UNIVERSITY OF BIRMINGHAM

*The University of Birmingham*

*School of Computer Science*

---

# Assignment 1 – News Classifier
# Deadline: 12:00pm, Nov 10, 2023

---


*Author:*
Jizheng Wan

*Reviewers:*
Pieter Joubert
Ahmad Ibrahim
Carl Wilding
Archie Powell


**Version 2.1**

An Assignment submitted for the UoB:

*Object Oriented Programming*

October 27, 2023

# Revision History

| Revision | Date | Author(s) | Description |
|---|---|---|---|
| 1.0 | 24/10/2023 | JW | Draft version for review. |
| 2.0 | 26/10/2023 | JW | (a) Changed a few typos.<br>(b) Modified the description of Task 4.3. |
| 2.1 | 27/10/2023 | JW | (a) Modified the description of Task 1.2 and made it more clear about converting all characters in the content to lowercase.<br>(b) Added a version history table. |

# Contents

# *Rules*

1. For each class refer to its corresponding test to verify field and method naming conventions.
2. Although there are many ways to construct an application, you are required to adhere to the rules as stipulated below (to achieve marks).
3. If variable names are not stipulated, you can use your own names for variables. This shows that you have written the application (we will check for plagiarism).
4. Inclusion of extra imports is strictly prohibited and will lead to a substantial penalty. You are restricted from using external libraries or any libraries that are not pre-included in the provided skeleton code. (So, don't use regular expressions and the Arrays class in this assignment.
5. Do NOT change or modify files included in the "resources" folder.
6. Do NOT modify the skeleton code. However, you are allowed to create your own methods if they are needed.
7. You MUST complete this assignment independently – Do NOT discuss or share your code with others, and Do NOT use ChatGPT! Any cheating behaviour will result in a zero score for this module and will be subject to punishment by the University.
8. It is *STRONGLY ADVISED AGAINST* utilizing any translation software (such as Google Translate) for the translation of this document.
9. The jUnit tests included in the skeleton code are basic and only scratch the surface in evaluating your code. Passing these tests does not guarantee a full mark.
10. Wrong file structure leads to a substantial penalty. Make sure you have followed the Submission Instructions on the Canvas page (the assignment page).
11. Creating your own .zip file without using the export function in IntelliJ may lead to a wrong file structure.

HINT: You can use the TODO window in IntelliJ (View | Tool Windows | TODO) to quickly jump between tasks.

# 1 Introduction

In this assignment, you are tasked with constructing a classifier that can categorise news articles based on their content.

Specifically, you will be working with a dataset consisting of 20 news articles sourced from the Sky News website (included in the "resources" folder). These articles can be broadly classified into two distinct categories, each representing a different topic. For instance, the first category encompasses articles like the one titled "Osiris-Rex's sample from asteroid Bennu will reveal secrets of our solar system". Conversely, the second category includes articles such as the one headlined "Bitcoin slides to five-month low amid wider sell-off".

The main idea here is to assess the semantic closeness of these 20 news articles by using the Term Frequency-Inverse Document Frequency (TF-IDF) embedding.

## 1.1 Term Frequency-Inverse Document Frequency Embedding

TF-IDF, which stands for Term Frequency-Inverse Document Frequency, is a popular numerical statistic that reflects how important a word is to a document in a collection or corpus. It's a widely used technique in information retrieval and text mining to evaluate the relevance of words within documents in a dataset.

TF-IDF Embedding is a technique where text documents are converted into vector representations such that each document is represented as a vector in a multidimensional space. Each dimension in this space corresponds to a unique word in the corpus vocabulary, and the value in each dimension is the TF-IDF weight of that word in the respective document.

A major advantage of using high-dimensional vectors for document representation is their compatibility with further numerical processing tasks, such as input for neural networks (NN). In essence, TF-IDF Embedding acts as a vectorisation procedure. Unlike one-hot encoding—which uniquely numerically identifies each vocabulary word, equating the maximum number to the vocabulary's size—TF-IDF Embedding maintains words' intrinsic relevance (or weight) throughout the transformation phase.

## 1.2 A Step-by-Step Guidance of TF-IDF Embedding

As suggested by its name, TF-IDF assigns a score or vectorises a word by calculating the product of the word's Term Frequency (TF) and the Inverse Document Frequency (IDF).

**Term Frequency:** The TF represents the occurrence of a term or word in relation to the document's total word count, expressing how frequently a specific term appears within it. TF is calculated by:

$$TF(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \tag{1}$$

where $f_{t,d}$ is the number of times a word ($t$) appears in a document $d$, and $\sum_{t' \in d} f_{t',d}$ is the total number of words in that document.

**Inverse Document Frequency:** The IDF signifies the representation of a term based on its occurrence across various documents in a corpus. It quantifies the rarity of a term by determining how frequently it appears, offering insight into the term's uniqueness or commonality within the corpus. It is calculated by:

$$IDF(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|} + 1 \tag{2}$$

where $N$ is the total number of documents in the corpus, $|\{d \in D : t \in d\}|$ is the number of documents where the word $t$ appears.

Then the final TF-IDF is calculated as:

$$TFIDF(t,d) = TF(t,d) * IDF(t,D) \tag{3}$$

To illustrate how TF-IDF Embedding works, consider we have three documents in a corpus ($N = 3$).

| Document | Contents |
|---|---|
| D1 | harry_potter is a student at hogwarts |
| D2 | voldemort used to be a student at hogwarts but graduated already |
| D3 | the parents of harry_potter studied at hogwarts as well |

Table 2: Documents in the corpus

Let $W$ be the vocabulary list which contains all the unique words in the corpus ($D1 - D3$). So, *W={harry_potter, is, a, student, at, hogwarts, voldemort, used, to, be, but, graduated, already, the, parents, of, studied, as, well}*. Then calculate the associated TF-IDF values in each document (denoted as $V_{document}$). For example, the TF-IDF values of $W$ in $D1$ can be calculated as:

- $TFIDF(harry\_potter, D1) = \frac{1}{6} \times (\log \frac{3}{2} + 1) = 0.23424418468469405$
- $TFIDF(is, D1) = \frac{1}{6} \times (\log \frac{3}{1} + 1) = 0.3497687147780183$
- $TFIDF(a, D1) = \frac{1}{6} \times (\log \frac{3}{2} + 1) = 0.23424418468469405$
- $TFIDF(student, D1) = \frac{1}{6} \times (\log \frac{3}{2} + 1) = 0.23424418468469405$
- $TFIDF(at, D1) = \frac{1}{6} \times (\log \frac{3}{3} + 1) = 0.16666666666666666$
- $TFIDF(hogwarts, D1) = \frac{1}{6} \times (\log \frac{3}{3} + 1) = 0.16666666666666666$
- $TFIDF(voldemort, D1) = \frac{0}{6} \times (\log \frac{3}{1} + 1) = 0$
- $TFIDF(used, D1) = \frac{0}{6} \times (\log \frac{3}{1} + 1) = 0$
- $TFIDF(to, D1) = \frac{0}{6} \times (\log \frac{3}{1} + 1) = 0$

- $TFIDF(be, D1) = \frac{0}{6} \times (\log \frac{3}{1} + 1) = 0$
- $TFIDF(but, D1) = \frac{0}{6} \times (\log \frac{3}{1} + 1) = 0$
- $TFIDF(graduated, D1) = \frac{0}{6} \times (\log \frac{3}{1} + 1) = 0$
- $TFIDF(already, D1) = \frac{0}{6} \times (\log \frac{3}{1} + 1) = 0$
- $TFIDF(the, D1) = \frac{0}{6} \times (\log \frac{3}{1} + 1) = 0$
- $TFIDF(parents, D1) = \frac{0}{6} \times (\log \frac{3}{1} + 1) = 0$
- $TFIDF(of, D1) = \frac{0}{6} \times (\log \frac{3}{1} + 1) = 0$
- $TFIDF(studied, D1) = \frac{0}{6} \times (\log \frac{3}{1} + 1) = 0$
- $TFIDF(as, D1) = \frac{0}{6} \times (\log \frac{3}{1} + 1) = 0$
- $TFIDF(well, D1) = \frac{0}{6} \times (\log \frac{3}{1} + 1) = 0$

Hence, the vector representation of the first document $V_{D1}$ ={*0.23424418468469405, 0.3497687147780183, 0.23424418468469405, 0.23424418468469405, 0.16666666666666666, 0.16666666666666666, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0*}

Following the same process, $V_{D2}$ can be calculated as:

- $TFIDF(harry\_potter, D2) = \frac{0}{11} \times (\log \frac{3}{2} + 1)$
- $TFIDF(is, D2) = \frac{0}{11} \times (\log \frac{3}{1} + 1)$
- $TFIDF(a, D2) = \frac{1}{11} \times (\log \frac{3}{2} + 1)$
- $TFIDF(student, D2) = \frac{1}{11} \times (\log \frac{3}{2} + 1)$
- $TFIDF(at, D2) = \frac{1}{11} \times (\log \frac{3}{3} + 1)$
- $TFIDF(hogwarts, D2) = \frac{1}{11} \times (\log \frac{3}{3} + 1)$
- $TFIDF(voldemort, D2) = \frac{1}{11} \times (\log \frac{3}{1} + 1)$
- $TFIDF(used, D2) = \frac{1}{11} \times (\log \frac{3}{1} + 1)$
- $TFIDF(to, D2) = \frac{1}{11} \times (\log \frac{3}{1} + 1)$
- $TFIDF(be, D2) = \frac{1}{11} \times (\log \frac{3}{1} + 1)$
- $TFIDF(but, D2) = \frac{1}{11} \times (\log \frac{3}{1} + 1)$
- $TFIDF(graduated, D2) = \frac{1}{11} \times (\log \frac{3}{1} + 1)$
- $TFIDF(already, D2) = \frac{1}{11} \times (\log \frac{3}{1} + 1)$
- $TFIDF(the, D2) = \frac{0}{11} \times (\log \frac{3}{1} + 1)$

- $TFIDF(parents, D2) = \frac{0}{11} \times (\log \frac{3}{1} + 1)$
- $TFIDF(of, D2) = \frac{0}{11} \times (\log \frac{3}{1} + 1)$
- $TFIDF(studied, D2) = \frac{0}{11} \times (\log \frac{3}{1} + 1)$
- $TFIDF(as, D2) = \frac{0}{11} \times (\log \frac{3}{1} + 1)$
- $TFIDF(well, D2) = \frac{0}{11} \times (\log \frac{3}{1} + 1)$

And $V_{D3}$:

- $TFIDF(harry\_potter, D3) = \frac{1}{9} \times (\log \frac{3}{2} + 1)$
- $TFIDF(is, D3) = \frac{0}{9} \times (\log \frac{3}{1} + 1)$
- $TFIDF(a, D3) = \frac{0}{9} \times (\log \frac{3}{2} + 1)$
- $TFIDF(student, D3) = \frac{0}{9} \times (\log \frac{3}{2} + 1)$
- $TFIDF(at, D3) = \frac{1}{9} \times (\log \frac{3}{3} + 1)$
- $TFIDF(hogwarts, D3) = \frac{1}{9} \times (\log \frac{3}{3} + 1)$
- $TFIDF(voldemort, D3) = \frac{0}{9} \times (\log \frac{3}{1} + 1)$
- $TFIDF(used, D3) = \frac{0}{9} \times (\log \frac{3}{1} + 1)$
- $TFIDF(to, D3) = \frac{0}{9} \times (\log \frac{3}{1} + 1)$
- $TFIDF(be, D3) = \frac{0}{9} \times (\log \frac{3}{1} + 1)$
- $TFIDF(but, D3) = \frac{0}{9} \times (\log \frac{3}{1} + 1)$
- $TFIDF(graduated, D3) = \frac{0}{9} \times (\log \frac{3}{1} + 1)$
- $TFIDF(already, D3) = \frac{0}{9} \times (\log \frac{3}{1} + 1)$
- $TFIDF(the, D3) = \frac{1}{9} \times (\log \frac{3}{1} + 1)$
- $TFIDF(parents, D3) = \frac{1}{9} \times (\log \frac{3}{1} + 1)$
- $TFIDF(of, D3) = \frac{1}{9} \times (\log \frac{3}{1} + 1)$
- $TFIDF(studied, D3) = \frac{1}{9} \times (\log \frac{3}{1} + 1)$
- $TFIDF(as, D3) = \frac{1}{9} \times (\log \frac{3}{1} + 1)$
- $TFIDF(well, D3) = \frac{1}{9} \times (\log \frac{3}{1} + 1)$

$V_{D1}$, $V_{D2}$ and $V_{D3}$ are the TF-IDF Embeddings of D1, D2 and D3.

## 1.3 Measuring the semantic closeness

Computational linguistics research holds that word (or document) meaning can be represented by its contextual information because similar contextual distributions tend to share between semantically similar words [2].

Moreover, there is a clustering/grouping trend for words/documents with similar meanings. Using the below figure (Fig. 1) [1] as an example, originally, it has been used to explain how Word2Vec (which is another popular embedding method) understands semantic relationships, like Paris and France are related the same way Beijing and China are (capital and country), and not in the same way Lisbon and Japan are. Consequently, those words that represent the concept of "Country" have been grouped together and separated from those that represent the "Capital" concept.

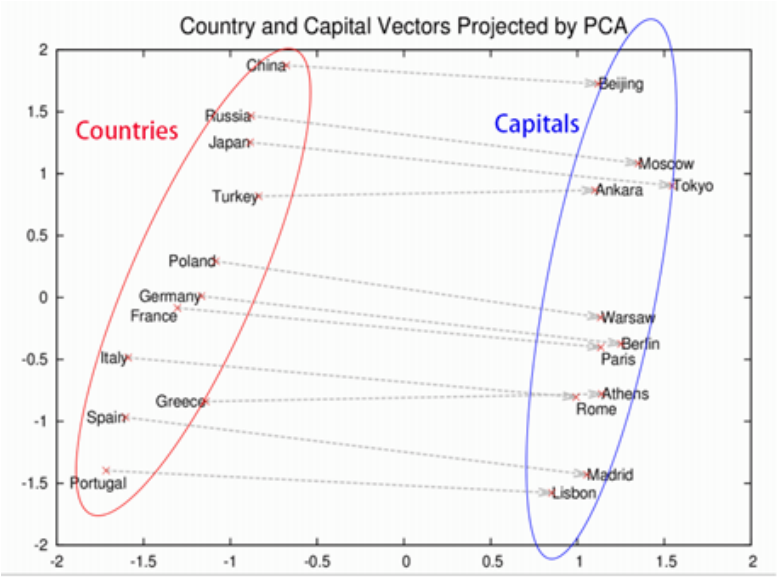


Figure 1: Clustering trend of vector representation

Moreover, the "closeness" of two words/documents can be measured by *CosineSimilarity* (*CS*) (also called cosine distance). *CS* is a measure of the cosine of the angle between two non-zero vectors ($\theta$ in Fig. 2). It is calculated by using Equation 4 listed below:

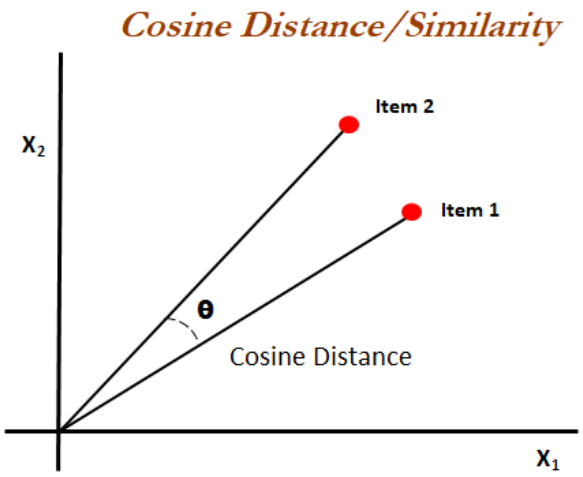


Figure 2: Cosine Similarity/Distance

$$CS(\overrightarrow{U},\overrightarrow{V}) = \frac{\overrightarrow{U}\cdot\overrightarrow{V}}{||\overrightarrow{U}||\,||\overrightarrow{V}||} = \frac{\sum_{i=1}^{n} u_i \times v_i}{\sqrt{\sum_{i=1}^{n} u_i^2} \times \sqrt{\sum_{i=1}^{n} v_i^2}} \tag{4}$$

where $\overrightarrow{U}$ is the vector representation of item 1 (in our case, $V_{D1}$) and $\overrightarrow{V}$ is the vector representation of the D2 ($V_{D2}$). $u_i$ is the i-th element in $\overrightarrow{U}$, and $v_i$ is the i-th element in $\overrightarrow{V}$.

To identify the most similar document to the first document ($D1$), we can calculate its $CS$ value with all the other documents, in this case $CS(V_{D1},V_{D2})$ and $CS(V_{D1},V_{D3})$. Then the higher the $CS$ value is, the closer a document is to D1.

Assuming that D1 and D2 belong to two different groups and the goal is to determine which group D3 should belong. Then we can calculate $CS(V_{D1},V_{D3})$ and $CS(V_{D2},V_{D3})$. If $CS(V_{D1},V_{D3}) > CS(V_{D2},V_{D3})$ then D3 is more semantically close to D1, and therefore should belong to the first group.

# 2 Task 1 - HtmlParser.java [10 Marks]

There is a string array type (String[]) variable called myHTMLS defined in the NewsClassifier class (NewsClassifier.java). This variable is allocated to hold the HTML codes/strings of relevant news articles[1]. For instance, the HTML string from the file

[1]The actual HTML string is quite complex, but most parts of the string can be ignored. Using 01.htm as an example, you only need to process lines 28 and 82 to get the title and content.

01.htm is stored in myHTMLS[0], and the string from the file 02.htm is stored in myHTMLS[1]. The actual file reading work is done by a provided method called loadHTML() in the Toolkit class (Toolkit.java). Test this class with the *Tester_HtmlParser.java* file.

The first task in this assignment is to build an HTML parser that can extract the title and the content of the news articles from the HTML code stored in the myHTMLS variable mentioned above.

## 2.1 Task 1.1 - *getNewsTitle*(*String*[] *_htmlCode*) [5 Marks]

The *_htmlCode* parameter contains the full HTML string of a specific news. E.g. 01.htm. Where the text that contains the title of the news is located between the *<title></title>* tag[2]. In the below example, the title should be "NASA launches Psyche mission to rare metal asteroid", without including " | Science & Tech News | Sky News".

```
<title>NASA launches Psyche mission to rare metal asteroid | Science &
    Tech News | Sky News</title>
```

- Use the *String.substring*() method to extract the title from the *_htmlCode* parameter.
- If the news article doesn't have a title, return "Title not found!". Otherwise, return the extracted title.

Hint: you may need to call the *substring*() method several times. The HTML string will be passed into this method via the *_htmlCode* parameter, so you don't need to call the Toolkit.loadHTML() method again.

## 2.2 Task 1.2 - *getNewsContent*(*String*[] *_htmlCode*) [5 Marks]

Use the 01.htm file as an example. Analyse the string on line 82 and figure out a way to extract the content of the "articleBody". In the below example, the content should be "NASA has launched a mission to a rare asteroid covered in metal that is two billion ..... it at a distance of between 47 and 440 miles (75 and 700 km) until at least 2031."

```
{"@context": "http://schema.org","@type":
    "NewsArticle","alternativeHeadline": "NASA launches Psyche mission
    to rare metal asteroid","articleBody": "NASA has launched a mission
    to a rare asteroid covered in metal that is two billion ..... it at
    a distance of between 47 and 440 miles (75 and 700 km) until at
    least 2031. ","mainEntityOfPage": {"@type": "WebPage","url":
    "/story/nasa-launches-mission-to....12983967"},"wordCount":
    "489","inLanguage": "en-GB","genre": "technology","publisher":
    {"@type": "Organization","@id": "#Publisher","name": "Sky","logo":
```

[2]There might be more than one *<title></title>* tags in the HTML string. It is the first appeared tag that you need to use.

```
{"@type": "ImageObject","@id": "#Logo","url":
"https://news.sky.com/assets/sky-news-logo.svg?v&#x3D;1","width":
"255","height": "60"}},"headline": "NASA launches Psyche mission to
rare metal asteroid","description": "","dateline": "London,
UK","copyrightHolder": {"@id":"#Publisher"},"author":
{"@id":"#Publisher"},"datePublished":
"2023-10-13T16:16:00Z","dateModified":
"2023-10-13T19:33:00Z","dateCreated":
"2023-10-13T16:18:00Z","image": {"@type": "ImageObject","url":
"https://e3.365dm.com/...._6320615.jpg","width": 2048,"height": 1152
} ,"url":"/story/nasa-launches-mission....12983967"}
```


- Use the *String.substring*() method to extract the content (articleBody) of the news from the *_htmlCode* parameter.
- Converts all of the characters in the content to lower case.
- If the news article doesn't have content, return "Content not found!". Otherwise, return the extracted content.

Hint: technically, you only need to call the *substring*() method once. The HTML string will be passed into this method via the *_htmlCode* parameter, so you don't need to call the Toolkit.loadHTML() method again.

# 3 Task 2 - NLP.java [9 Marks]

In the previous task, we extracted the title and content from the HTML string. Before further processing the content, a few pre-processing tasks (included in the NLP class) must be done. Testing this class with the *Tester_NLP.java* file.

## 3.1 Task 2.1 - *textCleaning*(*String _content*) [3 Marks]

The first pre-processing task is text cleaning. We need to **a)** convert the content to lowercase, and **b)** remove all the special characters from the extracted content. Essentially, only 'a'-'z', '0'-'9' and white space (' ') are allowed to exist in the content.

Complete this method to:

- Firstly, convert the content to lowercase.
- Secondly, go through the content character by character and remove/delete all the special characters from it.
- Once it's been done, return the cleaned content.

Hint: use the *.toCharArray()* method to convert the *_content* to a new character array, then use a loop to go through this array character by character.

### 3.2 Task 2.2 - $textLemmatization(String\ \_content)$ [3 Marks]

Text lemmatization is a vital preprocessing step in Natural Language Processing (NLP) that aims to reduce words to their base or root form. Essentially, it involves the process of transforming a word into its simplest form, which allows various inflexions and derivations of a word to be analyzed as a single item.

Lemmatization is a very complex task as it considers the context and grammatical attributes of words, such as part-of-speech, to determine the base form. In this assignment, we will simplify this process and only check if a word ends with "ing", "ed", "es" and "s" (please follow this order). If so, delete these character(s) from the end of the word.

For example, "apples" should be "appl" (check "es" before "s"), "playing" should be "play", "helped" should be "help" and "bananas" should be "banana".

Please complete this method accordingly.

Hint: the text lemmatization task is after the text cleaning task, so tokenise the content using the *.split()* method.

### 3.3 Task 2.3 - $removeStopWords(String\ \_content, String[]\ \_stopWords)$ [3 Marks]

The stop-word removal is a popular step in the preprocessing phase of NLP. Stop words are common words such as 'and', 'the', 'is', and 'in', which are considered to be of little value in text analysis because they occur frequently across various documents and are generally not essential for understanding the text's context or semantics.

Eliminating stop words from text data reduces the data's dimensionality and allows algorithms to focus on the more informative words and phrases. This results in a more efficient analysis, as the algorithms are not bogged down by the high frequency of common words, allowing them better to discern the meaningful patterns and relationships in the text.

There are two parameters in this method, *_content* is the original text and *_stopWords* is an array which contains all the stop-words.

Complete this method and remove all the stop-words from the original text. Once it's been done, return the modified text.

Hint: this is after the text cleaning task, so tokenise the content using the *.split()* method.

## 4 Task 3 - Vector.java [24 Marks]

The Vector class defines vector objects, and you need to complete the following methods to finish this class. Testing this class with the $Tester_Vector.java$ file.

## 4.1 Task 3.1 - $Vector(double[]\ \_elements)$ [0.5 Marks]

This is the constructor of the Vector class.

- Complete this constructor by assigning the *_elements* to the *doubElements* attribute.

## 4.2 Task 3.2 - $getElementatIndex(int\ \_index)$ [2 Marks]

This is the method to get an element at a specific vector index.

- Complete this method to return an element in this vector based on the *_index*.
- If the index is out of bounds, return -1 instead.

## 4.3 Task 3.3 - $setElementatIndex(double\ \_value,\ int\ \_index)$ [2 Marks]

This is the method to set an element at a specific vector index.

- Set the value of an element at *_index* to *_value*.
- If the index is out of bounds, modify the value of the last element instead.

## 4.4 Task 3.4 - $getAllElements()$ [0.5 Marks]

- Return all elements in this vector.

## 4.5 Task 3.5 - $getVectorSize()$ [0.5 Marks]

- Return the size/length of this vector.

## 4.6 Task 3.6 - $reSize(int\ \_size)$ [6 Marks]

- If *_size* equals to the current length of the vector, or *_size* $<= 0$ return the existing vector without any modifications.
- Otherwise, returns a new vector with the specified size/length:
  - If *_size* is smaller than the current length, only keep the first X elements (X=*_size*). For example, resizing [1.0,2.0,3.0,4.0,5.0] to 3 should give you [1.0,2.0,3.0].
  - Otherwise, assign -1.0 to the new elements added to the vector but existing elements remain unchanged. For example, resizing [1.0,2.0,3.0] to 5 should give you [1.0,2.0,3.0,-1.0,-1.0];

## 4.7 Task 3.7 - $add(Vector\ \_v)$ [2 Marks]

- Adding another vector $\_v$ to the current vector.
- If the length of $\_v$ is bigger than the length of the current vector, call the $reSize(int\ \_size)$ method to increase the length of the current vector (both vectors should have the same length);
- Otherwise, increase the length of $\_v$ and make it equal to the length of the current vector.

The formula of vector addition:

$$\overrightarrow{U} + \overrightarrow{V} = [u_1 + v_1,\ u_2 + v_2,\ ...,\ u_i + v_i] \tag{5}$$

where $u_i$ is the i-th element in the vector $\overrightarrow{U}$, $v_i$ is the i-th element in the vector $\overrightarrow{V}$.

## 4.8 Task 3.8 - $subtraction(Vector\ \_v)$ [2 Marks]

- Subtracting another vector $\_v$ to the current vector.
- If the length of $\_v$ is bigger than the length of the current vector, call the $reSize(int\ \_size)$ method to increase the length of the current vector (both vectors should have the same length);
- Otherwise, increase the length of $\_v$ and make it equal to the length of the current vector.

The formula of vector subtraction:

$$\overrightarrow{U} - \overrightarrow{V} = [u_1 - v_1,\ u_2 - v_2,\ ...,\ u_i - v_i] \tag{6}$$

where $u_i$ is the i-th element in the vector $\overrightarrow{U}$, $v_i$ is the i-th element in the vector $\overrightarrow{V}$.

## 4.9 Task 3.9 - $dotProduct(Vector\ \_v)$ [2 Marks]

- Dot product the current vector with vector $\_v$.
- As with the previous method, call the $reSize(int\ \_size)$ method to make them having the same length.

The formula of dot product:

$$\overrightarrow{U} \cdot \overrightarrow{V} = \sum_{i=1}^{n} u_i \times v_i \tag{7}$$

where $u_i$ is the i-th element in the vector $\overrightarrow{U}$, $v_i$ is the i-th element in the vector $\overrightarrow{V}$.

### 4.10 Task 3.10 - *cosineSimilarity*(*Vector _v*) [6.5 Marks]

- Calculate the *CS* value between the current vector and vector *_v*.
- As with the previous method, call the *reSize*(*int _size*) method to make them having the same length.

The formula of Cosine Similarity was introduced already - Equation 4, page 10.

## 5 Task 4 - NewsClassifier.java [57 Marks]

The constructor of this class will make three calls:

**a)** it will call the *Toolkit.loadHTML*() method and use the returned value to populate the *myHTMLs* variable, which contains all the HTML strings from the news articles located in the *resources* folder.

**b)** it will call the *Toolkit.loadStopWords*() method and use the returned value to populate the *myStopWords* variable.

**c)** it will call the *loadData*() method, which is one of the tasks you need to complete in this assignment.

The *Toolkit.loadHTML*() and *Toolkit.loadStopWords*() are provided already, and please do ***not*** modify these methods.

Please use the provided *main*() method to figure out how we would like this application to work.

### 5.1 Task 4.1 - *loadData*() [2 Marks]

The purpose of this method is to populate the *newsTitles* and *newsContents* variables by the return values from the *HtmlParser.getNewsTitle*() and *HtmlParser.getNewsContent*() methods completed in Task 1.1 (section 2.1) and Task 1.2 (section 2.2).

Complete this method accordingly with the following requirements:

- The size of the *newsTitles* and *newsContents* array should be equal to the number of the HTML files in the resources folder. 20 files are provided in this assignment, but we will use a different dataset in the auto-marking system. So make sure your code can handle this properly.
- For both *newsTitles* and *newsContents* arrays, the order of the elements must be consistent with the *myHTMLs* array. In other words, they are parallel arrays, *myHTMLs*[0] is the HTML string from the 01.htm file, and the title and the content of this article are stored in *newsTitles*[0] and *newsContents*[0] respectively.

## 5.2 Task 4.2 - *prePocessing*() [5 Marks]

This method will return the pre-processed news contents extracted from the last task (*newsContents*). There are three steps in the pre-processing task: 1) text cleaning, 2) text lemmatization and 3) removing stop-words (please do not change this order ).

Complete this method accordingly.

Hint: use the *NLP.textCleaning()*, *NLP.textLemmatization()* and *NLP.removeStopWords()* methods.

## 5.3 Task 4.3 - *calculateTFIDF*(*String*[] *_cleanedContents*) [10 Marks]

Calculate the TF-IDF Embedding by using the cleaned contents from the last task. Please refer to Section 1.2 to better understand the TF-IDF Embedding process.

The *vocabularyList* variable is used to store all the unique words in the ***cleaned contents***. It is the equivalent of $W$ in the harry_potter example discussed above. You are required to write a separate method in the next task to build this list.

The embedding results should be stored in a two-dimensional array – $myTFIDF$. Each row (the first dimension) represents a news article, and the second dimension is used to store the actual embedding of that article.

Complete this method accordingly with the following requirements:

- The size of the $myTFIDF$ must match the length of the cleaned contents and the length of the vocabulary list.
- The auto-marking system will use a different dataset. Hence, ensure that your code is adaptable to such modification. Specifically, ***avoid hardcoding*** the size of the $myTFIDF$, ensuring flexibility in handling different dataset sizes.

Hint: consult the *loadHTML*() and *trimArray*() methods in the Toolkit class for guidance on how to trim an array. When calculating the IDF, it is a bad idea to use the *String.contains*() method.

## 5.4 Task 4.4 - *buildVocabulary*(*String*[] *_cleanedContents*) [10 Marks]

Complete this method to build the vocabulary list based on the cleaned contents. Ensure the vocabulary list contains unique words from all news articles instead of one document.

## 5.5 Task 4.5 - *newsSimilarity*(*int _newsIndex*) [15 Marks]

By providing the index of a particular news item (*_newsIndex*), this method computes the Cosine Similarity (CS) value between that specific news item and all other news items within the corpus. The results will be saved in a two-dimensional array – *mySimilarity*. In this table structure, the first column represents the index of the second

news article used to calculate the Cosine Similarity, and the second column represents the value of the actual CS.

For example, assuming three news (D1, D2 and D3) are in the corpus, and the *_newsIndex* is 0. Moreover, CS(D1, D1) = 1, CS(D1, D2) = 0.1 and CS(D1, D3) = 0.5. Then $mySimilarity[0][0] = 1$, $mySimilarity[1][0] = 2$, $mySimilarity[2][0] = 3$, $mySimilarity[0][1] = 1$, $mySimilarity[1][1] = 0.1$ and $mySimilarity[2][1] = 0.5$

Then perform a sorting task on *mySimilarity* in descending order based on the second column – the CS value. Finally, return the sorted *mySimilarity* array.

Complete this method accordingly.

Hint: use the *Vector* class completed previously to calculate the CS values.

### 5.6 Task 4.6 - *groupingResults(String _firstTitle, String _secondTitle)* [15 Marks]

By providing the title of two news articles (*_firstTitle* and *_secondTitle*), this method will:

1. Go through all the news articles' titles in the *newsTitles* variable to identify the associated indexes.
2. Use the identified indexes to retrieve the related *newsTFIDF* results.
3. Calculate the CS values of the related news pairs to do the news classification.
4. *arrayGroup*1 is used for storing the indexes of the news which should be put in the first group. Similarly, *arrayGroup*2 is used for storing the indexes of the news which should be put in the second group.

N.B. there is no guarantee that group 1 and group 2 contain an equal number of news. Despite this, ensure the size of *arrayGroup*1 and *arrayGroup*2 align with the correct grouping result.

Complete this method accordingly.

Hint: Consult Section 1.3 for a more comprehensive understanding of this grouping process.

## 6 Expected Output

If all tasks have been completed correctly, the output produced by the main() method should match the following (ignore the colour):

```
0 1 NASA launches Psyche mission to rare metal asteroid
6 0.45405 NASA to explore giant metal asteroid Psyche - and it could
   reveal secrets of solar system
```

```
16 0.2453 NASA mission to return with 'pristine' samples from asteroid
   'which could one day hit Earth'
12 0.24055 Sir Brian May 'immensely proud' to be part of Osiris-Rex
   asteroid sample team
2 0.23359 NASA reveals 'incredible' findings from asteroid that could
   explain origins of life on Earth
10 0.21138 Osiris-Rex's sample from asteroid Bennu will reveal secrets
   of our solar system
14 0.20059 Osiris-Rex: NASA returns sample from asteroid Bennu to Earth
18 0.10207 What did we learn from NASA's first ever public meeting on
   UFOs?
8 0.09976 Astronaut who accidentally broke record for longest time spent
   in space finally returns to Earth
4 0.08495 NASA's Mars rover finds surprising mud cracks that hint planet
   once supported life
There are 10 news in Group 1, and 10 in Group 2.
=====Group 1=====
[1] - NASA launches Psyche mission to rare metal asteroid
[3] - NASA reveals 'incredible' findings from asteroid that could
   explain origins of life on Earth
[5] - NASA's Mars rover finds surprising mud cracks that hint planet
   once supported life
[7] - NASA to explore giant metal asteroid Psyche - and it could reveal
   secrets of solar system
[9] - Astronaut who accidentally broke record for longest time spent in
   space finally returns to Earth
[11] - Osiris-Rex's sample from asteroid Bennu will reveal secrets of
   our solar system
[13] - Sir Brian May 'immensely proud' to be part of Osiris-Rex asteroid
   sample team
[15] - Osiris-Rex: NASA returns sample from asteroid Bennu to Earth
[17] - NASA mission to return with 'pristine' samples from asteroid
   'which could one day hit Earth'
[19] - What did we learn from NASA's first ever public meeting on UFOs?
=====Group 2=====
[2] - Crypto trading should be treated like a type of gambling,
   influential MPs say
[4] - Bitcoin tops $30,000 for first time in 10 months
[6] - Bitcoin suffers briefly after Tesla sells majority of its holdings
[8] - Bitcoin rallies slightly but still set for record losing streak
   after Terra 'stablecoin' collapse
[10] - Bitcoin and US tech stocks hammered as global flight from risk
   intensifies
[12] - Razzlekhan: Cryptocurrency worth billions seized after
   self-proclaimed 'Crocodile of Wall Street' arrested in connection
   with exchange hack
[14] - Bitcoin slides to five-month low amid wider sell-off
[16] - Bitcoin hits record high as launch of new fund opens up market to
   wider class of investors
[18] - Bitcoin faces biggest one-day slump since last year as China
   announces curbs
```

```
[20] - Bitcoin value falls below $6,000 - the lowest level since
    mid-November
```

# References


[1] T. Mikolov, I. Sutskever, and L. L. Quoc, "Learning the meaning behind words," *Google Open Source Blog*, 2016. [Online]. Available: https://opensource.googleblog.com/2013/08/learning-meaning-behind-words.html

[2] G. A. Miller and W. G. Charles, "Contextual correlates of semantic similarity," *Language and Cognitive Processes*, vol. 6, pp. 1–28, 1 1991. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/01690969108406936