****

**Honours Degree in Computing**

**Data Analytics Assessment:**

**Analyse a dataset**

**Submitted by: Olayiwola Animashaun, B00103457**

**Submission date**

# Table of Contents . . .

# Overview

Marvel Comics is a publishing imprint from Marvel Entertainment, this imprint deals with publishing stories of comic book characters whose rights are owned by Marvel Entertainment themselves (MARVEL CORPORATE INFORMATION, 2020). DC Comics is similar as it is also comic publishing imprint but is instead owned by DC Entertainment, and it publishes its own stories for its own set of characters (Advertising, 2020). Finally, the NBA is the National Basketball Association for North America, it hosts a professional basketball league that is comprised of 30 teams between the USA and Canada. It is thought to be the premier league for men’s basketball in the world (The Editors of Encyclopaedia Britannica, 2022).

## Business Objective

## One of the main business objectives will be to cross-reference the most popular genres of comic book between DC & Marvel Comics.

* Another business objective will be to find out the most popular comic book character for both DC Comics & Marvel Comics
* For the NBA, the main business objective will be to find the most popular player over the past three seasons
* Also, the most popular team over the past three seasons will also be noted.

## Data Mining Objectives

* Scrape thirty articles/documents regarding DC Comics to find most popular genre/character
* Scrape thirty articles/documents regarding Marvel Comics to find most popular genre/character
* Scrape thirty articles/documents regarding NBA to find most popular team/player
* Build corpus with scraped data
* Create baseline model
* Create wordcloud for each topic to prove business objectives

## Scraping & Sourcing

The function seen in Figure 1, will be used to retrieve text data from the sourced websites. It uses the BeautifulSoup python library which pulls data out of html and xml files and stores it in an array. Only the text data is needed and all the of the resulting strings are joined together to return a large string output.

Graphical user interface, text

Description automatically generated

Figure 1. Function To Retrieve Text Data

Text, letter

Description automatically generated

Figure 2. URLs for DC Comics

Text, letter

Description automatically generated

Figure 3. URLs For Marvel Comics

Text, letter

Description automatically generated

Figure 4. URLs For The NBA

## Relevant HTML Elements Data

After all the websites in each array have been sourced, the relevant html data has to be selected. For this section, only text data from the ‘h1’ and ‘p’ html elements will be selected. These html elements contain most of the relevant text information that is required for the business objectives.

Text

Description automatically generated

Figure 5. dc\_docs Variable

Text

Description automatically generated

Figure 6. marv\_docs Variable

Text

Description automatically generated

Figure 7. nba\_docs Variable

## Building The Corpus

The corpus is built by using the Pandas python library’s DataFrame capabilities. The Pandas DataFrame is ‘Two-dimensional, size-mutable, potentially heterogeneous tabular data’ (pandas, 2022). As seen in Figure 8, all the text from the arrays are added into one variable and a corresponding variable contain labels that apply to each of the previous arrays. In Figure 9, the corpus is built using a function that takes in the all\_docs variable and the all\_labels variable and returns a dataframe with two separate columns. One column contains all the documents text and one contains the corresponding labels.

Text

Description automatically generated

Figure 8. all\_docs Variable & all\_labels Variables

Graphical user interface, text, application, email

Description automatically generated

Figure 9. Corpus

# Baseline Models

## Matrices & Vectorisation Techniques

For the count matrix that we can see in Figure 10, the values in the cells are meant to represent how may times an attribute occurs in a given document. In the figure, we can see that articles of speech such as ‘for’ and ‘the’ seem to appear in almost all of these documents at high rates. In Figure 11, we can see the normalised count matrix. The normalised count matrix cell values are calculated from a simple formula, which is the number of times a word appears in a document divided by the total number of words in a document. Finally in Figure 12, we get the TFIDF matrix which represents how much weight an attribute has in a given document. The higher a value is, the rarer that attribute is in the document. However, as we can see, when a cell has a value of 0, that indicates that the attribute doesn’t appear in any document. The matrices are made up of 90 rows and 22802 columns, this shows that there are 90 documents and 22802 unique attributes.

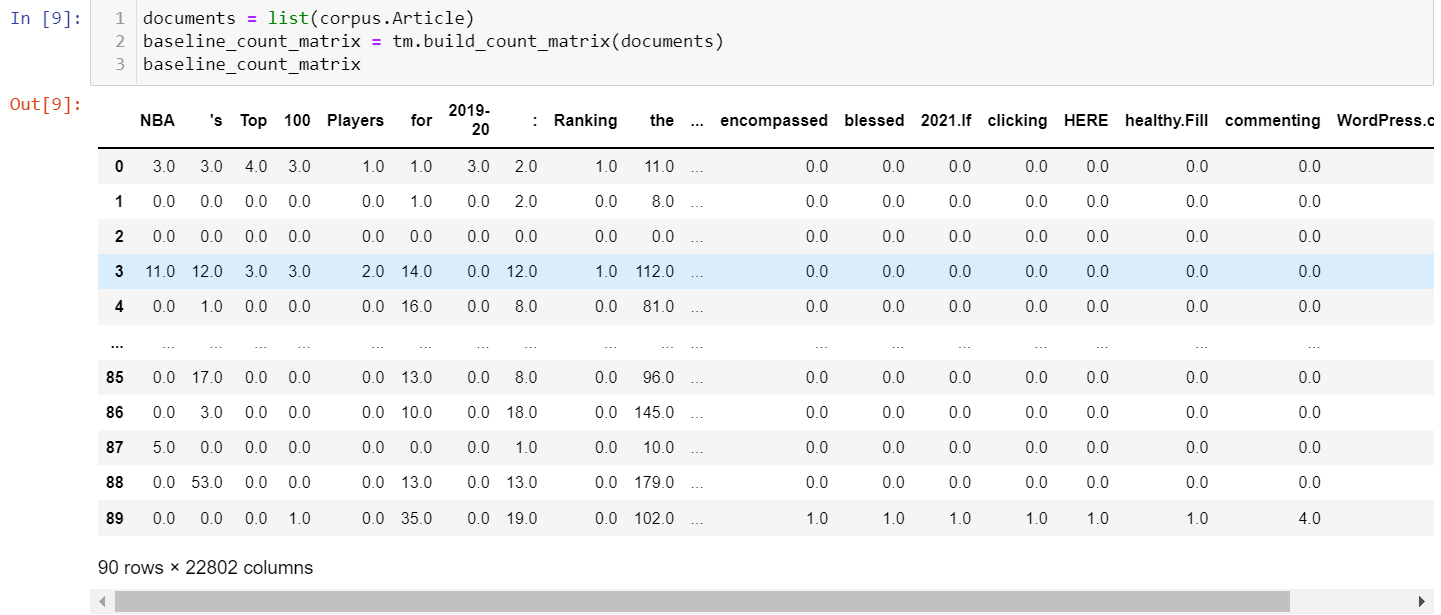


Figure 10. Count Matrix

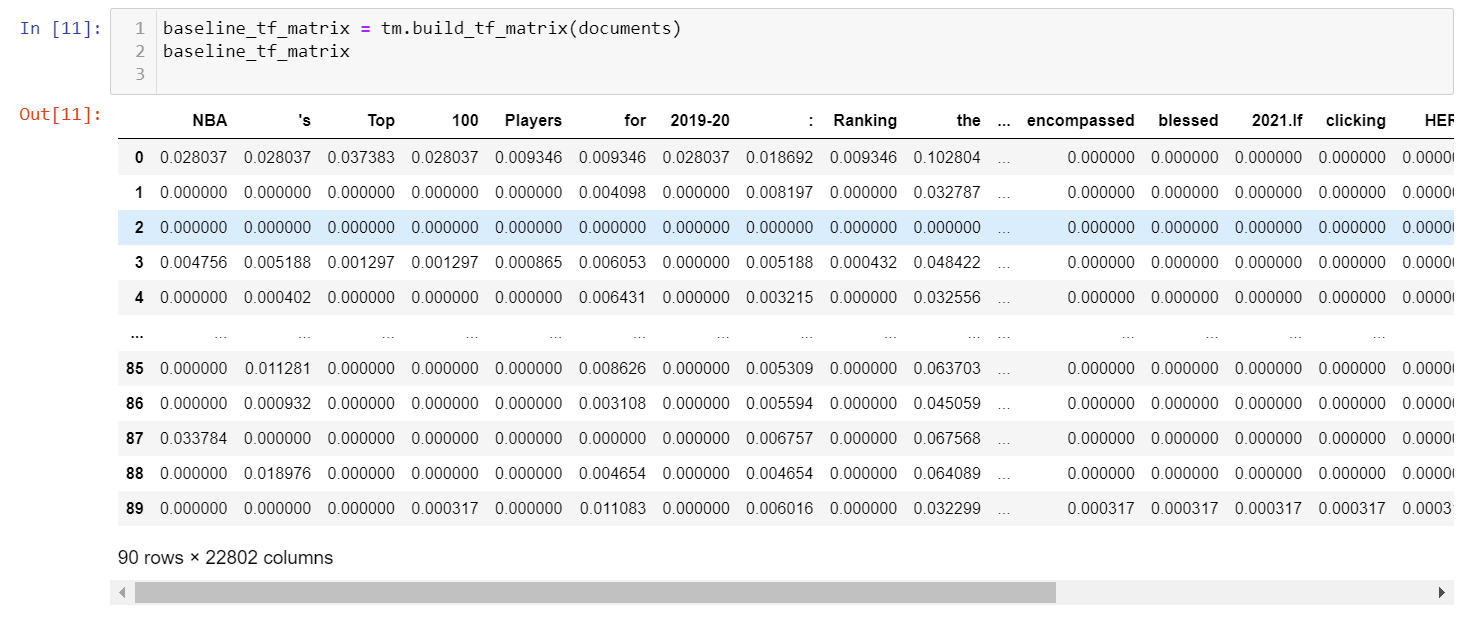


Figure 11. Normalised Count Matrix

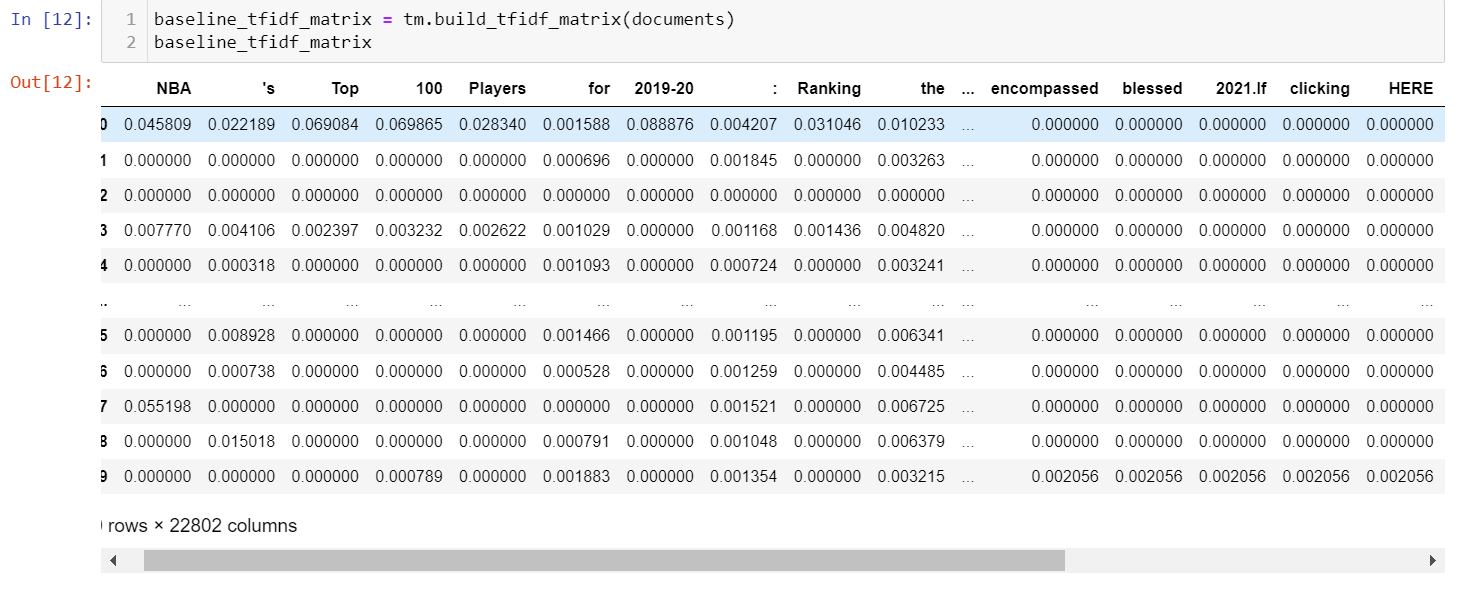


Figure 12. TFIDF Matrix

## Classification Algorithm

The Decision Tree Classifier class will be used as the classification algorithm for the baseline modelling. The Decision Tree Classifier is capable of performing multi-class classification on a dataset (scikit-learn developers, 2022). One of the main advantages of the Decision Tree Classifier it that it is capable of using different feature subsets and decision rules at different stages of classification (Du and Sun, 2022).

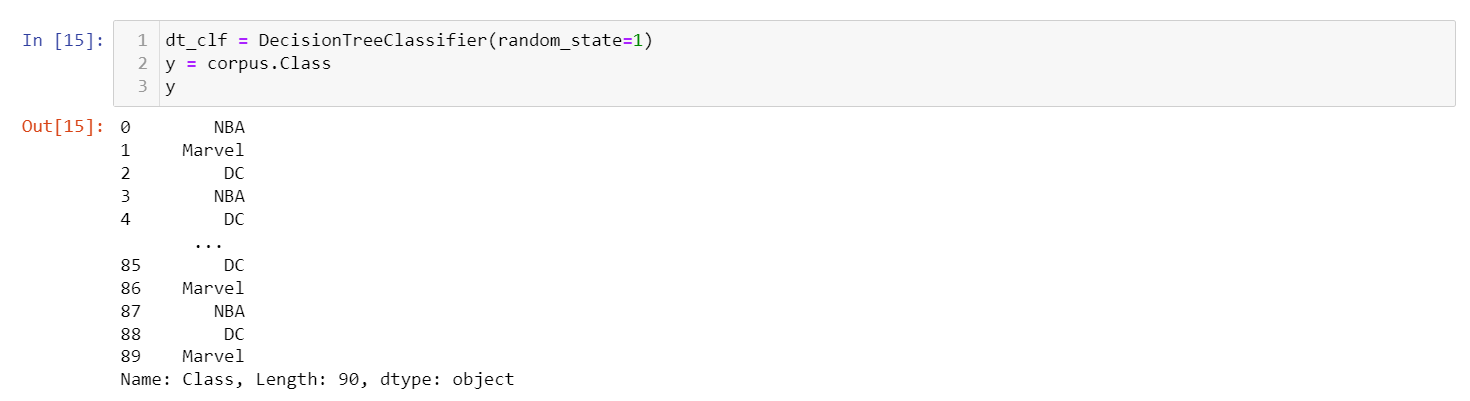


Figure 13. Decision Tree Classifier Class

## Cross Validation

After cross-validating the baseline models using the Decision Tree Classification algorithm, we get high values for all the matrices, indicating the data points in the models to be accurate.

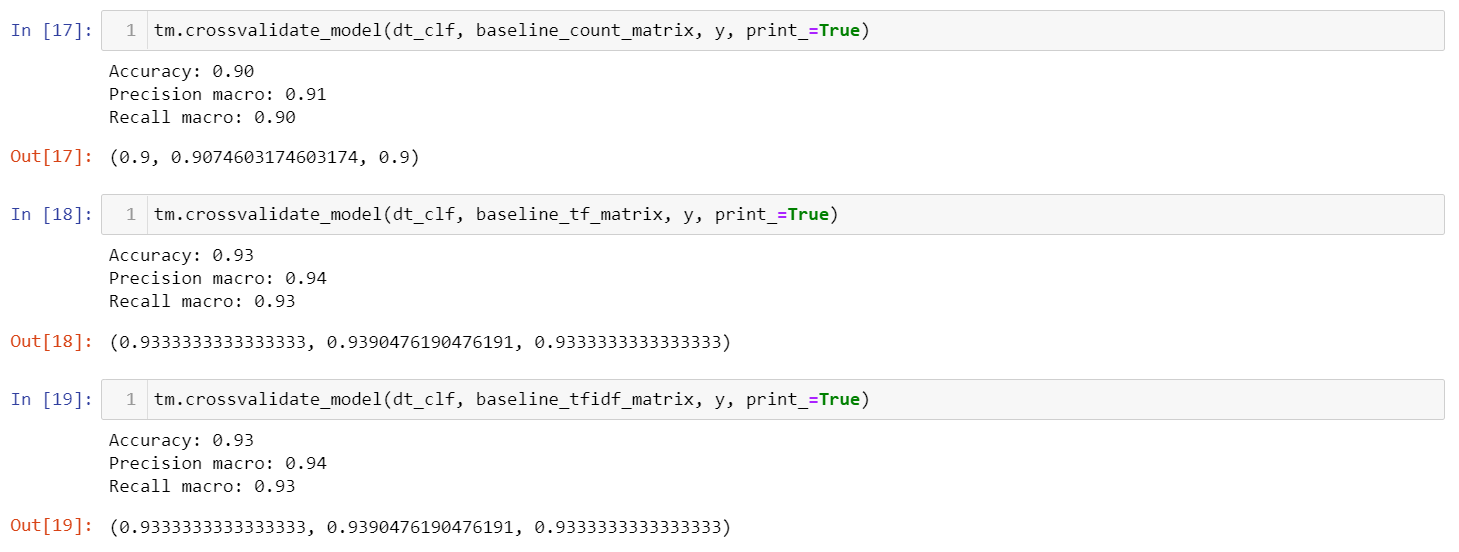


Figure 14. Cross Validation Values

# Data Understanding

## Word/Token Statistics

Figure 15 displays the most frequent tokens across the combined texts of each label from the corpus. We can see that the most common tokens across the texts are grammar tokens such as the comma, or definitive articles and conjunctive words such as ‘the’ and ‘of’. In Figure 16, it’s also evident that the most common tokens across all three texts are tokens such as exclamation marks and hashes.

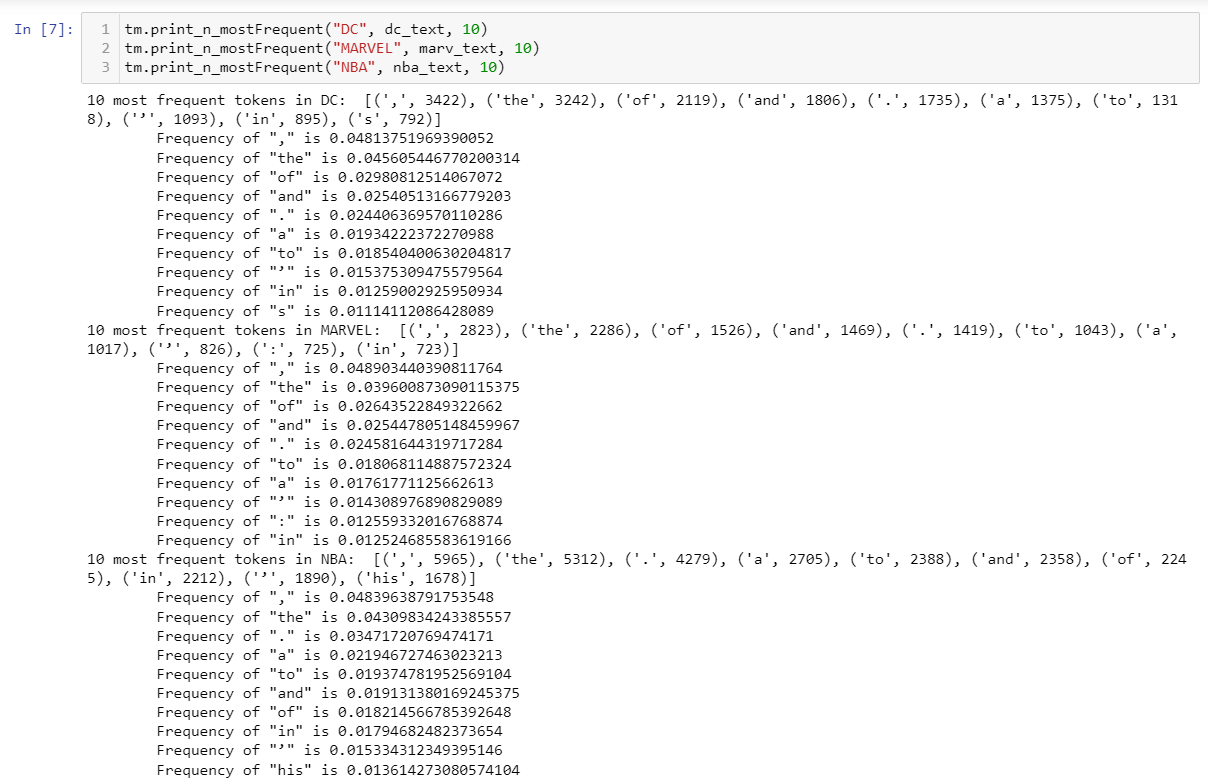


Figure 15. Frequent Tokens

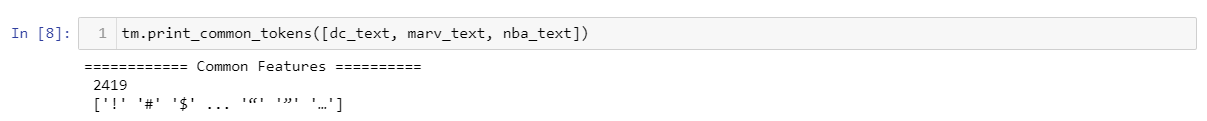


Figure 16. Common Tokens

## Visualisation Techniques

The frequency of Part Of Speech tags in the separate texts is visualised using bar charts as can be seen in Figure 17. All the texts have around the same frequency of adjectives in their texts, which is around 0.07 frequency. However, the NBA category edged out the DC and Marvel categories for nouns, sitting closer to 0.14 frequency, whilst the DC and Marvel categories sit around 0.12. WordClouds can also be used to show us the frequency of certain words in a text, the size of the word in the WordCloud indicates how many times the word appears in the text.

## 

Figure 17. Plotting POS Frequency

In Figure 18, we can see that the most frequently occurring terms in the DC category are words such as ‘character’, ‘Batman’, ‘comic’ and ‘Superman’. The letter ‘s’ also seems to be a major term in the text as well. There are also some notable synonyms in the text such as ‘comics’ and ‘comic books’ and also ‘heroes’ and ‘superheroes’.

Text

Description automatically generated

Figure 18. DC WorldCloud

In Figure 19, the most frequently occurring terms for the Marvel category seem to be ‘Marvel’, ‘one’, ‘comic’ and ‘new’. Again, the letter ‘s’ once again seems to be a frequently occurring letter. There are also synonyms such as ‘Marvel’ and ‘Marvel Comics’ or ‘comic’ and ‘comic book’.

Text

Description automatically generated

Figure 19. Marvel WordCloud

In Figure 20, we can see that some of the most frequently occurring terms in the NBA WordCloud are ‘season’, ‘team’, ‘player’, ‘game’ and ‘NBA’. Also, once again, we can see the letter ‘s’ being noted as one of the most frequently occurring letters. Some notable synonyms are ‘team’ and ‘franchise’, also ‘NBA’ and ‘league’.

Text

Description automatically generated

Figure 20. NBA WordCloud

## Clustering Algorithms

The goal of a clustering algorithm is to organise similar items in a dataset into groups. The Agglomerative Clustering class performs a hierarchal clustering using a bottom approach, each observation starts in its own cluster, and clusters are successively merged together (scikit-learn developers, 2022). In Figure 21, we’re able to see Agglomerative Clustering performed on the dataset with minimum linkage and two different measures (cosine & symmetric). The output of both of measures seem to not properly represent the three clusters of text documents. This most likely stems from the fact that a minimum amount of linkage is used for the clustering.

Graphical user interface, text, application, email

Description automatically generated

Figure 21. Minimum Linkage (Cosine & Symmetric)

For Figure 22, we can see that the clustering performed on the dataset is now using maximum linkage and two separate measures (cosine & symmetric). The output using a cosine measure seems to represent the dataset more accurately. More clusters from the dataset can be seen properly. However, the output using the symmetric measure is giving the same type of clustering as the first two attempts using minimum linkage.

Text

Description automatically generated

Figure 22. Maximum Linkage (Cosine & Symmetric)

# Main Data Preparation

# Text Pre-processing

## Data Cleaning

After completing the data understanding, it was clear that the dataset needed some initial cleaning. Many of the articles still contained html elements from the data scraping that was first conducted. The clean\_doc function was employed to clear these elements from the dataset. After the cleaning was completed, it can be seen that there is 22478 terms left. The count matrix now has an accuracy of 0.88, a precision macro of 0.89 and a recall macro of 0.88, this is a 0.02 drop in performance across the board. The normalised count matrix has an accuracy of 0.91, a precision macro of 0.91 and a recall macro of 0.91, this is a 0.02 drop in performance for the accuracy and recall macro from the baseline, and a 0.03 drop for the precision macro. Finally, the tfidf matrix also has an accuracy of 0.91, a precision macro of 0.91 and a recall macro of 0.91. Similar to the normalised count matrix, this is a 0.02 drop in performance from the baseline for the accuracy and recall macro and a 0.03 drop for the precision macro.

Table

Description automatically generated

Figure 23. Initial Cleaning Of Data

Graphical user interface, text, application

Description automatically generated

Figure 24. Clean Matrices

## Stop Words Removal

The visualisation of the texts that was conducted during the data understanding, showed the prevailing problem of stop words such as ‘s’ being considered as terms with an overwhelming influence in the text. To rectify this, we can use the list of universal stop words included in the nltk kit and run them through the remove\_sw function to rid the dataset of the stop words. After removing the stop words, it can be seen that there is 22294 terms left. After removal, the count matrix now has an accuracy of 0.92, a precision macro of 0.93 and a recall macro of 0.92. This is a 0.02 increase in performance from the baseline for all three. The normalised count matrix has an accuracy of 0.94, a precision macro of 0.95 and a recall macro of 0.94. This is a 0.01 increase in performance for all three from the baseline. Finally, the tfidf matrix has an accuracy of 0.94, a precision macro of 0.95 and a recall macro of 0.94. Again, this is a 0.01 increase in performance.

Text, letter

Description automatically generated

Figure 25. Universal Stop words

Text

Description automatically generated

Figure 26. Matrices After Removal

In an effort to be more precise, custom stop words can also be made for removal. For this, I gathered the stop words from the token statistics and visualisation that was performed in the data understanding section, once again, running them through the remove\_sw function. After removal, there were 22425 terms still left. The count matrix accuracy was now 0.94, the precision macro was 0.95 and the recall macro was 0.94. This is a 0.04 increase in performance across the board. The normalised count matrix accuracy was at a 0.93, the precision macro was at 0.94 and the recall macro was at 0.93. There was no change in performance in comparison to the baseline normalised count matrix. The tfidf matrix had an accuracy of 0.93, a precision macro of 0.94 and a recall macro of 0.93. Just like the normalised count, there was no change in performance for the tfidf matrix from the baseline.

Text

Description automatically generated

Figure 27. Custom Stop Word Removal

## Improving The Bag Of Words

After performing the data understanding and visualising the frequency of some of the major terms in the dataset, it was apparent that a lot of the major terms in the texts were synonymous. Essentially, many of the terms shared the same meaning. So, it was important to generalise these terms to potentially make the dataset more refined. After improving the bag of words, it can be seen that there are now 22452 terms. Now the count matrix accuracy sits at a 0.91, the precision macro sits at 0.92 and the recall macro sits at 0.91. This is a 0.01 increase in performance from the baseline values. The normalised count matrix now sits at 0.90, the precision macro sits at 0.90 and the recall macro also sits at 0.90. This is actually a drop in performance from the normalised count matrix baseline, the accuracy and recall macro drop 0.03 and the precision macro drops by 0.04. The tfidf matrix exhibits the same outcome, all the values sit at 0.90 with an identical drop in performance as the normalised count matrix.

Text

Description automatically generated

Figure 28. Bag Of Words

After applying all three pre-processing tasks to the dataset, it would seem as if the stop words removal task produced the best results across the three. The ability to create custom stop words can also prove beneficial in improving the performance of the task. The stop word removal pre-processing task will most likely be included in the final pipeline.

# Algorithm-Based Feature Selection

## Univariate Feature Selection

The univariate feature selection is a feature selection technique that uses statistical methods to select the most relevant features (Power, 2022). After applying the technique to the tfidf matrix, there were 90 of the most relevant terms left. The accuracy sits at 0.92, the precision macro sits at 0.93 and the recall macro sits at 0.92. This is a 0.01 drop in performance from the baseline scores. The most relevant terms produced by the univariate selection seem to concur with the terms that were noted to be frequent in the data understanding section. Term like ‘DC’ and ‘Batman’ took up major space in visualisations like the WordCloud.

Graphical user interface, text, application

Description automatically generated

Figure 29. Univariate Feature Selection

Chart

Description automatically generated with low confidence

Figure 30. Univariate FS Results

## Recursive Feature Elimination

The RFE feature selection technique uses a classifier that assigns weights to features, features are selected b recursively considering smaller and smaller sets of features (Power, 2022). The technique is applied to the tfidf matrix and limited to 100 terms. The accuracy after cross validating is now 0.92, the precision macro is 0.93 and the recall macro is 0.92. Similar to the univariate feature selection, there is an overall 0.01 drop in performance. However, unlike the univariate feature selection, the top terms did not seem to match up with or predict any of the words that were present during the data understanding section.

Text

Description automatically generated

Figure 31. RFE

Both techniques seem to produce identical scores, however, the univariate feature selection technique will most likely be used in the final pipeline as it runs easier and also concurs with the terms previously seen in the data understanding section better.

# Build Classification Models

# Evaluation

In conclusion, after mining this data using Python libraries, I can say out of the three vectorisation techniques that were preformed, the normalised counts seemed to produce the best results. It scored higher than the standard count, however, it held identical scores to the tfidf values.

After conducting the data preparation, it was evident that the stop word removal technique produced the best results in comparison to the data cleaning and bag of words techniques. Both the universal stop word removal and custom stop word removal scored higher on all three matrices than both the data cleaning and bag of words.

The univariate feature selection seemed to concur with the terms that were predicted during the data understanding section. This is why it would most likely be included in the final pipeline.

# References

1. Marvel Entertainment. 2020. *MARVEL CORPORATE INFORMATION*. [online] Available at: <https://www.marvel.com/corporate/about> [Accessed 14 March 2022].
2. DC. 2020. *Advertising*. [online] Available at: <https://www.dccomics.com/advertising> [Accessed 14 March 2022].
3. The Editors of Encyclopaedia Britannica, 2022. *National Basketball Association | History & Facts*. [online] Encyclopaedia Britannica. Available at: <https://www.britannica.com/topic/National-Basketball-Association> [Accessed 22 March 2022].
4. pandas, 2022. *pandas.DataFrame — pandas 1.4.1 documentation*. [online] Pandas.pydata.org. Available at: <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html> [Accessed 18 March 2022].
5. scikit-learn developers, 2022. *1.10. Decision Trees*. [online] scikit-learn. Available at: <https://scikit-learn.org/stable/modules/tree.html> [Accessed 2 April 2022].
6. Du, C. and Sun, D., 2022. 4 - Object Classification Methods. *Computer Vision Technology for Food Quality Evaluation,*, [online] pp.81-107. Available at: <https://www.sciencedirect.com/science/article/pii/B9780123736420500077> [Accessed 3 April 2022].
7. scikit-learn developers, 2022. *2.3. Clustering*. [online] scikit-learn. Available at: <https://scikit-learn.org/stable/modules/clustering.html#hierarchical-clustering> [Accessed 2 April 2022].
8. Power, A., 2022. *Feature Selection and Advanced Word Embeddings*.

# Appendix

import re, pandas as pd, numpy as np, requests, bs4, matplotlib.pyplot as plt

import wordcloud, nltk

from collections import Counter

import warnings

warnings.filterwarnings('ignore')

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import cross\_validate

from sklearn.metrics import recall\_score, precision\_score, accuracy\_score

import wordcloud

import text\_mining\_utils as tm

from sklearn.cluster import KMeans

from sklearn.cluster import AgglomerativeClustering

from sklearn.feature\_selection import f\_classif

nltk.download('stopwords')

# ### Using a scraper, source 90 texts/documents/articles: 30 for each category; describe the process employed and state the source websites/pages

# In[123]:

##function to retrieve text data

def retrieve\_text\_data(url, elems):

## page: gets url

page = requests.get(url)

## page\_data: stores url

page\_data = page.text

## soup: stores all relevant url data and strips away html tags

## data: array to stores data

soup = bs4.BeautifulSoup(page\_data, "html.parser").body

data = []

## The for loop scans through all the relevant elements and adds them into the data array

for e in elems:

data += soup.find\_all(e)

## data: the get\_text function is used to retrieve the text content of all the relevant elements in the data array

data = [el.get\_text() for el in data]

## The resulting strings are joined together and a string is returned

return ''.join(data)

## This array holds 30 urls that are relevant to DC Comics

dc\_urls = [

"https://screenrant.com/best-dc-comic-books-2021-according-reddit/",

"https://www.cbr.com/dc-comics-best-2020/",

"https://screenrant.com/best-dc-miniseries-2019/",

"https://www.comicbookherald.com/best-dc-comics-of-2019/",

"https://www.comicbookherald.com/best-dc-comics-of-2018/",

"https://screenrant.com/best-dc-comics-heroes-of-all-time-according-to-ranker/",

"https://www.ranker.com/crowdranked-list/best-dc-comics-heroes",

"https://www.shortlist.com/lists/best-dc-characters-402104",

"https://www.cbr.com/comic-genres-matched-members-justice-league/",

"https://www.gamesradar.com/best-dc-comics-stories/",

"https://libguides.colum.edu/comicsgraphicnovels/Genre",

"https://www.dccomics.com/characters",

"https://www.insider.com/best-dc-comic-heroes-2019-1",

"https://www.comicbookherald.com/best-dc-comics-of-2017/",

"https://medium.com/@2ndHandCopy/the-best-comics-of-2016-part-2-5-3267af59e4d3",

"https://rarecomics.wordpress.com/top-50-dc-comics-of-2015/",

"http://www.multiversitycomics.com/news-columns/the-ten-best-dc-comic-books-right-now/",

"https://www.one37pm.com/culture/movies-tv/best-dc-comics",

"https://www.toynk.com/blogs/news/best-dc-comics",

"https://www.complex.com/pop-culture/the-best-dc-comics-of-all-time/",

"https://www.cbr.com/best-dc-comics-all-time/",

"https://www.comicbookherald.com/the-best-dc-comics-of-2021/",

"https://www.comicbookherald.com/the-best-100-dc-comics-since-crisis-on-infinite-earths-1985/",

"https://www.cheatsheet.com/entertainment/dc-comics-greatest-superheroes-of-all-time.html/",

"https://wegotthiscovered.com/comicbooks/the-best-dc-comics-heroes/",

"https://www.comicsbookcase.com/updates/best-comics-2022-dc-comics",

"https://culturefly.com/blogs/culture-blog/best-dc-comic-storylines",

"https://couchguysports.com/ranking-the-best-dc-comic-characters/",

"https://www.jelly.deals/best-dc-comics-new-readers-superman-batman",

"https://whatculture.com/comics/10-greatest-dc-superheroes-of-all-time",

]

## This array holds 30 urls that are relevant to Marvel Comics

marv\_urls = [

"https://www.comicsbookcase.com/updates/best-comics-2022-marvel",

"https://www.toynk.com/blogs/news/best-marvel-comics",

"https://www.comicbookherald.com/best-marvel-comics-of-2021/",

"https://screenrant.com/best-marvel-comic-books-2021/",

"https://www.cbr.com/marvel-comics-best-stories-releases-2020/"

"https://www.comicbookherald.com/best-marvel-comics-of-2020/",

"https://www.comicsbookcase.com/updates/best-comics-2020-marvel",

"https://weirdsciencemarvelcomics.com/2021/01/03/marvel-comics-best-of-2020-year-in-review/",

"https://www.comicbookherald.com/best-marvel-comics-of-2019/",

"https://superherojunky.com/top-10-marvel-comics-of-2019/",

"https://www.comicbookherald.com/best-marvel-comics-of-2018/",

"https://www.comicbookherald.com/the-best-marvel-comics-of-2017/",

"https://www.comicbookherald.com/the-best-marvel-comics-of-2016/",

"https://www.pastemagazine.com/comics/the-10-best-comics-marvel-currently-publishes-2016/",

"https://aminoapps.com/c/comics/page/blog/top-10-best-marvel-comics-of-2016/GMin\_u0NWrEZb5mJ4LoZNwRD4EEYVe",

"https://rarecomics.wordpress.com/top-50-marvel-comics-of-2015/",

"https://www.gamesradar.com/best-marvel-comics-stories/",

"https://www.wsj.com/articles/BL-SEB-85907",

"https://www.gamesradar.com/marvel-characters/",

"https://screenrant.com/best-marvel-comics-heroes-of-all-time-according-to-ranker/",

"https://www.ranker.com/crowdranked-list/top-marvel-comics-superheroes",

"https://www.toynk.com/blogs/news/best-marvel-characters",

"https://lemonly.com/blog/top-10-most-popular-marvel-movie-characters",

"https://fictionhorizon.com/20-best-marvel-characters-of-all-time/",

"https://www.telltalesonline.com/28598/popular-marvel-characters/",

"https://www.marvel.com/articles/culture-lifestyle/the-wider-world-of-marvel-genres",

"https://screenrant.com/best-marvel-comic-books-ever-ranker/",

"https://www.cbr.com/comic-genres-matched-members-avengers/",

"https://www.marvel.com/comics/discover/1278/top-25-comics",

"https://www.one37pm.com/culture/news/best-marvel-graphic-novels",

"https://www.quora.com/Who-is-the-most-popular-Marvel-superhero",

]

## This array holds 30 urls that are relevant to the NBA

nba\_urls = [

"https://www.nbcsports.com/washington/wizards/2022-ranking-top-20-nba-players-right-now",

"https://sportsnaut.com/best-nba-players-right-now/",

"https://www.si.com/nba/2021/09/23/ranking-best-nba-players-top-100-2022-kevin-durant-giannis-antetokounmpo-lebron-james",

"https://www.ranker.com/list/best-nba-players-2022/patrick-alexander",

"https://www.washingtonpost.com/sports/interactive/2021/nba-top-100-players-2022/",

"https://thegameday.com/41847/article/2021-nba-top-100-players-2022/",

"https://www.si.com/nba/2020/12/14/top-100-nba-players-2021-daily-cover",

"https://morningconsult.com/2021/10/18/nba-players-curry-durant-poll/",

"https://www.theringer.com/nba/2021/5/4/22416337/top-25-nba-player-ranking-lebron-james-nikola-jokic",

"https://www.complex.com/sports/best-nba-players-rankings",

"https://www.persources.com/ranking-the-top-20-nba-players-2021/",

"https://www.ranker.com/crowdranked-list/top-current-nba-players",

"https://thesixersense.com/2021/09/17/nba-top-100-players-2021-22/",

"https://www.statista.com/statistics/1266006/nba-top-shot-nft-most-popular-cards/",

"https://www.interbasket.net/news/espns-100-best-nba-players-2020-21-nba-season-nbarank-list/31636/",

"https://www.stadium-maps.com/facts/nba-teams-popularity.html"

"https://www.lineups.com/articles/top-10-nba-players-in-the-2019-2020-season-kawhi-leonard-at-1/",

"https://www.sportingnews.com/ca/nba/news/who-are-the-best-players-in-the-nba-entering-the-2020-21-season/4n84f58mc6sz157jdx4p2u712",

"https://www.washingtonpost.com/graphics/2020/sports/nba-top-players-2020-2021/",

"https://www.nbcsports.com/boston/celtics/nbas-top-100-players-2019-20-ranking-top-25",

"https://www.si.com/nba/2018/09/10/top-100-nba-players-2019-lebron-james-stephen-curry-dirk-nowitzki",

"https://www.sportingnews.com/in/nba/news/who-are-the-best-nba-players-entering-2019-20-season/13vm1p03wlnre14hecrk060vn9",

"https://bleacherreport.com/articles/2889335-bleacher-reports-top-100-player-rankings-from-the-2019-20-nba-season",

"https://www.insider.com/ranked-top-nba-players-right-now-2020-12",

"https://www.si.com/extra-mustard/2022/02/17/lebron-james-lakers-lead-lids-jersey-sales",

"https://www.sportskeeda.com/basketball/10-best-selling-nba-jerseys-2021-far",

"https://www.nbcsports.com/washington/wizards/2022-nba-power-rankings-utah-jazz-take-top-spot-after-hot-streak",

"https://wegrynenterprises.com/2021/10/12/report-ranking-the-most-popular-nba-teams/",

"https://bolavip.com/en/nba/The-25-NBA-teams-with-most-fans-20200423-0002.html",

"https://www.statista.com/statistics/240382/facebook-fans-of-national-basketball-association-teams/",

"https://www.infoplease.com/us/basketball/top-grossing-nba-teams"

]

# In[113]:

retrieve\_text\_data("https://www.popcornbanter.com/5-best-dc-comics-stories-of-all-time/",

['h1', 'p'])

# ### Select the relevant html elements data, describing what you have retained, what you have removed and why; use the developer tools to aid your decisions

# In[122]:

## dc\_docs: retrieves text data from headings labeled as heading 1 and paragraphs of urls

dc\_docs = [retrieve\_text\_data(url, ['h1', 'p']) for url

in dc\_urls]

print(len(dc\_docs))

dc\_docs

# In[124]:

## marv\_docs: retrieves text data from headings labeled as heading 1 and paragraphs of urls

marv\_docs = [retrieve\_text\_data(url, ['h1', 'p']) for url

in marv\_urls]

print(len(marv\_docs))

marv\_docs

# In[116]:

c = 1

for url in dc\_urls:

print(retrieve\_text\_data(url, ['h1', 'p']))

print('-------------------', c)

c = c+1

# In[117]:

## nba\_docs: retrieves text data from headings labeled as heading 1 and paragraphs of urls

nba\_docs = [retrieve\_text\_data(url, ['h1', 'p']) for url

in nba\_urls]

print(len(nba\_docs))

nba\_docs

# In[127]:

## all\_docs: contains all previous url arrays

all\_docs = dc\_docs + marv\_docs + nba\_docs

## all\_labels: contains a list of labels for all the previous url arrays

all\_labels = (['DC'] \* len(dc\_docs) +

['Marvel'] \* len(marv\_docs) +

['NBA'] \* len(nba\_docs))

all\_docs

# In[128]:

## prints all the labels

len(all\_docs), all\_labels

# ### Build the corpus and explain the corresponding process

# In[129]:

def build\_corpus(docs, labels):

corpus = np.array(docs)

corpus = pd.DataFrame({'Article': corpus, 'Class': labels})

corpus = corpus.sample(len(corpus))

return corpus

corpus = build\_corpus(all\_docs, all\_labels)

corpus

# In[130]:

corpus.to\_csv('corpus.csv', columns=['Article', 'Class'], index=False)

# ### Making the documents readable

# In[137]:

print(dc\_text)

# In[138]:

print(marv\_text)

# In[139]:

print(nba\_text)

# ### Derive 3 matrices using 3 vectorisation techniques: counts, normalised counts and tfidf. Discuss the dimensionality and the differences between them

# In[3]:

corpus = pd.read\_csv('corpus.csv')

corpus

# In[9]:

documents = list(corpus.Article)

baseline\_count\_matrix = tm.build\_count\_matrix(documents)

baseline\_count\_matrix

# In[11]:

baseline\_tf\_matrix = tm.build\_tf\_matrix(documents)

baseline\_tf\_matrix

# In[12]:

baseline\_tfidf\_matrix = tm.build\_tfidf\_matrix(documents)

baseline\_tfidf\_matrix

# ### Choose at least 1 classification algorithm for baseline modelling;

# In[15]:

dt\_clf = DecisionTreeClassifier(random\_state=1)

y = corpus.Class

y

# ### Apply the algorithm to the 3 matrices; document and discuss their performance using cross validation

# In[17]:

tm.crossvalidate\_model(dt\_clf, baseline\_count\_matrix, y, print\_=True)

# In[18]:

tm.crossvalidate\_model(dt\_clf, baseline\_tf\_matrix, y, print\_=True)

# In[19]:

tm.crossvalidate\_model(dt\_clf, baseline\_tfidf\_matrix, y, print\_=True)

# ### Derive word/token statistics for each category and explain what they indicate

# In[3]:

documents = list(corpus.Article)

# In[4]:

baseline\_count\_matrix = tm.build\_count\_matrix(documents)

baseline\_count\_matrix

# In[5]:

attributes = sorted(set(list(baseline\_count\_matrix.columns)))

print(attributes)

# In[6]:

dc\_text = ' '.join(corpus.Article[corpus.Class == 'DC'])

marv\_text = ' '.join(corpus.Article[corpus.Class == 'Marvel'])

nba\_text = ' '.join(corpus.Article[corpus.Class == 'NBA'])

# In[7]:

tm.print\_n\_mostFrequent("DC", dc\_text, 10)

tm.print\_n\_mostFrequent("MARVEL", marv\_text, 10)

tm.print\_n\_mostFrequent("NBA", nba\_text, 10)

# In[8]:

tm.print\_common\_tokens([dc\_text, marv\_text, nba\_text])

# ### Use visualisations techniques (e.g., bar charts, word clouds) and identify frequently occuring terms, potential stop words, synonyms, concepts, and word variations comment on each topic/category

# In[9]:

texts = [dc\_text, marv\_text, nba\_text]

tm.plot\_POS\_freq(texts, 'JJ', ['dc', 'marvel', 'nba'])

# In[10]:

tm.plot\_POS\_freq(texts, 'NN', ['dc', 'marvel', 'nba'])

# In[11]:

tm.plot\_POS\_freq(texts, 'DT', ['dc', 'marvel', 'nba'])

# In[12]:

tm.generate\_cloud(dc\_text)

# In[13]:

tm.generate\_cloud(marv\_text)

# In[14]:

tm.generate\_cloud(nba\_text)

# ### Use 2 clustering algorithms with 2 different linkage schemes (e.g., minimum linkage vs. maximum linkage) and 2 different measures (e.g., symmetric vs. cosine) to identify the main clusters; give details of the algorithms, schemes and measures you tried, and what the results were: do they accurately identify the three clusters of text documents? If not, analyse the results to determine why not

# In[15]:

baseline\_tfidf\_matrix = tm.build\_tfidf\_matrix(documents)

baseline\_tfidf\_matrix

# In[16]:

y= corpus.Class

print(y)

# In[17]:

agg\_single\_cosine = AgglomerativeClustering(n\_clusters=3, affinity='cosine',

linkage='single')

agg\_single\_cosine.fit(baseline\_tfidf\_matrix)

agg\_single\_cosine\_labels = agg\_single\_cosine.labels\_

print(agg\_single\_cosine\_labels)

print(list(y))

# In[18]:

agg\_single\_symmetric = AgglomerativeClustering(n\_clusters=3, affinity='manhattan',

linkage='single')

agg\_single\_symmetric.fit(baseline\_tfidf\_matrix)

agg\_single\_symmetric\_labels = agg\_single\_symmetric.labels\_

print(agg\_single\_symmetric\_labels)

print(list(y))

# In[19]:

agg\_complete\_cosine = AgglomerativeClustering(n\_clusters=3, affinity='cosine',

linkage='complete')

agg\_complete\_cosine.fit(baseline\_tfidf\_matrix)

agg\_complete\_cosine\_labels = agg\_complete\_cosine.labels\_

print(agg\_complete\_cosine\_labels)

print(list(y))

# In[20]:

agg\_complete\_symmetry = AgglomerativeClustering(n\_clusters=3, affinity='manhattan',

linkage='complete')

agg\_complete\_symmetry.fit(baseline\_tfidf\_matrix)

agg\_complete\_symmetry\_labels = agg\_complete\_symmetry.labels\_

print(agg\_complete\_symmetry\_labels)

print(list(y))

# In[28]:

#do k++ clustering

km\_plus = KMeans(n\_clusters=3, random\_state=1, )

km\_plus.fit(baseline\_tfidf\_matrix)

km\_plus.fit\_predict(baseline\_tfidf\_matrix)

#obtain the labels

plus\_cluster\_labels = km\_plus.labels\_

##compare the cluster labels with the actual labels

print(plus\_cluster\_labels)

print(list(y))

# In[ ]:

#do k++ clustering

km\_plus = KMeans(n\_clusters=3, random\_state=1, )

km\_plus.fit(baseline\_tfidf\_matrix)

km\_plus.fit\_predict(baseline\_tfidf\_matrix)

#obtain the labels

plus\_cluster\_labels = km\_plus.labels\_

##compare the cluster labels with the actual labels

print(plus\_cluster\_labels)

print(list(y))

# ### Text preprocessing tasks: what preprocessing tasks are the most suitable for your data? Choose at least 3 tasks based on your findings from data understanding and discuss why they might be suitable. Document and discuss the incremental performance after each applied technique to the 3 matrices and decide whether they should be included in the final pipeline (justify your decisions)

# In[4]:

#Initial cleaning of data

clean\_data = corpus.copy()

clean\_data.Article = clean\_data.Article.apply(tm.clean\_doc)

clean\_data

# In[5]:

clean\_count\_matrix = tm.build\_count\_matrix(list(clean\_data.Article))

tm.crossvalidate\_model(dt\_clf, clean\_count\_matrix, y, print\_=True)

print("No. of terms after cleaning:", clean\_count\_matrix.shape[1])

# In[6]:

clean\_tf\_matrix = tm.build\_tf\_matrix(list(clean\_data.Article))

tm.crossvalidate\_model(dt\_clf, clean\_tf\_matrix, y, print\_=True)

print("No. of terms after cleaning:", clean\_tf\_matrix.shape[1])

# In[7]:

clean\_tfidf\_matrix = tm.build\_tfidf\_matrix(list(clean\_data.Article))

tm.crossvalidate\_model(dt\_clf, clean\_tfidf\_matrix, y, print\_=True)

print("No. of terms after cleaning:", clean\_tfidf\_matrix.shape[1])

# In[8]:

# Stop words removal

improved\_data = clean\_data.copy()

universal\_sw = nltk.corpus.stopwords.words('english')

print(universal\_sw)

# In[9]:

swr\_u\_data = improved\_data.copy()

swr\_u\_data.Article = swr\_u\_data.Article.apply(tm.remove\_sw, sw=universal\_sw)

swr\_u\_count\_matrix = tm.build\_count\_matrix(list(swr\_u\_data.Article))

tm.crossvalidate\_model(dt\_clf, swr\_u\_count\_matrix, y, print\_=True)

print("No. of terms after removal:", swr\_u\_count\_matrix.shape[1])

# In[10]:

swr\_u\_tf\_matrix = tm.build\_tf\_matrix(list(swr\_u\_data.Article))

tm.crossvalidate\_model(dt\_clf, swr\_u\_tf\_matrix, y, print\_=True)

print("No. of terms after removal:", swr\_u\_tf\_matrix.shape[1])

# In[11]:

swr\_u\_tfidf\_matrix = tm.build\_tfidf\_matrix(list(swr\_u\_data.Article))

tm.crossvalidate\_model(dt\_clf, swr\_u\_tfidf\_matrix, y, print\_=True)

print("No. of terms after removal:", swr\_u\_tfidf\_matrix.shape[1])

# In[12]:

custom\_sw = ['the', 'of', 'and', 'to', 'in', 'is', 'was', 'on', 's']

swr\_c\_data = improved\_data.copy()

swr\_c\_data.Article = swr\_c\_data.Article.apply(tm.remove\_sw, sw=custom\_sw)

swr\_c\_count\_matrix = tm.build\_count\_matrix(list(swr\_c\_data.Article))

tm.crossvalidate\_model(dt\_clf, swr\_c\_count\_matrix, y, print\_=True)

print("No. of terms after removal:", swr\_c\_count\_matrix.shape[1])

# In[13]:

swr\_c\_tf\_matrix = tm.build\_tf\_matrix(list(swr\_c\_data.Article))

tm.crossvalidate\_model(dt\_clf, swr\_c\_tf\_matrix, y, print\_=True)

print("No. of terms after removal:", swr\_c\_tf\_matrix.shape[1])

# In[14]:

swr\_c\_tfidf\_matrix = tm.build\_tfidf\_matrix(list(swr\_c\_data.Article))

tm.crossvalidate\_model(dt\_clf, swr\_c\_tfidf\_matrix, y, print\_=True)

print("No. of terms after removal:", swr\_c\_tfidf\_matrix.shape[1])

# In[15]:

#Improving the BOW

repl\_dictionary = {

'comics': ['comic(s)[-]books', 'stories'],

'superhero':['superheroes', 'hero(es)'],

'writer': ['author(s)', 'creator(s)'],

'NBA': ['league'],

'team': ['franchise(s)'],

'season': ['year']

}

improved\_data.Article = improved\_data.Article.apply(tm.improve\_bow, replc\_dict=repl\_dictionary)

improved\_count\_matrix = tm.build\_count\_matrix(list(improved\_data.Article))

tm.crossvalidate\_model(dt\_clf, improved\_count\_matrix, y, print\_=True)

print("No. of terms after improving the bow:", improved\_count\_matrix.shape[1])

# In[16]:

improved\_tf\_matrix = tm.build\_tf\_matrix(list(improved\_data.Article))

tm.crossvalidate\_model(dt\_clf, improved\_tf\_matrix, y, print\_=True)

print("No. of terms after improving the bow:", improved\_tf\_matrix.shape[1])

# In[17]:

improved\_tfidf\_matrix = tm.build\_tfidf\_matrix(list(improved\_data.Article))

tm.crossvalidate\_model(dt\_clf, improved\_tfidf\_matrix, y, print\_=True)

print("No. of terms after improving the bow:", improved\_tfidf\_matrix.shape[1])

# ### Algorithms-based Feature selection/reduction tasks: Choose at least 2 techniques to try. Document and discuss the performance after each applied technique and decidewhich one to include in the final p ipeline (justify your decision); of the terms chosen by the algorithms as being the most predictive, do they concur with the terms you thought would be the best predictors from data understanding?

# In[18]:

## Univariate Feature Selection

uni\_data = improved\_data.copy()

uni\_tfidf\_matrix = tm.build\_tfidf\_matrix(

list(uni\_data.Article))

uni\_reduced\_tfidf\_matrix = tm.univariate\_selection(

uni\_tfidf\_matrix, uni\_data.Class, scheme=f\_classif)

uni\_reduced\_tfidf\_scores = tm.crossvalidate\_model(

dt\_clf, uni\_reduced\_tfidf\_matrix, y)

print("No. of terms after applying anova feature selection:",

uni\_reduced\_tfidf\_matrix.shape[0])

# In[19]:

# RFE

rfe\_data = improved\_data.copy()

rfe\_tfidf\_matrix = tm.build\_tfidf\_matrix(

list(rfe\_data.Article))

rfe\_reduced\_tfidf\_matrix = tm.rfe\_selection(

dt\_clf, rfe\_tfidf\_matrix, y, n=100, step=2)

rfe\_tfidf\_scores = tm.crossvalidate\_model(

dt\_clf, rfe\_reduced\_tfidf\_matrix, y)

print("No. of terms after rfe:",

rfe\_reduced\_tfidf\_matrix.shape[1])

# In[4]:

## Hyperparameter Tuning

params = {

"criterion": ['gini', 'entropy'],

"max\_depth": range(3, 16),

"min\_samples\_split": range(2, 16),

"min\_samples\_leaf": range(3, 10),

"min\_impurity\_decrease": [0.01, 0.02, 0.03, 0.04, 0.05]

}

# In[5]:

## Baseline Count Matrix

documents = list(corpus.Article)

baseline\_count\_matrix = tm.build\_count\_matrix(documents)

baseline\_count\_scores = tm.crossvalidate\_model(dt\_clf, baseline\_count\_matrix, y, print\_=True)

# In[7]:

## change the params of the DT to the optimal ones above

opt\_baseline\_count\_clf = DecisionTreeClassifier(random\_state=1,

criterion='gini',

max\_depth=3,

min\_impurity\_decrease=0.01,

min\_samples\_split=2,

min\_samples\_leaf=5)

## retrain and get performance

opt\_baseline\_count\_scores = tm.crossvalidate\_model(opt\_baseline\_count\_clf,

baseline\_count\_matrix,

y)

# In[8]:

## Baseline TF Matrix

baseline\_tf\_matrix = tm.build\_tf\_matrix(documents)

tm.crossvalidate\_model(dt\_clf, baseline\_tf\_matrix, y, print\_=True)

# In[12]:

## change the params of the DT to the optimal ones above

opt\_baseline\_tf\_clf = DecisionTreeClassifier(random\_state=1,

criterion='gini',

max\_depth=4,

min\_impurity\_decrease=0.01,

min\_samples\_split=2,

min\_samples\_leaf=2)

## retrain and get performance

opt\_baseline\_tf\_scores = tm.crossvalidate\_model(opt\_baseline\_tf\_clf,

baseline\_count\_matrix,

y)

# In[ ]:

## Baseline TFIDF Matrix

baseline\_tfidf\_matrix = tm.build\_tfidf\_matrix(documents)

tm.crossvalidate\_model(dt\_clf, baseline\_tfidf\_matrix, y, print\_=True)

# In[ ]:

# In[4]:

#Initial cleaning of data

clean\_data = corpus.copy()

clean\_data.Article = clean\_data.Article.apply(tm.clean\_doc)

clean\_data

# In[8]:

clean\_count\_matrix = tm.build\_count\_matrix(list(clean\_data.Article))

tm.crossvalidate\_model(dt\_clf, clean\_count\_matrix, y, print\_=True)

print("No. of terms after cleaning:", clean\_count\_matrix.shape[1])

# In[ ]:

tm.search\_optimal\_params(dt\_clf, clean\_count\_matrix,

y, params)

# In[5]:

improved\_data = clean\_data.copy()

# In[14]:

# RFE

rfe\_data = improved\_data.copy()

rfe\_tfidf\_matrix = tm.build\_tfidf\_matrix(

list(rfe\_data.Article))

rfe\_reduced\_tfidf\_matrix = tm.rfe\_selection(

dt\_clf, rfe\_tfidf\_matrix, y, n=100, step=2)

rfe\_tfidf\_scores = tm.crossvalidate\_model(

dt\_clf, rfe\_reduced\_tfidf\_matrix, y)

print("No. of terms after rfe:",

rfe\_reduced\_tfidf\_matrix.shape[1])

# In[15]:

tm.search\_optimal\_params(dt\_clf, rfe\_reduced\_tfidf\_matrix,

y, params)

# In[17]:

## change the params of the DT to the optimal ones above

opt\_tfidf\_clf = DecisionTreeClassifier(random\_state=1,

criterion='gini',

max\_depth=3,

min\_impurity\_decrease=0.01,

min\_samples\_split=2,

min\_samples\_leaf=5)

## retrain and get performance

opt\_tfidf\_scores = tm.crossvalidate\_model(opt\_tfidf\_clf,

rfe\_reduced\_tfidf\_matrix,

y)

# In[11]:

## Univariate Feature Selection

uni\_data = improved\_data.copy()

uni\_tfidf\_matrix = tm.build\_tfidf\_matrix(

list(uni\_data.Article))

uni\_reduced\_tfidf\_matrix = tm.univariate\_selection(

uni\_tfidf\_matrix, uni\_data.Class, scheme=f\_classif)

uni\_reduced\_tfidf\_scores = tm.crossvalidate\_model(

dt\_clf, uni\_reduced\_tfidf\_matrix, y)

print("No. of terms after applying anova feature selection:",

uni\_reduced\_tfidf\_matrix.shape[0])

# In[12]:

## Hyperparameter Tuning

params = {

"criterion": ['gini', 'entropy'],

"max\_depth": range(3, 16),

"min\_samples\_split": range(2, 16),

"min\_samples\_leaf": range(3, 10),

"min\_impurity\_decrease": [0.01, 0.02, 0.03, 0.04, 0.05]

}

tm.search\_optimal\_params(dt\_clf, uni\_reduced\_tfidf\_matrix,

y, params)

# In[13]:

## change the params of the DT to the optimal ones above

opt\_tfidf\_clf = DecisionTreeClassifier(random\_state=1,

criterion='gini',

max\_depth=3,

min\_impurity\_decrease=0.01,

min\_samples\_split=2,

min\_samples\_leaf=3)

## retrain and get performance

opt\_tfidf\_scores = tm.crossvalidate\_model(opt\_tfidf\_clf,

uni\_reduced\_tfidf\_matrix,

y)