ElasticSearch and Zipf's and Heaps' laws

Josep de Cid

September 2018

Abstract

In this practice we are going to learn how does it work and how to use ElasticSearch and its APIs using Python alongside with Kibana to analyze the data, using information in the very first sections to fiddle with this new technologies and learn about the power of the inverted index of ElasticSearch. Lately we are going to dive into Zipf's and Heaps' laws and analyze them. All the source files are attached along with this document, having the classes to perform the analysis, and the main analysis code for Zipf's and Heaps' laws, in Jupyter Notebooks. Data insertion and counting words into a file must be done separately with the respective scripts provided in the source code.

Zipf's law

Before starting, we are going to consider two different methods to remove non-*proper* words. The first one, using an English dictionary and keeping only words that are part of it, and the second one, less restrictive, filtering and removing numbers, strange characters, URLs... using regular expressions.

By using regular expressions we can notice the only different words from the dictionary version, are some that appear very infrequently such as proper names or strange constructions or expressions that are negligible for the analysis giving, on the other side, it allows to appear some words with special characters or underscores that can be noisy and so henceforth, we are going to use the dictionary version **(fig. 1)**.

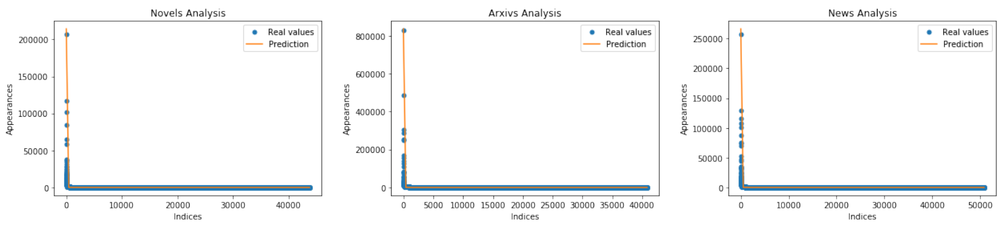


Figure 1: Distribution for novels, *arxiv* texts and news.

We can easily see that for the three different kind of indices, the distribution is pretty similar, and even the parameters may differ, the main idea and conclusions would be the same so we are going to focus on one type to simplify the process, (i.e. Novels).

As the assignment says, the very first (most frequent) terms are outliers which are noisy compared to the real curve fit. What really matters is the long tail. In the figure 2 we can compare a zoomed-on-tail fitting with all data and removing the first elements.

For choosing the most appropriate number of elements to behead, we will choose it empirically doing a trade-off minimizing both the number of elements to remove and the mean squared error. Only removing the most frequent term, the MSE drops drastically (from 1233212.9115 to 9154.1999). At 200 elements removed, MSE decreases minimally, so we will stop on there.

Even we are going to work with a naive approach, the best way to choose that $\hat{x}_{min}$ would be to minimize the following formula, being $S(x)$ the CDF for our observations and $P(X)$ the CDF for the power law that fits in the remaining region.

$$max_{x \geq x_{min}} \mid S(x) - P(x) \mid$$

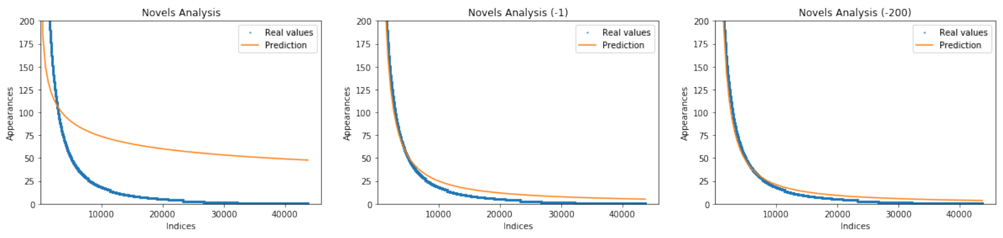


Figure 2: Zipf's curve fitting with all data, removing first elements and removing 200.

To check that our data follows a power law, we could just simply perform a log-log-scale plot of our data and see if it looks "straight enough" to be a power law, but this is not the best approach to follow, as such straight behavior is a necessary but by not sufficient condition [1].

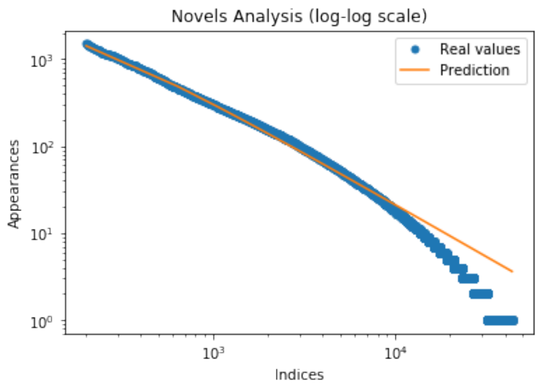


Figure 3: Zipf's fitting in log-log-scale that shows a straight-like shape.

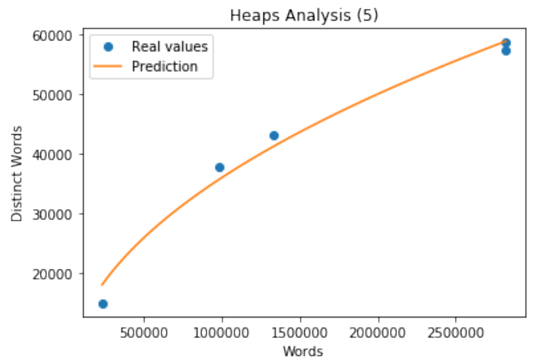
We need to estimate the parameters x_{min} and calculate the goodness-of-fit between the data and the power law. If the resulting $p-value > 0$, the power law is a plausible hypothesis for the data, otherwise it's rejected. To conclude, we have to compare the power law with alternative hypotheses via a likelihood ratio test. If every result is significantly different from zero, its sign indicates if it follows a power-law or not [2].

Therefore, it is correct to affirm that our data follows a Power-law distribution.

Heaps' law

For Heaps' law, we will deal only with novel documents, as the *corpora* is bigger and will lead us to better and more accurate results. To do that, we are going to generate some indices to store different amounts of text, merging novels that differ on text size, by just taking a subset of the *Power Set* of our novels.

As with "small" amounts of data we can have found a values for k and β that corresponds to an specific subset of our data and not to the whole *corpus*, let's see how does our constants values evolve as we take a greater amount of documents to fit a curve to our data **(fig. 4)**.



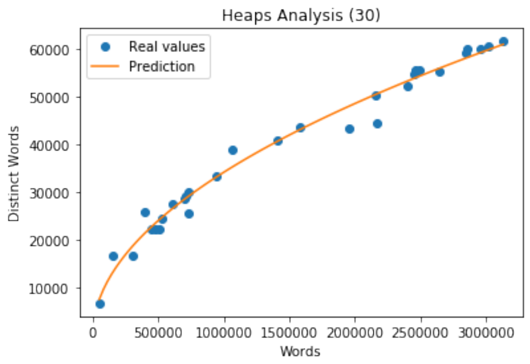


Figure 4: Heaps' function fitting with 5 and 30 subsets respectively.

For 30 subsets, we got the parameters $k = 31.6581$ and $\beta = 0.5057$. It is also curious to see that with English text corpora, k is usually between 10 and 100, and β goes from 0.4 to 0.6, so our data matches this conjecture [3].

We can easily observe that even if we increase the size of treated *corpus* and, as we have seen in lectures, the number of different words is described by a polynomial of degree less than 1 which fits our curve [4]. Again this can be seen by resorting to log-log plots and seeing the straightness of our curve **(fig. 5)** or applying similar methodologies as in previous section.

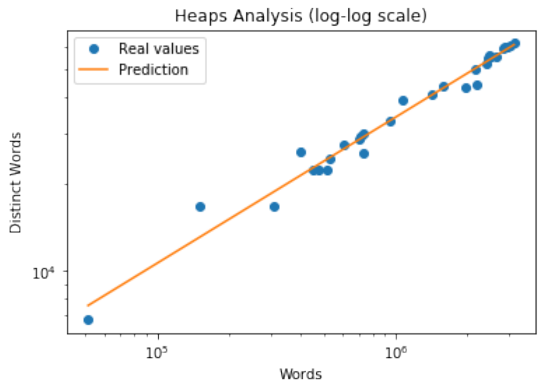


Figure 5: Heaps fitting in log-log-scale that shows a straight-like shape.

References

[1] Kevin Maher David Malone. Investigating the distribution of password choices. *Cryptography and Security*, Apr 2011.

[2] Clauset, Aaron, Cosma Rohilla, Newman, and M. E. J. Power-law distributions in empirical data. *Data Analysis, Statistics and Probability*, Feb 2009.

[3] The Stanford Natural Language Processing Group The Stanford NLP Group. Heaps' law: Estimating the number of terms, Apr 2009.

[4] Arias, Balcázar, Ferrer, and Gavaldá. Preprocessing, crawling and text laws, Sep 2018.