Information Retrieval of a *Harry Potter* Dataset

Lino Virgen
Texas Tech University

Patrice Fote
Texas Tech University

***Abstract*—Using a dataset of summaries of Harry Potter chapters, this report describes in detail the processes required to perform queries. The dataset will go through a series of stages such as tokenization, filtering, lemmatization and stemming to prepare it for processing. Afterward, each term of an input query was ranked according to its frequency and inverse document frequency. Finally, a tf-idf score was calculated for each document. A cosine similarity matrix was also produced to illustrate the similarities between the documents.**

Keywords: *information retrieval, tf-idf, data analysis, Harry Potter, query*

I. Introduction

Information retrieval is done every single second all around the globe thanks to search engines such as Google. These search engines look up words and sentences through millions and millions of websites. Then, they present the results to the user according to a unique match score. This report will present a number of stages that are intended to retrieve and analyze a dataset of Harry Potter summary chapters utilizing Python and a number of imported modules.

Section II will describe the dataset that was used along the project and the reasons of its choosing.

Section III will describe the steps took toward ranking documents given an input query.

Section IV will describe a new taken approach for constructing a document vector space and its implementation of K-Means and hierarchical clustering.

II. Dataset

The dataset used to complete the task set for our project is a collection of summaries to the chapters of the books in the Harry Potter Series By author JK Rowling's with a concentration on the 1st 3 books in the series of 7 namely: Harry Potter: The Philosopher's Stone, Harry Potter: The Chamber of Secrets, Harry Potter: The Prisoner of Azkaban. It consists of 57 documents, each containing on average 1000 to 1376 words which is the reason we chose to work with it. Most chapters were obtained from SparkNotes diligently split into individual *txt* files in order to facilitate processing.

III. TF-IDF Ranking

The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Variations of the tf-idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query. One of the simplest ranking functions is computed by summing the tf-idf for each query term; many more sophisticated ranking functions are variants of this simple model.

A. Pre-Processing

Data pre-processing means preparing the data for analysis. Most unstructured datasets consist of long texts without a formal structure. In this case, all the Harry Potter chapter summaries are long paragraphs that are easy to read for humans, but not for computers. Therefore, each document must be “cleaned” so we can extract the desired information. To achieve it, each document passed to a series of four processes. An imported module, nltk or natural language toolkit, was used in every preprocessing stage.

First, each document was tokenized. Tokenization is the process of segmenting strings in small units. Hence, each document was broken down into words and these words were put into an array. At this point, every word can be worked with individually for the following processes.

Second, each document was filtered. As most texts, punctuation and stop words (i.e. and, for, to, of, etc.) are abundant and comprise a big percentage the total word count. Moreover, they are non-relevant to the performed search. Therefore, they were all filtered from each document array.

Third, each document was lemmatized. This is process consists of changing every word to its base form. For example, “better” would be changed to “good” and “playing” would be changed to “play”. This step is necessary so that a word can be found regarding of its form.

Forth, each document was stemmed. Stemming consists of changing a word to a root form. Similar to lemmatization, stemming tries to get rid of derived words and produce terms that are easier to work with. For our project, the Porter method was used to stem every word in the array.

B. Query

In order to test the tf-idf ranking project, a query of 11 terms was came up with. The query was "magic owl wand study train school learn class fear teacher friend". In this manner, the ranker would check every term in the query and determine the term frequency and rarity. These processes are defined in the following subsection.

C. Term Frequency and Inverse Document Frequency

According to Jin [1], the term frequency or tf "of term *t* in document *d* is the number of times that *t* occurs in *d*." This property shows quantitatively how abundant a term is in each document. Although a high term frequency denotes relevance, it does not mean a document is more informative to the user. Hence the inverse document frequency is also calculated. Document frequency or df is "the number of documents *N* that contain *t*." The larger df is, the less important that term is. This is due to the commonality of the term and the irrelevance it has compared to other rare and key terms.

The term frequency of each term in the query was calculated by simply counting how many times a term appeared in each document. As shown in Figure 1, the most frequent term among all the documents was *magic* while the least frequent was *study*.

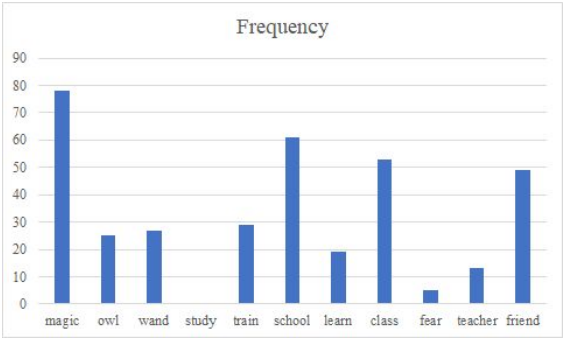

Figure 1. Total term frequency of each term in the query.

The inverse document frequency of each document was calculated as follows. First, for each term, the number of documents where it appeared was summed. Then the idf of each term was calculated utilizing the formula:

$$\text{idf} = \log10(N/\text{df})$$

The log function has as function to "dampen" the magnitude of idf since the rarity of a term does not grow proportionally with the number of documents matched. Figure 2 is a visualization of the idf scores. Thus, it can be noted that the term *fear* had a higher "rarity" score compared to the rest.

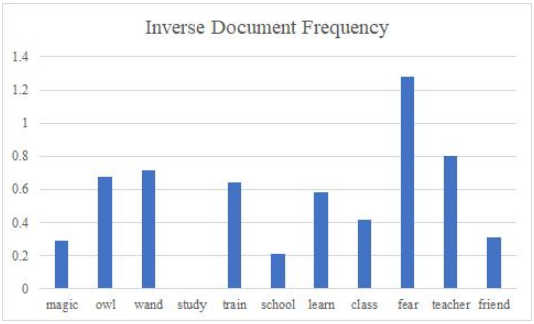

Figure 2. Inverse document frequency of each term in the query.

D. TF-IDF

Also called the term frequency-inverse document frequency, it can be seen as a statistical measure used to evaluate how important a word is to a document in a collection or corpus. In our example, we can observe with the rarity and frequency of the word *fear* in our query of 11 words from the Harry Potter Chapter summary dataset. In order to calculate the tf-idf weight, we first compute the normalized Term Frequency (TF) (the number of times a word appears in a document) divided by the total number of words in that document. With an average number of words per chapter being 1000 words, we measure how frequent a term occurs in that document. As seen in *Figure 1,* the rarity of words like *study* or *fear* is low while the frequency of *magic* is high. Since every document is different in length, it is possible that a term would appear much more times in long documents like chapters 5 and 57 would display a higher probability for the words *fear* or *teacher,* for instance than shorter ones as seen in short chapters like chapters 3 and 51. Thus, the term frequency is divided by the document length (the total number of terms in the document) as a way of normalization; the second part is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears like the words *magic*, *class*, *friend* or *school* seen in *Figure 1* to have the highest occurrence in our dataset which measures how important a term is. Whereas in computing TF, all terms are considered equally important in finding the inverse document frequency it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance, therefore, our setup will need to weigh down the frequent terms while scaling up the rare ones like *fear*, *study*, and *teacher*, by computing the following: IDF(t) = log10(Total number of documents / Number of documents with term t in it)

It can be noted in *Figure 3* that chapter 5 has the greatest matching score. Chapters 6, 32, and 57 are close behind. On the other hand, chapters 15 and 29 show no relevance to the query. For example, consider a chapter in our dataset document 57 containing 1000 words wherein the word study

appears 0 times. The term frequency (i.e., tf) for *study* is then (0 / 1000) = 0 Now, assume we have 56 more documents and the word study appears in 1 of these. Then, the inverse document frequency (i.e., idf) is calculated as log(57 / 1) = 1.75. Thus, the Tf-idf weight is the product of these quantities: 0 * 1.75 = 0 as seen in *Figure 1* and *Figure 2*.

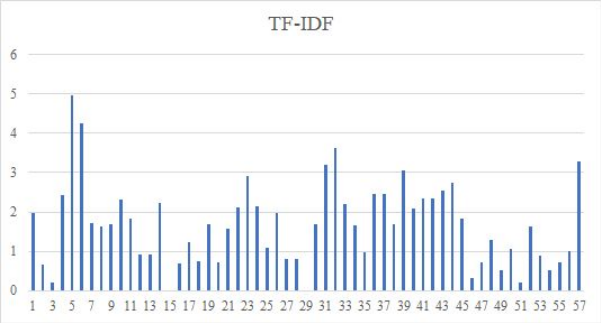

Figure 3. TF-IDF score of each document in the collection.

E. Vector Space

We model each document in vector space in order to facilitate tf-idf transformations. For that we calculate the tf-idf vector for the query, and compute the score of each document relative to this query, we multiply the tf scores by the idf values of each term, obtaining the matrix in *Figure 4*: (All the terms appeared only once in each document in our small collection, so the maximum value for normalization is 1.22 on the word fear" as seen in *Figure 2*. When computing the tf-idf values for the query terms we divide the frequency by the maximum frequency calculated and multiply with the idf values. TF-IDF values for each word of the document in our example and *N* is the dimension of the vectors:

$$\vec{a} \cdot \vec{b} = \|\vec{a}\| \|\vec{b}\| \cos\theta$$

$$\cos\theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|}$$

As observed, the definition of the dot product is a simple multiplication of each component from both vectors added together

F. *Cosine Similarity*

The cosine similarity between two vectors (or two documents on the Vector Space) is a measure that calculates the cosine of the angle between them. This metric is a measurement of orientation and not magnitude, it can be seen as a comparison between documents on a normalized space because we're not taking into the consideration only the magnitude of each word count (tf-idf) of each document, but the angle between the documents. What we have to do to build the cosine similarity equation is to solve the equation of the dot product for all vectors involved.

Figure 4 illustrates the similarity of all 57 chapters with each other. It can be observed that all chapters in the middle have a high or medium similarity while chapters at the beginning and at the end show very low similarity.

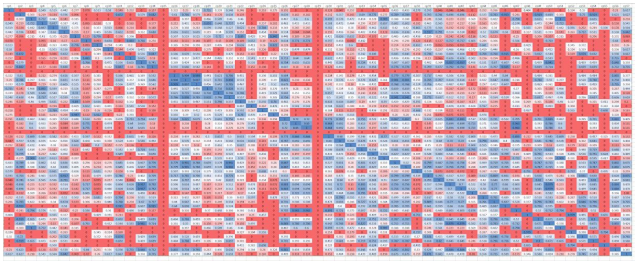

Figure 4. Heatmap showing the cosine similarity of the 57 documents against each other. Blue denotes high similarity, white denotes medium similarity and red denotes low similarity.

IV. Clustering

The second phase illustrated in this section required an increase in the number of documents and a different implementation of a document vector space in order to implement successful clustering methods.

A. New Dataset

This project required a minimum of 100 documents. Thus, it was decided to employ the actual Harry Potter books instead of chapter summaries. The chapters of the first 5 books were chosen for a total of 130 chapters (documents). Also, TXT files have been used previously, whereas this time PDF files were used for processing.

B. Preprocessing

This process was similar to the one described in the first project. However, since the dataset was comprised of PDF files *PyPDF2*, a PDF file reading module, was used instead of the standard Python file reader. This was important because many chapters contained pictures and other features that were not text.

Every document passed through tokenization, filtering, lemmatization, and stemming. Essentially, each document was broken down into individual words and put together in an array. Then, all punctuations were filtered out of such array. Having only words in the array, each word was converted to its original form to have a more standard set. Finally, each word was stemmed by removing unessential parts of it and account for similarities between other words.

C. Vector Space

In the last project, the vector space was created by utilizing the tf-idf score of each word in a document based on a given query or set of strings. Nevertheless, this second project is based on the overall similarities between the documents. Hence, the "given query" was a set of strings containing the vocabulary of all documents.

Moreover, the vector space for this project was constructed utilizing solely tf (term frequency) scores. The reason for this is that each book needs to be as distinct from each other as possible. In this manner, chapters from a particular book will be more similar to each other than to chapters in other books. This is supported by the fundamental idea of clustering: increasing similarity inside clusters and decreasing similarity between clusters.

The vector space was implemented using a dictionary of every word in each document with the count number of each document. Once the dictionary was set, each vector was constructed in an array using the term frequency in each chapter. Finally, each vector was normalized.

D. Cosine Similarity Matrix

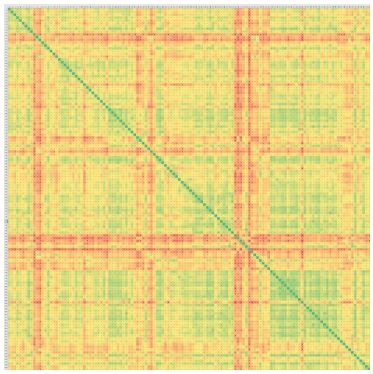

Figure 5. Cosine Similarity Matrix of 130 chapters

Creating a cosine similarity matrix was performed as in the previous project. Each document's vector representation was compared with each other using the cosine similarity matrix. Figure 5 is the formula that was used.

This cosine similarity matrix illustrates how similar documents are and should provide insight on how clusters will look like. Based on Figure 5, It was predicted that there were going to be squares of n * n documents that have high similarity between each other, where n would represent the amount of chapters in each book.

E. K-Means Clustering

The main task of project 2 was to implement two clustering algorithms, one of them being the K-Means clustering. The algorithm implemented was based on the following steps.

1. Initialize k centroids.
2. Assign vectors to the closest (higher cosine similarity) centroid.
3. Calculate the average vector of each cluster.
4. Repeat steps 2 and 3 until the centroids stop changing.

F. K-Means Clustering Visualizations

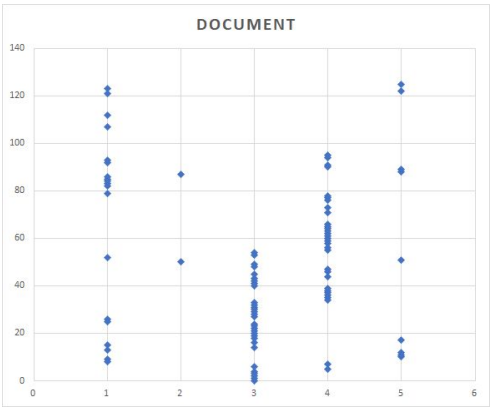

Figure 6. K-means clustering. K = 5. Initial centroids are proportionally distributed.

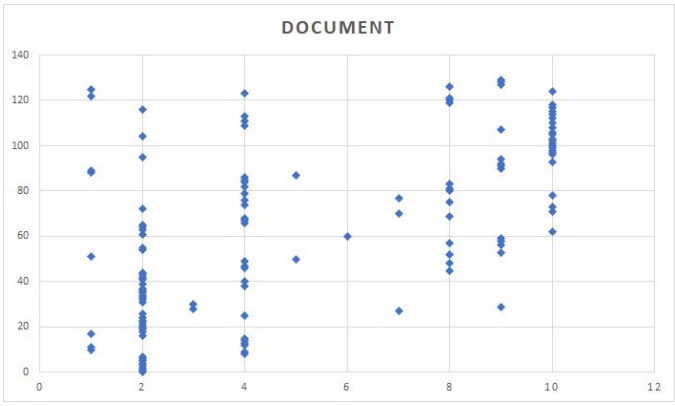

Figure 7. K-means clustering. K = 10. Initial centroids are proportionally distributed.

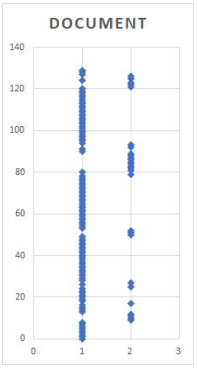

Figure 8. K-means clustering. K = 2. Initial centroids are the vectors at the edges.

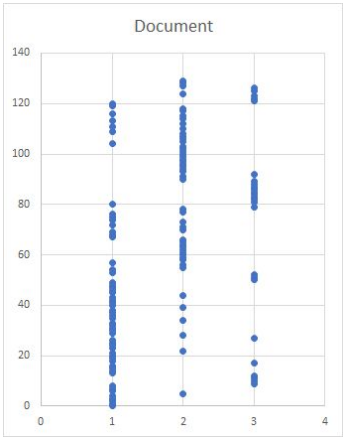

Figure 9. K-means clustering. K = 3. Initial centroids are the vectors at the edges.

For the first test of k-means clustering, k was 5 and the initial centroids were proportionally distributed among the number of vectors, which was 130. The result of this clustering can be appreciate it in Figure 6. The chart shows how each document was map to a cluster. It must be mention that the 130 documents are arranged in such a way that they are sorted by books and chapter. In other words, documents 1 – 20 represent the chapters in book 1, documents 21 – 40 represent the chapters in book 2, and so on. Therefore, the hypothesis was that the chart should have presented a step function. It was expected that chapters in the same book would cluster with each other, which was clearly not the case. The reason for this result is that the books are not different enough from each other to allow the implemented clustering algorithm to cluster them properly. Figure 7 illustrates the results of a clustering run with k equals to 10. It can be noted that, even though chapters from the same books are placed in different and varying clusters, adjacent chapters do tend to cluster with each other. This is also noticeable in Figures 8 and 9, where k is 2 and 3 respectively. In both figures the initial centroids were changed from proportionally distributed to the edges of the vector space. However, clustering did not seem to change noticeably.

G. Hierarchical Clustering

The hierarchical clustering was done using the same documents. There were 4 different approaches taken for this hierarchical clustering. They were all developed and visualized using libraries in R.

H. Hierarchical Clustering Visualizations

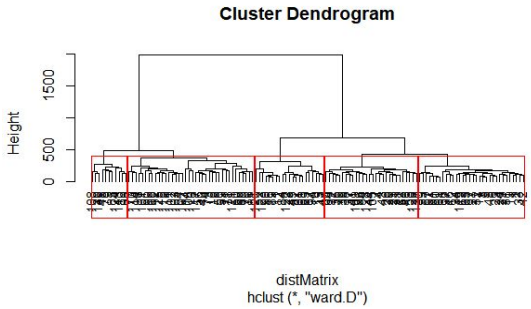

Figure 10. Hierarchical clustering. Ward's Method.

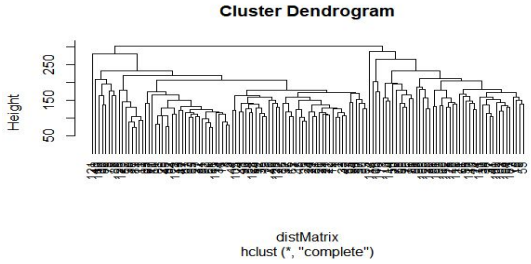

Figure 11. Hierarchical clustering. Complete Linkage.

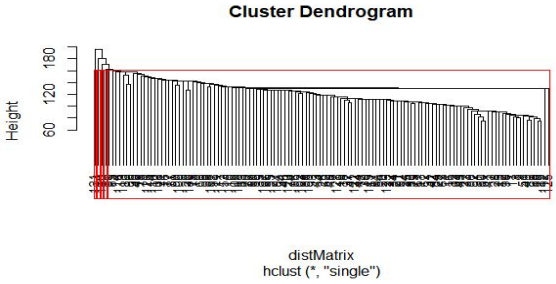

Figure 12. Hierarchical clustering. Single Linkage.

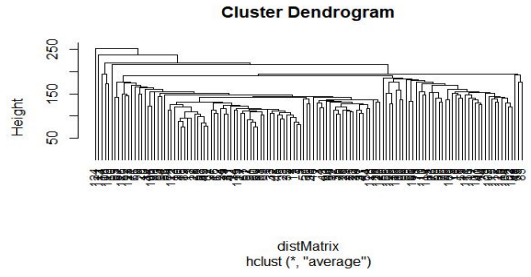

Figure 13. Hierarchical clustering. Average Linkage.

Based on Figures 9 - 13, the best hierarchical clustering result was done by the Ward's method. This one had 5 recognizable clusters which contained a similar

number of documents each. The rest of the approaches had less obvious clusters with few documents in each of them.

References

[1] F. Jin, "TF-IDF and Vector Space." CS 5364 - Information Retrieval. PowerPoint. Computer Science Department of Texas Tech University. Fall 2018