # Based on the provided dataset: 4 Democrat emails out of 7␣
      ↪total
```

```
[ ]: posterior_rep = prior_rep * np.prod(likelihood_rep)
     posterior_dem = prior_dem * np.prod(likelihood_dem)
```

```
[ ]: print(f"Posterior Prob Republican: {posterior_rep}")
     print(f"Posterior Prob Democrat: {posterior_dem}")

     # prediction based on the higher posterior prob
     if posterior_rep > posterior_dem:
         print("The mystery email was likely sent by the Republican party.")
     else:
         print("The mystery email was likely sent by the Democrat party.")
```

```
Posterior Prob Republican: 0.0
Posterior Prob Democrat: 0.047619047619047616
The mystery email was likely sent by the Democrat party.
```

These conclusions seem unreliable due to the methodology of analyzing word occurrences in emails from both Republican and Democratic sources. A particular instance where a word was absent in Republican communications reduced the likelihood to zero, which raises concerns about the robustness of these results. Additionally, the data set used for both parties appears to be insufficiently large, suggesting that more comprehensive data collection is necessary to draw more accurate conclusions.

1.2 1b

[ ]:
```
vocab_size=df.shape[1]
vocab_size
```

[ ]: 23

[ ]:
```
likelihood_rep_smoothed = np.array([(rep_word_count[word] + 1) /␣
↪(rep_word_count[word] + dem_word_count[word] + vocab_size) for word in␣
↪email_words])
likelihood_dem_smoothed = np.array([(dem_word_count[word] + 1) /␣
↪(rep_word_count[word] + dem_word_count[word] + vocab_size) for word in␣
↪email_words])

# Recalculate posterior probabilities with smoothing
posterior_rep_smoothed = prior_rep * np.prod(likelihood_rep_smoothed)
posterior_dem_smoothed = prior_dem * np.prod(likelihood_dem_smoothed)

# Output the smoothed results
print(f"Smoothed Posterior Prob for Republican: {posterior_rep_smoothed}")
print(f"Smoothed Posterior Prob for Democrat: {posterior_dem_smoothed}")

# Make a prediction based on the higher posterior probability with smoothing
if posterior_rep_smoothed > posterior_dem_smoothed:
    print("Smoothed Prediction: The mystery email was likely sent by the␣
↪Republican party.")
else:
    print("Smoothed Prediction: The mystery email was likely sent by the␣
↪Democrat party.")
```

```
Smoothed Posterior Prob for Republican: 7.032967032967033e-07
Smoothed Posterior Prob for Democrat: 2.8131868131868135e-06
Smoothed Prediction: The mystery email was likely sent by the Democrat party.
```

- Smoothing addresses the issue of encountering zero probabilities, as observed in scenario 1a, where the absence of a specific word in the Republican emails led to a zero probability. Through smoothing, we can mitigate this issue.
- Additionally, smoothing aids in enhancing the model's ability to generalize by allocating probabilities to words that have not been observed in the dataset.
- This technique also contributes to the robustness of the analysis. Particularly in situations where the dataset is limited, as in scenario 1a, the absence of certain words might be attributed to inadequate sampling rather than their true irrelevance in the context, such as in emails.

1.3 1c

```
[ ]: from functools import reduce
from operator import mul
import numpy as np


total_words_rep = df.loc['Republican', :].sum()
total_words_dem = df.loc['Democrat', :].sum()


likelihoods_rep_smoothed = {word: (df.loc['Republican', word] + 1) /␣
 ↪(total_words_rep + vocab_size) for word in vocab}
likelihoods_dem_smoothed = {word: (df.loc['Democrat', word] + 1) /␣
 ↪(total_words_dem + vocab_size) for word in vocab}


likelihood_diffs = {word: likelihoods_dem_smoothed[word] -␣
 ↪likelihoods_rep_smoothed[word] for word in vocab}


sorted_words = sorted(likelihood_diffs, key=likelihood_diffs.get, reverse=True)


top_2_words_dem_favored = sorted_words[:2]


top_2_words_dem_favored_diffs = {word: likelihood_diffs[word] for word in␣
 ↪top_2_words_dem_favored}
print("Top 2 Words to Increase Democrat Classification:",␣
 ↪top_2_words_dem_favored)
print("Differences in Likelihood:", top_2_words_dem_favored_diffs)
```

```
Top 2 Words to Increase Democrat Classification: ['expansion', 'abortion']
Differences in Likelihood: {'expansion': 0.043451652386780906, 'abortion':
0.020195838433292534}
```

To achieve this objective, we need to pinpoint words that demonstrate a significantly greater posterior probability of appearing in Democratic emails compared to Republican ones. Essentially, this involves focusing on words that are more likely to be found in communications from Democrats but are absent in those from Republicans.

1.4 1d

The Democrats have various strategies at their disposal to address the Republicans' approach to accessing their emails. They might:

- Consider the inclusion of email metadata, such as the sender's domain or the time the email was sent, to enhance their filtering techniques.

- Implement advanced NLP (Natural Language Processing) methods that account for context, thereby improving accuracy. For instance, the identification of keywords like 'expansion' and 'abortion' might not make coherent sense in certain arrangements, such as in the phrase 'healthcare expansion abortion'. Utilizing more sophisticated algorithms, such as Recurrent Neural Networks (RNNs), could offer a more effective solution.

2 2

[ ]:
```
import pandas as pd
q2_df= pd.read_csv('/content/hotelreviews_c.csv', index_col=False)
```

[ ]:
```
q2_df=q2_df.drop('Unnamed: 0', axis=1)
```

[ ]:
```
q2_df
```

[ ]:
```
      stars                                               text
0         4  As far as the room it self I would actually sa…
1         5  Wonderful as always, the consistent level of q…
2         4  Location was very convenient, although using t…
3         4  We were very tired when we arrived with a cran…
4         3  Somewhat updated, nice fireplace and clean. We…
…       …                                                 …
7495      5  We stayed here for the third time for our 5th …
7496      5  Friendly staff, wood floors in room. place see…
7497      5  stayed here the weekend of the Rock and Roll N…
7498      3  We stayed here to attend Kentucky Derby and sp…
7499      3  The sweeper hadnt been ran in our room when we…

[7500 rows x 2 columns]
```

[ ]:
```
median_star=q2_df['stars'].median(axis=0)
median_star
```

[ ]:
```
4.0
```

[ ]:
```
q2_df['label']=(q2_df['stars']>=median_star).astype(int)
```

[ ]:
```
q2_df
```

[ ]:
```
      stars                                               text  label
0         4  As far as the room it self I would actually sa…      1
1         5  Wonderful as always, the consistent level of q…      1
2         4  Location was very convenient, although using t…      1
3         4  We were very tired when we arrived with a cran…      1
4         3  Somewhat updated, nice fireplace and clean. We…      0
…       …                                                 …    …
7495      5  We stayed here for the third time for our 5th …      1
```

```
7496      5  Friendly staff, wood floors in room. place see…      1
7497      5  stayed here the weekend of the Rock and Roll N…      1
7498      3  We stayed here to attend Kentucky Derby and sp…      0
7499      3  The sweeper hadnt been ran in our room when we…      0

[7500 rows x 3 columns]
```

[ ]:
```
propor=q2_df['label'].value_counts(normalize=True)
```

[ ]:
```
print("The proportions of +ve  ratings - {} and -ve ratings are {}␣
↪respectively".format(propor[0],propor[1]))
```

```
The proportions of +ve  ratings - 0.24973333333333333 and -ve ratings are
0.7502666666666666 respectively
```

3 3

##3a

[ ]:
```
negative_words_path = '/content/negative-words.txt'
positive_words_path = '/content/positive-words.txt'
with open(negative_words_path, 'r') as file:
    negative_words = [line.strip() for line in file if not line.startswith(';')␣
↪and line.strip() != '']

with open(positive_words_path, 'r') as file:
    positive_words = [line.strip() for line in file if not line.startswith(';')␣
↪and line.strip() != '']
```

[ ]:
```
import re
# def q3_preprocess(doc):
#   text = re.sub(r'\\x[0-9A-Fa-f]{2}', ' ', doc)
#   text = re.sub(r'[0-9]',' ',text)
#   text = text.translate(str.maketrans('', '', string.punctuation)).lower()
#   tokens = word_tokenize(text)
#   stop_words = set(stopwords.words("english"))
#   output = [word for word in tokens if word not in stop_words]
#   return " ".join(output)

def q3_preprocess(doc):
    text = re.sub(r'\\x[0-9A-Fa-f]{2}', ' ', doc)
    text = re.sub(r'[0-9]', ' ', text)
    text = text.translate(str.maketrans('', '', string.punctuation)).lower()
    tokens = word_tokenize(text)
    stop_words = set(stopwords.words("english"))
    filtered_tokens = [word for word in tokens if word not in stop_words]
```

```
    return " ".join(filtered_tokens)
```

[ ]:
```
q3_preproc = [q3_preprocess(str(doc)) for doc in q2_df['text']]

q3_vectorizer = CountVectorizer()
q3_dtm = q3_vectorizer.fit_transform(q3_preproc)

vocab = q3_vectorizer.get_feature_names_out()
doc_labels = range(len(q3_preproc))
q3_dtm_df = pd.DataFrame(q3_dtm.toarray(), columns=vocab, index=doc_labels)
```

[ ]:
```
# Rename columns based on sentiment
ni, pi = 0, 0
for col in q3_dtm_df.columns:
    if col in negative_words:
        q3_dtm_df.rename(columns={col: f'negative{ni}'}, inplace=True)
        ni += 1
    elif col in positive_words:
        q3_dtm_df.rename(columns={col: f'positive{pi}'}, inplace=True)
        pi += 1

q3_positive = q3_dtm_df.filter(like='positive')
q3_negative = q3_dtm_df.filter(like='negative')

q3_p_sum = q3_positive.sum(axis=1)
q3_n_sum = q3_negative.sum(axis=1)
q3_results = q3_p_sum - q3_n_sum

results = q3_results.apply(lambda x: 'Negative' if x <= 0 else 'Positive')
q2_df['Sentiment'] = results

# print(q2_df)
# plt.hist(q3_results)
# plt.show()
```

##3c

[ ]:
```
def checker(t):
  if t is True:
    return 'Negative'
  return 'Positive'
results = q3_results.apply(lambda x: checker(x<=0))
results.value_counts()
```

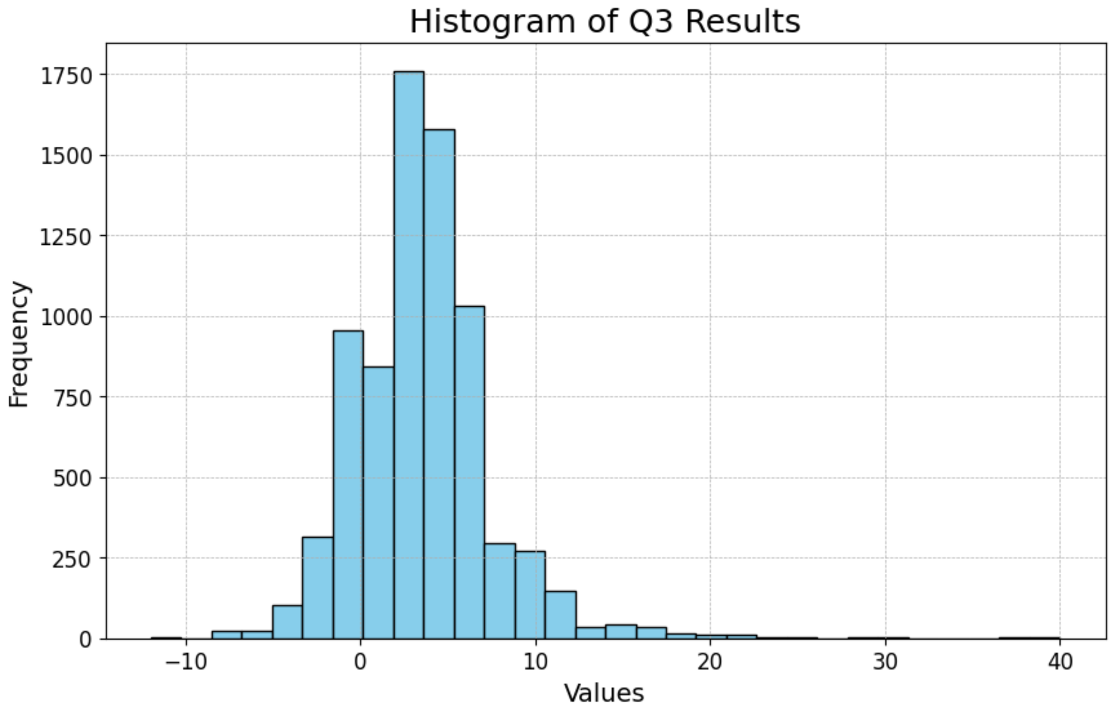
[ ]:
```
Positive    6082
Negative    1418
dtype: int64
```

[ ]:
```
q2_df['Sentiment']=results

q2_df.sample(n=5)
#already completed 3c above
```

[ ]:
```
      stars                                               text  label  \
486       3  It was an intermediate stop on a long journey,…      0
1506      4  The hotel is very out-dated - it had the felli…      1
4895      2  Don't know how this place has some many high r…      0
5377      4  We stayed here for one night after driving up …      1
737       4  Room was clean and spacious, when u walk in th…      1

     Sentiment
486   Positive
1506  Positive
4895  Negative
5377  Positive
737   Positive
```

##3b

[ ]:
```
#plotting the results
plt.figure(figsize=(10, 6))
plt.hist(q3_results, bins=30, color='skyblue', edgecolor='black')
plt.title('Histogram of Q3 Results', fontsize=18)

plt.xlabel('Values', fontsize=14)

plt.ylabel('Frequency', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.show()
```



[ ]:
```
q2_df
```

[ ]:
```
      stars                                               text  label  \
0         4  As far as the room it self I would actually sa…      1
1         5  Wonderful as always, the consistent level of q…      1
2         4  Location was very convenient, although using t…      1
3         4  We were very tired when we arrived with a cran…      1
4         3  Somewhat updated, nice fireplace and clean. We…      0
…       …                                                …    …
7495      5  We stayed here for the third time for our 5th …      1
7496      5  Friendly staff, wood floors in room. place see…      1
7497      5  stayed here the weekend of the Rock and Roll N…      1
7498      3  We stayed here to attend Kentucky Derby and sp…      0
7499      3  The sweeper hadnt been ran in our room when we…      0

     Sentiment
0     Positive
1     Positive
2     Positive
3     Positive
4     Positive
…          …
7495  Positive
7496  Positive
```

```
7497   Positive
7498   Positive
7499   Negative

[7500 rows x 4 columns]
```

[ ]:
```
neg_scores_mean=q3_results[q3_results.apply(lambda x: x<=0)].mean()
neg_scores_std=q3_results[q3_results.apply(lambda x: x<=0)].std()
neg_scores_std
```

[ ]:
```
1.6721383003973427
```

[ ]:
```
pos_scores_mean=q3_results[q3_results.apply(lambda x: x>0)].mean()
pos_scores_std=q3_results[q3_results.apply(lambda x: x>0)].std()
pos_scores_std
```

[ ]:
```
3.300332478429942
```

[ ]:
```
pd.DataFrame({'mean':[neg_scores_mean, pos_scores_mean], 'std':
 ↪[neg_scores_std, pos_scores_std]}, index=['negative', 'positive'])
```

[ ]:
```
              mean       std
negative -1.303949  1.672138
positive  4.621013  3.300332
```

[ ]:
```
results.value_counts(normalize=True)*100
```

[ ]:
```
Positive    81.093333
Negative    18.906667
dtype: float64
```

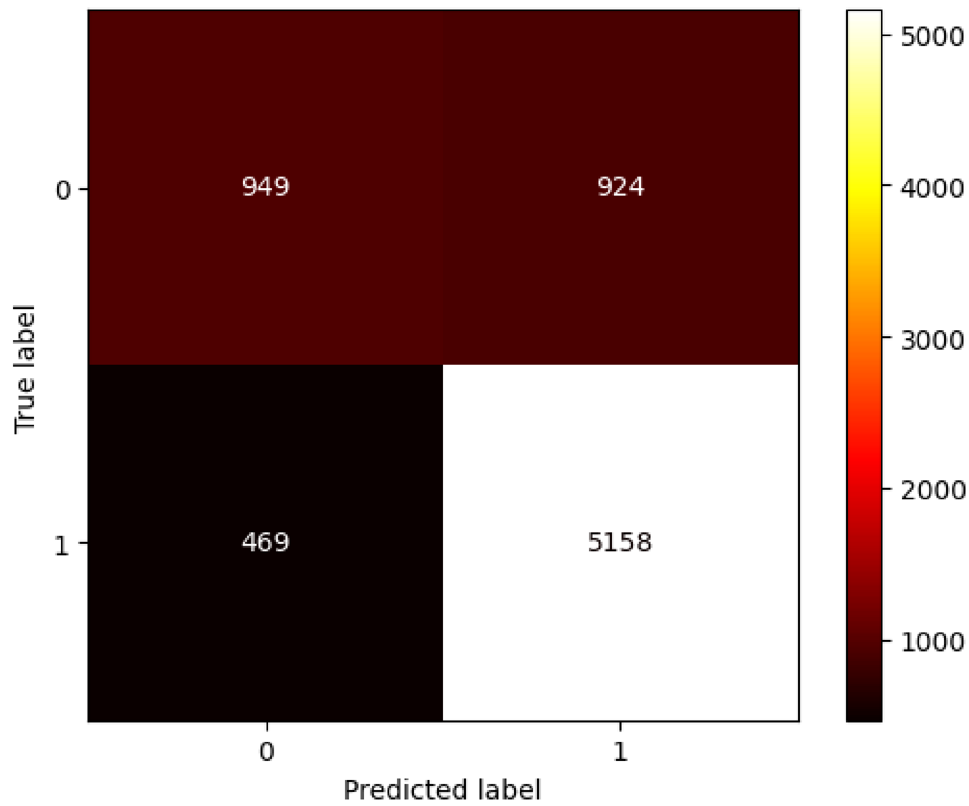
Observing that 75% of the star ratings fall above the empirical median, our approach to categorizing reviews as 1 or 0 based on this median explains the similarity in the current outcome proportions. Essentially, this indicates that the dictionary method we've applied has indeed achieved this distinction, albeit there's room for refinement to enhance its accuracy.

##3d

[ ]:
```
from sklearn.metrics import accuracy_score, classification_report,
 ↪confusion_matrix, f1_score, precision_score, recall_score,
 ↪ConfusionMatrixDisplay
def encoder(x):
    return 1 if x == 'Positive' else 0

y_test = q2_df['label']
y_pred = q2_df['Sentiment'].apply(encoder)

conf = confusion_matrix(y_test, y_pred)
display = ConfusionMatrixDisplay(conf)
```

```
display.plot(cmap='hot')
```

[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7c7ac35f8d30>



[ ]:
```
acc_score=accuracy_score(y_test, y_pred)
acc_score
```

[ ]: 0.8142666666666667

[ ]:
```
f1_score(y_test, y_pred)
```

[ ]: 0.8810316850286105

[ ]:
```
precision_score(y_test, y_pred)
```

[ ]: 0.8480762906938507

[ ]:
```
recall_score(y_test, y_pred)
```

[ ]: 0.9166518571174693

Our benchmark for comparison is the predominant class in our dataset, identified as the positive class, constituting 75% of the data. Despite this overwhelming majority of positive instances, the classifier's accuracy hovers only around 81%. This modest performance underscores the limitations of the dictionary approach we've explored in our discussions.

##3e

[ ]:
```python
# false_neg=q2_df[(q2_df['stars']==5) &␣
↪(q2_df['Sentiment']=='Negative')]['text'][:6]
# for i in false_neg:

false_neg = q2_df[(q2_df['stars'] == 5) & (q2_df['Sentiment'] ==␣
↪'Negative')]['text'][:5].tolist()

p, n = [], []

for review in false_neg:
    review_words = review.lower().split()
    for word in review_words:
        if word in negative_words:
            n.append(len(n))
        elif word in positive_words:
            p.append(len(p))

print(f"The number of positive words and negative words in these reviews are␣
↪{len(p)} and {len(n)}.")
print("The number of positive words and negative words in these reviews are {}␣
↪and {}".format(p, n))
```

```
The number of positive words and negative words in these reviews are 7 and 11.
The number of positive words and negative words in these reviews are [0, 1, 2,
3, 4, 5, 6] and [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

[ ]:
```python
false_neg
```

[ ]:
```
['Bad: No self parking.. had to reach down to flush toilet. Good: Location',
 "Bad: Toothpaste. I wish more hotels came with toothpaste in the bathroom . I
hate calling to get some. It's a necessity. That's my only complaint, but that's
pretty much for every hotel I've stayed in. Good: The bed was beyond
comfortable, didn't want to get up at all",
 "Das Mill's House Hotel ist fr mich ein typisches durchschnittliches
Stadthotel. Das Zimmer war eher klein aber mit guten Betten. Die Einrichtung war
hbsch, ausreichend und sauber. Da das Zimmer zum Innenhof lag war es sehr ruhig;
der Pool war geschlossen, obwohl es zum Baden noch warm genug gewesen wre.Das
Restaurant hatte die Grsse einer Bahnhofshalle und hat uns nicht zum Bleiben
'verfhrt'. Gerade nebenan gab es ein wundervolles Restaurant mit guter Bedienung
und gutem Essen.Die Bar bzw. die Lobby und auch der Innenhof mit seinem Brunnen
sieht gemtlich und einladend aus.Das Hotel liegt ideal fr alle Sehenswrdigkeiten
```

```
in Fussgnger-Nhe. Empfehlenswert ist der Gang ber die Kingsstreet mit seinen
vielfltigen Lden und der Besuch der Villengegend im French-Quarter. Nicht
besuchenswert ist der 'Market' - eine reine Touristenfalle.Wer nur kurz in
Charleston verweilt, kann dieses Hotel als Ausgangsstation buchen - ich kann es
empfehlen.Nebenbei: Savannah ist hnlich wie Charleston aber lebendiger und
vielfltiger. Ich wrde vermutlich eher in Savannah ein Hotel suchen und lnger
bleiben, und Charleston nur im Vorbeifahren besichtigen.",
 "The propose of this hotel is that you would fell like home. The main
differentiation from the Hampton inn for instance is that has a small door and a
separate living, so if you cook or simply pop a popcorn, the smell won't take
over the whole bedroom. I would recommend this hotel specially if you need to be
in… More",
 'It can be a little confusing to find in the dark without a GPS and the parking
is poorly signed behind the hotel (not the place next to the hotel). At 6:45 am
out took less than 10 mins to drive to the terminal. The hotel is very
comfortable and the staff all pleasant and helpful. They do have some… More']
```

Examining the reviews reveals a mix of positive and negative sentiments within them, such as one review containing both 'Bad' and 'Good'. Notably, there's also a review in German, for which we haven't applied the necessary preprocessing tools. Similarly, the remaining reviews display a combination of positive and negative language, highlighting a limitation of our dictionary approach: it doesn't grasp the context in which these words are used. The count of negative words in these reviews is comparable to, if not greater than, that of the positive words.

[ ]:
```python
false_pos = q2_df[(q2_df['stars'] == 1) & (q2_df['Sentiment'] ==␣
 ↪'Positive')]['text'][:5].tolist()
neg = []
pos = []

for review in false_pos:
    review_words = review.lower().split()
    for word in review_words:
        if word in negative_words:
            neg.append(word)
        if word in positive_words:
            pos.append(word)

print("The number of positive words and negative words in these reviews are {}␣
 ↪and {}.".format(len(pos), len(neg)))
print("The number of positive words and negative words in these reviews are {}␣
 ↪and {}".format(pos, neg))
```

```
The number of positive words and negative words in these reviews are 35 and 12.
The number of positive words and negative words in these reviews are ['right',
'gold', 'fine', 'good', 'extraordinarily', 'luxury', 'welcome', 'amenity',
'gold', 'polite', 'honor', 'available', 'grand', 'grand', 'available', 'worth',
'appreciate', 'worth', 'clearly', 'available', 'available', 'like', 'happy',
'loyalty', 'complimentary', 'nice', 'complimentary', 'worth', 'correct',
```

```
'thank', 'satisfied', 'super', 'ready', 'afford', 'well'] and ['hard', 'bad',
'worst', 'cold', 'marginally', 'unfortunately', 'lied', 'lying', 'deny',
'beware', 'lie', 'bothered']
```

[ ]:
```python
print(false_pos)
```

```
["on may26,2016 I pulled into parking lot with a flat tire I rented a room for 2
days one day cost 46.11 the 2nd day was57.11 the manger called the police and
had me removed for trying too change my tire. I am a woman I have a caste on my
right hand I couldn't change the tire on my… More", "Stayed 1 night as an SPG
Gold member through the Amex Platinum Fine Hotels Resorts program. The hard
product at the property is good but let down by extraordinarily bad service -
hands down the worst received at any 5* luxury hotel.- On reserving, the hotel
insisted that as a couple with a young child, we must book a rollaway bed at an
extra charge and we could not share one king bed for health and safety reasons.
Accepting the higher rate charge, on arrival, there was no rollaway provided.
Luckily we didn't want one but they were extremely insistent on this fact when
booking.- No welcome amenity offered as SPG Gold member.- Check-in experience
went from cold and marginally polite to confrontational. Hotel would not honor
the contractual one-category if available upgrade of the Amex FHR program. The
hotel's own website defines these room categories for sale: superior, deluxe,
grand luxe rooms, then 1 bedroom suites. Within the 1 bedroom suite category,
the room types are Astor, Deluxe, Madison. Having booked a grand luxe room, at
checkin their website showed availability for Deluxe and Madison suites. The
agent insisted their obligation was only a one-room type upgrade, and not a
category upgrade (which means any room type in that category). Therefore as no
Astor suite was available, no upgrade for us.- I spoke to the Front Desk
Manager, Susan, who insisted their interpretation of the FHR contract was
correct. Having now spent 20 mins waiting for them to check round the back as
they put it, with nowhere offered for my family to sit, I just wanted to get to
a room. I then called Amex, who contacted the hotel. Unfortunately by this point
Susan had allocated the available Deluxe suites to other guests or at least
blocked them, and lied to my Amex agent claiming that I had been seeking an
upgrade to a Madison suite worth 1000 extra, and the contract prohibits a triple
room type upgrade. I don't appreciate the hotel manipulating their inventory,
nor lying to my Amex agent when my request was for a Deluxe suite allowable by
the FHR contract and worth only approx 200-300 more. Amex advise the hotel
clearly has different definitions of available and category and these do not
equate to available for sale on their website and category of room as listed on
their website - sounds like they are happy to use different definitions to deny
loyalty benefits.- On being shown to our room, the agent advised a butler would
introduce himself in a couple of minutes. One hour later, no butler. I page the
butler and receive a call back, requesting a champagne ice bucket and child
amenities (these weren't provided even though child was listed on the
reservation). Another hour later, still no sign of anything. Page the butler
again, received the amenities… never received the requested ice bucket. Last
we ever saw of the butler. No offer of many of the complimentary services as
advertised on their website, but nice of them to provide tea cookies/milk.-
```

```
Ordered room service dinner (with the still missing ice bucket) - quality was
very average for the price and beware families, this hotel charges an 8 per
PERSON room service charge which I have never encountered at any other hotel. So
a family of 3 would pay a whopping 24 in addition to mandatory 20 gratuity and
sales tax. The waiter advised us just to lie and say we are just one person next
time! Called for the trays/trolley to be collected, 90mins later nobody had come
and it was collected by housekeeping at turndown.- Tried to organise family
activities through the concierge as advertised on their website, they
acknowledged our request but never heard back from them after that. - Advertised
functionality to stream your iPad media to the hotel TV does not work.- Included
complimentary breakfast worth 41 is the most basic continental breakfast ever
seen, consisting solely of 4 pastries, coffee and one orange juice. - On
checkout, the agent's only words from walking to the desk were Mr , here is your
bill, let me correct this error, hope to see you again. No how was your stay or
anything similar.When paying four figures to stay in a hotel room, this doesn't
meet any expectations and this stay was our first and last at a St Regis.Dear
msnz,Please allow us to thank you for posting your review on Trip Advisor. We
understand that our Director of Rooms has connected with you privately and we
hope you are satisfied with the resolution. We hope to be at your service again
soon and, until then, please contact our executive office for any additional
assistance you may need.Kind Regards,Hermann ElgerGeneral Manager", 'When I
checked in for my stay, the clerk could not even be bothered to get up out of
her chair to do my paperwork. My room did not even have a shower curtain on my
first night. The Super 8 was better.', "I was quoted a price over the phone of
383 weekly and even used my credit card to hold the room. On the day of check in
i arrived and my room wasn't even ready even after 5pm then I was told that room
would be over 500 per week and i couldn't afford that. All the man at the…
More", 'There was a mix up with our room and it was not handled well at all. We
returned to change clothes before dinner only to find all of our things had been
packed up and were sitting behind the front desk!']
```

Upon examining the reviews that were inaccurately labeled as positive despite having a 1-star rating, it's evident that the prevalence of positive terms surpasses that of negative ones, potentially leading to their misclassification by our model. Another contributing factor could be errors made by the users themselves, such as inadvertently selecting a 1-star rating.

4 Q 4

[ ]:
```
q4_df= q2_df
q4_df
```

[ ]:
```
      stars                                               text  label  \
0         4  As far as the room it self I would actually sa…      1
1         5  Wonderful as always, the consistent level of q…      1
2         4  Location was very convenient, although using t…      1
3         4  We were very tired when we arrived with a cran…      1
4         3  Somewhat updated, nice fireplace and clean. We…      0
```

```
…      …                                                   …    …
7495      5  We stayed here for the third time for our 5th …      1
7496      5  Friendly staff, wood floors in room. place see…      1
7497      5  stayed here the weekend of the Rock and Roll N…      1
7498      3  We stayed here to attend Kentucky Derby and sp…      0
7499      3  The sweeper hadnt been ran in our room when we…      0

     Sentiment
0     Positive
1     Positive
2     Positive
3     Positive
4     Positive
…         …
7495  Positive
7496  Positive
7497  Positive
7498  Positive
7499  Negative

[7500 rows x 4 columns]
```

[ ]:
```python
def q4_preprocess(doc):
  text = re.sub(r'\\x[0-9A-Fa-f]{2}', ' ', doc)
  text = re.sub(r'[0-9]',' ',text)
  text = text.translate(str.maketrans('', '', string.punctuation)).lower()

  tokens = word_tokenize(text)
  stop_words = set(stopwords.words("english"))
  nostop = [word for word in tokens if word not in stop_words]
  output = [PorterStemmer().stem(word) for word in nostop]
  return " ".join(output)

q4_new_df = q4_df['text']

q4_preproc = list(map(q4_preprocess, q4_new_df.apply(lambda x:str(x))))
```

[ ]:
```python
q4_tr_df=q4_df
q4_tr_df['text']=q4_preproc
q4_tr_df
```

[ ]:
```
      stars                                              text  label  \
0         4  far room self would actual say breakfast kind …      1
1         5  wonder alway consist level qualiti alway reass…      1
2         4  locat conveni although use free shuttl metro s…      1
3         4  tire arriv cranki babi pierr front desk help g…      1
4         3  somewhat updat nice fireplac clean book king j…      0
```

```
…      …                                                   …    …
7495      5  stay third time th anniversari hotel definit e…      1
7496      5  friendli staff wood floor room place seem rece…      1
7497      5  stay weekend rock roll new orlean marathon sta…      1
7498      3  stay attend kentucki derbi spent littl time ho…      0
7499      3  sweeper hadnt ran room got turn ac get cool ro…      0

     Sentiment
0     Positive
1     Positive
2     Positive
3     Positive
4     Positive
…         …
7495  Positive
7496  Positive
7497  Positive
7498  Positive
7499  Negative

[7500 rows x 4 columns]
```

##4a

[ ]:
```python
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
train_df, test_df = train_test_split(q4_df, test_size=0.2, random_state=42)
X_tr, y_tr = train_df['text'], train_df['Sentiment']
X_test, y_ts = test_df['text'], test_df['Sentiment']
```

[ ]:
```python
X_tr = X_tr.fillna('')
X_test = X_test.fillna('')
q4_vectorizer = CountVectorizer()
q4_nb_classifier = MultinomialNB()
q4_X_train_vectorized = q4_vectorizer.fit_transform(X_tr)
q4_X_test_vectorized = q4_vectorizer.transform(X_test)
```

[ ]:
```python
q4_nb_classifier.fit(q4_X_train_vectorized, y_tr)
```

[ ]:
```
MultinomialNB()
```

[ ]:
```python
y_pred_nb = q4_nb_classifier.predict(q4_X_test_vectorized)
```

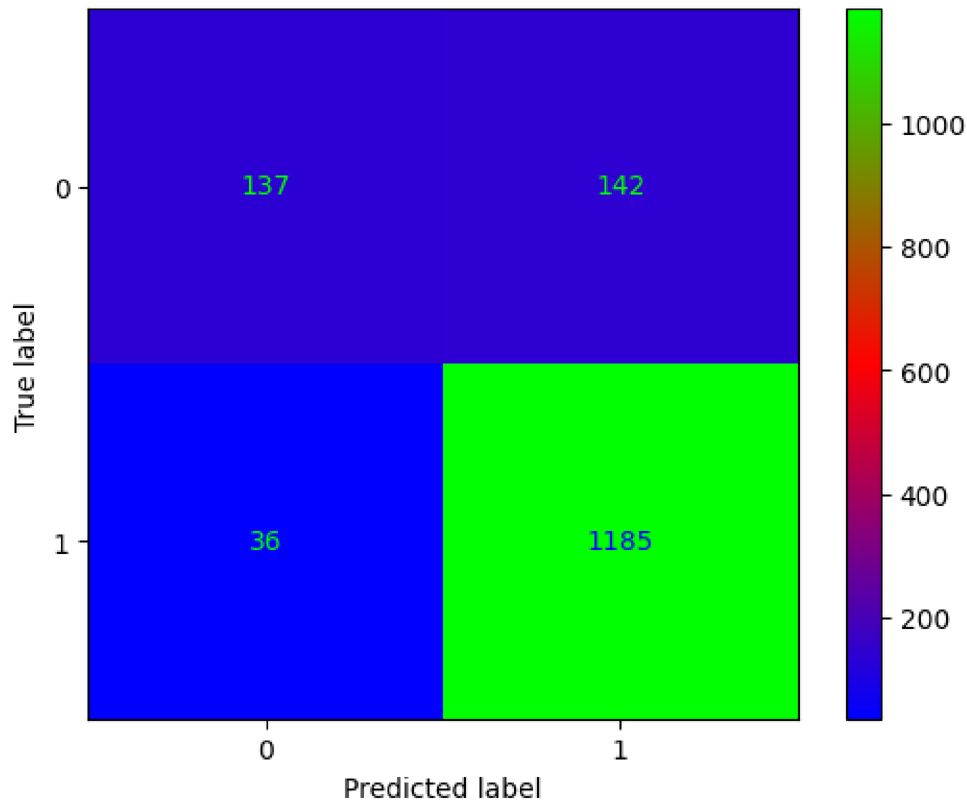
[ ]:
```python
# Calculate accuracy
accuracy_nb = accuracy_score(y_ts, y_pred_nb)
```

[ ]:
```python
accuracy_nb
```

[ ]: 0.8813333333333333

[ ]:
```
conf_nb=confusion_matrix(y_ts, y_pred_nb)
d_nb=ConfusionMatrixDisplay(conf_nb)
d_nb.plot(cmap='brg')
```

[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7c7ac4b94a60>



[ ]:
```
from sklearn.metrics import recall_score

recall = recall_score(y_ts, y_pred_nb, pos_label='Positive')

print("Recall Score:", recall)
```

Recall Score: 0.9705159705159705

[ ]:
```
f1_score(y_ts, y_pred_nb, pos_label='Positive')
```

[ ]: 0.9301412872841444

[ ]:
```
precision_score(y_ts, y_pred_nb, pos_label='Positive')
```

[ ]: 0.892991710625471

##4b

N-grams: Instead of only using individual words (unigrams), including bigrams (pairs of consecutive words) or trigrams (triplets of consecutive words) as features can capture more context and the relationship between words, which can be very useful for understanding sentiment.

Part-of-Speech Tags: Certain parts of speech, such as adjectives or adverbs, are often more closely tied to sentiment than others. Including the parts of speech of words as features can help the model identify sentiment-bearing terms more effectively.

Sentiment Lexicons: There are pre-compiled lists of words associated with positive or negative sentiments (e.g., SentiWordNet, VADER). Features could include counts of words in the document that appear in these lists, or aggregate scores based on these lists.

- Using N-grams like uni, bi or trigrams can help us extract more information and predict relationship between words.
- We can use parts of speech tagging. Some parts of speech like adjectives or adverbs are more tied into sentiment than others.
- Apart from this we can use word embedding with context with many options available like Word2vec or Bert.

5 5

[ ]:
```
q4_df['Sentiment'][:1001].value_counts()
```

[ ]:
```
Positive    812
Negative    189
Name: Sentiment, dtype: int64
```

[ ]:
```
def q5_preprocess(doc):
  text = re.sub(r'\\x[0-9A-Fa-f]{2}', ' ', doc)
  text = re.sub(r'[0-9]',' ',text)
  text = text.translate(str.maketrans('', '', string.punctuation)).lower()

  tokens = word_tokenize(text)
  stop_words = set(stopwords.words("english"))
  nostop = [word for word in tokens if word not in stop_words]
  output = [PorterStemmer().stem(word) for word in nostop]
  return " ".join(output)


q5_new_df = q4_df[:1000]

q5_preproc = list(map(q4_preprocess, q5_new_df['text'].apply(lambda x:str(x))))
q5_new_df['text']=q5_preproc
```

```
<ipython-input-80-ed880485e788>:16: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
```

```
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  q5_new_df['text']=q5_preproc
```

[ ]:
```
q5_new_df
```

[ ]:
```
     stars                                               text  label Sentiment
0        4  far room self would actual say breakfast kind …      1  Positive
1        5  wonder alway consist level qualiti alway reass…      1  Positive
2        4  locat conveni although use free shuttl metro s…      1  Positive
3        4  tire arriv cranki babi pierr front desk help g…      1  Positive
4        3  somewhat updat nice fireplac clean book king j…      0  Positive
..     …                                                …    …       …
995      5  pleasant hotel sout side town far colleg far e…      1  Positive
996      4  pleasent surpri qualiti room locat quit liter …      1  Positive
997      4  wasnt expect much hotel due low rate howev sta…      1  Positive
998      1  eaten mani mcmenamin son birthday chose one mc…      0  Negative
999      5  famili oper owner see need dan jenson even hel…      1  Positive

[1000 rows x 4 columns]
```

[ ]:
```python
from sklearn.svm import SVC
svm_classifier=SVC(kernel='linear')
```

[ ]:
```
q5_new_df
```

[ ]:
```
     stars                                               text  label Sentiment
0        4  far room self would actual say breakfast kind …      1  Positive
1        5  wonder alway consist level qualiti alway reass…      1  Positive
2        4  locat conveni although use free shuttl metro s…      1  Positive
3        4  tire arriv cranki babi pierr front desk help g…      1  Positive
4        3  somewhat updat nice fireplac clean book king j…      0  Positive
..     …                                                …    …       …
995      5  pleasant hotel sout side town far colleg far e…      1  Positive
996      4  pleasent surpri qualiti room locat quit liter …      1  Positive
997      4  wasnt expect much hotel due low rate howev sta…      1  Positive
998      1  eaten mani mcmenamin son birthday chose one mc…      0  Negative
999      5  famili oper owner see need dan jenson even hel…      1  Positive

[1000 rows x 4 columns]
```

##5a

- The Naive Bayes classifier stands out for its unexpected efficiency in reviewing classification, attributing its success to the application of prior probabilities. This method leverages the actual distribution of words within the reviews to make more informed decisions, while also being computationally efficient.

- SVM (Support Vector Machine) offers a substantial enhancement over the basic dictionary approach by not merely counting words based on their sentiment. Instead, it identifies the optimal hyperplane to segregate the reviews, effectively managing word interactions and assigning higher scores to terms that are strongly indicative of sentiment.
- Contrasting with the earlier dictionary-based methods, which rely on simple counts of positive and negative words, both the SVM and Naive Bayes techniques provide significant advancements in accurately classifying review sentiments.

##5b

[ ]:
```
svm_classifier = Pipeline([
    ('tfidf', TfidfVectorizer()),  # Convert text data to TF-IDF features
    ('svm', SVC(kernel='linear'))  # Linear SVM classifier
])
```

[ ]:
```
train_df
```

[ ]:
```
      stars                                             text  label  \
4664      5  bad noth good locat size room cocktail loung p…      1
4411      5  great work kristen even met sent idea like met…      1
7448      4                nice good locat staff friendli help      1
1919      4  place nice howev room price still make pay ext…      1
1298      5  couldnt nicer front desk courtesi check clean …      1
…       …                                              …    …
5191      5  hilton garden inn perfect strateg locat citi a…      1
5226      5  good great locat extrem comfort bed pillow won…      1
5390      5  awesom hotel breakfast unbeat love fact realli…      1
860       5  bad wifi chargeabl good locat casino cheap foo…      1
7270      5  weve visit dvr multipl time past year weve ple…      1

     Sentiment
4664  Negative
4411  Positive
7448  Positive
1919  Positive
1298  Positive
…          …
5191  Positive
5226  Positive
5390  Positive
860   Negative
7270  Positive

[6000 rows x 4 columns]
```

[ ]:
```
from sklearn.model_selection import KFold
num_folds = 5
final_accuracy = {}
```

```python
for test_size_percentage in range(90, 0, -10):
    train_df, test_df = train_test_split(q5_new_df, test_size=test_size_percentage/100, random_state=42)
    X_train, y_train = train_df['text'], train_df['label']
    X_test, y_test = test_df['text'], test_df['label']

    kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)

    scores_fold = []
    scores_val = []

    for train_index, test_index in kf.split(X_train):
        svm_classifier.fit(X_train.iloc[train_index], y_train.iloc[train_index])
        y_pred_fold = svm_classifier.predict(X_train.iloc[test_index])
        accuracy_fold = accuracy_score(y_train.iloc[test_index], y_pred_fold)
        scores_fold.append(accuracy_fold)

        y_pred_val = svm_classifier.predict(X_test)
        accuracy_val = accuracy_score(y_test, y_pred_val)
        scores_val.append(accuracy_val)

    test_split_str = f"{test_size_percentage}% Test Split"
    print(f"The train has finished for {test_split_str}")
    final_accuracy[f"{100 - test_size_percentage}% Train Split"] = np.mean(scores_val)
    print(f"The Accuracy of {test_split_str} = {np.mean(scores_val)}")
```

```
The train has finished for 90% Test Split
The Accuracy of 90% Test Split = 0.7575555555555555
The train has finished for 80% Test Split
The Accuracy of 80% Test Split = 0.7765000000000001
The train has finished for 70% Test Split
The Accuracy of 70% Test Split = 0.792
The train has finished for 60% Test Split
The Accuracy of 60% Test Split = 0.7993333333333333
The train has finished for 50% Test Split
The Accuracy of 50% Test Split = 0.798
The train has finished for 40% Test Split
The Accuracy of 40% Test Split = 0.8025
The train has finished for 30% Test Split
The Accuracy of 30% Test Split = 0.7979999999999998
The train has finished for 20% Test Split
The Accuracy of 20% Test Split = 0.842
The train has finished for 10% Test Split
The Accuracy of 10% Test Split = 0.834
```

[ ]:
```
feature_names = svm_classifier[0].get_feature_names_out()
coef  = svm_classifier[1].coef_.toarray().flatten()

top_positive_indices = coef.argsort()[-5:]
top_negative_indices = coef.argsort()[:5]

top_positive_words = feature_names[top_positive_indices]
top_negative_words = feature_names[top_negative_indices]
```

##5c

[ ]:
```
top_positive_words
```

[ ]:
```
array(['nice', 'right', 'love', 'comfort', 'great'], dtype=object)
```

[ ]:
```
top_negative_words
```

[ ]:
```
array(['dirti', 'worst', 'smell', 'check', 'disappoint'], dtype=object)
```

5.1 5d

[ ]:
```
def q5_d_preprocess(doc):
  text = re.sub(r'\\x[0-9A-Fa-f]{2}', ' ', doc)
  text = re.sub(r'[0-9]',' ',text)
  text = text.translate(str.maketrans('', '', string.punctuation)).lower()

  tokens = word_tokenize(text)
  stop_words = set(stopwords.words("english"))
  nostop = [word for word in tokens if word not in stop_words]
  output = [PorterStemmer().stem(word) for word in nostop]
  return " ".join(output)

q5_d_pre = list(map(q5_d_preprocess, q5_new_df['text'].apply(lambda x:str(x))))
q5_bigram_finder = BigramCollocationFinder.from_words(word_tokenize(" ".
 ↪join(q5_d_pre)))
```

[ ]:
```
svm_classifier_5d = Pipeline([
    ('tfidf', TfidfVectorizer(ngram_range=(1,2))),  # Convert text data to␣
 ↪TF-IDF features
    ('svm', SVC(kernel='linear'))  # Linear SVM classifier
])
```

[ ]:
```
from sklearn.model_selection import KFold
num_folds = 5
final_accuracy = {}

for i in range(10, 100,10):
```

```
    train_df, test_df = train_test_split(q5_new_df, test_size=i/100,␣
↪random_state=42)

    kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)
    index = kf.split(train_df['text'])
    scores_fold = []
    scores_val = []
    for _, (train_index, test_index) in enumerate(index):
      # print(train_index)
        svm_classifier_5d.fit(train_df.iloc[train_index]['text'], train_df.
↪iloc[train_index]['Sentiment'])
        y_pred_fold = svm_classifier_5d.predict(train_df.
↪iloc[test_index]['text'])
        accuracy_fold = accuracy_score(train_df.iloc[test_index]['Sentiment'],␣
↪y_pred_fold)
        scores_fold.append(accuracy_fold)
        y_pred_val = svm_classifier_5d.predict(test_df['text'])
        accuracy_val = accuracy_score(test_df['Sentiment'], y_pred_val)
        scores_val.append(accuracy_val)
    print(f"The train has finished for {str(i) + '% Train Split'}")

    kfold_accuracy = np.mean(scores_fold)
    final_accuracy[str(i) + "% Test Split"] = np.mean(scores_val)
    print(f"The Accuracy of {str(i) + '% Train Split'} = {np.mean(scores_val)}")
```

```
The train has finished for 10% Train Split
The Accuracy of 10% Train Split = 0.852
The train has finished for 20% Train Split
The Accuracy of 20% Train Split = 0.8479999999999999
The train has finished for 30% Train Split
The Accuracy of 30% Train Split = 0.8119999999999999
The train has finished for 40% Train Split
The Accuracy of 40% Train Split = 0.8045
The train has finished for 50% Train Split
The Accuracy of 50% Train Split = 0.7968000000000001
The train has finished for 60% Train Split
The Accuracy of 60% Train Split = 0.8053333333333332
The train has finished for 70% Train Split
The Accuracy of 70% Train Split = 0.8191428571428572
The train has finished for 80% Train Split
The Accuracy of 80% Train Split = 0.8092500000000001
The train has finished for 90% Train Split
The Accuracy of 90% Train Split = 0.8128888888888888
```

The model's performance has declined in comparison to the results seen in 5b, indicating that incorporating all bigrams adversely affects its accuracy. It suggests that a more selective approach, possibly focusing on the 'n' most significant bigrams, could potentially enhance the model's effectiveness.

#Q6

```
[ ]: def q6_preprocess(doc):
       text = re.sub(r'\\x[0-9A-Fa-f]{2}', ' ', doc)
       text = re.sub(r'[0-9]',' ',text)
       text = text.translate(str.maketrans('', '', string.punctuation)).lower()

       tokens = word_tokenize(text)
       stop_words = set(stopwords.words("english"))
       nostop = [word for word in tokens if word not in stop_words]
       output = [PorterStemmer().stem(word) for word in nostop]
       return " ".join(output)

     q6_df=q4_df.sample(n=500)
     q6_new_df = q6_df['text']

     q6_preproc = list(map(q6_preprocess, q6_new_df.apply(lambda x:str(x))))

     q6_tr_df=q4_df
     q6_df['text']=q6_preproc
```

##6a

```
[ ]: q6_train_df, q6_test_df = train_test_split(q6_df, test_size=0.2,␣
      ↪random_state=42)
     q6_X_tr, q6_y_tr = train_df['text'], train_df['Sentiment']
     q6_X_ts, q6_y_ts = test_df['text'], test_df['Sentiment']
```

```
[ ]: q6_svm_classifier = Pipeline([
         ('tfidf', TfidfVectorizer()),  # Convert text data to TF-IDF features
         ('svm', SVC(kernel='linear'))  # Linear SVM classifier
     ])
```

```
[ ]: q6_svm_classifier.fit(q6_X_tr, q6_y_tr)
```

```
[ ]: Pipeline(steps=[('tfidf', TfidfVectorizer()), ('svm', SVC(kernel='linear'))])
```

```
[ ]: q6_feature_names = q6_svm_classifier[0].get_feature_names_out()
     q6_coef  = q6_svm_classifier[1].coef_.toarray().flatten()

     q6_top_positive_indices = q6_coef.argsort()[-10:]
     q6_top_negative_indices = q6_coef.argsort()[:10]

     q6_top_positive_words = q6_feature_names[q6_top_positive_indices]
     q6_top_negative_words = q6_feature_names[q6_top_negative_indices]
```

```
[ ]: print("The 10 most important features are {}".format(q6_top_positive_words))
```

```
The 10 most important features are ['good' 'room' 'well' 'clean' 'friendli'
```

'nice' 'wonder' 'comfort' 'great'
 'staff']

##6b

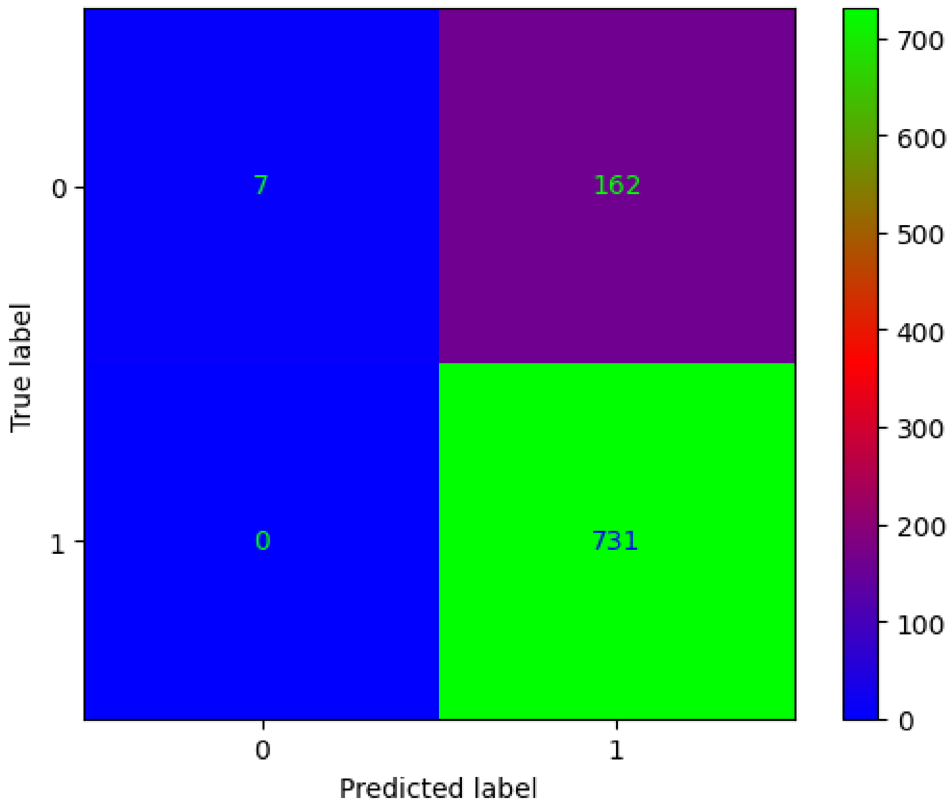
[ ]:
```
q6_ypred=q6_svm_classifier.predict(q6_X_ts)
```

[ ]:
```
q6_accuracy = accuracy_score(q6_y_ts, q6_ypred)
q6_precision=precision_score(q6_y_ts, q6_ypred, pos_label='Positive')
q6_f1=f1_score(q6_y_ts, q6_ypred, pos_label='Positive')
q6_recall=recall_score(q6_y_ts, q6_ypred, pos_label='Positive')
print("The accuracy is {}".format(q6_accuracy))
print("The precision is {}".format(q6_precision))
print("The F1 score is {}".format(q6_f1))
print("The recall is {}".format(q6_recall))
```

```
The accuracy is 0.82
The precision is 0.8185890257558791
The F1 score is 0.9002463054187193
The recall is 1.0
```

[ ]:
```
q6_conf=confusion_matrix(q6_y_ts, q6_ypred)
q6_d=ConfusionMatrixDisplay(q6_conf)
q6_d.plot(cmap='brg')
```

[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f13c7339660>
```



##6c

```
[ ]: c_options = [0.1, 0.5, 1]
     results = []
     for c_value in c_options:
         start_time = time.time()
         svm_model = Pipeline([
             ('tfidf_transformer', TfidfVectorizer()),
             ('svm_classifier', SVC(kernel='linear', C=c_value))
         ])
         svm_model.fit(X_train, y_train)
         predictions = svm_model.predict(X_test)
         accuracy = accuracy_score(y_test, predictions)
         end_time = time.time()
         execution_time = end_time - start_time
         results.append((c_value, accuracy, execution_time))

     for result in results:
         print(f"The value of C = {result[0]} resulted in an accuracy of {result[1]}␣
      ↪with a computation time of {result[2]} seconds")
```

```
The value of c = 0.1 had an accuracy of 0.82 and a computation time of
```

```
0.06960582733154297
The value of c = 0.5 had an accuracy of 0.82 and a computation time of
0.05773019790649414
The value of c =1 had an accuracy of 0.82 and a computation time of
0.060431480407714844
```

All C values resulted in identical accuracy levels, however, a C value of 0.5 offered the optimal computational efficiency.

##6d

[ ]:
```
kernel_types = ['linear', 'rbf', 'poly']
svm_results = []
for kernel in kernel_types:
    svm_pipeline = Pipeline([
        ('tfidf_vectorizer', TfidfVectorizer()),
        ('svm_model', SVC(kernel=kernel, C=1))
    ])
    svm_pipeline.fit(X_train, y_train)
    predictions = svm_pipeline.predict(X_test)
    accuracy = accuracy_score(y_test, predictions)
    svm_results.append((kernel, accuracy))

for kernel_result in svm_results:
    print(f"The {kernel_result[0]} kernel achieved an accuracy of␣
 ↪{kernel_result[1]}")
```

```
The linear kernel had an accuracy of 0.82
The rbf kernel had an accuracy of 0.8122222222222222
The poly kernel had an accuracy of 0.8122222222222222
```

Linear kernal has better accuracy