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.

```
##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)
```

Figure 1. Function To Retrieve Text Data

```
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://fictionhorizon.com/20-best-dc-comics-characters-of-all-time-ranked/",
    "https://www.ranker.com/crowdranked-list/best-dc-comics-heroes",
    "https://www.shortlist.com/lists/best-dc-characters-402104",
    "https://www.ign.com/articles/2019/06/26/the-top-25-heroes-of-dc-comics",
    "https://www.thetoptens.com/dc-comics-characters/",
    "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.dailydot.com/parsec/dc-heroes/",
    "https://www.listchallenges.com/top-100-dc-comics-characters",
    "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-2016/",
    "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.thetoptens.com/dc-superheroes/",
    "https://www.cheatsheet.com/entertainment/dc-comics-greatest-superheroes-of-all-time.html/"
    ]
```

Figure 2. URLs for DC 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",
    "https://www.thetoptens.com/best-marvel-super-heroes/"
    ]
```

Figure 3. URLs For Marvel Comics

```
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/4n84f58mc6sz15
    "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/13vm1p03wlnre14hecrk060vn
    "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"
    ]
```

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.

```
In [61]:
## 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
```

```
30

Out[61]: ['screenrant.comThe 10 Best DC Comic Books Of 2021, According To Reddit2021 marked the start of DC Comics\' launch of Infinit
e Frontier, and fans on Reddit celebrate the various great comic book series that it came with.The last year was another wher
e the comic book industry saw a boost in mainstream commercial and critical success. 2021 was also the year that marked DC Co
mics\' launch of their\xa0Infinite Frontier\xa0brand-wide initiative to\xa0progress beyond\xa02016\'s\xa0Rebirth\xa0era.RELAT
ED: 10 Best Daredevil Comics For New Fans To Start WithAside from superhero icons like Batman, Superman, and Wonder Woman, so
me great creator-driven series have also earned their respective acclaim. Comic book fans on Reddit have shared their thought
s and compiled their lists on what they believe to be the publisher\'s best work over the last year, ranging from their mainl
ine and alternate canon stories.While his live-action tenure on the then-DC Universe streaming platform was unfortunately cut
short, comic book fans commonly regard him as one of DC\'s best monster characters. Redditor LordCosmagog\xa0chimes in saying
"For me, the best books of the last 12 months have been Justice League Dark & Swamp Thing," with the\xa0Guardian of the Green
being one of JLD\'s prominent members historically.Ram V.\'s\xa0The Swamp Thing\xa0limited series garnered consistently high
praise across its 10-issue run in its introduction of the next Guardian of the Green and new compelling, emotional themes tha
t the likes of Alan Moore revitalized the character with. The great reception of the series led DC to greenlight another six
issues for another run.The recent trailer for\xa0The Batman\xa0has stirred even more fan excitement, making now a good time f
or fans to read Mattson Tomlin\'s\xa0Batman: The Imposter. Tomlin was brought on in 2019 to assist Reeves and Peter Craig in
writing the movie, and the premise and quality of the series seem like a clear influence from his time on it.It\'s perhaps th
e most grounded the Dark Knight has been in comics, focusing on a gritty, street-level crime-noir story of someone impersonat
ing Batman to ruin his young reputation. Redditor SequentialNation\xa0compiled a list including "the various excellent creato
```

Figure 5. dc_docs Variable

```
In [65]:
## 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
```

```
30

Out[65]: ['\n\n\n\nBest Comics of 2022 (So Far): Marvel ComicsAll manner of writing about comic booksBy Zack Quaintance – We've reache
d (roughly) the 1/4 through the year mark (very roughly), and as such, we're starting our annual tradition: ranking the best
comics of 2022 (so far). Each Friday through the end of the month, we'll have a new list. Today's is the best comics of 2022
– Marvel Comics. On the Fridays to come, we'll soon have lists as well for DC Comics, Image Comics, and just generally indie
comics.And hey, we're also doing things just a little bit differently this year. I'm going to stick with these lists and upda
te them in real time as more books arrive, right up through October. So if you like what you find here, make sure to bookmark
it and check back periodically through the rest of the year. Anyway! The best comics of 2022 Marvel are below…enjoy!1. Iron M
anWriter: Christopher CantwellPrimary Artist: CafuMost Recent Trade: Iron Man, Vol. 2 - Books of Korvac IIWhy This Is So Goo
d: Holy cow, where to start? This current run of Iron Man – which is a manageable two trades long at the time of this writing
– is the ideal of what I want from my superhero comics. First and foremost, it's character driven, like all of Christopher Ca
ntwell's comics tend to be. Rather than deploying a quip machine Tony Stark like we see in the movies, this book really looks
at Tony's psychology, drawing it instead from his long history in the comics. It tackles his roots as Iron Man, his drinking
problem, his troubles with relationships, and it tackles it well. There's plenty of big superhero fighting (we'll get into i
t) and sci-fi hi jinx (we'll get into that too) in these pages, but it's all driven by emotional complexity.That said, in bet
ween that intense examination, it's also a total romp of a sci-fi superhero adventure comic that draws from deep cuts in Marv
el continuity. We see Tony assembling a B (or C…or even D) list team of superheroes, we see him taking them to space, and we
see him squaring off with an equally-matched villain from the Avengers past in ways that will give you vibes of Star Trek: Th
e Original Series. It's really smart and hard to predict, and it's the best use of Iron Man I've seen in a comic in years, if
```

Figure 6. marv_docs Variable

```
In [14]:
## 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
```

```
30

Out[14]: ['\n2022: Ranking the top 20 NBA players right now\nWatch\n\n\n       src="https://www.facebook.com/tr?id=674090812743125&ev=
PageView&noscript=1"/>\n',
 '50 Best NBA players of 2022: Giannis leads the charge as injuries impact othersAs the season continues to heat up, the Best
NBA players are rising to the top. Making big impacts. Let’s take a look at the top 50 so far:Note: The article was originall
y written by Jarrod Castillo, since updated by the Sportsnaut team.Consistency has obviously been an issue for the 20-year-ol
d Edwards since Minnesota made him the No. 1 pick in the 2020 NBA Draft. Despite this, the flashes this Georgia product has s
hown thus far in his career makes him a top-50 NBA player. The kid is averaging nearly 22 points per game as a sophomore and
has seen both his effective field goal and three-point percentages increase a ton since his rookie campaign.The other half of
the vaunted “Splash Bros.,” Thompson just returned to action after a two-year hiatus. While he’s been limited in minutes as t
he Warriors look to get him up to speed, we have not seen too much of a drop-off in production. The three-time champion is av
eraging 16.8 points, 3.9 rebounds and what would be a career-best 2.9 assists. He’s also shooting 35% from distance while con
tinuing to play elite-level defense. We’re intrigued to see how Thompson looks once he’s playing his full 38 minutes per game
(26.5 thus far).Acquired from Sacramento during February’s NBA trade deadline, Haliburton is among the best young point guard
s in the NBA. In limited action with the Pacers, he’s averaging 19.2 points to go with 9.5 assists while shooting 41% from th
ree-point range. Having recently turned 22, he’s still not close to his prime. The sky is almost literally the limit for the
Iowa State product.Related: NBA schedule and fantasy basketball picksA first-round pick of the Nets back in 2017, Allen displ
ayed flashes of brilliance in Brooklyn before heading to Cleveland in the James Harden trade to open the 2020-21 season. Sinc
e joining the Cavaliers, he’s been an elite force for this up-and-coming squad. That includes Allen averaging 14.6 points and
```

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.

```
In [66]:
## 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
```

```
followed a group of "the DC Universe’s greatest detectives," as they tried to figure out who Leviathan was before they could
strike again\xa0and reveal who was the mastermind was behind the mask.Even though\xa0Bendis maintained that Leviathan wasn’t
a new character, but someone who existed in DC Comics canon, the final reveal of Leviathan\'s identity did leave many feeling
cheated. What saved this series and qualified it to be included on this "best of list" was the artwork of Alex Maleev. Malee
v, who has collaborated with Bendis on such memorable series as Daredevil and various Marvel and creator-owned properties, se
t the tone for the book and made the visual components of the story the reason to come back each issue.Related:\xa0DC\'s New
BATMAN is Everything Fans Love About The Dark KnightOn the surface, DCeased seemed like it was DC\'s\xa0answer to the Marvel
Zombies franchise. It proved to be a fast-paced series that focused on the idea of hope and perseverance in the face of devas
tating events.DCeased\xa0centers around the citizens and heroes of the DC Universe being assaulted by a terrifying disease th
at is spread through technology and turns people into zombielike creatures. This disease was the result of Darkseid bonding t
he Anti-Life equation with Cyborg\'s techno-organic components, which he then used to get back at the Justice League for stop
ping his invasion of Earth. The kicker was that no one was safe, and anyone from the regular citizen to the heroes and villai
ns was considered safe from being zombified as the disease spread.Tom Taylor, the scribe of DCeased, is no stranger to altern
ate DC universes having writing other limited series such as Injustice: Gods Among Us. In DCceased, Taylor presented a fast-p
aced story along with excellent small moments between the characters throughout the series.\xa0While the ending of the six-is
sue miniseries seemed a bit rushed, it was well executed and feels like there is more of the story to be told in the future.M
ore:\xa0The Best Marvel Comics Miniseries of 2019Chad Burdette is life long comic book reading and collector with many issue
s, some of which are bagged, boarded, and filed away. Chad is originally from West Virginia but now lives in Albany, NY.',
 "Best DC Comics of 2019Comic Book HeraldA Comic Book Reading Order Guide For Beginners & FansJune 28, 2019 by Dave 1 Comment
```

Figure 8. all_docs Variable & all_labels Variables

In [68]:

```
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
```

Out[68]:

	Article	Class
62	Top 100 NBA Players of 2022. And the Best Play...	NBA
19		DC
67	Stephen Curry Ranks as NBA's Most-Liked Player...	NBA
34	www.cbr.com	Marvel
73	\n Most valuable pl...	NBA
...	...	...
50	Top 10 Most Popular Marvel CharactersGet the b...	Marvel
57	The 30 Best Marvel Graphic Novels of All TimeS...	Marvel
70	Ranking The Top 20 NBA Players 2021The 2020-20...	NBA
17	\r\n\t\t\t\t\t\tTop 100 DC Comics Characters\r\n...	DC
40	Not found, error 404Comic Book HeraldA Comic B...	Marvel

90 rows × 2 columns

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.

In [9]:
```
documents = list(corpus.Article)
baseline_count_matrix = tm.build_count_matrix(documents)
baseline_count_matrix
```

Out[9]:

	NBA	's	Top	100	Players	for	2019-20	:	Ranking	the	...	encompassed	blessed	2021.lf	clicking	HERE	healthy.Fill	commenting	WordPress.c
0	3.0	3.0	4.0	3.0	1.0	1.0	3.0	2.0	1.0	11.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	1.0	0.0	2.0	0.0	8.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	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	0.0	0.0	0.0	0.0	
3	11.0	12.0	3.0	3.0	2.0	14.0	0.0	12.0	1.0	112.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	1.0	0.0	0.0	0.0	16.0	0.0	8.0	0.0	81.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
85	0.0	17.0	0.0	0.0	0.0	13.0	0.0	8.0	0.0	96.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
86	0.0	3.0	0.0	0.0	0.0	10.0	0.0	18.0	0.0	145.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
87	5.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	10.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
88	0.0	53.0	0.0	0.0	0.0	13.0	0.0	13.0	0.0	179.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
89	0.0	0.0	0.0	1.0	0.0	35.0	0.0	19.0	0.0	102.0	...	1.0	1.0	1.0	1.0	1.0	1.0	4.0	

90 rows × 22802 columns

Figure 10. Count Matrix

In [11]:
```
baseline_tf_matrix = tm.build_tf_matrix(documents)
baseline_tf_matrix

```

Out[11]:

	NBA	's	Top	100	Players	for	2019-20	:	Ranking	the	...	encompassed	blessed	2021.lf	clicking	HEF
0	0.028037	0.028037	0.037383	0.028037	0.009346	0.009346	0.028037	0.018692	0.009346	0.102804	...	0.000000	0.000000	0.000000	0.000000	0.0000
1	0.000000	0.000000	0.000000	0.000000	0.000000	0.004098	0.000000	0.008197	0.000000	0.032787	...	0.000000	0.000000	0.000000	0.000000	0.0000
2	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.0000
3	0.004756	0.005188	0.001297	0.001297	0.000865	0.006053	0.000000	0.005188	0.000432	0.048422	...	0.000000	0.000000	0.000000	0.000000	0.0000
4	0.000000	0.000402	0.000000	0.000000	0.000000	0.006431	0.000000	0.003215	0.000000	0.032556	...	0.000000	0.000000	0.000000	0.000000	0.0000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
85	0.000000	0.011281	0.000000	0.000000	0.000000	0.008626	0.000000	0.005309	0.000000	0.063703	...	0.000000	0.000000	0.000000	0.000000	0.0000
86	0.000000	0.000932	0.000000	0.000000	0.000000	0.003108	0.000000	0.005594	0.000000	0.045059	...	0.000000	0.000000	0.000000	0.000000	0.0000
87	0.033784	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.006757	0.000000	0.067568	...	0.000000	0.000000	0.000000	0.000000	0.0000
88	0.000000	0.018976	0.000000	0.000000	0.000000	0.004654	0.000000	0.004654	0.000000	0.064089	...	0.000000	0.000000	0.000000	0.000000	0.0000
89	0.000000	0.000000	0.000000	0.000317	0.000000	0.011083	0.000000	0.006016	0.000000	0.032299	...	0.000317	0.000317	0.000317	0.000317	0.0003

90 rows × 22802 columns

Figure 11. Normalised Count Matrix

In [12]:
```
baseline_tfidf_matrix = tm.build_tfidf_matrix(documents)
baseline_tfidf_matrix
```

Out[12]:

	NBA	's	Top	100	Players	for	2019-20	:	Ranking	the	...	encompassed	blessed	2021.lf	clicking	HERE
0	0.045809	0.022189	0.069084	0.069865	0.028340	0.001588	0.088876	0.004207	0.031046	0.010233	...	0.000000	0.000000	0.000000	0.000000	0.000000
1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000696	0.000000	0.001845	0.000000	0.003263	...	0.000000	0.000000	0.000000	0.000000	0.000000
2	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000
3	0.007770	0.004106	0.002397	0.003232	0.002622	0.001029	0.000000	0.001168	0.001436	0.004820	...	0.000000	0.000000	0.000000	0.000000	0.000000
4	0.000000	0.000318	0.000000	0.000000	0.000000	0.001093	0.000000	0.000724	0.000000	0.003241	...	0.000000	0.000000	0.000000	0.000000	0.000000
..	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
5	0.000000	0.008928	0.000000	0.000000	0.000000	0.001466	0.000000	0.001195	0.000000	0.006341	...	0.000000	0.000000	0.000000	0.000000	0.000000
6	0.000000	0.000738	0.000000	0.000000	0.000000	0.000528	0.000000	0.001259	0.000000	0.004485	...	0.000000	0.000000	0.000000	0.000000	0.000000
7	0.055198	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.001521	0.000000	0.006725	...	0.000000	0.000000	0.000000	0.000000	0.000000
8	0.000000	0.015018	0.000000	0.000000	0.000000	0.000791	0.000000	0.001048	0.000000	0.006379	...	0.000000	0.000000	0.000000	0.000000	0.000000
9	0.000000	0.000000	0.000000	0.000789	0.000000	0.001883	0.000000	0.001354	0.000000	0.003215	...	0.002056	0.002056	0.002056	0.002056	0.002056

rows × 22802 columns

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).

```
In [15]:
1 dt_clf = DecisionTreeClassifier(random_state=1)
2 y = corpus.Class
3 y

Out[15]: 0       NBA
         1    Marvel
         2        DC
         3       NBA
         4        DC
               ...
         85       DC
         86   Marvel
         87      NBA
         88       DC
         89   Marvel
         Name: Class, Length: 90, dtype: object
```

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.

```
In [17]:
1 tm.crossvalidate_model(dt_clf, baseline_count_matrix, y, print_=True)

Accuracy: 0.90
Precision macro: 0.91
Recall macro: 0.90

Out[17]: (0.9, 0.9074603174603174, 0.9)

In [18]:
1 tm.crossvalidate_model(dt_clf, baseline_tf_matrix, y, print_=True)

Accuracy: 0.93
Precision macro: 0.94
Recall macro: 0.93

Out[18]: (0.9333333333333333, 0.9390476190476191, 0.9333333333333333)

In [19]:
1 tm.crossvalidate_model(dt_clf, baseline_tfidf_matrix, y, print_=True)

Accuracy: 0.93
Precision macro: 0.94
Recall macro: 0.93

Out[19]: (0.9333333333333333, 0.9390476190476191, 0.9333333333333333)
```

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.

```
In [7]:
1 tm.print_n_mostFrequent("DC", dc_text, 10)
2 tm.print_n_mostFrequent("MARVEL", marv_text, 10)
3 tm.print_n_mostFrequent("NBA", nba_text, 10)
```

```
10 most frequent tokens in DC:  [(',', 3422), ('the', 3242), ('of', 2119), ('and', 1806), ('.', 1735), ('a', 1375), ('to', 131
8), ('’', 1093), ('in', 895), ('s', 792)]
        Frequency of "," is 0.04813751969390052
        Frequency of "the" is 0.045605446770200314
        Frequency of "of" is 0.02980812514067072
        Frequency of "and" is 0.02540513166779203
        Frequency of "." is 0.024406369570110286
        Frequency of "a" is 0.01934222372270988
        Frequency of "to" is 0.018540400630204817
        Frequency of "’" is 0.015375309475579564
        Frequency of "in" is 0.012590029259509 34
        Frequency of "s" is 0.01114112086428089
10 most frequent tokens in MARVEL:  [(',', 2823), ('the', 2286), ('of', 1526), ('and', 1469), ('.', 1419), ('to', 1043), ('a',
1017), ('’', 826), (':', 725), ('in', 723)]
        Frequency of "," is 0.048903440390811764
        Frequency of "the" is 0.039600873090115375
        Frequency of "of" is 0.02643522849322662
        Frequency of "and" is 0.025447805148459967
        Frequency of "." is 0.024581644319717284
        Frequency of "to" is 0.018068114887572324
        Frequency of "a" is 0.01761771125662613
        Frequency of "’" is 0.014308976890829089
        Frequency of ":" is 0.012559332016768874
        Frequency of "in" is 0.012524685583619166
10 most frequent tokens in NBA:  [(',', 5965), ('the', 5312), ('.', 4279), ('a', 2705), ('to', 2388), ('and', 2358), ('of', 224
5), ('in', 2212), ('’', 1890), ('his', 1678)]
        Frequency of "," is 0.04839638791753548
        Frequency of "the" is 0.04309834243385557
        Frequency of "." is 0.03471720769474171
        Frequency of "a" is 0.021946727463023213
        Frequency of "to" is 0.019374781952569104
        Frequency of "and" is 0.019131380169245375
        Frequency of "of" is 0.018214566785392648
        Frequency of "in" is 0.01794682482373654
        Frequency of "’" is 0.015334312349395146
        Frequency of "his" is 0.013614273080574104
```

Figure 15. Frequent Tokens

```
In [8]:
1 tm.print_common_tokens([dc_text, marv_text, nba_text])
```

```
============ Common Features ==========
 2419
 ['!' '#' '$' ... '“' '”' '…']
```

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.

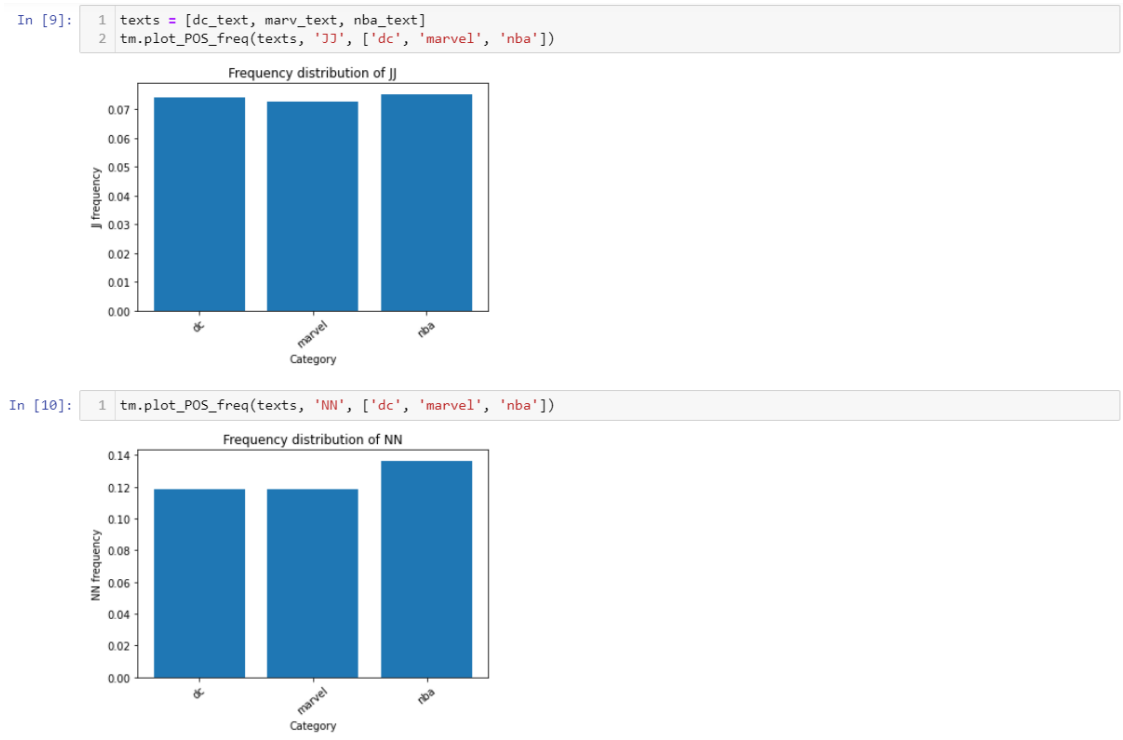

Figure 17. Plotting POS Frequency

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’.

```
In [12]:  1  tm.generate_cloud(dc_text)
```

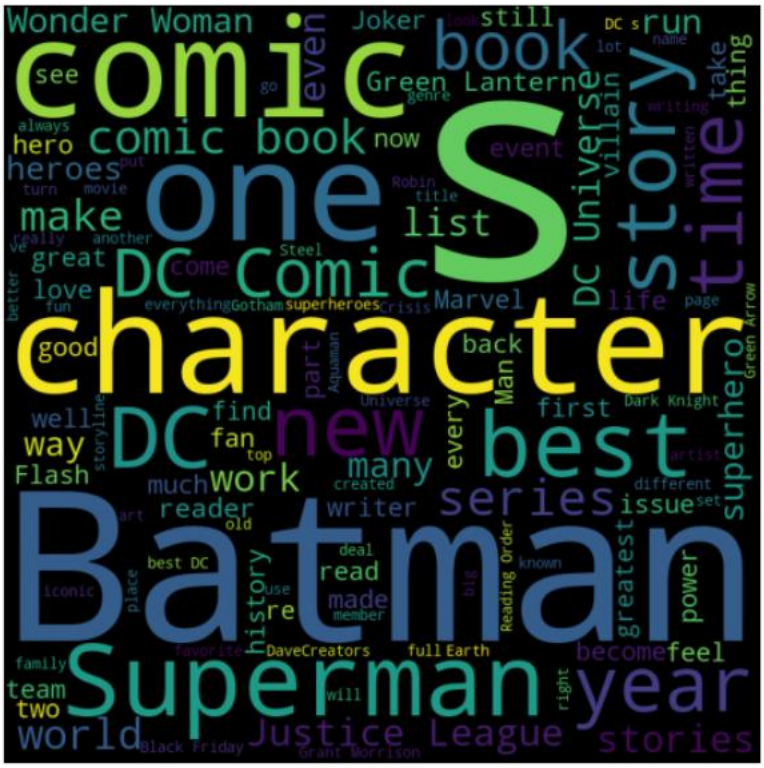

Figure 18. DC WorldCloud

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’.

```
In [13]:  1 tm.generate_cloud(marv_text)
          2
```

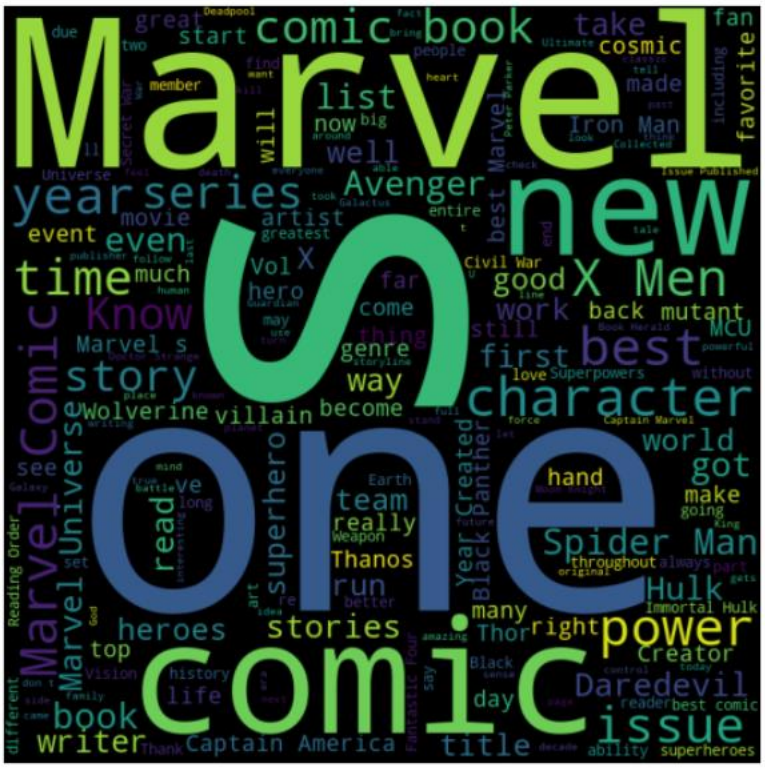

Figure 19. Marvel WordCloud

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’.

```
In [14]:  1 tm.generate_cloud(nba_text)
          2
```

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.

```
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))
```

```
[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 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 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 0 0 1 0 0 0 0 0 0 0 0 0]
['NBA', 'Marvel', 'DC', 'NBA', 'DC', 'NBA', 'Marvel', 'Marvel', 'DC', 'DC', 'DC', 'DC', 'NBA', 'DC', 'Marvel', 'DC', 'DC', 'D
C', 'NBA', 'NBA', 'DC', 'Marvel', 'DC', 'Marvel', 'DC', 'NBA', 'Marvel', 'NBA', 'NBA', 'NBA', 'Marvel', 'NBA', 'NBA', 'Marvel',
'Marvel', 'NBA', 'DC', 'DC', 'Marvel', 'DC', 'Marvel', 'NBA', 'NBA', 'NBA', 'NBA', 'NBA', 'Marvel', 'NBA', 'DC', 'DC', 'Marve
l', 'Marvel', 'DC', 'NBA', 'Marvel', 'Marvel', 'Marvel', 'Marvel', 'DC', 'NBA', 'Marvel', 'Marvel', 'Marvel', 'NBA', 'NBA', 'D
C', 'Marvel', 'NBA', 'DC', 'Marvel', 'DC', 'NBA', 'NBA', 'Marvel', 'Marvel', 'NBA', 'NBA', 'DC', 'NBA', 'DC', 'Marvel', 'Marve
l', 'DC', 'DC', 'DC', 'DC', 'Marvel', 'NBA', 'DC', 'Marvel']
```

```
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))
```

```
[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 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 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 0 0 1 0 0 0 0 0 0 0 0 0]
['NBA', 'Marvel', 'DC', 'NBA', 'DC', 'NBA', 'Marvel', 'Marvel', 'DC', 'DC', 'DC', 'DC', 'NBA', 'DC', 'Marvel', 'DC', 'DC', 'D
C', 'NBA', 'NBA', 'DC', 'Marvel', 'DC', 'Marvel', 'DC', 'NBA', 'Marvel', 'NBA', 'NBA', 'NBA', 'Marvel', 'NBA', 'NBA', 'Marvel',
'Marvel', 'NBA', 'DC', 'DC', 'Marvel', 'DC', 'Marvel', 'NBA', 'NBA', 'NBA', 'NBA', 'NBA', 'Marvel', 'NBA', 'DC', 'DC', 'Marve
l', 'Marvel', 'DC', 'NBA', 'Marvel', 'Marvel', 'Marvel', 'Marvel', 'DC', 'NBA', 'Marvel', 'Marvel', 'Marvel', 'NBA', 'NBA', 'D
C', 'Marvel', 'NBA', 'DC', 'Marvel', 'DC', 'NBA', 'NBA', 'Marvel', 'Marvel', 'NBA', 'NBA', 'DC', 'NBA', 'DC', 'Marvel', 'Marve
l', 'DC', 'DC', 'DC', 'DC', 'Marvel', 'NBA', 'DC', 'Marvel']
```

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.

```
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))
```

```
[0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1]
['NBA', 'Marvel', 'DC', 'NBA', 'DC', 'NBA', 'Marvel', 'Marvel', 'DC', 'DC', 'DC', 'DC', 'NBA', 'DC', 'Marvel', 'DC', 'DC', 'D
C', 'NBA', 'NBA', 'DC', 'Marvel', 'DC', 'Marvel', 'DC', 'NBA', 'Marvel', 'NBA', 'NBA', 'NBA', 'Marvel', 'NBA', 'NBA', 'Marvel',
'Marvel', 'NBA', 'DC', 'DC', 'Marvel', 'DC', 'Marvel', 'NBA', 'NBA', 'NBA', 'NBA', 'NBA', 'Marvel', 'NBA', 'DC', 'DC', 'Marve
l', 'Marvel', 'DC', 'NBA', 'Marvel', 'Marvel', 'Marvel', 'Marvel', 'DC', 'NBA', 'Marvel', 'Marvel', 'Marvel', 'NBA', 'NBA', 'D
C', 'Marvel', 'NBA', 'DC', 'Marvel', 'DC', 'NBA', 'NBA', 'Marvel', 'Marvel', 'NBA', 'NBA', 'DC', 'NBA', 'DC', 'Marvel', 'Marve
l', 'DC', 'DC', 'DC', 'DC', 'Marvel', 'NBA', 'DC', 'Marvel']
```

```
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))
```

```
[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 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 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 0 0 1 0 0 0 0 0 0 0 0 0]
['NBA', 'Marvel', 'DC', 'NBA', 'DC', 'NBA', 'Marvel', 'Marvel', 'DC', 'DC', 'DC', 'DC', 'NBA', 'DC', 'Marvel', 'DC', 'DC', 'D
C', 'NBA', 'NBA', 'DC', 'Marvel', 'DC', 'Marvel', 'DC', 'NBA', 'Marvel', 'NBA', 'NBA', 'NBA', 'Marvel', 'NBA', 'NBA', 'Marvel',
'Marvel', 'NBA', 'DC', 'DC', 'Marvel', 'DC', 'Marvel', 'NBA', 'NBA', 'NBA', 'NBA', 'NBA', 'Marvel', 'NBA', 'DC', 'DC', 'Marve
l', 'Marvel', 'DC', 'NBA', 'Marvel', 'Marvel', 'Marvel', 'Marvel', 'DC', 'NBA', 'Marvel', 'Marvel', 'Marvel', 'NBA', 'NBA', 'D
C', 'Marvel', 'NBA', 'DC', 'Marvel', 'DC', 'NBA', 'NBA', 'Marvel', 'Marvel', 'NBA', 'NBA', 'DC', 'NBA', 'DC', 'Marvel', 'Marve
l', 'DC', 'DC', 'DC', 'DC', 'Marvel', 'NBA', 'DC', 'Marvel']
```

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.

In [4]:
```
#Initial cleaning of data
clean_data = corpus.copy()
clean_data.Article = clean_data.Article.apply(tm.clean_doc)
clean_data
```

Out[4]:

	Article	Class
0	NBA's Top 100 Players for 2019-20: Ranking th...	NBA
1	Rare ComicsTop 50 Marvel Comics Of 2015Top 50 ...	Marvel
2	LibraryLibraryComics and Graphic Novels Columb...	DC
3	NBA Top 100 Players 2021-2022 | Ranking The Be...	NBA
4	Best DC Comics of 2019Comic Book HeraldA Comic...	DC
...	...	...
85	screenrant.comThe Best DC Comics Miniseries of...	DC
86	30 Best Marvel Comics Definitive GuideFree U.S...	Marvel
87	Stephen Curry Ranks as NBA's Most-Liked Player...	NBA
88	Best DC Comics storiesGamesRadar+ is supported...	DC
89	Weird Science Marvel ComicsMarvel Comics Best ...	Marvel

90 rows × 2 columns

Figure 23. Initial Cleaning Of 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])

Accuracy: 0.88
Precision macro: 0.89
Recall macro: 0.88
No. of terms after cleaning: 22478

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])

Accuracy: 0.91
Precision macro: 0.91
Recall macro: 0.91
No. of terms after cleaning: 22478

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])

Accuracy: 0.91
Precision macro: 0.91
Recall macro: 0.91
No. of terms after cleaning: 22478
```

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.

In [14]:
```
# Stop words removal
improved_data = clean_data.copy()
universal_sw = nltk.corpus.stopwords.words('english')
print(universal_sw)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'y
ourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself',
'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those',
'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'a
n', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'b
etween', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'of
f', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both',
'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very',
's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'ar
en', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "have
n't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "should
n't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

Figure 25. Universal Stop words

In [15]:
```
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])
```

```
Accuracy: 0.92
Precision macro: 0.93
Recall macro: 0.92
No. of terms after removal: 22294
```

In [16]:
```
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])
```

```
Accuracy: 0.94
Precision macro: 0.95
Recall macro: 0.94
No. of terms after removal: 22294
```

In [17]:
```
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])
```

```
Accuracy: 0.94
Precision macro: 0.95
Recall macro: 0.94
No. of terms after removal: 22294
```

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.

In [18]:
```
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])
```

```
Accuracy: 0.94
Precision macro: 0.95
Recall macro: 0.94
No. of terms after removal: 22425
```

In [19]:
```
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])
```

```
Accuracy: 0.93
Precision macro: 0.94
Recall macro: 0.93
No. of terms after removal: 22425
```

In [20]:
```
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])
```

```
Accuracy: 0.93
Precision macro: 0.94
Recall macro: 0.93
No. of terms after removal: 22425
```

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.

In [21]:
```
#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])
```

```
Accuracy: 0.91
Precision macro: 0.92
Recall macro: 0.91
No. of terms after improving the bow: 22452
```

In [22]:
```
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])
```

```
Accuracy: 0.90
Precision macro: 0.90
Recall macro: 0.90
No. of terms after improving the bow: 22452
```

In [23]:
```
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])
```

```
Accuracy: 0.90
Precision macro: 0.90
Recall macro: 0.90
No. of terms after improving the bow: 22452
```

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.

In [26]:
```
## 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])
```

```
('DC', 103.04304351383176)
('Batman', 89.88744180615656)
('Superman', 77.94561663486404)
('Wonder', 42.48320481800983)
('Marvel', 35.36081085733576)
('NBA', 34.926781841324456)
('Woman', 34.28310463091443)
('Captain', 28.253974264748138)
('Lantern', 26.819260183295523)
('Hulk', 21.54085049058878)
('comic', 21.52173577047774)
('America', 20.951227343357562)
('Spider-Man', 20.492595398935897)
```

Figure 29. Univariate Feature Selection

```
Accuracy: 0.92
Precision macro: 0.93
Recall macro: 0.92
No. of terms after applying anova feature selection: 90
```

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.

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])
```

```
[('H.G', 11177), ('KID', 11177), ('tales', 11176), ('CASES', 11176), ('comics.Whether', 11175), ('COLT', 11175), ('humanity.The
re', 11174), ('CRIME', 11174), ('restoring', 11173), ('OUTLAW', 11173), ('affair', 11172), ('TRUE', 11172), ('distinctive', 111
71), ('aimed', 11171), ('Zods', 11170), ('audience.Joe', 11170), ('Luthors', 11169), ('magazine', 11169), ('pitted', 11168),
('OFFICIAL', 11168), ('RankedSuperman', 11167), ('look-out', 11167), ('Origins', 11166), ('avenue', 11166), ('ideology.RELATE
D', 11165), ('MY', 11165), ('civilisations.The', 11164), ('slide', 11164), ('myths', 11163), ('ROMANCE', 11163), ('Aztec', 1116
2), ('letters', 11162), ('warrior', 11161), ('industry.By', 11161), ('pantheon.Wonder', 11160), ('boom', 11160), ('Universes',
11159), ('occurred', 11159), ('Loki', 11158), ('girls', 11158), ('Norse', 11157), ('mashed', 11157), ('legends', 11156), ('boy
s', 11156), ('further.Mythology', 11155), ('Westerns', 11155), ('Noir', 11154), ('COWBOY', 11154), ('character.From', 11153),
('ROMANCES', 11153), ('therefore', 11152), ('light-hearted', 11152), ('loves', 11151), ('RANGELAND', 11151), ('comics.Batman',
11150), ('LOVE', 11150), ('normally', 11149), ('for-the-most-part', 11149), ('atmosphere', 11148), ('weary', 11148), ('MembersT
he', 11147), ('nameless', 11147), ('Kids', 11146), ('forged', 11146), ('Forgotten', 11145), ('electrify', 11145), ('emulate.REL
ATED', 11144), ('balm', 11144), ('sequences', 11143), ('storm', 11143), ('underneath', 11142), ('sort-of', 11142), ('survives',
11141), ('cloud', 11141), ('represent', 11140), ('voices', 11140), ('NBAOf', 11139), ('lingered', 11139), ('Matched', 11138),
('decency', 11138), ('Genres', 11137), ('values.With', 11137), ('ComicsDon', 11136), ('ilk', 11136), ('ComicsIndie', 11135),
('rationing', 11135), ('ComicsImage', 11134), ('Korean', 11134), ('low-key', 11133), ('1951', 11133), ('quarterly', 11132), ('N
urse', 11132), ('intro', 11131), ('moniker', 11131), ('pointed', 11130), ('Atlas', 11130), ('InformationFinally', 11129), ('BAT
TLE', 11129), ('KrewMore', 11128), ('Nellie', 11128)]
Accuracy: 0.92
Precision macro: 0.93
Recall macro: 0.92
No. of terms after rfe: 100
```

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)