COMP6237 DATA MINING

Coursework 2: Understanding Data

Richa Ranjan
University of Southampton
rr2n17@soton.ac.uk.com

ABSTRACT

This paper provides an insight into performing exploratory and descriptive data mining on 24 text books. The idea is to explore relationships between these text data items, using standard data mining techniques, such as *clustering*. The results show how these text books are related, in terms of content, as well as common writing styles of some authors.

1 INTRODUCTION

The data to be analyzed is a corpus of 24 text books, written about Antiquity, around the 18th and 19th centuries. The data is in the form of HTML document for each page of the book, where the original books have been scanned to get their respective OCR images, produced as part of the Google Books Library Project. This article provides details of processing this data and exploring some interesting relationships between each text.

2 PRE-PROCESSING

Parsing: To extract all text data from HTML files, the Python library *Beautiful Soup* was used, with an HTML parser.[1] Iterating through the *span* elements of all HTML files, only the text items were extracted using `get_text()` function, excluding the titles and other details. The `ocr_cinfo` class was used because the text items were placed in this class, and each word was separately placed in different `span` tags. This was helpful in avoiding unwanted mergers between words of subsequent page.

Tokenizing and Stemming: Basic sanitization of the text was done using the *regular expressions* from Python's `re` library where extra whitespaces, unwanted newline characters, non-alphabetical characters were removed and the text was *tokenized* and *stemmed* using the `SnowballStemmer` library.

3 FEATURE EXTRACTION: [BOW / TF-IDF]

As both 'Bag of Words' and TF-IDF yield similar results, the *Term Frequency-Inverse Document Frequency* vectorizer was used for feature extraction. First, a *Term-Document Matrix* was defined to find out word occurrence counts in the documents considering *bigrams*. The minimum and maximum frequencies were specified to be 0.2 and 0.8 respectively, to consider the words that have at least occurred in 20% or 80% of the documents. This avoids insignificant words. Also, the *stopwords* used in "english" were removed. The TF-IDF weights were calculated where the words occurring frequently in a document, and less frequently in the corpus were assigned higher weights. This TF-IDF matrix is shown in figure 1.

As this was a huge sparse matrix, the *cosine dissimilarity* was calculated to be worked upon by further clustering techniques.[2]

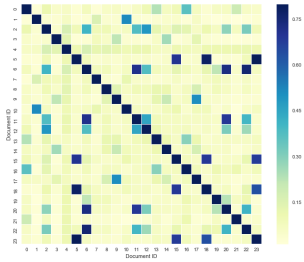

Figure 1: TF-IDF Document Similarity Matrix

4 CLUSTERING

Clustering techniques are used when there is no prior knowledge of what the clusters/groups should contain. Here, some of the techniques are described.

4.1 Agglomerative Hierarchical Clustering

In this bottom-up, "**Ward**" clustering method, the cosine dissimilarity matrix was used for **linkage criterion**, and at each step, the merges were recorded into a binary tree like structure, linking the clusters. This tree is known as a **Dendogram**.[3] `ward` and `dendogram` are both Python libraries. This clustering is shown in figure 2. The dendogram shows three main clusters, **Jewish**(related to

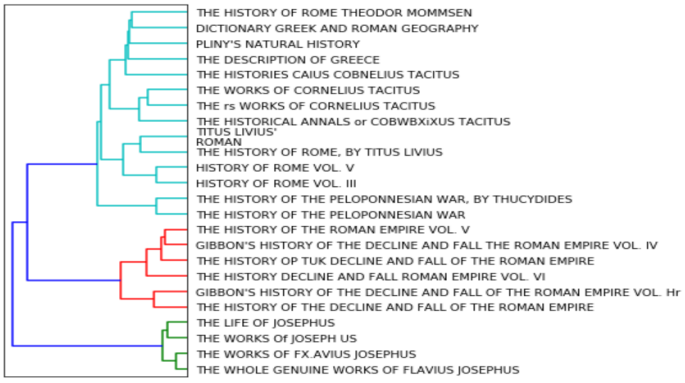

Figure 2: Dendogram of Agglomerative Hierarchical Clustering on cosine dissimilarity matrix

Josephus), **Roman**(Rise and Fall) and **Non-Roman**(*Peloponnesian*). It is found that the books titled "*The History of Roman Empire*" and "*The History of Rome*" are placed in different clusters. The documents are clustered on the basis of book titles, and not on the content. These titles, when slightly tweaked, are observed to change clusters quickly. The *Dictionary of Greek and Roman Geography* moved to a

far different cluster. *Gibbon's History of Rise and Fall* books belong to same main cluster, but as the dendogram branches out, the two volumes get separated to different branches, and different hierarchical levels. It is observed that with the insertion of words like "**of**", the items can change clusters. Thus the clusters are unstable and items are loosely bound. Moreover, the relationship between every pair of items are calculated and then recalculated when items are merged, making the algorithm slow.[3] Alternatively, a different approach called K-means is used.

4.2 K-Means Clustering

This featurespace clustering algorithm is used to group the data into predetermined 'K' clusters, where the items are iteratively assigned to their nearest centroids.[3] Initially, the K value is chosen as 5.

4.2.1 ***K-Means and MDS:***. With the **Multidimensional Scaling**, the cosine dissimilarity matrix is converted to a 2 dimensional array, for the purpose of plotting in 2D space. The result of K-means with MDS is shown in figure 3. The plot shows that based on the

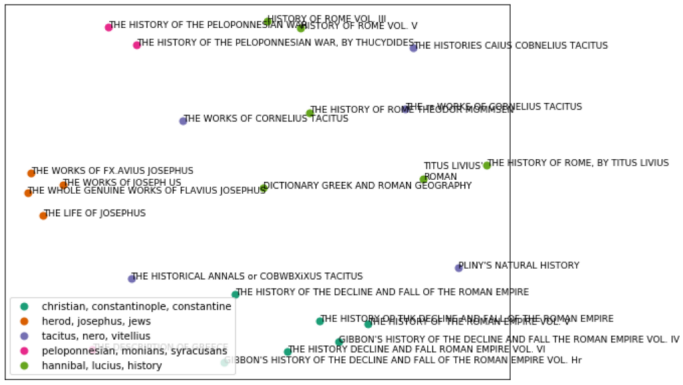

Figure 3: K-Means clustering with MDS

top words from each cluster like **herod, josephus, lucius, peloponnesian**, the books are clustered in a way that *The History of the Decline And Fall Of The Roman Empire* series belong to one cluster, *The History of Rome* belong to another, *Josephus* related works into another, and so on. The first 2 books belong to different clusters in spite of their similar titles, but because of the difference in contents, specified by the TF-IDF matrix and cosine dissimilarity.

4.2.2 ***K-Means and PCA:***. The **Principal Component Analysis** provides a more distinct view of the clusters(3, in this case). `StandardScaler` from Python's `sklearn` library was used to normalize the data. Also, if the K-value is changed to 3 and then plotted with PCA results, the clusters are as shown in figure 4. In PCA, the cluster 2 documents are densely packed. *Pliny's Natural History* and *Peloponnesian War* are placed in the same cluster within PCA, which is different from K-Means. Cluster 2 is tightly packed and owing to a greater Euclidean distance, the *Greek and Roman Dictionary* cannot be assigned to centroid 1 even after few iterations.

4.3 DBSCAN Clustering based on Authors

Another aspect worth investigating was, if the authors of these books had similar writing styles. Using the same cosine dissimilarity

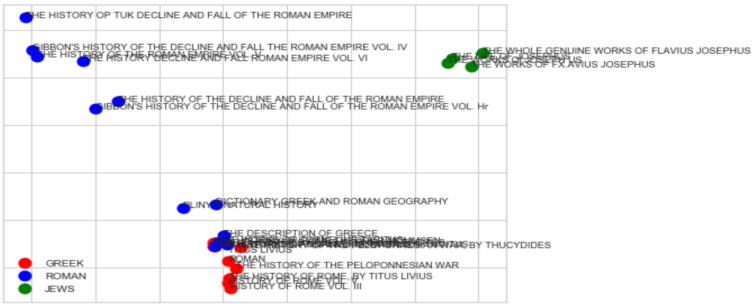

Figure 4: K-Means clustering with PCA

matrix, the authors were clustered with **Density-Based Spatial Clustering of Applications with Noise**. DBSCAN showed that *Thomas Bowdler*, *Edward Gibbon* and *William Whiston* belong to same (**Roman**) cluster, but in different clusters within PCA. This projects the idea that terms like **constantine**, **jerusalem**, **titus** map to multiple authors' books, inferring that these authors have similar writing styles. DBSCAN infers that *Arthur Murphy* uses more Jewish terms like **jerusalem**, **josephus** and so he belongs to the **Jews** books genre, and *William Smith* belongs to the **Roman** genre. Within PCA, these two belong to the same cluster, and are tightly bound. *George Baker* uses more **Non-Roman** terms and is at a farther distance from the Roman and Jews clusters. This could be a good piece of result for determining authors of same genre/style.

5 LSA, SVD, TSNE

Latent Semantic Analysis with **Singular Value Decomposition** provided interesting clustering patterns as well. LSA, when plotted against principal components, had books of similar genre pointing to similar directions, and the dissimilar ones were orthogonal, as explained by the cosine similarity.[4] Similar to above results, *The History of Rise and Fall of the Roman Empire* books had vectors pointing to similar directions, as opposed to *The Life of Josephus* which was orthogonal to these. **TSNE**, another visualization library from Python's `yellowbrick` package was used, and yielded similar visual representations of the above clusters.

6 CONCLUSION

Two different criteria for clustering documents can be title-based and content-based. The language style based clustering yields distinct clusters. K-means with dimensionality reduction using PCA is a better clustering technique for this type of data.
Future Enhancements: Other approaches like **Word2Vec** (to produce a high dimensional vector space), K-Nearest Neighbors and Latent Dirichlet Allocation (LDA) could be used as enhancements to gain some more interesting insights into text data.

REFERENCES

[1] https://www.crummy.com/software/BeautifulSoup/bs4/doc/
[2] https://web.stanford.edu/~jurafsky/slp3/15.pdf
[3] Toby Segaran. Programming Collective Intelligence: Building Smart Web 2.0 Applications. O'Reilly, 2007
[4] http://www.datascienceassn.org/sites/default/files/users/user1/lsa_presentation_final.pdf