NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

# CZ4034 - Information Retrieval

## Assignment Report

**Group: 28**


| Group Members | Matriculation Number |
|---|---|
| Chua Wen Rong, Jonathan | U2021875L |
| Mok Ngai Yiu, Enoch | U2021641D |
| Singh Aishwarya | U1923952C |
| Tan Wen Xiu | U1921771H |


**Submitted on: 9th April 2023**

# Table of Contents

# 1. Project Overview

## 1.1. Motivation

In recent years, the gaming industry has been growing rapidly. This trend was witnessed during the Covid-19 pandemic, which confined people to their homes. Ever since, the number of gamers in the gaming industry has increased significantly. Popular Role Playing Games (RPGs) such as Elden Ring and God of War have sold over 16 million units since launch with earnings reaching more than 1.1k percent profits. Popular RPG games are usually paid for, which ranges from prices between $30 to $100.

Twitter is a popular social media platform where gamers interact with each other and freely share their opinions on the games. Therefore, we will extract reviews on RPGs from Twitter and leverage sentiment analysis techniques to gain insights about the general sentiments towards these games. Through this project, we hope to provide valuable insights into the overall reviews of these RPGs.

## 1.2. Goals

The goal of our information retrieval system is to equip our users with:

- The sentimentality of various RPG (Positive, Neutral and Negative)
- Decision making capabilities on whether to purchase a particular RPG

## 1.3. Choice of RPGs

There are a large number of RPGs in the market. For this project, we crawled, indexed and classified a few RPGs which were featured in multiple reviews and are relatively popular amongst gamers in recent years.

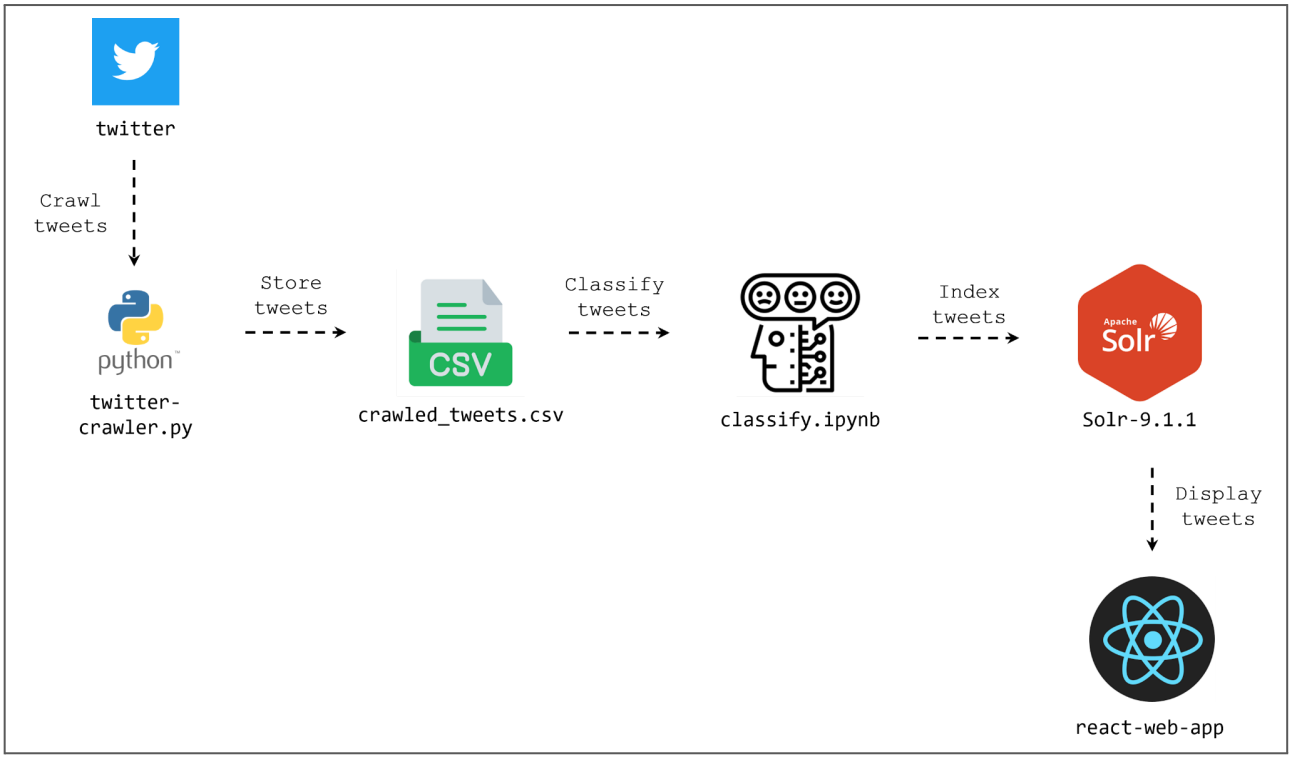
The RPGs that are crawled and indexed in our system include:

- Hogwarts Legacy (HGL)
- Stardew Valley (SDV)
- God of War (GOW)
- Cyberpunk 2077 (Cyberpunk)
- Monster Hunter World (MHW)
- Elden Ring (ELR)

### 1.4. Framework

Our information retrieval system employs the framework shown in *Figure 1*.

***Figure 1: Framework used for information retrieval system***




The list below presents the required deliverables for the project -

1. **Link to the video presentation**: https://youtu.be/ZNdU3hZTPOY

2. **Link to crawled data + queries + results + evaluation dataset + classification results**: https://drive.google.com/file/d/1xLCxgcz3q001uOQEnK1pIQaChPkxsahX/view?usp=share_link

3. **Link to source code**: https://drive.google.com/file/d/111TLUviPZ9vihB8FSB8o-dd9x3XWtmb4/view?usp=share_link

## 2. Data Crawling

This section covers part 4.1 of the assignment – data crawling. Twitter was chosen as the social media platform where we crawled tweets on various RPGs.

### 2.1. Approach - How we crawled the corpus

To crawl tweets from Twitter, we used a python library, **`snscrape`**, which consists of a myriad of scrapers including Twitter.

```
# Example Code:
import snscrape.modules.twitter as sntwitter

(sntwitter.TwitterSearchScraper("#QUERY from:USER lang:en
since:2023-02-10 until:2023-02-19").get_items()
```

With reference to the above code fragment, we constrained our query search using various flags such as:

1. User: **`from:USER`**
2. Language: **`lang:en`**
3. Time Period: **`since:2023-02-10 until:2023-02-19`**

From the query response, we retrieved the following information from the tweet:

1. username
2. verified
3. followersCount
4. rawContent
5. date
6. replyCount
7. retweetCount
8. likeCount
9. url
10. hashtags

To ensure that we are able to scrape relevant tweets more effectively, we used various hashtags and user account names to query twitter for tweets. The specific hashtags used for the crawling can found in **Appendix 8**.

The crawled data is then stored in the Apache Solr database. Solr is a popular open-source tool that provides full-text search capabilities. Solr also provides a RESTful API which allows us to perform queries on our indexed data. The various index strategies are covered in **Section 3 (Indexing)**.

### 2.2. Use Cases - Potential Queries

Users of our information retrieval system will be able to perform queries to find out about the following:

- Sentimentality of tweet about a RPG
- Positive and negative feedbacks of a RPG
    - In terms of graphics, game mechanics, bugs, etc.
- Developers of the game

Some potential and specific queries that the user may wish to search on our system are:

- “Is hogwarts legacy fun?”
- “Are there bugs in cyberpunk?”
- “Developers of elden ring”

Relevant tweets will be returned to the user, providing them with useful information to make informed decisions, for example, before purchasing a specific RPG.

### 2.3. Analysis of Crawled Corpus

With the aid of SNScrape, the team was able to collectively scrape over 94,000 tweets on the chosen RPGs. After crawling, it was observed that a number of tweets across RPGs consisted primarily of hashtags and links to twitch live streams. There were also many tweets that had unrelated content (nothing to do with the chosen RPG), which could have potentially affected the performance of our system’s classification as well as the efficiency of our search engine. Therefore, a number of pre-processing steps were performed before classification and indexing to better cater to the goals of our system. This has been further described in **Section 4.1** of the report.

***Table 1: Insights about the crawled tweets (corpus)***

| Type of Statistics | Count |
|---|---|
| Total number of tweets (records) crawled | 94565 |
| Total number of tweets (records) after pre-processing | 79550 |
| Total number of words in the corpus | 1804962 |
| Total number of unique words in the corpus | 264232 |
| Number of RPGs chosen | 6 |
| Number of labels/sentiments | 3 |

In addition to this, the figures below depict some important illustrations of the crawled corpus that helped the team decide on appropriate pre-processing steps. Additional figures have been included in the appendix.

***Figure 2: Frequency Distribution of games before preprocessing (left) and after preprocessing (right)***

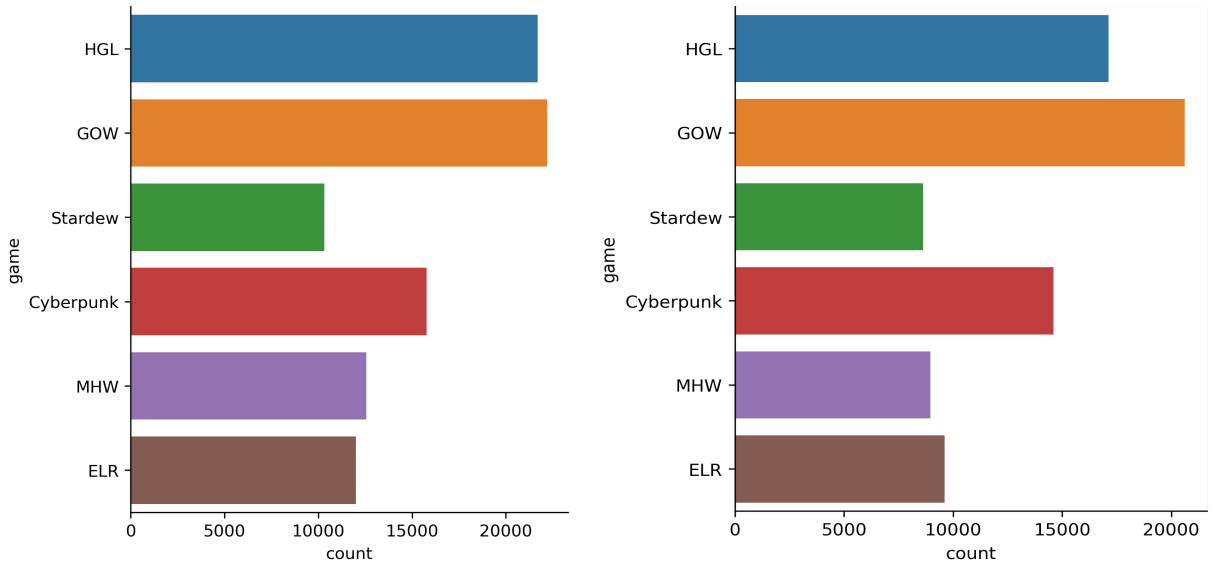
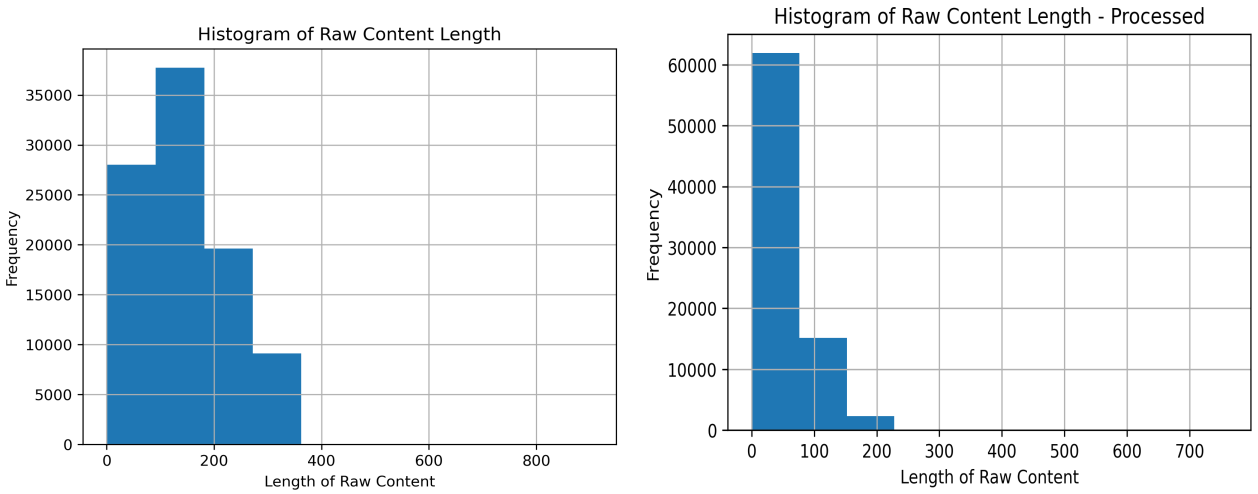
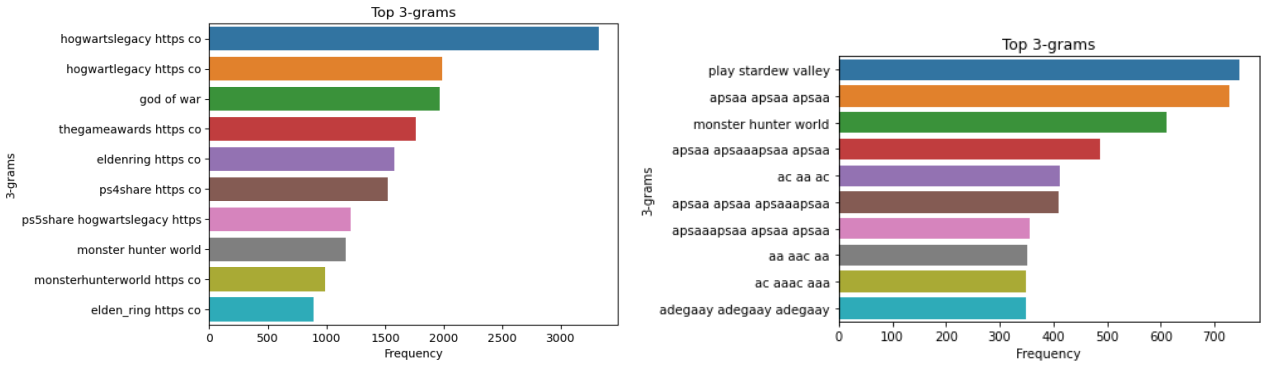



***Figure 3: Distribution of tweet length before preprocessing (left) and after preprocessing (right)***



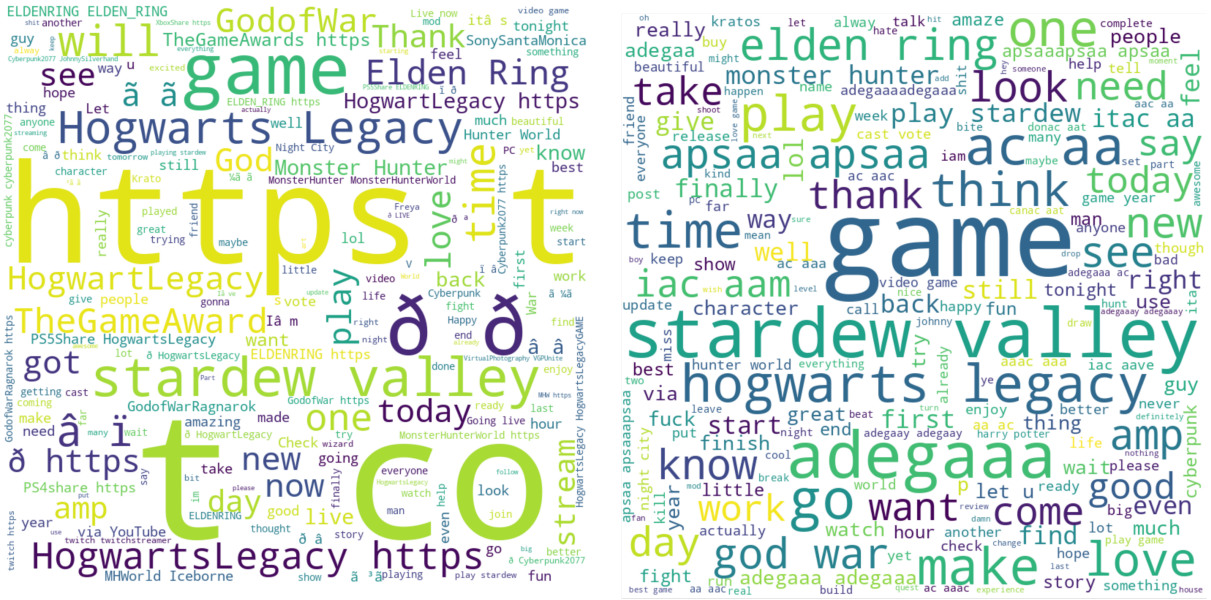

Shown in *Figure 3*, the number of long tweets after preprocessing has greatly decreased. This could be due to the steps that we took during preprocessing which included removal of special characters and words.

***Figure 4: Top-3 grams before and after preprocessing***



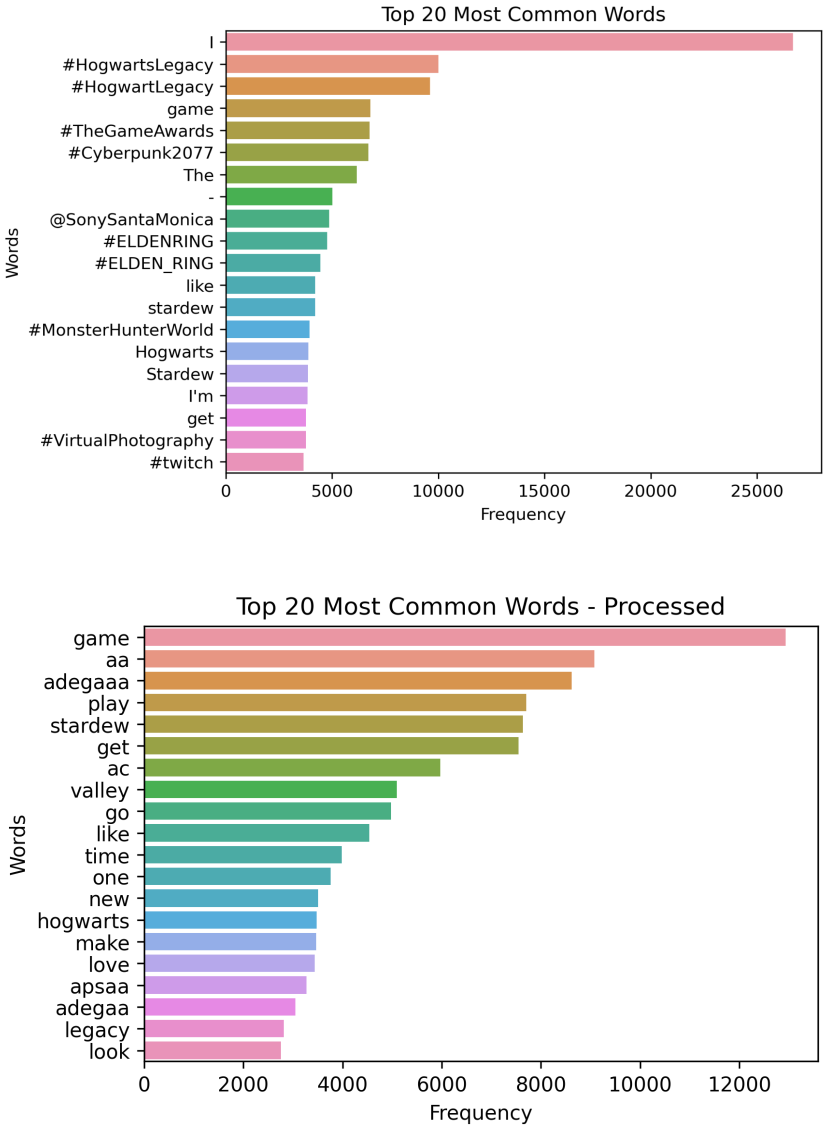

Referring to *Figure 4*, the top-3 grams before preprocessing mostly included the “https” term, which is the header for a url link. After preprocessing and removing these links, the top-3 grams did not include this term.

***Figure 5: Wordcloud - Top 200 words before and after preprocessing***




We used wordclouds to find out the top 200 most frequent words in our corpus and compared the wordclouds before and after preprocessing. Compared to the wordcloud on the left in *Figure 5* which has a lot of noise, the preprocessed corpus is cleaner.

***Figure 6: Most frequent words before and after preprocessing***




## 3. Indexing

This section covers part 4.2 of the assignment - Indexing. As introduced in **Section 2.1** of this report, we used Apache Solr (version 9.1.1) as the backend database to index and store our crawled tweets. Solr provides many useful indexing strategies for tokenization, stemming, stopword removal and spelling suggestions.

Additionally, Solr uses Lucene search library in its search engine implementation, which uses a variant of the Term Frequency - Inverse Document Frequency (TF-IDF) retrieval algorithm for ranked retrieval.

### 3.1. Approach

The crawled tweets are loaded into the Solr for indexing with the field names as shown in *Figure 7*.

***Figure 7: Fields used for indexing into Solr***

```
<field name="TextBlob_Analysis" type="plongs"/>
<field name="TextBlob_Polarity" type="pdoubles"/>
<field name="TextBlob_Subjectivity" type="pdoubles"/>
<field name="_username" type="text_general"/>
<field name="class" type="plongs"/>
<field name="date" type="pdates"/>
<field name="final_class" type="pdoubles"/>
<field name="followersCount" type="plongs"/>
<field name="game" type="text_general"/>
<field name="hashtags" type="text_general"/>
<field name="id" type="string" multiValued="false" indexed="true" required="true" stored="true"/>
<field name="inference" type="plongs"/>
<field name="labels" type="pdoubles"/>
<field name="likeCount" type="pdoubles"/>
<field name="processed" type="text_general"/>
<field name="rawContent" type="text_gen_preprocess"/>
<field name="replyCount" type="plongs"/>
<field name="retweetCount" type="plongs"/>
<field name="spellcheck" type="text_general" multiValued="true" indexed="true" stored="false"/>
<field name="suggest" type="suggest_title" multiValued="true" indexed="true" stored="false"/>
<field name="url" type="text_general"/>
<field name="username" type="text_general"/>
<field name="vader_comp" type="pdoubles"/>
<field name="vader_neg" type="pdoubles"/>
<field name="vader_neu" type="pdoubles"/>
<field name="vader_pos" type="pdoubles"/>
<field name="vader_preprocessing_text" type="text_general"/>
<field name="verified" type="booleans"/>
```

#### 3.1.1. Index Preprocessing

The content of the tweets are pre-processed with various filter strategies during indexing. The series of filters applied to the content of the tweet can be visualised using Solr's analysis tool.

Using the following tweet as an example:

***"The Pathless looks amazingly gorgeous #TheGameAwards"***

1. **Tokenizer** - *WhitespaceTokenizerFactory*

   The content of the tweet will first be tokenized, removing whitespaces in between words, such that each word can be stored in a lexical representation.

| WT | text | The | Pathless | looks | amazingly | gorgeous | #TheGameAwards |
|---|---|---|---|---|---|---|---|
| | raw_bytes | [54 68 65] | [50 61 74 68 6c 65 73 73] | [6c 6f 6f 6b 73] | [61 6d 61 7a 69 6e 67 6c 79] | [67 6f 72 67 65 6f 75 73] | [23 54 68 65 47 61 6d 65 41 77 61 72 64 73] |
| | start | 0 | 4 | 13 | 19 | 29 | 38 |
| | end | 3 | 12 | 18 | 28 | 37 | 52 |
| | positionLength | 1 | 1 | 1 | 1 | 1 | 1 |
| | type | word | word | word | word | word | word |
| | termFrequency | 1 | 1 | 1 | 1 | 1 | 1 |
| | position | 1 | 2 | 3 | 4 | 5 | 6 |

Note: The words have been tokenized by removing whitespaces.

2. **Case Folding** - *LowerCaseFilterFactory*

   The words are then converted to lowercase to decrease the size of the dictionary as we do not require differentiating words between lower and upper cases.

| LCF | text | the | pathless | looks | amazingly | gorgeous | #thegameawards |
|---|---|---|---|---|---|---|---|
| | raw_bytes | [74 68 65] | [70 61 74 68 6c 65 73 73] | [6c 6f 6f 6b 73] | [61 6d 61 7a 69 6e 67 6c 79] | [67 6f 72 67 65 6f 75 73] | [23 74 68 65 67 61 6d 65 61 77 61 72 64 73] |
| | start | 0 | 4 | 13 | 19 | 29 | 38 |
| | end | 3 | 12 | 18 | 28 | 37 | 52 |
| | positionLength | 1 | 1 | 1 | 1 | 1 | 1 |
| | type | word | word | word | word | word | word |
| | termFrequency | 1 | 1 | 1 | 1 | 1 | 1 |
| | position | 1 | 2 | 3 | 4 | 5 | 6 |

Note: All the words have been converted to lowercase.

3. **Stopword** - *StopFilterFactory*

   Stopwords will be removed and excluded from the dictionary, as these words do not provide useful information in our use case.

| SF | text | | pathless | looks | amazingly | gorgeous | #thegameawards |
|---|---|---|---|---|---|---|---|
| | raw_bytes | | [70 61 74 68 6c 65 73 73] | [6c 6f 6f 6b 73] | [61 6d 61 7a 69 6e 67 6c 79] | [67 6f 72 67 65 6f 75 73] | [23 74 68 65 67 61 6d 65 61 77 61 72 64 73] |
| | start | | 4 | 13 | 19 | 29 | 38 |
| | end | | 12 | 18 | 28 | 37 | 52 |
| | positionLength | | 1 | 1 | 1 | 1 | 1 |
| | type | | word | word | word | word | word |
| | termFrequency | | 1 | 1 | 1 | 1 | 1 |
| | position | | 2 | 3 | 4 | 5 | 6 |

Note: The stopword *"the"* has been removed from the document.

4. **Stemming** - *SnowballPorterFilterFactory*

   To further reduce the size of our dictionary in Solr for more efficient retrieval, we used the porter stemmer algorithm to stem the words.

| SF | text | pathless | look | amaz | gorgeous | #thegameaward |
|---|---|---|---|---|---|---|
| | raw_bytes | [70 61 74 68 6c 65 73 73] | [6c 6f 6f 6b] | [61 6d 61 7a] | [67 6f 72 67 65 6f 75 73] | [23 74 68 65 67 61 6d 65 61 77 61 72 64] |
| | start | 4 | 13 | 19 | 29 | 38 |
| | end | 12 | 18 | 28 | 37 | 52 |
| | positionLength | 1 | 1 | 1 | 1 | 1 |
| | type | word | word | word | word | word |
| | termFrequency | 1 | 1 | 1 | 1 | 1 |
| | position | 2 | 3 | 4 | 5 | 6 |
| | keyword | false | false | false | false | false |

   Note: The word “amazingly” has been stemmed and stored as “amaz”.

5. **N-Gram** - NGramFilterFactory

   The N-gram filter is applied to the words, which can be used for spelling correction.

| NGTF | text | pa | pat | path | pathl | pathle | pathles | pathless |
|---|---|---|---|---|---|---|---|---|
| | raw_bytes | [70 61] | [70 61 74] | [70 61 74 68] | [70 61 74 68 6c] | [70 61 74 68 6c 65] | [70 61 74 68 6c 65 73] | [70 61 74 68 6c 65 73 73] |
| | start | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| | end | 12 | 12 | 12 | 12 | 12 | 12 | 12 |
| | positionLength | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | type | word | word | word | word | word | word | word |
| | termFrequency | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | position | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| | keyword | false | false | false | false | false | false | false |

   Note: N-gram filter has been applied to the word *“pathless”*, which is then stored in the dictionary as *“pa”, “pat”, “path”, “pathl”, “pathle”, “pathles”, pathless”*.

### 3.1.2. Query Preprocessing

Similarly, user’s queries are also passed through a series of similar filters, as follows:

1. **Tokenizer** - *WhitespaceTokenizerFactory*
2. **Case Folding** - *LowerCaseFilterFactory*
3. **Stopword** - *StopFilterFactory*
4. **Stemming** - *SnowballPorterFilterFactory*

Notice that the N-gram filter is not applied on the query, because doing so will split the query into multiple sets. This will in turn cause Solr to perform document retrieval based on each set of the query, resulting in the retrieval of non-relevant documents.

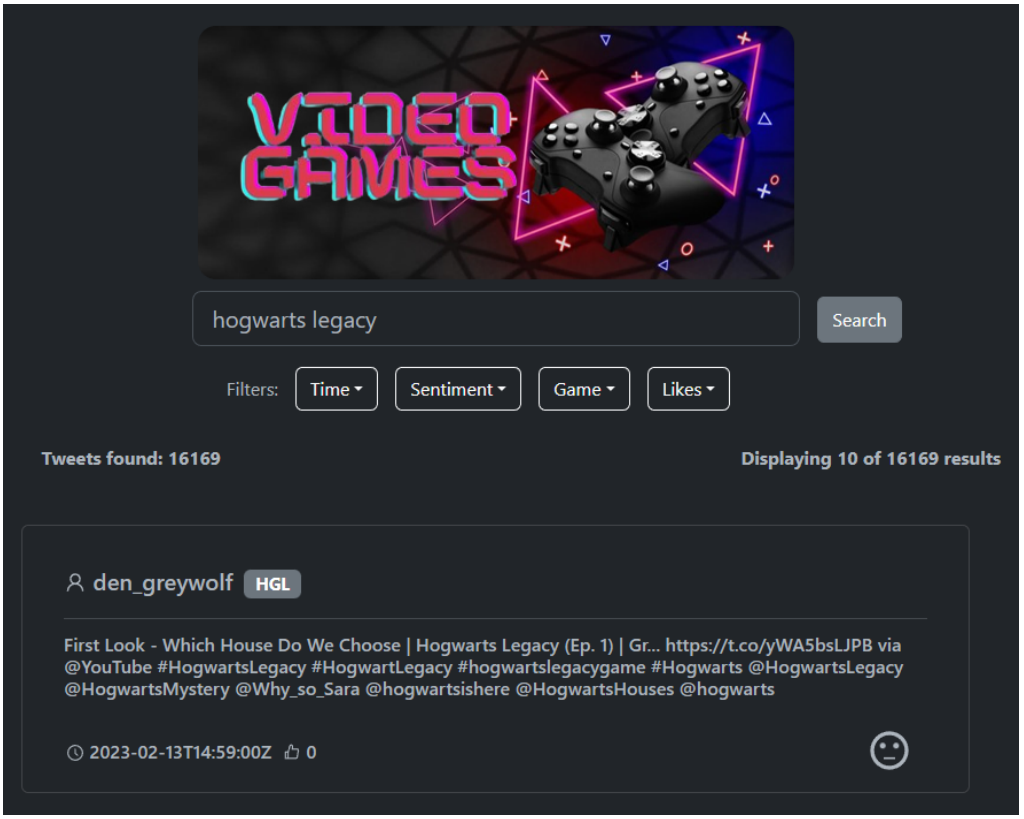
For example, if the user’s query is *“hogwarts”* and assuming we parsed this query into a tri-gram filter. It will split the original query into *“hog”, “ogw”, “gwa”, “war”, “art”, “rts”*. This will result in the retrieval of irrelevant documents based on *“hog”, “war”* and *“art”*, in response to the query *“hogwarts”*.

Furthermore, these filters are required to prevent user's queries from failing to fetch any documents.

For example, the user's query may contain words with capital letters, and if we do not convert them to lower-case, we would not be able to fetch any documents. This is because the words in the documents were indexed into the dictionary with lower-case letters.

## 3.2. Search Engine User Interface

***Figure 8: Search Engine User Interface***




With reference to *Figure 8*, our search engine user interface (UI) encompasses a simplistic design with a search field and multiple filters, providing users with a myriad of ways to query and organise tweets fetched from Solr database.

### 3.2.1. Query Component

RESTful APIs were used to query the Solr database to retrieve documents based on various parameters.

An example URL query is shown below:

***http://127.0.0.1:8983/solr/CZ4034/select?q=rawContent:(hogwarts AND legacy)&start=0&rows=10***

Upon receiving the query *"hogwarts legacy"* from the user, our search engine UI will first split them up into individual words. It will then utilise boolean algebraic manipulation (by adding the boolean *"AND"* in between the split words) to retrieve the relevant documents.

In this case, the boolean query *"hogwarts AND legacy"* will match and retrieve all documents that contain both "hogwarts" and "legacy" in the Solr database.

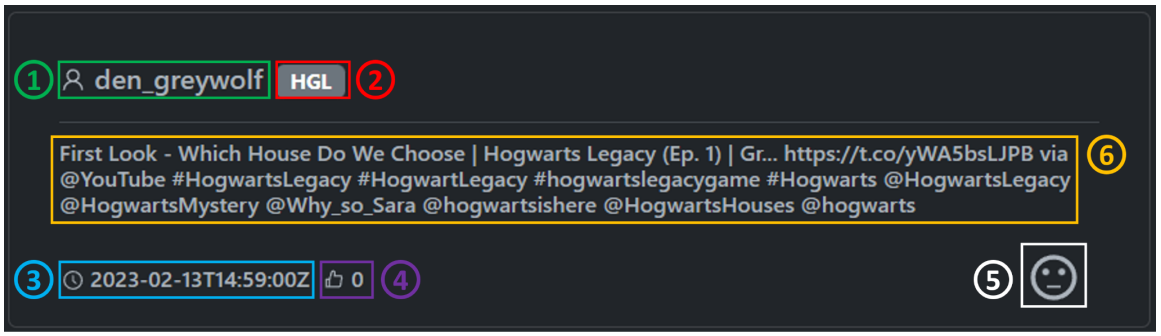
***Figure 9: Query response from Solr database***


```
{
  "responseHeader":{
    "status":0,
    "QTime":1,
    "params":{
      "q":"rawContent:(hogwarts AND legacy)",
      "indent":"true",
      "q.op":"OR",
      "useParams":"",
      "_":"1680113755940"}},
  "response":{"numFound":16169,"start":0,"numFoundExact":true,"docs":[
      {
        "verified":[false],
        "followersCount":[7],
        "rawContent":"First Look - Which House Do We Choose | Hogwarts Legacy (Ep. 1) | Gr... https://t.co/yWA5bsLJPB via @YouTube #HogwartsLegacy #HogwartLegacy
        "date":["2023-02-13T14:59:00Z"],
        "replyCount":[0],
        "retweetCount":[0],
        "likeCount":[0.0],
        "url":["https://twitter.com/den_greywolf/status/1625147771995971585"],
        "hashtags":["['HogwartsLegacy', 'HogwartLegacy', 'hogwartslegacygame', 'Hogwarts']"],
        "game":["HGL"],
        "processed":["['first', 'look', 'which', 'house', 'do', 'we', 'choose', 'hogwarts', 'legacy', 'ep', '1', 'gr', 'https', 't', 'co', 'wa5bsljpb', 'via']"],
        "vader_preprocessing_text":["first look - which house do we choose | hogwarts legacy (ep. 1) |   via          "],
        "vader_neg":[0.0],
        "vader_neu":[1.0],
        "vader_pos":[0.0],
        "vader_comp":[0.0],
        "class":[0],
        "TextBlob_Subjectivity":[0.333333333],
        "TextBlob_Polarity":[0.25],
        "TextBlob_Analysis":[1],
        "inference":[0],
        "final_class":[0.0],
        "id":"e390d526-4533-486a-9092-66231fbc82d9",
        "_username":["den_greywolf"],
        "_version_":1761542253241696256},
```


The response is then generated as shown in *Figure 9* and sent back to the UI to be displayed.

### 3.2.2. Display Tweet Result Component of UI

***Figure 10: Search Engine User Interface***




Documents are retrieved from Solr and displays the following columns to the user (with reference to *Figure 10*):

1. **Username** (represented by the profile icon)
    - User who posted the tweet
2. **RPG label** (represented by a badge
    - Which game the tweet was crawled from
3. **Date & Time** (represented by a clock icon)
    - Date/time that tweet was posted
4. **Likes** (represented by thumbs up icon)
    - Number of likes for the tweet
5. **Sentiment** (represented by emoji icon)
    - Sentimentality of the tweet
        1. Happy face - positive
        2. Neutral face - neutral
        3. Sad face - negative
6. **Tweet Content**
    - Content of the user's tweet

### 3.2.3. Filter Component of UI

We have implemented several filters to allow user to retrieve documents based on:

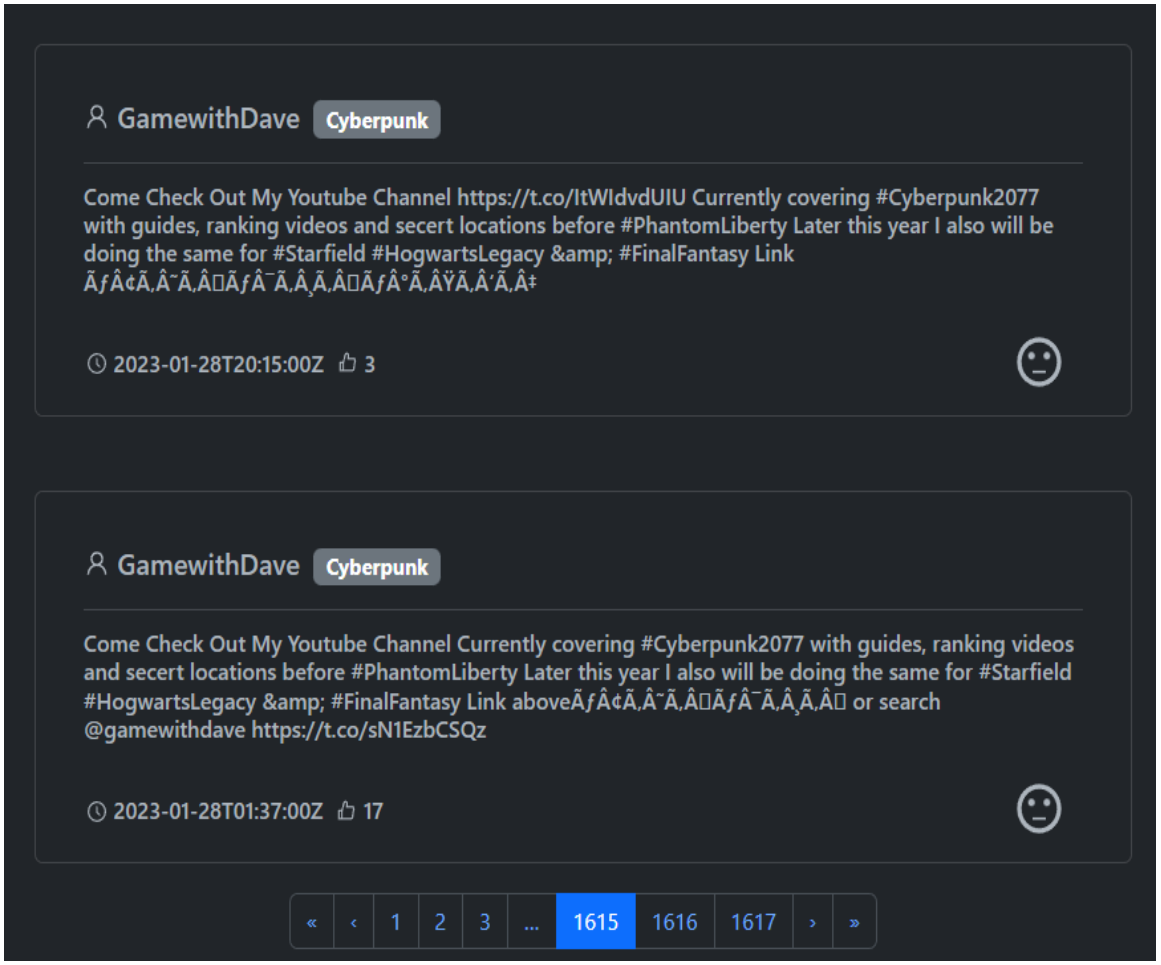
1. **Time** - most recent / least recent tweet
2. **Sentiment** - type of sentiment classified for the tweet (positive, neutral, negative)
3. **Game** - six types of RPG (Hogwarts Legacy, Stardew Valley, Elden Ring, Monster Hunter World, God of War, Cyberpunk 2077)
4. **Likes** - number of likes the tweet post has

The filters were implemented by appending additional parameters to the query request to the Solr database. The following parameters were used for the filter implementations:

1. **Time** - `&sort="date asc/desc"`
2. **Sentiment** - `&fq:({sentiment})`
    - sentiment = {1, 0, -1}
3. **Game** - `&fq:({game})`
    - game = {HGL, Stardew, ELR, MHW, GOW, Cyberpunk}
4. **Likes** - `&sort="likeCount asc/desc"`

### 3.2.4. Pagination Component

***Figure 11: UI Pagination Component***




Our search engine queries and displays up to 10 documents at a time to the user. With the pagination component (as shown in *Figure 11*), the UI allows users to navigate and browse through more tweets retrieved from the database.

This feature was implemented by appending the *&start={start} &rows={end}* parameters to the URL query. The *"start"* parameter specifies the starting document id and the *"end"* specifies the ending document id. Using simple calculations, we are able to compute the starting and ending document id to be retrieved for each page number that the user clicks on.

This feature is important as it allows for scalability avoiding querying and storing all the documents retrieved in the memory storage of the UI. It is impractical to fetch all the documents from the database and store them on the UI, since the storage resources on the client-side are limited. This is particularly useful if there are large amounts of documents stored in the database.

### 3.3. Results of Five Queries

***Table 2: Results of the five queries***

| Query | Time taken (ms) | # Results | Top Tweet |
|---|---|---|---|
| “great game” | 5 | 667 | *@corybarlog @SonySantaMonica OMG what a Game you have created. I want the next chapter. Great gameplay, great game design, Great story & great characters.* |
| “bad game” | 2 | 280 | *Scrolling through youtube listening to #ELDEN_RING critiques and alot of them are: game bad cause too hard, game bad because idk where to go, or game bad because its a souls game in general. And these to me are the weakest reasons to bad mouth the game.* |
| “hogwarts is fun” | 4 | 524 | *Hogwarts Legacy funny knight encounter https://t.co/gHwf7zpZUB via @YouTube This game is so fun! check out this funny/rare moment while i was just walking around Hogwarts. #funny #HogwartLegacy* |
| “amazing game” | 4 | 740 | *And boom!Thanks to everyone who worked on this amazing game!God of war 2 was my favorite god of war game but now this game is my most favorite one. You have done an awesome job on this game. The story, the gameplay. Everything is amazing in this game! @corybarlog @SonySantaMonica https://t.co/uVNj7RWG9F* |
| “favourite game of all time” | 11 | 26 | *Favourite games:Stardew Valley, Slime Rancher, Little Nightmares Least favourite:Subnautica(i like it, just don't play it much) Guilty Pleasure:Stardew Valley (i killed so much time with it) Most recently played:Inscryption @WE____EP @Louis_2507 i think you haven't done this yet* |

### 3.4. Innovations for Indexing and Ranking

To further improve the querying capabilities of system, which allows users to perform more accurate searches, we have performed the following enhancements to our system:

1. Spell Checks - suggests correct spelling of word when user misspell the words while querying the system
2. Filters - provide users more options to organise retrieved data by time, sentiment, game and number of likes.

#### 3.4.1. Spell Checking Capabilities

The spellchecker provides suggestions on misspelt words in the query. The component uses the Levenshtein edit distance algorithm, which computes the number of changes required to transform a word to another. The lower the value of the edit distance, the more similar a word is to the other.

To implement the spellchecker, we have added the following code snippets into the *solrconfig.xml* file:

1. Spell Check Request Handler -
    a. Handles query requests for words that require spell checking.
    b. The code snippet is shown in *Figure 12*.

***Figure 12: Spellchecker Request Handler Configuration***

```
<requestHandler name="/spell" class="solr.SearchHandler" startup="lazy">
    <lst name="defaults">
      <str name="spellcheck.dictionary">default</str>
      <str name="spellcheck">on</str>
      <str name="spellcheck.extendedResults">true</str>
      <str name="spellcheck.count">10</str>
      <str name="spellcheck.alternativeTermCount">5</str>
      <str name="spellcheck.maxResultsForSuggest">5</str>
      <str name="spellcheck.collate">true</str>
      <str name="spellcheck.collateExtendedResults">true</str>
      <str name="spellcheck.maxCollationTries">10</str>
      <str name="spellcheck.maxCollations">5</str>
    </lst>
    <arr name="last-components">
      <str>spellcheck</str>
    </arr>
</requestHandler>
```

2. Spell Check Component -
    a. Component which leverages Solr's edit distance implementation to calculate the similarity between terms.
    b. The code snippet is shown in *Figure 13*.

***Figure 13: Spellchecker Component Configuration***

```
<searchComponent name="spellcheck" class="solr.SpellCheckComponent">
  <lst name="spellchecker">
    <str name="classname">solr.IndexBasedSpellChecker</str>
    <!-- field to use -->
    <str name="field">spellcheck</str>
    <!-- buildOnCommit|buildOnOptimize -->
    <str name="buildOnCommit">true</str>
    <!-- $solr.solr.home/data/spellchecker-->
    <str name="spellcheckIndexDir">./spellchecker</str>
    <str name="accuracy">0.7</str>
    <float name="thresholdTokenFrequency">.0001</float>
  </lst>
</searchComponent>
```

In our implementation, if the query from the user fetches less than five documents from the Solr database, we perform a request to the spell check with the user's original query. The Solr database will then compute the edit distance value between the query and all of the words indexed in its dictionary, and return suggested words with low edit distances.

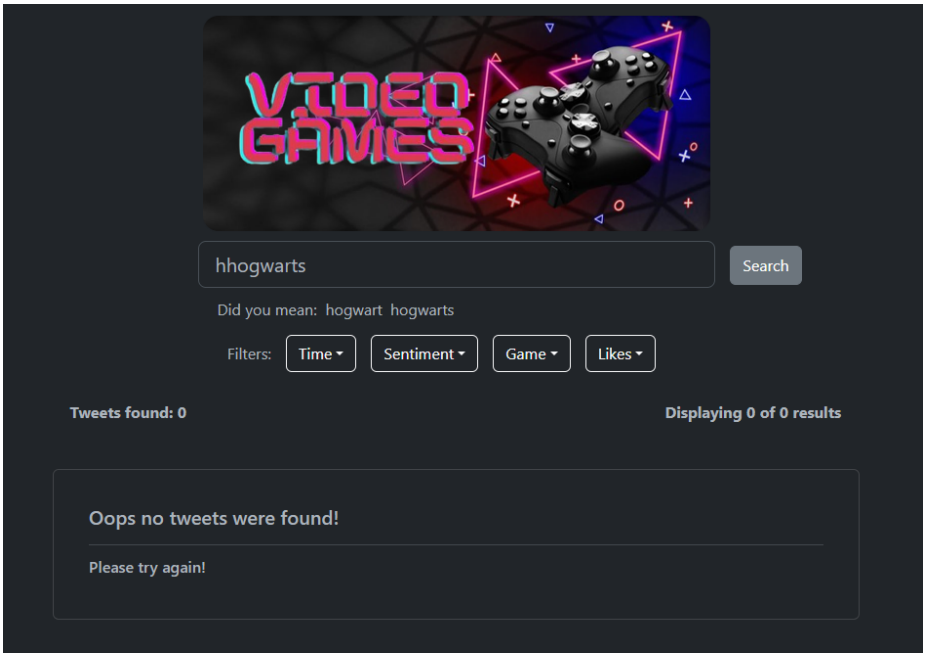
With reference to *Figure 14*, the misspelt word is "hhogwarts" and the correct words returned are shown under "suggestions".

***Figure 14: Response from querying Solr with incorrect spelling***


```
"response":{"numFound":0,"start":0,"numFoundExact":true,"docs":[]
},
"spellcheck":{
  "suggestions":[
    "hhogwarts",{
      "numFound":5,
      "startOffset":11,
      "endOffset":20,
      "origFreq":0,
      "suggestion":[{
          "word":"hogwarts",
          "freq":4036},
        {
          "word":"hogwart",
          "freq":39},
        {
          "word":"hogwart's",
          "freq":10},
        {
          "word":"hogwartz",
          "freq":7},
        {
          "word":"hogwartsai",
          "freq":12}]}],
  "correctlySpelled":false,
```

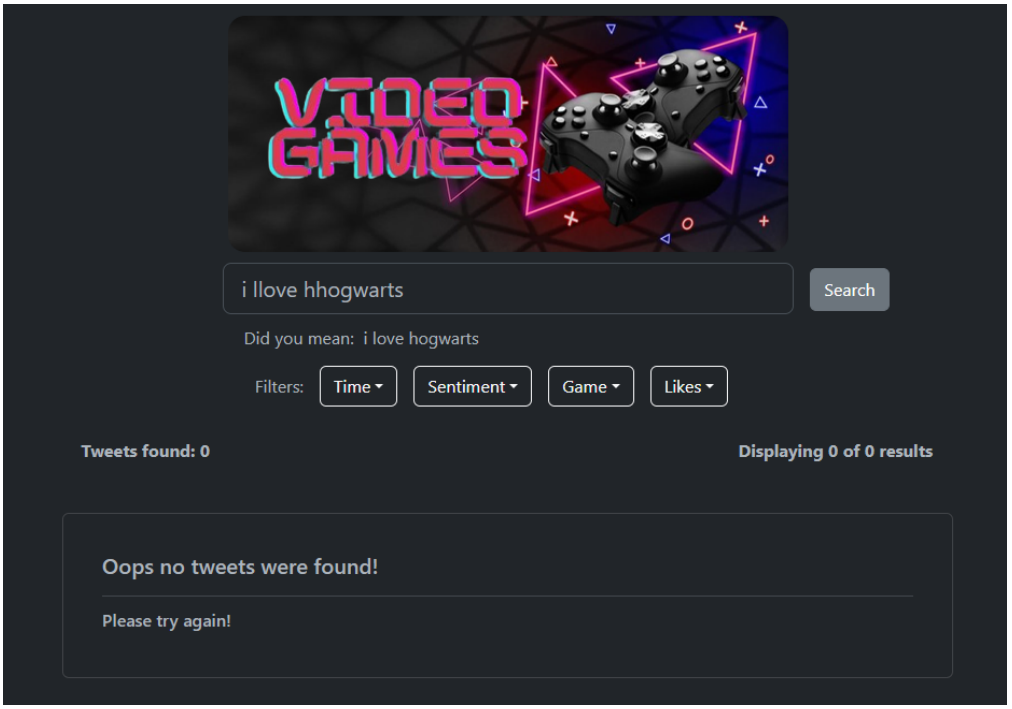

In the search engine UI, “Did you mean: {suggested_word}” will be displayed to the user as shown in *Figure 15*. Only the top two suggested words will be displayed on the UI.

***Figure 15: Display spelling suggestions on UI***



Our implementation also takes into account spelling correction for multi-term searches as shown in *Figure 16*. Notice that there are two misspelt terms, *"llove"* and *"hhogwarts"*, which was corrected to *"love"* and *"hogwarts"*.

***Figure 16: Spelling suggestions for multi-term searches***




### 3.4.2. Filtering Capabilities

We have also implemented several filters which allows users to retrieve queries based on additional metadata of the tweet. For implementation details of the filters, refer to **Section 3.2.3**.

Filters which were implemented includes:

1. **Time** - this filter allows users to retrieve the tweets by the order of most or least recent. This can provide users with information regarding the change in sentimentality of the games over a given timeframe. For example, the RPG Cyberpunk 2077, had poor sentiment when it first released as it was filled with bugs. However, as the bugs were fixed and more content was released for the game, the overall sentiment of the game went up.

2. **Sentiment** - filtering based on sentiments allows users to easily find tweets that are regarding a specific sentimentality. When deciding to purchase a game, users may like to filter based on positive or negative sentiments to weigh the cost and benefits of purchasing the game.

3. **Game** - filtering by the type of RPG, allows users to only fetch tweets that were obtained from that game itself. This is useful as some tweets from

another RPG may contain references to a particular RPG that the user is searching on.

4. **Likes** - providing users with a filter on the number of likes that the tweet receives is important. This is because the number of likes a tweet receives could mean that many people resonate with the tweet, and share the same opinion. For example, a tweet with a positive sentiment and many likes regarding a RPG, could possibly outweigh multiple tweets with negative sentiments and zero number of likes.

Therefore, with the implementation of filters and spell check, our users are able to perform more constrained and accurate searches to retrieve more relevant information about RPGs.

# 4. Classification

This section covers part 4.3 of the assignment - Classification. To obtain the final classified labels for indexing, multiple steps were carried out to ensure that the crawled corpus is suitable for classification and indexing purposes. The subsequent subsections describe in detail the methodologies incorporated to make the crawled corpus suitable for final classification.

## 4.1. Data Preprocessing

Data preprocessing is the process of converting raw data into a cleaned dataset. It is an important part of data preparation, that takes the raw data and transforms it into a format that is more predictable and analyzable to perform the classification task.

This step is essential because the data crawled from Twitter is typically noisy, and therefore, the dataset needs to be preprocessed in order to check for duplicates, hashtags, and other inconsistencies, as detailed below:

### 4.1.1. Removal of duplicates

Duplicates were removed to prevent duplicate tweets from resulting in higher weights than non-duplicated ones, which could affect the model performance, and reduce accuracy.

### 4.1.2. Removal of hashtags, URLs and twitter handles

Hashtags usually contain useful information about the context of a tweet and its content. However, the problem with hashtags is that individual words are not separated with a space. This makes the content difficult to interpret and understand. If the hashtag is made of two or more words, it will remain as noise in the data, and hence, should be removed.

URLs do not provide any information when we try to analyse text from words, especially on Twitter, hence we removed them from the text. Similarly, twitter handles do not give any valuable information because they are not recognized as words that carry any meaning, and were removed.

### 4.1.3. Lowercase characters

Converting all characters to lowercase helps to minimise noise in the data and ensure that the same words are analysed in the same way regardless of their letter case.

#### 4.1.4. Removal of special characters

Special characters including exclamation marks, and characters with accents were also removed from the text as they may not even be in the English language and do not add any information to our analysis.

#### 4.1.5. Emotions replacement

Emojis are a great source of information for sentiment analysis, as they convey the feelings of their writers, hence we chose to not discard them. Instead, we used a mapping function, which replaces each emoji with its corresponding text translation.

#### 4.1.6. Spell Correction

Spelling correction is another important preprocessing technique, especially when working with tweets or comments. Since we see incorrect spelling words mostly in these areas of text, it is essential to replace misspelt words with correctly spelt ones.

#### 4.1.7. Tokenization

Tokenization is the process of breaking up a text document into individual units of meanings, known as tokens. It facilitates the analysis of the text by interpreting the sequence of words and can aid in discerning the meaning of the text.

#### 4.1.8. Lemmatization

Lemmatization is the process of grouping various inflected forms of a word together. By treating words with the same root as synonyms and considering them as a single entity, this approach helps algorithms detect the overall concept of the text, rather than focusing on individual words. Hence, we used lemmatization to reduce the number of unique words in the dataset, thereby enhancing the strength of the model.

### 4.2. Evaluation Dataset

Manually labelling an evaluation dataset entails annotating a portion of the data with ground truth labels to serve as a reference for evaluating the performance of a machine learning model. The following steps were employed in this project:

1. Choose a representative subset of the data that is large enough to offer a credible assessment of the model's performance while being manageable. The diversity of the data, as well as any potential biases or outliers, were considered in the selection. The team decided to use 10% of the shuffled raw, crawled dataset.

2. The next step is to create a labelling strategy that describes which categories or labels will be allocated to each data point. To guarantee that the labels are accurate and dependable, the labelling scheme was made to be clear,

consistent, and unambiguous. The chosen labels were -1 (negative), 0 (neutral) and 1 (positive).

3. Quality control procedures such as inter-annotator agreement and spot-checking were used to assure the correctness and dependability of the labels.

## 4.3. Methodologies

This section covers the different techniques that were covered to perform classification on the preprocessed dataset.

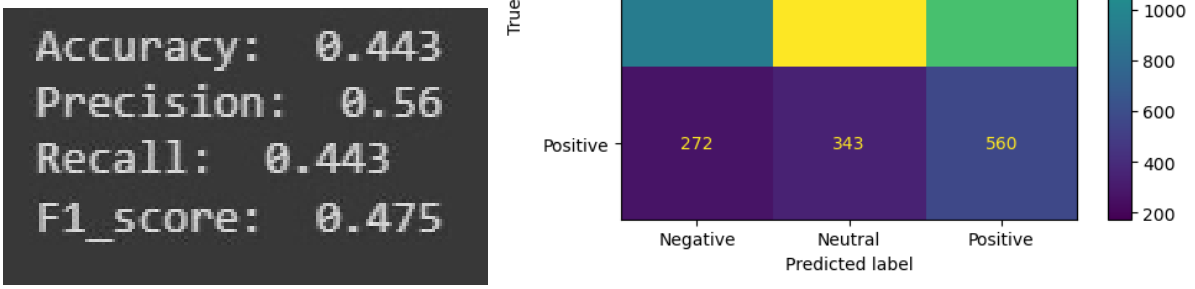
### 4.3.1. Random Forest

Random forest is a classification algorithm based on the aggregation of several decision trees to create a more accurate and robust prediction model.

A decision tree splits the data into smaller subsets based on some criteria, such as a feature value or a threshold. Each split creates a branch in the tree, and each leaf node represents a class label or a numerical value. Using different subsets of data and features, the random forest algorithm creates many decision trees. This reduces overfitting of a single decision tree, and instead averages the predictions of all the decision trees to get the final output.

*Figure 17* represents the results obtained from the Random Forest Model:

***Figure 17: Results of Random Forest***




### 4.3.2. Naive Bayes

Naive Bayes classifiers are easy to implement and train, as they only require a linear number of parameters and can be estimated by tabulating the frequencies of the features in the training data. They are also fast to make predictions, as they only

need to perform simple arithmetic operations. Naive Bayes classifiers can handle both discrete and continuous features, by using different probability distributions for each type of feature. For our project, we modelled features using a multinomial distribution.

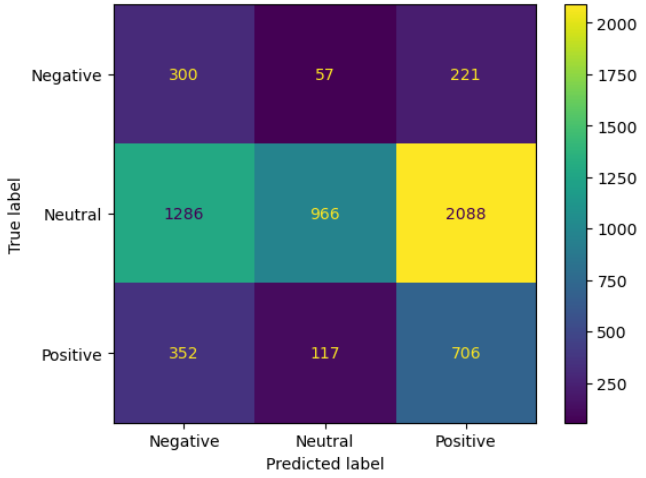
Naive Bayes classifiers are known to perform well in many situations, especially when the data is noisy or incomplete. However, they also have some limitations, such as:

- They may not capture the interactions or dependencies between the features, which may affect the accuracy of the predictions.
- They may suffer from zero-frequency problems, when a feature-class combination does not occur in the training data, resulting in a zero probability estimate.

*Figure 18* represents the results obtained from the Naive Bayes classifier:

***Figure 18: Results of Multinomial Naive Bayes***

```
Accuracy:  0.34062573512114797
Precision:  0.594
Recall:  0.341
F1_score:  0.346
```




### 4.3.3. Valence Aware Dictionary for Sentiment Reasoning (VADER)

VADER is a sentiment analysis tool with a lexicon and rules that are especially built for assessing the sentiment of text posted in social media situations. VADER is trained on a huge corpus of social media material, including Twitter and Reddit, and is intended to handle the platform's particular linguistic traits, such as the usage of slang, acronyms, and emoticons.

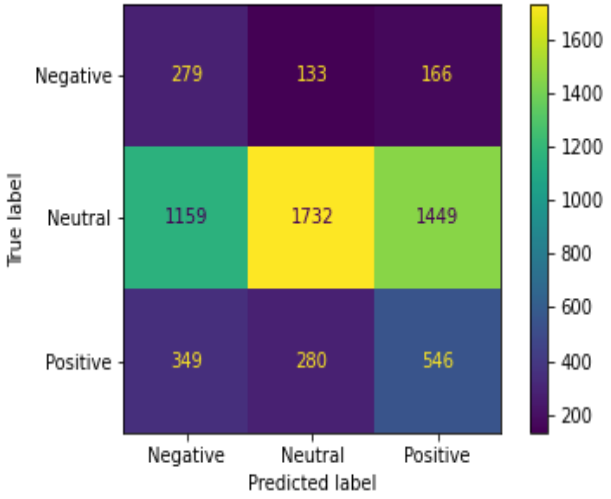
The tool comes with a collection of heuristics that aid in capturing the strength, polarity, and context of sentiment expressions in text. VADER, for example, can detect negation, amplification, and sarcasm in text. VADER assigns sentiment scores to each piece of text, indicating how positive, negative, or neutral the attitude communicated is. The valence ratings of specific lexical characteristics in the text are

combined with heuristics meant to capture the strength, polarity, and context of sentiment expressions to produce these scores.

Employing the VADER model on the dataset yielded the following results (shown in *Figure 19*):

***Figure 19: Results of VADER***

```
Accuracy:  0.4196619071065157
Precision:  0.639
Recall:  0.42
F1_score:  0.466
```




### 4.3.4. TextBlob

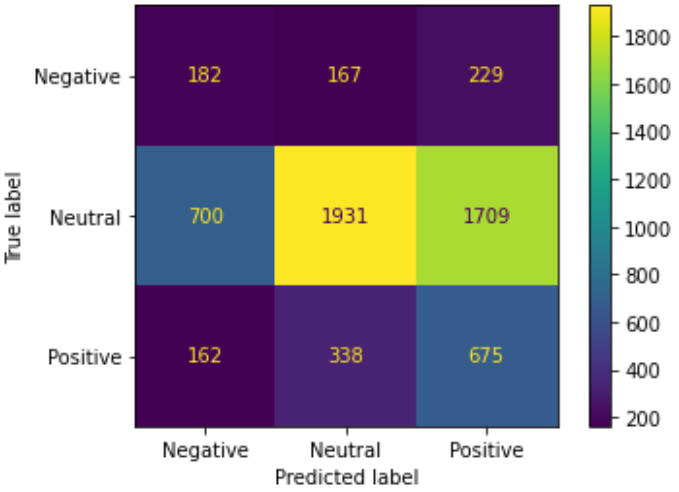
TextBlob is a Python package for text processing. For a given piece of text, TextBlob can provide both polarity scores (ranging from -1 for negative sentiment to +1 for positive sentiment) and subjectivity scores (ranging from 0 for objective text to 1 for subjective text). As a result, it is a handy tool for social media posts.

The identification of subjectivity is a common subtask in the Sentiment Analysis area. Subjective sentences are used to express personal opinion, emotion, or judgement, whereas objective sentences are used to express factual facts. TextBlob computes subjectivity by examining the 'intensity'. The intensity of a word influences whether it modifies the next word. The TextBlob model was used to determine the subjectivity of each tweet. Together with subjectivity, another prominent subtask is polarity identification. Polarity refers to the overall sentiment expressed by a certain sentence, phrase, or word, and polarity detection seeks to separate the viewpoint into "positive" and "negative." The TextBlob model was used to determine the polarity of each tweet.

The results for the same can be illustrated below (shown in *Figure 20*):

***Figure 20: Results of TextBlob***

```
Accuracy:  0.45757426555063185
Precision:  0.631
Recall:  0.458
F1_score:  0.496
```




### 4.3.5. Bidirectional Long Short-Term Memory Neural Network

Since text is a type of sequential data, we use the Bidirectional Long Short-Term Memory (Bi-LSTM) neural networks for the classification. LSTM is a type of Recurrent Neural Network architecture that can remember information for long periods of time. It uses gates, which control the flow of information into and out of the network, allowing the model to maintain a “memory” of past inputs and prevent the network from forgetting relevant information.

As compared to the standard LSTM network that processes the input sequence from start to end, the Bi-LSTM processes the input from both directions, giving the network access to contextual information from both past and future time steps. This provides additional context to the neural network and allows a better learning model on the classification task.

Using keras, we build a Bi-LSTM model with an attention layer and a learning rate of 0.001 with the Adam optimizer. The model was trained on 25 epochs. Figure 21 illustrates the results for the model.

***Figure 21: Results of Bi-LSTM***

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| -1 | 0.90 | 0.83 | 0.86 | 6017 |
| 0 | 0.84 | 0.84 | 0.84 | 5511 |
| 1 | 0.79 | 0.88 | 0.83 | 4382 |
| accuracy | | | 0.85 | 15910 |
| macro avg | 0.84 | 0.85 | 0.84 | 15910 |
| weighted avg | 0.85 | 0.85 | 0.85 | 15910 |

### 4.3.6. RoBERTa

RoBERTa embeddings (Robustly Optimised BERT Pre Training Approach) are a sort of pre-trained language model based on the BERT architecture (Bidirectional Encoder Representations from Transformers). RoBERTa is a transformer-based neural network that was trained on a large corpus of heterogeneous text data, with an emphasis on refining the pre-training procedure to increase embedding quality.

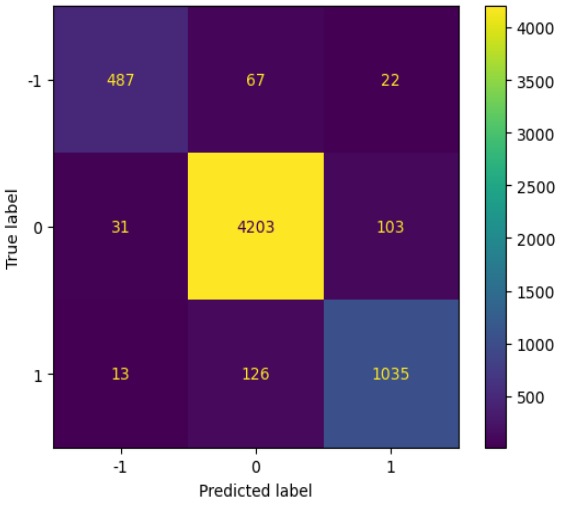
During pre-training, the RoBERTa model employs a masked language modelling (MLM) goal in which a fraction of the input tokens are randomly masked, and the model is taught to predict the masked tokens based on the surrounding context. This method enables the model to acquire rich contextual representations of words and phrases capable of capturing complicated links and meanings. It creates its embeddings by running the input text through the pre-trained RoBERTa model, which generates a vector representation of each word or subword token in the input. These embeddings can subsequently be employed in sentiment classification as features. They are especially beneficial for jobs that need a thorough comprehension of context and semantics because they can record complicated links between words and phrases.

To apply the RoBERTa model for sentiment classification, the team built a FeedForward layer after the RoBERTa transformer output for classification, followed by a fully connected layer. The team then uses a RoBERTa-Base model that has been pretrained on the tweets. Following pretraining, the team discovered that fine-tuning it on a smaller sample of video game tweets was easier.

Figure 22 shows the results for the RoBERTa model.

***Figure 22: Results of RoBERTa***

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| -1.0 | 0.92 | 0.85 | 0.88 | 576 |
| 0.0 | 0.96 | 0.97 | 0.96 | 4337 |
| 1.0 | 0.89 | 0.88 | 0.89 | 1174 |
| accuracy | | | 0.94 | 6087 |
| macro avg | 0.92 | 0.90 | 0.91 | 6087 |
| weighted avg | 0.94 | 0.94 | 0.94 | 6087 |



### 4.4. Innovations for Classification

Additional features can be used to improve the accuracy of the sentiment classification. Other than using only the text data to determine the sentiments of the tweets, we have also experimented the following features to find out their effects on the sentiment classification.

These features include:

1. Tweet statistics
2. Textblob polarity and subjectivity
3. Emotion sentiment analysis

#### 4.4.1. Tweet statistics

On top of using the text information available in tweets for classification, the statistics corresponding to the tweet, such as the number of likes, replies and retweets could be useful to help determine the sentiments of the tweets. For example, a tweet that has more likes may be more likely to contain a positive or negative sentiment (rather than be neutral). We aim to improve the extraction of sentiments from the tweets by introducing the number of likes, replies and retweets as an input alongside the text data.

We build on the Bi-LSTM model introduced in **Section 4.3.5**. An additional dense layer was added to the model to add in the LikeCount, ReplyCount and RetweetCount data in the form of integers so that the model takes into account these additional features.

The model was trained for 25 epochs and the results are depicted below (shown in *Figure 23*):

***Figure 23: Results for Innovation with likes, replies and retweets***

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| -1 | 0.88 | 0.81 | 0.85 | 6017 |
| 0 | 0.77 | 0.86 | 0.81 | 5511 |
| 1 | 0.83 | 0.80 | 0.82 | 4382 |
| accuracy | | | 0.82 | 15910 |
| macro avg | 0.83 | 0.82 | 0.82 | 15910 |
| weighted avg | 0.83 | 0.82 | 0.82 | 15910 |

### 4.4.2. Textblob Polarity and Subjectivity

Textblob is a package that is able to assign a score to the tweets according to their polarity and subjectivity, described in **Section 4.3.4**. These scores can be useful as the model can better understand whether the text is subjective or objective, and the general sentiment of the text, while analysing the raw text itself. The scores were added into the Bi-LSTM neural network for the classification and the results are as follows (shown in *Figure 24*):

***Figure 24: Results for Innovation with Textblob Polarity and Subjectivity***

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| -1 | 0.90 | 0.82 | 0.86 | 6017 |
| 0 | 0.81 | 0.84 | 0.83 | 5511 |
| 1 | 0.79 | 0.86 | 0.83 | 4382 |
| accuracy | | | 0.84 | 15910 |
| macro avg | 0.84 | 0.84 | 0.84 | 15910 |
| weighted avg | 0.84 | 0.84 | 0.84 | 15910 |

### 4.4.3. Emotion Sentiment Analysis

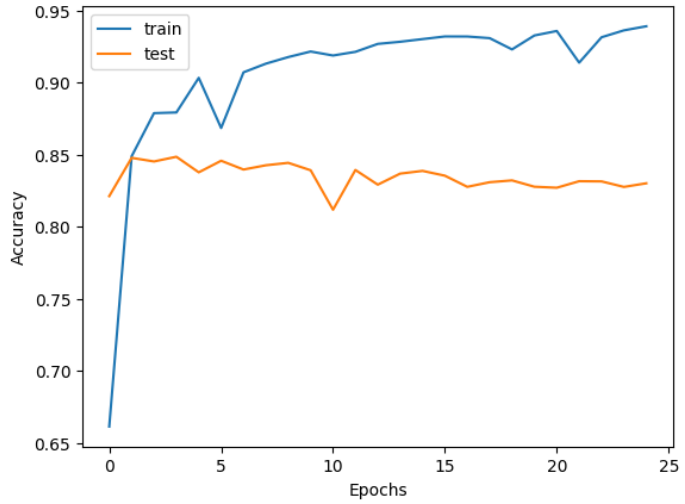
Emotions in text can play a part in allowing us to determine whether the sentiment of the text is positive, negative or neutral, hence we are interested to find out whether including the analysis of emotions in the classification will produce better results.

We used a pre-trained emotion sentiment analysis classification model to classify emotions in the tweets. The model was trained on 6 diverse dataset and predicts Ekman's 6 basic emotions, as well as a neutral class. The emotions are anger, disgust, fear, joy, neutral, sadness and surprise. For a text document, the predicted probability for each of the emotions are determined from 0 to 1 and the emotion that has the highest probability is the label for that text document. The model has been fine-tuned on DistilRoBERTa-base and we used it to predict the emotions present in the tweets.

The scores for each emotion is used as an input for the sentiment classification model as an enhancement to the text input.

***Figure 25: Results for Innovation with Emotion Sentiment Analysis***

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| -1 | 0.89 | 0.80 | 0.84 | 6017 |
| 0 | 0.80 | 0.85 | 0.82 | 5511 |
| 1 | 0.79 | 0.85 | 0.82 | 4382 |
| accuracy | | | 0.83 | 15910 |
| macro avg | 0.83 | 0.83 | 0.83 | 15910 |
| weighted avg | 0.83 | 0.83 | 0.83 | 15910 |




The graph in *Figure 25* shows that as the number of epochs trained increases, the accuracy value increases. It is noteworthy that the accuracy of the training model demonstrates a steady increase, while the validation accuracy reaches a plateau. This observation suggests that the model might be overfitting as the number of epochs increases, which is attributed to its growing familiarity with the training data. Nevertheless, the overfitting of the model on the training data may lead to inaccurate predictions when faced with a new set of unseen data. Hence, there is an increase in training accuracy but not in validation accuracy.

### 4.4.4. Evaluation

The 3 experiments above reflect results that are similar to the original Bi-LSTM model presented in **Section 4.3.5**. The 3 experiments have accuracies ranging from 0.82 to 0.84 while the original model has an accuracy of 0.85. While the new models may not have improved the accuracy from the original model, these experiments are still insightful as they show that the text data still holds more weight within the classification model and the other inputs may not have a great impact on the final classification. Furthermore, overfitting of the model may occur if more inputs are used in the model, causing it to have a high training accuracy but a lower validation accuracy.

Future explorations for these innovations may include other types of features of the texts such as slang detection to more accurately identify the sentiments of the tweets for classification. Other types of deep learning models may also be introduced when making these comparisons to find out what are the important features that may affect the accuracy of sentiment classification.

### 4.5 Classification Evaluation

The RoBERTa model outperformed all other models tested, demonstrating the accuracy and usefulness of Bi-directional LSTM and attention layers in text classification. Additional innovations to the classification models were achieved, as explained in the **Section 4.4**. *Table 3* summarises the results of different methods of classification -

***Table 3: Summary of Models and Evaluation Metrics***

| Model | F1 | Precision | Recall | Accuracy |
|---|---|---|---|---|
| Multinomial Naive Bayes | 0.339 | 0.663 | 0.324 | 0.324 |
| Random Forest | 0.494 | 0.624 | 0.45 | 0.45 |
| VADER | 0.466 | 0.639 | 0.42 | 0.420 |
| TextBlob | 0.496 | 0.631 | 0.458 | 0.458 |
| Bi-LSTM+Attention Neural Network | 0.857 | 0.629 | 0.848 | 0.855 |
| Roberta Model & Embeddings | 0.94 | 0.94 | 0.94 | 0.94 |
| **Innovations** | | | | |
| Bi-LSTM+Attention Neural Network (Tweet statistics) | 0.83 | 0.82 | 0.82 | 0.82 |
| Bi-LSTM+Attention Neural Network (Textblob) | 0.84 | 0.84 | 0.84 | 0.84 |
| Bi-LSTM+Attention Neural Network (Emotion Analysis) | 0.83 | 0.83 | 0.83 | 0.83 |

## 5. Conclusion

In conclusion, the team has built an information retrieval system which includes crawling of data, indexing, and developing a user interface for the search engine. Text analysis in the form of sentiment classification was also carried out to do basic analysis of the textual data. Our information retrieval system is capable of retrieving relevant tweets for popular RPGs with their corresponding sentiments.

Future developments to our project may include:

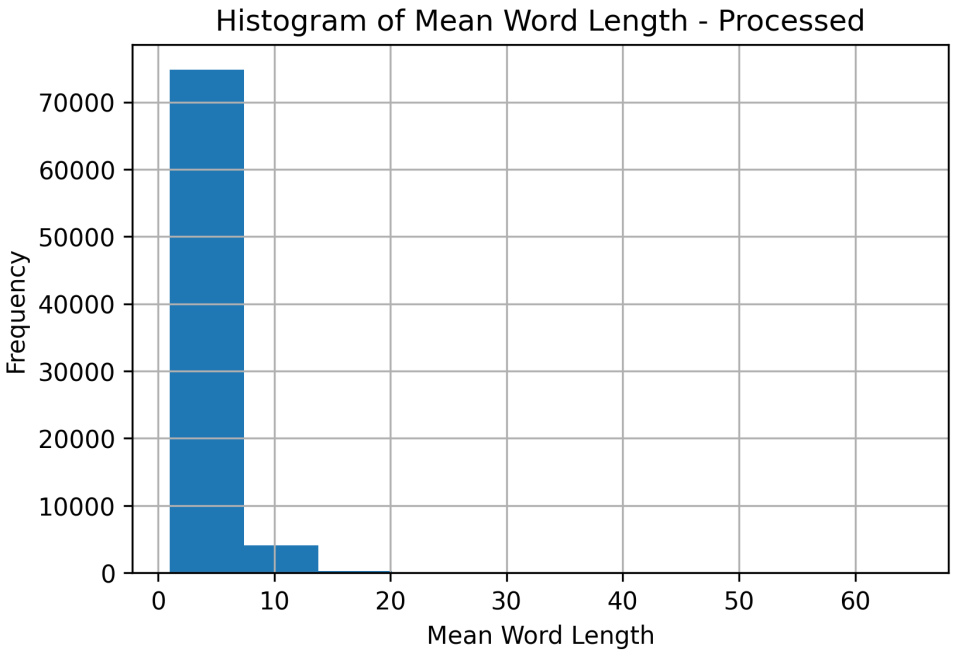
1. One possible future direction for the research is to broaden the reach of the information retrieval system beyond popular/mainstream RPGs. Crawling and indexing data for a greater variety of games could be included, allowing users to search for information on a broader range of games.
2. Another possible enhancement would be to provide users with personalised recommendations based on their search history and interests.
3. Exploring ways to expand the dataset beyond Twitter, such as including data from Reddit or other social media sites, might increase the system's coverage.

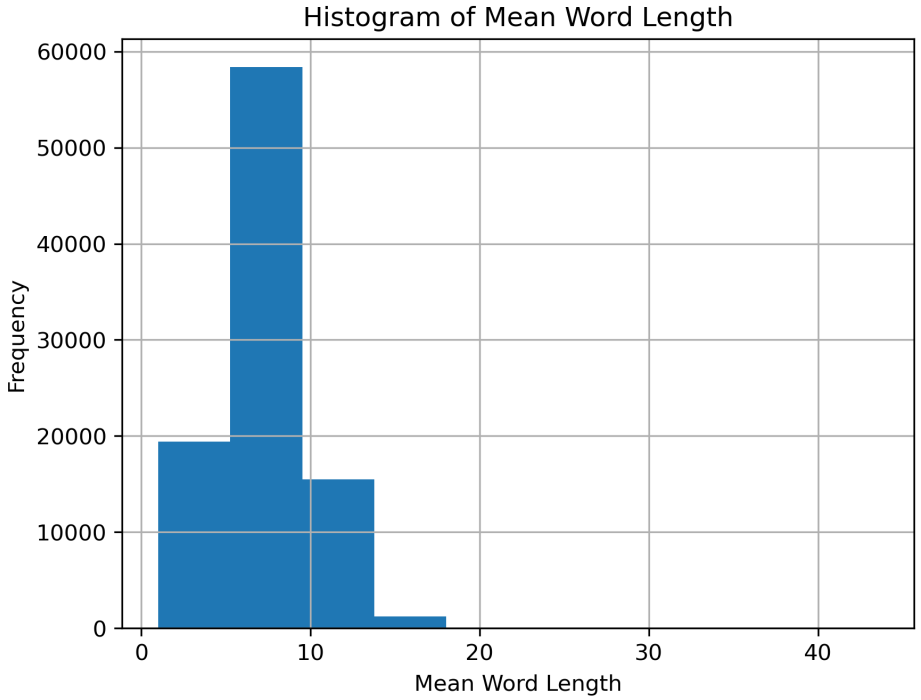
Overall, the project has demonstrated the team's ability to create a functional information retrieval system and provides a strong foundation for further development and expansion.

## 6. Appendix

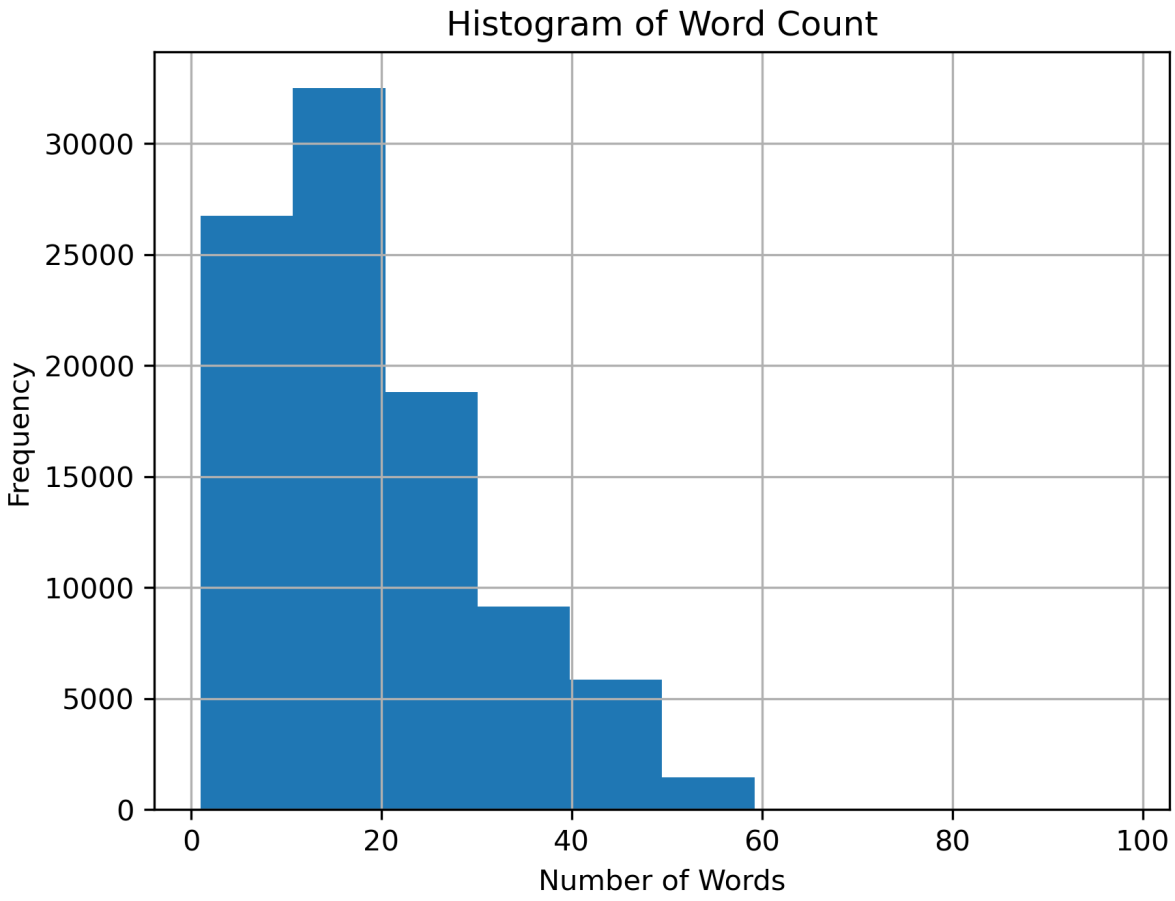
This section presents additional illustrations captured in the code:

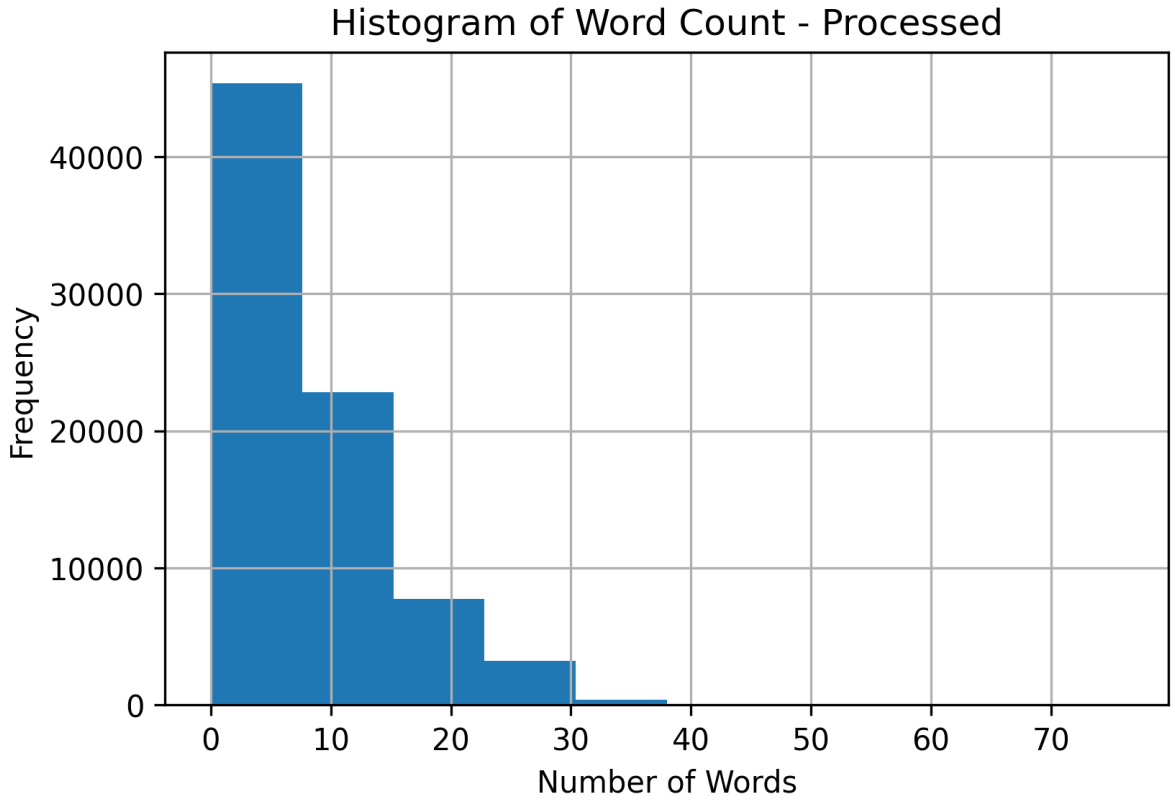
***Appendix 1: Histogram of mean word length before and after preprocessing***



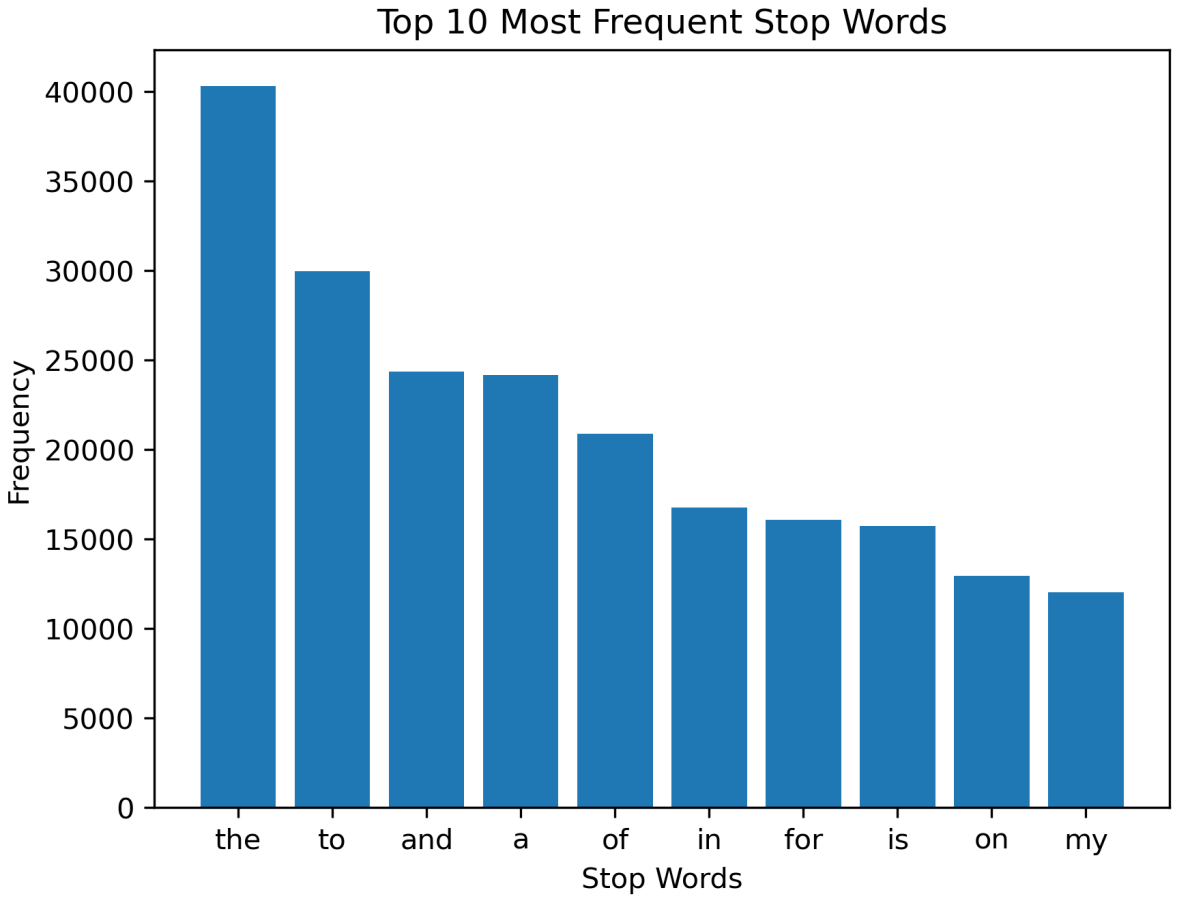



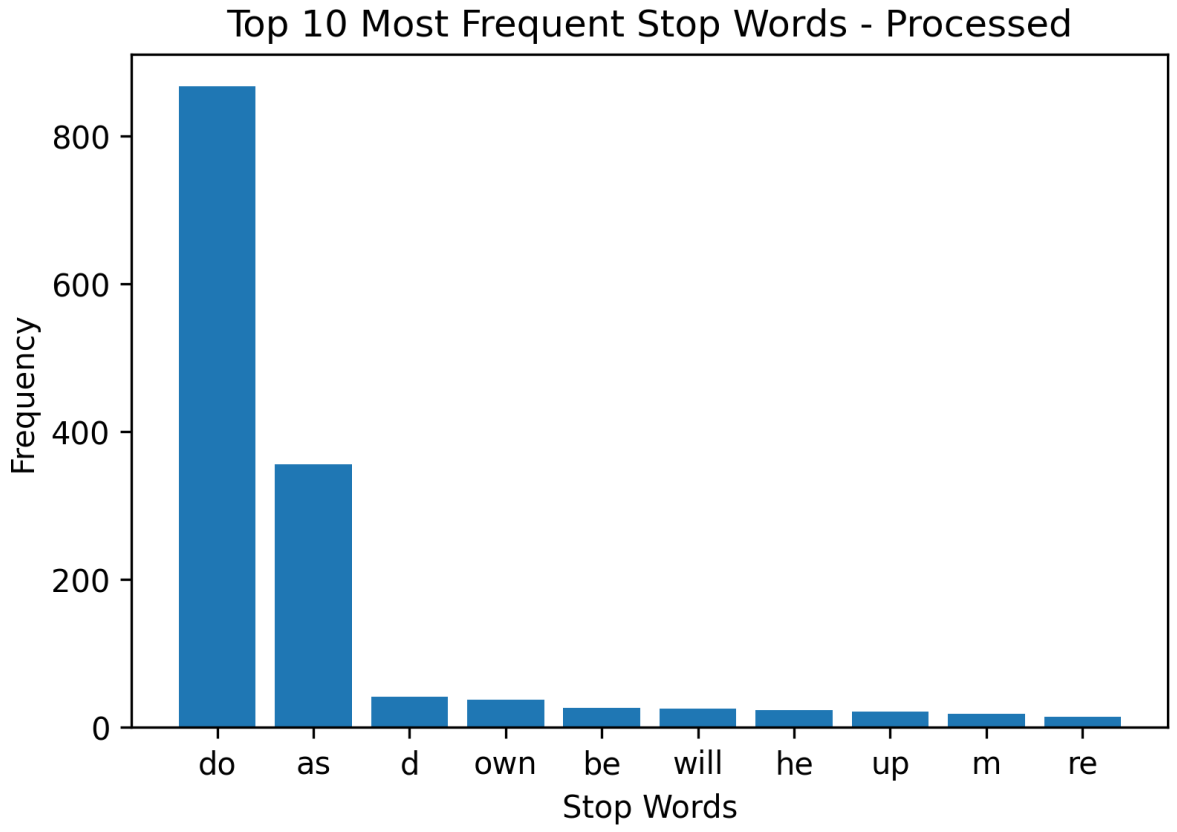
***Appendix 2: Histogram of word count before and after preprocessing***



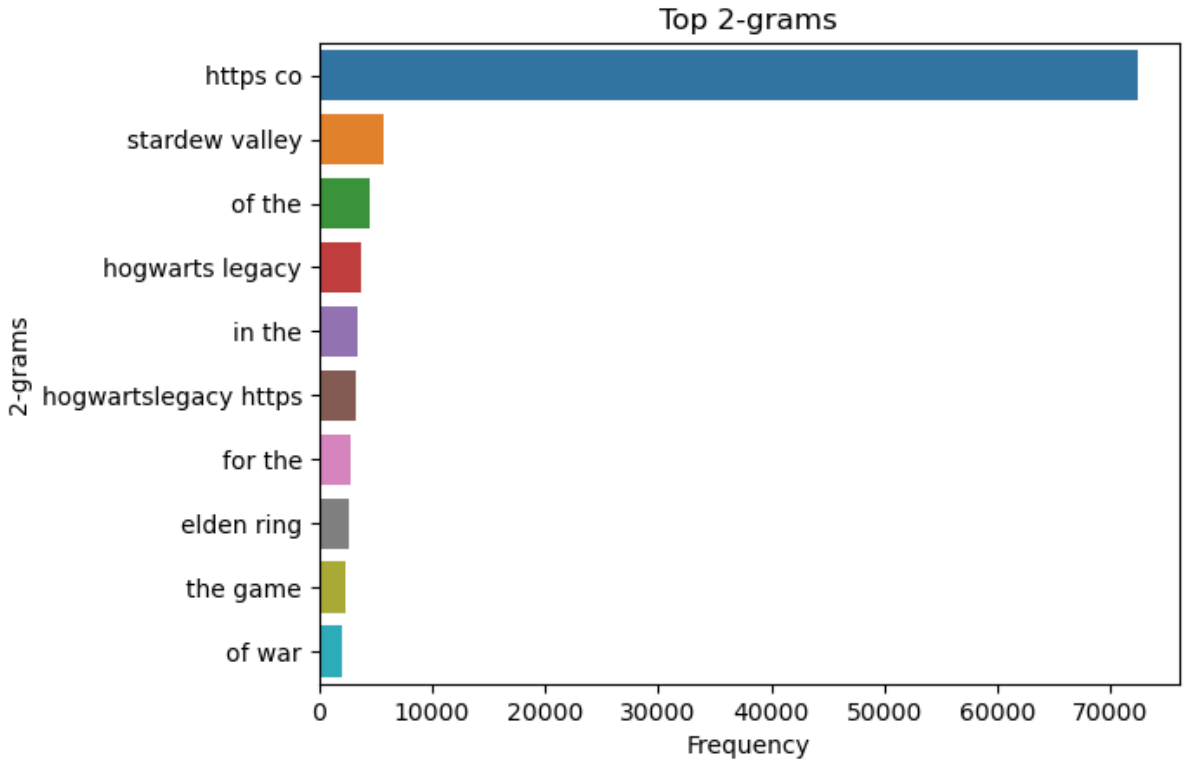



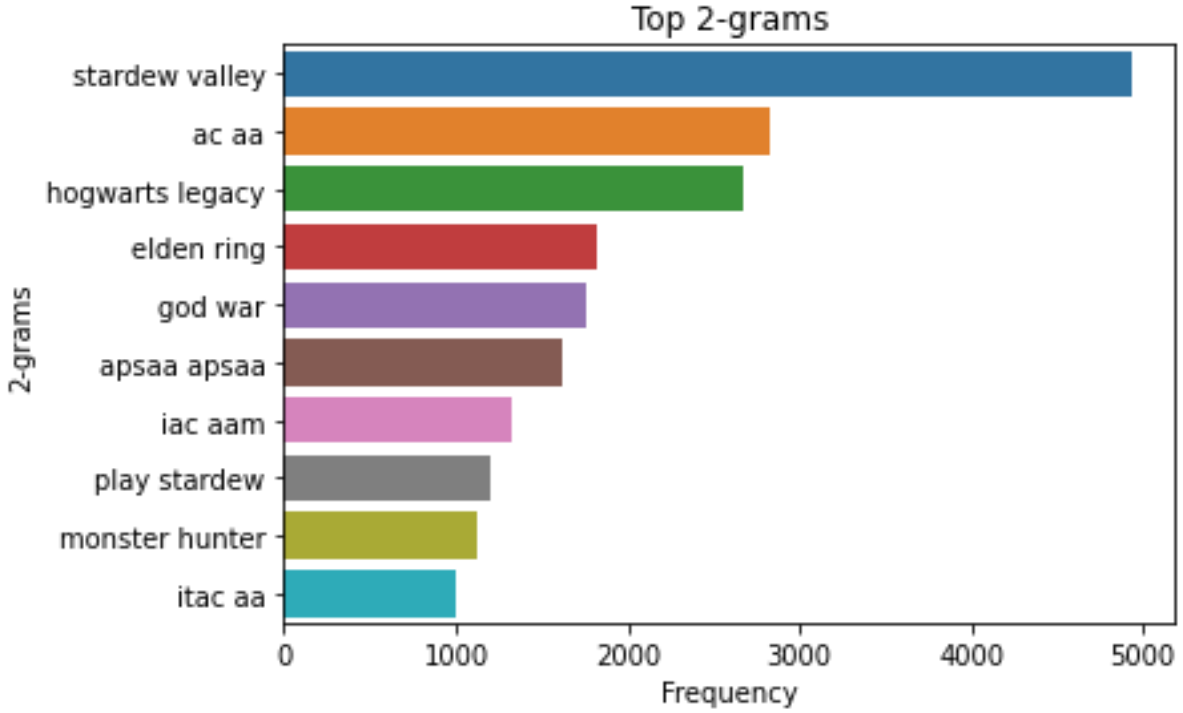
***Appendix 3: Stopwords before and after preprocessing***



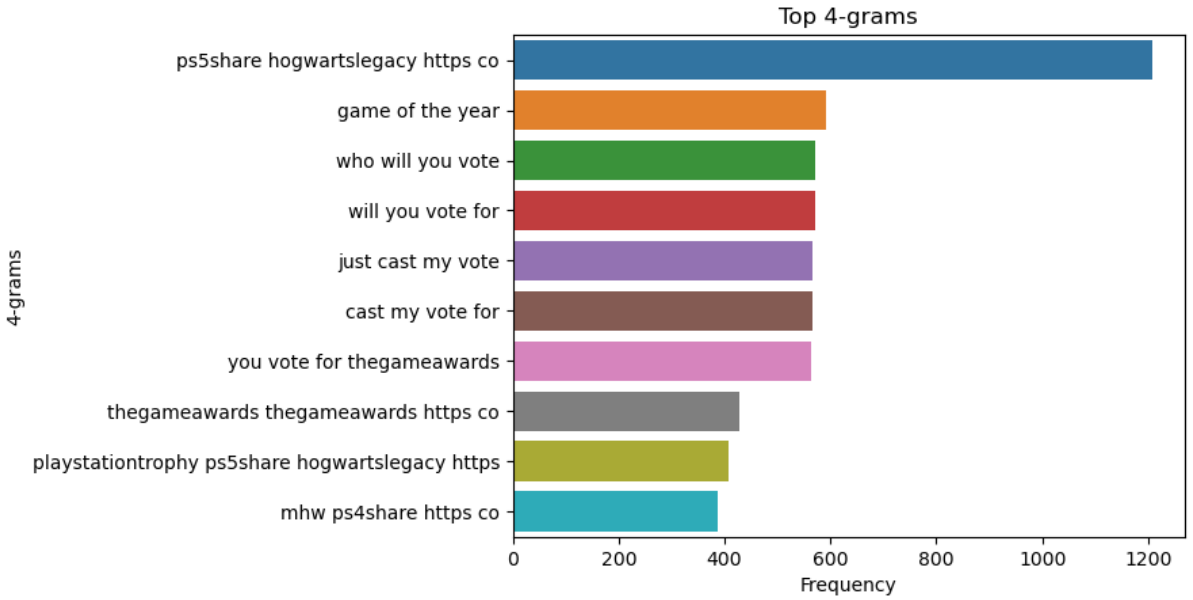



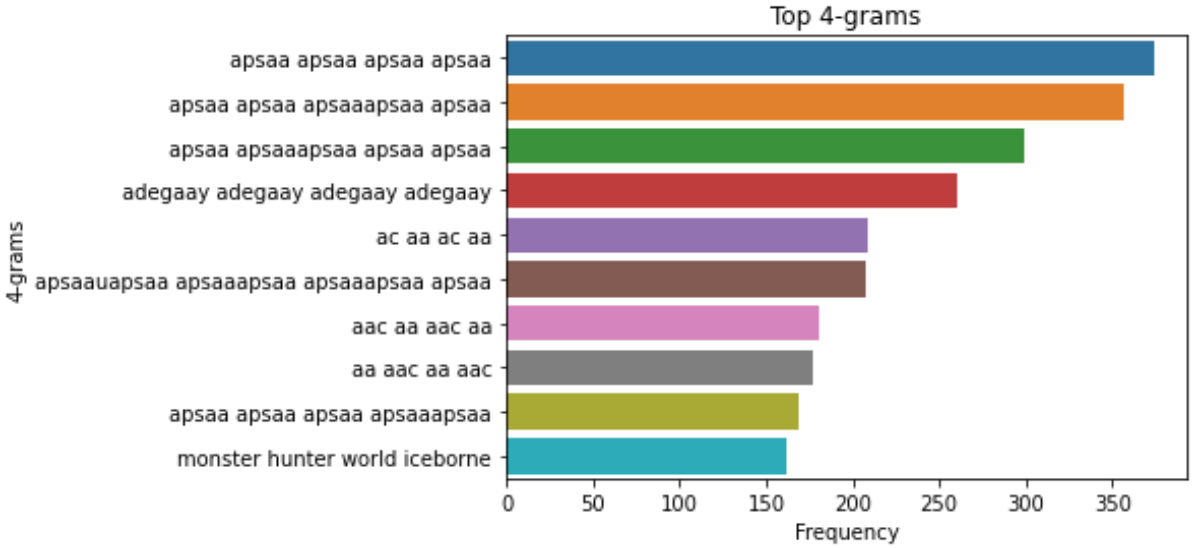
***Appendix 4: Top 2-grams before and after preprocessing***



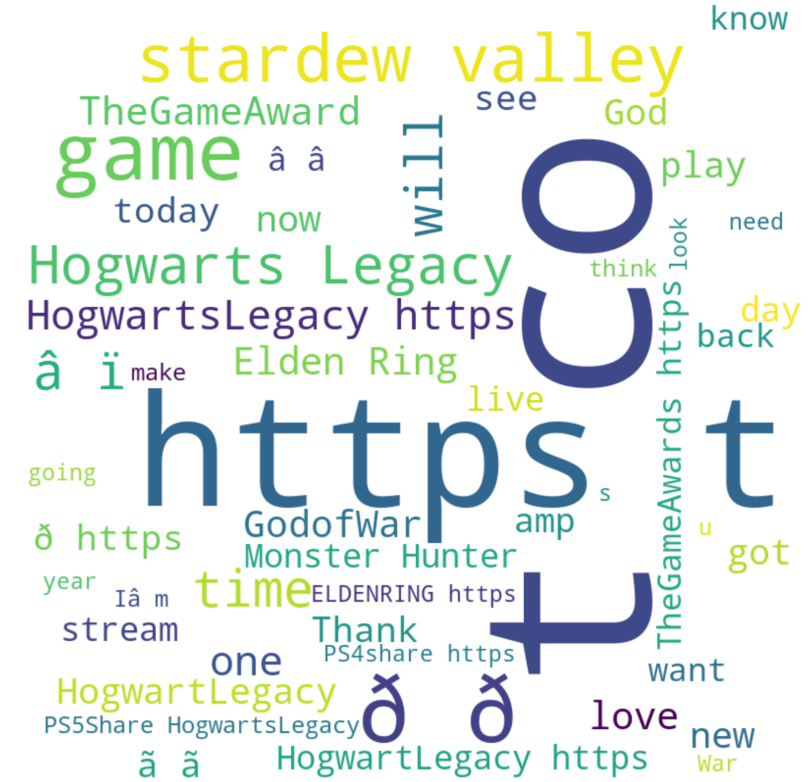



## *Appendix 5: Top 4-grams before and after preprocessing*

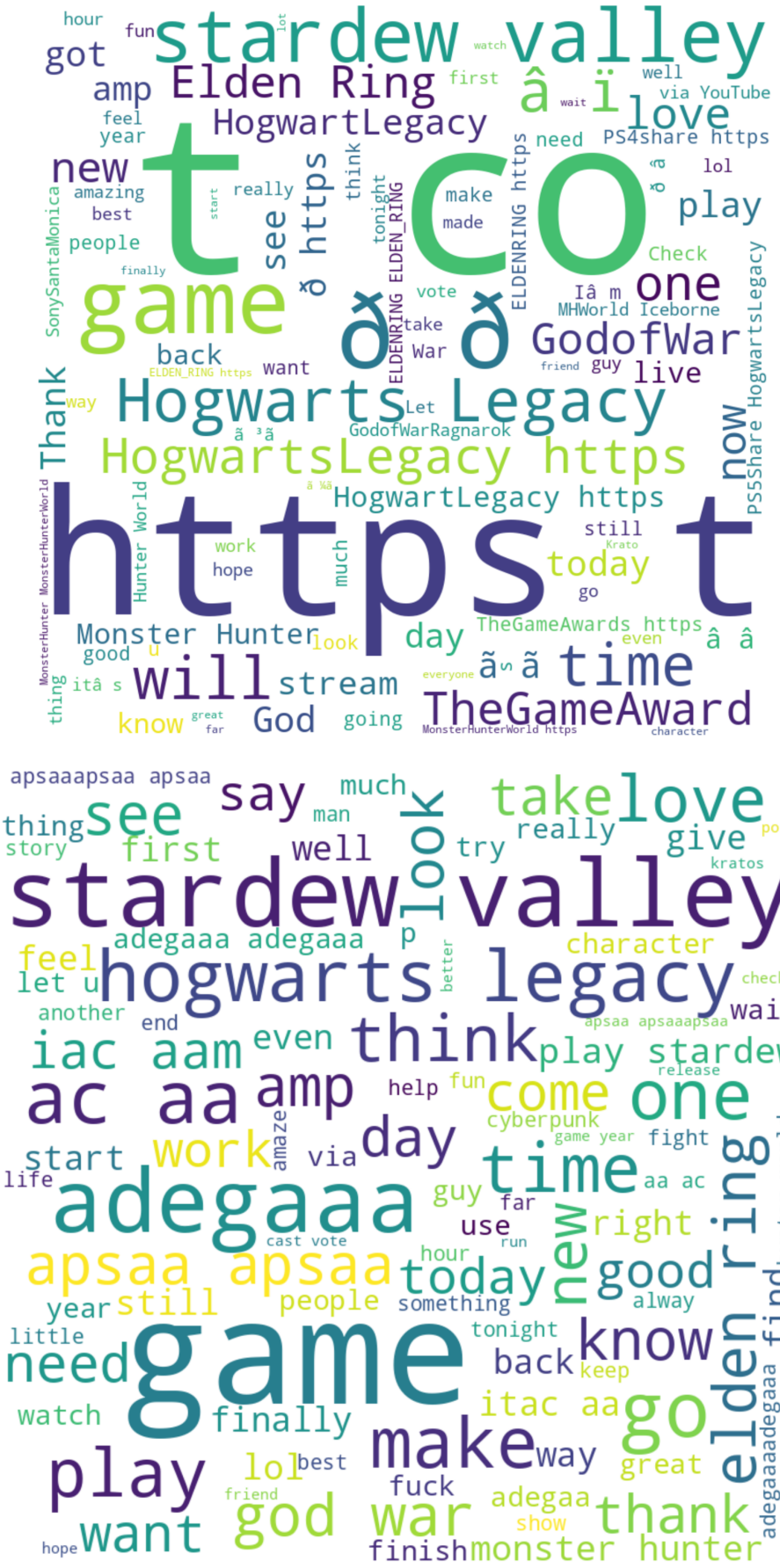


***Appendix 6: Wordcloud - 50 words before and after preprocessing***




***Appendix 7: Wordcloud - 100 words before and after preprocessing***



### *Appendix 8: Specific hashtags used for each RPGs*

**Game 1 - Hogwarts Legacy**

Hashtags:

```
'#HogwartsLegacy','#hogwartslegacy','#HogwartsLegacyGAME',
'#hogwartslegacygame' ,'#HogwartLegacy','#harrypottergame',
'#HogwartsLegacyGame','#hogwartslegacydrops',
'#HOGWARTSLEGACY'
```

User accounts:

```
'HogwartsLegacy', 'Hogwarts_Legacy', 'HogwartsLegacy_',
'HogLegNews'
```

**Game 2 - Stardew Valley**

Hashtags:

```
'#stardewvalley', '#stardew', '#stardewvalleyfarmer',
'#stardewvalleygame'
```

User accounts:

```
'ConcernedApe', 'rStardewValley'
```

**Game 3 - God of War**

Hashtags:

```
'#GodOfWar', '#Kratos', '#Atreus', '#NorseMythology',
'#Odin', '#Ragnarok', '#GameOfTheYear',
'#BeTheGodOfWar', '#SonySantaMonica', '#BladesOfChaos',
'#Fenrir', '#KratosAndAtreus', '#TheWorldSerpent',
‘#GOW’
```

User accounts:

```
'SonySantaMonica'
```

**Game 4 - Cyberpunk2077**

Hashtags:

```
'#Cyberpunk2077','#PhantomLiberty','#CDProjektRed',
'#NightCity','#CyberBug2077','#GlitchCity','#Brokenpunk2077,
'#CDProjektWrecked', '#Refundpunk2077', '#Cyberfail2077',
'#Crash2077','#BuggyNightCity', '#JohnnySilverhand',
'#Braindance', '#Arasaka'
```

User accounts:

```
'CyberpunkGame', 'CP2077Community', 'CDPROJEKTRED',
'CDPRED_Support'
```

**Game 5 - Elden Ring**

Hashtags:

```
'#ELDENRING', '#EldenRing', '#eldenring',
'#Eldenring', '#Elden_Ring', '#EldenRIng',
'#elden_ring', '#eldenRing', '#JourneytoEldenRing',
'#ELdenRing', '#eldenringisbad', '#eldenringps5',
'#EldenRing2022'
```

User accounts:

```
'EldenRingDesc', 'EldenMemet_', 'elden_lean', 'ELDENRING',
'EldenQuotes', 'eldenmfjohn', 'HimboOfFerelden',
'EldenLordCody', 'eldenringjurny', 'PSNEldenRingPVP',
'eldenringcount', 'EldenKing_Wal', 'EldenRingTips'
```

**Game 6 - Monster Hunter World**

Hashtags:

```
'#monsterhunterworld', '#mhworld', '#mhw',
'#monsterhunterworldpc', '#monsterhuntworldxboxone'
```