Wikipedia: Clustering similar attention curves

Large Scale Data Engineering: Final Report

Ilias Diamantakos
VU Amsterdam
2553162
2553162@student.vu.nl

Panagiotis Eustratiadis
VU Amsterdam
2573653
2573653@student.vu.nl

ABSTRACT

Each year Wikipedia publishes view page statistics for their projects. We define an attention curve as the interpolation of clicks a page received over time. In this paper we describe the process of clustering pages with similar attention curves, our methodology, approach, and difficulties we faced. For the implementation part of this study we use Apache Spark, which is state of the art in distributed computing, and a cluster of computers over SURFsara, provided to us by the VU.

General Terms

Big Data, Clustering, Data Analytics, Wikipedia

Keywords

Big data, Clustering, Data analytics, Data mining, Data visualization, Spark

1. INTRODUCTION

For our course assignment in Large Scale Data Engineering, we investigated and analysed Wikipedia published page statistics by using state of the art systems such as Hadoop - open source implementation of Map-Reduce - and Spark.

Wikipedia is a mature website, ranked among the ten most popular websites in the web, with 500 million unique visitors per month. It is a free-content Internet encyclopedia which contains over 38 million articles in over 250 different languages. There are many academic studies about Wikipedia and its social aspect, for example Wikipedia's population and the impact of false content.[9]. However recent studies have shown that Wikipedia's data can also be used for interesting behavior predictions such as recognising patterns in Wikipedia page views to devise stock market moves and strategies [7]. In our paper, we cluster pages that have similar attention curves. Our work is two-fold consisting well-researched topics namely Data clustering and Time series similarity measures.

In machine learning, data clustering is a fundamental learning procedure, either supervised, or unsupervised. Most algorithms that result in data clustering, partition the data into a pre-specified number of subsets or partitions k, and come in two flavors: soft and hard. The latter, which is our chosen variation for the nature of our problem, uses a definitive way of clustering the data such that each data entity is in one and only one cluster in contrast with soft clustering; where each entity has a probability of belonging in each of the clusters. Parametric clustering is much simpler to achieve and is also more scalable. Scalability, is something we had in mind from the beginning of our daunting task since it involved large scale data analytics, meaning traditional data processing applications become inadequate when facing such large and complex data sets. Our data set consists of roughly 830GB of data and about 17500 files where each file is a one hour interval of view count per Wikipedia page. The main tool to parallelise and distribute the data pre-processing and clustering is Spark [11], which runs over a Hadoop cluster that was offered to us for the purpose of the course by the VU, in cooperation with SURFsara. [4]

Clustering data involves transforming it into vectors, and requires a similarity or distance measure, also known as distortion function, such as squared Euclidean distance, Frechet distance and Dynamic Time Warping (DTW). We decided to try two distance measures in combination with clustering. One is Spark's *MLlib*, which is a machine learning library tailored to be used with data parallelisation. It offers the *k-means*|| clustering algorithm which is a scalable version of the *k-means++* [6] and uses the Euclidean distance. Our other approach is implementing our own version of *k-medoids* with DTW as a distance measure. One big challenge we had to overcome, was the missing entries when there was no view during an hour. While this issue is explained in detail later, essentially the problem was that we ended up with different vector lengths. DTW is a proper solution to this since it can be used when two sequences do not line up in X-axis. The method ”warps” the time axis of one (or both) sequences to achieve a better alignment.

2. RELATED WORK

Our work involved research on various and multidisciplinary topics. From Distributed Computing, Machine Learning and Data Mining to Distortion Measurements and Statistics. Following are bits & pieces from various papers we based our work upon, divided by topics.

2.1 FastDTW

As previously mentioned, one of the similarity measures we use is Dynamic Time Warping. The straightforward way to compute the algorithm is by constructing a warping matrix and searching for the optimal warping path. We can also introduce an adjustment window to eliminate the corner cases of the matrix since they is highly improbable. However the complexity is $O(nm)$, where n,m are the lengths of the vectors. When introducing the warping window w the time complexity reduces to *O(nmw)*. Even so, DTW is not very scalable so in our problem it would be a drawback. So, we based our implementation on FastDTW. FastDTW computes an approximation of the optimal path between the two vectors. The optimal path is, in it's normal version, "brute-forced". In the paper, the authors use a multilevel approach by first "downsampling" the time series. Then they find the (near) optimal path by incrementally projecting it to a higher sampled vector. This will continue until the (near) optimal path is found for the full resolution of the vector. They also prove that the approximation is accurate enough and the complexity reduces to *O(n)*. From this trade-off we are forced to pick the lesser complexity by sacrificing a bit of the accuracy.[10]

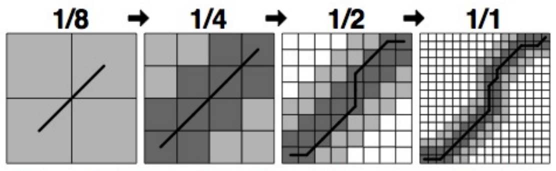

Figure 1: Depiction of multilayered approach

2.2 Time Series Classification

One related project we identified was Alex Minnaar's article on time series classification. Although our paper is not really based on his project, it provided us with some interesting insights during the start of our work. In the article, the author wants to create a predictor using time series usage information. His first approach was using the classic probabilistic classification algorithms in combination with typical statistic information. The other was clustering by *k-NN*. The *k-NN* clustering algorithm was also our initial though, however since Spark provided us with *k-means* we decided to use the latter instead. In our custom implementation we used a variation of k-means. This is because computing the mean of two series some possible vectors might be flattened out. However, the author of the article also uses DTW as a similarity measure in his clustering algorithm. [5]

3. RESEARCH QUESTIONS

We identified a series of research questions that motivate the work demonstrated on this paper. Those are demonstrated from both a technological as well as a social standpoint.

1. Can we identify differences or similarities in various clusters? Why and when are "trending" topics trending? This may lead to a deeper understanding of events and situations. E.g. Are "Sinterklaas" in the Netherlands and Santa Claus clustered together?

2. Is there correlation between worldwide events and the Wikipedia page clicks? Can we identify occurrences that are clustered together because of a incident that happened? E.g. In the event of an election The page of a politician and the page of his party.

3. About 55% of our Wikipedia dumps are entries with redirects [8]. Do the redirect entries correlate with the corresponding page, meaning are they clustered together?

4. How can we visualize our data in a way that useful or constructive conclusions may be extracted?

4. IMPLEMENTATION

To orchestrate the data transformations and reduces, we used the Python programming language with Spark also called *pySpark*. Our clustering implementations are also in *pySpark* and the visualization was created using the *d3* javascript library. The modular framework we developed is divided in the following steps. Our implementation can be found on [1]

4.1 Data pre-processing

The Wikipedia page view statistics dataset can be downloaded from the Wikipedia downloads directory. [3]

Data Overview. Our data is for the year 2014 and consists of 17500 files of about 830GB of data. There are two file types within the data set:

1. projectcounts: This file type provides with aggregated information of the corresponding pagecount file time-wise. Specifically, projectcount files have the total views per hour per project generated by summing up the entries in the pagecount files. For this paper these files were not used at all and were filtered out during our data-preprocessing step.

2. pagecounts: Every file contains the total page count for a specific hour for every page served to a user. The format is in CSV with *spaces* as separators and they are compressed using gzip. The files are of the following format:

pagecounts-{year}{month}{day}-{hour}{minute}{second}.gz

The time used in the filenames is in UTC timezone and each timestamp refers to the end of the log instead of the beginning. The columns in each file are: the Wikipedia project, the requested page name, the number of the requests for that page in the last hour and the size of the response. This information is explained in more details below:

The Wikipedia project is split in two parts. The first is an abbreviation of the language of the project and its pages, e.g. "en" for English, "gr" for Greek etc. The second part is optional and is the abbreviation of the name of the project prefixed by a dot. If there is no second part, then the project is the main project of the corresponding language. In case there is, the abbreviation describes the specific subproject of the language. (e.g. ".b" is for wikibooks).

The basic entity of Wikipedia is a page. Pages are plain text documents and can represent a variety of information. They can be customized using the wiki markup language which is then rendered upon user request. The main pages of Wikipedia are articles which contain encyclopedic information.

Wikipedia pages are subject to various namespaces depending on the service they provide. They are dictated with a prefix on the page followed by a colon. When there is no prefix, the page belongs to the main namespace which is where the articles are.

Many pages are redirect pages. These are content-less pages which redirect the user to another page, usually an article. They can represent aliases for articles, abbreviations, different yet still correct spellings but also misspellings. About half of the articles in Wikipedia are actually redirect pages. [8]

Challenges. One of our greatest challenges was essentially time. Many things would have been solved if we had much more time. The reason is that we identified many issues within the data set [2], creating an obstacle to proper classification of pages. Having in mind that clustering is heuristic based in conjunction with our bloated data set we realized things would get very "experimental". To make matters worse, the similarity measure, which is the most important link to our chain, must be tailored for the nature of our problem in order to be effective. And while these matters concern the methodology we used, implementation wise, we knew that it would be a great challenge to put the above components into play with such huge amount of data. For instance we first needed to rearrange the whole data set from the beginning in order to produce a vector for each of the Wikipedia pages.

In order to rotate our data set to create these vectors, by trial-and-error, we tried many different ways; the goal being to not waste too much time with this heavy procedure but also with little loss of data.

Let *f1* and *f2* the two files that we need to combine. We mapped and combined the files as key-pair RDD's as follows:

For every case we first filter out the corrupted or empty lines.

1.*(project, page), [value]* and *combineByKey*

First produce the union of *f1* and *f2*. Very importantly, note by doing this there is automatically loss of data which is the missing entries in either data set. Then do a *combineByKey* to aggregate the lists of values. This ended up being very slow since adding up lists comes with an overhead.

2.*(project, page), "value"* and *reduceByKey*

Same as 1, with the difference of using *reduceByKey* to add the two string values together. Surprisingly this method also was very slow. In contrast with *groupByKey* the workers first reduce what they already have locally since the types of input and ouput are the same and when they are done, they send their aggregated version to other workers. This means less shuffling but string addition proved to be a big overhead to negate the lesser shuffling.

3.*(project, page), value* and *groupByKey*

Same as 1 & 2 with the difference of using *groupByKey* to create key pairs of .*(project, page), iterable*. This method was eventually the fastest and our final choice to rotate the whole data set. It took us about 24 hours to complete the whole job, and we mainly had our jobs working during the night to not disturb other users too much. To do this we used 60 workers each having 20GB memory and 8 cores. One important issue is that we first aggregated each month, since many times the job failed for various reason such as connections, failures of workers etc. After having a data set for each month, we did the final aggregation to end up with one huge vector for each of our pages.

4.*(project, page), value* and *fullOuterJoin*

Finally, the most correct version with no data loss at all, was instead of using union to combine the two files use join and specifically *fullOuterJoin*. By doing *fullOuterJoin* for every file we would keep record of the missing entries since fullOuterJoin combines entries by key but when a key was missing from a file it inserts a *None* instead. This would let us know exactly where the missing entries were, substitute *None* with zero and end up with same vector lengths. Being the ideal solution, this method proved to be extremely time consuming. Our estimation is it would take more than 10 nights of trial-and-error to end up with the final rotated data set. However, due to course time constraints we were unable to use this method and needed to find different solutions to the loss of data which are in more detail explained below.

To solve the interpolation problem with tried various approaches. Firstly, as previously mentioned we tried to implement our own clustering algorithm to run on the SURFsara Hadoop cluster. This brings the interpolation problem to the DTW and leaves the similarity measure to "warp" the missing information. Another effort we made was the following: After rotating the data set we have as result 12 months of data for each month as well as the final aggregated data (let's call this *final-data*). We processed these 12 months and aggregated for each page the sum of the views. So for every page in e.g. January we have one value for the total views of that page. Then we aggregated those 12 months into one file ending up with a 12-value-vector for each page. Due to the problems explained earlier we were forced to use this data to perform our clustering and analysis. However, the initial idea was to use these 12-value-vectors to do a re-sampling of our *final-data*. We tried to do this "upsampling" by inserting zeros. However instead of randomly inserting zeros between actual values or inserting zeros in a normalised way, we had information we could use, to do this with more accuracy. Specifically, since we had the sum per month, we could slice the final vectors into 12 unequal parts depending on the summed up values. Then examine from the 12-value-vectors how the distribution is and devise weights for the "upsampling". However, this proved to be much harder than we though since we could not find a method that would without FFT (Fast Fourier Transformation) upsample the data. Specifically, the problem was that most common methods increased the rate by an integer. So this would either double, triple etc our vectors instead of normalizing them into equal lengths.

Finally, we faced many problems with SURFsara. For instance, many workers were missing modules such as *scipy* and *numpy*. Due to this we reimplemented some of our code in *Scala*. After communication with the helpdesk, the (computer) workers had their dependencies fixed so we were able to continue our daunting task.

4.2 Clustering

Abstractly, the clustering methodology initially planned were as follows:

1. Create a feature vector representation of the data.
2. Implement similarity measures between those vectors.
3. Implement a standard clustering algorithm using said similarity measures. There are two different approaches to that method: unsupervised clustering (where k is not given) and supervised clustering (where k is given).

However, limitations in our data, software, time, and know-how, lead us to follow a number of different approaches to our clustering methods.

Our initial goal was to cluster the dataset using a built-in clustering algorithm like K-means, because Spark already provided support for that. However, our dataset was not consistent, meaning that a lot of values were missing between files. That made the Euclidean square distance of K-means not of much use, because we had different length in vectors. We figured out that evenly distributing zeros was not a good idea, due to the enormous variance between the vector sizes.

Unfortunately, Spark does not provide a nice way to override or extend its classes, so our idea to override the Euclidean distance with DTW was short lived.

Our next approach was implementing our own clustering algorithm from scratch, following K-means and K-medoids. This proved to be extremely hard, because a lot of parallelization that is normally taken care of, needed to be implemented all over again. In the end, this idea was put aside, due to the amount of time spent solving issues as opposed to making actual progress.

After several issues (missing libraries from workers, downtime, etc) with the Hadoop cluster in SURFsara were solved, we went back to our initial idea of using Spark's built in clustering algorithms. We wanted to cluster tha dataset into 2, 3, 4, 5, 10, 15 and 20 clusters, in order to make a comparative study on the quality of the clustering when upscaling k. We went as far as training the relevant models, and taking out chunks of the clustered data in order to visualize them.

4.3 Visualisation

For our visualization we chose the following representation: We represent the Wikipedia pages as nodes in a forced layout, and color them according to the cluster in which they belong. On mouseover, we graph the attention curve so that our user might see the similarity in the clustered instances.

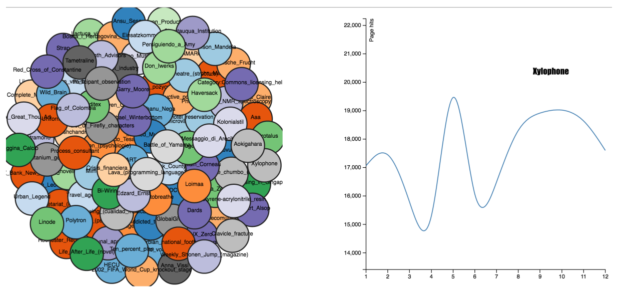

Figure 2: Visualisation of the page Xylophone

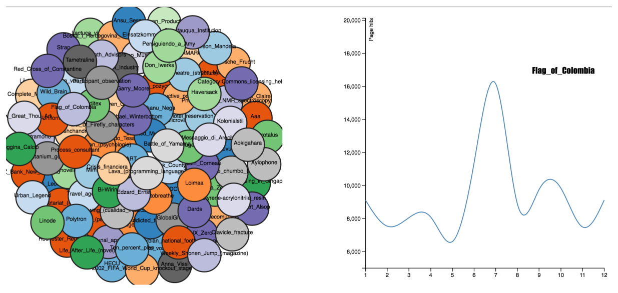

Figure 3: Visualisation of the page Flag of Colombia

A core and quite obvious problem of this approach is the cluttering of the data as the number of pages scales up. A less obvious but still essential problem that makes the visualization harder, is that the nodes are not grouped by color, but scattered around the layout instead.

5. RESULTS

While this project did not achieve excellence, there are aspects that gave us a better understanding of the problem, but more importantly, large scale data analysis.

This system has the potential to answer all of our research questions, given a better rework of the visualization module. While we failed to make optimal use of the tools provided to us, this problem could certainly be solved given more time or more experienced hands.

6. LIMITATIONS & FUTURE WORK

Many of the limitations are already previously mentioned and are mostly connected with the dataset itself and the restricted time we had for our work. The Analytics team tracks these known problems [2] such as redirect counts, packet loss etc. Another issue was that there were duplicate pages with different capitalisation such as "Main_page" and "main_page". These pages are counted twice. We could have firstly normalised the page names, however we would have to do a *reduceByKey* for every file in our data set before doing anything at all. This would be very complex so we did not filter out the duplicates.

Above all, what was extracted from this project is how time consuming it is to do computations on large scale data. Solving a single error could take three to four hours alone. As suggestion for next years we would recommend a smaller scope in more time available.

7. CONCLUSION

In this paper we described the process of clustering pages with similar attention curves. Moreover, we elaborated on similarity measures in combination with various clustering algorithms. While we were not able to make a great implementation due to many challenges we faced, we provided insights on different aspects of our problem. From scalability issues to data set problems that we faced, we were able to overcome some of them while failing to do others, either due to time limitation or our inexperience with the field.

8. REFERENCES

[1] 0xe1d1a/wikicurves. https://github.com/0xe1d1a/wikicurves.

[2] Analytics/Data/Pagecounts-raw - Wikitech.

[3] Page view statistics for Wikimedia projects. https://dumps.wikimedia.org/other/pagecounts-raw/.

[4] SURFsara. https://www.surf.nl/en/about-surf/subsidiaries/surfsara/.

[5] Time Series Classification and Clustering with Python. http://alexminnaar.com/time-series-classification-and-clustering-with-python.html.

[6] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii. Scalable K-means++. *Proc. VLDB Endow.*, 5(7):622–633, Mar. 2012.

[7] Helen Susannah Moat, Chester Curme, Adam Avakian, Dror Y. Kenett, H. Eugene Stanley, and Tobias Preis. Quantifying Wikipedia Usage Patterns Before Stock Market Moves. *Scientific Reports*, 3, 2013.

[8] B. M. Hill and A. Shaw. Consider the Redirect: A Missing Dimension of Wikipedia Research. In *Proceedings of The International Symposium on Open Collaboration*, OpenSym '14, pages 28:1–28:4, New York, NY, USA, 2014. ACM.

[9] R. Priedhorsky, J. Chen, S. T. K. Lam, K. Panciera, L. Terveen, and J. Riedl. Creating, destroying, and restoring value in wikipedia. In *Proceedings of the 2007 International ACM Conference on Supporting Group Work*, GROUP '07, pages 259–268, New York, NY, USA, 2007. ACM.

[10] S. Salvador and P. Chan. Toward Accurate Dynamic Time Warping in Linear Time and Space. *Intell. Data Anal.*, 11(5):561–580, Oct. 2007.

[11] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster Computing with Working Sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.