Solution_2506673

July 19, 2025

CMM706 - Text Analytics Coursework

Kaneel Dias
IIT ID: 20242001
RGU ID: 2506673

[1]:
```
import nltk
nltk.download('stopwords')
nltk.download('punkt_tab')
from nltk.corpus import stopwords
from nltk import ngrams
from textblob import TextBlob
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
from transformers import pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
import torch
from keras import Sequential, Input
from keras.src.layers import Dense, Dropout
from keras.src.utils import to_categorical
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from transformers import BertTokenizer, BertModel
from sklearn.model_selection import cross_val_predict, KFold
from sklearn.naive_bayes import MultinomialNB
import numpy as np
import pandas as pd
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.feature_extraction.text import CountVectorizer
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,␣
↪rand_score, jaccard_score
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score
from sklearn.model_selection import cross_val_predict
from sklearn.svm import SVC
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\kanee\AppData\Roaming\nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt_tab to
[nltk_data]     C:\Users\kanee\AppData\Roaming\nltk_data…
[nltk_data]   Package punkt_tab is already up-to-date!
C:\Users\kanee\.virtualenvs\sri-lanka-hotel-reviews\Lib\site-
packages\tqdm\auto.py:21: TqdmWarning: IProgress not found. Please update
jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

Task 1 - Describe the dataset

The data is saved in a CSV file, `reviews.csv`. We can read this file and display the first few rows to understand its structure and contents.

[2]:
```python
reviews = pd.read_csv("reviews.csv")
reviews.head()
```

[2]:
```
    review_id                                         review_url  location_id  \
0  1016464488  https://www.tripadvisor.com/ShowUserReviews-g2…     11953119
1  1016435128  https://www.tripadvisor.com/ShowUserReviews-g2…     11953119
2  1016307864  https://www.tripadvisor.com/ShowUserReviews-g2…     11953119
3  1016165618  https://www.tripadvisor.com/ShowUserReviews-g2…     11953119
4  1015472232  https://www.tripadvisor.com/ShowUserReviews-g2…     11953119

              hotel_name     city             timestamp  rating  \
0  Nh Collection Colombo  Colombo  2025-07-04T05:58:58Z       1
1  Nh Collection Colombo  Colombo  2025-07-04T01:38:54Z       5
2  Nh Collection Colombo  Colombo  2025-07-03T05:15:10Z       5
3  Nh Collection Colombo  Colombo  2025-07-02T06:36:14Z       5
4  Nh Collection Colombo  Colombo  2025-06-28T00:50:47Z       5

                                 title  \
0                     not a good stay
1                Definitely recommend!
2                       Wonderful stay
3  My favorite 4+ star hotel in Colombo
4                 Excellent food and stay

                                              text travel_date  \
0  Found lighters in the toilet paper rolls in a …  2025-06-30
1  The hotel is just excellent! The food is so go…  2025-07-31
2  Comfortable stay..cooperative staff..fast serv…  2025-07-31
3  We live in New York area, but my spouse is fam…  2025-07-31
4  Excellent food especially indian corner lot of…  2025-06-30
```

```
          username  value_rating  room_rating  location_rating  \
0  Curious04015869441           NaN          NaN              NaN
1        742saltanata           5.0          5.0              5.0
2         847shivanig           5.0          5.0              5.0
3         jacksF8984QN           NaN          NaN              NaN
4         Vinayaksahni           5.0          5.0              5.0

   cleanliness_rating  service_rating  sleep_rating
0                 NaN             NaN           NaN
1                 5.0             5.0           5.0
2                 5.0             4.0           5.0
3                 NaN             NaN           NaN
4                 5.0             5.0           5.0
```

1.1. Nature of the data

As it can be seen, the dataset contains a collection of hotel reviews with the following columns: - `review_id`: Unique identifier for each review (as provided by Trip Advisor). - `review_url`: URL of the review on Trip Advisor. - `location_id`: Unique identifier for the hotel (as provided by Trip Advisor). - `hotel_name`: Name of the hotel. - `city`: City where the hotel is located. - `timestamp`: Date and time when the review was posted. - `rating`: Rating given by the reviewer (1 to 5 stars). - `title`: Title of the review. - `text`: Full text of the review. - `travel_date`: Date when the reviewer stayed at the hotel. - `value_rating`: Rating for the value for money. - `room_rating`: Rating for the room. - `location_rating`: Rating for the location. - `cleanliness_rating`: Rating for the cleanliness. - `service_rating`: Rating for the service. - `sleep_rating`: Rating for the sleep quality.

We can check the shape of the data to understand how many rows and columns are present in the dataset.

[3]:
```
reviews.shape
```

[3]:
```
(7367, 17)
```

As seen above, the dataset contains 7,367 reviews, with each review having 15 columns.

We can also check how many unique values are present in each column to understand the nature of the data.

[4]:
```
reviews.nunique()
```

[4]:
```
review_id            5186
review_url           5186
location_id           566
hotel_name            563
city                  161
timestamp            5186
rating                  5
title                4691
```

```
text                   5186
travel_date             181
username               5019
value_rating              5
room_rating               5
location_rating           5
cleanliness_rating        5
service_rating            5
sleep_rating              5
dtype: int64
```

We can see that for most columns, there are exactly 5,186 unique columns, including for `review_id` this indicates that our dataset contains a significant number of duplicate reviews.

Aside from that, we can infer some additional information from the above. - This dataset comprises reviews from 566 different hotels, each with its own unique `location_id`. - There are 161 distinct cities represented in the dataset, each with its own unique `city` name. - There are only 5 unique values for `rating`, which indicates that the ratings are on a 1 to 5 scale. - There are 5,019 unique values for `username`, indicating that as many users have contributed multiple reviews.

1.2. Removing duplicates

Since the dataset contains duplicate reviews, we can remove them to ensure that each review is unique.

[5]:
```
reviews = reviews.drop_duplicates()
reviews.shape
```

[5]:
```
(5186, 17)
```

After removing duplicates, we have 5,186 unique reviews in the dataset.

1.2 Distribution of reviews by hotel

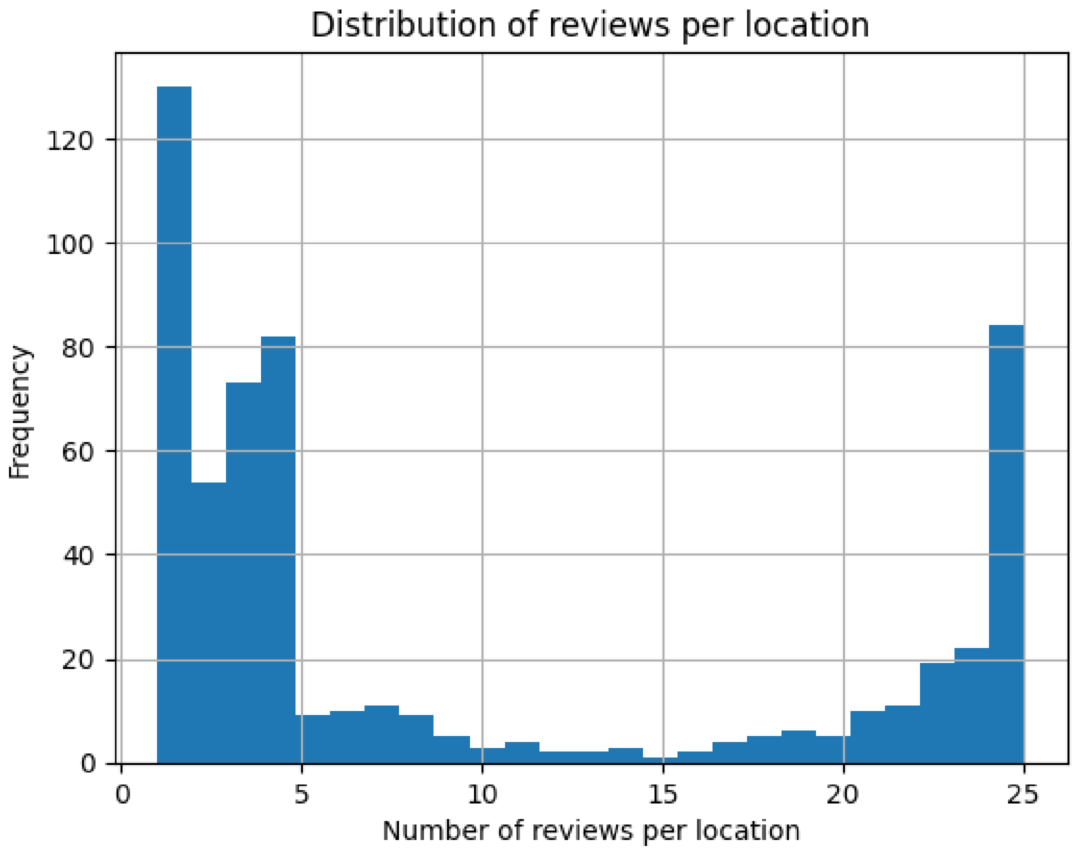
First, we can check the number of unique hotels in the dataset.

[6]:
```
reviews["location_id"].unique().shape[0]
```

[6]:
```
566
```

Next, we can visualize the number of reviews per hotel to understand the distribution of reviews across different hotels. We can use a histogram to visualize this distribution.

[7]:
```
reviews["location_id"].value_counts().hist(bins=25)

plt.xlabel("Number of reviews per location")
plt.ylabel("Frequency")
plt.title("Distribution of reviews per location")
plt.show()
```



Here, we can see that most hotels have a small number of reviews. Most hotels have between 1 and 5 reviews. This may due to a quirk of the Trip Advisor API, which only returns the 5 reviews per API request. You would need to make additional API calls with a different offset to get more reviews for each hotel.

We can see a sharp drop-off after 5 reviews. This also may have something do with the API.

Finally, we can see a sharp spike at 25 reviews, and no hotels have more than 25 reviews. This is likely due to the API returning a maximum of 25 reviews per hotel.

1.3. Distribution of reviews by city

First, we can check the number of unique cities in the dataset.

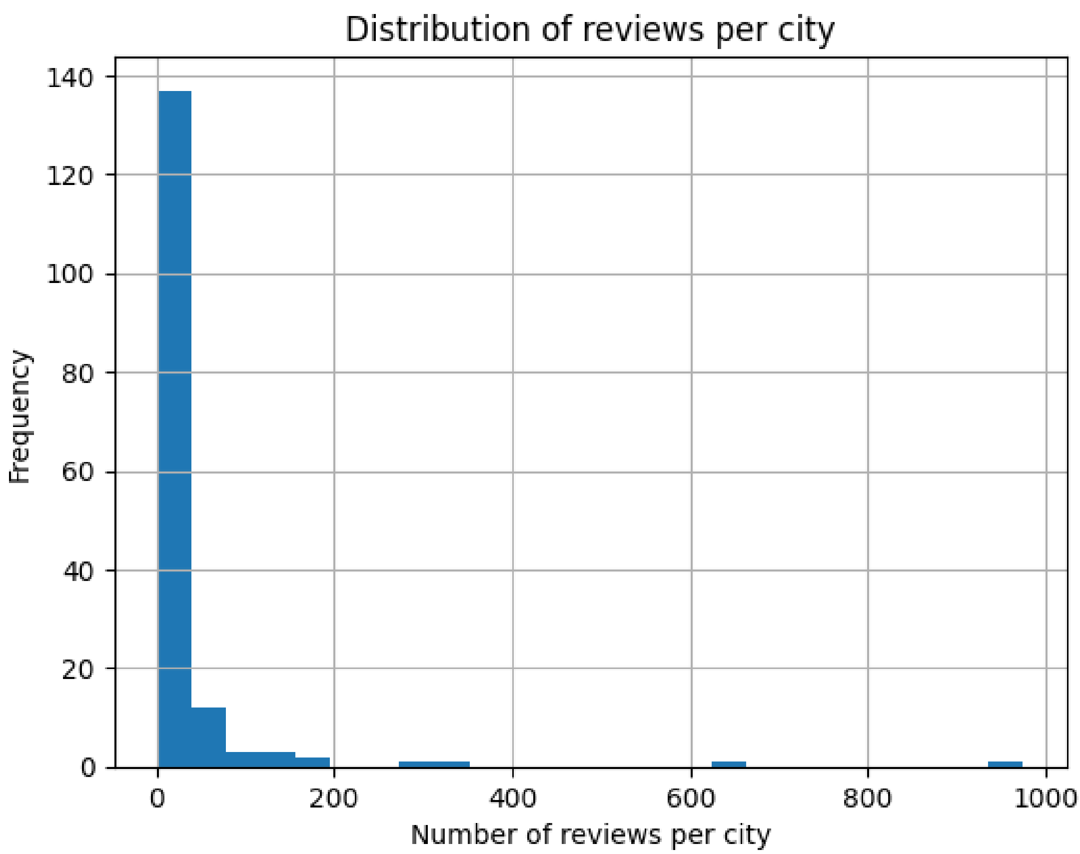
[8]:
```
reviews["city"].unique().shape[0]
```

[8]: 161

Next, we can visualize the number of reviews per city to understand the distribution of reviews across different cities. We can use a histogram to visualize this distribution.

[9]:

```
reviews["city"].value_counts().hist(bins=25)

plt.xlabel("Number of reviews per city")
plt.ylabel("Frequency")
plt.title("Distribution of reviews per city")
plt.show()
```



Here once again, we see a very unbalanced distribution of reviews across cities. Most cities have a small number of reviews, with a few cities having a large number of reviews.

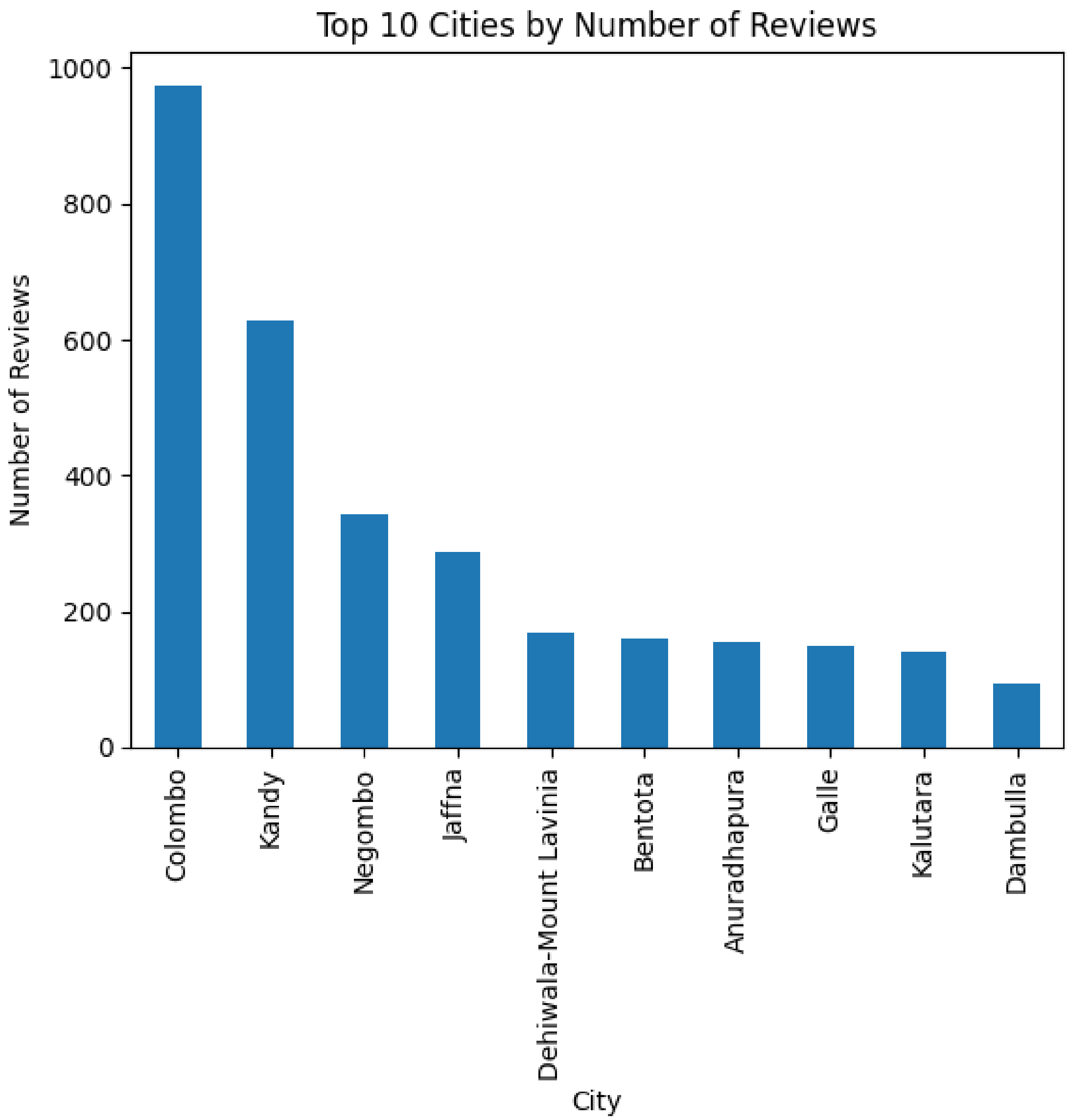
This can be explained by the fact that a few cities are more tourist/business hotspots. And a majority of the hotels as well as visitos are located in these cities.

We can also visualize the top 10 cities with the most reviews to get a better understanding of the distribution.

[10]:

```
top_cities = reviews["city"].value_counts().head(10)

top_cities.plot(kind='bar')
plt.xlabel("City")
plt.ylabel("Number of Reviews")
```

```
plt.title("Top 10 Cities by Number of Reviews")
plt.show()
```

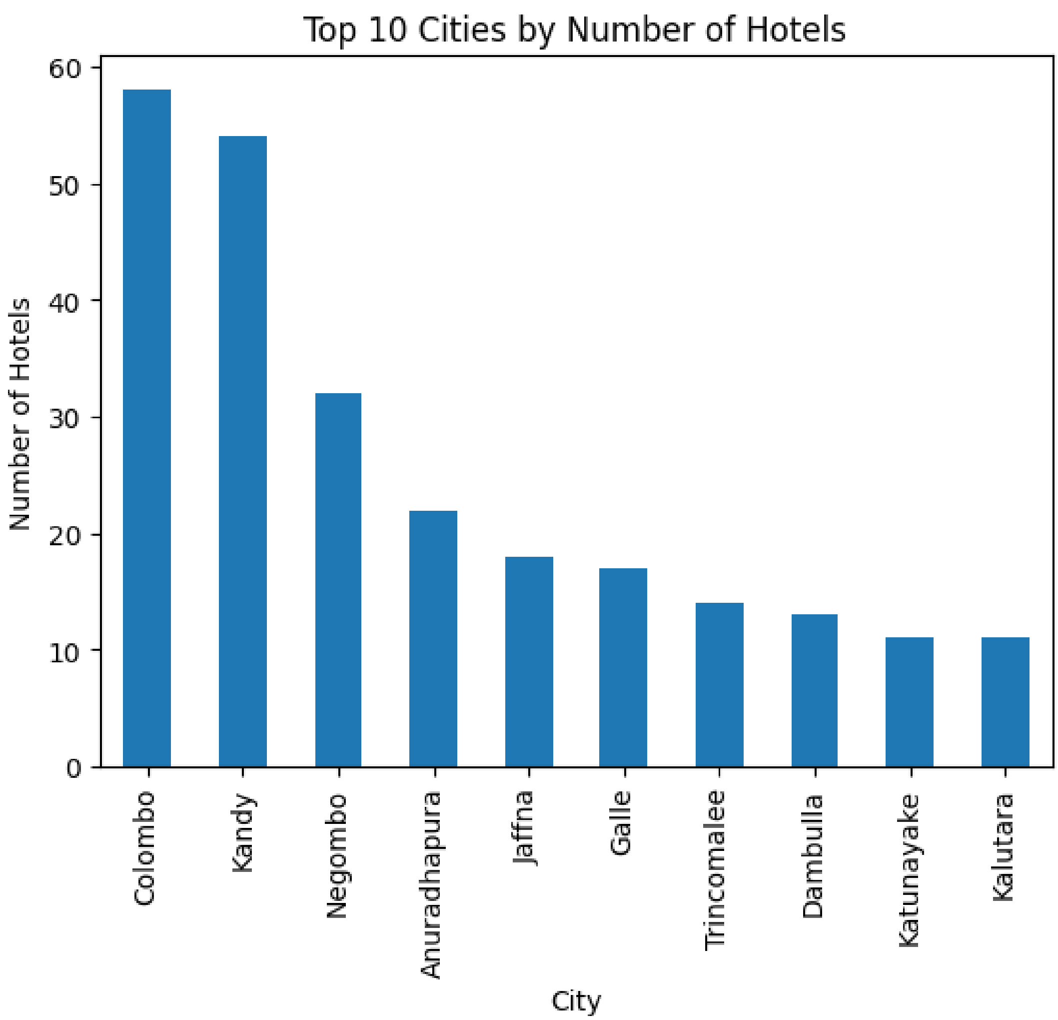


Here, we can see the most populous cities in Sri Lanka are also the ones with the most reviews. Another factor affecting this may be how the data was collected using the API.

As there were several interruptions/issues with the API, it is possible that the data collection was not uniform across all cities.

And now, we can visualize the number of unique hotels in each city to understand the distribution of hotels across different cities.

```
[11]: hotels_per_city = reviews.groupby("city")["location_id"].nunique().
      ↪sort_values(ascending=False)
      hotels_per_city.head(10).plot(kind='bar')
      plt.xlabel("City")
      plt.ylabel("Number of Hotels")
      plt.title("Top 10 Cities by Number of Hotels")
      plt.show()
```

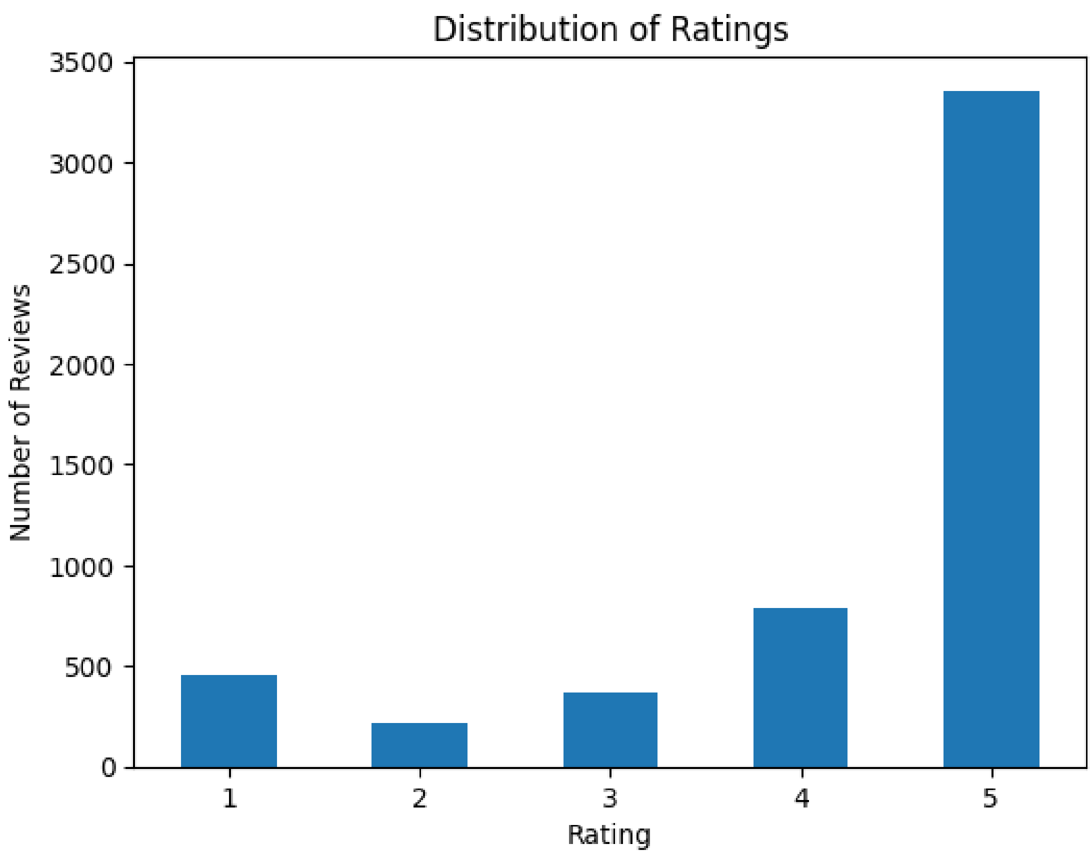


This seems to follow the same pattern as the number of reviews per city. The most populous cities have the most hotels.

1.4. Distribution of ratings

We can now move onto checking the distribution of ratings in the dataset.

[12]:
```
reviews["rating"].value_counts().sort_index().plot(kind='bar')
plt.xlabel("Rating")
plt.ylabel("Number of Reviews")
plt.title("Distribution of Ratings")
plt.xticks(rotation=0)
plt.show()
```



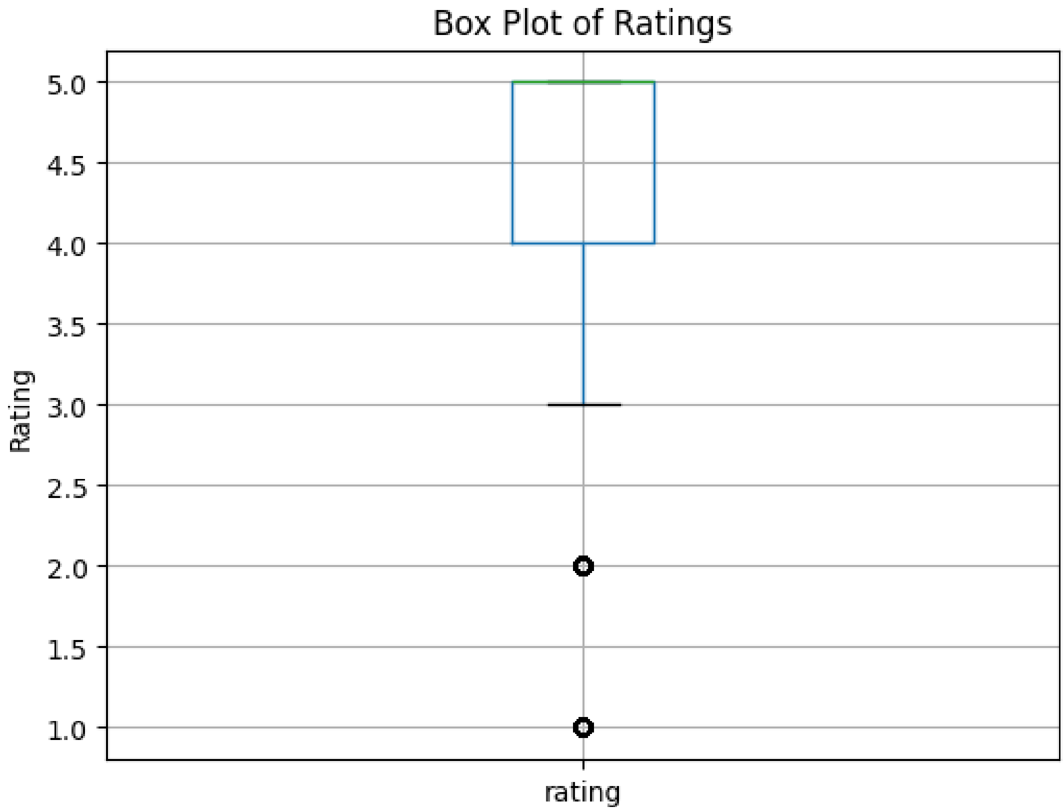
As expected, it seems that most reviews are positive, with a majority of the reviews having a rating of 4 or 5 stars. This can be explained by the fact that people are more likely to leave a review if they had a positive experience.

But also, we do see a significant number of 1-star reviews, with a very low number of 2-star reviews. This may indicate that people are more likely to leave a review if they had a very positive or very negative experience, rather than a neutral experience.

We can also draw a boxplot to visualize the distribution of ratings and identify any potential outliers.

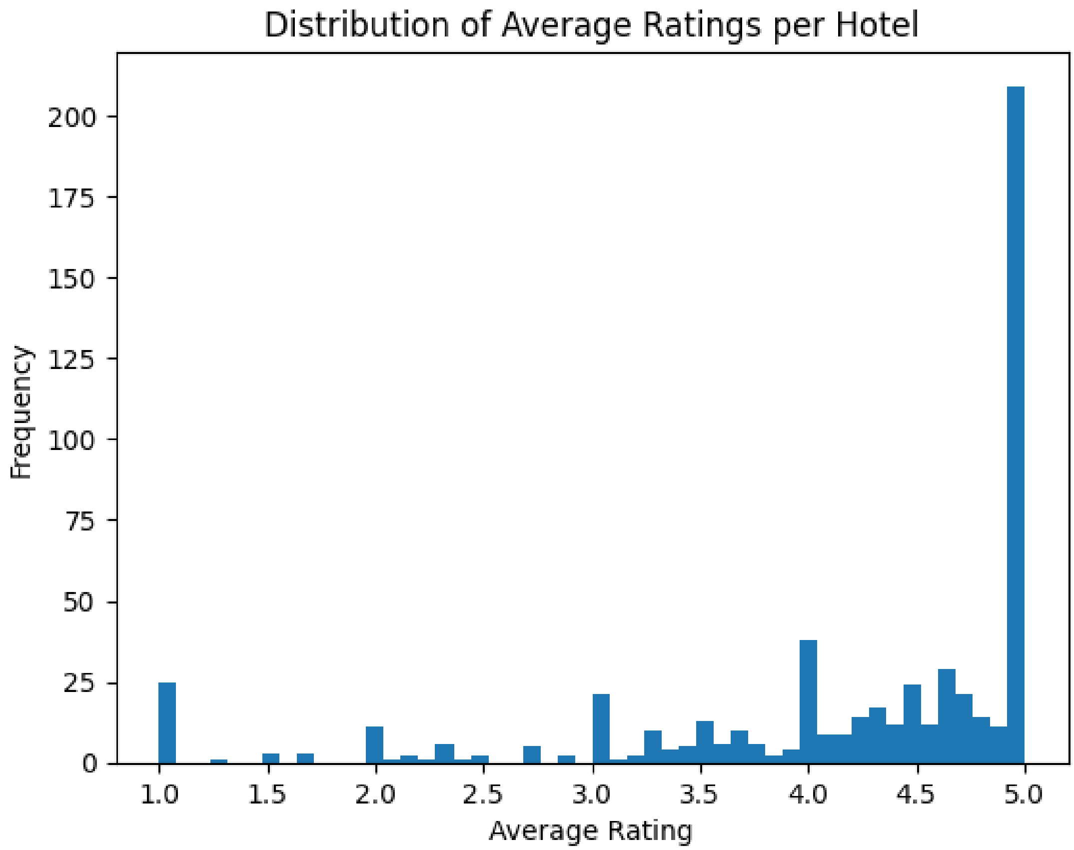
[13]:
```
reviews.boxplot(column="rating")
plt.ylabel("Rating")
plt.title("Box Plot of Ratings")
```

```
plt.show()
```



We can also visualize the distribution of the average rating by hotel. This will help us understand how the ratings are distributed across different hotels.

[14]:
```
avg_rating_per_hotel = reviews.groupby("location_id")["rating"].mean().
  ↪sort_values(ascending=False)

avg_rating_per_hotel.plot(kind='hist', bins=50)
plt.xlabel("Average Rating")
plt.ylabel("Frequency")
plt.title("Distribution of Average Ratings per Hotel")
plt.show()
```

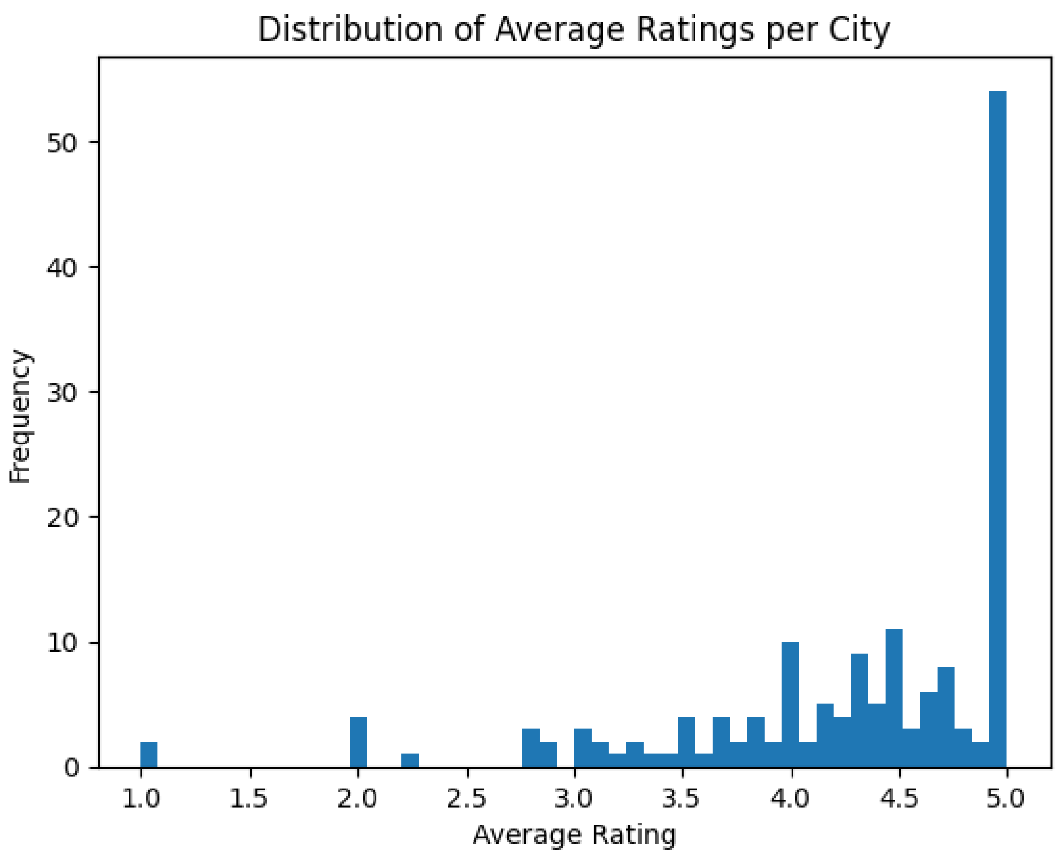


Here, we see that a majority of hotels have a 5-star rating. This is unexpected, as we would expect a more balanced distribution of ratings across hotels. This may be due to the fact that the data was collected using the API, which may have a bias towards reviews with higher ratings.

We also see peaks at whole numbers. This may be because most hotels have one, two, or a very small number of reviews, and thus the average rating is likely to be a whole number.

We can do the same to analyze the distribution of average ratings by city. This will help us understand how the ratings are distributed across different cities.

[15]:
```
# Plot average rating for city, plot histogram of the average ratings
avg_rating_per_city = reviews.groupby("city")["rating"].mean().
 ↪sort_values(ascending=False)
# histogram of average ratings
avg_rating_per_city.plot(kind='hist', bins=50)
plt.xlabel("Average Rating")
plt.ylabel("Frequency")
plt.title("Distribution of Average Ratings per City")
plt.show()
```



While here, we do see a similar pattern as the average ratings by hotel, we do see a more balanced distribution of ratings across cities. This may be due to the fact that there are more hotels in each city, and thus the average rating is less likely to be skewed by a small number of reviews.

1.5. Cleaning of text data

Now that we have described the dataset, we can move on to cleaning the text data. The text data consists of the `title` and `text` columns, which contain the title and full text of the review, respectively.

First, we can check the content of `title` column in the first few rows to understand its structure and contents.

[16]:
```
reviews["title"].head(10)
```

[16]:
```
0                          not a good stay
1                    Definitely recommend!
2                           Wonderful stay
3     My favorite 4+ star hotel in Colombo
4                  Excellent food and stay
5                              Outstanding
```

```
6                               I like it
7             Everything  is wonderful
8                         Great experience
9           Very good and highly recommend
Name: title, dtype: object
```

We notice some flaws in this data which will affect us later on when we try to analyze the text data - The titles are not in lowercase, which may affect our analysis later on. - The titles contain punctuation, special characters, and emoji, which would affect our analysis. - There is unnecessary whitespace in the titles, which may affect our analysis later on.

Our goal is to clean the text data so that it is in a consistent format, which will make it easier to analyze later on.

[17]:
```python
# convert title to lowercase
reviews["title"] = reviews["title"].str.lower()

# remove all non-alphanumeric
reviews["title"] = reviews["title"].str.replace(r'[^\w\s]', '', regex=True)

# remove unnecessary whitespace
reviews["title"] = reviews["title"].str.strip()
reviews['title'] = reviews['title'].str.replace(r'\s+', ' ', regex=True)

reviews["title"].head(10)
```

[17]:
```
0                          not a good stay
1                     definitely recommend
2                           wonderful stay
3     my favorite 4 star hotel in colombo
4                   excellent food and stay
5                              outstanding
6                                i like it
7                   everything is wonderful
8                         great experience
9          very good and highly recommend
Name: title, dtype: object
```

And we can do the same for the `text` columns

[18]:
```python
# convert text to lowercase
reviews["text"] = reviews["text"].str.lower()

# remove all non-alphanumeric
reviews["text"] = reviews["text"].str.replace(r'[^\w\s]', '', regex=True)

# remove unnecessary whitespace
reviews["text"] = reviews["text"].str.strip()
reviews['text'] = reviews['text'].str.replace(r'\s+', ' ', regex=True)
```

```
reviews["text"].head(10)
```

[18]:
```
0    found lighters in the toilet paper rolls in a …
1    the hotel is just excellent the food is so goo…
2    comfortable staycooperative stafffast service …
3    we live in new york area but my spouse is fami…
4    excellent food especially indian corner lot of…
5    spotless and immaculate premises the room is s…
6    house keeping and also respition i good and ni…
7    everything was amazing the breakfast and lunch…
8    we were there last week with our family and it…
9    very accommodating staff and lovely restaurant…
Name: text, dtype: object
```

Since the goal is to analyze the text data, we can combine the `title` and `text` columns into a single column called `review`. This will make it easier to analyze the text data later on.

[19]:
```
# combine title and text
reviews["review"] = reviews["title"] + " " + reviews["text"]
reviews["review"].head(10)
```

[19]:
```
0    not a good stay found lighters in the toilet p…
1    definitely recommend the hotel is just excelle…
2    wonderful stay comfortable staycooperative sta…
3    my favorite 4 star hotel in colombo we live in…
4    excellent food and stay excellent food especia…
5    outstanding spotless and immaculate premises t…
6    i like it house keeping and also respition i g…
7    everything is wonderful everything was amazing…
8    great experience we were there last week with …
9    very good and highly recommend very accommodat…
Name: review, dtype: object
```

1.6. Analyzing the corpus

[20]:
```
# total number of words
reviews["review"].str.split().str.len().sum()
```

[20]:
```
481190
```

We can see that there are 481,190 words in the dataset.

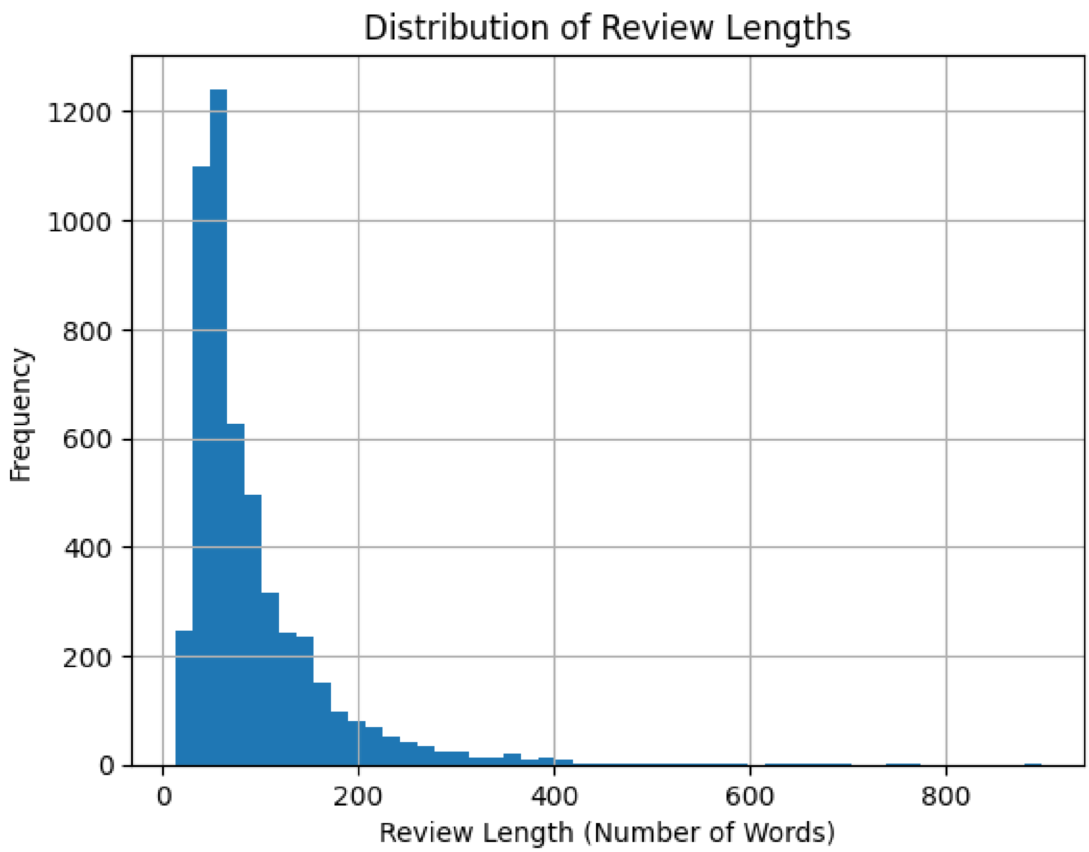
[21]:
```
# total number of unique words
reviews["review"].str.split().explode().nunique()
```

[21]:
```
18137
```

There are 18,134 unique words in the corpus

Next we can visualize the distribution of review lengths to understand how long the reviews are. This will help us understand the nature of the reviews and how they vary in length.

[22]:
```
# distribution of review lengths
reviews["review_length"] = reviews["review"].str.split().str.len()
reviews["review_length"].hist(bins=50)
plt.xlabel("Review Length (Number of Words)")
plt.ylabel("Frequency")
plt.title("Distribution of Review Lengths")
plt.show()
```

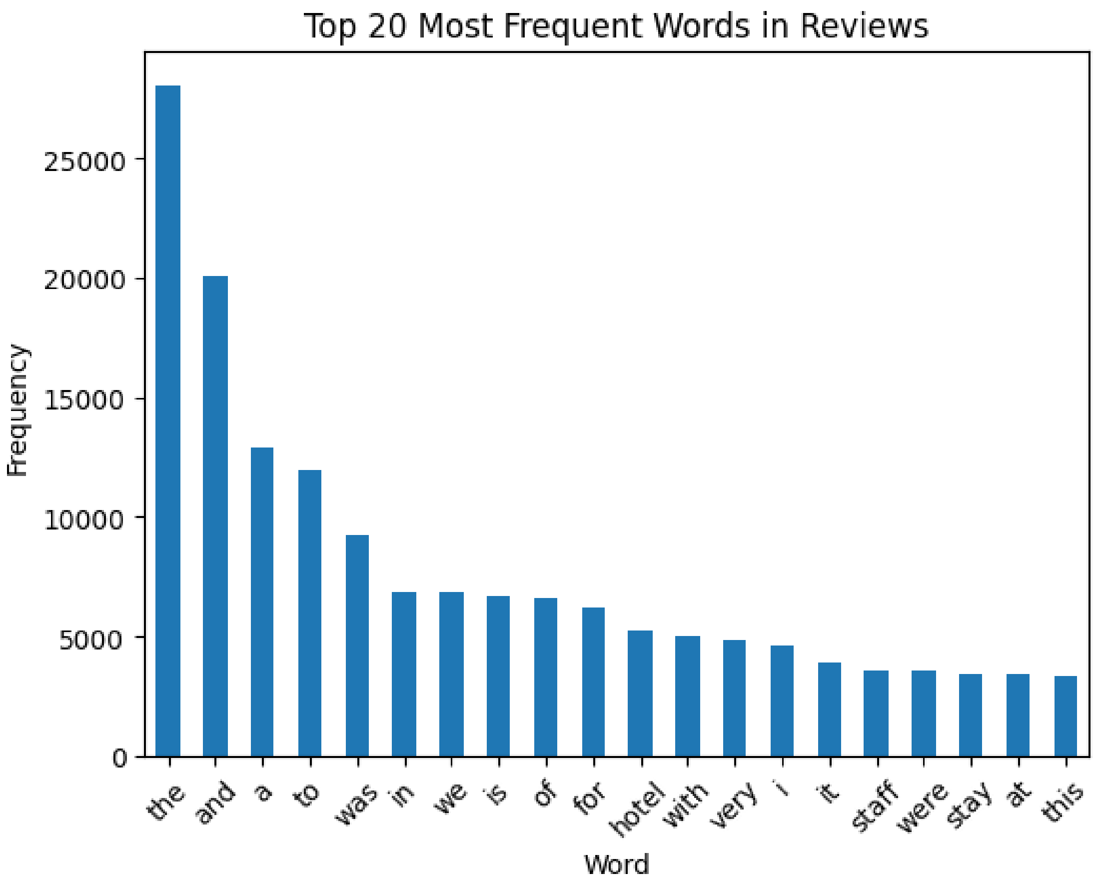


As expected. this distribution is highly skewed, with most reviews being relatively short. This is expected as most people would not take the time and make the effort to write long, detailed reviews.

Another interesting analysis would be to visualize the most frequent words in the reviews. This will help us understand the common themes and topics discussed in the reviews.

[23]:
```
word_counts = reviews["review"].str.split().explode().value_counts()
word_counts.head(20).plot(kind='bar')
plt.xlabel("Word")
```

```
plt.ylabel("Frequency")
plt.title("Top 20 Most Frequent Words in Reviews")
plt.xticks(rotation=45)
plt.show()
```



Here we run into a problem. The most frequent words in the reviews are common stopwords such as "the", "and", "to", etc. These words do not provide much information about the content of the reviews and can be considered noise in our analysis. To address this, we can remove stopwords from the reviews.

[24]:
```
stop_words = set(stopwords.words('english'))
def remove_stopwords(text):
    words = word_tokenize(text)
    return ' '.join([word for word in words if word not in stop_words])
reviews["review"] = reviews["review"].apply(remove_stopwords)
reviews["review"].head(10)
```

[24]:
```
0    good stay found lighters toilet paper rolls no…
1    definitely recommend hotel excellent food good…
```

```
2    wonderful stay comfortable staycooperative sta…
3    favorite 4 star hotel colombo live new york ar…
4    excellent food stay excellent food especially …
5    outstanding spotless immaculate premises room …
6    like house keeping also respition good nice li…
7    everything wonderful everything amazing breakf…
8    great experience last week family great experi…
9    good highly recommend accommodating staff love…
Name: review, dtype: object
```

[25]:
```python
reviews["review"].str.split().str.len().sum()
```

[25]:
```
263408
```

After removing stopwords, we can see that the total number of words in the dataset has decreased significantly.

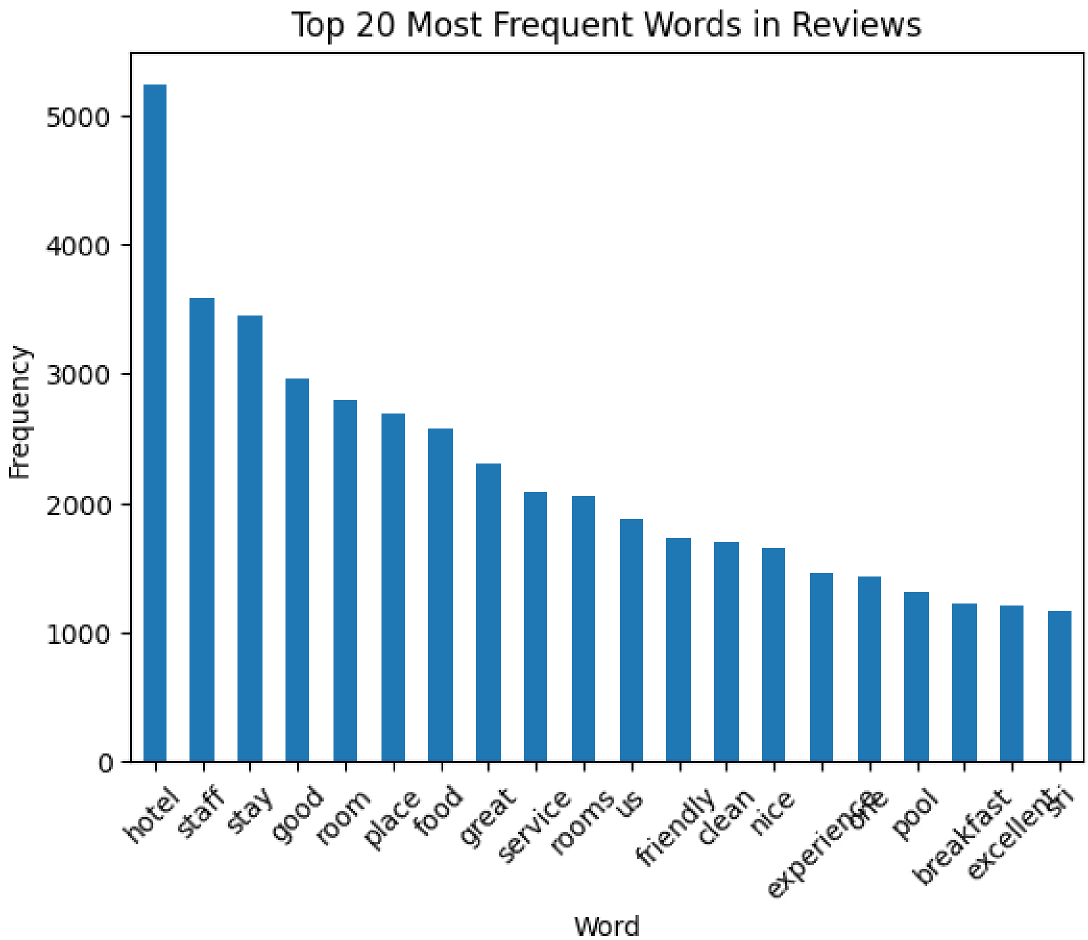
[26]:
```python
reviews["review"].str.split().explode().nunique()
```

[26]:
```
17998
```

However, the number of unique words in the corpus has not changed significantly.

We can now once again visualize the top 20 most frequent words in the reviews after removing stopwords.

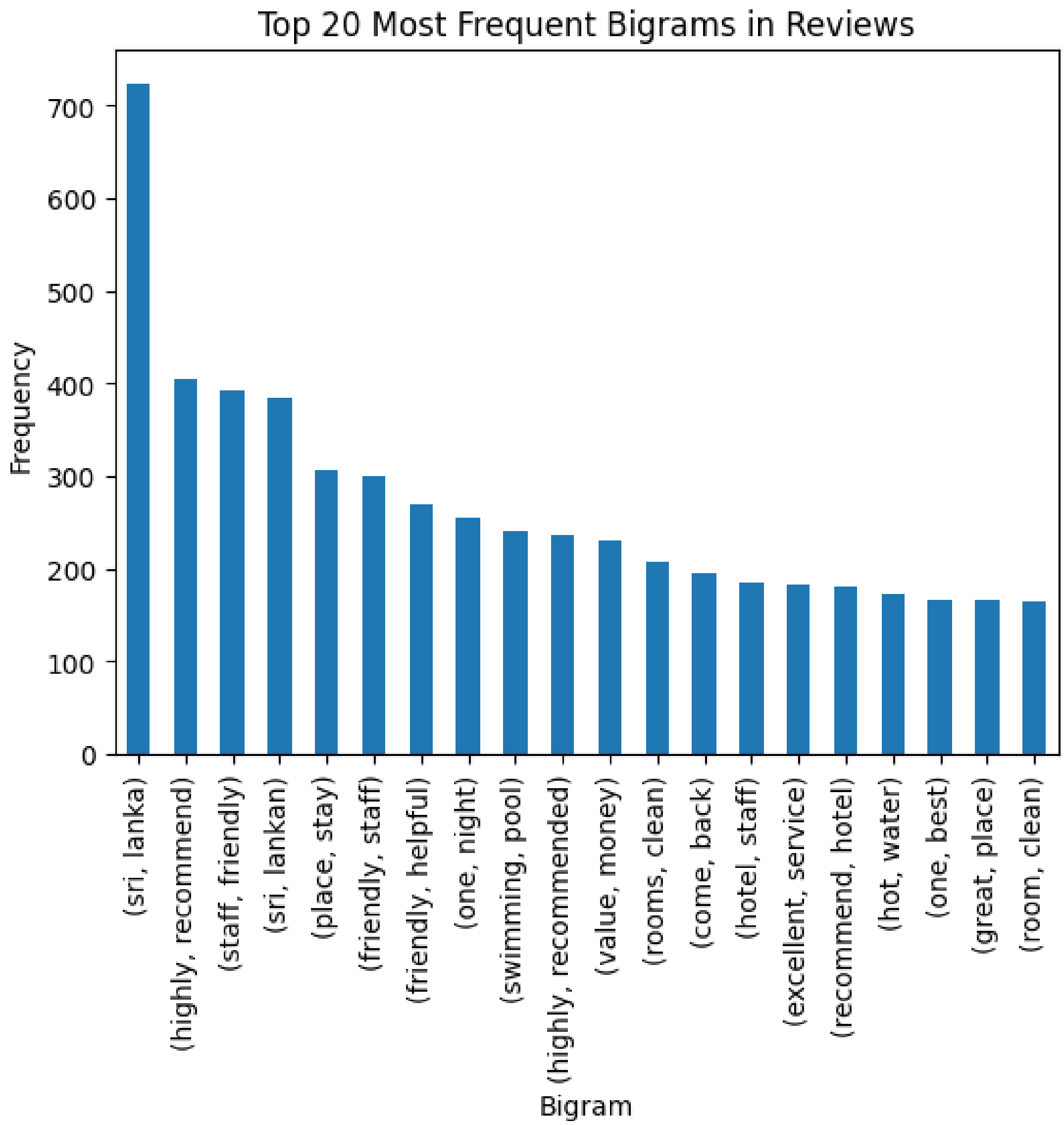
[27]:
```python
word_counts = reviews["review"].str.split().explode().value_counts()
word_counts.head(20).plot(kind='bar')
plt.xlabel("Word")
plt.ylabel("Frequency")
plt.title("Top 20 Most Frequent Words in Reviews")
plt.xticks(rotation=45)
plt.show()
```



Now, we can see that the most frequent words in the reviews are more meaningful and provide more information about the content of the reviews.

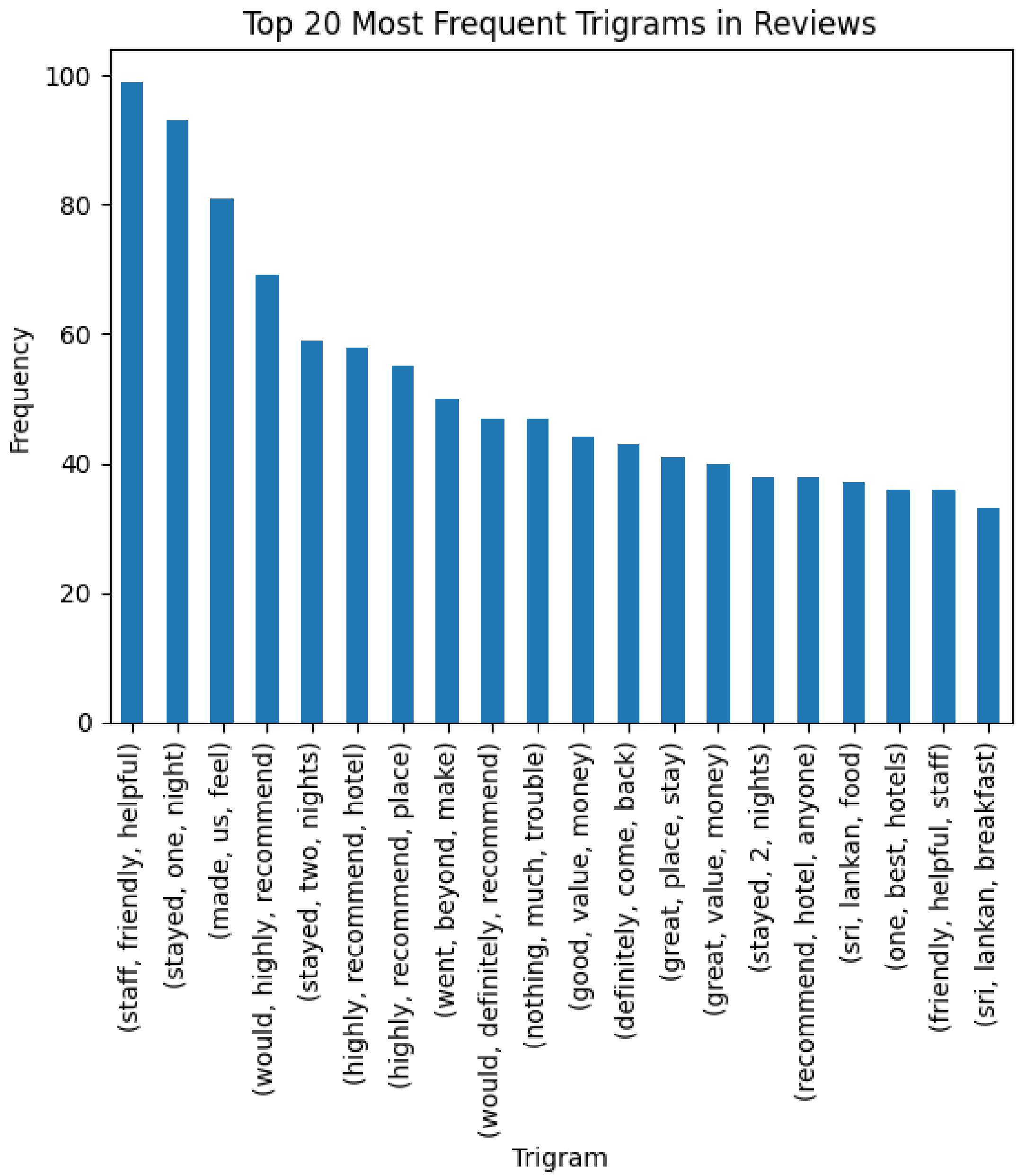
We can also check the most common bi-grams and tri-grams in the reviews to understand the common phrases used in the reviews.

[28]:
```
def get_ngrams(text, n):
    words = word_tokenize(text)
    return list(ngrams(words, n))

bigrams = reviews["review"].apply(lambda x: get_ngrams(x, 2)).explode().
  ↪value_counts()
bigrams.head(20).plot(kind='bar')
plt.xlabel("Bigram")
plt.ylabel("Frequency")
plt.title("Top 20 Most Frequent Bigrams in Reviews")
plt.xticks(rotation=90)
plt.show()
```



Here we can see some clear terms that is related to the reviews.

[29]:
```
trigrams = reviews["review"].apply(lambda x: get_ngrams(x, 3)).explode().
 ↪value_counts()
trigrams.head(20).plot(kind='bar')
plt.xlabel("Trigram")
plt.ylabel("Frequency")
plt.title("Top 20 Most Frequent Trigrams in Reviews")
plt.xticks(rotation=90)
plt.show()
```



Again we see some common terms we would expect to see in a review. These phrases can provide more context and meaning to the reviews, and can be useful for further analysis.

Now we can keep only the columns which are relevant for our analysis.

[30]:
```
reviews = reviews[["review_id", "location_id", "hotel_name", "city", "review",␣
  ↪"rating"]]
reviews.head()
```

```
[30]:     review_id  location_id              hotel_name     city  \
      0  1016464488     11953119  Nh Collection Colombo  Colombo
      1  1016435128     11953119  Nh Collection Colombo  Colombo
      2  1016307864     11953119  Nh Collection Colombo  Colombo
      3  1016165618     11953119  Nh Collection Colombo  Colombo
      4  1015472232     11953119  Nh Collection Colombo  Colombo

                                                    review  rating
      0  good stay found lighters toilet paper rolls no…       1
      1  definitely recommend hotel excellent food good…       5
      2  wonderful stay comfortable staycooperative sta…       5
      3  favorite 4 star hotel colombo live new york ar…       5
      4  excellent food stay excellent food especially …       5
```

Finally, we save this cleaned dataset to a new CSV file for further analysis.

```
[31]: reviews.to_csv("cleaned_reviews.csv", index=False)
```

Task 2 - Establishing Ground Truth

First we load the cleaned reviews dataset from task 1.

```
[32]: reviews = pd.read_csv("cleaned_reviews.csv")
      reviews.head()
```

```
[32]:     review_id  location_id              hotel_name     city  \
      0  1016464488     11953119  Nh Collection Colombo  Colombo
      1  1016435128     11953119  Nh Collection Colombo  Colombo
      2  1016307864     11953119  Nh Collection Colombo  Colombo
      3  1016165618     11953119  Nh Collection Colombo  Colombo
      4  1015472232     11953119  Nh Collection Colombo  Colombo

                                                    review  rating
      0  good stay found lighters toilet paper rolls no…       1
      1  definitely recommend hotel excellent food good…       5
      2  wonderful stay comfortable staycooperative sta…       5
      3  favorite 4 star hotel colombo live new york ar…       5
      4  excellent food stay excellent food especially …       5
```

Task 2.1. Using TextBlob

We can use TextBlob to analyze the sentiment of the reviews.

```
[33]: reviews['text_blob_sentiment'] = reviews['review'].apply(lambda text:␣
        ↪TextBlob(text).sentiment.polarity)
```

```
[34]: reviews['text_blob_sentiment'].describe()
```

[34]:
```
count    5186.000000
mean        0.367365
std         0.230712
min        -0.825000
25%         0.265239
50%         0.405107
75%         0.514089
max         1.000000
Name: text_blob_sentiment, dtype: float64
```

When we look at a summary of the sentiment scores, we can see that they range from -1 to 1, where -1 is very negative and 1 is very positive. The mean value is 0.367, indicating a generally positive sentiment across the reviews. The standard deviation is 0.231.

[35]:
```
reviews.head()
```

[35]:
```
    review_id  location_id             hotel_name     city  \
0  1016464488     11953119  Nh Collection Colombo  Colombo
1  1016435128     11953119  Nh Collection Colombo  Colombo
2  1016307864     11953119  Nh Collection Colombo  Colombo
3  1016165618     11953119  Nh Collection Colombo  Colombo
4  1015472232     11953119  Nh Collection Colombo  Colombo

                                              review  rating  \
0  good stay found lighters toilet paper rolls no…       1
1  definitely recommend hotel excellent food good…       5
2  wonderful stay comfortable staycooperative sta…       5
3  favorite 4 star hotel colombo live new york ar…       5
4  excellent food stay excellent food especially …       5

   text_blob_sentiment
0             0.333333
1             0.470238
2             0.386667
3             0.279339
4             0.519048
```

Here, we can see that the sentiment scores are continuous values. To convert these into binary sentiment labels, we can use a threshold of 0, where scores above 0 are considered positive and scores below or equal to 0 are considered negative.

[36]:
```
reviews['text_blob_sentiment'] = reviews['text_blob_sentiment'].apply(lambda x:␣
↪1 if x > 0 else -1)
reviews.head()
```

[36]:
```
    review_id  location_id             hotel_name     city  \
0  1016464488     11953119  Nh Collection Colombo  Colombo
1  1016435128     11953119  Nh Collection Colombo  Colombo
```

```
2  1016307864     11953119  Nh Collection Colombo  Colombo
3  1016165618     11953119  Nh Collection Colombo  Colombo
4  1015472232     11953119  Nh Collection Colombo  Colombo

                                              review  rating  \
0  good stay found lighters toilet paper rolls no…       1
1  definitely recommend hotel excellent food good…       5
2  wonderful stay comfortable staycooperative sta…       5
3  favorite 4 star hotel colombo live new york ar…       5
4  excellent food stay excellent food especially …       5

   text_blob_sentiment
0                    1
1                    1
2                    1
3                    1
4                    1
```

We can check whether the sentiment analysis is correct by looking at some 5-star and some 1-start reviews.

[37]:
```python
five_star_reviews = reviews[reviews['rating'] == 5]
five_star_reviews.head()
```

[37]:
```
    review_id  location_id             hotel_name     city  \
1  1016435128     11953119  Nh Collection Colombo  Colombo
2  1016307864     11953119  Nh Collection Colombo  Colombo
3  1016165618     11953119  Nh Collection Colombo  Colombo
4  1015472232     11953119  Nh Collection Colombo  Colombo
5  1015273964     11953119  Nh Collection Colombo  Colombo

                                              review  rating  \
1  definitely recommend hotel excellent food good…       5
2  wonderful stay comfortable staycooperative sta…       5
3  favorite 4 star hotel colombo live new york ar…       5
4  excellent food stay excellent food especially …       5
5  outstanding spotless immaculate premises room …       5

   text_blob_sentiment
1                    1
2                    1
3                    1
4                    1
5                    1
```

[38]:
```python
one_star_reviews = reviews[reviews['rating'] == 1]
one_star_reviews.head()
```

[38]:
```
     review_id  location_id                  hotel_name     city  \
0   1016464488     11953119      Nh Collection Colombo  Colombo
22  1013561310     11953119      Nh Collection Colombo  Colombo
76   568472844     11899031  De Colombo Boutique Hotel  Colombo
77   568472643     11899031  De Colombo Boutique Hotel  Colombo
80   565729487     11899031  De Colombo Boutique Hotel  Colombo

                                               review  rating  \
0   good stay found lighters toilet paper rolls no…       1
22  dont complaint restaurant food cold otherwise …       1
76  filthy towers probably worst hotel ive stayed …       1
77  stay peril total disappointing start holiday s…       1
80  absolutely disgusting stay bug infested overpr…       1

    text_blob_sentiment
0                     1
22                   -1
76                   -1
77                   -1
80                   -1
```

Here, we can see that the TextBlob sentiment analysis is generally correct, as the 5-star reviews have a positive sentiment score and the 1-star reviews have a negative sentiment score.

However, there is a single 1-star review that has a positive sentiment score.

Task 2.2. Using VADER

Now, we can use VADER (Valence Aware Dictionary and sEntiment Reasoner) to analyze the sentiment of the reviews. VADER is particularly effective for social media texts and short reviews.

[39]:
```python
sentiment = SentimentIntensityAnalyzer()
reviews['vader_sentiment'] = reviews['review'].apply(lambda text: sentiment.
 ↪polarity_scores(text))
```

[40]:
```python
reviews.head()
```

[40]:
```
    review_id  location_id             hotel_name     city  \
0  1016464488     11953119  Nh Collection Colombo  Colombo
1  1016435128     11953119  Nh Collection Colombo  Colombo
2  1016307864     11953119  Nh Collection Colombo  Colombo
3  1016165618     11953119  Nh Collection Colombo  Colombo
4  1015472232     11953119  Nh Collection Colombo  Colombo

                                              review  rating  \
0  good stay found lighters toilet paper rolls no…       1
1  definitely recommend hotel excellent food good…       5
2  wonderful stay comfortable staycooperative sta…       5
3  favorite 4 star hotel colombo live new york ar…       5
```

```
4  excellent food stay excellent food especially …       5

   text_blob_sentiment                                    vader_sentiment
0                    1  {'neg': 0.0, 'neu': 0.847, 'pos': 0.153, 'comp…
1                    1  {'neg': 0.0, 'neu': 0.316, 'pos': 0.684, 'comp…
2                    1  {'neg': 0.0, 'neu': 0.586, 'pos': 0.414, 'comp…
3                    1  {'neg': 0.0, 'neu': 0.716, 'pos': 0.284, 'comp…
4                    1  {'neg': 0.0, 'neu': 0.597, 'pos': 0.403, 'comp…
```

Here, we are given a dictionary with four keys: 'neg', 'neu', 'pos', and 'compound'. The 'compound' score is a normalized score that ranges from -1 (most negative) to +1 (most positive). We will use this score for our binary sentiment classification.

[41]:
```
reviews["vader_sentiment"] = reviews["vader_sentiment"].apply(lambda x:␣
 ↪x["compound"])
reviews.head()
```

[41]:
```
    review_id  location_id             hotel_name     city  \
0  1016464488     11953119  Nh Collection Colombo  Colombo
1  1016435128     11953119  Nh Collection Colombo  Colombo
2  1016307864     11953119  Nh Collection Colombo  Colombo
3  1016165618     11953119  Nh Collection Colombo  Colombo
4  1015472232     11953119  Nh Collection Colombo  Colombo

                                              review  rating  \
0  good stay found lighters toilet paper rolls no…       1
1  definitely recommend hotel excellent food good…       5
2  wonderful stay comfortable staycooperative sta…       5
3  favorite 4 star hotel colombo live new york ar…       5
4  excellent food stay excellent food especially …       5

   text_blob_sentiment  vader_sentiment
0                    1           0.4404
1                    1           0.9666
2                    1           0.8720
3                    1           0.9493
4                    1           0.9451
```

[42]:
```
reviews["vader_sentiment"].describe()
```

[42]:
```
count    5186.000000
mean        0.811598
std         0.450619
min        -0.994600
25%         0.925750
50%         0.969800
75%         0.985500
max         0.999300
```

```
Name: vader_sentiment, dtype: float64
```

Here, we can see that the average sentiment is 0.812, indicating a generally positive sentiment across the reviews. The standard deviation is 0.451, suggesting some variability in the sentiment scores.

We can once again convert these continuous sentiment scores into binary labels using a threshold of 0, where scores above 0 are considered positive and scores below or equal to 0 are considered negative.

[43]:
```
reviews['vader_sentiment'] = reviews['vader_sentiment'].apply(lambda x: 1 if x␣
↪> 0 else -1)
reviews.head()
```

[43]:
```
    review_id  location_id             hotel_name     city  \
0  1016464488     11953119  Nh Collection Colombo  Colombo
1  1016435128     11953119  Nh Collection Colombo  Colombo
2  1016307864     11953119  Nh Collection Colombo  Colombo
3  1016165618     11953119  Nh Collection Colombo  Colombo
4  1015472232     11953119  Nh Collection Colombo  Colombo

                                              review  rating  \
0  good stay found lighters toilet paper rolls no…       1
1  definitely recommend hotel excellent food good…       5
2  wonderful stay comfortable staycooperative sta…       5
3  favorite 4 star hotel colombo live new york ar…       5
4  excellent food stay excellent food especially …       5

   text_blob_sentiment  vader_sentiment
0                    1                1
1                    1                1
2                    1                1
3                    1                1
4                    1                1
```

And then check some 5-star and 1-star reviews to see if the sentiment analysis is correct.

[44]:
```
five_star_reviews = reviews[reviews['rating'] == 5]
five_star_reviews.head()
```

[44]:
```
    review_id  location_id             hotel_name     city  \
1  1016435128     11953119  Nh Collection Colombo  Colombo
2  1016307864     11953119  Nh Collection Colombo  Colombo
3  1016165618     11953119  Nh Collection Colombo  Colombo
4  1015472232     11953119  Nh Collection Colombo  Colombo
5  1015273964     11953119  Nh Collection Colombo  Colombo

                                              review  rating  \
1  definitely recommend hotel excellent food good…       5
```

```
2  wonderful stay comfortable staycooperative sta…       5
3  favorite 4 star hotel colombo live new york ar…       5
4  excellent food stay excellent food especially …       5
5  outstanding spotless immaculate premises room …       5

   text_blob_sentiment  vader_sentiment
1                    1                1
2                    1                1
3                    1                1
4                    1                1
5                    1                1
```

[45]:
```
one_star_reviews = reviews[reviews['rating'] == 1]
one_star_reviews.head()
```

[45]:
```
     review_id  location_id                 hotel_name     city  \
0   1016464488     11953119      Nh Collection Colombo  Colombo
22  1013561310     11953119      Nh Collection Colombo  Colombo
76   568472844     11899031  De Colombo Boutique Hotel  Colombo
77   568472643     11899031  De Colombo Boutique Hotel  Colombo
80   565729487     11899031  De Colombo Boutique Hotel  Colombo

                                               review  rating  \
0   good stay found lighters toilet paper rolls no…       1
22  dont complaint restaurant food cold otherwise …       1
76  filthy towers probably worst hotel ive stayed …       1
77  stay peril total disappointing start holiday s…       1
80  absolutely disgusting stay bug infested overpr…       1

    text_blob_sentiment  vader_sentiment
0                     1                1
22                   -1               -1
76                   -1               -1
77                   -1               -1
80                   -1               -1
```

Similar to TextBlob, the VADER sentiment analysis is generally correct, as the 5-star reviews have a positive sentiment score and the 1-star reviews have a negative sentiment score.

But once again, there is a single 1-star review that has a positive sentiment score.

Task 2.3. Using Transformers

Since our transformer model has a maximum input length of 512 tokens, we need to limit the length of the reviews to this maximum length.

[46]:
```
def limit_tokens(text, max_length=512):
    return text[:max_length]
```

```
reviews['transformer_review'] = reviews['review'].apply(limit_tokens)
```

Now, we can use a pre-trained transformer model for sentiment analysis.

This provides two outputs, `score` and `label`, where `score` is the confidence score of the sentiment label and `label` is either 'POSITIVE' or 'NEGATIVE'.

[48]:
```
sentiment_classifier = pipeline("sentiment-analysis",␣
↪model="distilbert-base-uncased-finetuned-sst-2-english")
reviews['transformer_sentiment'] = reviews['transformer_review'].apply(lambda␣
↪text: sentiment_classifier(text)[0])

reviews['transformer_sentiment_score'] = reviews['transformer_sentiment'].
↪apply(lambda x: x['score'])
reviews['transformer_sentiment_label'] = reviews['transformer_sentiment'].
↪apply(lambda x: x['label'])
reviews.head()
```

```
WARNING:tensorflow:From C:\Users\kanee\.virtualenvs\sri-lanka-hotel-
reviews\Lib\site-packages\tf_keras\src\losses.py:2976: The name
tf.losses.sparse_softmax_cross_entropy is deprecated. Please use
tf.compat.v1.losses.sparse_softmax_cross_entropy instead.


Device set to use cpu
```

[48]:
```
    review_id  location_id             hotel_name     city  \
0  1016464488     11953119  Nh Collection Colombo  Colombo
1  1016435128     11953119  Nh Collection Colombo  Colombo
2  1016307864     11953119  Nh Collection Colombo  Colombo
3  1016165618     11953119  Nh Collection Colombo  Colombo
4  1015472232     11953119  Nh Collection Colombo  Colombo

                                              review  rating  \
0  good stay found lighters toilet paper rolls no…       1
1  definitely recommend hotel excellent food good…       5
2  wonderful stay comfortable staycooperative sta…       5
3  favorite 4 star hotel colombo live new york ar…       5
4  excellent food stay excellent food especially …       5

   text_blob_sentiment  vader_sentiment  \
0                    1                1
1                    1                1
2                    1                1
3                    1                1
4                    1                1

                                  transformer_review  \
0  good stay found lighters toilet paper rolls no…
```

```
1  definitely recommend hotel excellent food good…
2  wonderful stay comfortable staycooperative sta…
3  favorite 4 star hotel colombo live new york ar…
4  excellent food stay excellent food especially …

                             transformer_sentiment  \
0  {'label': 'POSITIVE', 'score': 0.9806791543960…
1  {'label': 'POSITIVE', 'score': 0.999824583530426}
2  {'label': 'POSITIVE', 'score': 0.9993433356285…
3  {'label': 'POSITIVE', 'score': 0.9975016713142…
4  {'label': 'POSITIVE', 'score': 0.9997791647911…

   transformer_sentiment_score transformer_sentiment_label
0                     0.980679                    POSITIVE
1                     0.999825                    POSITIVE
2                     0.999343                    POSITIVE
3                     0.997502                    POSITIVE
4                     0.999779                    POSITIVE
```

We can calculate an overall sentiment score by multiplying the score by -1 if the label is 'NEGATIVE', and leaving it as is if the label is 'POSITIVE'. This way, we can convert the sentiment scores into a binary format where positive sentiment is represented by 1 and negative sentiment by -1.

[49]:
```
reviews['transformer_sentiment'] = reviews.apply(
    lambda row: row['transformer_sentiment_score'] * -1 if␣
 ↪row['transformer_sentiment_label'] == 'NEGATIVE' else␣
 ↪row['transformer_sentiment_score'],
    axis=1
)
reviews = reviews.drop(columns=["transformer_review",␣
 ↪"transformer_sentiment_score", "transformer_sentiment_label"])
reviews.head()
```

[49]:
```
    review_id  location_id             hotel_name     city  \
0  1016464488     11953119  Nh Collection Colombo  Colombo
1  1016435128     11953119  Nh Collection Colombo  Colombo
2  1016307864     11953119  Nh Collection Colombo  Colombo
3  1016165618     11953119  Nh Collection Colombo  Colombo
4  1015472232     11953119  Nh Collection Colombo  Colombo

                                              review  rating  \
0  good stay found lighters toilet paper rolls no…       1
1  definitely recommend hotel excellent food good…       5
2  wonderful stay comfortable staycooperative sta…       5
3  favorite 4 star hotel colombo live new york ar…       5
4  excellent food stay excellent food especially …       5
```

```
   text_blob_sentiment  vader_sentiment  transformer_sentiment
0                    1                1               0.980679
1                    1                1               0.999825
2                    1                1               0.999343
3                    1                1               0.997502
4                    1                1               0.999779
```

[50]:
```
reviews['transformer_sentiment'].describe()
```

[50]:
```
count    5186.000000
mean        0.623263
std         0.763787
min        -0.999799
25%         0.982515
50%         0.999153
75%         0.999708
max         0.999882
Name: transformer_sentiment, dtype: float64
```

Here, we can see that the average sentiment is 0.623, indicating a generally positive sentiment across the reviews. The standard deviation is 0.764.

Once again, we can convert these continuous sentiment scores into binary labels using a threshold of 0, where scores above 0 are considered positive and scores below or equal to 0 are considered negative.

[51]:
```
reviews['transformer_sentiment'] = reviews['transformer_sentiment'].
  ↪apply(lambda x: 1 if x > 0 else -1)
reviews.head()
```

[51]:
```
    review_id  location_id              hotel_name     city  \
0  1016464488     11953119  Nh Collection Colombo  Colombo
1  1016435128     11953119  Nh Collection Colombo  Colombo
2  1016307864     11953119  Nh Collection Colombo  Colombo
3  1016165618     11953119  Nh Collection Colombo  Colombo
4  1015472232     11953119  Nh Collection Colombo  Colombo

                                              review  rating  \
0  good stay found lighters toilet paper rolls no…       1
1  definitely recommend hotel excellent food good…       5
2  wonderful stay comfortable staycooperative sta…       5
3  favorite 4 star hotel colombo live new york ar…       5
4  excellent food stay excellent food especially …       5

   text_blob_sentiment  vader_sentiment  transformer_sentiment
0                    1                1                      1
1                    1                1                      1
```

```
2                      1                1                        1
3                      1                1                        1
4                      1                1                        1
```

And then check some 5-star and 1-star reviews to see if the sentiment analysis is correct.

[52]:
```python
five_star_reviews = reviews[reviews['rating'] == 5]
five_star_reviews.head()
```

[52]:
```
    review_id  location_id             hotel_name     city  \
1  1016435128     11953119  Nh Collection Colombo  Colombo
2  1016307864     11953119  Nh Collection Colombo  Colombo
3  1016165618     11953119  Nh Collection Colombo  Colombo
4  1015472232     11953119  Nh Collection Colombo  Colombo
5  1015273964     11953119  Nh Collection Colombo  Colombo

                                              review  rating  \
1  definitely recommend hotel excellent food good…       5
2  wonderful stay comfortable staycooperative sta…       5
3  favorite 4 star hotel colombo live new york ar…       5
4  excellent food stay excellent food especially …       5
5  outstanding spotless immaculate premises room …       5

   text_blob_sentiment  vader_sentiment  transformer_sentiment
1                    1                1                      1
2                    1                1                      1
3                    1                1                      1
4                    1                1                      1
5                    1                1                      1
```

[53]:
```python
one_star_reviews = reviews[reviews['rating'] == 1]
one_star_reviews.head()
```

[53]:
```
     review_id  location_id                  hotel_name     city  \
0   1016464488     11953119      Nh Collection Colombo  Colombo
22  1013561310     11953119      Nh Collection Colombo  Colombo
76   568472844     11899031  De Colombo Boutique Hotel  Colombo
77   568472643     11899031  De Colombo Boutique Hotel  Colombo
80   565729487     11899031  De Colombo Boutique Hotel  Colombo

                                               review  rating  \
0   good stay found lighters toilet paper rolls no…       1
22  dont complaint restaurant food cold otherwise …       1
76  filthy towers probably worst hotel ive stayed …       1
77  stay peril total disappointing start holiday s…       1
80  absolutely disgusting stay bug infested overpr…       1

    text_blob_sentiment  vader_sentiment  transformer_sentiment
```

```
0                      1                 1                       1
22                    -1                -1                      -1
76                    -1                -1                      -1
77                    -1                -1                      -1
80                    -1                -1                      -1
```

Similar to TextBlob and VADER, the transformer sentiment analysis is generally correct, as the 5-star reviews have a positive sentiment score and the 1-star reviews have a negative sentiment score.

Once again, there is a single 1-star review that has a positive sentiment score.

Task 2.4. Establishing Ground Truth with Majority Voting

Here, we simply use majority voting to establish the ground truth sentiment for each review. We will consider the sentiment labels from TextBlob, VADER, and the transformer model.

[54]:
```
# majority voting to establish ground truth
def majority_vote(row):
    votes = [row['text_blob_sentiment'], row['vader_sentiment'],␣
 ↪row['transformer_sentiment']]
    return max(set(votes), key=votes.count)

reviews['ground_truth_sentiment'] = reviews.apply(majority_vote, axis=1)
reviews.head()
```

[54]:
```
    review_id  location_id             hotel_name     city  \
0  1016464488     11953119  Nh Collection Colombo  Colombo
1  1016435128     11953119  Nh Collection Colombo  Colombo
2  1016307864     11953119  Nh Collection Colombo  Colombo
3  1016165618     11953119  Nh Collection Colombo  Colombo
4  1015472232     11953119  Nh Collection Colombo  Colombo

                                              review  rating  \
0  good stay found lighters toilet paper rolls no…       1
1  definitely recommend hotel excellent food good…       5
2  wonderful stay comfortable staycooperative sta…       5
3  favorite 4 star hotel colombo live new york ar…       5
4  excellent food stay excellent food especially …       5

   text_blob_sentiment  vader_sentiment  transformer_sentiment  \
0                    1                1                      1
1                    1                1                      1
2                    1                1                      1
3                    1                1                      1
4                    1                1                      1

   ground_truth_sentiment
```

```
0                       1
1                       1
2                       1
3                       1
4                       1
```

Finally, we remove the unnecessary columns and save the reviews with the ground truth sentiment to a new CSV file.

[55]:
```
reviews = reviews[["review_id", "location_id", "hotel_name", "city", "review",␣
↪"rating", "ground_truth_sentiment"]]
reviews.to_csv("ground_truth_reviews.csv", index=False)
```

Task 3 - Feature Extraction

First we will load the ground truth reviews dataset.

[56]:
```
reviews = pd.read_csv("ground_truth_reviews.csv")
reviews.head()
```

[56]:
```
    review_id  location_id             hotel_name     city  \
0  1016464488     11953119  Nh Collection Colombo  Colombo
1  1016435128     11953119  Nh Collection Colombo  Colombo
2  1016307864     11953119  Nh Collection Colombo  Colombo
3  1016165618     11953119  Nh Collection Colombo  Colombo
4  1015472232     11953119  Nh Collection Colombo  Colombo

                                              review  rating  \
0  good stay found lighters toilet paper rolls no…       1
1  definitely recommend hotel excellent food good…       5
2  wonderful stay comfortable staycooperative sta…       5
3  favorite 4 star hotel colombo live new york ar…       5
4  excellent food stay excellent food especially …       5

   ground_truth_sentiment
0                       1
1                       1
2                       1
3                       1
4                       1
```

3.1. Bag of Words (BoW)

Here, we will create a Bag of Words (BoW) representation of the reviews. This involves tokenizing the text and creating a matrix where each row corresponds to a review and each column corresponds to a word in the vocabulary.

```
[57]: vectorizer = CountVectorizer()
X = vectorizer.fit_transform(reviews["review"])
```

```
[58]: vocabulary = vectorizer.get_feature_names_out()
print(f"Size of BoW vocabulary: {len(vocabulary)}")
```

```
Size of BoW vocabulary: 17968
```

We can see here that there are 17,968 unique words in the vocabulary extracted from the reviews.

```
[59]: bow_matrix = pd.DataFrame(X.toarray(), columns=vocabulary)
print(f"Shape of the BoW matrix: {bow_matrix.shape}")
```

```
Shape of the BoW matrix: (5186, 17968)
```

The shape of the BoW matrix is (5186, 17968), meaning there are 5186 reviews, and each vector has 17968 features corresponding to the unique words in the vocabulary.

We can even print the first row of the BoW matrix to see how it looks.

```
[60]: print(bow_matrix.iloc[0])
```

```
000                      0
01                       0
0111and                  0
0120                     0
0130                     0
                        ..
              0
                   0
                    0
                  0
                     0
Name: 0, Length: 17968, dtype: int64
```

Let's check for some words from the first review that are present.

```
[61]: print(bow_matrix.iloc[0][bow_matrix.iloc[0] > 0])
```

```
beds        1
booked      1
even        1
found       1
give        1
good        1
lighters    1
non         1
paper       1
rolls       1
room        1
smoking     1
```

```
stay        1
though      1
toilet      1
twin        1
us          1
Name: 0, dtype: int64
```

3.2. Term Frequency-Inverse Document Frequency (TF-IDF)

Here, we will create a Term Frequency-Inverse Document Frequency (TF-IDF) representation of the reviews.

[62]:
```
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(reviews['review'])
```

[63]:
```
feature_names = tfidf_vectorizer.get_feature_names_out()
print(f"Size of TF-IDF vocabulary: {len(feature_names)}")
```

```
Size of TF-IDF vocabulary: 17968
```

Once again, we can see that there are 17,968 unique words in the vocabulary extracted from the reviews.

[64]:
```
tfidf_matrix_df = pd.DataFrame(tfidf_matrix.toarray(), columns=feature_names)
print(f"Shape of the TF-IDF matrix: {tfidf_matrix_df.shape}")
```

```
Shape of the TF-IDF matrix: (5186, 17968)
```

The shape of the TF-IDF matrix is also (5186, 17968), meaning there are 5186 reviews, and each vector has 17968 features corresponding to the unique words in the vocabulary.

Once again, we can print the first row of the TF-IDF matrix to see how it looks.

[65]:
```
print(tfidf_matrix_df.iloc[0])
```

```
000                     0.0
01                      0.0
0111and                 0.0
0120                    0.0
0130                    0.0
                      …
             0.0
                  0.0
                   0.0
                 0.0
                    0.0
Name: 0, Length: 17968, dtype: float64
```

Let's check for some words from the first review that are present in the TF-IDF matrix.

[66]:
```
print(tfidf_matrix_df.iloc[0][tfidf_matrix_df.iloc[0] > 0])
```

```
beds        0.203157
booked      0.181399
even        0.144703
found       0.203431
give        0.205972
good        0.092402
lighters    0.406354
non         0.280659
paper       0.296387
rolls       0.342779
room        0.095741
smoking     0.374567
stay        0.084388
though      0.192924
toilet      0.232811
twin        0.320513
us          0.112083
Name: 0, dtype: float64
```

We can also check for the top 10 words with the highest TF-IDF scores in the first review.

[67]:
```python
top_tfidf_words = tfidf_matrix_df.iloc[0].nlargest(10)
print("Top 10 words with highest TF-IDF scores in the first review:")
print(top_tfidf_words)
```

```
Top 10 words with highest TF-IDF scores in the first review:
lighters    0.406354
smoking     0.374567
rolls       0.342779
twin        0.320513
paper       0.296387
non         0.280659
toilet      0.232811
give        0.205972
found       0.203431
beds        0.203157
Name: 0, dtype: float64
```

3.3. Word2Vec

Here, we will create a Word2Vec model using the reviews.

First, we need to tokenize the reviews into words.

[68]:
```python
tokenized_reviews = [word_tokenize(review.lower()) for review in␣
    ↪reviews['review']]
```

Now we can train a Word2Vec model on the tokenized reviews. We will use a vector size of 500, a window size of 100, and set the minimum count to 0 to include all words.

[69]:
```
w2v_model = Word2Vec(sentences=tokenized_reviews, vector_size=500, window=100,␣
↪min_count=0, workers=8, sg=1)
w2v_model.save("word2vec.model")
```

[70]:
```
print(f"Shape of the Word2Vec matrix: {w2v_model.wv.vectors.shape}")
```

```
Shape of the Word2Vec matrix: (17998, 500)
```

We can see that the Word2Vec model has a shape of (17998, 500), meaning there are 17,998 unique words in the vocabulary, and each word is represented by a 500-dimensional vector.

We can also check the vector value for a specific word, such as "bed".

[71]:
```
bed_vector = w2v_model.wv['bed']
print(f"Vector for 'bed': {bed_vector}")
```

```
Vector for 'bed': [ 0.1844372  -0.10257292  0.17788999  0.19952342  0.04292466
-0.23675644
  0.07076817  0.03539717  0.08850911 -0.02314061 -0.05752808  0.03549338
  0.00104585  0.01297401 -0.10800468 -0.25414315  0.05870118 -0.07489647
 -0.05281977  0.22001526 -0.02574226 -0.04577768  0.17006585 -0.21410057
  0.10951288 -0.00938682 -0.00614369  0.04501679 -0.34815875  0.03551801
  0.20308386 -0.08911312 -0.00210324  0.03012024  0.02330348  0.06697601
  0.02439669 -0.0392347  -0.05458113  0.05747749  0.10461799 -0.10335528
 -0.06747732  0.23971984 -0.11673975 -0.1081759  -0.05466682  0.13536918
 -0.0706562  -0.03587937 -0.0635895  -0.10628379 -0.05294625 -0.21889162
  0.07317803 -0.04108338  0.08783511  0.03113046 -0.077149   -0.18124133
  0.22803335  0.03314252  0.03984741 -0.11962193 -0.01882334 -0.17456776
  0.00993659  0.0074919  -0.00315017  0.05624627 -0.06785657 -0.2537139
 -0.04047243  0.19330123 -0.09077385 -0.00248228  0.08260996 -0.05269075
  0.21162105  0.12569617 -0.07086682  0.09819639 -0.18350449 -0.01780526
  0.08135381  0.0509728  -0.05724541 -0.12373818  0.03000416 -0.1400245
 -0.12389022 -0.00928606 -0.00850368  0.10941684  0.02235218  0.0925041
  0.18771885 -0.06542277  0.04460561 -0.11018941  0.15884621  0.10101486
 -0.19079687  0.13145168  0.04006653  0.09559311 -0.23538916  0.13027026
 -0.0608473  -0.02373417  0.02534174  0.09073629  0.01537811 -0.04433034
 -0.07183558 -0.02136111  0.10014264 -0.05549701  0.04337163 -0.00269196
  0.07182343 -0.03978065  0.11838983  0.03614376  0.07210702 -0.08669329
  0.05842778 -0.19676714  0.09631534 -0.06437867  0.12647738 -0.12307542
 -0.11893753  0.09435421  0.02830199  0.01360792  0.02965821  0.06090072
  0.20286013 -0.09405614  0.0672066  -0.08115898 -0.1463719  -0.02037554
  0.12356167  0.07674226  0.1242403  -0.13465942  0.249863    0.01174712
  0.15742685  0.26871774  0.08630387  0.01488803 -0.13662189 -0.02946596
 -0.02057092  0.05359181 -0.1739224   0.07280898  0.01284409 -0.1048597
 -0.11698365  0.09645552 -0.05968834 -0.18271609  0.03553054  0.2858857
  0.03477913  0.1284718   0.04283534  0.04078577 -0.1148545  -0.1492925
 -0.00948055  0.10382608 -0.0536807   0.02989614 -0.1355156   0.00936795
 -0.23955984  0.0109306  -0.04211406  0.1202722   0.12914526  0.00764124
  0.06758852 -0.13247806  0.01794667 -0.01466513  0.01749976  0.0813588
```

0.13025267	-0.1139527	-0.1304318	0.01662214	0.04170086	0.06869387
0.21191332	-0.00291214	0.06528	0.10417991	0.04089952	-0.06065143
0.09657624	0.15240748	-0.03604958	-0.08699867	-0.09102692	-0.01020489
0.18935949	-0.0288148	-0.04831329	0.05867115	-0.110004	-0.12592196
-0.02835615	0.04446037	-0.04769205	0.3301477	0.10166533	-0.03467563
-0.04535127	-0.1879205	0.19813636	0.00468122	0.37384614	-0.121879
0.13759273	0.05992829	0.00868221	-0.13821822	-0.0227396	0.20636643
-0.08954371	0.05370031	-0.17238416	-0.13562934	0.36359206	0.02252229
-0.02948765	0.04623323	0.10440234	-0.09579884	-0.03734337	0.03039018
0.00370041	-0.12875576	-0.08935651	-0.06142213	0.05394038	-0.11041756
0.02735144	0.07022378	0.08170774	0.11622757	0.07580069	0.11852384
-0.08064437	0.20107414	0.0698503	-0.21630147	0.05455851	-0.15781043
-0.01130259	0.00177089	-0.08615237	0.13433433	0.00674329	0.00978961
-0.02347769	0.09800322	-0.05924483	0.0336457	-0.20902789	-0.21730685
0.18360788	-0.1231401	0.06249287	-0.00997195	0.07403973	0.3866196
0.00303953	-0.09894336	0.14151378	-0.19258685	0.11768466	-0.07250569
0.1906392	0.09587739	-0.00519396	0.15513924	0.00587651	0.19310343
-0.06110229	-0.00657703	0.16287848	-0.2533163	-0.10225767	-0.02848588
0.15117213	0.04414863	0.02823743	0.08061945	-0.02965169	-0.09444863
-0.03985021	0.03990638	0.08846393	-0.00830092	-0.02906027	-0.25177756
-0.04820682	0.21360989	-0.02228772	-0.03160191	-0.29868057	-0.08829036
-0.0055447	-0.07729341	0.12049271	0.03320733	0.08600439	0.04823029
-0.13137276	-0.12478767	-0.0583314	0.14139807	-0.11057524	0.02426602
-0.07097711	0.11269736	-0.08817058	-0.01939551	-0.01506976	-0.11306027
-0.11382524	-0.08168425	-0.10848306	-0.08359256	0.10838749	0.00587904
0.19163083	-0.05902981	-0.03796811	-0.03585653	0.02226594	0.13302755
-0.0717176	-0.01678049	0.07352668	-0.07032949	-0.07993732	-0.00239308
-0.27789432	-0.02713069	-0.00803474	0.02933297	-0.0956839	-0.13428153
0.23717013	0.03932515	0.12096186	0.05711147	-0.30678374	0.15655318
0.10951999	-0.01547423	-0.08053595	0.11253118	0.1019598	-0.06835664
-0.01003085	-0.08676268	0.03677994	0.09001233	-0.13048448	-0.1342329
0.10663333	-0.08072104	0.2103484	0.00936687	0.111664	-0.10017502
-0.09030594	-0.24114743	-0.10134518	0.13722616	-0.09039776	0.11768287
0.03593387	0.00550915	-0.0457013	-0.11314183	0.20402893	0.11998287
0.07357303	0.0522606	-0.04026178	0.00476254	0.05223504	0.15283154
0.01087755	-0.05453857	-0.05031351	-0.05882723	0.13010505	0.26927716
-0.03781779	0.15343364	-0.08478757	-0.18718164	-0.01588611	-0.03523844
0.15920942	-0.13679066	-0.11451174	0.07795897	0.18215215	-0.09209527
-0.16078253	0.01273965	-0.08074788	-0.12065414	-0.26705375	-0.05478381
0.15320173	0.03840606	0.04441883	0.04861359	-0.01462221	-0.1391946
0.10580122	0.02720133	-0.06785638	-0.20764206	0.05509708	-0.07706539
-0.2980109	0.2059985	0.0686525	-0.14547126	-0.00953484	0.03094622
0.16438697	-0.03447533	-0.03830413	0.05705978	0.08336231	0.0582453
-0.09951843	-0.03183495	0.04855189	0.06038515	-0.02482061	-0.0957569
0.00988365	0.01556278	0.30419785	0.04328809	0.05516292	0.01867631
0.01384997	-0.02892135	-0.01058154	-0.16792232	-0.08213946	-0.09214213
0.02592845	0.1697343	0.20541114	-0.16216546	-0.05227989	-0.08842223
0.07058875	0.05275696	0.10998929	-0.04809195	-0.25033048	0.19683686

```
 0.21345729 -0.30450276 -0.09525409  0.06865762  0.05323928  0.02467292
-0.2656329   0.22957374  0.20621075  0.01433345  0.12088017  0.14621434
 0.25872752  0.17769612  0.12526731  0.3354954  -0.16168287  0.07708745
-0.3161554   0.21210252]
```

We can also find the most similar words to “bed” using the Word2Vec model.

[72]:
```
similar_words = w2v_model.wv.most_similar('bed')
print(f"Most similar words to 'bed': {similar_words}")
```

```
Most similar words to 'bed': [('squeezed', 0.6514195799827576), ('partially',
0.6497069597244263), ('duvets', 0.6381158828735352), ('airconditioner',
0.6350743770599365), ('dressing', 0.6348100304603577), ('ragged',
0.6344547867774963), ('doubles', 0.6331870555877686), ('bathroom',
0.6308902502059937), ('deet', 0.6283610463142395), ('drenching',
0.6272135376930237)]
```

We can also perform analogy tasks using the Word2Vec model. For example, we can find a word that is to “colombo” as “galle” is to “city”.

[73]:
```
analogy_result = w2v_model.wv.most_similar(positive=['colombo', 'galle'],␣
  ↪negative=['city'], topn=1)
print(f"Analogy result for 'colombo' - 'city' + 'galle': {analogy_result}")
```

```
Analogy result for 'colombo' - 'city' + 'galle': [('fort', 0.4800596237182617)]
```

Here, we can see that the model determines that Colombo - City + Galle = Fort. Which makes intuitive sense.

We can also perform other analogy tasks, such as finding a word that is to “bed” as “internet” is to “sleep”.

[74]:
```
analogy_result = w2v_model.wv.most_similar(positive=['bed', 'internet'],␣
  ↪negative=['sleep'], topn=1)
print(f"Analogy result for 'bed' - 'sleep' + 'internet': {analogy_result}")
```

```
Analogy result for 'bed' - 'sleep' + 'internet': [('cons', 0.5540193319320679)]
```

Here, it determined that Bed - Sleep + Internet = Connection. Which also makes sense.

Let’s check another analogy task, such as finding a word that is to “bed” as “water” is to “pillow”.

[75]:
```
analogy_result = w2v_model.wv.most_similar(positive=['bed', 'water'],␣
  ↪negative=['pillow'], topn=1)
print(f"Analogy result for 'bed' - 'pillow' + 'water': {analogy_result}")
```

```
Analogy result for 'bed' - 'pillow' + 'water': [('hot', 0.46690458059310913)]
```

Here, it determined that Bed - Pillow + Water = Hot. This is unexpected, and highlights the limitations of the model in understanding certain relationships.

We can also check for some common relationships, but those which might not be present in the dataset.

```
[76]: result = w2v_model.wv.most_similar(positive=['king', 'woman'],␣
       ↪negative=['man'], topn=1)
      print(f"Analogy result for 'king' - 'man' + 'woman': {result}")
```

```
Analogy result for 'king' - 'man' + 'woman': [('anjalee', 0.6017975211143494)]
```

Here, we see that the model struggles to come up with a meaningful analogy for this relationship, which highlights the limitations of the dataset to generalize.

Finally, we can vectorize the reviews using the Word2Vec model by averaging the word vectors for each review.

```
[77]: def get_review_vector(review, model):
          tokens = word_tokenize(review.lower())
          vector = sum(model.wv[token] for token in tokens if token in model.wv) /␣
       ↪len(tokens)
          return vector

      w2v_review_vectors = reviews['review'].apply(lambda x: get_review_vector(x,␣
       ↪w2v_model))
      w2v_review_vectors = np.vstack(w2v_review_vectors.values)
      w2v_review_vectors = pd.DataFrame(w2v_review_vectors)
      print(f"Shape of the Word2Vec review vectors: {w2v_review_vectors.shape}")
```

```
Shape of the Word2Vec review vectors: (5186, 500)
```

We can see that the dataset now consists of 5186 reviews, and each review is represented by a 500-dimensional vector. We can even print the first review vector to see how it looks.

```
[78]: print(w2v_review_vectors[0])
```

```
0       0.027821
1      -0.012155
2       0.006719
3       0.050971
4       0.013252
          …
5181    0.037094
5182    0.023887
5183    0.002189
5184    0.010953
5185   -0.018435
Name: 0, Length: 5186, dtype: float32
```

3.4. Doc2Vec

```
[79]: tokenized_reviews = [word_tokenize(review.lower()) for review in␣
       ↪reviews['review']]
      tagged_data = [TaggedDocument(words=words, tags=[str(i)]) for i, words in␣
       ↪enumerate(tokenized_reviews)]
```

```
doc2vec_model = Doc2Vec(vector_size=500, window=50, min_count=1, workers=8, epochs=20)
doc2vec_model.build_vocab(tagged_data)
doc2vec_model.train(tagged_data, total_examples=doc2vec_model.corpus_count, epochs=doc2vec_model.epochs)

doc2vec_review_vectors = np.vstack([doc2vec_model.infer_vector(words) for words in tokenized_reviews])
doc2vec_review_vectors = pd.DataFrame(doc2vec_review_vectors)
```

[80]:
```
print(f"Shape of the Doc2Vec review vectors: {doc2vec_review_vectors.shape}")
```

```
Shape of the Doc2Vec review vectors: (5186, 500)
```

[81]:
```
inferred_vector = doc2vec_model.infer_vector("the bed was uncomfortable".lower().split())
print(inferred_vector.shape)
```

```
(500,)
```

[82]:
```
print(inferred_vector)
```

```
[ 8.14211834e-03  1.31451143e-02  3.59016210e-02  2.35563572e-02
 -1.72582902e-02 -1.78530458e-02  1.12911575e-02  2.36368217e-02
  4.73257998e-04 -3.00145242e-03 -4.35618870e-03 -7.57505465e-03
  1.90240648e-02 -8.55872594e-03  1.38425007e-02 -6.38915822e-02
 -1.95247047e-02 -3.06914952e-02 -1.01258615e-02 -6.48445077e-03
  7.89745897e-03 -2.13737171e-02  2.19342373e-02  2.16143578e-03
  9.87065677e-03  1.86061487e-02  4.81344759e-03 -2.83058663e-03
 -4.70013842e-02 -3.01680826e-02  1.72006823e-02 -7.06574321e-03
  1.75223816e-02 -1.38048949e-02  2.22008731e-02  4.99573629e-03
  1.47084659e-02 -4.80641946e-02 -1.85609162e-02 -3.38403843e-02
 -2.26015914e-02 -9.47795343e-03 -1.34116355e-02  8.63720663e-03
 -6.85438141e-03 -2.45347377e-02  5.83647052e-03  1.72961671e-02
 -1.59516546e-03  2.50510871e-03 -2.46076882e-02  3.00840964e-03
  5.04804915e-03 -1.28899524e-02 -6.08506566e-03 -2.20296113e-03
  1.70543627e-03 -1.04843322e-02  2.08804794e-02  4.35620593e-03
 -1.08019740e-03 -1.19080418e-03 -3.08983377e-03  1.11899478e-02
  1.03067448e-02  2.38062046e-03  1.24980407e-02 -2.90035969e-04
 -7.30294921e-03 -3.19750723e-03 -1.34192291e-03 -1.08558480e-02
 -8.83528497e-03 -2.41176551e-03 -3.43314302e-03  1.63053330e-02
  1.51641585e-03  1.00254882e-02  1.36248963e-02 -3.06389388e-03
 -9.73680057e-03 -1.43420221e-02 -2.82455292e-02  2.25210991e-02
 -4.20174897e-02  2.39656549e-02 -3.88939679e-03  1.05792494e-03
  8.54237098e-03  2.93832570e-02 -1.93916950e-02 -6.28843298e-03
 -1.40150869e-02  1.10858921e-02  7.02906982e-04  3.35212960e-03
  1.53463027e-02  3.17692943e-03  3.33373062e-02  2.26130746e-02
```

-3.95886526e-02	2.15662178e-02	6.46969420e-04	3.09840273e-02
-8.82148370e-03	-3.75598529e-03	4.40398511e-03	2.12907282e-04
-1.50321070e-02	1.33399656e-02	3.02734599e-03	-2.95790769e-02
-1.33207003e-02	4.55105193e-02	1.74903730e-03	-6.92192744e-03
5.08153765e-03	-3.97513760e-03	-1.26758656e-02	-7.47352326e-03
-2.13850569e-02	-2.15617218e-03	4.17007357e-02	-1.38851441e-02
3.11486423e-02	1.52447494e-02	-2.62220930e-02	1.08288871e-02
8.94723553e-03	3.83546366e-03	2.06282381e-02	2.24679764e-02
-9.24212579e-03	-6.83576753e-03	1.01697464e-02	2.90319249e-02
7.04600709e-03	-5.48070995e-04	-1.55855017e-02	-7.28860795e-02
1.62423011e-02	-4.26918194e-02	1.58082675e-02	-2.28615496e-02
-6.75709627e-04	1.07614202e-02	9.31901112e-03	-2.53673606e-02
3.44024226e-03	4.28603822e-03	2.18082331e-02	-1.98675389e-03
7.56186061e-03	2.28212234e-02	-3.60808484e-02	1.96443107e-02
4.15601116e-03	-2.15595923e-02	-1.68759786e-02	1.75523981e-02
7.23599503e-03	1.77823827e-02	-2.88605504e-02	-8.39517824e-03
-4.49439790e-03	1.41611481e-02	3.11868507e-02	2.83237547e-02
1.25568099e-02	2.93745138e-02	1.69718470e-02	2.11197026e-02
-2.90731993e-03	3.33325490e-02	1.25741342e-03	-2.10899618e-02
2.06158962e-03	-2.13190280e-02	-8.19643121e-03	1.22465165e-02
-2.89915968e-02	3.85076832e-03	5.69374813e-03	1.88882947e-02
-3.72204557e-02	-1.87863335e-02	-3.85234281e-02	2.47649848e-02
-8.70522205e-03	1.80523861e-02	4.66399826e-02	-3.05057433e-03
1.94879193e-02	-1.02060372e-02	-3.68806795e-04	1.35512026e-02
-1.81755852e-02	-1.38428053e-02	-7.48117315e-03	2.33398844e-02
-9.78255365e-03	1.96507620e-03	2.74449699e-02	-2.42640008e-03
3.26544940e-02	9.47623572e-04	-1.15988578e-03	2.20016651e-02
-2.72523873e-02	-3.41567653e-03	-1.08785657e-02	-1.89356471e-03
-4.75032767e-03	-1.10408599e-02	1.22793410e-02	-1.21903662e-02
-1.33229038e-02	-2.69959564e-03	-1.13362232e-02	-6.20455900e-03
1.48149822e-02	-4.50535957e-03	1.04721738e-02	1.39651401e-02
-7.20698340e-03	9.11838142e-05	1.79773811e-02	-3.06551550e-02
1.84923857e-02	-9.27864574e-03	1.22854952e-02	7.39600742e-03
-1.30939735e-02	9.02333204e-03	3.16240243e-04	-3.51376226e-03
-1.22426711e-02	-9.02819075e-03	1.10045681e-02	1.37810772e-02
4.81623312e-04	-1.17232567e-02	-2.00321618e-02	-8.52048956e-03
-8.33276473e-03	-1.20838440e-03	1.09436759e-03	1.78085137e-02
2.11842116e-02	1.84948940e-03	2.16454286e-02	-3.78238857e-02
1.16898706e-02	-1.24419881e-02	-3.10899247e-03	1.27625782e-02
1.53046881e-03	1.30382515e-02	6.38863246e-04	6.19271584e-03
-1.46255894e-02	1.34909118e-03	-3.45696881e-02	4.98443935e-03
7.51739647e-03	-9.00719955e-04	3.83511360e-04	1.18069621e-02
7.07295304e-03	-5.54007769e-04	-7.22196326e-03	2.14823876e-02
1.04927495e-02	1.46514727e-02	-7.63035798e-03	5.52075775e-03
-1.23977847e-03	-2.63745673e-02	2.30468158e-02	3.28347534e-02
1.57490354e-02	1.10391397e-02	-1.52164642e-02	-2.02449448e-02
3.74686681e-02	-3.10126878e-02	1.40468804e-02	-9.50395688e-03
1.38137517e-02	-1.10185174e-02	-1.84038922e-03	2.02081725e-03

-2.02731858e-03	1.46455010e-02	1.36682820e-02	1.17059667e-02
-7.88831431e-03	-9.65412334e-03	-1.97602995e-02	9.52102616e-03
-7.38785835e-04	-1.50904600e-02	9.20843612e-03	-9.69444122e-03
1.08534619e-02	-1.48039740e-02	-7.00354017e-03	1.75997559e-02
7.30644865e-03	8.52386351e-04	-2.23306157e-02	-2.16682293e-02
-3.71274189e-03	6.93621207e-03	1.06294791e-03	-1.41253732e-02
1.67157575e-02	-1.66540267e-03	8.61939602e-03	1.07654827e-02
-3.40699428e-03	-9.46407206e-03	-1.84087772e-02	-1.11179063e-02
-8.83353688e-03	6.67387061e-03	-1.71599705e-02	-1.17527479e-02
9.12631489e-03	-8.07846617e-03	3.54071055e-03	8.63390416e-03
-6.07155217e-03	6.35386538e-03	1.74715195e-03	-3.62051986e-02
-7.35901436e-03	9.62435175e-03	-1.21763991e-02	-8.29651835e-04
-6.46600965e-04	-1.03210937e-02	2.60339063e-02	-2.72295196e-02
2.22854502e-02	-1.97200198e-02	-4.28833477e-02	1.80722643e-02
1.30424174e-02	1.82691421e-02	-8.19358882e-03	-2.64582224e-03
-4.31418717e-02	4.02320623e-02	-3.33374143e-02	5.02535366e-02
1.61983464e-02	-3.59373190e-03	1.20675396e-02	-1.70755461e-02
6.50275080e-03	1.97942778e-02	-2.40895734e-03	5.02277445e-03
-6.24647411e-03	-1.94343850e-02	3.80854402e-03	-9.30823945e-03
-5.34892492e-02	5.66614419e-03	6.54547068e-04	3.10542214e-06
-1.31421015e-02	-2.48511098e-02	4.62271785e-03	-9.12742969e-03
1.08840764e-02	-1.73527226e-02	-1.53933768e-03	6.63914892e-04
1.53386025e-02	7.72957271e-03	1.79114863e-02	1.89251080e-02
-8.36788490e-03	-1.58678684e-02	-2.34990520e-03	2.13480573e-02
1.00940391e-02	9.10778530e-03	-9.34944116e-03	-3.88462981e-03
-4.79784422e-03	7.32472818e-03	2.07071025e-02	2.52604615e-02
1.29564842e-02	1.45341677e-03	4.71587572e-03	1.29662938e-02
-8.96378886e-03	1.54698957e-02	6.97157858e-03	-7.80395558e-03
-9.99747775e-03	-7.26961705e-04	-4.16230457e-03	1.87336598e-02
6.01681322e-03	-7.09126610e-03	-1.09332539e-02	-1.75784566e-02
-9.17331968e-03	-2.40913313e-02	3.80891259e-03	2.62956414e-03
-8.47814896e-04	9.60234739e-03	2.03599278e-02	-2.25568824e-02
-7.82922935e-03	1.98404975e-02	-9.67325456e-03	1.99839696e-02
-1.39071122e-02	9.56366304e-03	-1.20488890e-02	5.48257306e-03
-9.16128606e-03	-3.04137036e-04	-2.04953440e-02	2.82452609e-02
2.01897398e-02	1.96121633e-02	-2.56375261e-02	-2.75002723e-03
1.28089339e-02	1.78786851e-02	9.99556109e-03	-7.05302041e-03
1.69155039e-02	-2.19507851e-02	-1.66611280e-03	-1.24825276e-02
3.23854126e-02	1.37595925e-02	2.94441711e-02	1.77154243e-02
-2.41319020e-03	7.46816024e-03	-3.67770088e-03	-1.81598449e-03
1.22830819e-03	-4.59236652e-02	8.93895607e-03	1.02031687e-02
-1.26962019e-02	-2.40468234e-02	-8.44850391e-03	-1.56275667e-02
-3.67027777e-03	2.12494638e-02	-2.73944121e-02	-1.20731359e-02
-7.10142870e-03	-1.63374841e-02	2.64681969e-03	-4.20766398e-02
-1.33319767e-02	8.16050917e-03	-1.45454789e-02	-1.65110156e-02
9.43245646e-03	-2.77263112e-02	7.46253971e-03	-3.82646482e-04
2.99044978e-02	-1.27645005e-02	-3.56607959e-02	-7.34105194e-03
-5.61096333e-03	-3.43938954e-02	2.50285811e-04	1.28215011e-02

```
 8.45091138e-03  2.97411531e-03 -2.01925132e-02  4.87901038e-03
 1.33877806e-02  8.26594245e-04  4.51056566e-03  6.33422658e-03
 1.67629737e-02  1.49249388e-02  9.77325533e-03  4.79354663e-03
-3.04058511e-02 -1.24189463e-02 -9.08910763e-03  1.32789779e-02]
```

[83]:

```
similar_docs = doc2vec_model.dv.most_similar([inferred_vector], topn=5)

for doc_id, similarity in similar_docs:
    print(f"Document {doc_id}: Similarity={similarity:.4f}")
    print("Review:", reviews['review'].iloc[int(doc_id)])
    print("---")
```

```
Document 1494: Similarity=0.8206
Review: interesting hotel staff made feel incredibly welcome room spotless
wellappointed highlight stay food every meal culinary delight dishes beautifully
presented also bursting flavor breakfast buffet particularly impressive offering
wide variety options freshly prepared dining onsite restaurant true pleasure
menu catered tastes whether breakfast lunch dinner every meal exceeded
expectations hotel mustvisit food lover looking luxurious stay
---
Document 2614: Similarity=0.8160
Review: beautiful hotel really enyoyed staying beautifull hotel room veary clean
air con host frendly arranged us tour really nice tuctuc driver tharindu took us
see waterfalls came back got us fresh juice dinner breakfast delicius
---
Document 947: Similarity=0.8104
Review: perfect place stay pleasant stay senani hotel clean rooms good food
helpful staff meals amantha made sure everything thank much definitely come back
hotel future
---
Document 2784: Similarity=0.8082
Review: cheap price stayed hotel two nights budget room okay price nice clean
big basic room pressure shower breakfast basic owners sweet helpful safari hotel
great
---
Document 177: Similarity=0.8058
Review: avoid staying 8 days havent cleaned room changed bedsheets called owner
even apologized told want cleaned soon possible asked checking please avoid
---
```

Finally, we can save all the feature matrices to CSV files for further use.

[84]:

```
feature_matrices = {
    'bow': bow_matrix,
    'tfidf': tfidf_matrix_df,
    'word2vec': w2v_review_vectors,
    'doc2vec': doc2vec_review_vectors
}
```

[85]:
```python
for name, matrix in feature_matrices.items():
    print(f"Saving {name} feature matrix...")
    matrix.to_csv(f"feature_matrix_{name}.csv", index=False)
```

```
Saving bow feature matrix…
Saving tfidf feature matrix…
Saving word2vec feature matrix…
Saving doc2vec feature matrix…
```

Task 4 - Text Classification

First we load the feature matrices and the ground truth sentiment labels that we compiled in task 3.

[86]:
```python
bow_data = pd.read_csv('feature_matrix_bow.csv')
tfidf_data = pd.read_csv('feature_matrix_tfidf.csv')
word2vec_data = pd.read_csv('feature_matrix_word2vec.csv')
doc2vec_data = pd.read_csv('feature_matrix_doc2vec.csv')
```

[87]:
```python
reviews = pd.read_csv('ground_truth_reviews.csv')
ground_truth = reviews['ground_truth_sentiment'].values
```

Now we define a function, that can apply a classifier to the feature matrix and the ground truth labels, and print out the accuracy, precision, recall, F1 score and confusion matrix.

[88]:
```python
def classifier(features, ground_truth, name, clf):
    y_pred = cross_val_predict(clf, features, ground_truth, cv=5)
    print(f'Accuracy for {name}: {accuracy_score(ground_truth, y_pred):.4f}')
    print(f"Precision for {name}: {precision_score(ground_truth, y_pred,␣
↪average='weighted'):.4f}")
    print(f"Recall for {name}: {recall_score(ground_truth, y_pred,␣
↪average='weighted'):.4f}")
    print(f"F1 Score for {name}: {f1_score(ground_truth, y_pred,␣
↪average='weighted'):.4f}")

    cm = confusion_matrix(ground_truth, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm)
    disp.plot(text_kw={'color': 'black'})
    plt.title(f'Confusion Matrix on {name} Features')
    plt.show()

    return f1_score(ground_truth, y_pred, average='weighted')
```

We also create a dictionary to store the F1 scores for each classifier and feature matrix combination for later comparison.

[89]:
```python
f1_scores = {
    "naive_bayes": {
        "bow": 0.0,
```

```
        "tfidf": 0.0,
        "word2vec": 0.0,
        "doc2vec": 0.0
    },
    "svm": {
        "bow": 0.0,
        "tfidf": 0.0,
        "word2vec": 0.0,
        "doc2vec": 0.0
    },
    "random_forest": {
        "bow": 0.0,
        "tfidf": 0.0,
        "word2vec": 0.0,
        "doc2vec": 0.0
    }
}
```

4.1. Naive Bayes Classifier

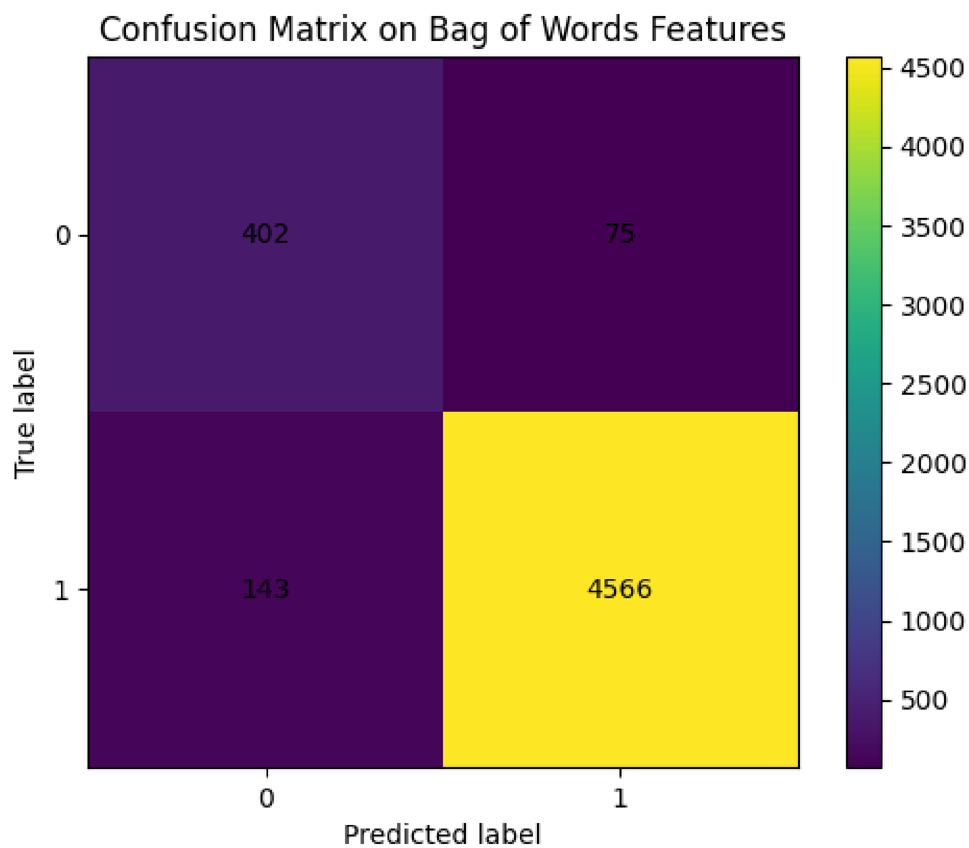
We first try using the Naive Bayes classifier on the different feature matrices.

We selected the Naive Bayes classifier because it is simple, fast, and often effective for text classification tasks.

We use `MultinomialNB` for Bag of Words, TF-IDF, and Word2Vec. And we use `GaussianNB` for GloVe.

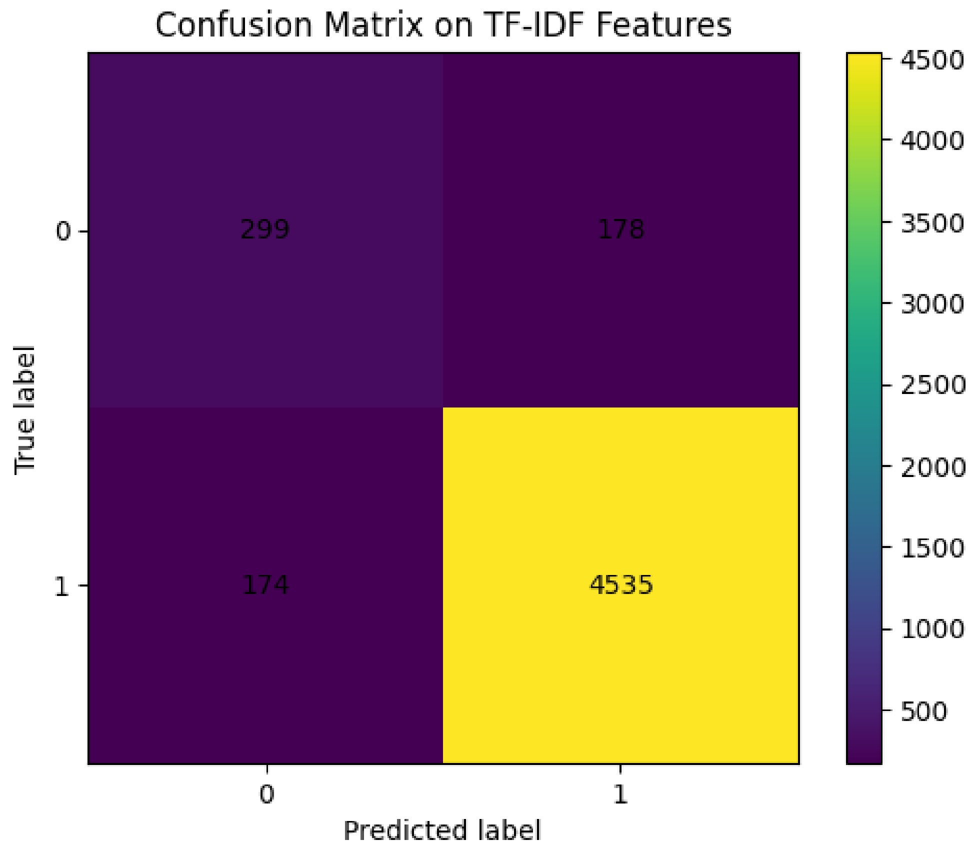
[90]:
```
f1_scores["naive_bayes"]["bow"] = classifier(bow_data, ground_truth, "Bag of␣
↪Words", MultinomialNB())
```

```
Accuracy for Bag of Words: 0.9580
Precision for Bag of Words: 0.9612
Recall for Bag of Words: 0.9580
F1 Score for Bag of Words: 0.9592
```



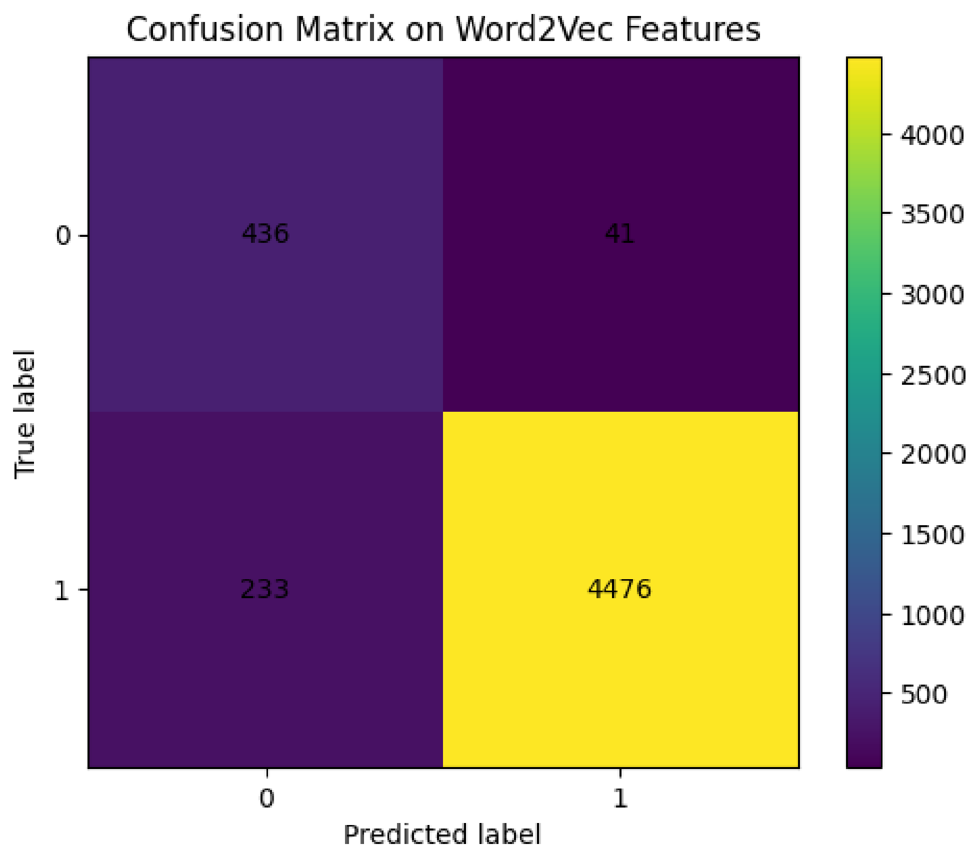
[91]:
```
f1_scores["naive_bayes"]["tfidf"] = classifier(MinMaxScaler().
 ↪fit_transform(tfidf_data), ground_truth, "TF-IDF", MultinomialNB())
```

```
Accuracy for TF-IDF: 0.9321
Precision for TF-IDF: 0.9319
Recall for TF-IDF: 0.9321
F1 Score for TF-IDF: 0.9320
```



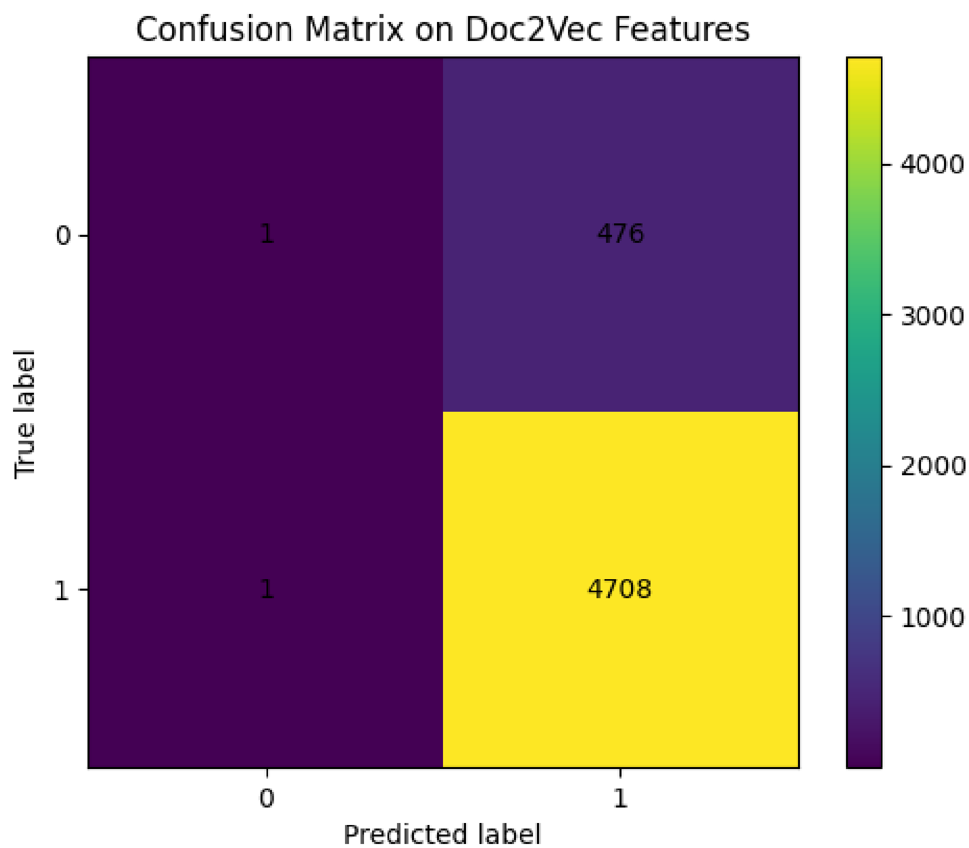
[92]:
```
f1_scores["naive_bayes"]["word2vec"] = classifier(MinMaxScaler().
    ↪fit_transform(word2vec_data), ground_truth, "Word2Vec", MultinomialNB())
```

```
Accuracy for Word2Vec: 0.9472
Precision for Word2Vec: 0.9597
Recall for Word2Vec: 0.9472
F1 Score for Word2Vec: 0.9510
```




```
[93]: f1_scores["naive_bayes"]["doc2vec"] = classifier(MinMaxScaler().
        ↪fit_transform(doc2vec_data), ground_truth, "Doc2Vec", MultinomialNB())
```

```
Accuracy for Doc2Vec: 0.9080
Precision for Doc2Vec: 0.8706
Recall for Doc2Vec: 0.9080
F1 Score for Doc2Vec: 0.8646
```



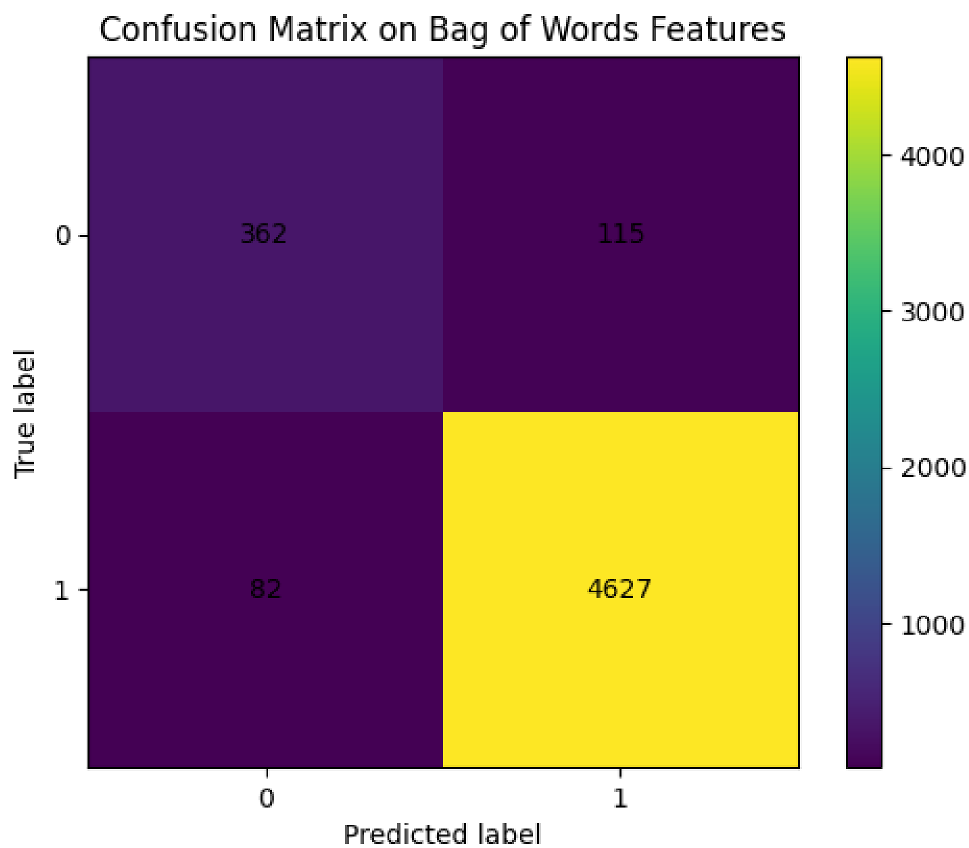
4.2. Support Vector Machine Classifier

Next, we apply the Support Vector Machine (SVM) classifier to the feature matrices.

We use `SVC` with a linear kernel, which is often effective for text classification tasks.

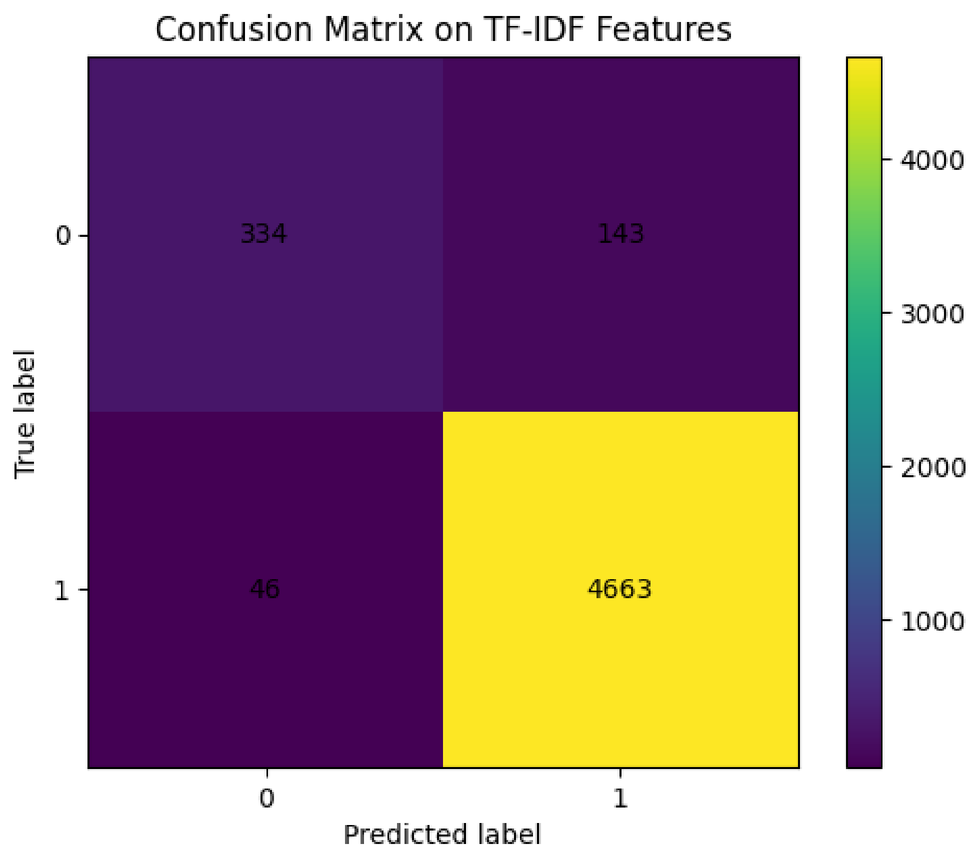
[94]:
```
f1_scores["svm"]["bow"] = classifier(bow_data, ground_truth, "Bag of Words",␣
↪SVC(kernel='linear'))
```

```
Accuracy for Bag of Words: 0.9620
Precision for Bag of Words: 0.9610
Recall for Bag of Words: 0.9620
F1 Score for Bag of Words: 0.9614
```



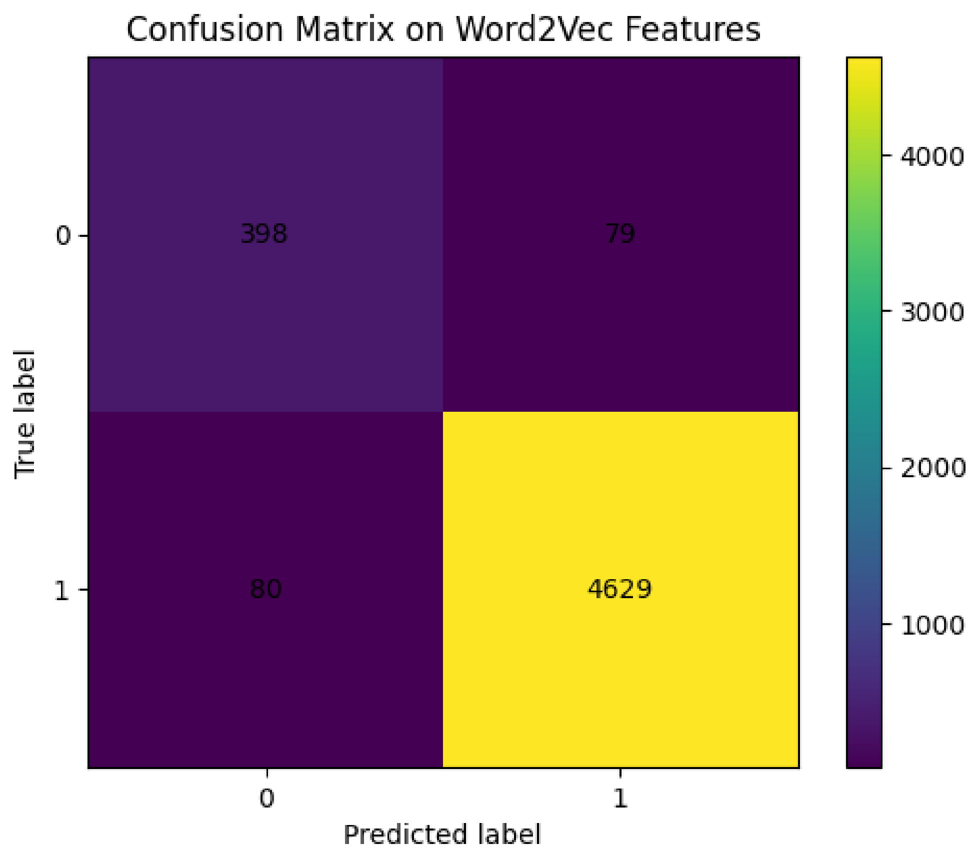
[95]:
```
f1_scores["svm"]["tfidf"] = classifier(tfidf_data, ground_truth, "TF-IDF",␣
    ↪SVC(kernel='linear'))
```

```
Accuracy for TF-IDF: 0.9636
Precision for TF-IDF: 0.9618
Recall for TF-IDF: 0.9636
F1 Score for TF-IDF: 0.9617
```



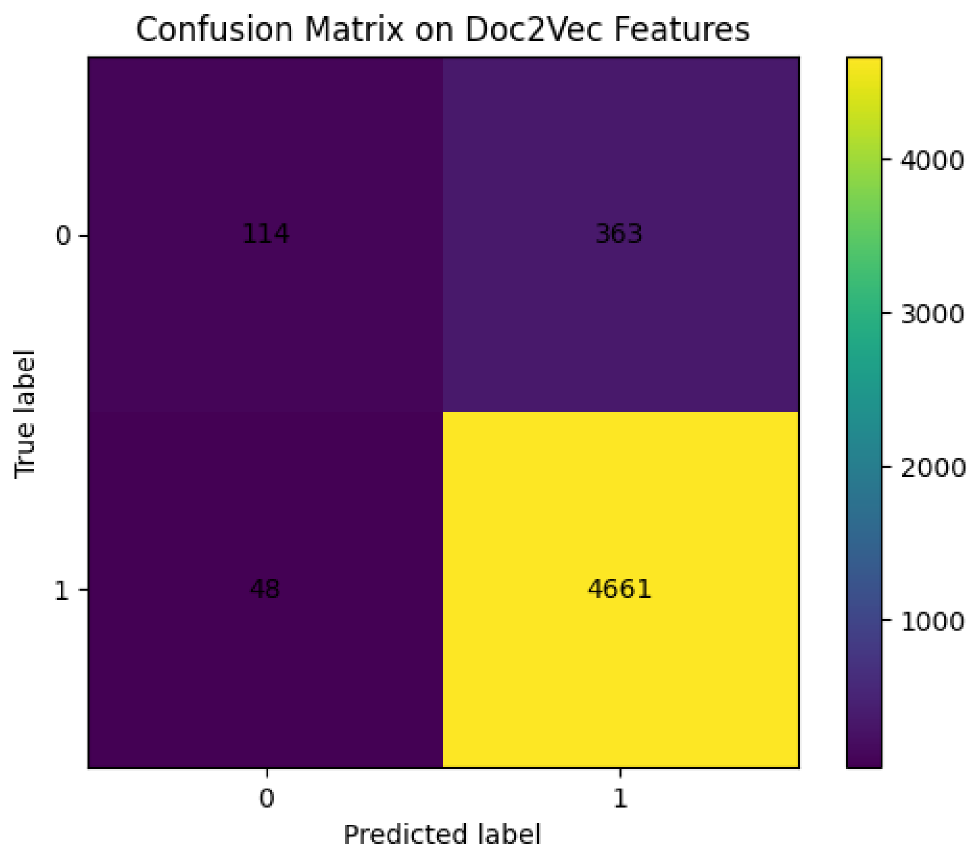
[96]:
```
f1_scores["svm"]["word2vec"] = classifier(word2vec_data, ground_truth,␣
↪"Word2Vec", SVC(kernel='linear'))
```

```
Accuracy for Word2Vec: 0.9693
Precision for Word2Vec: 0.9694
Recall for Word2Vec: 0.9693
F1 Score for Word2Vec: 0.9694
```



[97]:
```
f1_scores["svm"]["doc2vec"] = classifier(doc2vec_data, ground_truth, "Doc2Vec",␣
↪SVC(kernel='linear'))
```

```
Accuracy for Doc2Vec: 0.9207
Precision for Doc2Vec: 0.9071
Recall for Doc2Vec: 0.9207
F1 Score for Doc2Vec: 0.9025
```



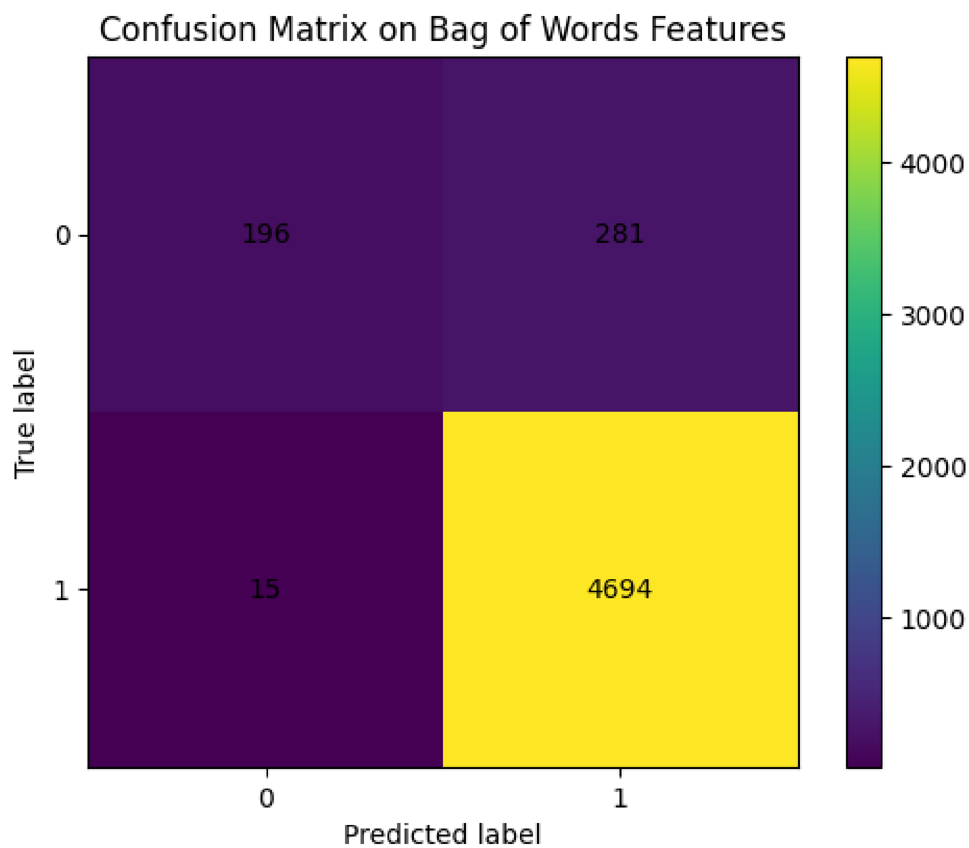
4.3. Random Forest Classifier

Finally, we apply the Random Forest classifier to the feature matrices.

We use `RandomForestClassifier`, which is an ensemble method that can handle high-dimensional data and is robust to overfitting.

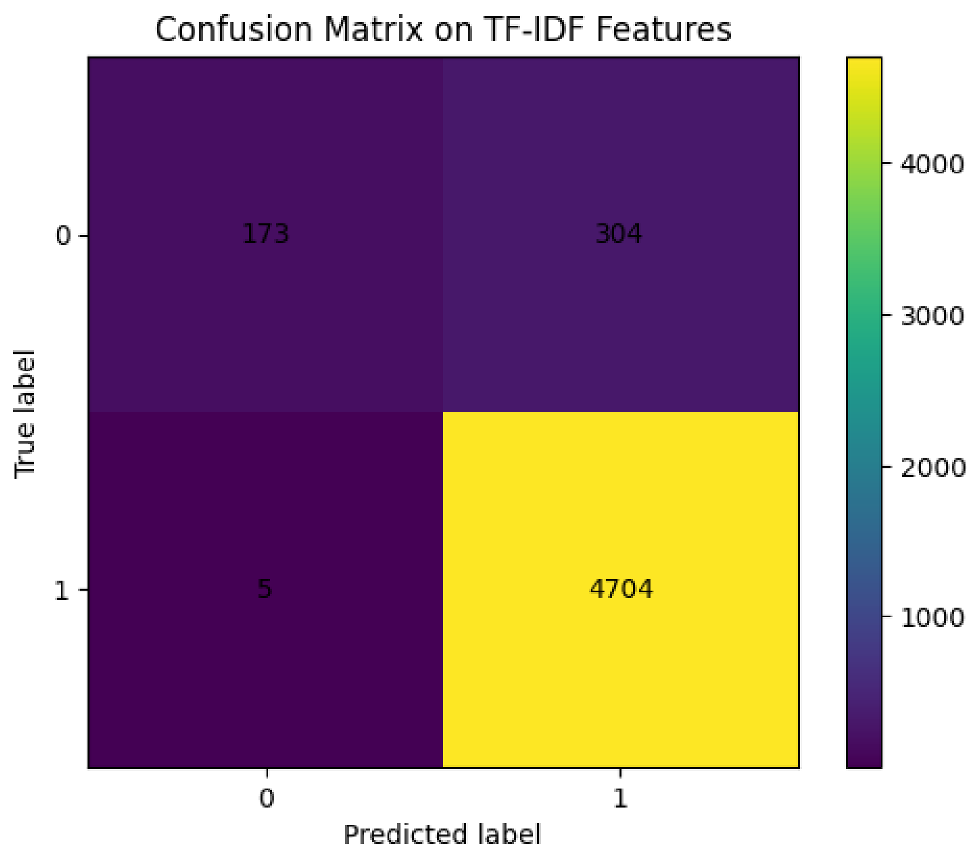
[98]:
```
f1_scores["random_forest"]["bow"] = classifier(bow_data, ground_truth, "Bag of
↪Words", RandomForestClassifier())
```

```
Accuracy for Bag of Words: 0.9429
Precision for Bag of Words: 0.9422
Recall for Bag of Words: 0.9429
F1 Score for Bag of Words: 0.9327
```



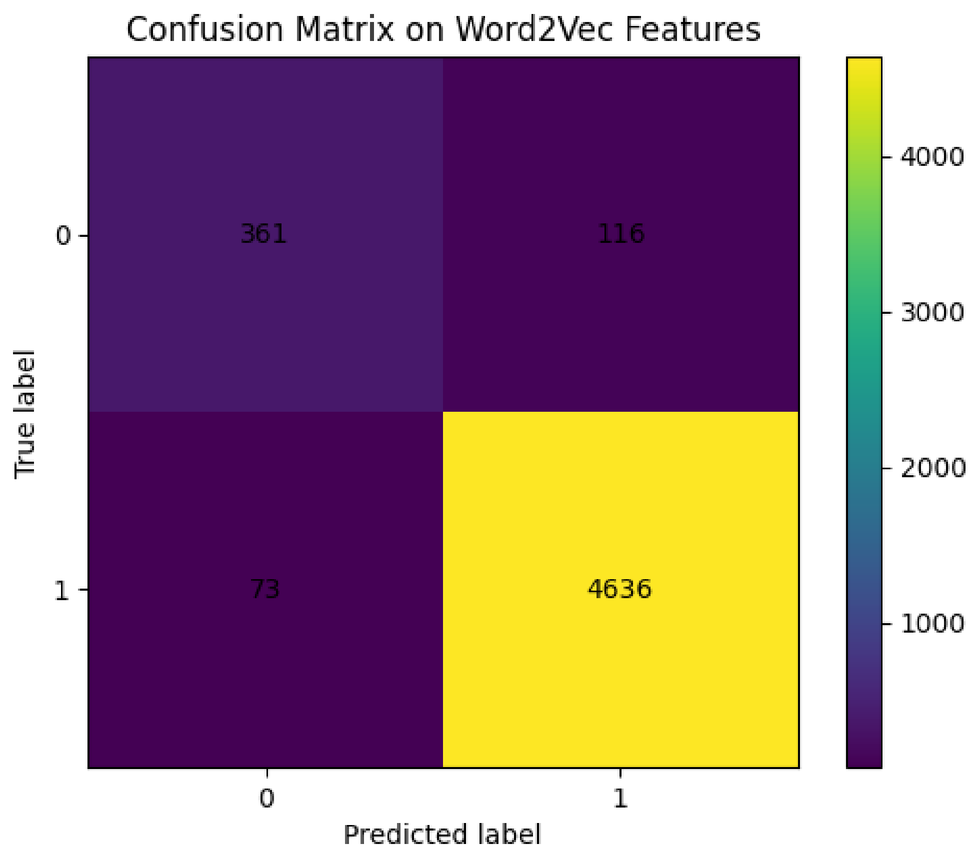
[99]:
```
f1_scores["random_forest"]["tfidf"] = classifier(tfidf_data, ground_truth,
    "TF-IDF", RandomForestClassifier())
```

```
Accuracy for TF-IDF: 0.9404
Precision for TF-IDF: 0.9423
Recall for TF-IDF: 0.9404
F1 Score for TF-IDF: 0.9277
```



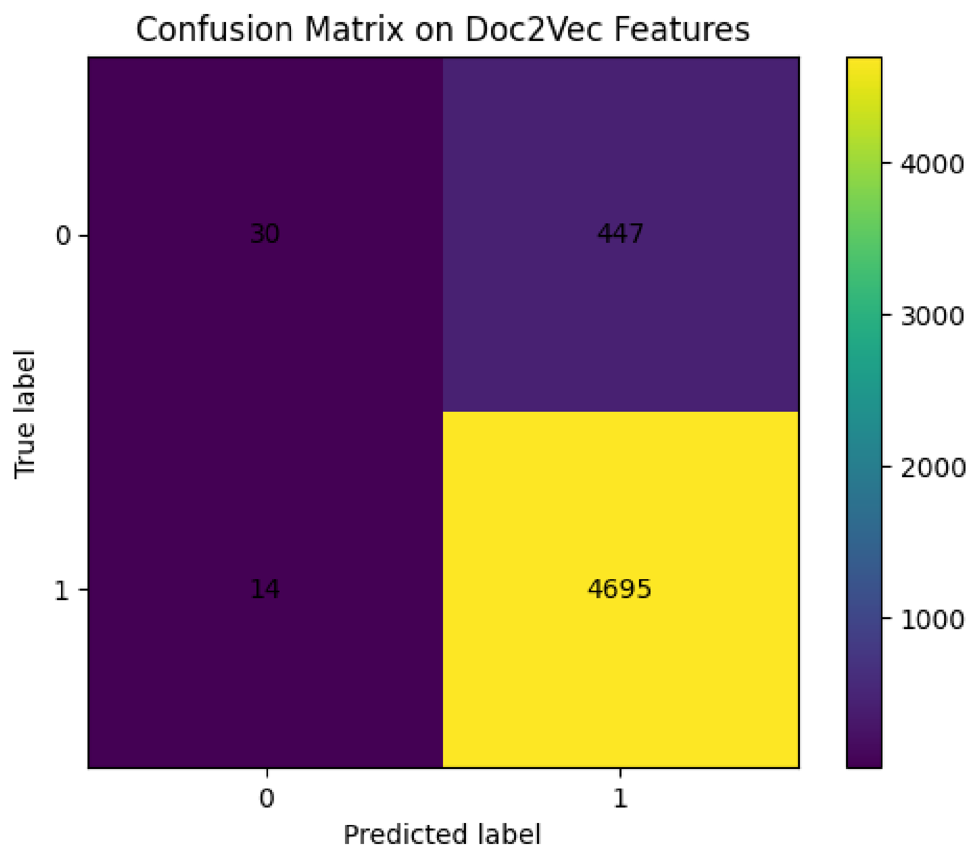
[100]:
```
f1_scores["random_forest"]["word2vec"] = classifier(word2vec_data,
    ground_truth, "Word2Vec", RandomForestClassifier())
```

```
Accuracy for Word2Vec: 0.9636
Precision for Word2Vec: 0.9624
Recall for Word2Vec: 0.9636
F1 Score for Word2Vec: 0.9628
```



[101]:
```
f1_scores["random_forest"]["doc2vec"] = classifier(doc2vec_data, ground_truth,␣
↪"Doc2Vec", RandomForestClassifier())
```

```
Accuracy for Doc2Vec: 0.9111
Precision for Doc2Vec: 0.8918
Recall for Doc2Vec: 0.9111
F1 Score for Doc2Vec: 0.8761
```



Finally, we print out the F1 scores for each classifier and feature matrix combination in a DataFrame for easy comparison.

[102]:
```
f1_df = pd.DataFrame(f1_scores)
print(f1_df)
```

```
          naive_bayes       svm  random_forest
bow          0.959209  0.961399       0.932673
tfidf        0.931996  0.961679       0.927734
word2vec     0.951042  0.969355       0.962778
doc2vec      0.864625  0.902497       0.876121
```

From the results, we can see that the SVM classifier with Word2Vec has the highest F1 score.

Task 5 - Using Pretrained Vectors

First load the ground truth dataset which contains the reviews and their corresponding sentiment labels.

[103]:
```
reviews = pd.read_csv('ground_truth_reviews.csv')
ground_truth = reviews['ground_truth_sentiment'].values
```

5.1. Using BERT for Text Embeddings

Here, we choose to go with BERT (Bidirectional Encoder Representations from Transformers) for generating text embeddings. BERT is a powerful transformer-based model that captures the context of words in a sentence.

[104]:
```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')
```

We can use BERT to generate embeddings for each review. The embeddings will be the output of the [CLS] token, which is designed to capture the overall meaning of the text.

[105]:
```
def get_bert_embedding(text):
    inputs = tokenizer(text, return_tensors='pt', truncation=True,␣
 ↪padding=True, max_length=128)
    with torch.no_grad():
        outputs = model(**inputs)
    cls_embedding = outputs.last_hidden_state[:, 0, :].squeeze().numpy()
    return cls_embedding
```

Then we can apply this function to the reviews in our dataset to generate the embeddings.

[106]:
```
reviews = pd.read_csv('ground_truth_reviews.csv')
bert_vectors = reviews['review'].apply(get_bert_embedding)
bert_vectors = np.vstack(bert_vectors.values)
```

[107]:
```
print(bert_vectors.shape)
```

```
(5186, 768)
```

Here, we can see that the shape of `bert_vectors` is `(5186, 768)`, where 768 is the dimensionality of the BERT embeddings.

5.2. Classifying with BERT Embeddings

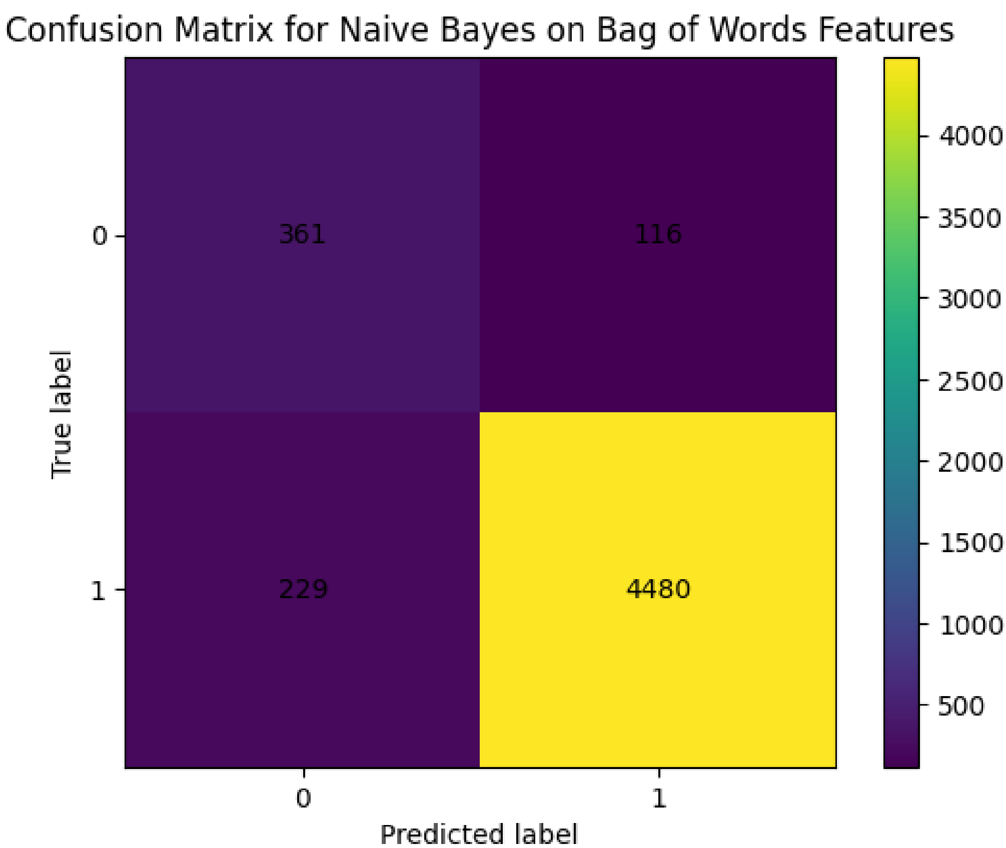
Here, similar to task 4, we will use the BERT embeddings to train various classifiers and evaluate their performance. We will use Naive Bayes, SVM, and Random Forest.

[108]:
```
def classifier(features, ground_truth, name, clf):
    y_pred = cross_val_predict(clf, features, ground_truth, cv=5)
    print(f'Accuracy for {name}: {accuracy_score(ground_truth, y_pred):.4f}')
    print(f"Precision for {name}: {precision_score(ground_truth, y_pred,␣
 ↪average='weighted'):.4f}")
    print(f"Recall for {name}: {recall_score(ground_truth, y_pred,␣
 ↪average='weighted'):.4f}")
    print(f"F1 Score for {name}: {f1_score(ground_truth, y_pred,␣
 ↪average='weighted'):.4f}")

    cm = confusion_matrix(ground_truth, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm)
```

```
    disp.plot(text_kw={'color': 'black'})
    plt.title(f'Confusion Matrix for Naive Bayes on {name} Features')
    plt.show()

    return f1_score(ground_truth, y_pred, average='weighted')
```

[109]:
```
classifier(MinMaxScaler().fit_transform(bert_vectors), ground_truth, "Bag of␣
 ↪Words", MultinomialNB())
```

```
Accuracy for Bag of Words: 0.9335
Precision for Bag of Words: 0.9414
Recall for Bag of Words: 0.9335
F1 Score for Bag of Words: 0.9366
```

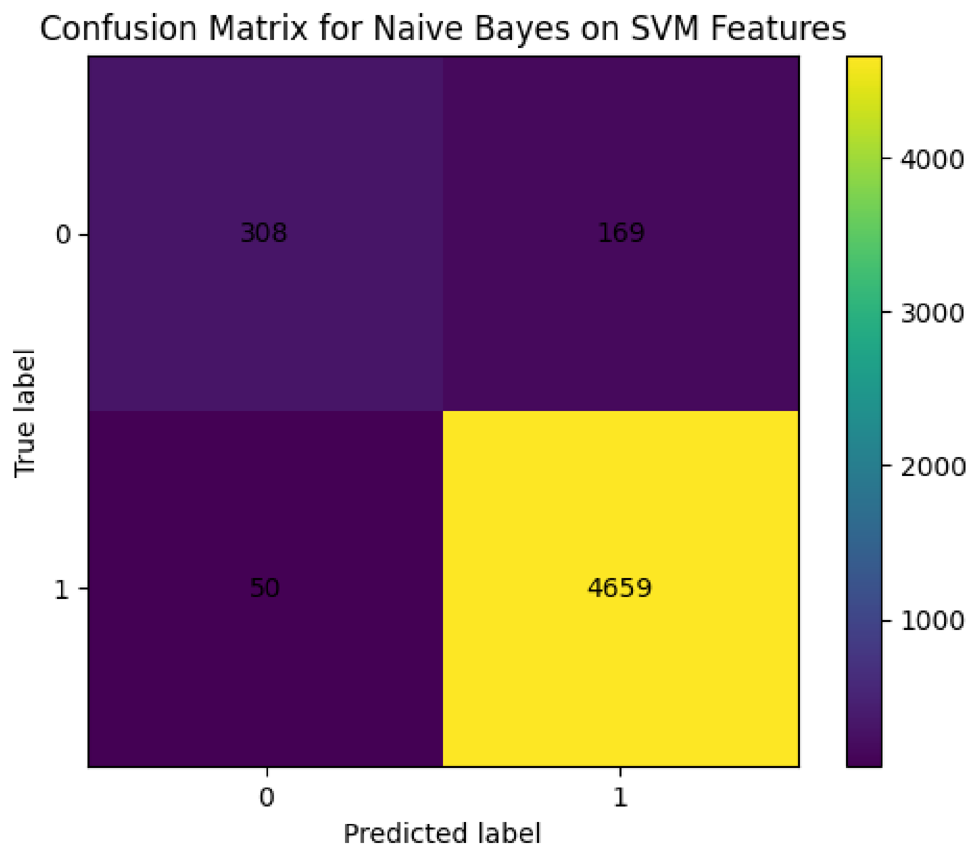


[109]: 0.9365934570685694

[110]:
```
classifier(bert_vectors, ground_truth, "SVM", SVC(kernel='rbf'))
```

```
Accuracy for SVM: 0.9578
Precision for SVM: 0.9554
Recall for SVM: 0.9578
```

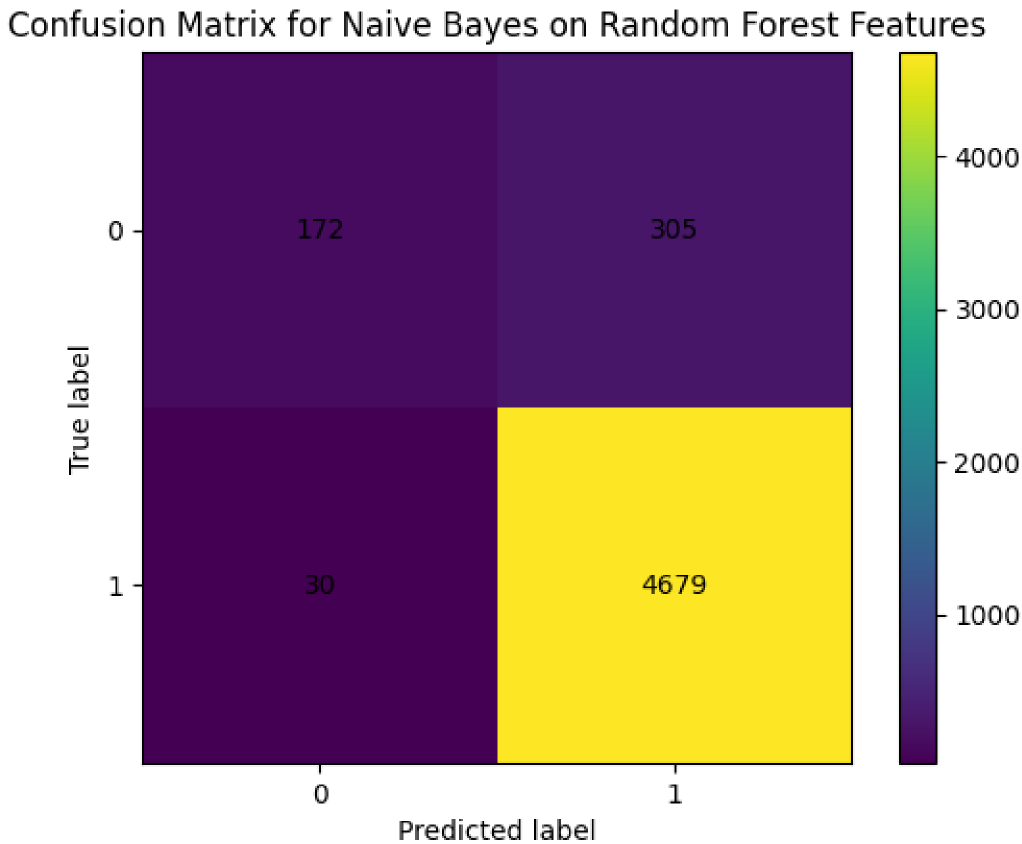
```
F1 Score for SVM: 0.9550
```



[110]: 0.9550252450711283

[111]:
```
classifier(bert_vectors, ground_truth, "Random Forest",␣
↪RandomForestClassifier())
```

```
Accuracy for Random Forest: 0.9354
Precision for Random Forest: 0.9308
Recall for Random Forest: 0.9354
F1 Score for Random Forest: 0.9232
```



[111]: 0.9232382196854205

We see that the SVM classifier performs the best with an F1 score of 0.955. But we were able to achieve a better performance using Word2Vec and SVM.

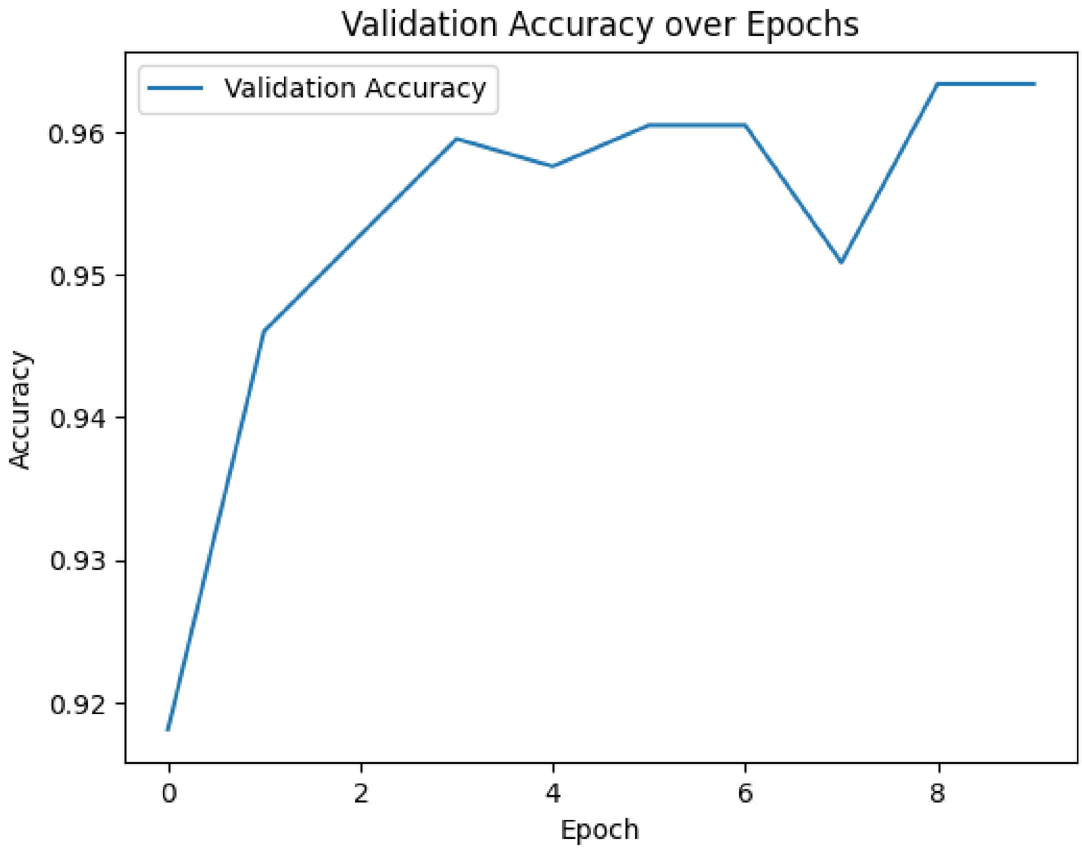
5.3. Using BERT Embeddings in a Neural Network

Here, we build and train a simple neural network using the BERT embeddings. The architecture consists of two hidden layers with ReLU activation and dropout for regularization. The hyperparameters were tuned to achieve the best performance after several trials.

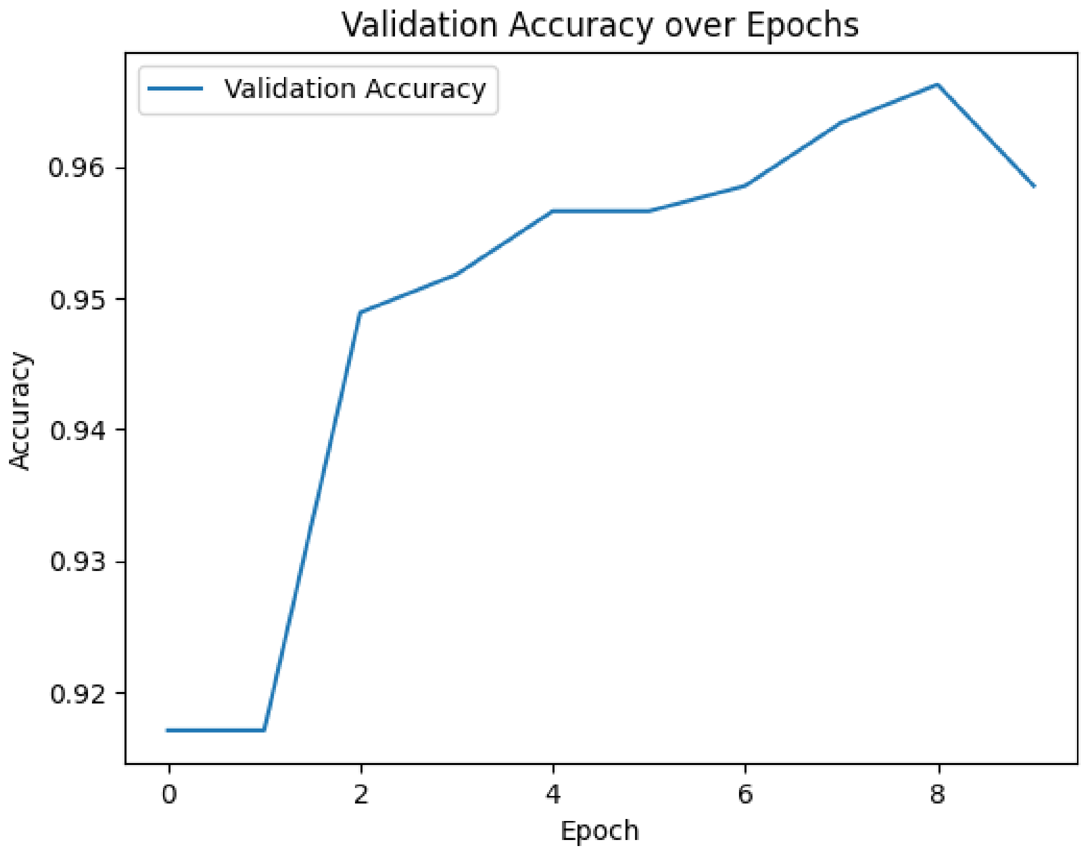
[112]:
```
X = bert_vectors
le = LabelEncoder()
y = le.fit_transform(ground_truth)
y_cat = to_categorical(y, num_classes=len(np.unique(y)))
kf = KFold(n_splits=5, shuffle=True, random_state=42)
acc_scores, prec_scores, rec_scores, f1_scores = [], [], [], []

for train_idx, val_idx in kf.split(X):
    X_train, X_val = X[train_idx], X[val_idx]
    y_train, y_val = y_cat[train_idx], y_cat[val_idx]
```

```
    model = Sequential([
        Input(shape=(X_train.shape[1],)),
        Dense(512, activation='relu'),
        Dropout(0.6),
        Dense(512, activation='relu'),
        Dropout(0.6),
        Dense(2, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])

    history = model.fit(
        X_train, y_train,
        epochs=10,
        batch_size=512,
        validation_data=(X_val, y_val),
        verbose=0
    )

    # Plot validation accuracy
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.title('Validation Accuracy over Epochs')
    plt.legend()
    plt.show()

    y_val_pred = model.predict(X_val)
    y_val_pred_labels = np.argmax(y_val_pred, axis=1)
    y_val_true_labels = np.argmax(y_val, axis=1)

    acc = accuracy_score(y_val_true_labels, y_val_pred_labels)
    prec = precision_score(y_val_true_labels, y_val_pred_labels,␣
↪average='weighted', zero_division=0)
    rec = recall_score(y_val_true_labels, y_val_pred_labels,␣
↪average='weighted', zero_division=0)
    f1 = f1_score(y_val_true_labels, y_val_pred_labels, average='weighted',␣
↪zero_division=0)

    acc_scores.append(acc)
    prec_scores.append(prec)
    rec_scores.append(rec)
    f1_scores.append(f1)

    print(f'Fold: Accuracy={acc:.4f}, Precision={prec:.4f}, Recall={rec:.4f},␣
↪F1={f1:.4f}')
```

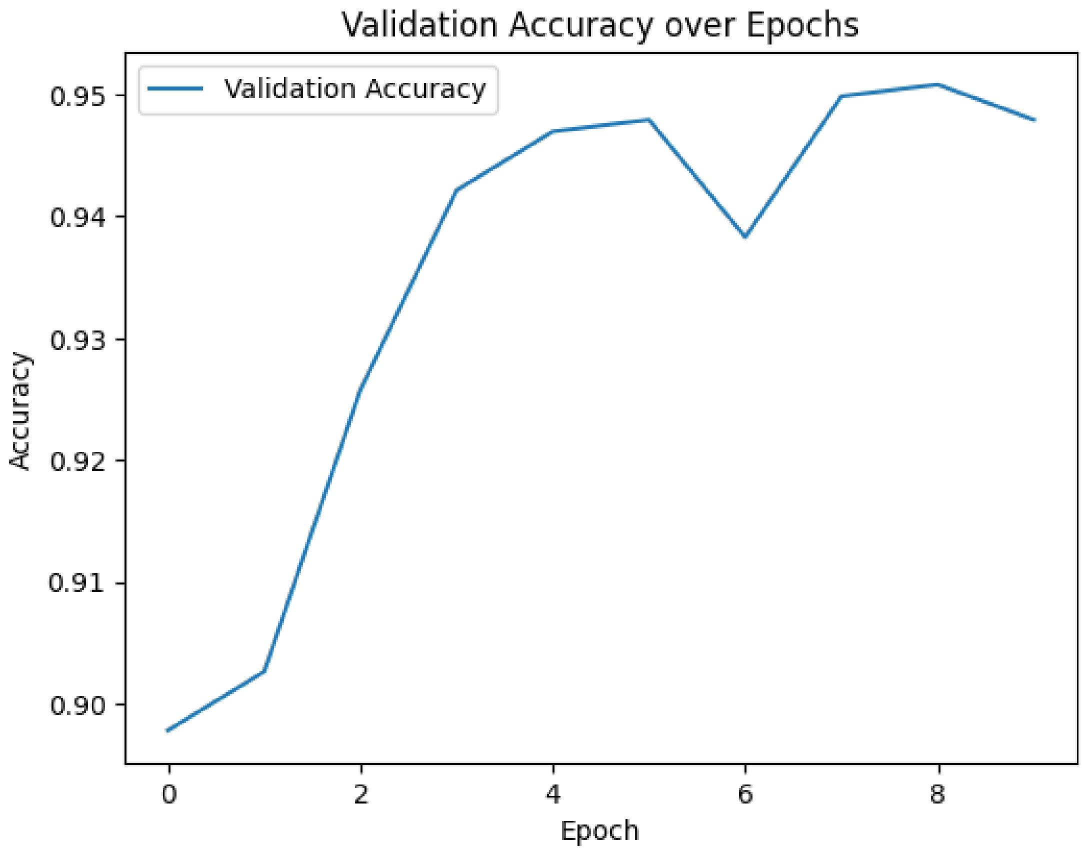
```
print(f'Mean Accuracy: {np.mean(acc_scores):.4f}')
print(f'Mean Precision: {np.mean(prec_scores):.4f}')
print(f'Mean Recall: {np.mean(rec_scores):.4f}')
print(f'Mean F1 Score: {np.mean(f1_scores):.4f}')
```



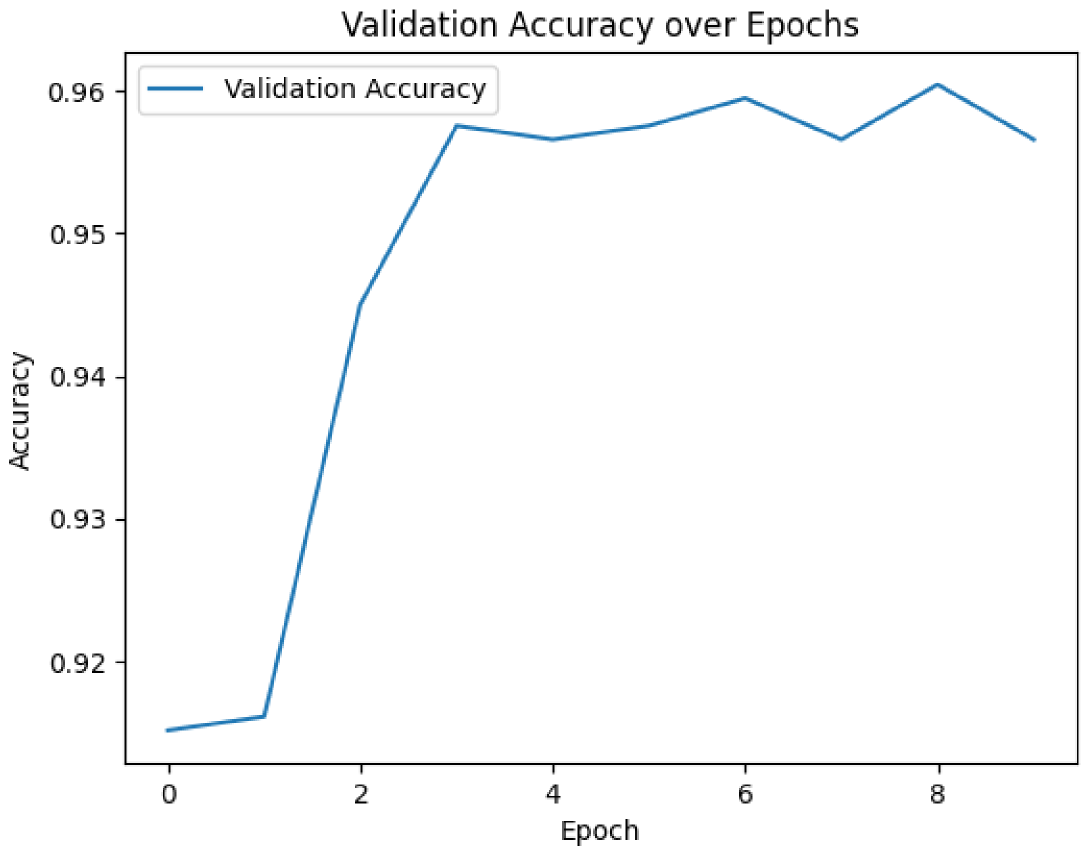

```
33/33              0s 4ms/step
Fold: Accuracy=0.9634, Precision=0.9656, Recall=0.9634, F1=0.9643
```



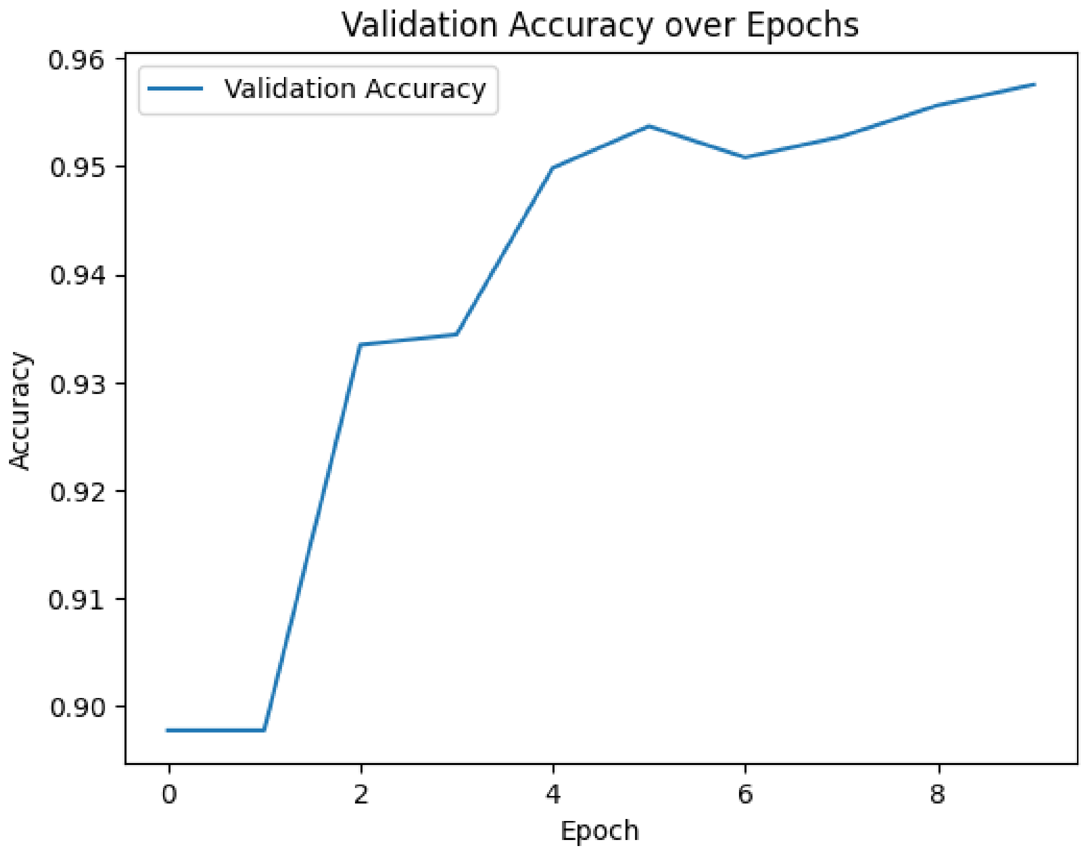

```
33/33              0s 4ms/step
Fold: Accuracy=0.9585, Precision=0.9575, Recall=0.9585, F1=0.9580
```




```
33/33              0s 4ms/step
Fold: Accuracy=0.9479, Precision=0.9450, Recall=0.9479, F1=0.9456
```




```
33/33              0s 4ms/step
Fold: Accuracy=0.9566, Precision=0.9549, Recall=0.9566, F1=0.9556
```




```
33/33              0s 4ms/step
Fold: Accuracy=0.9576, Precision=0.9579, Recall=0.9576, F1=0.9577
Mean Accuracy: 0.9568
Mean Precision: 0.9562
Mean Recall: 0.9568
Mean F1 Score: 0.9562
```

Here, we see that the neural network achieves a mean F1 score of 0.958, which is comparable to the SVM classifier.

The plots of the validation accuracy over epochs show that the model is learning and improving its performance over time, but there is diminishing returns after around 5 epochs.

Task 6 - Text Clustering as a Proxy for Ground Truth

Once again, we will use the cleaned reviews dataset from Task 1.

[113]:
```
reviews = pd.read_csv('cleaned_reviews.csv')["review"]
```

6.1. Topic Modelling with LDA

We first need to vectorize the text data. We will use a `CountVectorizer` to convert the text into a bag-of-words model, filtering out very common and very rare words.

[114]:
```
vectorizer = CountVectorizer(max_df=0.9, min_df=5, stop_words='english')
X_counts = vectorizer.fit_transform(reviews)
```

Next, we use LDA to find topics in the reviews.

The value for number of topics was set to 4 after some brief trial and error. This is not a hard limit, but it seems to work well for this dataset.

I've set my RGU ID as the random state for reproducibility.

[115]:
```
n_topics = 4

lda = LatentDirichletAllocation(n_components=n_topics, random_state=2506673) #
 ↪This is my RGU ID lol
lda.fit(X_counts)
```

[115]:
```
LatentDirichletAllocation(n_components=4, random_state=2506673)
```

Next, for each topic, we will find the words that are most unique to that topic compared to the others. This will help us understand what each topic is about.

[116]:
```
n_top_words = 15

topic_word = lda.components_
feature_names = vectorizer.get_feature_names_out()

for topic_idx in range(n_topics):
    # Mean of other topics
    other_topics = np.delete(topic_word, topic_idx, axis=0)
    diff = topic_word[topic_idx] - other_topics.mean(axis=0)
    top_indices = np.argsort(diff)[-n_top_words:][::-1]
    print(f"Words unique to Topic {topic_idx + 1}:")
    print(" ".join([feature_names[i] for i in top_indices]))
    print()
```

```
Words unique to Topic 1:
service stay experience thank hospitality special excellent staff amazing
wonderful thanks team great highly mr

Words unique to Topic 2:
good nice hotel clean rooms great friendly staff place location helpful pool
beach view food

Words unique to Topic 3:
room didnt dont hotel bad water asked night booked said bathroom poor bed dirty
worst
```

```
Words unique to Topic 4:
beautiful sri house views lovely tea peaceful lanka villa kandy nature lankan
relaxing garden perfect
```

Based on these results, we can assign the topics to the following categories: - Topic 1 - Positive feedback on hotel staff and hospitality - Topic 2 - Positive feedback on hotel amenities and services - Topic 3 - Negative feedback - Topic 4 - Positive feedback on hotel location and surrounding (indirect to the hotel itself)

> Note: Regardless of any different combination I tried, I was unable to cluster for topics based on different aspects.

6.2. Manual Labeling of Reviews

A random set of 50 reviews were selected, and labelled according to the four topics identified above. The labels were assigned based on the most prominent topic in each review.

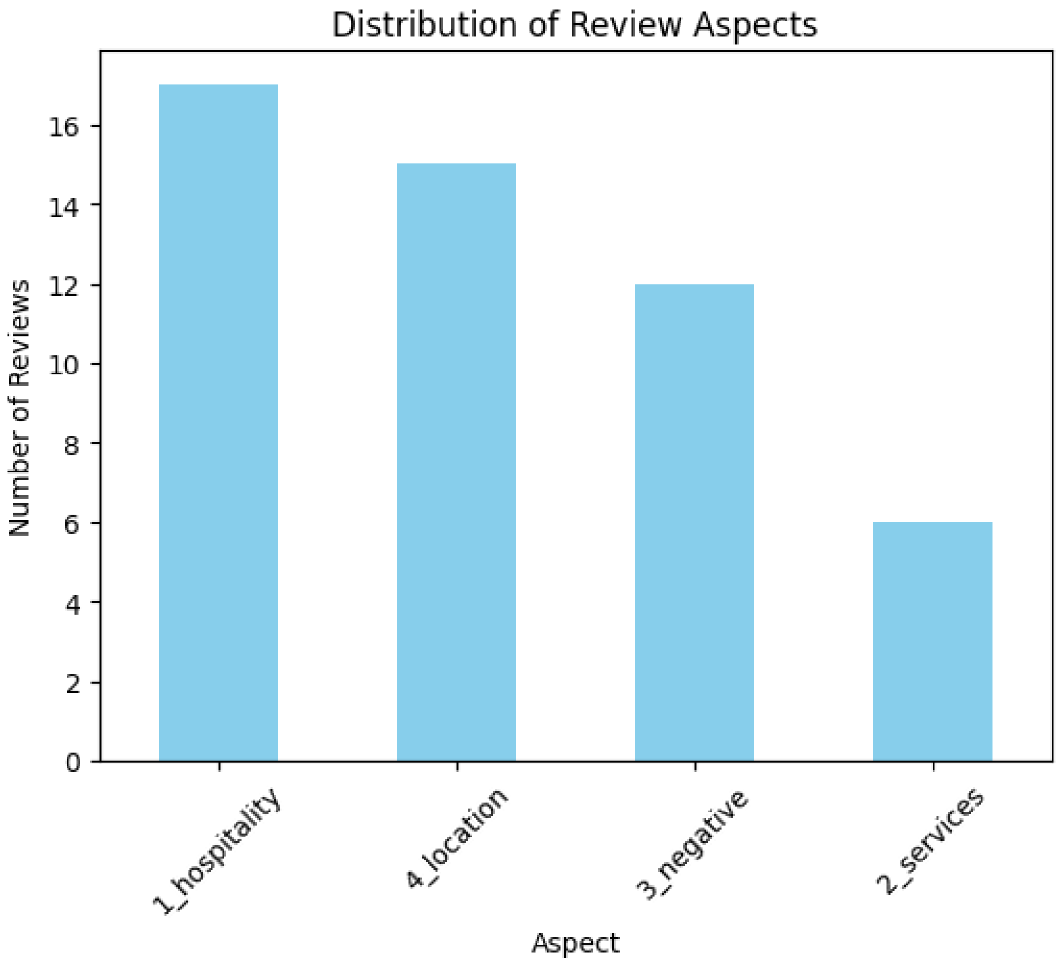
[117]:
```python
labeled_reviews = pd.read_csv('cleaned_reviews_labelled_aspect.csv')
labeled_reviews.head()
```

[117]:
```
    review_id  location_id                 hotel_name     city  \
0   697985229      3220199               Subhas Hotel   Jaffna
1  1015796423       306381  Ramada by Wyndham Colombo  Colombo
2   827297036      5863531               Royal Castle  Negombo
3   997068714     23326905              Kenrish Hotel  Wadduwa
4  1015645770      2510666    Jetwing Lagoon Wellness  Negombo

                                              review  rating         aspect
0  landed perfect hotel jaffna located prime loca…       4     4_location
1  great service stay great inusha madhavi prasan…       5  1_hospitality
2  honey moon night stayed two nights honeymoonwe…       5     2_services
3  wcc 87 al batch getogether exciting moment ken…       5     4_location
4  spa jetwing lagoon superb head shoulder foot m…       5     2_services
```

As seen above, there is a new column added named `aspect`, which contains the topic label for each review.

We can visualize the distribution of these labels to see how many reviews fall into each category.

[118]:
```python
labeled_reviews['aspect'].value_counts().plot(kind='bar', color='skyblue')
plt.title('Distribution of Review Aspects')
plt.xlabel('Aspect')
plt.ylabel('Number of Reviews')
plt.xticks(rotation=45)
plt.show()
```



We can see that this dataset is imbalanced, with the majority of reviews falling into the “1. Positive feedback on hotel staff and hospitality” topic, and with only 5 reviews in the “2. Positive feedback on hotel amenities and services” topic.

6.3. Evaluating Clustering with Manual Labels

Now, we will use these manually labeled reviews to validate our clustering results. We will assign the topics identified by LDA to the reviews and compare them with the manually assigned labels.

[119]:
```
new_counts = vectorizer.transform(labeled_reviews['review'])
topic_probs = lda.transform(new_counts)
assigned_topics = topic_probs.argmax(axis=1)

labeled_reviews['assigned_topic'] = assigned_topics
labeled_reviews.head()
```

[119]:

```
    review_id  location_id                  hotel_name     city  \
0   697985229      3220199                Subhas Hotel   Jaffna
1  1015796423       306381  Ramada by Wyndham Colombo  Colombo
2   827297036      5863531                Royal Castle  Negombo
3   997068714     23326905               Kenrish Hotel  Wadduwa
4  1015645770      2510666     Jetwing Lagoon Wellness  Negombo

                                              review  rating         aspect  \
0  landed perfect hotel jaffna located prime loca…       4     4_location
1  great service stay great inusha madhavi prasan…       5  1_hospitality
2  honey moon night stayed two nights honeymoonwe…       5     2_services
3  wcc 87 al batch getogether exciting moment ken…       5     4_location
4  spa jetwing lagoon superb head shoulder foot m…       5     2_services

  assigned_topic
0              1
1              0
2              1
3              3
4              0
```

Here we can see that the labels assigned by LDA are numerical indices corresponding to the topics. We will map these indices to the topic names for better readability.

[120]:

```
labeled_reviews['assigned_topic'] = labeled_reviews['assigned_topic'].
 ↪astype(str)
labeled_reviews['assigned_topic'] = labeled_reviews['assigned_topic'].map({
    "0": "1_hospitality",
    "1": "2_services",
    "2": "3_negative",
    "3": "4_location"
})

labeled_reviews.head(20)
```

[120]:

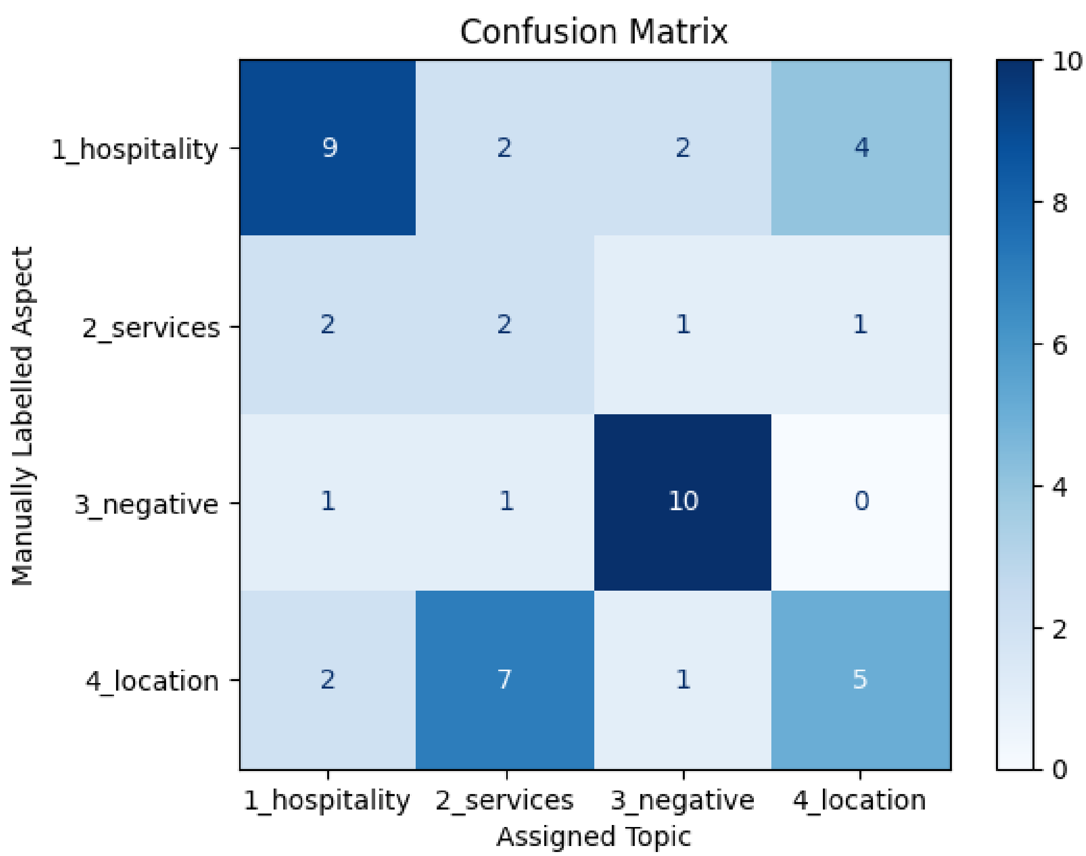
```
     review_id  location_id                          hotel_name  \
0    697985229      3220199                        Subhas Hotel
1   1015796423       306381           Ramada by Wyndham Colombo
2    827297036      5863531                        Royal Castle
3    997068714     23326905                       Kenrish Hotel
4   1015645770      2510666             Jetwing Lagoon Wellness
5    889286748      1160171      The Oasis Ayurveda Beach Resort
6    865299810     23587000                     Earl's Rajarata
7   1012507426       505588               Berjaya Hotel Colombo
8   1014148799       316671                Mermaid Hotel & Club
9    999901918     13336245                          Ahas Gawwa
10  1013826636       579219                   Pegasus Reef Hotel
11   459110674      2450524              Hotel Yapahuwa Paradise
```

```
12   634339078     1815619                         Monty Hotel
13   729160625    19516569                      The Gray Villa
14  1016177550     8116054  Anantara Peace Haven Tangalle Resort
15  1000205592    27506367      Euphoria White House Nuwaraeliya
16  1007429132     5279732               Radisson Hotel Colombo
17   430382261    10752000                     High Rich Resort
18   222893281     2450525                              Dulyana
19   986372189    13219004             The Grand Mountain Hotel

                      city                                             review  \
0                   Jaffna  landed perfect hotel jaffna located prime loca…
1                  Colombo  great service stay great inusha madhavi prasan…
2                  Negombo  honey moon night stayed two nights honeymoonwe…
3                  Wadduwa  wcc 87 al batch getogether exciting moment ken…
4                  Negombo  spa jetwing lagoon superb head shoulder foot m…
5               Hambantota  everything fine good people oasis hotel resort…
6             Anuradhapura  great hotel wonderful staff definitely returni…
7  Dehiwala-Mount Lavinia  great stay stay comfortable service welcoming …
8                 Kalutara  wonderful stay mermaid hotel club went hotel a…
9                  Padukka  quality time ahas gawwa place beautiful staff …
10                 Wattala  closes nice place pegasus reef close destiny a…
11                Yapahuwa  buffet food less quality visited august 2015 o…
12                  Ampara  ecofriendly hotel sweeping gaze paddy fields p…
13                   Galle  volunteer center great hostel volunteers worki…
14                Tangalle  nice need additional activities season beaches…
15            Nuwara Eliya  satisfied awful experience hotel staff incredi…
16                 Colombo  honeymoon stay honeymoon stay photographs sess…
17               Aluthgama  great place grab bite stayed across river reso…
18            Anuradhapura  smiles make lapses night stay business trip fi…
19                  Matale  disappointing honeymoon experience grand mount…

    rating         aspect assigned_topic
0        4     4_location     2_services
1        5  1_hospitality  1_hospitality
2        5     2_services     2_services
3        5     4_location     4_location
4        5     2_services  1_hospitality
5        5  1_hospitality     3_negative
6        5  1_hospitality     3_negative
7        5  1_hospitality  1_hospitality
8        5  1_hospitality  1_hospitality
9        5     4_location     4_location
10       5     4_location     2_services
11       3     2_services     2_services
12       4     4_location     4_location
13       3     4_location     3_negative
14       4     2_services     3_negative
```

```
15        1     3_negative  1_hospitality
16        5  1_hospitality  1_hospitality
17        5  1_hospitality     2_services
18        3     3_negative     3_negative
19        1     3_negative     3_negative
```

Now, we can clearly see the assigned topics for each review. We can compare these with the manually assigned labels to evaluate the clustering performance.

[121]:
```
conf_matrix = confusion_matrix(labeled_reviews['aspect'],␣
 ↪labeled_reviews['assigned_topic'])
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix,␣
 ↪display_labels=["1_hospitality", "2_services", "3_negative", "4_location"])
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.xlabel('Assigned Topic')
plt.ylabel('Manually Labelled Aspect')
plt.show()
```



As we can see from the confusion matrix, there are a significant number of classifications, especially for the `4_location` topic, that are misclassified as `2_services`. This indicates that the clustering is not perfect and there is some overlap between the topics.

Another explanation for this could be that the topics are not as distinct as we would like them to be, and there is some ambiguity in the reviews that makes it difficult to assign them to a single topic. We faced this same issue while manually labelling the reviews, as some reviews could be classified into multiple topics.

But for `1_hospitality` and `3_negative`, the clustering seems to be more accurate, with fewer misclassifications.

[122]:
```python
accuracy = accuracy_score(labeled_reviews['aspect'],␣
 ↪labeled_reviews['assigned_topic'])
precision = precision_score(labeled_reviews['aspect'],␣
 ↪labeled_reviews['assigned_topic'], average='weighted')
recall = recall_score(labeled_reviews['aspect'],␣
 ↪labeled_reviews['assigned_topic'], average='weighted')
f1 = f1_score(labeled_reviews['aspect'], labeled_reviews['assigned_topic'],␣
 ↪average='weighted')

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
```

```
Accuracy: 0.5200
Precision: 0.5600
Recall: 0.5200
F1 Score: 0.5287
```

Finally, we can summarize the performance of our clustering model using the usual performance metrics. We have an accuracy of 0.52, which is not very high, but it is better than random guessing. The precision, recall, and F1 score are also relatively low, indicating that the model is not performing well in distinguishing between the different topics.

We can also use the Rand Index and Jaccard Coefficient to evaluate the clustering performance.

[123]:
```python
rand_index = rand_score(labeled_reviews['aspect'],␣
 ↪labeled_reviews['assigned_topic'])
print(f"Rand Index: {rand_index:.4f}")
```

```
Rand Index: 0.6988
```

[124]:
```python
jaccard_coefficient = jaccard_score(labeled_reviews['aspect'],␣
 ↪labeled_reviews['assigned_topic'], average="macro")
print(f"Jaccard Coefficient: {jaccard_coefficient:.4f}")
```

```
Jaccard Coefficient: 0.3523
```

For Rand Index, we receive a value of 0.6988, which indicates a moderate level of agreement between the clustering and the manual labels.

For Jaccard Coefficient, we receive a value of 0.3523, which indicates a low level of agreement between the clustering and the manual labels. Since this metric is based on the size of the intersection and union of the sets - we can expect this score to be lower as the dataset is not very distinct and there is a significant amount of overlap between the topics.

6.4. Implementing an Aspect Based Sentiment Classifier

Now, we can use the LDA model we trained in 6.3, to assign cluster labels for all of the reviews.

[125]:
```
reviews = pd.read_csv('cleaned_reviews.csv')

counts = vectorizer.transform(reviews['review'])
topic_probs = lda.transform(counts)
assigned_topics = topic_probs.argmax(axis=1)
reviews["aspect"] = assigned_topics
reviews["aspect"] = reviews["aspect"].astype(str)
reviews["aspect"] = reviews["aspect"].map({
    "0": "1_hospitality",
    "1": "2_services",
    "2": "3_negative",
    "3": "4_location"
})

reviews.head()
```

[125]:
```
    review_id  location_id              hotel_name     city  \
0  1016464488     11953119  Nh Collection Colombo  Colombo
1  1016435128     11953119  Nh Collection Colombo  Colombo
2  1016307864     11953119  Nh Collection Colombo  Colombo
3  1016165618     11953119  Nh Collection Colombo  Colombo
4  1015472232     11953119  Nh Collection Colombo  Colombo

                                              review  rating         aspect
0  good stay found lighters toilet paper rolls no…       1     3_negative
1  definitely recommend hotel excellent food good…       5     2_services
2  wonderful stay comfortable staycooperative sta…       5     2_services
3  favorite 4 star hotel colombo live new york ar…       5     2_services
4  excellent food stay excellent food especially …       5  1_hospitality
```

Since in task 3 and 4, we observed that the best performance for the classification model appeared when we used a Word2Vec embeddings with a Support Vector Machine, we will be doing the same for this task as well to create an aspect based sentiment classifier.

[126]:
```
word2vec_data = pd.read_csv('feature_matrix_word2vec.csv')
ground_truth = reviews['aspect'].values
```

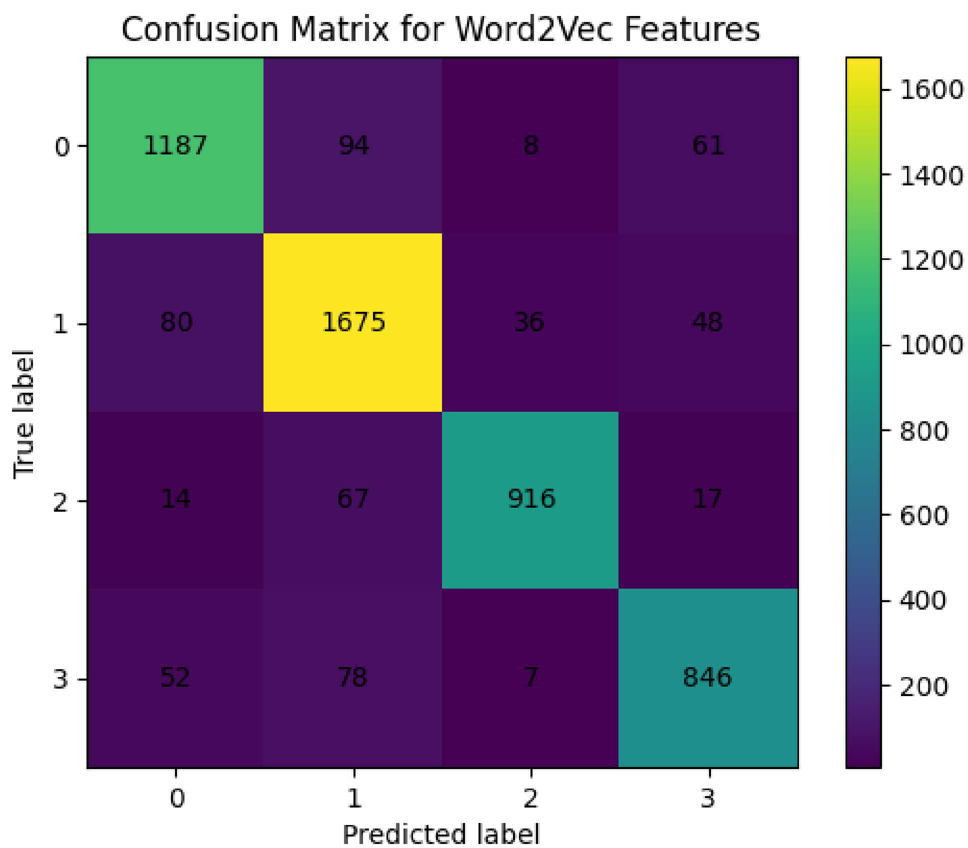
[127]:
```
def classifier(features, ground_truth, name, clf):
    y_pred = cross_val_predict(clf, features, ground_truth, cv=5)
    print(f'Accuracy for {name}: {accuracy_score(ground_truth, y_pred):.4f}')
```

```
    print(f"Precision for {name}: {precision_score(ground_truth, y_pred,␣
↪average='weighted'):.4f}")
    print(f"Recall for {name}: {recall_score(ground_truth, y_pred,␣
↪average='weighted'):.4f}")
    print(f"F1 Score for {name}: {f1_score(ground_truth, y_pred,␣
↪average='weighted'):.4f}")

    cm = confusion_matrix(ground_truth, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm)
    disp.plot(text_kw={'color': 'black'})
    plt.title(f'Confusion Matrix for {name} Features')
    plt.show()

    return f1_score(ground_truth, y_pred, average='weighted')
```

And now we can run this model against the established ground truth labels to see how well it performs.

[128]:
```
classifier(word2vec_data, ground_truth, "Word2Vec", SVC(kernel='linear'))
```

```
Accuracy for Word2Vec: 0.8916
Precision for Word2Vec: 0.8923
Recall for Word2Vec: 0.8916
F1 Score for Word2Vec: 0.8917
```



[128]: 0.8917355182703203

As it can be seen above, this model performs quite well, with an accuracy and F1-score over 89%. This is expected as the LDA model assigned reviews to a cluster based on the keywords present within it. This model is able to capture that same information quite well.