Honours Degree in Computing

Text Analytics Assessment

Submitted by: Paul Price, B00106786

28/02/2024

Declaration

I/We declare that this material, which I/We now submit for assessment, is entirely my/our own work and has not been taken from the work of others, except where otherwise stated. I/We have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references.

I/We understand that plagiarism, collusion, and copying are grave and serious offences and accept the penalties that would be imposed should I/we engage in plagiarism, collusion or copying. I acknowledge that copying someone else's assignment, or part of it, is wrong, and that submitting identical work to others constitutes a form of plagiarism. I/We have read and understood the colleges plagiarism policy 3AS08 (available here).

This material, or any part of it, has not been previously submitted for assessment for an academic purpose at this or any other academic institution.

I have not allowed anyone to copy my work with the intention of passing it off as their own work.

Name: Paul Price Dated: 28/02/2024

Contents

Overview

Background

For this assignment we will be mining a text-based dataset. The dataset we will be using for this project is 'TripAdvisor Hotel Reviews'. This dataset contains two columns, one column contains text for the reviews for various hotels, the contents of each row in this column varies from a few words to multiple sentences. The second column consists of a single number for each row that ranges between 1 and 5, this number represents the number of stars the hotel has received from the review. This is quite a large dataset as there are 20,492 rows in total, 1 row for the column's titles and 20,491 containing data entries. Using the rating numbers, we can quickly search for all positive reviews by finding all rows containing a 4 or 5 in this column or search for all negative reviews by finding rows containing a 1 or 2, 3 would be considered average. From searching these numbers individually, we retrieved the following information:

- There are 1,421 rows containing a 1-star review.
- There are 1,793 rows containing a 2-star review.
- There are 2,184 rows containing a 3-star review.
- There are 6,039 rows containing a 4-star review.
- There are 9,054 rows containing a 5-star review.

Rows with a 1-star review typically suggests that the person who wrote the review was extremely unsatisfied with their experience and would never return to the hotel again. Rows with a 2-star review suggests that the person who wrote the review was extremely unsatisfied with their experience but might've found some minor redeeming qualities that prevented them from giving 1 star. By adding the number of 1 star and 2-star reviews together, we can determine that 3,214 of the reviews are negative which makes up approximately 15.68% of the overall reviews. Rows with a 3-star review suggest the reviewer had an average experience that could be worse but also could be better. There are 2,184 average reviews which makes up approximately 10.66% of the overall reviews. Rows with a 4-star review typically suggests that the person who wrote the review had an overall positive experience and would return to the hotel again but there is still room for improvement. Rows with a 5-star review suggests that the person who wrote the review was very happy with their experience and would definitely return to the hotel again. We can get the number of positive reviews by adding the 4-star and 5-star reviews together, there are 15,093 positive reviews which makes up approximately 73.66% of the overall reviews. After gathering all of this information we can determine that there is a class imbalance as there are far more positive reviews than there are negative or average reviews.

Business Objectives

The business objectives are the potential ways this data could be used from a business standpoint.[1] Some ways that this data could be used to improve business include the following:

- **Improve customer satisfaction:** the negative reviews can be further examined to determine the source of the customers dissatisfaction so that the hotel management can come up with potential solutions to improve overall customer satisfaction and attract more customers, the higher the customer satisfaction the more likely the customers are to return as well as recommend the hotel to others. This can also be done for the 3 and 4-star reviews, the reviews can be examined so that the business can determine what their strengths are and what they may need to improve on.
- **Marketing and improve reputation:** the positive reviews can be used for marketing purposes to show potential customers that past guests had a pleasant experience during their stay and suggest that they will likely have the same experience if they chose this hotel as opposed to others with less positive reviews. The more positive reviews, the better the hotel's reputation, which will attract more customers.
- **Revenue growth:** if the previously mentioned business objectives are successful then the hotel should see an increase in business which will in turn increase their revenue.

Mining Objectives

The mining objectives are what information we hope to retrieve from mining the data in order to assist us in achieving our business objectives.[1] Our mining objectives for this dataset are:

- **Sentiment analysis:** we can classify reviews into positive, negative, and neutral sentiments based on the star ratings and textual content. This will make it easier to investigate the data and achieve our business objectives.[1]
- **Feature extraction:** we can extract important features or attributes from the textual content of the dataset that correlate with positive or negative reviews, such as cleanliness, staff friendliness, location, amenities, etc. This will make it easier for a hotel to determine their strengths and weaknesses. For example, if cleanliness were mentioned in a number of negative reviews then the hotel management know that that is an area they must improve on to increase customer satisfaction.[1]
- **Predictive modeling:** Develop predictive models to forecast future trends in customer satisfaction and identify potential issues before they escalate.

Baseline Models

Vectorization is the process of converting raw textual data into numerical vectors that can be used as input for machine learning algorithms or other mathematical models. In natural language processing (NLP), vectorization is essential because most machine learning algorithms require numerical input rather than raw text.[2] Vectorization techniques involve representing text data in a structured format that preserves important information about the text's content, such as word frequencies, semantic meaning, or relationships between words.[2] We can derive matrices from textual data using vectorization techniques. Some of these techniques include:

- **Counts** - Counts vectorization represents text documents as numerical vectors based on how many times each word appears in each document.[3]
- **Normalized counts** - Normalized counts vectorization is similar to counts vectorization, but the counts are normalized by the total number of words in each document to account for document length.[3]
- **TF-IDF (Term Frequency-Inverse Document Frequency)** – The TF-IDF measures how important a word is in a document compared to how common it is across all documents.[3]

In order to get our desired matrices we must create a .ipynb file and enter in the necessary code. We must code the following:

```
import pandas as pd, numpy as np
import nltk, re
from string import punctuation
from collections import Counter

from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')

%ls

import text_mining_utils as tmu

data = pd.read_csv('tripadvisor_hotel_reviews.csv')

data.sample(3)
```

Which gives us an output of:

```
 Volume in drive C is Acer
 Volume Serial Number is 4E51-463B

 Directory of c:\Users\paulj\OneDrive\Desktop\TextAnalyticsAssignment

29/02/2024  20:57    <DIR>          .
29/02/2024  20:57    <DIR>          ..
29/02/2024  21:02    <DIR>          __pycache__
29/02/2024  20:57             6,560 text_mining_utils.py
28/02/2024  09:08            31,916 TextAnalysis_PeerReview_Template.docx
28/02/2024  09:07           160,315 TextAnalyticsAssessment2024.pdf
29/02/2024  10:58            35,865 TextAnalyticsAssignment.docx
01/03/2024  10:06            25,506 TextAnalyticsAssignment.ipynb
28/02/2024  10:49        14,966,021 tripadvisor_hotel_reviews.csv
               6 File(s)     15,226,183 bytes
               3 Dir(s)  123,949,219,840 bytes free
```

	Review	Rating
16262	great hotel staff chose stay executive suite n...	5
17149	poor food no personal service room lovely avai...	2
2794	paradisus paradise paradisus punta cana best r...	5

In the above code we must first call the necessary imports. Once we have done this, we assign the data variable to the csv file containing our database. Anytime we want to access data in our database file from here on out we now only need to call the word 'data'. We then use "data.sample(3)" to print a random sample of 3 rows from the database, this tells us whether the file is being successfully read or not. We can see from the output that the file is being successfully read as the sample containing database information has been printed. The number on the far left represents which number row was chosen from the database while the rest is the information retrieved from that row.

Once we are satisfied that the database is being read and information is being retrieved successfully we can move on to the next step.

First, we must assign a value to the clf classifier and the text column we will be analysing. We will be using the Naïve Bayes classifier for this assignment. The Naïve Bayes classifier is a supervised machine learning algorithm, which is used for classification tasks, like text classification.[4] Naive Bayes classifiers make assumptions on which group a text belongs to by determining probabilities of the text containing certain features. Naïve Bayes is commonly used for text analysis for things like sentiment analysis and spam detection.[4] Naïve Bayes is needed for our database as it is quite large and would otherwise take an extremely long time to analyse if we were to use an alternative classifier like SVC. Once we have imported Naïve Bayes, we may then assign it the value of clf for future use. Once this is done we must then assign a value to the text column we will be analysing, in this case we will assign it the value of "y". Once the necessary values have been assigned we can move onto building our baseline count matrix. Once the baseline count matrix is built we will output it's 'head', this means we will output the

first five rows of the matrix so we can determine that the baseline count matrix is being built correctly.

```
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB()
y = data.Rating

baseline_count_matrix = tmu.build_count_matrix(data.Review)
baseline_count_matrix.head()
```

Once this code is ran the out put should look like this:

	'	*	+	,	-	.	/	0	0'8	00	...	é_Çål	éenny	êtyle	î_	î_asically	î_ere	î_here	î_hese	î_ölthough	öreat
0	0	0	0	11	1	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	20	1	4	1	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	2	0	16	0	7	1	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	13	0	0	1	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	33	0	1	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 53231 columns

Each row represents a text row analysed from the database, and each column represents a word, number or symbol, the number where they intersect tells us how many times the word, number or symbol appears in that particular text. You may notice that the words in the top row are spelt using weird symbols, this is because the words are being tokenized as part of the analytics process. From the numbers at the bottom of the output image we can see that this head contains 5 rows and 53231 columns.

Once we are happy with the count matrix we can move onto building our tf matrix. Same as before we will be outputting the head to determine if the data is correct.

```
baseline_tf_matrix = tmu.build_tf_matrix(data.Review)
baseline_tf_matrix.head()
```

This gives us the following output:

	'	*	+	,	-	.	/	0	0'8	00	...	é_Çål	éenny	êtyle	î_	î_asically	î_ere	î_here	î_hese	î_ölthough	öreat
0	0.0	0.000	0.0	0.110000	0.010000	0.000000	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.000	0.0	0.070922	0.003546	0.014184	0.003546	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.008	0.0	0.064000	0.000000	0.028000	0.004000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.000	0.0	0.125000	0.000000	0.000000	0.009615	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.000	0.0	0.146667	0.000000	0.004444	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 53231 columns

The tf matrix is similar to the count matrix but instead of us getting the number of times the word appears in the text we are instead getting the “term frequency” which is the frequency in which the word appears in the text. The term frequency is calculated by dividing the number of times a word appears in the document by the total number of words in the document.[3]

Once we have calculated the term frequency we must then calculate the tf-idf (Term Frequency-Inverse Document Frequency). This is calculated by multiplying the term frequency by the

inverse document frequency. The inverse document frequency is how important a word, number or symbol is to the entire database and it is calculated by dividing the total number of text rows in the database by the number of text rows containing the given word, number or symbol.[3] To build our tf-idf matrix and output its head we must run the following code:

```
baseline_tfidf_matrix = tmu.build_tfidf_matrix(data.Review)
baseline_tfidf_matrix.head()
```

This will give us an output of:

	'	*	+	,	-	.	/	0	0'8	00	...	é_Çål	éenny	êtyle	î_	î_asically	î_ere	î_here	î_hese	î_ölthough	öreat
0	0.0	0.000000	0.0	0.259180	0.046176	0.000000	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.000000	0.0	0.208208	0.020402	0.062704	0.026245	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.113073	0.0	0.172938	0.000000	0.113930	0.027249	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.000000	0.0	0.251560	0.000000	0.000000	0.048785	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.000000	0.0	0.389064	0.000000	0.017753	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 53231 columns

Now that we have created our three matrices and chosen Naïve Bayes as our baseline model, we can output the classification report to see a summarized list of metrics derived from analysing the text column. A classification report will contain the following information:

- **Accuracy** – this is the measure of how correct, consistent, and complete the data in the is.
- **Precision** – This is the ratio of how often positive data is correctly predicted.
- **Recall** – This is similar to precision, but where precision focus' on the accuracy of positive predictions recall focus' on capturing as many actual positive instances as possible. This mean that whilst precision tackles the issue of false positives, recall is tackling the issue of false negatives.
- **F1-Score** – This is a mix of precision and recall, giving a balanced view of the model's performance.

We can print our classification report using the following code:

```
tmu.printClassifReport(clf, baseline_count_matrix, y)
tmu.printClassifReport(clf, baseline_tf_matrix, y)
tmu.printClassifReport(clf, baseline_tfidf_matrix, y)
```

Which gives us an output of:

```
              precision    recall  f1-score   support

           1       0.74      0.37      0.50      1421
           2       0.41      0.34      0.37      1793
           3       0.29      0.07      0.11      2184
           4       0.43      0.50      0.46      6039
           5       0.68      0.80      0.74      9054

    accuracy                           0.57     20491
   macro avg       0.51      0.42      0.44     20491
weighted avg       0.55      0.57      0.54     20491

              precision    recall  f1-score   support

           1       0.00      0.00      0.00      1421
           2       0.00      0.00      0.00      1793
           3       0.00      0.00      0.00      2184
           4       0.00      0.00      0.00      6039
           5       0.44      1.00      0.61      9054

    accuracy                           0.44     20491
   macro avg       0.09      0.20      0.12     20491
weighted avg       0.20      0.44      0.27     20491

              precision    recall  f1-score   support
...
    accuracy                           0.44     20491
   macro avg       0.31      0.20      0.13     20491
weighted avg       0.30      0.44      0.28     20491
```

Let's break down what these numbers mean from looking at the first line. The precision is 0.74 meaning that 74% of instances where correctly predicted positive, the recall is 0.37 meaning that 37% of instances where correctly predicted and the f1-score is 0.50 which is 50%. The following rows follow the same logic. We can see at the bottom that the accuracy is 0.57 or 57%. This will need to be improved upon. Naïve Bayes is perfect to help us analyse text data compared to other methods due to its efficiency handling high-dimensional text data, its tendency not to be slowed by irrelevant features, requiring small training datasets, and fast training and prediction.[4] We can evaluate its performance using cross validation.

```python
from sklearn.model_selection import cross_val_score

clf_count = MultinomialNB()
clf_tf = MultinomialNB()
clf_tfidf = MultinomialNB()

scores_count = cross_val_score(clf_count, baseline_count_matrix, y, cv=5)
scores_tf = cross_val_score(clf_tf, baseline_tf_matrix, y, cv=5)
scores_tfidf = cross_val_score(clf_tfidf, baseline_tfidf_matrix, y, cv=5)
```

```
print("Count Matrix:")
print("Accuracy:", scores_count.mean())
print("Standard Deviation:", scores_count.std())

print("\nTF Matrix:")
print("Accuracy:", scores_tf.mean())
print("Standard Deviation:", scores_tf.std())

print("\nTF-IDF Matrix:")
print("Accuracy:", scores_tfidf.mean())
print("Standard Deviation:", scores_tfidf.std())
```

In the above code we are creating and calling our Naïve Bayes classifiers and performing cross validation. Cross validation is where the data is split into 5 subsets (folds) and trains the classifier on 4 of them while testing it on the remaining one, repeating this process 5 times with different combinations of training and testing subsets.[5] The cv parameter specifies the number of folds for cross-validation. In this case, cv=5 means 5-fold cross-validation. We then print our results which gives us the following output:

```
Count Matrix:
Accuracy: 0.5654181149302446
Standard Deviation: 0.012990277777493908

TF Matrix:
Accuracy: 0.4418525343526157
Standard Deviation: 0.00014070972327049348

TF-IDF Matrix:
Accuracy: 0.44336538414599813
Standard Deviation: 0.00016028317569423002
```

The accuracy tells us the average accuracy across all the testing subsets for each matrix and the standard deviation tells us about the accuracy consistencies across the subset.

Data Understanding

Data understanding is the thorough examination of the structure, content, and quality of a dataset to gain insights before analysis. It includes tasks such as describing data characteristics, exploring patterns and relationships, and assessing data quality.[1]

Word/Token Statistics

One method used to better understand data is to generate word/token statistics.[6] This can help us down the line when it comes to things like sentiment analysis and feature extraction. In the context of hotel reviews it can be helpful to tell hotel owners and managers what words come up most frequently in their negative reviews, for example, if “hygiene” is one of the words most commonly associated with 1 star reviews then the hotel owners/managers know that this

is an area they need to improve. We can generate word/token statistics to find the words/tokens most commonly associated with each star rating using the following code:

```
from text_mining_utils import print_n_mostFrequent

one_star_reviews = ' '.join(data[data['Rating'] == 1]['Review'])
two_star_reviews = ' '.join(data[data['Rating'] == 2]['Review'])
three_star_reviews = ' '.join(data[data['Rating'] == 3]['Review'])
four_star_reviews = ' '.join(data[data['Rating'] == 4]['Review'])
five_star_reviews = ' '.join(data[data['Rating'] == 5]['Review'])

print_n_mostFrequent("1-star", one_star_reviews, 10)
print_n_mostFrequent("2-star", two_star_reviews, 10)
print_n_mostFrequent("3-star", three_star_reviews, 10)
print_n_mostFrequent("4-star", four_star_reviews, 10)
print_n_mostFrequent("5-star", five_star_reviews, 10)
```

This gives us an output of:

```
=== 10 most frequent tokens in 1-star ===
        Frequency of "," is: 0.10230332540501728
        Frequency of "not" is: 0.02122133464973298
        Frequency of "hotel" is: 0.020133061077951802
        Frequency of "room" is: 0.01834918996544451
        Frequency of "n't" is: 0.00939056679980254
        Frequency of "no" is: 0.008823991383565948
        Frequency of "did" is: 0.007359870753489207
        Frequency of "stay" is: 0.00677085670690661
        Frequency of "staff" is: 0.005385271283040883
        Frequency of "rooms" is: 0.005020643539918324
=== 10 most frequent tokens in 2-star ===
        Frequency of "," is: 0.1006741910497779
        Frequency of "not" is: 0.020609064036294082
        Frequency of "room" is: 0.016807353661813757
        Frequency of "hotel" is: 0.01649120311092142
        Frequency of "n't" is: 0.009322489369437727
        Frequency of "no" is: 0.00782472613458529
        Frequency of "did" is: 0.007216136324117545
        Frequency of "good" is: 0.005876448364711275
        Frequency of "stay" is: 0.005402222538372773
        Frequency of "rooms" is: 0.005354799955738923
=== 10 most frequent tokens in 3-star ===
        Frequency of "," is: 0.10125935037258464
        Frequency of "hotel" is: 0.0181071842327331
        Frequency of "not" is: 0.01694866770599425
        Frequency of "room" is: 0.01585093754022627
        Frequency of "n't" is: 0.009257405209033568
        Frequency of "good" is: 0.00878541699443626
        Frequency of "did" is: 0.006897464136047027
```

```
        Frequency of "great" is: 0.006271722184876353
        Frequency of "nice" is: 0.006235965501952315
        Frequency of "no" is: 0.00590700401905116
=== 10 most frequent tokens in 4-star ===
        Frequency of "," is: 0.10306036978608633
        Frequency of "hotel" is: 0.019449323241673075
        Frequency of "room" is: 0.013998671377819712
        Frequency of "not" is: 0.012154279717605894
        Frequency of "great" is: 0.010220488014204979
        Frequency of "good" is: 0.009220582915789286
        Frequency of "n't" is: 0.00819041934121796
        Frequency of "nice" is: 0.006641735516161464 4
        Frequency of "staff" is: 0.006421673871393224
        Frequency of "did" is: 0.006187858373826969
=== 10 most frequent tokens in 5-star ===
        Frequency of "," is: 0.10471827346110371
        Frequency of "hotel" is: 0.022947402922106574
        Frequency of "room" is: 0.01274797356069233
        Frequency of "great" is: 0.01121632581026057
        Frequency of "not" is: 0.009419948819013443
        Frequency of "staff" is: 0.008207656978136727
        Frequency of "stay" is: 0.007169750272082831
        Frequency of "n't" is: 0.00673378743502578
        Frequency of "good" is: 0.0060026304842023875
        Frequency of "just" is: 0.005427999949575383
```

Visualization

We can use visualization techniques as a better method of identifying the difference in the frequencies between the words/tokens.

Bar Charts

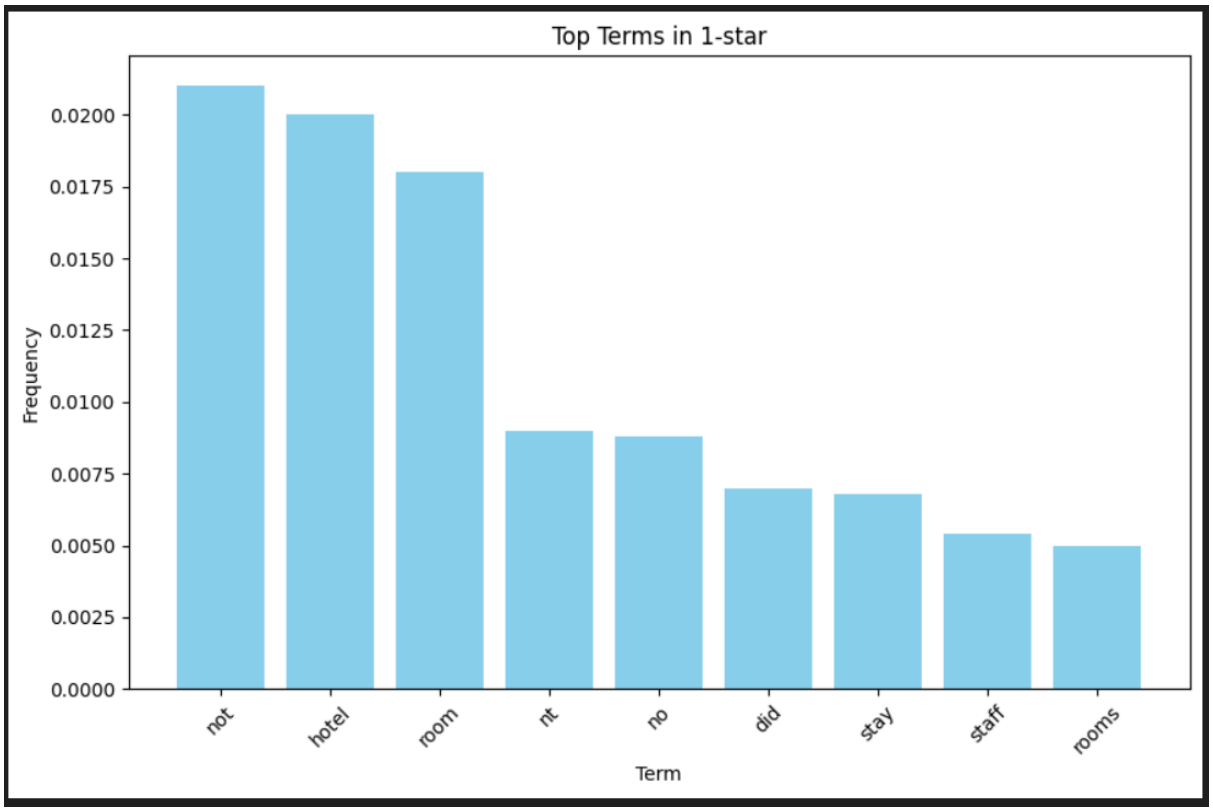
One method we can use is the creation of bar charts. We can create a bar chart for each star rating to visualize the most common words associated with each rating. For this I removed the comma from the bar chart as it didn't tell us anything and skewed the data.

1-star:

The code for creating the bar chart for the 1 star reviews is detailed below:

```
def plot_bar_chart(category, top_terms):
    plt.figure(figsize=(10, 6))
    plt.bar(range(len(top_terms)), top_terms.values(), align='center',
color='skyblue')
    plt.xticks(range(len(top_terms)), top_terms.keys(), rotation=45)
    plt.ylabel('Frequency')
    plt.xlabel('Term')
    plt.title(f'Top Terms in {category}')
    plt.show()

category = '1-star'
top_terms = { 'not': 0.021, 'hotel': 0.020, 'room': 0.018, 'nt': 0.009, 'no':
0.0088, 'did': 0.007, 'stay': 0.0068, 'staff': 0.0054, 'rooms': 0.005}

plot_bar_chart(category, top_terms)
```

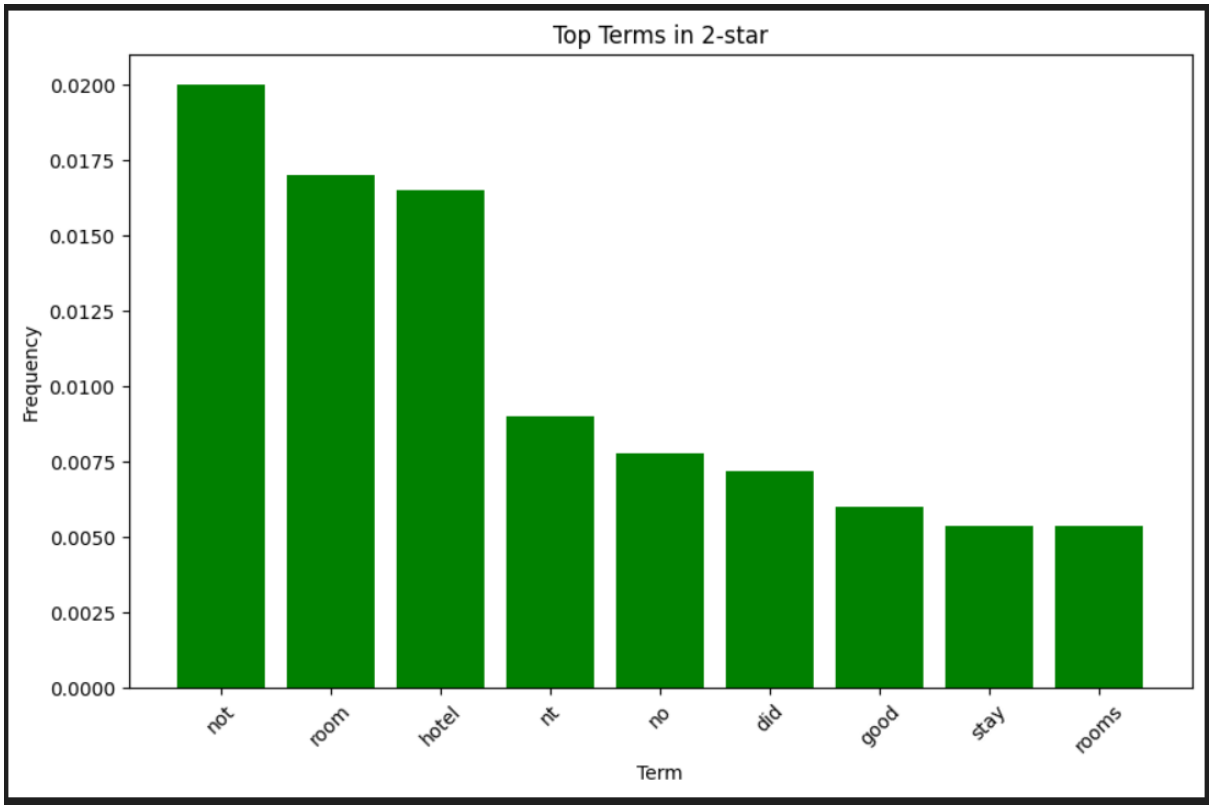
Which gives us the following:



Following the same logic we can do the same for the rest of the reviews.

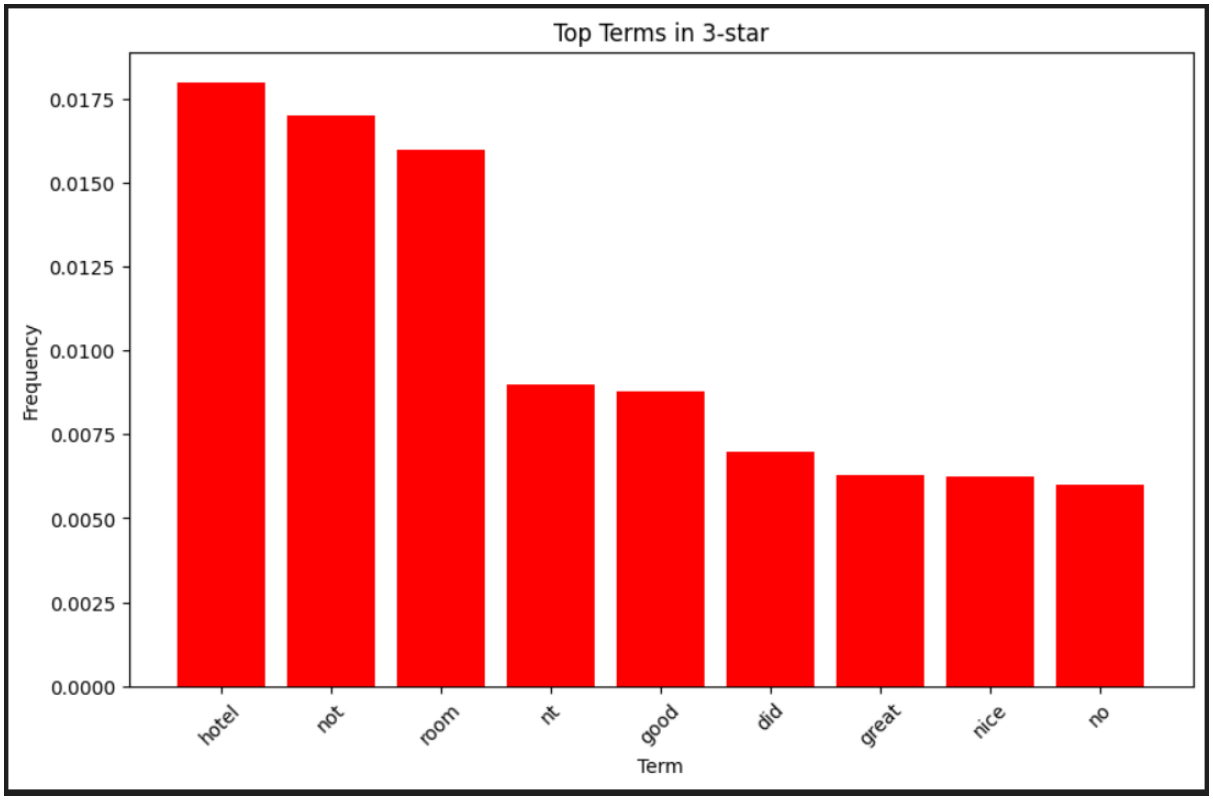
2-Star

```
def plot_bar_chart(category, top_terms):
    plt.figure(figsize=(10, 6))
    plt.bar(range(len(top_terms)), top_terms.values(), align='center',
color='green')
    plt.xticks(range(len(top_terms)), top_terms.keys(), rotation=45)
    plt.ylabel('Frequency')
    plt.xlabel('Term')
    plt.title(f'Top Terms in {category}')
    plt.show()

category = '2-star'
top_terms = { 'not': 0.02, 'room': 0.017, 'hotel': 0.0165, 'nt': 0.009, 'no':
0.0078, 'did': 0.0072, 'good': 0.006, 'stay': 0.0054, 'rooms': 0.0054}

plot_bar_chart(category, top_terms)
```



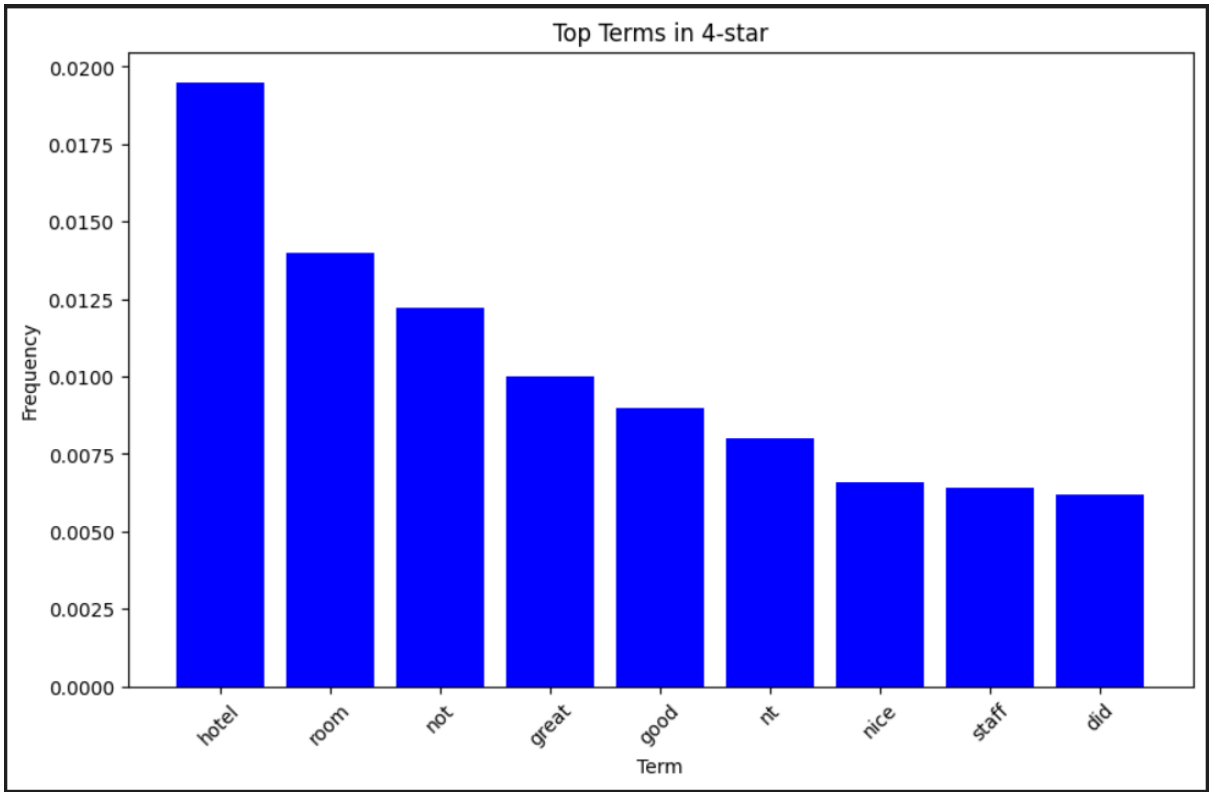
3-Star

```
def plot_bar_chart(category, top_terms):
    plt.figure(figsize=(10, 6))
    plt.bar(range(len(top_terms)), top_terms.values(), align='center',
color='red')
    plt.xticks(range(len(top_terms)), top_terms.keys(), rotation=45)
    plt.ylabel('Frequency')
    plt.xlabel('Term')
    plt.title(f'Top Terms in {category}')
    plt.show()

category = '3-star'
top_terms = { 'hotel': 0.018, 'not': 0.017, 'room': 0.016, 'nt': 0.009,
'good': 0.0088, 'did': 0.007, 'great': 0.00627, 'nice': 0.00624, 'no': 0.006}

plot_bar_chart(category, top_terms)
```



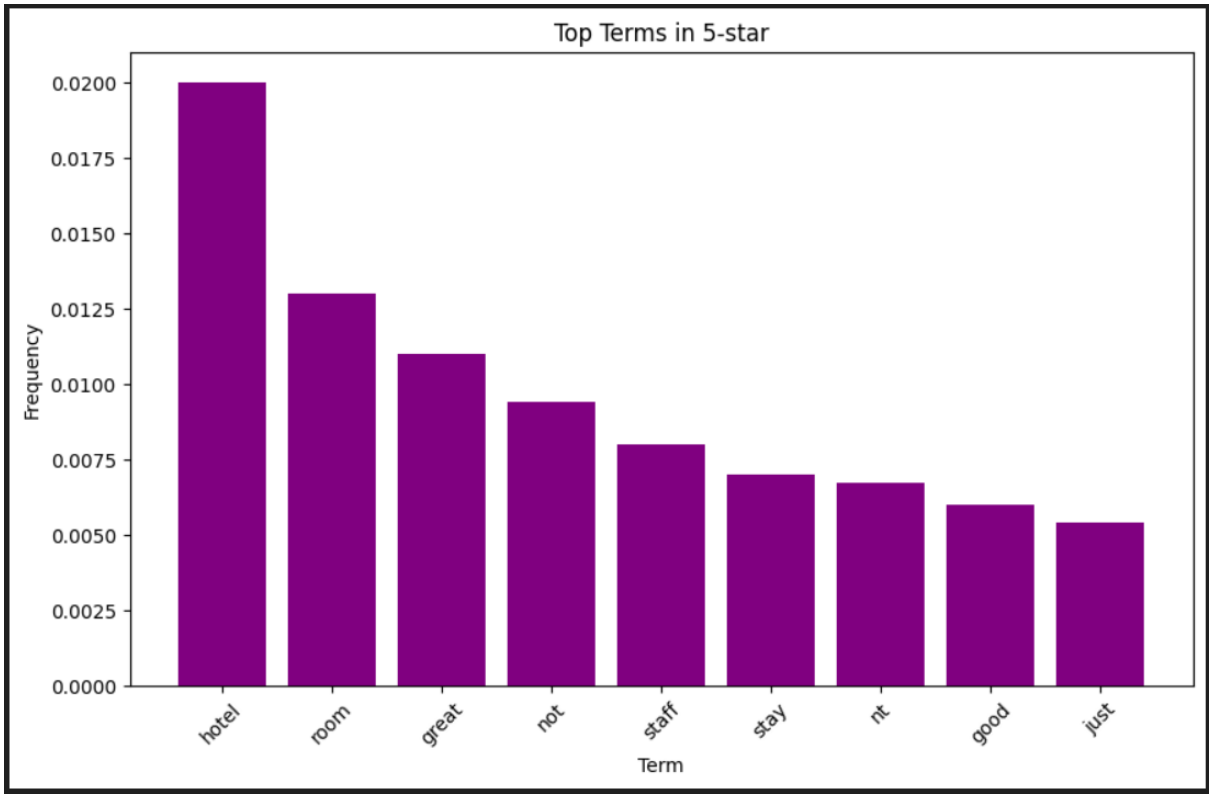
4-Star

```
def plot_bar_chart(category, top_terms):
    plt.figure(figsize=(10, 6))
    plt.bar(range(len(top_terms)), top_terms.values(), align='center',
color='blue')
    plt.xticks(range(len(top_terms)), top_terms.keys(), rotation=45)
    plt.ylabel('Frequency')
    plt.xlabel('Term')
    plt.title(f'Top Terms in {category}')
    plt.show()

category = '4-star'
top_terms = {'hotel': 0.0195, 'room': 0.014, 'not': 0.0122, 'great': 0.01,
'good': 0.009, 'nt': 0.008, 'nice': 0.0066, 'staff': 0.0064, 'did': 0.0062}

plot_bar_chart(category, top_terms)
```



5-Star

```
def plot_bar_chart(category, top_terms):
    plt.figure(figsize=(10, 6))
    plt.bar(range(len(top_terms)), top_terms.values(), align='center',
color='purple')
    plt.xticks(range(len(top_terms)), top_terms.keys(), rotation=45)
    plt.ylabel('Frequency')
    plt.xlabel('Term')
    plt.title(f'Top Terms in {category}')
    plt.show()

category = '5-star'
top_terms = {'hotel': 0.02, 'room': 0.013, 'great': 0.011, 'not': 0.0094,
'staff': 0.008, 'stay': 0.007, 'nt': 0.0067, 'good': 0.006, 'just': 0.0054}

plot_bar_chart(category, top_terms)
```



Word Clouds

Word clouds are visual representations of text data, where the size of each word indicates its frequency or importance within the text. For this assignment we will be generating a word cloud for each star rating. Each word cloud contains the 10 most frequent words/tokens for each rating. We generate the word clouds using the following code:

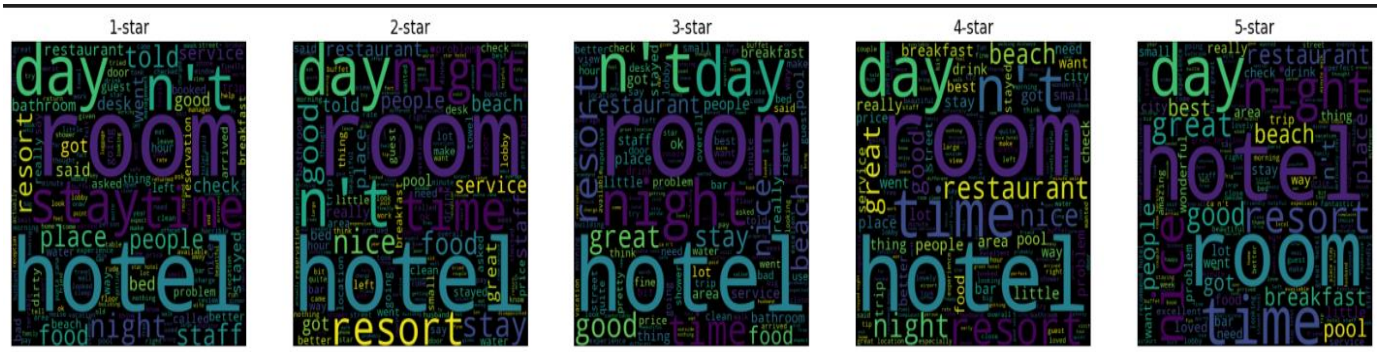
```
tmu.generate_wordclouds([one_star_reviews, two_star_reviews,
three_star_reviews, four_star_reviews, five_star_reviews],
['1-star', '2-star', '3-star', '4-star', '5-star'],
'black')
```

This is calling code for word cloud creation that already exists in our text_mining_utils.py file. The necessary code extract is detailed below.

```
## function to generate the word cloud for a given topic/class
def generate_cloud(text, topic, bg_colour='black', min_font=10):
    cloud = wordcloud.WordCloud(width=700, height=700, random_state=1,
background_color=bg_colour, min_font_size=min_font).generate(text)
    plt.figure(figsize=(7, 7), facecolor=None)
    plt.imshow(cloud)
    ##plt.axis('off')
    plt.tight_layout(pad=0)
    plt.xlabel(topic)
    plt.xticks([])
    plt.yticks([])
```

```
## function to generate multiple word clouds for a set of
topics/classes/categories
def generate_wordclouds(texts, categories, bg_colour, min_font=10):
  fig = plt.figure(figsize=(21, 7))
  for i in range(len(texts)):
    ax = fig.add_subplot(1,5,i+1)
    cloud = wordcloud.WordCloud(width=700, height=700, random_state=1,
                    background_color=bg_colour,
                    min_font_size=min_font).generate(texts[i])
    ax.imshow(cloud)
    ax.axis('off')
    ax.set_title(categories[i])
```

Which gives us:



Now that we have identified all of our most frequent terms we can clean up our data a little. One method of doing this is to identify potential stop words and remove them from further analysis. Stop words appear frequently across all topics/categories but do not provide significant meaning or insights. Some stop words that I have identified from the list of most frequent terms include “n’t”, “just”, “people”, “thing”, etc. We can also clean up our data by handling synonyms and word variations. If multiple words are on the list and have the same meaning we can combine their frequencies as not they represent the same sentiment and thus the same conclusions can be drawn. Similarly we can combine the frequencies for words with slight variations such as plural/singular words like room and rooms. Once we have done this it will give us a more diverse set of data to work with rather than multiple synonyms and variations of the same word(s).

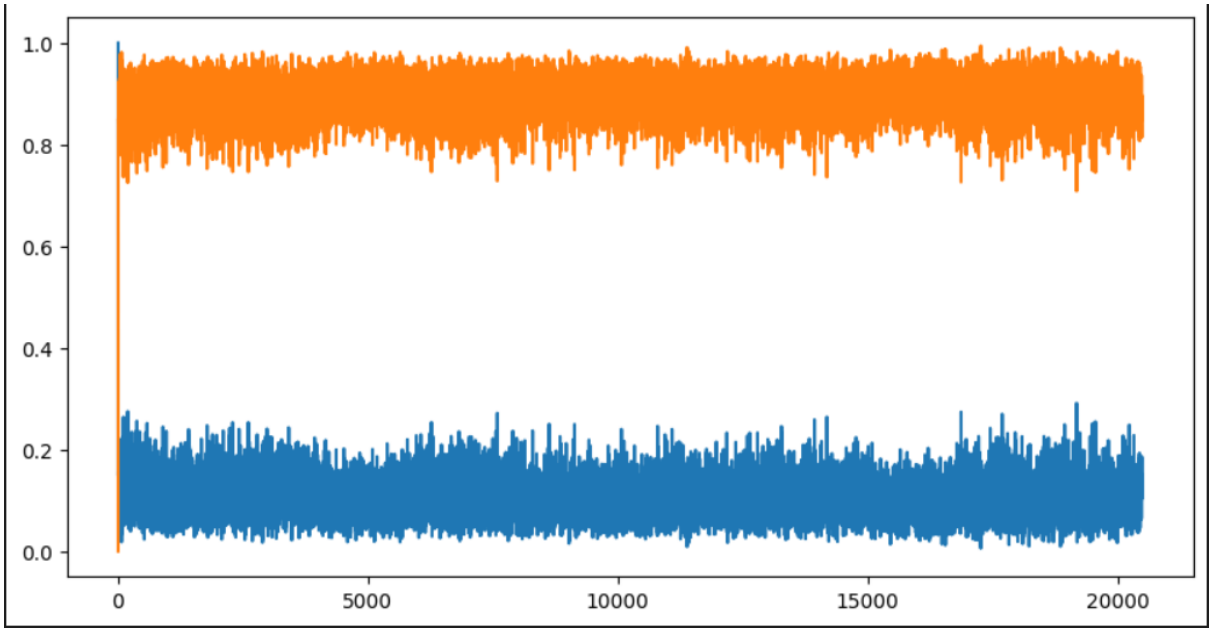
Clustering

Clustering is a machine learning technique used to group similar data points together based on their features. Clustering can be used for anomaly detection, if an object is far away from all other clusters it can be flagged as an outlier or anomaly.[7] In order for us to successfully sort our data into clusters we must first calculate cosine similarity and cosine distance. Cosine similarity is a measure of similarity between two vectors in a multidimensional space, it measures the cosine of the angle between the vectors, indicating how similar they are in direction.[7] Cosine distance, on the other hand, is a measure of dissimilarity or distance between two vectors. It has values ranging from 0 (identical) to 1 (completely different).

```
from sklearn.metrics import adjusted_rand_score, fowlkes_mallows_score
```

```
from sklearn.metrics.pairwise import cosine_distances, cosine_similarity
from sklearn.metrics import calinski_harabasz_score, davies_bouldin_score

sims = cosine_similarity(baseline_tfidf_matrix)
dists = cosine_distances(baseline_tfidf_matrix)

plt.figure(figsize=(10, 5))
plt.plot(sims[0])
plt.plot(dists[0])
```

In the above code we first make the necessary imports. Once this is done we calculate the cosine similarity and cosine distance on the tfidf matrix and assign them values. Once this is done we then plot them on a graph. The outputted graph is shown below:

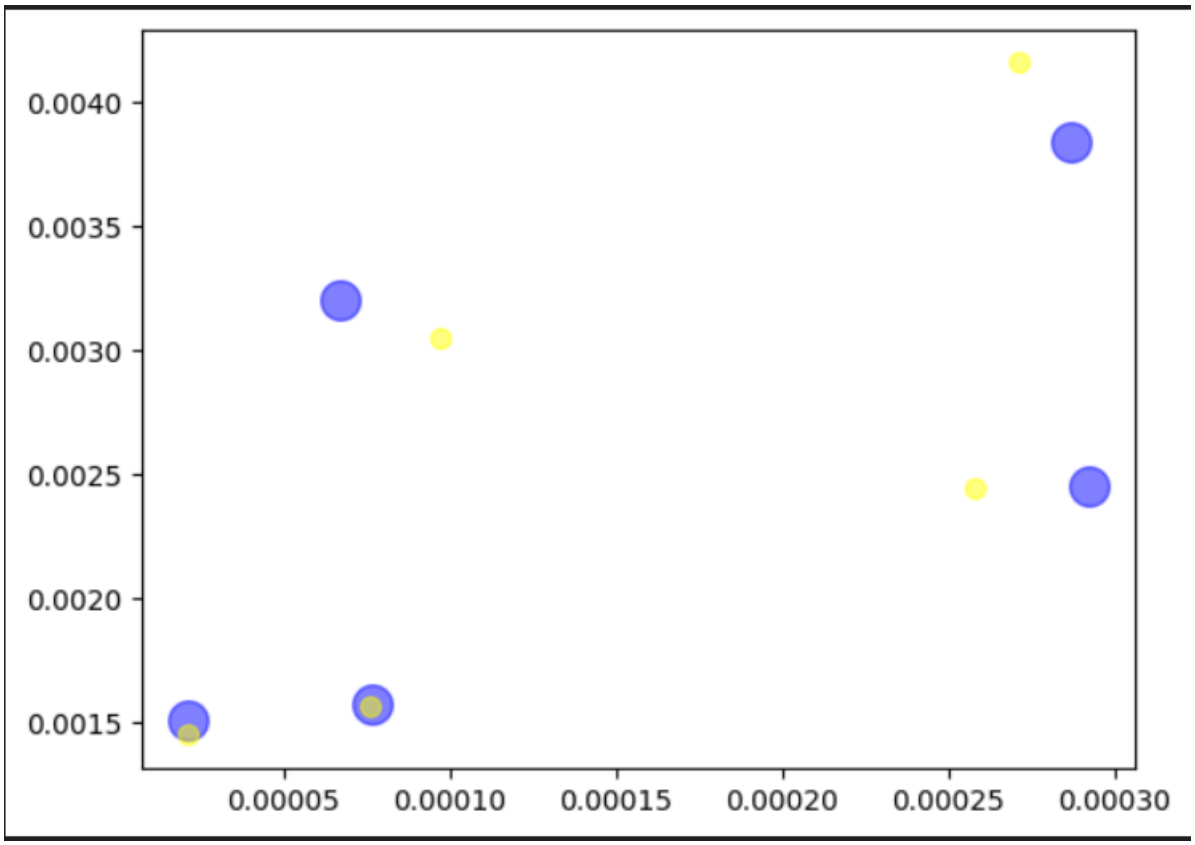


Now that this is done we can begin to perform K-means clustering. We start by making the necessary imports. Following this we then convert our matrix to a sparse matrix, we do this because we are dealing with an extremely large dataset which otherwise would take an extremely long time to load, using a sparse matrix significantly speeds up the process.[9] We can then begin performing K-Means clustering with random initialization. This involves randomly selecting data points from the dataset as initial cluster centers. The clustering algorithm then assigns data points to the nearest cluster center and updates the centers until they meet, resulting in clusters where each data point is assigned to the nearest center.[7] We the results and plot them to our graph. We then repeat the earlier step with K-Mean++ initialization rather than random initialization. We then print the results and plot them to our graph. We have set 'k=5' so we should have 5 clusters. Finally, we show our graph.

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from scipy.sparse import csr_matrix

if not isinstance(baseline_tfidf_matrix, csr_matrix):
```

```
    baseline_tfidf_matrix = csr_matrix(baseline_tfidf_matrix)

k = 3
kmeans_r = KMeans(n_clusters=k, init='random', n_init=1)
cluster_labels_r = kmeans_r.fit_predict(baseline_tfidf_matrix)
cluster_centers_r = kmeans_r.cluster_centers_

print(list(y))
print(cluster_labels_r)
plt.scatter(cluster_centers_r[:, 0], cluster_centers_r[:, 1], c='blue', s=200,
alpha=0.5)

kmeans_p = KMeans(n_clusters=k, init='k-means++', n_init=1)
cluster_labels_p = kmeans_p.fit_predict(baseline_tfidf_matrix)
cluster_centers_p = kmeans_p.cluster_centers_

print(list(y))
print(cluster_labels_p)
plt.scatter(cluster_centers_p[:, 0], cluster_centers_p[:, 1], c='yellow',
s=50, alpha=0.5)

plt.show()
```

This gives us an output of:



In this case the blue clusters represent the randomly initialized clusters and the yellow clusters represent the k-means++ initialized clusters. We have assigned a smaller size to the yellow clusters so you can differentiate between the two different types of clusters in cases where they overlap.

We then calculate and print the ARI and FMI scores for both Kmeans Random and Kmeans++. The ARI score tells us how similar the clusters are to the actual groups in the data. A score of 1 means they match perfectly, while 0 means they're completely random. The FMI score measures how well the clusters agree with the actual groups. A score of 1 means they're perfectly aligned, while 0 means there's no agreement.[8]

```
print("Kmeans Random - ARI score:",
      adjusted_rand_score(cluster_labels_r, list(y)))
print("Kmeans++ - ARI score:",
      adjusted_rand_score(cluster_labels_p, list(y)))
print("Kmeans Random - FMI score:",
      fowlkes_mallows_score(cluster_labels_r, list(y)))
print("Kmeans++ - FMI score:",
      fowlkes_mallows_score(cluster_labels_p, list(y)))
```

Which gives us an output of:

```
Kmeans Random - ARI score: 0.020919834560430596
Kmeans++ - ARI score: 0.02141036524269264
Kmeans Random - FMI score: 0.27254905752947384
Kmeans++ - FMI score: 0.2719074894961275
```

We can repeat these processes to get the hyperparameter scores for the baseline_count_matrix aswell as the baseline_tf_matrix.

```
if not isinstance(baseline_count_matrix, csr_matrix):
    baseline_count_matrix = csr_matrix(baseline_count_matrix)

k = 5
kmeans_a = KMeans(n_clusters=k, init='random', n_init=1)
cluster_labels_a = kmeans_a.fit_predict(baseline_count_matrix)
cluster_centers_a = kmeans_a.cluster_centers_

kmeans_b = KMeans(n_clusters=k, init='k-means++', n_init=1)
cluster_labels_b = kmeans_b.fit_predict(baseline_tfidf_matrix)
cluster_centers_b = kmeans_b.cluster_centers_
print("Kmeans Random - ARI score:",
      adjusted_rand_score(cluster_labels_a, list(y)))
print("Kmeans++ - ARI score:",
      adjusted_rand_score(cluster_labels_b, list(y)))
print("Kmeans Random - FMI score:",
      fowlkes_mallows_score(cluster_labels_a, list(y)))
print("Kmeans++ - FMI score:",
```

```
      fowlkes_mallows_score(cluster_labels_b, list(y)))
```

```
Kmeans Random - ARI score: 0.029558816302968322
Kmeans++ - ARI score: 0.027459457707341803
Kmeans Random - FMI score: 0.3885298963918797
Kmeans++ - FMI score: 0.2769463492941767
```

```
if not isinstance(baseline_count_matrix, csr_matrix):
    baseline_count_matrix = csr_matrix(baseline_count_matrix)

k = 5
kmeans_c = KMeans(n_clusters=k, init='random', n_init=1)
cluster_labels_c = kmeans_c.fit_predict(baseline_count_matrix)
cluster_centers_c = kmeans_c.cluster_centers_

kmeans_d = KMeans(n_clusters=k, init='k-means++', n_init=1)
cluster_labels_d = kmeans_d.fit_predict(baseline_tfidf_matrix)
cluster_centers_d = kmeans_d.cluster_centers_

print("Kmeans Random - ARI score:",
      adjusted_rand_score(cluster_labels_c, list(y)))
print("Kmeans++ - ARI score:",
      adjusted_rand_score(cluster_labels_d, list(y)))
print("Kmeans Random - FMI score:",
      fowlkes_mallows_score(cluster_labels_c, list(y)))
print("Kmeans++ - FMI score:",
      fowlkes_mallows_score(cluster_labels_d, list(y)))
```

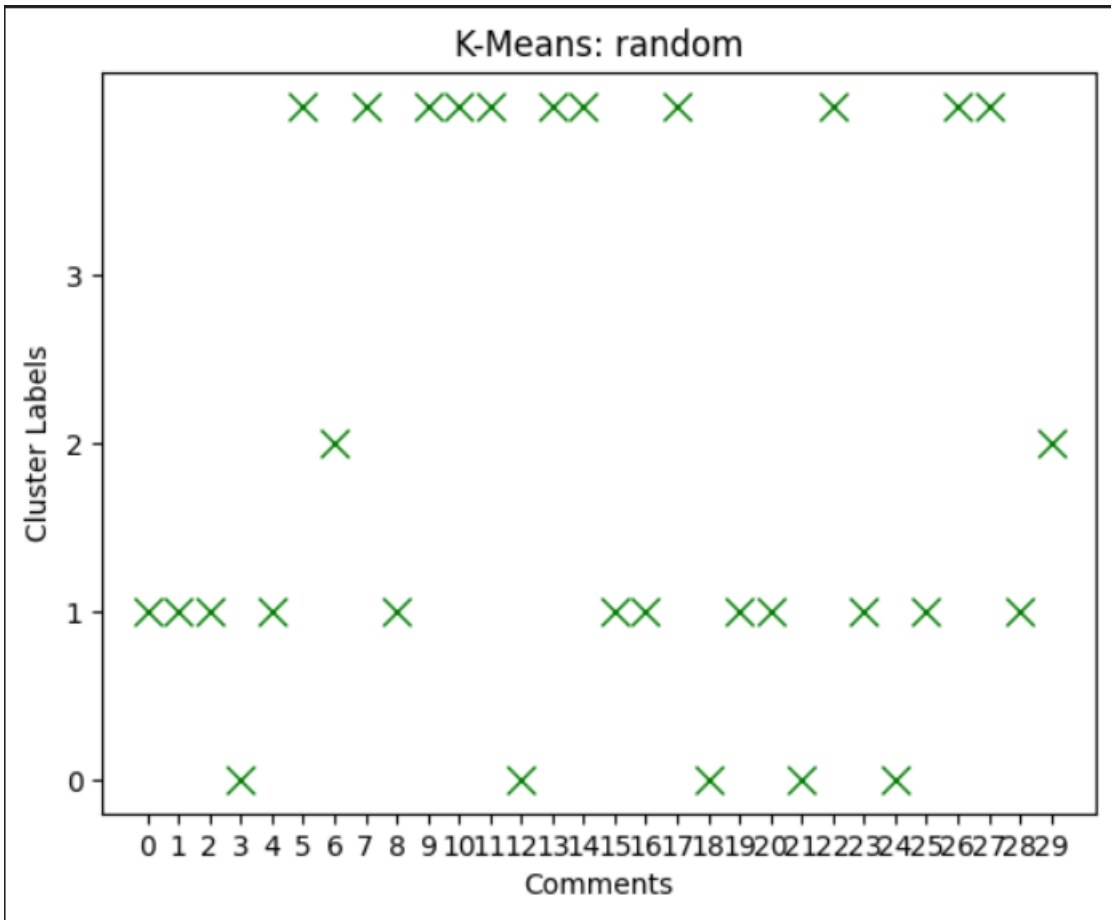
```
Kmeans Random - ARI score: 0.029558816302968322
Kmeans++ - ARI score: 0.022340388461117626
Kmeans Random - FMI score: 0.3885298963918797
Kmeans++ - FMI score: 0.2728017469468138
```

The next thing we do is plot cluster allocation of Kmeans random for the first 30 rows. Since we have been using a sparse matrix we must convert this to dense data. The below code creates a Pandas DataFrame from the first 30 rows of the TF-IDF matrix. It converts the sparse matrix to a dense matrix using “.todense()” and assigns column names.[9] We then plot the clusters on a graph.

Tfidf matrix

```
num_terms = baseline_tfidf_matrix.shape[1]
baseline_tfidf_matrix_df = pd.DataFrame(baseline_tfidf_matrix[:30].todense(),
columns=[f'term_{i+1}' for i in range(num_terms)])

tmu.plot_clusters(baseline_tfidf_matrix_df, cluster_labels_r[:30], "K-Means:
random", 'Comments', 'Cluster Labels')
```

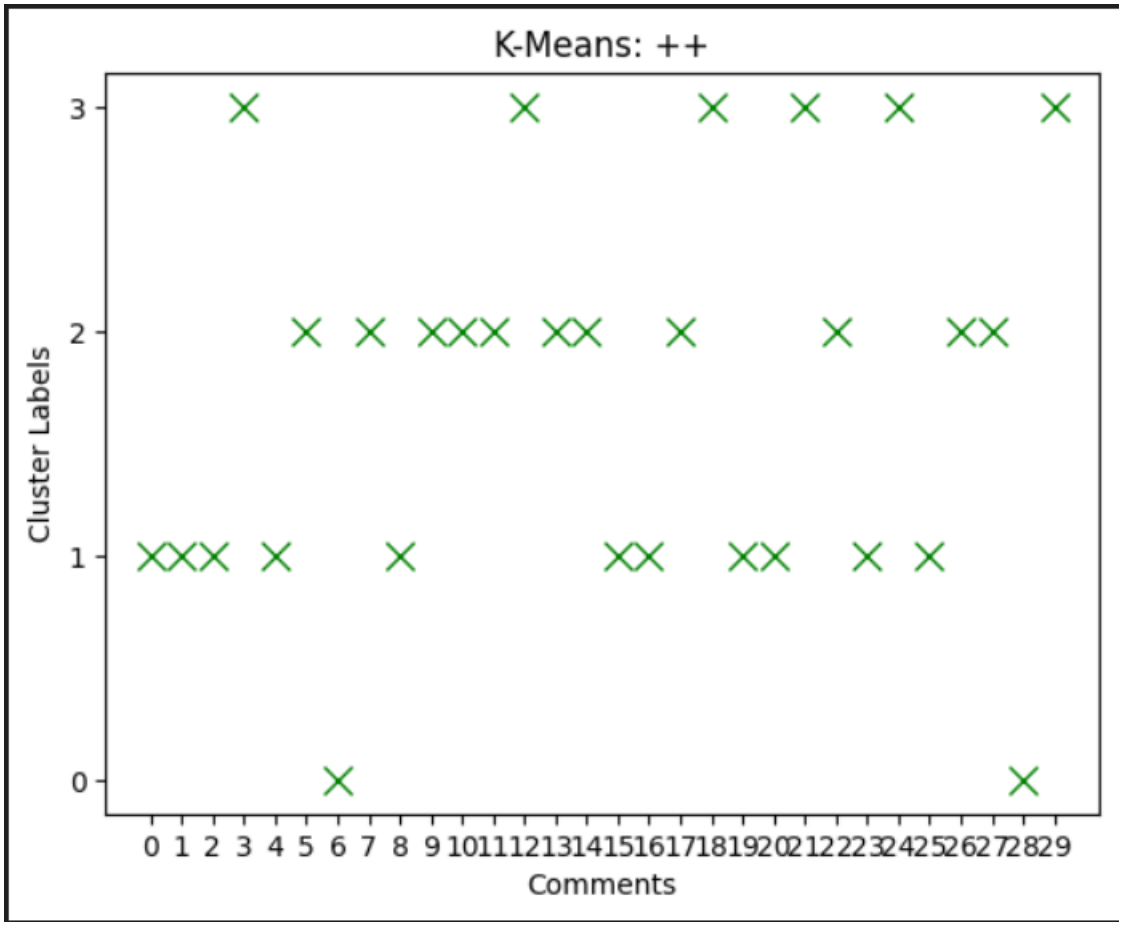
This gives us an output of:



We then repeat the process for Kmeans ++.

```
num_terms = baseline_tfidf_matrix.shape[1]
baseline_tfidf_matrix_df = pd.DataFrame(baseline_tfidf_matrix[:30].todense(),
columns=[f'term_{i+1}' for i in range(num_terms)])

tmu.plot_clusters(baseline_tfidf_matrix_df, cluster_labels_p[:30], "K-Means:
++", 'Comments', 'Cluster Labels')
```

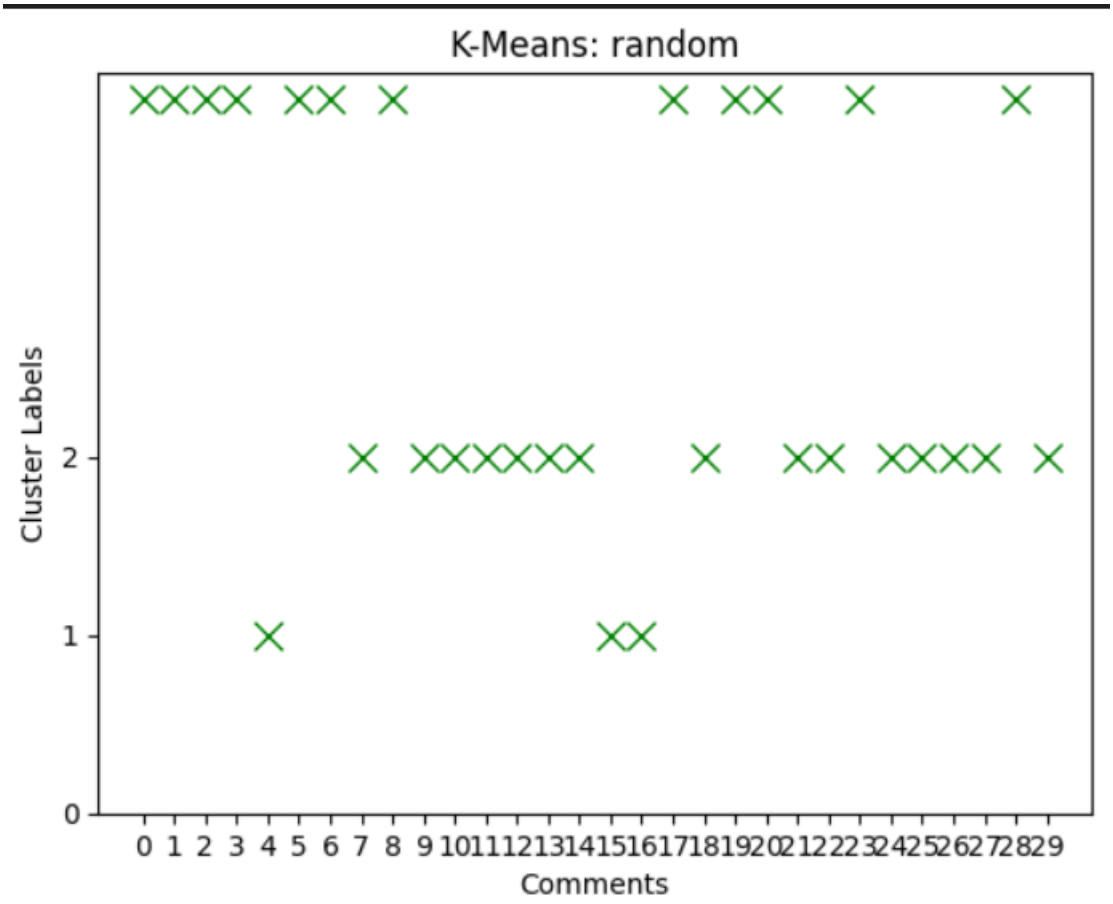
Which gives us an output of:



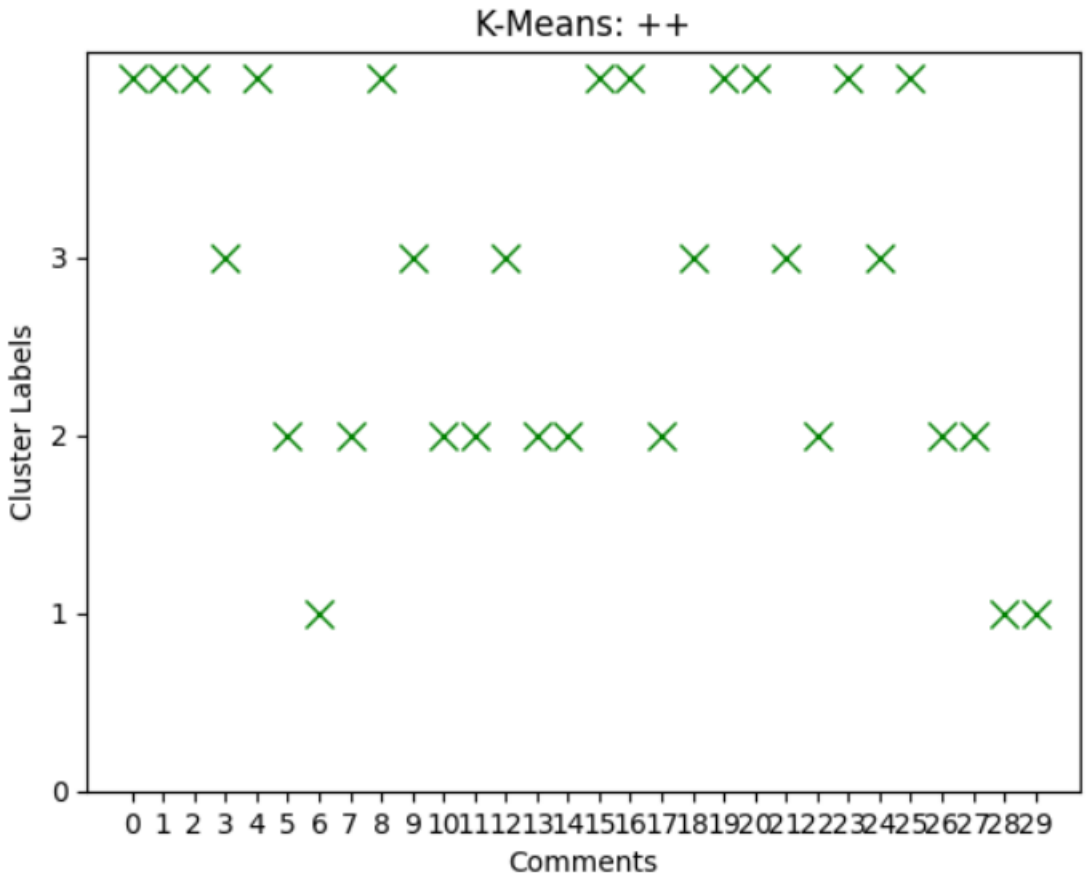
We can do this for each matrix and then cross examine the results.

Count matrix

```
num_terms = baseline_count_matrix.shape[1]
baseline_count_matrix_df = pd.DataFrame(baseline_count_matrix[:30].todense(),
columns=[f'term_{i+1}' for i in range(num_terms)])

tmu.plot_clusters(baseline_count_matrix_df, cluster_labels_a[:30], "K-Means:
random", 'Comments', 'Cluster Labels')
```

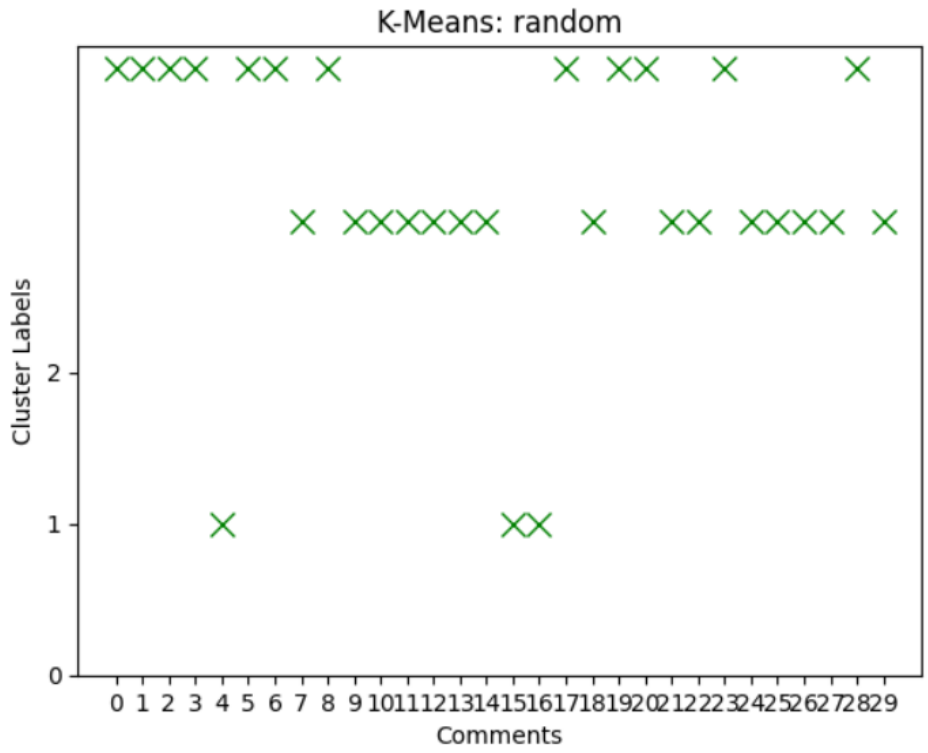



```
num_terms = baseline_count_matrix.shape[1]
baseline_count_matrix_df = pd.DataFrame(baseline_count_matrix[:30].todense(),
columns=[f'term_{i+1}' for i in range(num_terms)])

tmu.plot_clusters(baseline_count_matrix_df, cluster_labels_b[:30], "K-Means:
++", 'Comments', 'Cluster Labels')
```

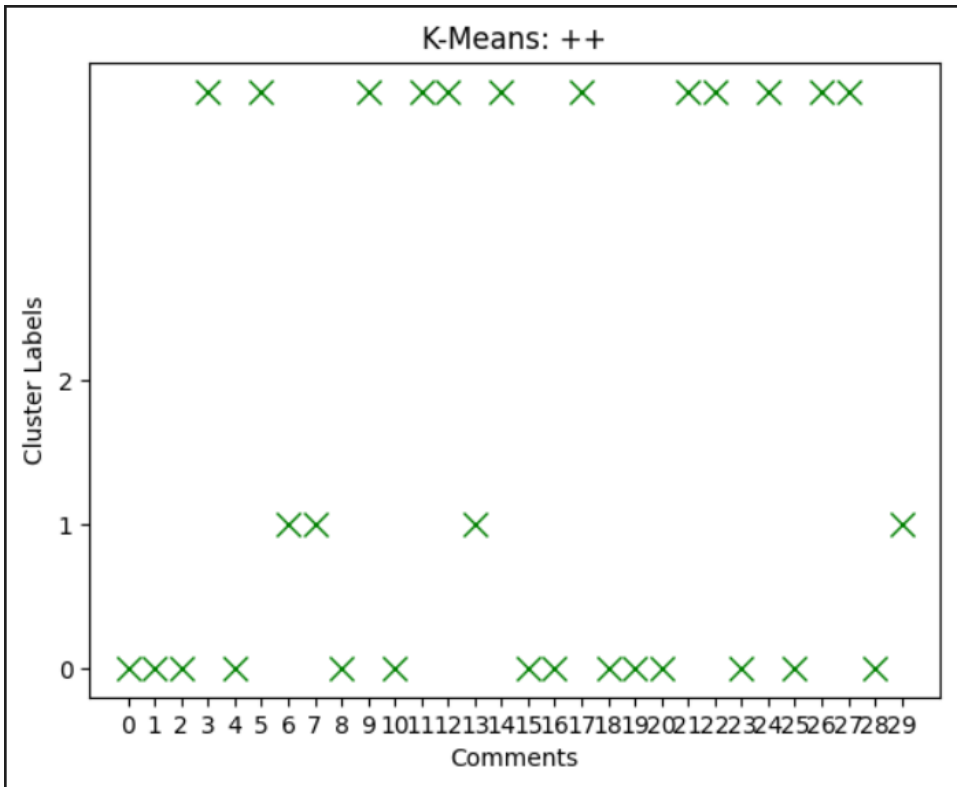


Tf matrix

```
num_terms = baseline_tf_matrix.shape[1]
baseline_tf_matrix_df = pd.DataFrame(baseline_tf_matrix[:30],
columns=[f'term_{i+1}' for i in range(num_terms)])

tmu.plot_clusters(baseline_tf_matrix_df, cluster_labels_c[:30], "K-Means:
random", 'Comments', 'Cluster Labels')
```




```
num_terms = baseline_tf_matrix.shape[1]
baseline_tf_matrix_df = pd.DataFrame(baseline_tf_matrix[:30],
columns=[f'term_{i+1}' for i in range(num_terms)])

tmu.plot_clusters(baseline_tf_matrix_df, cluster_labels_d[:30], "K-Means: ++",
'Comments', 'Cluster Labels')
```



Main Data Preparation

Data preparation, also known as data preprocessing, is the process of transforming raw data into a format that is suitable for analysis or machine learning tasks. It involves cleaning, organizing, and structuring data to make it usable for the intended purpose.[1]

Text Preprocessing

For our dataset the text preprocessing tasks that I believe best suits our dataset includes the following:

- Dealing with synonyms, concepts/hypernyms, and word variations
- Stop Words Removal
- Punctuation Removal

These text preprocessing tasks where chosen for this dataset as they are best suited for sorting the data. Other text preprocessing tasks such as case transformation or elimination short tokens where deemed not suitable for a review based dataset because they would either not have contributed anything of importance to this particular type of dataset or possibly even hindered it's use.

Before applying these text preprocessing tasks we must first clean our data using the following code to deal with mistakes in the dataset such as multiple consecutive white spaces or use of parenthetical notes.

```
clean_operations = {
r'(\(.+?\))+' : '',
r'(\[.+?\])+' : '',
r'\s+' : ' ',
r'\s{2,}' : ' ',
}
clean_data = data.copy()
clean_data.Review = clean_data.Review.apply(tmu.clean_doc,
clean_operations=clean_operations)
clean_data.head()

clean_count_matrix = tmu.build_count_matrix(clean_data.Review)
clean_tf_matrix = tmu.build_tf_matrix(clean_data.Review)
clean_tfidf_matrix = tmu.build_tfidf_matrix(clean_data.Review)

tmu.printClassifReport(clf, clean_count_matrix, y)
tmu.printClassifReport(clf, clean_tf_matrix, y)
tmu.printClassifReport(clf, clean_tfidf_matrix, y)
```

We then print out the clean matrices to determine whether or not the cleaning process has had a negative impact on our data.

	precision	recall	f1-score	support
1	0.74	0.37	0.50	1421
2	0.41	0.34	0.37	1793
3	0.29	0.07	0.11	2184
4	0.43	0.50	0.46	6039
5	0.68	0.80	0.74	9054
accuracy			0.57	20491
macro avg	0.51	0.42	0.44	20491
weighted avg	0.55	0.57	0.54	20491

	precision	recall	f1-score	support
1	0.00	0.00	0.00	1421
2	0.00	0.00	0.00	1793
3	0.00	0.00	0.00	2184
4	0.00	0.00	0.00	6039
5	0.44	1.00	0.61	9054
accuracy			0.44	20491
macro avg	0.09	0.20	0.12	20491
weighted avg	0.20	0.44	0.27	20491

	precision	recall	f1-score	support
1	1.00	0.00	0.00	1421
2	0.00	0.00	0.00	1793
3	0.00	0.00	0.00	2184
4	0.12	0.01	0.01	6039
5	0.45	1.00	0.62	9054
accuracy			0.44	20491
macro avg	0.31	0.20	0.13	20491
weighted avg	0.30	0.44	0.28	20491

As can be seen from the above classification report the cleaning process had no impact on our data so it is safe to use these cleaned matrices without corrupting our findings.

Dealing with synonyms, concepts/hypernyms, and word variations

Dealing with synonyms, concepts/hypernyms, and word variations is a text preprocessing task which allows us to take multiple different words with similar meanings and converting them to one word.[6] An example of this would be converting the words “lovely”, “great”, “good” or “nice” to just “good”. This is useful as it means we have just one word to deal with that covers all interpretations which makes processing our data easier without changing the meaning of the sentences.[6] This is very useful for our dataset as we have many different words in our dataset that mean the same thing, this would be a good way of simplifying the dataset for further use.

In the below code we are applying this technique to our dataset. For this I have chosen to deal with multiple variations of the words ‘room’, ‘hotel’ and ‘good’ as these are the most common synonyms and word variations that appear in our dataset.

```
repl_dict = {
'room': [r'\broom(s)\b', r'\bhotelroom\b' r'\bbedroom?\b'],
'hotel': [r'\bhotel\b', r'\bresort\b'],
'good': [r'\bgood?\b', r'\bgreat?\b', r'\bnice?\b']
}

docs = clean_data.Review
docs_repl = docs.apply(tmu.improve_bow, repl_dict=repl_dict)
```

```
new_count_matrix = tmu.build_count_matrix(docs_repl)
new_tf_matrix = tmu.build_tf_matrix(docs_repl)
new_tfidf_matrix = tmu.build_tfidf_matrix(docs_repl)
```

Once we have replaced all the necessary words we then build new matrices using our new data. We can then print new classification reports to see if this text preprocessing task has had any major impact on our data.

```
tmu.printClassifReport(clf, new_count_matrix, y)
tmu.printClassifReport(clf, new_tf_matrix, y)
tmu.printClassifReport(clf, new_tfidf_matrix, y)
```

✓ 27m 19.2s

```
              precision    recall  f1-score   support

           1       0.75      0.37      0.50      1421
           2       0.41      0.34      0.37      1793
           3       0.29      0.07      0.12      2184
           4       0.42      0.50      0.46      6039
           5       0.68      0.80      0.73      9054

    accuracy                           0.56     20491
   macro avg       0.51      0.42      0.44     20491
weighted avg       0.54      0.56      0.54     20491

              precision    recall  f1-score   support

           1       0.00      0.00      0.00      1421
           2       0.00      0.00      0.00      1793
           3       0.00      0.00      0.00      2184
           4       0.00      0.00      0.00      6039
           5       0.44      1.00      0.61      9054

    accuracy                           0.44     20491
   macro avg       0.09      0.20      0.12     20491
weighted avg       0.20      0.44      0.27     20491

              precision    recall  f1-score   support
...
    accuracy                           0.44     20491
   macro avg       0.31      0.20      0.13     20491
weighted avg       0.30      0.44      0.28     20491
```

Now lets go through the matrices and highlight the changes. Green highlight indicates the score has improved compared to the last classification report, red indicates the score has gotten worse compared to the last classification report, if the score isn't highlighted then it hasn't changed compared to the last classification report.

Count Matrix

	Precision	Recall	F1-score	support
1	0.75	0.37	0.50	1421
2	0.41	0.34	0.37	1793
3	0.29	0.07	0.12	2184
4	0.43	0.50	0.46	6039
5	0.68	0.80	0.74	9054
Accuracy			0.57	20491
Macro avg	0.51	0.42	0.44	20491

Weighted avg	0.55	0.57	0.54	20491

TF matrix

	Precision	Recall	F1-score	support
1	0.00	0.00	0.00	1421
2	0.00	0.00	0.00	1793
3	0.00	0.00	0.00	2184
4	0.00	0.00	0.00	6039
5	0.44	1.00	0.61	9054
Accuracy			0.44	20491
Macro avg	0.09	0.20	0.12	20491
Weighted avg	0.20	0.44	0.27	20491

TFIDF Matrix

	Precision	Recall	F1-score	support
Accuracy			0.44	20491
Macro avg	0.31	0.20	0.13	20491
Weighted avg	0.30	0.44	0.28	20491

From looking at the classification reports I can see that there has been a slight improvement as some of our parameter show a slight increase of 0.01. Most of our parameters remain unchanged.

Stop words

Stop words are common words that occur frequently in a language but typically do not carry significant meaning or contribute much to the understanding of the text. We can remove stop words to get our sentences to its simplest possible form to make the data easier to process without changing it's meaning. This is a good text preprocessing task for our dataset as the reviews in our dataset contain a lot of stop words that ultimately aren't contributing to the data.[6]

This first thing we must do to remove stop words is to download the English universal stopword library. We can then print that out to get an idea of what stop words we will be removing.

```
nltk.download('stopwords')
univ_sw = nltk.corpus.stopwords.words('english')
print(univ_sw)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'sh
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\paulj\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Once the universal stop words library has been downloaded and we are satisfied with the stop words that will be removed we can then begin removing the stop words from our dataset using the following code:

```
nltk.download('punkt')
```

```
no_univsw_docs = docs.apply(tmu.remove_sw_punct, to_remove=univ_sw)

new_count_matrix = tmu.build_count_matrix(no_univsw_docs)
new_tf_matrix = tmu.build_tf_matrix(no_univsw_docs)
new_tfidf_matrix = tmu.build_tfidf_matrix(no_univsw_docs)
```

If we wanted to we could remove custom stop words using the below code but I chose not to as no other words come to mind that could be safely removed without changing the understanding of the reviews.

custom_sw = ['decrease', 'increase'] no_customsw_docs = no_univsw_docs.apply(tmu.remove_sw_punct, to_remove=custom_sw)

We can then check if this had any impact on our dataset by generating more classification reports.

```
              precision    recall  f1-score   support

           1       0.74      0.37      0.50      1421
           2       0.41      0.32      0.36      1793
           3       0.27      0.06      0.10      2184
           4       0.42      0.50      0.45      6039
           5       0.68      0.80      0.73      9054

    accuracy                           0.56     20491
   macro avg       0.50      0.41      0.43     20491
weighted avg       0.54      0.56      0.53     20491

              precision    recall  f1-score   support

           1       0.00      0.00      0.00      1421
           2       0.00      0.00      0.00      1793
           3       0.00      0.00      0.00      2184
           4       0.00      0.00      0.00      6039
           5       0.44      1.00      0.61      9054

    accuracy                           0.44     20491
   macro avg       0.09      0.20      0.12     20491
weighted avg       0.20      0.44      0.27     20491

              precision    recall  f1-score   support
...
    accuracy                           0.44     20491
   macro avg       0.31      0.20      0.13     20491
weighted avg       0.30      0.44      0.28     20491
```

Count Matrix

	Precision	Recall	F1-score	support
1	0.74	0.37	0.50	1421
2	0.41	0.32	0.36	1793
3	0.29	0.06	0.10	2184
4	0.42	0.50	0.45	6039
5	0.68	0.80	0.73	9054
Accuracy			0.56	20491
Macro avg	0.50	0.41	0.43	20491
Weighted avg	0.54	0.56	0.53	20491

TF matrix

	Precision	Recall	F1-score	support
1	0.00	0.00	0.00	1421
2	0.00	0.00	0.00	1793
3	0.00	0.00	0.00	2184
4	0.00	0.00	0.00	6039
5	0.44	1.00	0.61	9054
Accuracy			0.44	20491
Macro avg	0.09	0.20	0.12	20491
Weighted avg	0.20	0.44	0.27	20491

TFIDF Matrix

	Precision	Recall	F1-score	support
Accuracy			0.44	20491
Macro avg	0.31	0.20	0.13	20491
Weighted avg	0.30	0.44	0.28	20491

We can see from the classification report that the removal of stop words has had a negative impact on our scores. Numerous scores from our count matrix has dropped my 0.01. Given how the scores have changed so little though, it is safe to continue to the next step.

Punctuation Removal

Punctuation removal is a text preprocessing step where any punctuation marks present in a piece of text are stripped or removed.[6] Punctuation marks include characters like periods (.), commas (,) and various other symbols used in writing. The process of punctuation removal typically involves iterating through each character in the text and checking if it is a punctuation mark. If it is, the punctuation mark is removed or replaced with whitespace.[6] Punctuation removal is often performed as a preprocessing step in natural language processing (NLP) tasks such as text classification, sentiment analysis, and language modeling. Removing punctuation helps simplify the text and can improve the performance of downstream tasks by reducing the vocabulary size and noise in the data.[6]

```
import string
punct = list(string.punctuation)
no_punct_docs = no_univsw_docs.apply(tmu.remove_sw_punct, to_remove=punct)
new_count_matrix = tmu.build_count_matrix(no_punct_docs)
new_tf_matrix = tmu.build_tf_matrix(no_punct_docs)
new_tfidf_matrix = tmu.build_tfidf_matrix(no_punct_docs)
print(new_count_matrix.head(5))
```

In the above code we are generating a list of common punctuation marks and then removing them from the data. We are then generating new matrices on the new text. We are then outputting the head of the new count matrix so we can see what punctuation remains.

Before:

```
   '  *  +   ,  -  .  /  0  00  000  ...  é_Çål  éenny  êtyle  î_  \
0  0  0  0  11  1  0  0  0   0    0  ...      0      0      0   0

   î__asically  î__ere  î__here  î__hese  î_ölthough  öreat
0            0       0        0        0           0      0

[1 rows x 53185 columns]
```

After:

```
   '  +  ,  -  .  /  0  00  000  0001  ...  é_Çål  éenny  êtyle  î_  \
0  0  0  0  1  0  0  0   0    0     0  ...      0      0      0   0

   î__asically  î__ere  î__here  î__hese  î_ölthough  öreat
0            0       0        0        0           0      0

[1 rows x 53183 columns]
```

If we compare this the matrix head from earlier we can see that 11 commas have been removed from the first row showing that punctuation marks have been successfully removed.

We then must generate new classification reports to determine whether our scores have improved.

```
tmu.printClassifReport(clf, new_count_matrix, y)
tmu.printClassifReport(clf, new_tf_matrix, y)
tmu.printClassifReport(clf, new_tfidf_matrix, y)
```

```
              precision    recall  f1-score   support

           1       0.74      0.39      0.51      1421
           2       0.41      0.32      0.36      1793
           3       0.27      0.07      0.11      2184
           4       0.42      0.50      0.46      6039
           5       0.68      0.80      0.73      9054

    accuracy                           0.56     20491
   macro avg       0.50      0.41      0.43     20491
weighted avg       0.54      0.56      0.54     20491

              precision    recall  f1-score   support

           1       0.00      0.00      0.00      1421
           2       0.00      0.00      0.00      1793
           3       0.00      0.00      0.00      2184
           4       0.00      0.00      0.00      6039
           5       0.44      1.00      0.61      9054

    accuracy                           0.44     20491
   macro avg       0.09      0.20      0.12     20491
weighted avg       0.20      0.44      0.27     20491

              precision    recall  f1-score   support
...
    accuracy                           0.44     20491
   macro avg       0.31      0.20      0.13     20491
weighted avg       0.30      0.44      0.28     20491
```

Count Matrix

	Precision	Recall	F1-score	support
1	0.74	0.39	0.51	1421
2	0.41	0.32	0.36	1793
3	0.27	0.07	0.11	2184
4	0.42	0.50	0.46	6039
5	0.68	0.80	0.73	9054
Accuracy			0.56	20491
Macro avg	0.50	0.41	0.43	20491
Weighted avg	0.54	0.56	0.54	20491

TF matrix

	Precision	Recall	F1-score	support
1	0.00	0.00	0.00	1421
2	0.00	0.00	0.00	1793
3	0.00	0.00	0.00	2184
4	0.00	0.00	0.00	6039
5	0.44	1.00	0.61	9054
Accuracy			0.44	20491
Macro avg	0.09	0.20	0.12	20491
Weighted avg	0.20	0.44	0.27	20491

TFIDF Matrix

	Precision	Recall	F1-score	support
Accuracy			0.44	20491
Macro avg	0.31	0.20	0.13	20491
Weighted avg	0.30	0.44	0.28	20491

We can see from the classification report that there has been limited change, some improvement compared the last classification report is noted and one parameter has gotten worse but the vast majority of our scores remain unchanged.

After applying these text preprocessing techniques the only one I would include in the final pipeline is dealing with synonyms, concepts/hypernyms, and word variations. This is the only one I would choose as it is the only text preprocessing task to show improved scores in the classification report. Stop words had the opposite affect and produced slightly worse scores whilst punctuation removal produced minimal change and although the changes where an improvement compared to the stop word classification report it was ultimately worse that the classification report output before text preprocessing.

Algorithms-based Feature selection/reduction tasks

In text analysis, feature selection involves choosing the most relevant words or phrases from text documents to represent the data for analysis or model training. This process helps to identify terms that contribute most to the task at hand, such as sentiment analysis or document classification. Feature selection is a preprocessing step that has several benefits including

reducing overfitting, improving accuracy and reducing training time. Reducing the number of features is very important for reducing noise and increasing performance.[10] Feature selection can be very useful in for our dataset as it can help us to achieve out business and data mining goals, features selection can be used to determine common features that appear in negative and positive reviews so hotel management know their strengths and weakness.

The three types of feature selection we will be choosing from includes the following:

1. **Statistical Univariate Feature Selection**: This approach evaluates each word or phrase individually based on statistical measures such as term frequency or inverse document frequency (TF-IDF). It selects features (words or phrases) that exhibit strong statistical associations with the target variable, such as sentiment or document category. For example, it might identify words that frequently appear in positive or negative sentiment documents. However, it doesn't consider how these words interact with each other within the context of the text.[10]
2. **Classification Algorithm-based Univariate Feature Selection**: This method assesses the importance of each feature within the context of a specific classification algorithm, such as Naive Bayes or Support Vector Machines. It ranks features based on their contribution to the classifier's performance. It might prioritize words that significantly affect the accuracy of a sentiment analysis classifier. However, it might overlook interactions between words that could enhance predictive performance.[10]
3. **Multivariate Feature Selection**: Multivariate feature selection methods consider interactions and dependencies between features (words or phrases) within the text. For example, instead of selecting individual words, it might select pairs or groups of words that frequently occur together and contribute to the classification task. While more computationally intensive, multivariate methods offer a more comprehensive understanding of the relationships between features in the text data.[10]

In summary, each feature selection method in the context of text analysis offers different insights and trade-offs. For the purpose of this assignment, we are told to try 2 of these techniques. The first technique chosen is Statistical Univariate Feature Selection, I chose this as it specialises in identifying words that frequently appear in positive or negative sentiments which is perfect for hotel reviews. The second technique chosen is Classification Algorithm-based Univariate Feature Selection as it is known to work extremely well with Naïve Bayes which is the classification algorithm we are using.

Statistical Univariate Feature Selection

For statistical univariate feature selection we can use anova or chi-square to retrieve the k best features. Chi-square and analysis of variance (ANOVA) are both statistical tests used in feature selection tasks.

Chi-square: Chi-square is a method used to determine the association between two categorical variables, making it ideal for scenarios like text analysis where features are categorical (e.g., words or terms) and the target variable is also categorical (e.g., sentiment labels). By finding the chi-square statistic, it determines the degree of independence between each feature and the target variable, with higher scores indicating stronger associations and higher relevance for feature selection.[11]

ANOVA: ANOVA is a test primarily used to assess differences in means across multiple groups, making it suitable for continuous variables. In feature selection, ANOVA is often used when features represent continuous or numerical data. [11]

For this assignment we will be checking both and comparing the results side by side.

```
from sklearn.feature_selection import chi2, f_classif

count_chi2_matrix = tmu.stat_univariate_fs(new_count_matrix, y,
weight_method=chi2,
selection_method='k_best',
num_features=25, scores_to_print=25
)
```

```
Top 25 features:
       Attribute       Weight
32744        not  5760.178918
47955       told  2948.489774
21730       good  2942.854803
32582         no  2560.953122
52467      worst  2137.003077
15274      dirty  2061.762977
3              ,  1803.085345
40505       room  1789.259223
46994   terrible  1541.310907
33309         ok  1526.199914
23920   horrible  1291.586646
40793       rude  1250.918523
29466    manager  1236.367300
15046        did  1128.587790
5844         bad  1075.902233
18171  excellent  1064.868160
52313  wonderful  1047.096524
31854        n't  1036.826083
41025       said   965.614468
19434    finally   924.646312
14771       desk   904.101269
36406       poor   882.878403
5              .   879.817588
9225      called   853.403120
18951  fantastic   838.684609
```

This output shows the top 25 features selected using the chi-square statistic as part of the univariate feature selection process. Each feature is associated with a weight, which represents its chi-square score. Features with higher weights are considered more informative and have a stronger association with the target variable which in this case is our ratings.

```
count_anova_matrix = tmu.stat_univariate_fs(new_count_matrix, y,
weight_method=f_classif,
selection_method='k_best',
num_features=25, scores_to_print=25
```

```
Top 25 features:
       Attribute      Weight
32744        not  468.290095
52467      worst  459.015148
15274      dirty  410.118935
47955       told  324.599204
32582         no  314.707160
46994   terrible  298.939507
21730       good  297.425557
23920   horrible  260.932009
40793       rude  246.945827
33309         ok  245.873365
40505       room  197.358601
18171  excellent  188.386621
5844         bad  186.886954
52313  wonderful  179.141648
29466    manager  176.822264
19434    finally  159.631351
36406       poor  156.603889
23261    helpful  155.764139
18951  fantastic  151.234141
8496       broken  146.358859
28546   location  145.625645
35295    perfect  144.194814
14771       desk  139.721658
28836      loved  138.091675
19417     filthy  138.016509
```

This output displays the top 25 features selected using the Analysis of Variance (ANOVA) statistic as part of the univariate feature selection process. Each feature is associated with a weight, which represents its ANOVA F-value. Features with higher weights are considered more relevant or informative for distinguishing between different categories or groups within the target variable (in this case, the star ratings).

```
tmu.printClassifReport(clf, count_chi2_matrix, y)
tmu.printClassifReport(clf, count_anova_matrix, y)
```

	precision	recall	f1-score	support
1	0.53	0.54	0.53	1421
2	0.33	0.27	0.30	1793
3	0.33	0.15	0.21	2184
4	0.43	0.31	0.36	6039
5	0.60	0.81	0.69	9054
accuracy			0.53	20491
macro avg	0.44	0.42	0.42	20491
weighted avg	0.49	0.53	0.50	20491

	precision	recall	f1-score	support
1	0.55	0.59	0.57	1421
2	0.34	0.30	0.32	1793
3	0.34	0.15	0.21	2184
4	0.42	0.33	0.37	6039
5	0.62	0.79	0.69	9054
accuracy			0.53	20491
macro avg	0.45	0.43	0.43	20491
weighted avg	0.50	0.53	0.51	20491

This output shows the classification reports for both chi2 and anova. We can see that for both classification reports the scores are quite good for ratings that are 1 or 5 but the scores for the others are much poorer. This makes sense as it is much easier to predict a review that is extremely negative as 1 or a review that is very positive as a 5 but it can be quite difficult to determine reviews in-between. A potential fix to this is to lump 1 and 2 star reviews together as 'negative' reviews and 4 and 5 star reviews together as 'positive' reviews. The algorithm will find it much easier to accomplish feature selection when it has to be less specific about how positive or negative a review is. Nonetheless, we can see from the above output that anova has produced slightly better results than chi2.

Classification Algorithm based Univariate Feature Selection

When performing classification algorithm based univariate feature selection you can use either Linear SVM or Decision Tree. The choice between Linear SVM and Decision Tree depends on various factors such as the nature of the data, the complexity of the problem, and the interpretability of the model.

Linear Support Vector Machine (SVM) is a machine learning algorithm used for classification tasks. It works by finding the best way to separate different classes of data points. The goal of linear SVM is to maximize the margin between the classes. By doing this, linear SVM aims to create a boundary that best separates the classes while minimizing the chance of misclassification.[10]

A Decision Tree is a tree-like structure used in machine learning for both classification and regression tasks. It recursively splits the data into smaller subsets based on the features that best separate the target variable. The goal is to create branches that result in the purest subsets possible, where each subset ideally contains only one class of the target variable.[10] Decision

Trees are known for their interpretability and ease of understanding, as the resulting tree can be visualized and interpreted like a flowchart, showing the decision-making process. They are capable of capturing complex relationships between features and the target variable, making them suitable for a wide range of tasks. However, decision trees can be prone to overfitting, especially when the tree is deep or when the data is noisy. Regularization techniques and ensemble methods like Random Forests are often used to mitigate this issue.

For this assignment we will be checking both and comparing the results side by side.

```
linear_svm = LinearSVC(random_state=1)
svm_weighted_matrix = tmu.clf_univariate_fs(new_count_matrix, y, linear_svm,
num_features=25, scores_to_print=25)
```

This gives us an output of:

```
Top 25 features:
           Attribute    Weight
19931          foget  1.726184
21148           gawk  1.698335
40273      riverside  1.594129
52419       workouts  1.551486
11423         cobble  1.460318
19942        foilage  1.399307
15907          domus  1.392133
22888       hardwood  1.390890
20146       foretold  1.374990
25802         inward  1.354597
52340      wonferful  1.340032
5237            atic  1.303804
19022    fashionable  1.300339
29460     manageable  1.300150
1211             320  1.296880
44704       stairway  1.295137
4072        andaluca  1.293610
6684      beautfully  1.279282
23963  hospitalitywe  1.276973
49952         unused  1.275191
48082          tores  1.267326
44135       spanning  1.259473
25594      interfere  1.258216
5389      attributes  1.257805
43981        sorting  1.251339
```

This output shows the top 25 features selected using linear SVC as part of the classification algorithm based univariate feature selection process. These weights reflect the importance or relevance of each feature in the classification task. Higher weights suggest that the feature has a stronger influence on the classification decisions made by the SVM model.

```
dt = DecisionTreeClassifier(random_state=1)
tree_weighted_matrix = tmu.clf_univariate_fs(new_count_matrix, y, dt,
num_features=25, scores_to_print=25)
```

```
Top 25 features:
       Attribute    Weight
32744        not  0.030377
21730       good  0.028924
52313  wonderful  0.016423
3               ,  0.016364
18171  excellent  0.014943
18951  fantastic  0.010337
40505       room  0.009783
52467      worst  0.009623
24000      hotel  0.009305
15274      dirty  0.008643
28836      loved  0.008236
28546   location  0.007854
35295    perfect  0.007839
33309         ok  0.007702
44891       stay  0.006765
5               .  0.006696
47955       told  0.006571
7165        best  0.006557
32582         no  0.005964
11152      clean  0.005576
31854        n't  0.004953
44624      staff  0.004939
5581     average  0.004816
23261    helpful  0.004591
23472     highly  0.004326
```

This output shows the top 25 features selected using the decision tree classifier as part of the classification algorithm based univariate feature selection process. These weights reflect the importance or relevance of each feature in the classification task.

```
tmu.printClassifReport(clf, svm_weighted_matrix, y)
tmu.printClassifReport(clf, tree_weighted_matrix, y)
```

	precision	recall	f1-score	support
1	0.65	0.56	0.60	1421
2	0.42	0.42	0.42	1793
3	0.35	0.27	0.31	2184
4	0.48	0.49	0.49	6039
5	0.72	0.77	0.74	9054
accuracy			0.59	20491
macro avg	0.53	0.50	0.51	20491
weighted avg	0.58	0.59	0.58	20491

	precision	recall	f1-score	support
1	0.60	0.66	0.63	1421
2	0.34	0.39	0.37	1793
3	0.35	0.31	0.33	2184
4	0.51	0.42	0.46	6039
5	0.71	0.78	0.74	9054
accuracy			0.58	20491
macro avg	0.50	0.51	0.51	20491
weighted avg	0.57	0.58	0.57	20491

This output shows the classification reports for both linear SVC and decision tree. We can see that for both classification reports the scores are quite good for ratings that are 1 or 5 but the scores for 2, 3 and 4 are much poorer. This is similar to the output for statistical univariate feature selection as it is much easier to predict a review that is extremely negative as 1 or a review that is very positive as a 5 but it can be quite difficult to determine reviews in-between. We can see from the above output that linear SVC has produced slightly better results than decision tree.

From looking at the terms output most of the terms output are expected in hotel reviews such as good, bad, excellent, dirty, clean, etc. But linear SVC is an exception, the words output for linear SVC are quite unique and some feature misspellings, this leads me to believe that linear SVC should not be featured in the final pipeline. Out of the remaining processes ANOVA produces the most efficient results as it has a similar precision, recall and accuracy score to the others but is the only one to ignore punctuation and consists only of words relevant to our cause. For this reason I have determined that this feature selection process is best suited to be used in our final pipeline.

Hyperparameter Tuning

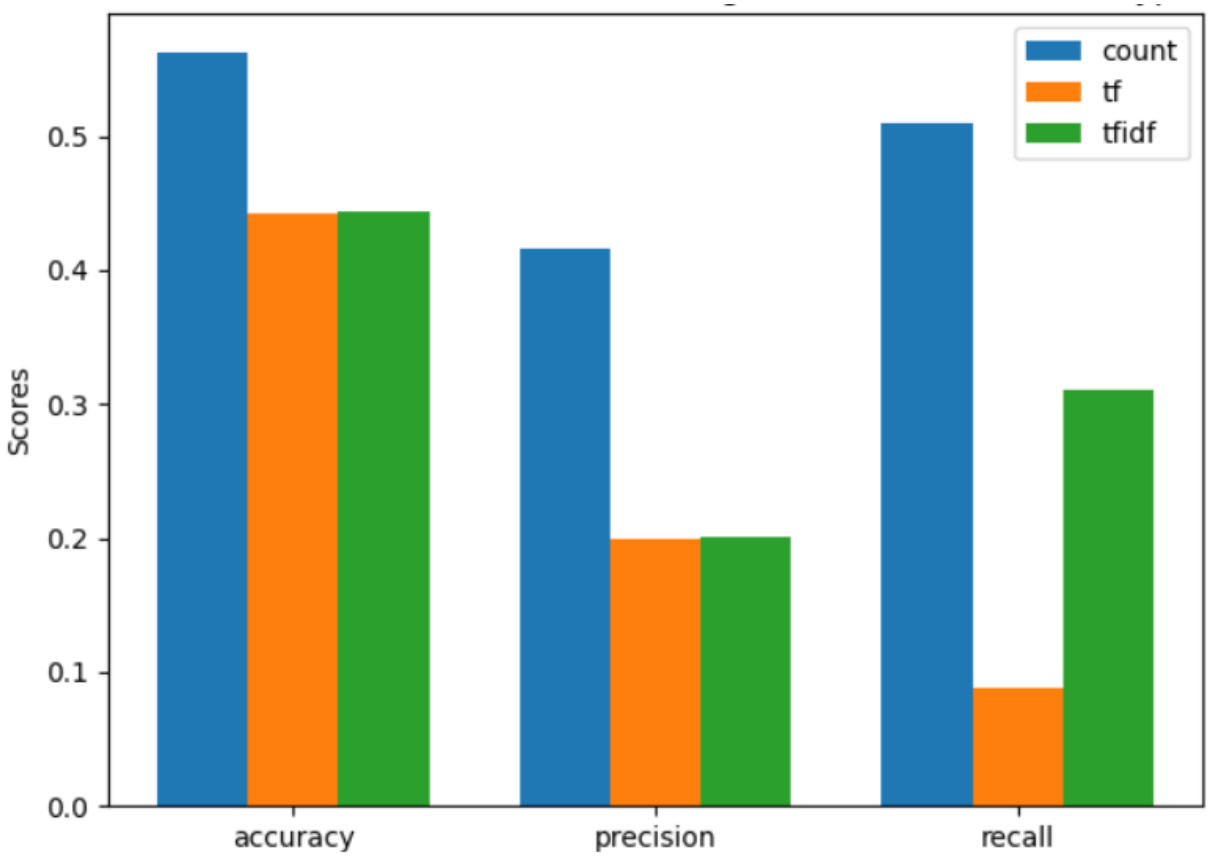
Hyperparameter tuning in text analytics involves optimizing the settings of a machine learning model to achieve the best performance for a given text analysis task. This process entails selecting an appropriate model, defining the hyperparameters to be tuned, specifying a search space for these hyperparameters, and choosing an evaluation metric.[12] By using techniques like grid search, different combinations of hyperparameters are evaluated on a validation set

until the optimal configuration is found. This optimized model is then further evaluated on a separate test set to ensure its performance. By fine-tuning the hyperparameters, text analytics models can be tailored to specific tasks, resulting in improved accuracy and effectiveness in real-world applications.[12]

For our dataset we will be training three different classifiers on our final chosen feature selection model. The classifiers chosen include linear SVC, naïve bayes and decision tree. We will compare the performance of the matrices before and after hyperparameter tuning to ensure that performance has improved. To evaluate the performances of the matrices before hyperparameter tuning we must run the following code:

```
matrices = [new_count_matrix, new_tf_matrix, new_tfidf_matrix]
matrices_names = ['count', 'tf', 'tfidf']
tmu.plot_avg_performance_for_3matrices(clf, 'Performance before hyperparameter
tuning',
matrices, matrices_names, y)
```

Which outputs the following:



As we can see the accuracy, precision and recall for the count matrix is quite high but the tf and tfidf matrices are much lower. This will improve following hyperparameter tuning.

```
from sklearn.naive_bayes import ComplementNB
from sklearn.model_selection import GridSearchCV

dt_clf = DecisionTreeClassifier(random_state=1)
```

```
dt_params = {
'criterion': ['gini', 'entropy'],
'max_depth': range(3, 11),
'min_samples_leaf': range(3, 10),
'min_samples_split': range(3, 10),
'min_impurity_decrease': [0.01, 0.02, 0.03]
}
dt_grid_search = GridSearchCV(dt_clf, param_grid=dt_params)
dt_grid_search.fit(count_anova_matrix, y)
print(dt_grid_search.best_params_)
print(dt_grid_search.best_estimator_)
print(dt_grid_search.best_score_)
```

This code segment aims to fine-tune the hyperparameters of a Decision Tree classifier using Grid Search Cross-Validation, a technique for searching through a specified set of hyperparameters to find the combination that yields the best performance. First, a DecisionTreeClassifier is instantiated, then a dictionary, dt_params, is defined, containing the hyperparameters to be optimized, along with their ranges of values. GridSearchCV is then used to search through this parameter grid, fitting the classifier to the data (count_anova_matrix representing feature matrix and y the target variable) and evaluating its performance through cross-validation. Finally, the best combination of hyperparameters, the corresponding estimator, and its mean cross-validated score are printed, providing insights into the optimal configuration for the Decision Tree classifier. This code outputs the following results:

```
{'criterion': 'entropy', 'max_depth': 5, 'min_impurity_decrease': 0.01, 'min_samples_leaf': 3, 'min_samples_split': 3}
DecisionTreeClassifier(criterion='entropy', max_depth=5,
                       min_impurity_decrease=0.01, min_samples_leaf=3,
                       min_samples_split=3, random_state=1)
0.4593234360271423
```

The number 0.4593234360271423 in the output represents the mean cross-validated score achieved by the DecisionTreeClassifier model. This score is a measure of how well the model performs on unseen data and is calculated by averaging the performance across different folds in the cross-validation process.

```
dt_results = pd.DataFrame(dt_grid_search.cv_results_)
dt_results.head()
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	param_max_depth	param_min_impurity_decrease	param_min_samples_leaf	param_mi
0	0.027705	0.020552	0.003598	0.003713	gini	3	0.01	3	
1	0.015506	0.001500	0.001801	0.000400	gini	3	0.01	3	
2	0.014304	0.002888	0.001400	0.000490	gini	3	0.01	3	
3	0.015302	0.003200	0.001601	0.000491	gini	3	0.01	3	
4	0.013800	0.000751	0.001302	0.000600	gini	3	0.01	3	

We then store the results of the cross-validation performed during the grid search for the DecisionTreeClassifier model. This DataFrame contains information about various hyperparameters tested, their corresponding mean validation scores, standard deviations, and other relevant metrics.

We then do the same for Linear SVC.

```
from sklearn.svm import LinearSVC
from sklearn.model_selection import GridSearchCV

svm_clf = LinearSVC(random_state=1)
svm_params = {
    'C': range(1, 6),
}
svm_grid_search = GridSearchCV(svm_clf, param_grid=svm_params)
svm_grid_search.fit(count_anova_matrix, y)
print(svm_grid_search.best_params_)
print(svm_grid_search.best_estimator_)
print(svm_grid_search.best_score_)
```

```
{'C': 5}
LinearSVC(C=5, random_state=1)
0.5301835096253047
```

This process enables the selection of the most effective 'C' value for the LinearSVC model, enhancing its performance on the given dataset.

```
svm_results = pd.DataFrame(svm_grid_search.cv_results_)
svm_results.head()
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score	split4_test
0	4.371761	0.353592	0.005568	0.006445	1	{'C': 1}	0.539400	0.520498	0.515373	0.535383	0.5
1	4.281534	0.029285	0.002122	0.000214	2	{'C': 2}	0.533057	0.514641	0.512201	0.538067	0.5
2	4.436366	0.046155	0.001814	0.000408	3	{'C': 3}	0.540376	0.518546	0.507565	0.538555	0.5
3	4.682632	0.262501	0.001995	0.000015	4	{'C': 4}	0.542327	0.519522	0.514885	0.532943	0.5
4	6.790452	2.492141	0.002499	0.001052	5	{'C': 5}	0.539888	0.518302	0.513909	0.548316	0.5

Lastly we repeat the process for the Naïve Bayes classifier.

```
nb_clf = ComplementNB()
nb_params = {
'alpha': [0, .1, .2, .3, .4, .75, 1],
'norm': [True, False]
}
nb_grid_search = GridSearchCV(nb_clf, param_grid=nb_params)
nb_grid_search.fit(count_anova_matrix, y)
print(nb_grid_search.best_params_)
print(nb_grid_search.best_estimator_)
print(nb_grid_search.best_score_)
```

```
{'alpha': 1, 'norm': False}
ComplementNB(alpha=1)
0.4923628839230509
```

```
nb_results = pd.DataFrame(nb_grid_search.cv_results_)
nb_results.head()
```

mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	param_norm	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score
0.018309	0.007789	0.001403	0.000488	0	True	{'alpha': 0, 'norm': True}	0.501098	0.479746	0.475598	0.497316
0.011907	0.000499	0.001599	0.000491	0	False	{'alpha': 0, 'norm': False}	0.484508	0.483163	0.496340	0.506833
0.015510	0.001820	0.001796	0.000412	0.1	True	{'alpha': 0.1, 'norm': True}	0.501342	0.479502	0.475598	0.497316
0.014918	0.002622	0.001791	0.000397	0.1	False	{'alpha': 0.1, 'norm': False}	0.484508	0.483163	0.496340	0.506833
0.012600	0.002333	0.001601	0.000489	0.2	True	{'alpha': 0.2, 'norm': True}	0.501342	0.479502	0.475598	0.497316

Now that the hyperparameters have been tuned across all three classifiers and the optimal parameters have been selected we must then print the classification reports to compare the results.

```
optimal_dt = DecisionTreeClassifier(max_depth=3, min_impurity_decrease=0.01,
min_samples_leaf=3, min_samples_split=3, random_state=1)
optimal_svm = LinearSVC(C=5, random_state=1)
optimal_nb = ComplementNB(alpha=0.3, norm=True)
tmu.printClassifReport(optimal_dt, count_anova_matrix, y)
```

```
tmu.printClassifReport(optimal_svm, count_anova_matrix, y)
tmu.printClassifReport(optimal_nb, count_anova_matrix, y)
```

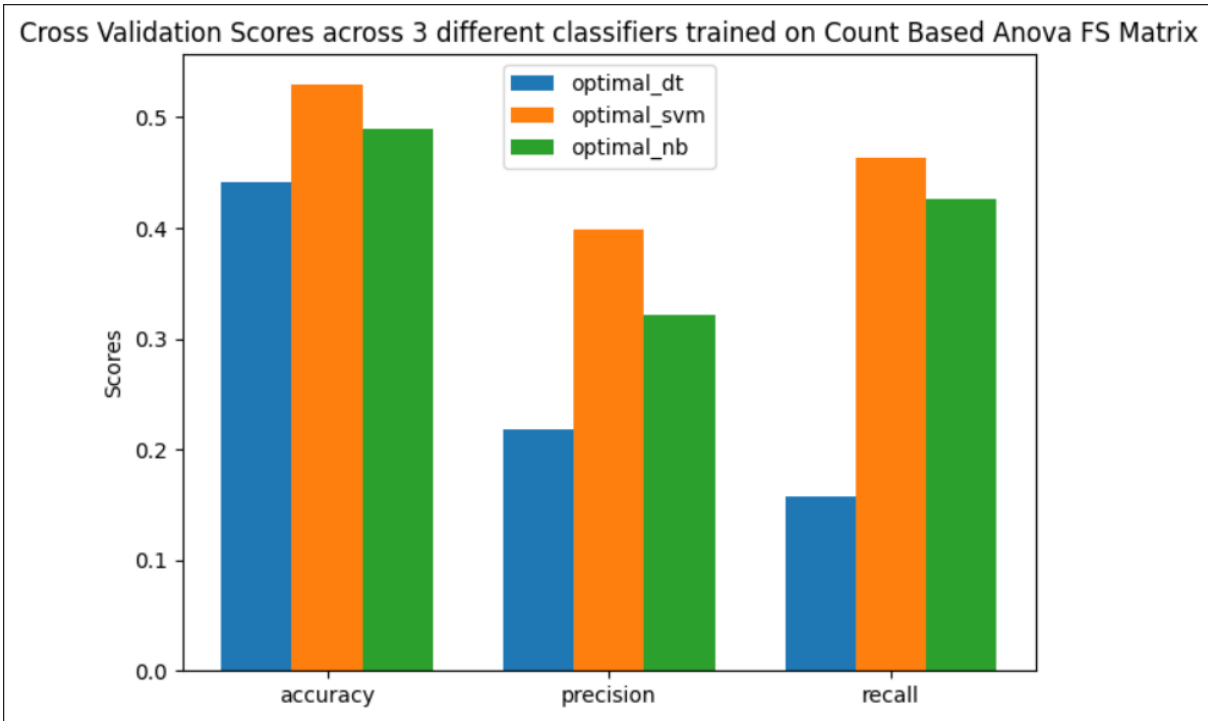
	precision	recall	f1-score	support
1	0.00	0.00	0.00	1421
2	0.00	0.00	0.00	1793
3	0.00	0.00	0.00	2184
4	0.29	0.27	0.28	6039
5	0.50	0.82	0.62	9054
accuracy			0.44	20491
macro avg	0.16	0.22	0.18	20491
weighted avg	0.31	0.44	0.36	20491

	precision	recall	f1-score	support
1	0.54	0.58	0.56	1421
2	0.38	0.15	0.22	1793
3	0.39	0.10	0.16	2184
4	0.43	0.31	0.36	6039
5	0.58	0.85	0.69	9054
accuracy			0.53	20491
macro avg	0.46	0.40	0.40	20491
weighted avg	0.49	0.53	0.48	20491

	precision	recall	f1-score	support
1	0.60	0.18	0.28	1421
2	0.28	0.12	0.17	1793
3	0.22	0.07	0.10	2184
4	0.36	0.59	0.45	6039
5	0.68	0.64	0.66	9054
accuracy			0.49	20491
macro avg	0.43	0.32	0.33	20491
weighted avg	0.49	0.49	0.47	20491

We then must check the performance of our matrices across the three different classifiers post hyperparameter tuning.

```
clfs = [optimal_dt, optimal_svm, optimal_nb]
clf_names = ['optimal_dt', 'optimal_svm', 'optimal_nb']
tmu.plot_avg_performance_across_3clfs(clfs, clf_names, count_anova_matrix,
'Count Based Anova FS Matrix', y)
```



We can see from the above graph that linear SVC is performing best, closely followed my Naïve Bayes. The decision tree classifier is by far the worst performing classifier when it comes to precision and recall, as far as accuracy goes it performs quite well but is still the worst out of the three.

Overall evaluation/Conclusion

Now that our dataset has been successfully analysed, let's review the results and determine whether the outcome can be considered a success and how it may be improved. Let's first look at our vectorization techniques. Out of the three vectorization techniques chosen for analysing the dataset the count vectorization produced the best results. This is unsurprising as it is one of the simplest form of vectorization as count vectorization is merely the process of counting the number of times a word/symbol/number appeared in a row. The other vectorization techniques where slightly more complex therefore when they where used for creating our matrices it is to be expected that the predictions aren't as accurate.

Next, let's evaluate our data preparation techniques. The data preparation techniques we used included dealing with synonyms, concepts/hypernyms, and word variations, removing stop words and removing punctuation. Out of these three data preparation techniques we can see that dealing with synonyms, concepts/hypernyms, and word variations produced the best results by comparing the classification reports. This was the one technique chosen to make it to the final pipeline as it was the only one that seemingly improved our data analysis.

Out of the feature selection models tried and tested the one that we determined best suited to the task at hand was the ANOVA model. We tested four different techniques including linear SVC, decision tree, anova and chi2. To compare these models we output the 25 most predicted terms using them and also printed classification reports to assess their accuracy and precision. Although linear SVC had the highest scores on the classification report I determined it not worthy of being in the final pipeline as the 25 terms it output appeared to be inaccurate, many

seemed to be uncommonly used words and many featured misspellings. Once linear SVC was ruled out it was tough to choose between the remaining three as they all shared similar classification scores but I settled on ANOVA as the right choice for the final pipeline as it predicted only relevant terms without getting caught on punctuation such as commas, periods, etc.

Of the 25 terms output some of the terms that where output most frequently include 'not', 'good', 'dirty', 'worst', 'excellent', 'no'. This is unsurprising and concur with those I predicted during data understanding; this is because these are terms that are commonly associated with hotel reviews. This is a good sign as many of these words make it easy to determine whether the review is positive or negative meaning we can use this data for sentiment analyse if we wish. For example, if a review features the word 'worst' then it is safe to determine that the review is negative.

Overall, the assignment was successful although improvements can still be made. One drawback I noticed is the low scores for the middle numbers in the classification reports. That is because although it is somewhat easy to predict whether a review is positive or negative it is difficult to be specific enough to determine the difference between a 1 star or 2 star review or a 4 star or 5 star review. In order to get higher scores in future we should group numbers that suggest the same sentiment together. We can group reviews that have 1 and 2 stars together and label them simply as 'Negative', then we can group reviews with 4 and 5 stars together and label them as 'Positive', reviews with 3 stars are neither positive nor negative so we will create a separate group for that called 'Average'. This mean instead of have five sets of scores to predict we will have three. By generalizing the reviews sentiment this should make them easier to predict thus increasing their classification scores.

In conclusion, this project was a success but there is still work to be done before this data is actionable. Through examination and data analysis techniques, we've gained valuable insights into customer sentiments, preferences, and areas to be improved. This project has not only enhanced our understanding of the hospitality industry but has also equipped us with strategies to increase customer satisfaction and improve overall guest experience.

References

1. Lecture 1: Introduction to Text Mining by Aurelia Power - https://vle-bn.tudublin.ie/pluginfile.php/250000/mod_resource/content/7/Lecture1.pdf
2. Vectorization Techniques in NLP [Guide] by Abhishek Jha - https://neptune.ai/blog/vectorization-techniques-in-nlp-guide
3. Lecture 4: More Vectorisation Methods and Text Understanding by Aurelia Power - https://vle-bn.tudublin.ie/pluginfile.php/258769/mod_resource/content/9/Lecture%204.pdf
4. Text Classification: What it is And Why it Matters - https://monkeylearn.com/text-classification/#:~:text=Tutorial-,What%20is%20Text%20Classification%3F,and%20all%20over%20the%20web.
5. Cross-validation: evaluating estimator performance - https://scikit-learn.org/stable/modules/cross_validation.html
6. Lecture 7: Text Preprocessing by Aurelia Power - https://vle-bn.tudublin.ie/pluginfile.php/262853/mod_resource/content/10/Lecture7.pdf

7. Lecture 5: Clustering 1 by Aurelia Power - https://vle-bn.tudublin.ie/pluginfile.php/178320/mod_resource/content/10/Lecture5.pdf
8. Lecture 6: Clustering 2 by Aurelia Power - https://vle-bn.tudublin.ie/pluginfile.php/259762/mod_resource/content/6/Lecture6.pdf
9. A Gentle Introduction to Sparse Matrices for Machine Learning by Jason Brownlee - https://machinelearningmastery.com/sparse-matrices-for-machine-learning/#:~:text=A%20sparse%20matrix%20is%20a,of%20its%20coefficients%20are%20zero.
10. Lecture 8: Advanced Word Embeddings and Feature Selection by Aurelia Power - https://vle-bn.tudublin.ie/pluginfile.php/464485/mod_resource/content/1/Lecture8.pdf
11. ANOVA and Chi-Square by Chandradip Banerjee - https://medium.com/@chandradip93/anova-and-chi-square-aea693c4eb96#:~:text=In%20summary%2C%20ANOVA%20is%20used,or%20independence%20between%20categorical%20variables.
12. Hyperparameter tuning - https://www.geeksforgeeks.org/hyperparameter-tuning/

Appendix

TextAnalyticsAssignment.ipynb

```
import pandas as pd, numpy as np
import nltk, re
from string import punctuation
from collections import Counter

from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB

import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')

%ls

import text_mining_utils as tmu

data = pd.read_csv('tripadvisor_hotel_reviews.csv')

data.sample(3)
clf = MultinomialNB()
y = data.Rating

baseline_count_matrix = tmu.build_count_matrix(data.Review)
```

```
baseline_count_matrix.head()
baseline_tf_matrix = tmu.build_tf_matrix(data.Review)
baseline_tf_matrix.head()
baseline_tfidf_matrix = tmu.build_tfidf_matrix(data.Review)
baseline_tfidf_matrix.head()
tmu.printClassifReport(clf, baseline_count_matrix, y)
tmu.printClassifReport(clf, baseline_tf_matrix, y)
tmu.printClassifReport(clf, baseline_tfidf_matrix, y)
from sklearn.model_selection import cross_val_score

clf_count = MultinomialNB()
clf_tf = MultinomialNB()
clf_tfidf = MultinomialNB()

scores_count = cross_val_score(clf_count, baseline_count_matrix, y, cv=5)
scores_tf = cross_val_score(clf_tf, baseline_tf_matrix, y, cv=5)
scores_tfidf = cross_val_score(clf_tfidf, baseline_tfidf_matrix, y, cv=5)

print("Count Matrix:")
print("Accuracy:", scores_count.mean())
print("Standard Deviation:", scores_count.std())

print("\nTF Matrix:")
print("Accuracy:", scores_tf.mean())
print("Standard Deviation:", scores_tf.std())

print("\nTF-IDF Matrix:")
print("Accuracy:", scores_tfidf.mean())
print("Standard Deviation:", scores_tfidf.std())
from text_mining_utils import print_n_mostFrequent

one_star_reviews = ' '.join(data[data['Rating'] == 1]['Review'])
two_star_reviews = ' '.join(data[data['Rating'] == 2]['Review'])
three_star_reviews = ' '.join(data[data['Rating'] == 3]['Review'])
four_star_reviews = ' '.join(data[data['Rating'] == 4]['Review'])
five_star_reviews = ' '.join(data[data['Rating'] == 5]['Review'])

print_n_mostFrequent("1-star", one_star_reviews, 10)
print_n_mostFrequent("2-star", two_star_reviews, 10)
print_n_mostFrequent("3-star", three_star_reviews, 10)
print_n_mostFrequent("4-star", four_star_reviews, 10)
print_n_mostFrequent("5-star", five_star_reviews, 10)
def plot_bar_chart(category, top_terms):
    plt.figure(figsize=(10, 6))
    plt.bar(range(len(top_terms)), top_terms.values(), align='center',
color='skyblue')
    plt.xticks(range(len(top_terms)), top_terms.keys(), rotation=45)
    plt.ylabel('Frequency')
```

```
    plt.xlabel('Term')
    plt.title(f'Top Terms in {category}')
    plt.show()

category = '1-star'
top_terms = { 'not': 0.021, 'hotel': 0.020, 'room': 0.018, 'nt': 0.009, 'no':
0.0088, 'did': 0.007, 'stay': 0.0068, 'staff': 0.0054, 'rooms': 0.005}

plot_bar_chart(category, top_terms)
def plot_bar_chart(category, top_terms):
    plt.figure(figsize=(10, 6))
    plt.bar(range(len(top_terms)), top_terms.values(), align='center',
color='green')
    plt.xticks(range(len(top_terms)), top_terms.keys(), rotation=45)
    plt.ylabel('Frequency')
    plt.xlabel('Term')
    plt.title(f'Top Terms in {category}')
    plt.show()

category = '2-star'
top_terms = { 'not': 0.02, 'room': 0.017, 'hotel': 0.0165, 'nt': 0.009, 'no':
0.0078, 'did': 0.0072, 'good': 0.006, 'stay': 0.0054, 'rooms': 0.0054}

plot_bar_chart(category, top_terms)
def plot_bar_chart(category, top_terms):
    plt.figure(figsize=(10, 6))
    plt.bar(range(len(top_terms)), top_terms.values(), align='center',
color='red')
    plt.xticks(range(len(top_terms)), top_terms.keys(), rotation=45)
    plt.ylabel('Frequency')
    plt.xlabel('Term')
    plt.title(f'Top Terms in {category}')
    plt.show()

category = '3-star'
top_terms = { 'hotel': 0.018, 'not': 0.017, 'room': 0.016, 'nt': 0.009,
'good': 0.0088, 'did': 0.007, 'great': 0.00627, 'nice': 0.00624, 'no': 0.006}

plot_bar_chart(category, top_terms)
def plot_bar_chart(category, top_terms):
    plt.figure(figsize=(10, 6))
    plt.bar(range(len(top_terms)), top_terms.values(), align='center',
color='blue')
    plt.xticks(range(len(top_terms)), top_terms.keys(), rotation=45)
    plt.ylabel('Frequency')
    plt.xlabel('Term')
    plt.title(f'Top Terms in {category}')
    plt.show()
```

```
category = '4-star'
top_terms = {'hotel': 0.0195, 'room': 0.014, 'not': 0.0122, 'great': 0.01,
'good': 0.009, 'nt': 0.008, 'nice': 0.0066, 'staff': 0.0064, 'did': 0.0062}

plot_bar_chart(category, top_terms)
def plot_bar_chart(category, top_terms):
    plt.figure(figsize=(10, 6))
    plt.bar(range(len(top_terms)), top_terms.values(), align='center',
color='purple')
    plt.xticks(range(len(top_terms)), top_terms.keys(), rotation=45)
    plt.ylabel('Frequency')
    plt.xlabel('Term')
    plt.title(f'Top Terms in {category}')
    plt.show()

category = '5-star'
top_terms = {'hotel': 0.02, 'room': 0.013, 'great': 0.011, 'not': 0.0094,
'staff': 0.008, 'stay': 0.007, 'nt': 0.0067, 'good': 0.006, 'just': 0.0054}

plot_bar_chart(category, top_terms)
tmu.generate_wordclouds([one_star_reviews, two_star_reviews,
three_star_reviews, four_star_reviews, five_star_reviews],
['1-star', '2-star', '3-star', '4-star', '5-star'],
'black')
from sklearn.metrics import adjusted_rand_score, fowlkes_mallows_score
from sklearn.metrics.pairwise import cosine_distances, cosine_similarity
from sklearn.metrics import calinski_harabasz_score, davies_bouldin_score

sims = cosine_similarity(baseline_tfidf_matrix)
dists = cosine_distances(baseline_tfidf_matrix)
plt.figure(figsize=(10, 5))
plt.plot(sims[0])
plt.plot(dists[0])
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from scipy.sparse import csr_matrix

if not isinstance(baseline_tfidf_matrix, csr_matrix):
    baseline_tfidf_matrix = csr_matrix(baseline_tfidf_matrix)

k = 5
kmeans_r = KMeans(n_clusters=k, init='random', n_init=1)
cluster_labels_r = kmeans_r.fit_predict(baseline_tfidf_matrix)
cluster_centers_r = kmeans_r.cluster_centers_

print(list(y))
print(cluster_labels_r)
```

```
plt.scatter(cluster_centers_r[:, 0], cluster_centers_r[:, 1], c='blue', s=200,
alpha=0.5)

kmeans_p = KMeans(n_clusters=k, init='k-means++', n_init=1)
cluster_labels_p = kmeans_p.fit_predict(baseline_tfidf_matrix)
cluster_centers_p = kmeans_p.cluster_centers_

print(list(y))
print(cluster_labels_p)
plt.scatter(cluster_centers_p[:, 0], cluster_centers_p[:, 1], c='yellow',
s=50, alpha=0.5)

plt.show()
from sklearn.metrics import adjusted_rand_score, fowlkes_mallows_score
from sklearn.metrics.pairwise import cosine_distances, cosine_similarity
from sklearn.metrics import calinski_harabasz_score, davies_bouldin_score

print("Kmeans Random - ARI score:",
      adjusted_rand_score(cluster_labels_r, list(y)))
print("Kmeans++ - ARI score:",
      adjusted_rand_score(cluster_labels_p, list(y)))
print("Kmeans Random - FMI score:",
      fowlkes_mallows_score(cluster_labels_r, list(y)))
print("Kmeans++ - FMI score:",
      fowlkes_mallows_score(cluster_labels_p, list(y)))

if not isinstance(baseline_count_matrix, csr_matrix):
    baseline_count_matrix = csr_matrix(baseline_count_matrix)

k = 5
kmeans_a = KMeans(n_clusters=k, init='random', n_init=1)
cluster_labels_a = kmeans_a.fit_predict(baseline_count_matrix)
cluster_centers_a = kmeans_a.cluster_centers_

kmeans_b = KMeans(n_clusters=k, init='k-means++', n_init=1)
cluster_labels_b = kmeans_b.fit_predict(baseline_tfidf_matrix)
cluster_centers_b = kmeans_b.cluster_centers_

print("Kmeans Random - ARI score:",
      adjusted_rand_score(cluster_labels_a, list(y)))
print("Kmeans++ - ARI score:",
      adjusted_rand_score(cluster_labels_b, list(y)))
print("Kmeans Random - FMI score:",
      fowlkes_mallows_score(cluster_labels_a, list(y)))
print("Kmeans++ - FMI score:",
      fowlkes_mallows_score(cluster_labels_b, list(y)))

if not isinstance(baseline_count_matrix, csr_matrix):
```

```
    baseline_count_matrix = csr_matrix(baseline_count_matrix)

k = 5
kmeans_c = KMeans(n_clusters=k, init='random', n_init=1)
cluster_labels_c = kmeans_c.fit_predict(baseline_count_matrix)
cluster_centers_c = kmeans_c.cluster_centers_

kmeans_d = KMeans(n_clusters=k, init='k-means++', n_init=1)
cluster_labels_d = kmeans_d.fit_predict(baseline_tfidf_matrix)
cluster_centers_d = kmeans_d.cluster_centers_

print("Kmeans Random - ARI score:",
      adjusted_rand_score(cluster_labels_c, list(y)))
print("Kmeans++ - ARI score:",
      adjusted_rand_score(cluster_labels_d, list(y)))
print("Kmeans Random - FMI score:",
      fowlkes_mallows_score(cluster_labels_c, list(y)))
print("Kmeans++ - FMI score:",
      fowlkes_mallows_score(cluster_labels_d, list(y)))

num_terms = baseline_count_matrix.shape[1]
baseline_count_matrix_df = pd.DataFrame(baseline_count_matrix[:30].todense(),
columns=[f'term_{i+1}' for i in range(num_terms)])

tmu.plot_clusters(baseline_count_matrix_df, cluster_labels_a[:30], "K-Means:
random", 'Comments', 'Cluster Labels')

num_terms = baseline_count_matrix.shape[1]
baseline_count_matrix_df = pd.DataFrame(baseline_count_matrix[:30].todense(),
columns=[f'term_{i+1}' for i in range(num_terms)])

tmu.plot_clusters(baseline_count_matrix_df, cluster_labels_b[:30], "K-Means:
++", 'Comments', 'Cluster Labels')

num_terms = baseline_tf_matrix.shape[1]
baseline_tf_matrix_df = pd.DataFrame(baseline_tf_matrix[:30],
columns=[f'term_{i+1}' for i in range(num_terms)])

tmu.plot_clusters(baseline_tf_matrix_df, cluster_labels_c[:30], "K-Means:
random", 'Comments', 'Cluster Labels')

num_terms = baseline_tf_matrix.shape[1]
baseline_tf_matrix_df = pd.DataFrame(baseline_tf_matrix[:30],
columns=[f'term_{i+1}' for i in range(num_terms)])

tmu.plot_clusters(baseline_tf_matrix_df, cluster_labels_d[:30], "K-Means: ++",
'Comments', 'Cluster Labels')
```

```
num_terms = baseline_tfidf_matrix.shape[1]
baseline_tfidf_matrix_df = pd.DataFrame(baseline_tfidf_matrix[:30].todense(),
columns=[f'term_{i+1}' for i in range(num_terms)])

tmu.plot_clusters(baseline_tfidf_matrix_df, cluster_labels_r[:30], "K-Means:
random", 'Comments', 'Cluster Labels')
num_terms = baseline_tfidf_matrix.shape[1]
baseline_tfidf_matrix_df = pd.DataFrame(baseline_tfidf_matrix[:30].todense(),
columns=[f'term_{i+1}' for i in range(num_terms)])

tmu.plot_clusters(baseline_tfidf_matrix_df, cluster_labels_p[:30], "K-Means:
++", 'Comments', 'Cluster Labels')
clean_operations = {
r'(\(.+?\))+' : '',
r'(\[.+?\])+' : '',
r'\s+' : ' ',
r'\s{2,}' : ' ',
}
clean_data = data.copy()
clean_data.Review = clean_data.Review.apply(tmu.clean_doc,
clean_operations=clean_operations)
clean_data.head()

clean_count_matrix = tmu.build_count_matrix(clean_data.Review)
clean_tf_matrix = tmu.build_tf_matrix(clean_data.Review)
clean_tfidf_matrix = tmu.build_tfidf_matrix(clean_data.Review)
tmu.printClassifReport(clf, clean_count_matrix, y)
tmu.printClassifReport(clf, clean_tf_matrix, y)
tmu.printClassifReport(clf, clean_tfidf_matrix, y)
repl_dict = {
'room': [r'\broom(s)\b', r'\bhotelroom\b' r'\bedroom?\b'],
'hotel': [r'\bhotel\b', r'\bresort\b'],
'good': [r'\bgood?\b', r'\bgreat?\b', r'\bnice?\b']
}

docs = clean_data.Review
docs_repl = docs.apply(tmu.improve_bow, repl_dict=repl_dict)

new_count_matrix = tmu.build_count_matrix(docs_repl)
new_tf_matrix = tmu.build_tf_matrix(docs_repl)
new_tfidf_matrix = tmu.build_tfidf_matrix(docs_repl)

print(new_count_matrix.head(1))
print(new_tf_matrix.head(1))
new_tfidf_matrix.head(1)
tmu.printClassifReport(clf, new_count_matrix, y)
tmu.printClassifReport(clf, new_tf_matrix, y)
```

```
tmu.printClassifReport(clf, new_tfidf_matrix, y)
nltk.download('stopwords')
univ_sw = nltk.corpus.stopwords.words('english')
print(univ_sw)
nltk.download('punkt')
no_univsw_docs = docs.apply(tmu.remove_sw_punct, to_remove=univ_sw)

new_count_matrix = tmu.build_count_matrix(no_univsw_docs)
new_tf_matrix = tmu.build_tf_matrix(no_univsw_docs)
new_tfidf_matrix = tmu.build_tfidf_matrix(no_univsw_docs)

print(new_count_matrix.head(1))
print(new_tf_matrix.head(1))
new_tfidf_matrix.head(1)
tmu.printClassifReport(clf, new_count_matrix, y)
tmu.printClassifReport(clf, new_tf_matrix, y)
tmu.printClassifReport(clf, new_tfidf_matrix, y)
import string
punct = list(string.punctuation)
no_punct_docs = no_univsw_docs.apply(tmu.remove_sw_punct, to_remove=punct)
new_count_matrix = tmu.build_count_matrix(no_punct_docs)
new_tf_matrix = tmu.build_tf_matrix(no_punct_docs)
new_tfidf_matrix = tmu.build_tfidf_matrix(no_punct_docs)
print(new_count_matrix.head(1))
tmu.printClassifReport(clf, new_count_matrix, y)
tmu.printClassifReport(clf, new_tf_matrix, y)
tmu.printClassifReport(clf, new_tfidf_matrix, y)
from sklearn.feature_selection import chi2, f_classif

count_chi2_matrix = tmu.stat_univariate_fs(new_count_matrix, y,
weight_method=chi2,
selection_method='k_best',
num_features=25, scores_to_print=25
)
count_anova_matrix = tmu.stat_univariate_fs(new_count_matrix, y,
weight_method=f_classif,
selection_method='k_best',
num_features=25, scores_to_print=25
)
tmu.printClassifReport(clf, count_chi2_matrix, y)
tmu.printClassifReport(clf, count_anova_matrix, y)
linear_svm = LinearSVC(random_state=1)
svm_weighted_matrix = tmu.clf_univariate_fs(new_count_matrix, y, linear_svm,
num_features=25, scores_to_print=25)

dt = DecisionTreeClassifier(random_state=1)
tree_weighted_matrix = tmu.clf_univariate_fs(new_count_matrix, y, dt,
num_features=25, scores_to_print=25)
```

```
tmu.printClassifReport(clf, svm_weighted_matrix, y)
tmu.printClassifReport(clf, tree_weighted_matrix, y)
matrices = [new_count_matrix, new_tf_matrix, new_tfidf_matrix]
matrices_names = ['count', 'tf', 'tfidf']
tmu.plot_avg_performance_for_3matrices(clf, 'Performance before hyperparameter
tuning',
matrices, matrices_names, y)
from sklearn.naive_bayes import ComplementNB
from sklearn.model_selection import GridSearchCV

dt_clf = DecisionTreeClassifier(random_state=1)
dt_params = {
'criterion': ['gini', 'entropy'],
'max_depth': range(3, 11),
'min_samples_leaf': range(3, 10),
'min_samples_split': range(3, 10),
'min_impurity_decrease': [0.01, 0.02, 0.03]
}
dt_grid_search = GridSearchCV(dt_clf, param_grid=dt_params)
dt_grid_search.fit(count_anova_matrix, y)
print(dt_grid_search.best_params_)
print(dt_grid_search.best_estimator_)
print(dt_grid_search.best_score_)
dt_results = pd.DataFrame(dt_grid_search.cv_results_)
dt_results.head()
from sklearn.svm import LinearSVC
from sklearn.model_selection import GridSearchCV

svm_clf = LinearSVC(random_state=1)
svm_params = {
    'C': range(1, 6),
}
svm_grid_search = GridSearchCV(svm_clf, param_grid=svm_params)
svm_grid_search.fit(count_anova_matrix, y)
print(svm_grid_search.best_params_)
print(svm_grid_search.best_estimator_)
print(svm_grid_search.best_score_)

svm_results = pd.DataFrame(svm_grid_search.cv_results_)
svm_results.head()
nb_clf = ComplementNB()
nb_params = {
'alpha': [0, .1, .2, .3, .4, .75, 1],
'norm': [True, False]
}
nb_grid_search = GridSearchCV(nb_clf, param_grid=nb_params)
nb_grid_search.fit(count_anova_matrix, y)
print(nb_grid_search.best_params_)
```

```
print(nb_grid_search.best_estimator_)
print(nb_grid_search.best_score_)
nb_results = pd.DataFrame(nb_grid_search.cv_results_)
nb_results.head()
optimal_dt = DecisionTreeClassifier(max_depth=3, min_impurity_decrease=0.01,
min_samples_leaf=3, min_samples_split=3, random_state=1)
optimal_svm = LinearSVC(C=5, random_state=1)
optimal_nb = ComplementNB(alpha=0.3, norm=True)
tmu.printClassifReport(optimal_dt, count_anova_matrix, y)
tmu.printClassifReport(optimal_svm, count_anova_matrix, y)
tmu.printClassifReport(optimal_nb, count_anova_matrix, y)
clfs = [optimal_dt, optimal_svm, optimal_nb]
clf_names = ['optimal_dt', 'optimal_svm', 'optimal_nb']
tmu.plot_avg_performance_across_3clfs(clfs, clf_names, count_anova_matrix,
'Count Based Anova FS Matrix', y)
```

text_mining_utils.py

```
"""
Created on Wed Feb  7 11:42:55 2024

@author: aurelia power
"""

import re, pandas as pd, numpy as np, matplotlib.pyplot as plt
import nltk
from collections import Counter
import warnings
warnings.filterwarnings('ignore')
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import classification_report
from sklearn.model_selection import cross_val_predict
import wordcloud

from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage
from pandas.plotting import parallel_coordinates

from sklearn.feature_selection import GenericUnivariateSelect
from sklearn.feature_selection import SelectFromModel
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.feature_selection import RFECV


################################################
```

```
############ text visualisations ############

"""
takes in a list of tagged documents and a POS(as a string)
returns the normalised count of a POS for each tagged document
"""
def normalisePOSCounts(tagged_docs, pos):
    counts = []
    for doc in tagged_docs:
        count = 0
        for pair in doc:
            if pair[1] == pos:
                count += 1
        counts.append(count)
    lengths = [len(doc) for doc in tagged_docs]
    return [count/length for count, length in zip(counts, lengths)]

"""
takes in a list of documents, a POS(as a string), and a list of
categories/labels
it tags the documents and calls the above function
it then plots the normalised frequency of the POS across all labels
"""
def plotPOSFreq(docs, pos, labels):
    tagged_docs = [nltk.pos_tag(nltk.word_tokenize(doc)) for doc in docs]
    normalised_counts = normalisePOSCounts(tagged_docs, pos)
    plt.bar(np.arange(len(docs)), normalised_counts, align='center')
    plt.xticks(np.arange(len(docs)), labels, rotation=40)
    plt.xlabel('Label (Category)')
    plt.ylabel(pos + ' frequency')
    plt.title('Frequency distribution of ' + pos)

## function to generate the word cloud for a given topic/class
def generate_cloud(text, topic, bg_colour='black', min_font=10):
    cloud = wordcloud.WordCloud(width=700, height=700, random_state=1,
background_color=bg_colour, min_font_size=min_font).generate(text)
    plt.figure(figsize=(7, 7), facecolor=None)
    plt.imshow(cloud)
    ##plt.axis('off')
    plt.tight_layout(pad=0)
    plt.xlabel(topic)
    plt.xticks([])
    plt.yticks([])

## function to generate multiple word clouds for a set of
topics/classes/categories
def generate_wordclouds(texts, categories, bg_colour, min_font=10):
  fig = plt.figure(figsize=(21, 7))
```

```
  for i in range(len(texts)):
    ax = fig.add_subplot(1,3,i+1)
    cloud = wordcloud.WordCloud(width=700, height=700, random_state=1,
                      background_color=bg_colour,
                      min_font_size=min_font).generate(texts[i])
    ax.imshow(cloud)
    ax.axis('off')
    ax.set_title(categories[i])

############################################################
########## vectorisation functions ######################
"""=minFont
NOTE: all vectorisers from sklearn discard punctuation, which may not be
appropriate.
So, I have specified a regex to deal with this situation.
"""
token_regex = r"\w+(?:'\w+)?|[^\w\s]"

"""
takes in a list of documents, applies the CountVectoriser from sklearn
using the following params by default: decode_error='replace',
strip_accents=None,
lowercase=False, ngram_range=(1, 1)  then it builds and returns a data frame
"""
def build_count_matrix(docs, decode_error='replace', strip_accents=None,
lowercase=False, token_pattern=token_regex, ngram_range=(1, 1)):
    vectorizer = CountVectorizer(decode_error=decode_error,
strip_accents=strip_accents, lowercase=lowercase, token_pattern=token_pattern,
ngram_range=ngram_range)
    X = vectorizer.fit_transform(docs)
    terms = list(vectorizer.get_feature_names_out())
    count_matrix = pd.DataFrame(X.toarray(), columns=terms)
    return count_matrix.fillna(0)

## function to generate a matrix with normalised frequencies  same params as
above
def build_tf_matrix(docs, decode_error='replace', strip_accents=None,
lowercase=False, token_pattern=token_regex, ngram_range=(1, 1)):
    count_matrix = build_count_matrix(docs, decode_error=decode_error,
strip_accents=strip_accents, lowercase=lowercase, token_pattern=token_pattern,
ngram_range=ngram_range)
    doc_lengths = count_matrix.sum(axis=1)
    return count_matrix.divide(doc_lengths, axis=0)

## function to generate a matrix with tfidfs scores  same params as above
def build_tfidf_matrix(docs, decode_error='replace', strip_accents=None,
lowercase=False, token_pattern=token_regex, ngram_range=(1, 1)):
```

```
    vectorizer = TfidfVectorizer(decode_error=decode_error,
strip_accents=strip_accents, lowercase=lowercase, token_pattern=token_pattern,
ngram_range=ngram_range)
    X = vectorizer.fit_transform(docs)
    terms = list(vectorizer.get_feature_names_out())
    tfidf_matrix = pd.DataFrame(X.toarray(), columns=terms)
    return tfidf_matrix.fillna(0)

############################################################
############# validation functions ######################

"""function to train and x-validate across acc, rec, prec
and get the classification report"""
def printClassifReport(clf, X, y, folds=5):
    predictions = cross_val_predict(clf, X, y, cv=folds)
    print(classification_report(y, predictions))


"""function to train and x-validate across acc, rec, prec
for 3 different matrices and plot them"""
def plot_avg_performance_for_3matrices(clf, clf_name,matrices, matrices_names,
y, cv=5):
  from sklearn.metrics import classification_report
  from sklearn.model_selection import cross_val_predict
  results = [classification_report(cross_val_predict(clf, matrix, y, cv=5), y,
output_dict=True) for matrix in matrices]
  labels = ['accuracy', 'precision', 'recall']
  x = np.arange(len(labels)) ; width = 0.25 ; fig, ax = plt.subplots()
  rects1 = ax.bar(x , [results[0]['accuracy'], results[0]['macro
avg']['precision'], results[0]['macro avg']['recall']], width,
label=matrices_names[0])
  #ax.bar_label(rects1, padding=3)
  rects2 = ax.bar(x + width,  [results[1]['accuracy'], results[1]['macro
avg']['precision'], results[1]['macro avg']['recall']], width,
label=matrices_names[1])
  #ax.bar_label(rects2, padding=3)
  rects3 = ax.bar(x + width*2, [results[2]['accuracy'], results[2]['macro
avg']['precision'], results[2]['macro avg']['recall']], width,
label=matrices_names[2])
  #ax.bar_label(rects3, padding=3)
  ax.set_ylabel('Scores')
  ax.set_title('Cross Validation Scores across 3 different matrices ' +
clf_name)
  ax.set_xticks(x + width); ax.set_xticklabels(labels)
  ax.legend(); fig.tight_layout(); plt.show()

"""function to train and x-validate across acc, rec, prec
and across 3 different classifiers and plot them"""
```

```
def plot_avg_performance_across_3clfs(clfs, clf_names, matrix, matrix_name, y,
cv=5):
  results = [classification_report(cross_val_predict(clf, matrix, y, cv=5), y,
output_dict=True) for clf in clfs]
  labels = ['accuracy', 'precision', 'recall']
  x = np.arange(len(labels)) ; width = 0.25 ; fig, ax = plt.subplots()
  rects1 = ax.bar(x , [results[0]['accuracy'], results[0]['macro
avg']['precision'], results[0]['macro avg']['recall']], width,
label=clf_names[0])
  #ax.bar_label(rects1, padding=3)
  rects2 = ax.bar(x + width,  [results[1]['accuracy'], results[1]['macro
avg']['precision'], results[1]['macro avg']['recall']], width,
label=clf_names[1])
  #ax.bar_label(rects2, padding=3)
  rects3 = ax.bar(x + width*2, [results[2]['accuracy'], results[2]['macro
avg']['precision'], results[2]['macro avg']['recall']], width,
label=clf_names[2])
  #ax.bar_label(rects3, padding=3)
  ax.set_ylabel('Scores')
  ax.set_title('Cross Validation Scores across 3 different classifiers trained
on ' + matrix_name)
  ax.set_xticks(x + width); ax.set_xticklabels(labels)
  ax.legend(); fig.tight_layout(); plt.show()


############################################################
############## word stats functions ######################
## function to print the n most frequent tokens in a text belonging to a given
topic
def print_n_mostFrequent(topic, text, n):
    tokens = nltk.word_tokenize(text)
    counter = Counter(tokens)
    n_freq_tokens = counter.most_common(n)
    print("=== "+ str(n) + " most frequent tokens in "  + topic + " ===")
    for token in n_freq_tokens:
        print("\tFrequency of", "\"" + token[0] + "\" is:",
token[1]/len(tokens))

## function to find the frequency of a token in several texts belonging to
same topics/classes
def token_percentage(token, texts):
    token_count = 0
    all_tokens_count = 0
    for text in texts:
        tokens = nltk.word_tokenize(text)
        token_count += tokens.count(token)
        all_tokens_count += len(tokens)
    return token_count/all_tokens_count * 100
```

```
############################################################
############## preprocessing functions ####################
## function to carry out some initial cleaning
def clean_doc(doc, clean_operations):
    for key, value in clean_operations.items():
        doc = re.sub(key, value, doc)
    return doc

## function to resolve contractions
def resolve_contractions(doc, contr_dict):
    for key, value, in contr_dict.items():
        doc = re.sub(key, value, doc)
    return doc

## function to carry out concept typing, resolve synonyms and word variations
def improve_bow(doc, repl_dict):
    for key in repl_dict.keys():
        for item in repl_dict[key]:
            doc = re.sub(item, key, doc, flags=re.IGNORECASE)
    return doc

## function to remove tokens using POS  tags
def remove_terms_by_POS(doc, tags_to_remove):
    tagged_doc = nltk.pos_tag(nltk.word_tokenize(doc)) ## (sea, 'NN')
    new_doc = [pair[0] for pair in tagged_doc if pair[1] not in
tags_to_remove]
    new_doc = ' '.join(new_doc)
    ## replace space before punctuation sign
    return re.sub(r' (?=[!\.,?:;])', "", new_doc)

## function to lower case at the beginning of the sentence only
def lower_at_begining(doc):
    sents = nltk.sent_tokenize(doc)
    ##tokenised_sents = [nltk.word_tokenize(token) in sent for sent in
sents]##
    tokenised_sents = [re.sub(sent[0], sent[0].lower(), sent)
                       for sent in sents]
    return ' '.join(tokenised_sents)

## function to remove stop words and/or punctuation
def remove_sw_punct(doc, to_remove):
    tokens = nltk.word_tokenize(doc)
    return re.sub(r' (?=[!\.,?:;])', "",
                  ' '.join([token for token in tokens if token not in
to_remove]))


## function to remove short tokens
def remove_by_token_len(doc, n):
```

```
    tokens = nltk.word_tokenize(doc);
    return re.sub(r' (?=[!\.,?:;])', "",
                  ' '.join([token for token in tokens if len(token) > n]))

## function to remove digits
def remove_d(doc):
    return re.sub(r'\d+', '', doc)

## function to carry out stemming
def stem_doc(doc, stemmer):
    tokens = nltk.word_tokenize(doc)
    return ' '.join([stemmer.stem(t) for t in tokens])


##############################################################################
############################## clustering ####################################
def k_means_clustering(X, k=2, initialisation='random'):
    model = KMeans(k, init=initialisation, random_state=1)
    model.fit(X)
    cluster_labels = model.labels_
    cluster_centers = model.cluster_centers_
    return (model, cluster_labels, cluster_centers)

def centroids_across_terms(X, centers, labels, title):
    labels = sorted(set(labels))
    centersFrame = pd.DataFrame(centers, index=labels, columns=X.columns)
    centersFrame['cluster'] = labels
    plt.figure(figsize=(10, 5))
    plt.title(title)
    parallel_coordinates(centersFrame, 'cluster', marker='o')
    plt.legend(labels, loc='best')

def agglom_clustering(X, k=2, simMethod='cosine', linkMethod='ward'):
    model= AgglomerativeClustering(n_clusters=k, affinity=simMethod,
linkage=linkMethod)
    model.fit(X)
    return (model, model.labels_)

def visualise_dendrogram(X, linkageMethod='single', xlabel='', ylabel=''):
    Z = linkage(X, linkageMethod)
    plt.figure(figsize=(10, 7))
    plt.title('Hierarchical Clustering Dendrogram')
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    dendrogram(Z, labels=X.index, leaf_rotation=90)

def plot_clusters(X, labels, title, xlabel, ylabel):
    clustersDF = pd.DataFrame([labels], columns=X.index)
```

```
    plt.plot(clustersDF.iloc[0], color='green', marker='x', linewidth=0,
markersize=10)
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.xticks(list(clustersDF.columns))
    plt.yticks(np.arange(len(set(labels))))


##############################################################################
######################### feature selection ##################################
##############################################################################

# a function that will allow us to pass the method for weighting, the method
for selection,
# and the number of features to retain;
# it also has the option to output the scores of top n features
def stat_univariate_fs(X, y, weight_method, selection_method, num_features,
scores_to_print=25):
    X_reduced = GenericUnivariateSelect(score_func=weight_method,
mode=selection_method,
                        param=num_features).fit(X, y)
    scores = pd.DataFrame(X_reduced.scores_)
    columns = pd.DataFrame(X.columns)
    features_scores = pd.concat([columns, scores], axis=1)
    features_scores.columns = ['Attribute', 'Weight']
    print("Top", scores_to_print, "features:")
    print(features_scores.nlargest(scores_to_print, 'Weight'))
    return X_reduced.transform(X)

# a function that will allow us to use a specific algorithm
# for weighting and the number of features to retain
# it also has the option to output the scores of top n features
def clf_univariate_fs(X, y, learner,
                              num_features, scores_to_print=25):
    learner = learner.fit(X, y)
    scores = None
    if 'feature_importances_' in learner.__dir__():
        scores = pd.DataFrame(learner.feature_importances_)
    elif 'coef_' in learner.__dir__():
        scores = pd.DataFrame([np.max(np.abs(x)) for x in learner.coef_.T])
    columns = pd.DataFrame(X.columns)
    features_scores = pd.concat([columns, scores], axis=1)
    features_scores.columns = ['Attribute', 'Weight']
    print("Top", scores_to_print, "features:")
    print(features_scores.nlargest(scores_to_print, 'Weight'))
    return SelectFromModel(learner, prefit=True).transform(X)

# a function that will allow us to use a specific algorithm
```

```
# for weighting and the number of features to retain from sequential selection
def sequential_fs(X, y, learner, num_features, direction='forward'):
    seq_selector = SequentialFeatureSelector(learner,
            n_features_to_select=num_features, direction=direction);
    X_reduced = seq_selector.fit_transform(X, y);
    print([feature for feature in seq_selector.get_feature_names_out()]);
    return X_reduced;

# a function that will allow us to use a specific algorithm
# for weighting and the number of features to retain from rfe selection
def rfe_fs(X, y, learner, step=2):
    rfe_selector = RFECV(learner, step=step);
    X_reduced = rfe_selector.fit_transform(X, y);
    # print selected features - those that have a rank of 1
    print([feature for feature in rfe_selector.get_feature_names_out()]);
    return X_reduced;
```