Honours Degree in Computing

COMP H4016 - Text Analysis

Submitted by: Disi Pepiq, B00138946

Submission date

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: _____Disi Pepiq_____

Dated: _____04/04/23____

(Printing your name here will be taken as a digital signature)

Overview

a)

- This dataset is British Airways Passenger Reviews (2016 - 2023) dataset which has two columns in total.
- One column is called reviews which contains the customer's reviews of the airline.
- The other column is a class label called stars which is the customers overall satisfaction with the airline's services, as measured on a scale of 1 to 5 stars.
- This dataset has 2500 texts for reviews and 2500 TEXTS for star column.
- The star column contains the 5 unique values such as '5', '3', '1', '9', '7'.
- Reviews contains 2498 unique values.

Business objectives

- Improving the customers star ratings according to their airline reviews
- Identifying a specific part of the airline that needs to be improved.
- Try to address the problem by replying to the customer's reviews.

Mining objectives

- Classification: which include Building a predictive model
    - Using a predictive model to classify the customers star rating based on their review.
- Visualisation which includes, bar charts, histograms, word clouds
    - For example, a pie chart showing positive and negative customer reviews.
    - Word Clouds that show the most used words in customer reviews
- Clustering: Putting customers in groups with similar related reviews or a similar star rating

Baseline Models

Counts

- Counts provide information on how often a word or token appears in a document.
- The cell entry of counts shows the precise number of times a phrase, feature or unique token appears in a particular document.

Normalised counts

- Normalised counts are shown by the cell entry: word count or total word count of a document.
- Normalised counts are calculated by dividing a document's raw word count by the total amount of words in the document.

Tfidf

- In TF-IDF, word wights are assigned by combining term frequency (TF) and inverse document frequency (IDF).
- The tfidf entries are used to determine the use of various qualities.
- By counting the word occurrences, this heuristic method captures global relations.

Dimensionalities

Based on the total number of unique words in corpus, all three matrices have the same dimensionality, but the weighting scheme causes the TF-IDF to be sparser.

I chose the Decision tree classifier algorithm for text mining.

- The Decision tree algorithm is a machine learning nonparametric supervised approach used for classification.
- The objective of the decision tree is to build a model that can make predictions of the target variable value, by learning basic decision rules drawn from data elements. (scikit-learn, 2024)
- This algorithm is capable of handling text data, when linked with suitable text vectorisation methods such as TF-IDF.
- Decision tree algorithm is very easy to implement and not too hard to understand.
- It is easy to add new scenarios and make changes with the tree algorithm. (Grigore, 2023)

Count matrix Performance.

	precision	recall	f1-score	support
1	0.06	0.05	0.05	227
3	0.45	0.51	0.48	1137
5	0.28	0.27	0.27	682
7	0.10	0.08	0.09	227
9	0.09	0.08	0.09	227
accuracy			0.32	2500
macro avg	0.20	0.20	0.20	2500
weighted avg	0.31	0.32	0.31	2500

- Overall, the count matrix has a 32 percent accuracy.
- It has a 20 percent precision and 20 percent recall.

Normalised count matrix Performance.

	precision	recall	f1-score	support
1	0.05	0.04	0.04	227
3	0.44	0.47	0.46	1137
5	0.26	0.26	0.26	682
7	0.09	0.07	0.08	227
9	0.08	0.08	0.08	227
accuracy			0.30	2500
macro avg	0.18	0.18	0.18	2500
weighted avg	0.29	0.30	0.30	2500

- Overall, the normalised count matrix has a 30 percent accuracy.
- It has an 18 percent precision and 18 percent recall.

tfidf matrix Performance.

	precision	recall	f1-score	support
1	0.10	0.08	0.09	227
3	0.45	0.48	0.47	1137
5	0.25	0.24	0.25	682
7	0.11	0.11	0.11	227
9	0.09	0.08	0.08	227
accuracy			0.31	2500
macro avg	0.20	0.20	0.20	2500
weighted avg	0.30	0.31	0.31	2500

- Overall, the count matrix has a 31 percent accuracy.
- It has a 20 percent precision and 20 percent recall.

In all three matrices label 3 perform better than the other labels relation to precision, recall, and F1-score. In all three matrices the performance is quite similar with small differences, but the Counts Matrix has the nest performance.

Data Understanding

f) Derive word/token statistics for each category

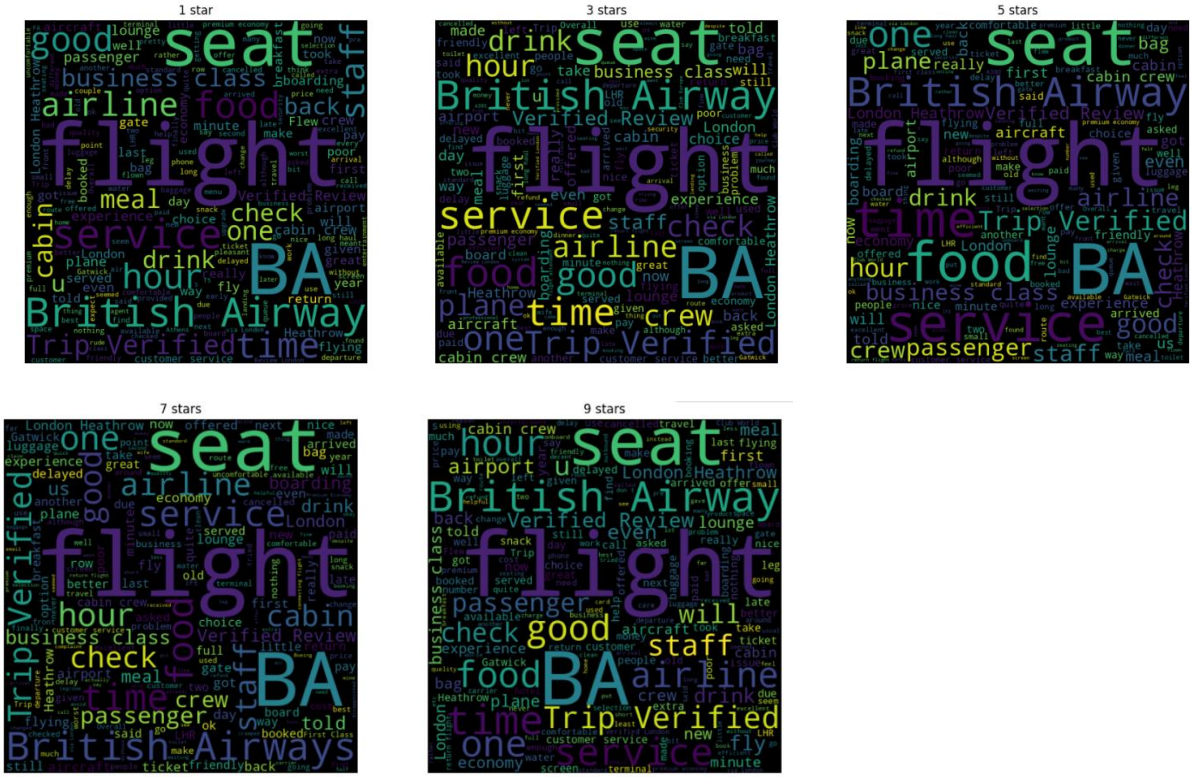
=== 10 most frequent tokens in 1 star ===
Frequency of "." is: 0.05017566197448775
Frequency of "the" is: 0.0360283888613803
Frequency of "to" is: 0.030534531135790244
Frequency of "," is: 0.027351394685341066
Frequency of "and" is: 0.026431821932989083
Frequency of "a" is: 0.020277758128787342
Frequency of "was" is: 0.0182971398929523
Frequency of "I" is: 0.017754827756949847
Frequency of "of" is: 0.012543915493621938
Frequency of "in" is: 0.01181297304944472
=== 10 most frequent tokens in 3 stars ===
Frequency of "." is: 0.05194715106895611
Frequency of "the" is: 0.03814760519610246
Frequency of "to" is: 0.030105577624757792
Frequency of "and" is: 0.026826798126147456
Frequency of "," is: 0.026026757430435768
Frequency of "a" is: 0.019792914321653355
Frequency of "was" is: 0.019483071971291024
Frequency of "I" is: 0.017189313676071384
Frequency of "of" is: 0.01250005780640865
Frequency of "in" is: 0.011626024907625359
=== 10 most frequent tokens in 5 stars ===
Frequency of "." is: 0.05064283474443399
Frequency of "the" is: 0.036523988711194735
Frequency of "to" is: 0.030072122922546254
Frequency of "and" is: 0.026944183129507682
Frequency of "," is: 0.02524302289118846
Frequency of "a" is: 0.019410473502665414
Frequency of "was" is: 0.019151771715271246
Frequency of "I" is: 0.01861084979617435
Frequency of "of" is: 0.013107557227971151
Frequency of "in" is: 0.011861084979617435
=== 10 most frequent tokens in 7 stars ===
Frequency of "." is: 0.05144087686612306
Frequency of "the" is: 0.03866534394734932
Frequency of "to" is: 0.03174526361634687
Frequency of "and" is: 0.026059183624089623
Frequency of "," is: 0.025938203198722448
Frequency of "was" is: 0.020711848822860462
Frequency of "a" is: 0.018703573761765346
Frequency of "I" is: 0.01843741682595756
Frequency of "of" is: 0.011686709090469162
Frequency of "in" is: 0.011493140409881681
=== 10 most frequent tokens in 9 stars ===
Frequency of "." is: 0.05213897423776885
Frequency of "the" is: 0.035665327345781137
Frequency of "to" is: 0.031127393051288112
Frequency of "and" is: 0.026896714724651383
Frequency of "," is: 0.023729614748286457
Frequency of "a" is: 0.01945166627274876
Frequency of "I" is: 0.018742614039234225
Frequency of "was" is: 0.01845899314582841

Frequency of "of" is: 0.012668399905459702
Frequency of "in" is: 0.011368470810683054

- This shows the frequency of tokens of each star rating.
- Overall, the highest frequency is the full stop “.”

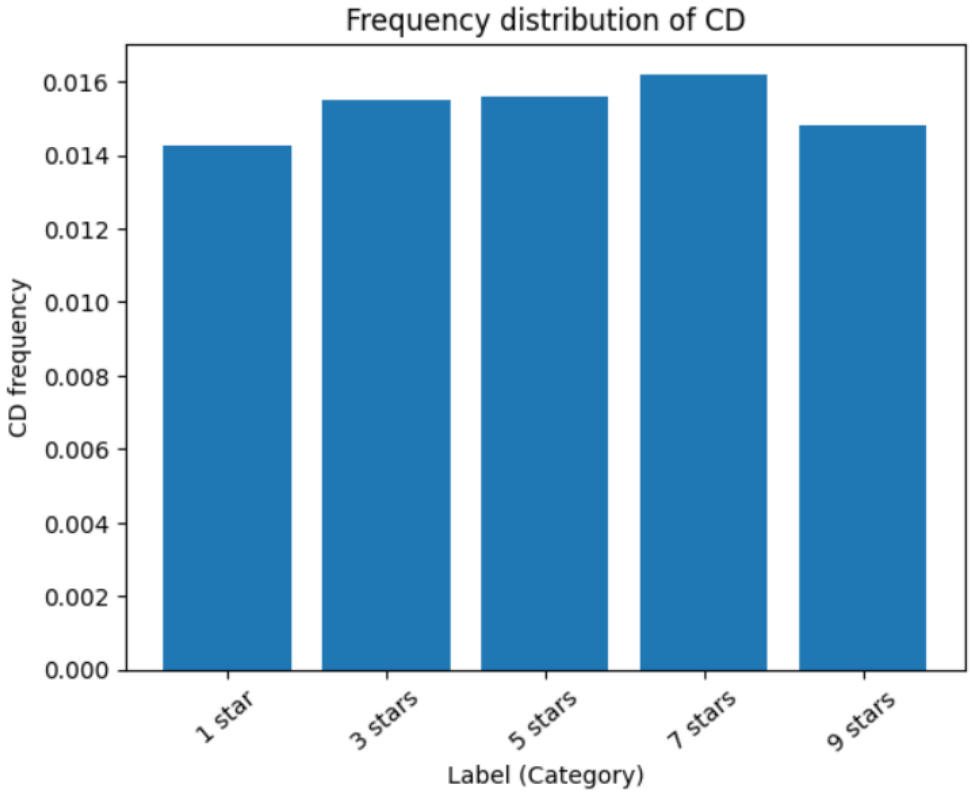
g) Visualisations techniques

Word clouds.



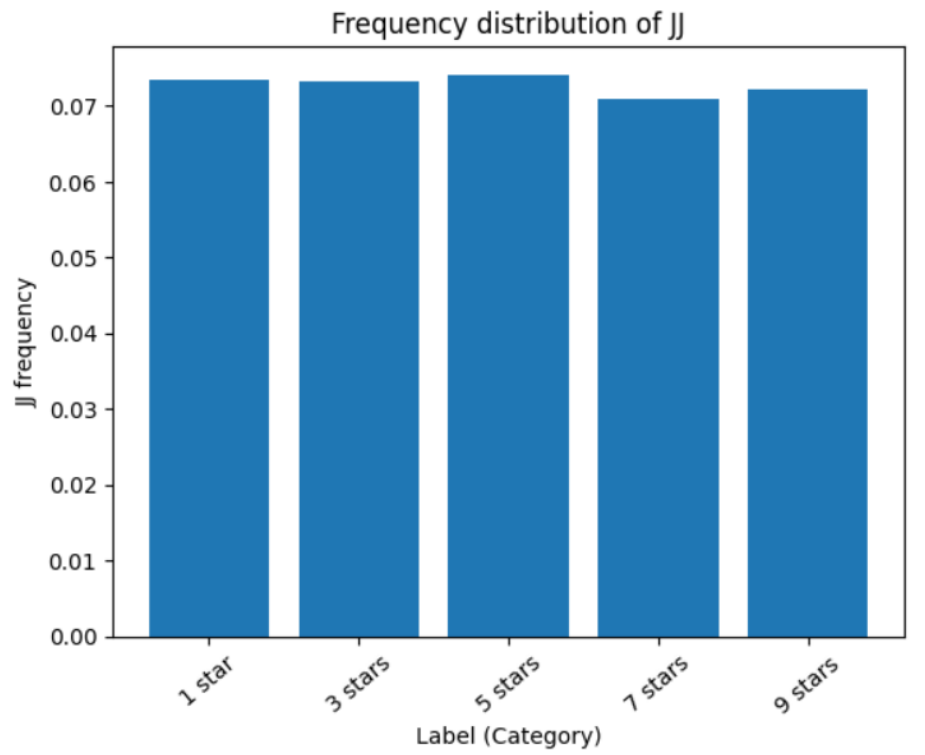
This shows the most occurring word associated with each star rating in the British Airways Passenger Reviews dataset.

Frequency of CD (Cardinal number)



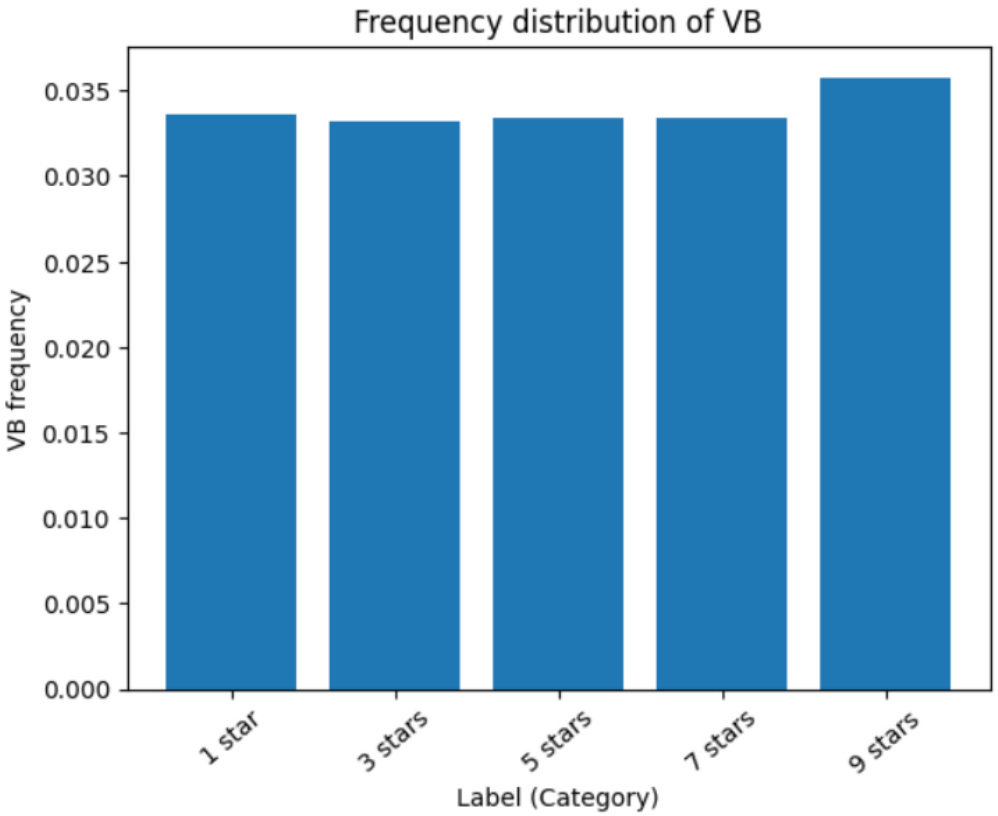
- This bar chart shows the occurrence of Cardinal numbers in the dataset.
- The 7 stars rating has the highest number of Cardinal numbers in the dataset.

Frequency of JJ (Adjectives)



- This bar chart shows the occurrence of Adjectives in the dataset.
- The 5 stars rating has the highest number of Adjectives in the dataset.

Frequency of VB (verb)



- This bar chart shows the occurrence of Verbs in the dataset.
- The 9 stars rating has the highest number of verbs in the dataset.

h) I chose the K means algorithm for clustering.

How does this algorithm work?

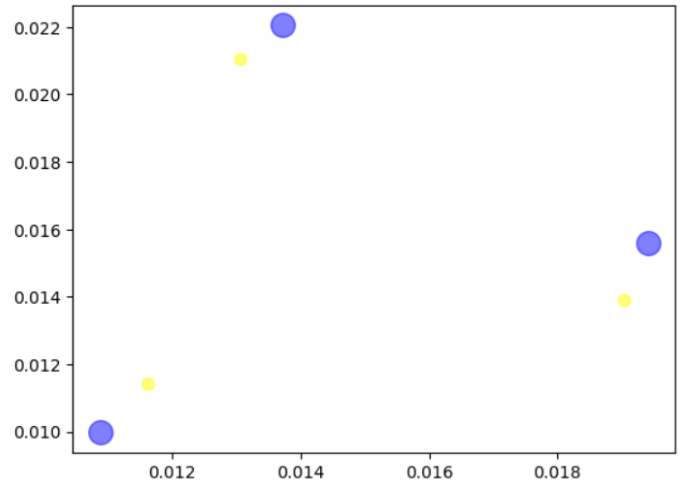
- A cluster mean is first represented by each of the k randomly chosen rows from the data set.
- Based on how similar an object is to the clusters mean, the object for each remaining object is allocated to the cluster where it is most similar.
- For each cluster a new mean is calculated when all objects have been assigned to the cluster.
- This new mean is now used to compare all rows in the database, and once more, each object is assigned to cluster where it's most similar.
- After every object has been redistributed again, the procedure is repeated, and a new mean for the cluster is calculated.
- I have tried the range 3 for kmeans random and kmeans++

```
model_r, cluster_labels_r, cluster_centers_r = tmu.k_means_clustering(
tfidf_matrix, 3, 'random')
print(list(y))
print(cluster_labels_r)
```

```
model_p, cluster_labels_p, cluster_centers_p = tmu.k_means_clustering(
tfidf_matrix, 3, 'k-means++')
print(list(y))
print(cluster_labels_p)
```

Results

```
KMeans Random - ARI score: -2.5711570468305967e-05
KMeans++ - ARI score: 0.0002991075292248948
KMeans Random - FMI score: 0.32712225123421784
KMeans++ - FMI score: 0.3270072936967492
```



- I have tried the kmeans random clustering and the kmeans ++ clustering and visualised it in a graph.
- I have computed the kmeans random and the kmeans ++ clustering clustering with ARI and FMI

- kmeans ++ has a higher ARI score (0.) than the kMeans random.
- kmeans ++ the kMeans random have very similar score FMI scores.
- These scores suggest that the text data did not cluster effectively using either the random or the K-Means++ approach.

Main Data Preparation:

a) Text preprocessing tasks

1.Basic Cleaning data

- I first tried to discover what cleaning operations I need for my dataset.
- So, I printed the head and tail of my dataset to see what cleaning operations I can use.

```
                                              reviews stars
0  ✅ Trip Verified |  I had the most fantastic BA...     5
1  ✅ Trip Verified |  Couldn't book in online. Ar...     3
2  ✅ Trip Verified | London Heathrow to Mumbai in...     3
3  ✅ Trip Verified | Keflavík, Iceland to London ...     5
4  ✅ Trip Verified | Terrible Experience with Bri...     5
                                             reviews stars
2490  Heathrow to Las Vegas with British Airways, an...     5
2491  Smooth take off and landing. Plane about 75%  ...     3
2492  Gatwick to Venice with British Airways. The pl...     3
2493  Round trip London to Luanda in British Airways...     1
2494  Not impressed with British Airways at all. We ...     9
2495  Roundtrip with British Airways from Bangkok to...     7
2496  Awful customer service. My wife and I have flo...     3
2497  I travelled from London to Sydney via Singapor...     5
2498  British Airways have just moved Cape Town flig...     3
2499  Bengaluru to Heathrow. My first long haul flig...     3
```

24:

✅ Trip Verified | Original flight was cancelled with no explanation. There was no representative to help in rebooking We had to call customer service (long distance call) ..

- In my dataset I found that there were two white spaces as seen at row 0, 1, and 24.
- In row 24 you see that there is paranthetical notes in the dataset (long distance call)

```
clean_operations = {
r'\s+' : ' ', ## any type of white space to be replaced with a single white space
r'\s{2,}' : ' ', ## 2 or more consecutive white spaces to be replaced with a single white space
r'(\(.+?\))+' : '', ## paranthetical notes to be replaced by empty string
```

- As seen above I used these cleaning operations to try to clean my data. I'm replacing any two or any type of white space with a single space, I'm replacing parenthetical notes that contain additional information with empty strings.

	reviews	stars
0	✅ Trip Verified | I had the most fantastic BA ...	5
1	✅ Trip Verified | Couldn’t book in online. Arr...	3
2	✅ Trip Verified | London Heathrow to Mumbai in...	3
3	✅ Trip Verified | Keflavík, Iceland to London ...	5
4	✅ Trip Verified | Terrible Experience with Bri...	5

	reviews	stars
2490	Heathrow to Las Vegas with British Airways, an...	5
2491	Smooth take off and landing. Plane about 75% f...	3
2492	Gatwick to Venice with British Airways. The pl...	3
2493	Round trip London to Luanda in British Airways...	1
2494	Not impressed with British Airways at all. We ...	9
2495	Roundtrip with British Airways from Bangkok to...	7
2496	Awful customer service. My wife and I have flo...	3
2497	I travelled from London to Sydney via Singapor...	5
2498	British Airways have just moved Cape Town flig...	3
2499	Bengaluru to Heathrow. My first long haul flig...	3

24:

✅ Trip Verified | Original flight was cancelled with no explanation. There was no representative to help in rebooking We had to call customer service..

- When applying these, the data looks more cleaning without any 2 or more white space and with no parenthetical notes.

Count Matrix performance.

	precision	recall	f1-score	support
1	0.07	0.06	0.07	227
3	0.46	0.50	0.48	1137
5	0.29	0.29	0.29	682
7	0.08	0.06	0.07	227
9	0.09	0.08	0.09	227
accuracy			0.33	2500
macro avg	0.20	0.20	0.20	2500
weighted avg	0.31	0.33	0.32	2500

Normalised count matrix Performance.

	precision	recall	f1-score	support
1	0.08	0.07	0.08	227
3	0.46	0.49	0.48	1137
5	0.29	0.31	0.30	682
7	0.09	0.08	0.09	227
9	0.08	0.07	0.07	227
accuracy			0.33	2500
macro avg	0.20	0.20	0.20	2500
weighted avg	0.31	0.33	0.32	2500

Tfidf matrix Performance.

	precision	recall	f1-score	support
1	0.10	0.10	0.10	227
3	0.44	0.47	0.46	1137
5	0.26	0.25	0.26	682
7	0.11	0.10	0.10	227
9	0.06	0.06	0.06	227
accuracy			0.31	2500
macro avg	0.19	0.19	0.19	2500
weighted avg	0.30	0.31	0.30	2500

The accuracy performance for the Count increased accuracy from 32 to 33%, the normalised count increased accuracy from 30 to 33% and for tfidf the accuracy did not change (31 percent), This task could be included into the final pipeline, where the count and the normalised count showed a boost in performance.

Dealing with synonyms, concepts/hypernyms, and word variations

```
repl_dict = {
    'London Heathrow': [r'\bLHR\b',r'\bHeathrow\b', ],
    'staff': [r'\bcrew\b', r'\bcabin crew\b'],
     'food': [r'\bmeal\b', r'\bsnack\b', r'\bbreakfast\b'],
    'flight': [r'\bfly\b', r'\bflying\b'],
    'BA': [r'\bBritish Airway\b']
}
```

- So, I tried this as my 2nd task dealing with synonyms, concepts/hypernyms, and word variations. In this task, first I used the word clouds to identify any synonyms, concepts or word variations in my dataset.
- The frequency of London Heathrow will be replaced with two variations of the term "LHR" and "Heathrow." . This is the same for the other words.
- \b ensures that the pattern matches whole words.
- The purpose of this task is to standardise terms by replacing synonyms, concepts, or variations with a common term.

Count Matrix performance.

	precision	recall	f1-score	support
1	0.08	0.07	0.07	227
3	0.47	0.51	0.49	1137
5	0.29	0.29	0.29	682
7	0.10	0.08	0.09	227
9	0.07	0.05	0.06	227
accuracy			0.33	2500
macro avg	0.20	0.20	0.20	2500
weighted avg	0.31	0.33	0.32	2500

Normalised count matrix Performance.

	precision	recall	f1-score	support
1	0.08	0.07	0.08	227
3	0.46	0.49	0.48	1137
5	0.29	0.31	0.30	682
7	0.09	0.08	0.09	227
9	0.08	0.07	0.07	227
accuracy			0.33	2500
macro avg	0.20	0.20	0.20	2500
weighted avg	0.31	0.33	0.32	2500

Tfidf matrix Performance.

	precision	recall	f1-score	support
1	0.10	0.10	0.10	227
3	0.44	0.47	0.46	1137
5	0.26	0.25	0.26	682
7	0.11	0.10	0.10	227
9	0.06	0.06	0.06	227
accuracy			0.31	2500
macro avg	0.19	0.19	0.19	2500
weighted avg	0.30	0.31	0.30	2500

After adding synonyms, concepts/hypernyms, and word variations, the accuracy performance for the Count, normalised count and tfidf did not show any significant change compared to clean data. This task could be included into the final pipeline.

Case transformation

- In this task I'm going to transform all to text to lowercase to see if it will boost the accuracy performance of my dataset or not.
- In my word clouds discovered that my dataset contains uppercase such as, BA, British Airways

Count Matrix performance.

	precision	recall	f1-score	support
1	0.10	0.09	0.09	227
3	0.45	0.49	0.47	1137
5	0.29	0.29	0.29	682
7	0.08	0.06	0.07	227
9	0.08	0.07	0.07	227
accuracy			0.32	2500
macro avg	0.20	0.20	0.20	2500
weighted avg	0.31	0.32	0.31	2500

Normalised count matrix Performance.

	precision	recall	f1-score	support
1	0.08	0.07	0.08	227
3	0.46	0.49	0.48	1137
5	0.29	0.31	0.30	682
7	0.09	0.08	0.09	227
9	0.08	0.07	0.07	227
accuracy			0.33	2500
macro avg	0.20	0.20	0.20	2500
weighted avg	0.31	0.33	0.32	2500

Tfidf matrix Performance.

	precision	recall	f1-score	support
1	0.10	0.10	0.10	227
3	0.44	0.47	0.46	1137
5	0.26	0.25	0.26	682
7	0.11	0.10	0.10	227
9	0.06	0.06	0.06	227
accuracy			0.31	2500
macro avg	0.19	0.19	0.19	2500
weighted avg	0.30	0.31	0.30	2500

After doing the Case transformation, the accuracy performance for the Count decreased from 33% to 32% and for normalised count and tfidf the accuracy performance did not change. I don't think this task could be included into the final pipeline.

b) Algorithms-based Feature selection tasks

Statistical Univariate Feature Selection

Using Chi2

Top 25 features:

	Attribute	Weight
11717	seniors	90.118943
2	#	63.989898
9231	marriage	50.066079
991	B787	40.254861
12658	sweets	40.254861
1090	BUD	40.052863
3566	Tobago	40.052863
8680	jaded	40.052863
2304	KUL	36.928161
11183	rep	36.632929
4633	bag	35.156403
1079	BKK	34.220340
2773	Oman	34.220340
8464	infants	34.220340
7319	featured	33.589353
3337	Spain	33.473453
3448	TAP	32.444934
952	Athens	31.928102
1193	Bombay	30.682044
12910	thru	30.540200
671	82A	30.039648
1542	DM	30.039648
2298	K2	30.039648
5663	comparing	30.039648
9404	minimalist	30.039648

Using anova

Top 25 features:

	Attribute	Weight
991	B787	10.232917
5663	comparing	7.595270
9404	minimalist	7.595270
6226	dedicated	6.756187
7319	featured	6.601564
4950	bothered	6.507100
4307	annoyed	6.480652
9899	operates	5.965096
9192	managers	5.835805
1344	Certainly	5.469409
3527	They're	5.376162
9723	noon	5.376162
12787	tells	5.376162
13603	visited	5.376162
2012	Gold	5.114769
12658	sweets	5.068684
376	2x2	5.041009
785	Added	5.041009
1065	BAD	5.041009
2335	Knowing	5.041009
2499	MS	5.041009
2655	NCL	5.041009
2694	News	5.041009
3271	Showers	5.041009
3366	States	5.041009

- The image above shows the top 25 features for the chi2 and anova with attribute and weight
- Theres only a small amount of concurrence between the Statistical Univariate Feature and with terms found in the word cloud.
    - Words found in Chi2 and Anova like B787, sweets can be aligned to terms in the word cloud.
    - Words like "seniors," "marriage," "minimalist, don't really correspond to the word cloud terms.

The performance of chi2

	precision	recall	f1-score	support
1	0.38	0.03	0.05	227
3	0.46	0.98	0.62	1137
5	0.29	0.01	0.03	682
7	0.81	0.06	0.11	227
9	0.00	0.00	0.00	227
accuracy			0.46	2500
macro avg	0.39	0.22	0.16	2500
weighted avg	0.40	0.46	0.31	2500

The performance of anova

	precision	recall	f1-score	support
1	0.44	0.07	0.12	227
3	0.46	0.99	0.63	1137
5	0.00	0.00	0.00	682
7	0.79	0.08	0.15	227
9	0.50	0.03	0.06	227
accuracy			0.47	2500
macro avg	0.44	0.23	0.19	2500
weighted avg	0.37	0.47	0.32	2500

- The accuracy performance of using chi2 and anova showed a massive boost in the accuracy compared to the other matrices.
- For Chi2 and Anova, the accuracy performance shown an increase from the data cleaning count (33%) to 46% for Chi2 and 47% for Anova.

Classification Algorithm based Univariate Feature Selection

- For this I'm using the count_matrix, and decision tree algorithm.

Top 25 features:

	Attribute	Weight
12843	the	0.016319
3912	a	0.011160
13711	we	0.010172
13	.	0.009827
13682	was	0.007844
12879	this	0.007649
11	,	0.007370
8362	in	0.007346
3517	The	0.007235
12833	than	0.007160
13870	with	0.007145
840	Airways	0.007110
8022	have	0.007073
12100	so	0.006917
13757	were	0.006682
1234	British	0.006555
992	BA	0.006530
7808	good	0.006396
9870	on	0.005911
4284	an	0.005870
12	-	0.005862
2450	London	0.005815
5138	cabin	0.005729
13569	very	0.005488
8668	it	0.005454

- The image above shows the top 25 features for Classification Algorithm with attribute and weight.
- Theres a small amount of concurrence between the Classification Algorithm with decision tree and with terms found in the word cloud.

- Words found in Classification Algorithm like London, cabin can be aligned to terms in the word cloud
- Words like " the," " we," "minimalist, don't really align to the word cloud terms.

The performance

	precision	recall	f1-score	support
1	0.11	0.11	0.11	227
3	0.47	0.48	0.48	1137
5	0.29	0.29	0.29	682
7	0.12	0.10	0.11	227
9	0.09	0.10	0.10	227
accuracy			0.33	2500
macro avg	0.22	0.22	0.22	2500
weighted avg	0.32	0.33	0.33	2500

- The performance did show any significant change compared to the clean data performance. This feature selection will not be used for the final pipeline.

- Overall, The Chi2 and ANOVA have shown a massive accuracy performance compared to Classification Algorithm based Univariate Feature Selection
- The Chi2 and ANOVA are the most predictive with its high accuracy (47%) and (46%).
- I will use the Statistical Univariate Feature Selection for the final pipeline because the performance was very good.

Hyperparameter tuning.

GridSearchCV

- For the hyperparameter tuning I'm using grid search cv with the decision tree algorithm
- Cross-validation is used by GridSearchCV to train and assess models for every possible combination of hyperparameters. The model with the best performance is chosen based on a certain score metric. This aids in maximising the decision tree algorithm's performance by identifying the hyperparameters that are most compatible with the dataset.

3 hyperparameters

- The decision tree's maximum depth is controlled by the max_depth
- The minimal amount of samples that must exist at a leaf node is determined by the min_samples_leaf. By making sure that every leaf node represents a specific quantity of data, it also helps to avoid overfitting.

- The minimum amount of samples needed to separate an internal node is determined by min_samples_split

```
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]
}
```

- The image above shows the different type of parameters I used.
- For the range I used 3,11 for maximum depth of the tree, 3,10 for minimum number of samples at a leaf node, 3,10 for min samples split and 0.01, 0.02, and 0.03 for minimum impurity decrease.
- I then tried to use different ranges for the hyperparameters such as changing the max_depth to 2, 10, min_samples_leaft to 2,9 and min_samples_split to 2,9 to seen if the GridSearch prefers to use lower ranges.

```
##GridSearchCV
dt_clf = DecisionTreeClassifier(random_state=1)
dt_params = {
'criterion': ['gini', 'entropy'],
'max_depth': range(2, 10),
'min_samples_leaf': range(2, 9),
'min_samples_split': range(2, 9),
'min_impurity_decrease': [0.01, 0.02, 0.03]
}
```

```
{'criterion': 'gini', 'max_depth': 2, 'min_impurity_decrease': 0.01, 'min_samples_leaf': 2, 'min_samples_split': 2}
DecisionTreeClassifier(max_depth=2, min_impurity_decrease=0.01,
                       min_samples_leaf=2, random_state=1)
```

- By changing the ranges to lower ranges for max_dept, min_samples_leaf and min_samples_split, I have seen that the GridSearch prefers to use lower ranges.

The performance

	precision	recall	f1-score	support
1	0.00	0.00	0.00	227
3	0.45	1.00	0.63	1137
5	0.00	0.00	0.00	682
7	0.00	0.00	0.00	227
9	0.00	0.00	0.00	227
accuracy			0.45	2500
macro avg	0.09	0.20	0.13	2500
weighted avg	0.21	0.45	0.28	2500

The accuracy performance of GridSearchCV using decision tree is 45%. Comparing this performance to the count_anova_matrix, the count_anova_matrix has an accuracy of 47 %. This means that there is a decrease from 47% to 45% .

Conclusion

a) Discuss which vectorization technique produced best results (2%).

The vectorization technique that produced best results was the count. The Count had an accuracy performance of 32% which had a better performance than other vectorization techniques. The count technique stands out for its robustness and simplicity, producing very good results.

b) Discuss which preparation techniques produced best results (2%).

The cleaning data produced the best results and dealing with synonyms, and word variations produced similar results. For the cleaning data, the accuacry performance for Count was 33%, 33% for the normalised and tfidf only 31%. By replacing any two or any type of white space with a single space and all parenthetical notes with empty strings made the data cleaner and increased the accuracy performance more.

c) Overall, discuss which model (the final pipeline) produced the best results (2%).

For the final pipeline, the text preprocessing task: data cleaning and Statistical Univariate Feature Selection produced the best results. So overall, for data cleaning, the accuracy performance for the Count increased accuracy from 32 to 33%, the normalised count increased from 30 to 33% and for tfidf the accuracy did not change (31 percent), the count and the normalised count showed a boost in performance. For Chi2 and Anova with the decision tree alogorimth , the accuracy performance shown an increase from the data cleaning count (33%) to 46% for Chi2 and 47% for Anova.

d) What features were the most predictive? Do they concur with those you identified during data understanding (2%)?

The Chi2 and Anova were the most predictive with an accuracy performance of 46 and 47 percent. There is a small amount of concurrence between the Statistical Univariate Feature and with terms found in the word cloud. Words found in Chi2 and Anova such as B787, sweets can be aligned to terms in the word cloud. But also, there's words such as "seniors," "marriage", that don't really correspond to the terms in the word cloud.

e) Discuss the limitations of your project and provide recommendations for future improvements (2%).

The limitations of the project were that the overall accuracy of the dataset using vectorization technique was low roughly around 30%. This suggests that there might be unresolved underlying issues with the dataset or issues with the classification task. For future improvements it is recommended to use different algorithms or other new preprocessing techniques that could increase the model performance

References

- Power, A. (2024) COMP H4016 - Text Analysis. Available at: https://vle-bn.tudublin.ie/course/view.php?id=262 (Accessed: 4 March 2024).

- Grigore (2023) *What is a good net promoter score? (2023 NPS benchmark)*, *Retently*. Available at: https://www.retently.com/blog/good-net-promoter-score/ (Accessed: 28 February 2024).

- scikit-learn (2024) *1.10. decision trees*, *scikit*. Available at: https://scikit-learn.org/stable/modules/tree.html (Accessed: 01 March 2024).

- Shakya, A. (2022) *Text classification using KNN*, *Medium*. Available at: https://medium.com/@ashins1997/text-classification-456513e18893 (Accessed: 02 March 2024).

Code

```
## 1
import pandas as pd, numpy as np
import nltk, re
from string import punctuation
from collections import Counter
from nltk.corpus import stopwords
from wordcloud import WordCloud



from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
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
from sklearn.feature_selection import chi2, f_classif
from gensim.models import Word2Vec
from sklearn.model_selection import GridSearchCV



from sklearn.svm import SVC

import matplotlib.pyplot as plt
%matplotlib inline


import warnings
warnings.filterwarnings('ignore')

import text_mining_utils as tmu

## 2
data = pd.read_csv('British_Airway.csv')
print("Shape of the DataFrame:", data.shape)
```

```
data.stars = data.stars.astype('string')
data.info()
```

```
stats = data.describe()
stats
```

```
duplicate_reviews = stats['reviews'].loc['top']
print(duplicate_reviews)
data[data.reviews == duplicate_reviews]
```

```
## keep only one the 2 duplicated reviews
data = data.drop_duplicates(subset=['reviews'])
print(data.shape)
new_stats = data.describe()
new_stats
```

```
y = data.stars
dt_clf = DecisionTreeClassifier(random_state=1)

count_matrix = tmu.build_count_matrix(data.reviews)
tf_matrix = tmu.build_tf_matrix(data.reviews)
tfidf_matrix = tmu.build_tfidf_matrix(data.reviews)


tmu.printClassifReport(dt_clf, count_matrix, y)
tmu.printClassifReport(dt_clf, tf_matrix, y)
tmu.printClassifReport(dt_clf, tfidf_matrix, y)
```

```
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
one_all = ' '.join(data.reviews[data.stars == '1'])
three_all = ' '.join(data.reviews[data.stars == '3'])
five_all = ' '.join(data.reviews[data.stars == '5'])
seven_all = ' '.join(data.reviews[data.stars == '7'])
nine_all = ' '.join(data.reviews[data.stars == '9'])

# output the first 100 characters from each
one_all[:100], three_all[:100], five_all[:100], seven_all[:100], nine_all[:100]
```

```
all_texts = [one_all, three_all, five_all, seven_all, nine_all]
tmu.plotPOSFreq(all_texts, 'CD', ['1 star', '3 stars', '5 stars', '7 stars', '9 stars'])

tmu.plotPOSFreq(all_texts, 'JJ', ['1 star', '3 stars', '5 stars', '7 stars', '9 stars'])
```

```
tmu.plotPOSFreq(all_texts, 'VB', ['1 star', '3 stars', '5 stars', '7 stars', '9 stars'])
```

```
one_docs = list(data.reviews[data.stars == '1'])
one_category = ' '.join(one_docs)

three_docs = list(data.reviews[data.stars == '3'])
three_category = ' '.join(three_docs)

five_docs = list(data.reviews[data.stars == '5'])
five_category = ' '.join(five_docs)

seven_docs = list(data.reviews[data.stars == '7'])
seven_category = ' '.join(seven_docs)

nine_docs = list(data.reviews[data.stars == '9'])
nine_category = ' '.join(nine_docs)

tmu.print_n_mostFrequent("1 star", one_category, 10)
tmu.print_n_mostFrequent("3 stars", three_category, 10)
tmu.print_n_mostFrequent("5 stars", five_category, 10)
tmu.print_n_mostFrequent("7 stars", seven_category, 10)
tmu.print_n_mostFrequent("9 stars", nine_category, 10)

tmu.generate_wordclouds([one_category, three_category, five_category],
['1 star', '3 stars', '5 stars'],
'black')

tmu.generate_wordclouds([seven_category, nine_category],
['7 stars', '9 stars'],
'black')


## compute cosine similarities
sims = cosine_similarity(tfidf_matrix)

## compute cosine distances
dists = cosine_distances(tfidf_matrix)
```

```
## do kmeans random clustering
model_r, cluster_labels_r, cluster_centers_r = tmu.k_means_clustering(
tfidf_matrix, 3, 'random')
print(list(y))
print(cluster_labels_r)
```

```
## do kmeans++ clustering
model_p, cluster_labels_p, cluster_centers_p = tmu.k_means_clustering(
tfidf_matrix, 3, 'k-means++')
print(list(y))
```

```
print(cluster_labels_p)
```

```
## visualise cluster centers from both approaches
plt.scatter(cluster_centers_r[:, 0], cluster_centers_r[:, 1], c='blue', s=200, alpha=0.5)
plt.scatter(cluster_centers_p[:, 0], cluster_centers_p[:, 1], c='yellow', s=50, alpha=0.5)

## Evaluate clusters using the ground truth (the label column): compute the
## ARI and FMI scores for both k-means approaches: random and ++
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)))
```

```
##Discovering the clean_operations of the Dataset
print(data.head(5))
print(data.tail(10))
# Print row 24
print("24:")
print(data.iloc[24]['reviews'])
```

```
clean_operations = {
r'\s+' : ' ', ## any type of white space to be replaced with a single white space
r'\s{2,}' : ' ', ## 2 or more consecutive white spaces to be replaced with a single white space
r'(\(.+?\))+' : '', ## paranthetical notes to be replaced by empty string


}
clean_data = data.copy()
clean_data.reviews = clean_data.reviews.apply(tmu.clean_doc, clean_operations=clean_operations)
print(clean_data.head(5))
print(clean_data.tail(10))
# Print row 24
print("24:")
print(clean_data.iloc[24]['reviews'])

dt_clf = DecisionTreeClassifier(random_state=1)
y = clean_data.stars
```

```
## generate new matrices: they should have fewer columns
clean_count_matrix = tmu.build_count_matrix(clean_data.reviews)
clean_tf_matrix = tmu.build_tf_matrix(clean_data.reviews)
clean_tfidf_matrix = tmu.build_tfidf_matrix(clean_data.reviews)
clean_tfidf_matrix.head()

## generate new models and evaluate them
tmu.printClassifReport(dt_clf, clean_count_matrix, y)
tmu.printClassifReport(dt_clf, clean_tf_matrix, y)
tmu.printClassifReport(dt_clf, clean_tfidf_matrix, y)
```

```
## create a dictionary for synonyms, variations and more general concepts
## the key is the replacement, the values are lists of regex that capture
## the strings to be replaced
## NOTE: below are only some examples, but you should elaborate
repl_dict = {
   'London Heathrow': [r'\bLHR\b',r'\bHeathrow\b', ],
   'staff': [r'\bcrew\b', r'\bcabin crew\b'],
    'food': [r'\bmeal\b', r'\bsnack\b', r'\bbreakfast\b'],
   'flight': [r'\bfly\b', r'\bflying\b'],
   'BA': [r'\bBritish Airway\b']
}
## apply the improve_bow function to all texts/documents from clean data
docs = clean_data.reviews
docs_repl = docs.apply(tmu.improve_bow, repl_dict=repl_dict)
## generate new matrices
# 1. count-based
count_matrix = tmu.build_count_matrix(docs_repl)
# 2. normalised count-based
tf_matrix = tmu.build_tf_matrix(docs_repl)
# 3. tfidf-based
tfidf_matrix = tmu.build_tfidf_matrix(docs_repl)
## output a row from each; they should have the same number of columns,
## but different cell entries
print(count_matrix.head(1))
print(tf_matrix.head(1))
tfidf_matrix.head(1)

## generate new models and evaluate them
tmu.printClassifReport(dt_clf, count_matrix, y)
tmu.printClassifReport(dt_clf, clean_tf_matrix, y)
tmu.printClassifReport(dt_clf, clean_tfidf_matrix, y)

## apply the lower function to all texts/documents from above data
lower_docs = docs_repl.apply(str.lower)
## generate new matrices
# 1. count-based
```

```
count_matrix = tmu.build_count_matrix(lower_docs)
# 2. normalised count-based
tf_matrix = tmu.build_tf_matrix(lower_docs)
# 3. tfidf-based
tfidf_matrix = tmu.build_tfidf_matrix(lower_docs)
## output a row from each; they should have the same number of columns,
## but different cell entries
print(count_matrix.head(1))
print(tf_matrix.head(1))
tfidf_matrix.head(1)

## generate new models and evaluate them
tmu.printClassifReport(dt_clf, count_matrix, y)
tmu.printClassifReport(dt_clf, clean_tf_matrix, y)
tmu.printClassifReport(dt_clf, clean_tfidf_matrix, y)
```

```
## 1. remove universal stop words
nltk.download('stopwords')
univ_sw = nltk.corpus.stopwords.words('english')
print(univ_sw)
```

```
nltk.download('punkt')
## apply the remove_sw_punct function to all texts/documents from above data
no_univsw_docs = lower_docs.apply(tmu.remove_sw_punct, to_remove=univ_sw)
## generate new matrices
# 1. count-based
count_matrix = tmu.build_count_matrix(no_univsw_docs)
# 2. normalised count-based
tf_matrix = tmu.build_tf_matrix(no_univsw_docs)
# 3. tfidf-based
tfidf_matrix = tmu.build_tfidf_matrix(no_univsw_docs)
## output a row from each; they should have the same number of columns,
## but different cell entries
print(count_matrix.head(1))
print(tf_matrix.head(1))
tfidf_matrix.head(1)
## generate new models and evaluate them
tmu.printClassifReport(dt_clf, count_matrix, y)
tmu.printClassifReport(dt_clf, clean_tf_matrix, y)
tmu.printClassifReport(dt_clf, clean_tfidf_matrix, y)
```

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

```
# using anova
count_anova_matrix = tmu.stat_univariate_fs(clean_count_matrix, y, weight_method=f_classif,
selection_method='k_best',
num_features=25, scores_to_print=25
)
```

```
## create the models and get performance
tmu.printClassifReport(dt_clf, count_chi2_matrix, y)
tmu.printClassifReport(dt_clf, count_anova_matrix, y)
```

```
dt_clf = DecisionTreeClassifier(random_state=1)
y = clean_data.stars
tree_weighted_matrix = tmu.clf_univariate_fs(clean_count_matrix, y, dt_clf,
num_features=25, scores_to_print=25)
tmu.printClassifReport(dt_clf, tree_weighted_matrix, y)
```

```
##GridSearchCV
dt_clf = DecisionTreeClassifier(random_state=1)
dt_params = {
'criterion': ['gini', 'entropy'],
'max_depth': range(2, 10),
'min_samples_leaf': range(2, 9),
'min_samples_split': range(2, 9),
'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_)
{'criterion': 'gini', 'max_depth': 2, 'min_impurity_decrease': 0.01, 'min_samples_leaf': 2,
'min_samples_split': 2}
DecisionTreeClassifier(max_depth=2, min_impurity_decrease=0.01,
 min_samples_leaf=2, min_samples_split=2, random_state=1)

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

```
optimal_dt = DecisionTreeClassifier(max_depth=3, min_impurity_decrease=0.01,
min_samples_leaf=3, min_samples_split=3, random_state=1)
```

```
tmu.printClassifReport(optimal_dt, count_anova_matrix, y)
```