REPORT: ElasticSearch and Zipf's and Heap's laws

CAIM - FIB - UPC

Q1 - 2019/20

BERNAT GENÉ ŠKRABEC

ORIOL VIDAL TERUEL

GROUP 11

Zipf's Law

Study

Zipf's Law is an empirical law which mainly states that, given a sufficiently large text in any language, the frequency of a word is inversely proportional to its position in the frequency table. Said rank-frequency distribution follows a power law, i.e., for some a,b,c, the distribution will have the shape of $f = \frac{c}{(rank+b)^a}$. In this lab exercise we wish to test the validity of this claim.

For this endeavour, we used ElasticSearch to index three large corpora of text (using *IndexFiles.py* script). After that, we used a Python script named *CountWords.py* that lists every word that appears at least once in the corpus, together with its frequency.

However, there are many terms counted by the script which are not proper words, such as numbers, url's, dates, etc., which may interfere with the natural distribution. For this reason, we wrote a script named *remove.py* (attached to the delivery file) that will leave only proper words.
Then, to check if the distribution actually follows a power law, we plotted it in a log-log scale, and used the Curve fitting tools provided by **scipy**, to find the parameters which would describe a fitting function.

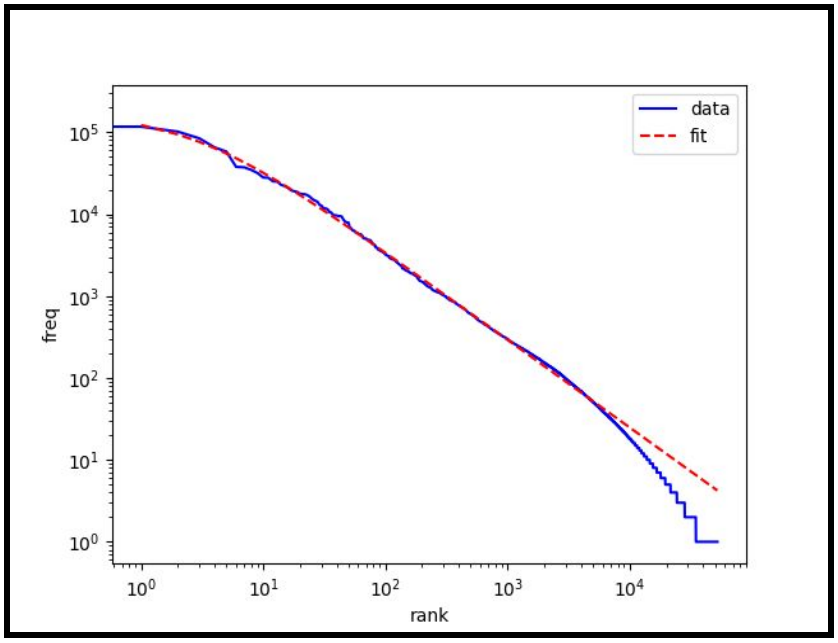
```
def f(rank, a, b, c):
    return c / ((rank + b)**a)

if len(sys.argv) < 2:
    print("Usage: script.py [fitxer]")

lastFreq = 0
lis = []
for line in reversed(list(open(sys.argv[1]))):
    for word in line.split():
        if word[-1] == ',':
            if word[:-1] != lastFreq:
                lastFreq = int(word[:-1])
                lis.append(lastFreq)

plt.loglog(lis, 'b-', label='data')

xdata = np.linspace(50, len(lis) - 5, len(lis) - 55)
ydata = np.array(lis[50:-5])


popt, pcov = curve_fit(f, xdata, ydata)
plt.loglog(xdata, f(xdata, *popt), 'r--', label='fit')
```

Screenshot of *showZipfsDistr.py*

Results

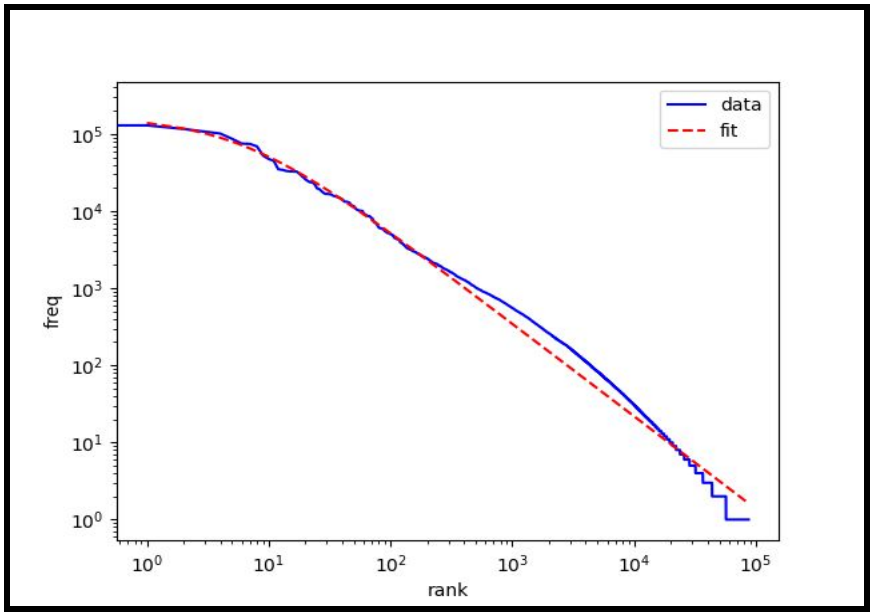
Unfortunately, in the first attempt studying Zipf's distribution we were unable to find the right parameters for the function that would fit our distribution in an acceptable way. We then ignored the first elements of the table, because we focused in obtaining a good fit for mid-frequency words, resulting in a satisfactory fitting. The parameters for the function are:

- a = 1.07193419e+00
- b = 2.60429762e+00
- c = 4.83202782e+05



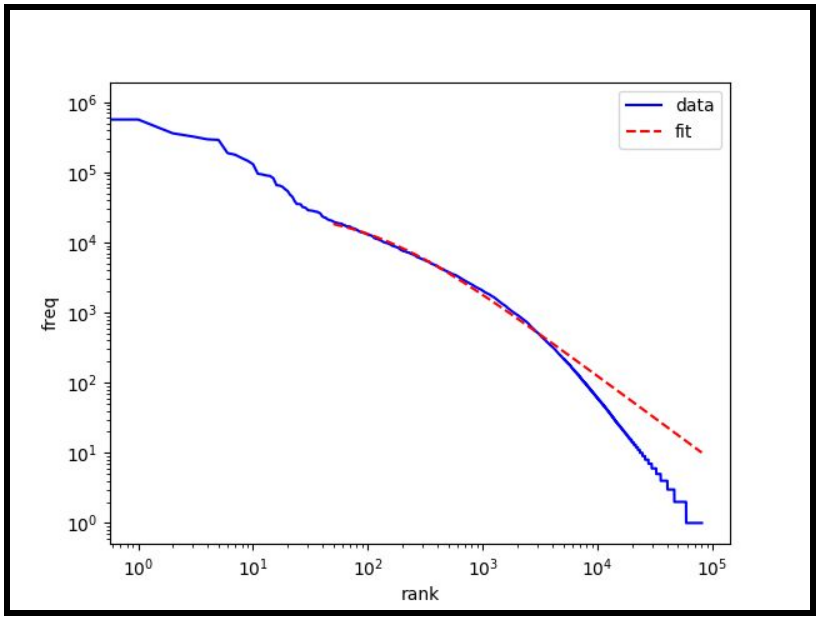
Zipf's distribution with Novels corpus

This was with the novels corpus, which is much cleaner than the other two. However, we followed the same steps for the other two corpora and those are the results:



Zipf's distribution with News corpus

As it can be observed in the image, we obtained a satisfactory fitting, with the parameters a = 1.20177874e+00, b = 5.83465943e+00, c = 1.40053812e+06.



Zipf's distribution with Arxiv corpus

With this corpus we had quite a harder time achieving the fitting, and the result is not completely satisfactory. To achieve it, we had to ignore several terms from both the head and the tail of the frequency table, specifically 50 for the most frequent words and 5 for the least ones, after making some tests with other values . The parameters a, b, c obtained are, respectively, 1.20860913e+00, 1.12084425e+02, 8.57899078e+06.

We think that the reason for the behaviour of this corpus is that there may be some forced structures in those texts, for example, a table were a usually infrequent word is repeated several times because of the nature of the paper. Therefore, the distribution may diverge from what it would be natural.

Conclusion

In conclusion, from what we gathered from this experiment, Zipf's Law seems to hold, but it holds better when the nature of the texts studied follows the natural structure of the language, just as it happens with novels, unlike other kind of texts, which have more irregular distributions.

Heaps' Law

Study

Heaps' Law is another empirical law regarding texts and frequencies of words, which states that for a given text of size N (number of total words), the number of distinct terms is $k \times N^{\beta}$ for some k and β.

In this case we only experimented with the novels corpus. To test the validity of the law, our plan was to have several indices of files of increasing size (of 1 file up to the maximum, 33). Then count the total number of terms and the number of distinct terms and plot the values for each index. After that, we used the Curve fitting tools to find the **k** and **β** parameters that would adhere to the empirical distribution.

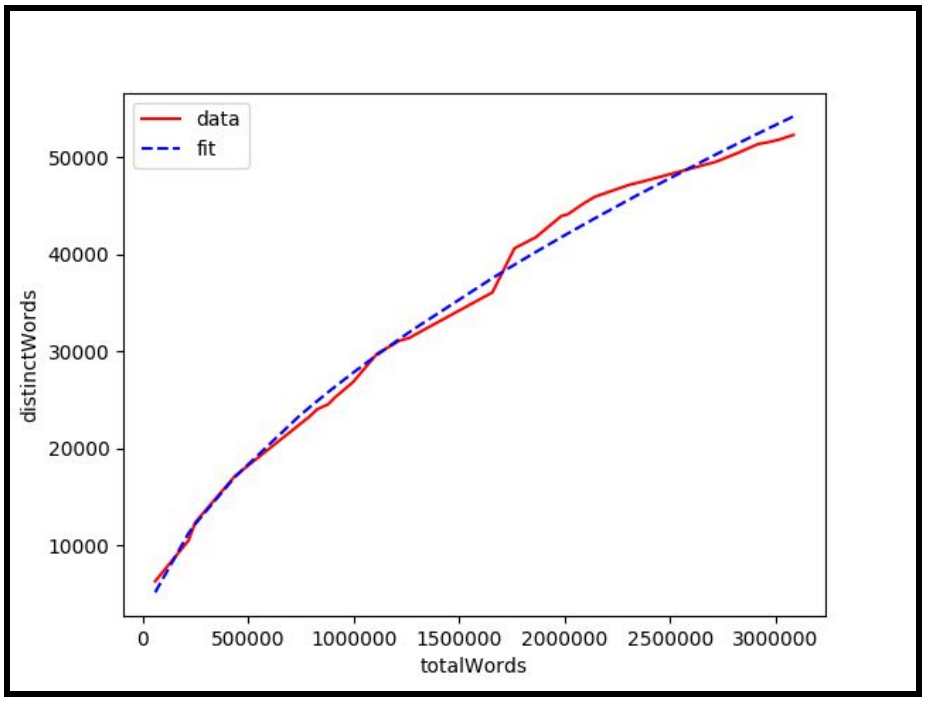
We modified the *IndexFiles.py* script so that it takes an additional argument, nfiles, which is the number of files to index.

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--path', required=True, default=None, help='Path to the files')
    parser.add_argument('--index', required=True, default=None, help='Index for the files')
    parser.add_argument('--nfiles', required=False, default=None, help='Number of files to be read')

    args = parser.parse_args()

    path = args.path
    index = args.index
    nfiles = int(args.nfiles)

    lfiles = generate_files_list(path)
    if nfiles != None:
        print('Indexing %d files'%nfiles)
    else:
        print('Indexing %d files'%len(lfiles))
    print('Reading files ...')
    # Reads all the documents in a directory tree and generates an index operation for each
    ldocs = []
    for f in lfiles:
        if nfiles <= 0:
            break
        nfiles -= 1
        ftxt = codecs.open(f, "r", encoding='iso-8859-1')
```

IndexFiles.py's modification

Then, we wrote a script *indexCountRemove.sh* that creates an index with one novel, another with two, another with three, and so on, until the largest index, which contains all the novels. Then, it counts the words in each index, removes unwanted terms, and reports the total number of terms and distinct terms of each index (calling *script2.py*). Both scripts are attached to the delivery file.

After that, we wrote *showHeapsDistr.py* that plots the pairs of values found and used the Curve fitting tools to find **k** and **β** parameters.



Heap's distribution using novels corpus

As we can see, with k=7.94593028 and β=0.59082388, we find a very good approximation.

Conclusions

We are generally satisfied with the results obtained, although we realized that the value of **k** obtained is not within the expected range, which is between 10 and 100, but somewhat close. On the other hand, **β** is between 0.4 and 0.6, as it was expected. We believe that this is due to the fact that in indices with few files, and therefore fewer words, the law is distributed in a more irregular way.