Information Retrieval Final Project

Jose Millan - Rodney Ledesma
jampmil@gmail.com - reln13st@gmail.com
Data Mining & Knowledge Management Masters Program
Università degli Studi del Piemonte Orientale 'Amedeo Avogadro'
Alessandria, Italy

The current document presents the solutions for the Laboratories 1, 2 and 4 from the Information Retrieval course, based on the course from the Universitat Politècnica de Catalunya with the same name. The three laboratories focus in understanding Zipf'and Heaps' laws, working with tf-idf for documents and calculating the metrics for the Erdös-Rényi and Watts-Strogatz Network Models, respectively.

1 Lab1: Introduction to Lucene. Zipf's and Heaps' laws

1.1 Introduction

The first laboratory introduces Lucene [4] as a library for indexing and searching text files. Based on a given set of text files (novels and news articles), after processing these files it is possible to validate both Zipf's and Heaps laws for documents.

Zipf's law states that given some corpus of natural language utterances, the frequency of any word is inversely proportional to its rank in the frequency table. This rank-frequency distribution follows a power law $f = c * (rank + b)^a)$, where the triple (a, b, c) describes the distribution [9].

Heaps' law is an empirical law which describes the number of distinct words in a document (or set of documents) as a function of the document length (so called type-token relation). It can be formulated as $V_R(N) = K * N^\beta$ where V^R is the number of distinct words in an instance text of size n. K and β are free parameters determined empirically. With English text corpora, typically K is between 10 and 100, and β is between 0.4 and 0.6 [8].

1.2 Objectives

The main objective of this laboratory is to validate that Zipf's and Heaps' laws hold using a corpus of documents. This work also aims to get familiar with Lucene [4], as a Java library for indexing and searching text files, along with Luke [5], a development and diagnostic tool, which accesses already existing Lucene indexes.

1.3 Implementation

Using a collection of novels given for this laboratory, Lucene was used to index it. Additionally, with the given tool `CountWords3.java`, the unique words were extracted from the index previously created, along with their frequency in the texts. After this, a manual work of removing non-proper terms such as numbers, url's, binary or unreadable text, dates, etc, was performed over the results obtained.

Having this, the challenge now is to validate both Zipf's and Heaps' laws. For the case of the Zipf's law, and using the frequencies previously found, a Microsoft Excel spreadsheet (`.\src_lab1\zipf_heaps_validation.xlsx`) was used to calculate the rank-frequency distribution and plot the results. For finding the correct parameters of this distribution, a small R script (`.\src_lab1\calculate_zipf_heaps_parameters.R`) was created to find the correct parameters of the power law (a, b, c), using a Nonlinear Least Squares estimation (`nls` function from the `stats` R package) [7].

On the other hand, for validating the Hips' law, the collection of novels was divided in three different collections, with varying amounts of books, and then, using the same spreadsheet (`.\src_lab1\zipf_heaps_validation.xlsx`), along with the R script (`.\src_lab1\calculate_zipf_heaps_parameters.R`), the values of K and β for the Heaps' law were found.

1.4 Results

Using the procedure explained in Section 1.3, we validated the Zipf's and Heaps' laws. For the Zip'f law we found the best power law parameters to be $a = -1.918939$, $b = 1217.086$ and $c = 1023224000$, with a residual sum-of-squares of 1999. With this results, the logarithmic distribution of the frequencies can be appreciated in Figure 1, where the found distribution closely approximates the original, specially in the for the long tail.

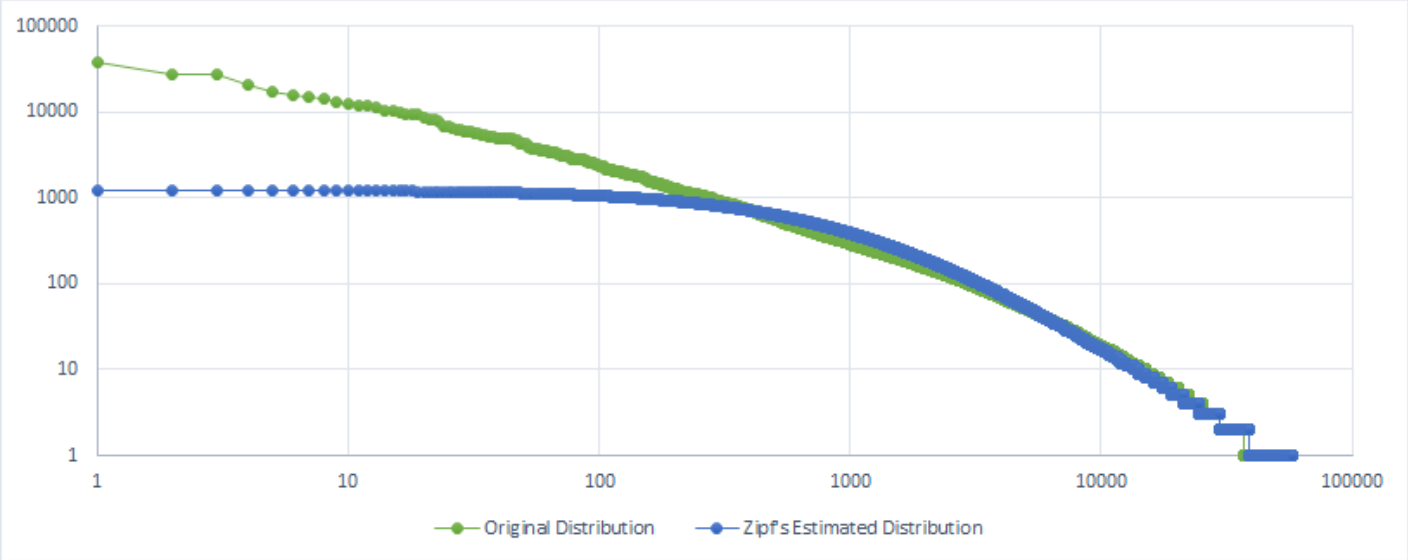

Figure 1: Zipf's Law Logarithmic Distributions

For the case of the Heaps' law, the parameters found were $k = 40.2453$ and $\beta = 0.4885$ with a residual sum-of-squares of 4549678, which correctly approximates the total number of words in the collections, as it can be seen in Table 1.

Table 1: Heaps' Law Results

Collection	Words (unique)	Words (count)	Heaps'
1	365124	19310	20988
2	94563	11901	10848
3	707227	29780	28988

1.5 Conclusions

Based on the results obtained, we were able to appreciate that given a random text corpus both Zipf's and Heap's laws hold. With the graphical representation of the words frequency distribution, along with the parameters found with a residual sum-of-squares of 1999 and 4549678 for the Zipf's and Heaps' laws' parameters' approximation respectively, we can conclude that for the given collections the laws hold.

The parameters of the laws can be correctly approximated (using different methods, in our case a Nonlinear Least Squares approximation) to fit the data. It is also relevant to realize that these parameters can further be used for obtaining predictions of the distributions of different texts within the same kind.

2 Lab2: Programming on Lucene

2.1 Introduction

The second laboratory brings in the tf-idf weighting scheme. Tf-idf stands for term frequency-inverse document frequency. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. 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 [1].

Typically, the tf-idf weight is composed by two terms:

- TF: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often normalized.
- IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

2.2 Objectives

The objective of the laboratory is to understand the tf-idf weight scheme for representing documents as vectors and the cosine similarity measure. The result will be a program that reads two files, computes the tf-idf representation and computes the cosine similarity of those two vectors.

2.3 Implementation

The demo of Lucene library, version 3.6.2, does not store the term frequency vector, so it is required to change a line in the source code. After compiling, we created the indexes of the novels provided in the previous laboratory. Then, we completed the `TfIdfViewer.java` provided file (located in `.\src_lab2\`).

The main function of the `TfIdfViewer` class is `toTfIdf()`, where completing the part of getting the weights was necessary. For each term, we get the document frequency using the Lucene function. Then, we normalize the term frequency dividing by the maximum frequency. To get the inverse document frequency, we divide the number of documents by the document frequency and we take the logarithm of the result. Finally, we multiply the term frequency by the inverse document frequency.

Additionally to this, the function `cosineSimilarity()` was also required to complete. First, we normalize the term vectors dividing each weight by the square root of individual weights squared. Then, we compute the inner product, comparing each term of the two documents. If the compared pair is equal, we add the multiplication of their weights. The resulting sum is the cosine similarity.

2.4 Results

We made some experiments with a few documents containing a small number of words. And the program calculated correctly the similarities. Then we tested with the provided dataset of novels and the Table 2 shows some comparisons.

The first row shows that comparing the same document the similarity is 1, as it should be. Comparing two volumes of the works of Poe, the result is 0.2354, so they are similar enough to say that they are from the same author and the same genre, but no too much to say they are repetitive. Comparing two books of Wells, it appears that they are almost different. However, comparing his book, *Time Machine* with the second volume of the works of Poe, they have a 10% of similarity. Finally, comparing two completely different books, as Darwin's *Origin of Species* with Dickens' *A Christmas Carol*, the similarity is almost zero.

Table 2: Cosine similarities

Document 1	Document 2	Cosine Similarity
PoeWorksVol1.txt	PoeWorksVol1.txt	1.0000
PoeWorksVol1.txt	PoeWorksVol2.txt	0.2354
WellsTimeMachine.txt	WellsWarofTheWorlds.txt	0.0772
WellsTimeMachine.txt	PoeWorksVol2.txt	0.1041
DickensAChristmasCarol.txt	DarwinOriginofSpecies.txt	0.0032

2.5 Conclusions

After testing with small documents and a set of know novels, we can confirm that the program correctly calculates the tf-idf weights and the cosine similarity. Comparing a document with itself, the similarity is maximum, and comparing two unrelated documents the similarity is almost null.

The tf-idf is used for a lot of purposes in Information Retrieval and Text Mining, such as scoring and ranking a document's relevance in search engines, or stop-words filtering. Besides, it is easy to implement, as proved in this work. The only difficulty of this laboratory was that the provided code uses an outdated version of Lucene and it was necessary to change the source code.

3 Lab4: Network Metrics

3.1 Introduction

The final laboratory introduces two different graph models: the Erdös-Renyi and the Watts-Strogatz models. Evaluating different metrics for each one of the models it is possible to understand the results seen in class and reproduce them.

The Erdös-Renyi model (ER model) takes two parameters: n, the number of vertices in the resulting network, and p, the probability of having an edge between any two pairs of nodes. A graph following this model is generated by connecting pairs of vertices with probability p, independently for each pair of vertices. Erdös-Renyi graphs have $\binom{n}{2} * p$ edges in expectation.

Similarly, the Watts-Strogatz model (WS model) takes two parameters as well: n, the number of vertices in the resulting network, and p, the probability of random rewiring edges in the initial network. A graph following this model is generated by initially laying all nodes out in a circle, and connecting each node to its four closest nodes. After that, we randomly reconnect each edge with probability p.

3.2 Objectives

The main purpose of this laboratory is to compute several different metrics for the ER and WS models. With these metrics, then the results of the graphs introduced in class are reproduced.

3.3 Implementation

In order to be able to compute the different metrics for the two models and plot the results, a small Python script, using a Jupyter Notebook [6], was developed (`.\src_lab4\lab4.ipynb`). These development was possible by using the `networkx` [2] Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.

For the EM model the first step to be done was computing the average shortest path as a function of number of nodes. As defined by Erdős and Rényi [3], if the probability of having an edge between any two pairs of nodes $p > \frac{(1+\epsilon)ln(n)}{n}$ then a graph will almost surely be connected. For our case we use this formula as the probability, with an $\epsilon = 0.8$.

With this probabilities there were no problems computing the metric (which fails to calculate if the graph is disconnected). Then for each number of nodes the corresponding value of average shortest path were calculated.

In the case of the WS model, a fixed number of nodes (600) and a fixed number of neighbors (5) for each node were used. Then for a given vector of rewiring probabilities, the normalized average shortest path and normalized clustering coefficient were computed as it was described in the lab.

3.4 Results

With the approach defined in Section 3.3, the two graphs introduced in class were reproduced. In Figure 2 it can be observed that with the increase of the graph node number, the average shortest path increases rapidly at first, but then stabilizes with higher values of n.

On the other hand, in Figure 3 it can be appreciated the results of the WS model, where the clustering coefficient and the average shortest-path length are plotted as a function of the parameter p.

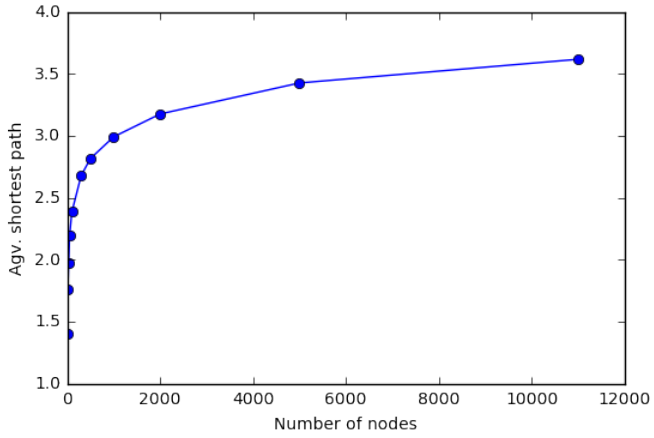

Figure 2: Erdös-Renyi Model Results

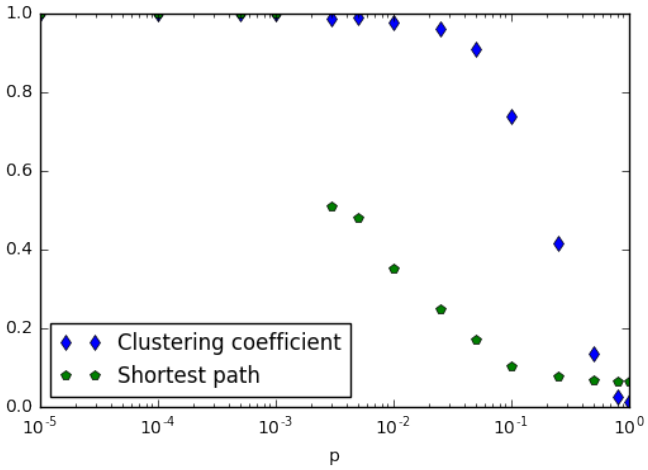

Figure 3: Watts-Strogatz Model Results

3.5 Conclusions

With the parameters used to model the two different kind of graphs, we were able to reproduce the models seen in class. For the EM model, the main challenge was defining a good way to calculate the probability of having an edge between two pairs of nodes. Different approaches were used (including having $p = E/(N * (N-1)/2)$ as seen in class), nevertheless the final approach is able to generate a complete graph, corresponding with the expected results.

For the WS model, the development was straight forward, where the `networkx` library helped considerably for the implementation of the model. Different trials for the number of nodes were performed, nevertheless the final result with $n = 600$ gave us the correct results.

References

[1] Tf-idf: A single-page tutorial - information retrieval and text mining. http://www.tfidf.com/. [Online; accessed 05-02-2017].

[2] NetworkX developer team. NetworkX. https://networkx.github.io/, 2017. [Online; accessed 04-02-2017].

[3] P. Erdős and A Rényi. On the evolution of random graphs. In *Publication of the Mathematical Institute of the Hungarian Academy of Sciences*, pages 17–61, 1960.

[4] The Apache Software Foundation. Apache lucene. http://lucene.apache.org/, 2016. [Online; accessed 04-02-2017].

[5] Google. Google code archive - luke. https://code.google.com/archive/p/luke/, 2016. [Online; accessed 04-02-2017].

[6] Project Jupyter. Jupyter Notebook. http://jupyter.org/, 2017. [Online; accessed 04-02-2017].

[7] R Core Team and contributors worldwide. Nonlinear Least Squares. https://stat.ethz.ch/R-manual/R-devel/library/stats/html/nls.html, 2017. [Online; accessed 04-02-2017].

[8] Wikipedia. Heaps' law. https://en.wikipedia.org/wiki/Heaps'_law, 2017. [Online; accessed 04-02-2017].

[9] Wikipedia. Zipf's law. https://en.wikipedia.org/wiki/Zipf's_law, 2017. [Online; accessed 04-02-2017].