Unsupervised Corpus Partitioning of a Large Scale Search Engine

Jean-Paul Schmetz
Department of Computer Science
Stanford University
Stanford, CA 15213
jschmetz@stanford.edu

Abstract

Using a very large query logs we aim to partition a web scale document corpus in order to (1) efficiently shard the indexing to optimize the probability of results sharing ranking functions, (2) provide an efficient basis to classify the (latent) intent of a query and (3) provide additional useful features e.g. adult content filters, personal cluster page ranks, etc.

1 Introduction

Large scale search engines need to partition their indices into shards. The partitioning is done 'by document' (not 'by term') meaning that usually one entire document will be indexed in one shard (at least). Without optimization (something even with optimization), all shards would then be queried for each query. Scoring functions are applied at the shard level and the highest scoring subset of each shard is sent to a post-processing function that re-ranks and reduces the results and creates the result page. Large scale search engines (Google, Yandex, ...) are partitioning more intelligently (using different schemes: 'freshness', topics-related partitioning, quality-related partitioning etc...).

We would like to machine learn an optimal partitioning in an unsupervised way that attempts to respect the following constraints:

- given a document, we output in which partition(s) it needs to be indexed in.
- given a query, we know which partition(s) would most likely have the answer with some form of probability or ranking
- documents that co-occur on a search result page should also co-occur in one partition (so that they get scored with the same ranking function and the same context)
- we do not want to end up with 1 partition or one partition per domain.
- we should recognize some intuitive reason behind the partitioning (e.g. sites serving similar needs should land in the same partition etc...)

This would allow the system to truly respect the possible intents of a query and reflect it in the result page as well as offering the possibility to machine learn each partition's ranking function (outside of the scope of this paper). Ideally the optimal partition would be such that each basic intent of a query is answered by one partition using an optimized ranking function for that intent.

2 Data

As training data, we have a very large search query log from a large representative user panel (hundred of thousands of users) in Germany collected over 2 years. As a first simplification, we decided

Table 1: Raw data excerpt

Query	Domain	Count
miley cyrus	de.omg.yahoo.com	82
miley cyrus	de.twitter.com	3
miley cyrus	de.wikipedia.org	1682
miley cyrus	en.wikipedia.org	60
miley cyrus	es.wikipedia.org	3
30 seconds to mars	audioinkradio.com	1
30 seconds to mars	bs.serving-sys.com	1
30 seconds to mars	de.omg.yahoo.com	3
30 seconds to mars	de.wikipedia.org	193
30 seconds to mars	de.wikipedia.org	99

to reduce the url to the domain component including subdomain (e.g query ”miley cyrus” goes to ”de.omg.yahoo.com”). This makes the partitioning a domain clustering problem. This reduces dimensionality tremendously without introducing too naive assumptions (domains tend to serve one main intent).

In this setup, we can identify 260 million query-domain pairs. There are 6.29 million domains visible in the dataset but only 3 million which we have seen in more than 2 unique pairs.

We see 54.37 million unique queries. 20 million of these queries are leading to 2 or more domains. This is an important fact as this will provide the main connection between the domains.

We have decided to focus on complete and untransformed queries (no linguistic analysis, spell correction or other transformation incl. character decoding). The important consequence of this choice is that there is a large probability that if a given query led to two different domains, it was most likely because a user clicked of both URLs during a unique session with a clear albeit latent intent.

We can test the performance of our algorithms on roughly 250,000 new daily pairs as well as on a much bigger dataset of 10 billion query-url pairs and a crawler emitting billions of documents per week.

3 Construction of the similarity matrix

We start with $n =$ 6,290,890 domains. We then build a square symmetric matrix S of size (n, n) with each element i, j set to the sum of the minimum number of times we have seen a query answered by both domain i and domain j. For example, for the dataset in table[1], we would set:

$$Sraw_{i,j} = \min(82, 1682) + \min(3, 193)$$

where $i =$ de.omg.yahoo.com and $j =$ de.wikipedia.org

If $i = j$, we set the value to 0.

We then need to normalize this matrix. We first normalize by row:

$$Sn_{i,j} = Sraw_{i,j} / \sum_{j=1}^{n} Sraw_{i,j}$$

We then re-symmetrize the matrix by taking the minimum element of the matrix and its transpose:

$$S_{i,j} = \min(Sn_{i,j}, Sn_{j,i})$$

This normalization is extremely important for all further steps as it makes sure to get the right similarity order. Specifically we fix two problems: in the original and in the nomalized matrix, large popular domains like wikipedia are the closest neighbors of all other domains and in the normalized matrix small domains (with e.g. only a handful of co-occurences with one single larger domain) have a similarity of 1 with the larger domain. Only after the two normalization do we have the expected order of similarity for a given domain d:

$$\text{sim(small domain, } d) < \text{sim(d,large popular domain)} < \text{sim(}d\text{,very similar domain to } d)$$

We also create a matrix diagonal matrix D where $D_{i,i} = \sum_j S_{i,j}$

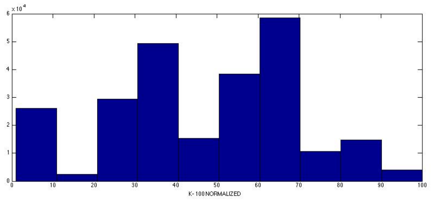
4 Spectral Clustering

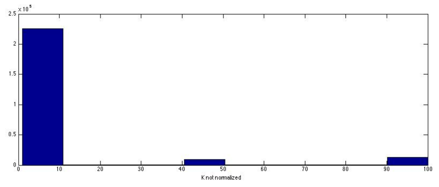
Because the matrix is relatively large, we remove the domains i for which have the $D_{i,i} < \epsilon$, this is warranted because these domains do not actually contribute to the clustering at all. Even when taking a very low ϵ (very close to zero), the dimensions get reduced to $n = 493,468$. Interestingly even at this level of dimensionality reduction we lose extremely little information. In fact we still have 97.33% of all unique queries we have seen. The domains that are dropped in this reduction process can either be easily re-clustered later (e.g. hundreds of thousands of these are subdomains that are easily clustered to wherever their top-domain end up), clustered in the cluster where the query they were associated with points to or simply left as singleton clustered together in a tail index.

In order to do spectral clustering, we constructed two different graph Laplacian matrices: unnormalized $L = D - S$ and normalized $L = I - D^{-1/2}SD^{-1/2}$. We then computed the k eigenvectors corresponding to the k smallest eigenvalues of these Laplacian matrices and then apply k-means to the resulting matrix of vectors (or its normalized version)

5 Results of Spectral Clustering

We tried different implementations of spectral clustering and suffered from two issues: (1) Spectral clustering is computationally very expensive at this size and (2) any level of k that was computable (e.g. 1000) did not deliver usable clusters. Depending on whether we used normalized or unnormalized Laplacian we either ended up with clusters where there was one dominant cluster (so dominant that all domains with any information would cluster together) – see right picture with k=100 non-normalized – or with a well distributed clustering (in terms of number of domains they contained) but each cluster containing a varied mix of topics and site types (see left- k=100 normalized). Both clusterings were measured to be very far from random (measured by number of cross-links between clusters) but unsuitable for our purpose.





The problem seems to require a much larger amount of clusters to be usable and while spectral clustering is theoretically possible at high levels of k, it is by no means trivial to implement.

6 Naive "greedy" algorithm

Because the problem seems to require a much larger number of clusters to offer a reasonable clustering and would benefit from a more hierarchical clustering, we propose a naive and greedy algorithm to cluster the dataset. In each iteration the algorithm goes through all the clusters (we start with each domain being a singleton cluster) and cluster each domain with its nearest neighbor as long as the similarity is above a low threshold (in this case 0.005). The algorithm is very greedy in that if A is B's nearest neighbor and B's nearest neighbor is C, we collapse A, B and C into a cluster. We iterate until the algorithm stops clustering neighbors because there are no similarities left above

Table 2: Naive and Greedy Clustering Iterations

Iteration	**Number of clusters created**	**information in clusters**
1	94361	0.8730
2	16204	0.8397
3	2859	0.6298
4	392	0.5689
5	36	0.5689
6	3	0.5689

threshold. In the second and later iterations, we compute the average similarities of the cluster members and domains outside of the cluster and combine with the cluster of the closest nearest neighbor (on average) outside of the cluster. The algorithm is therefore very greedy (it will cluster with anything above threshold) and very naive (it will cluster with another cluster even if only one member of the other cluster is the closest on average)

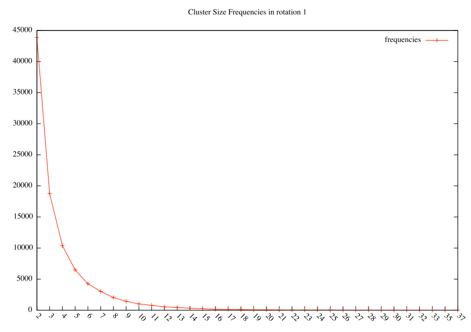
7 Results of Naive ”greedy” algorithm

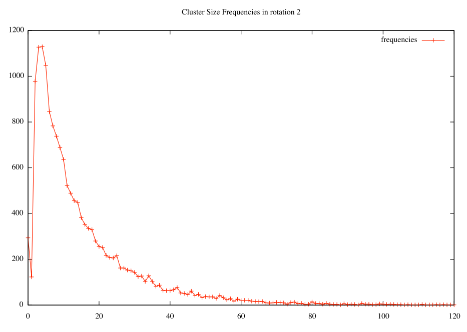
The algorithm converges in 7 iterations. Because it is too greedy the best and most useful clusterings happen at iteration 1 and 2. In later iterations, the clustering starts to lose its usefulness (much as the spectral clustering did). In table [2], we present the number of cluster at each iteration along with the percentage of information preserved in these clusters (ie. the percentages of the original queries that these clusters contain fully).

It must be noted that most of the useful clusters do not actually cluster after step 2 (causing the percentage of information to drop sharply after step 2). In steps 4-7, the clusters simply combine with one another.

If we assume that the clustering reached in step 2 is optimal (it is not formally but is not very far from it), the optimal k required to reach this in spectral clustering would be somewhere around 150,000 (because of the singleton clusters dropped in step1 and clusters that were perfectly clustered – in the sense that they had no nearest neighbors above threshold – in step 1 and did not ”survive” to step 2).

This algorithm generates clusters with very healthy sizes. Most clusters include 2-4 domains in step 1 (and none above 37) and most clusters are at sizes between 10 and 20 in step 2.





In a more visual sense, the clustering of ”spiegel.de” (one of the top news sites in Germany) results in the following 2 first steps (cleaning up the www...de):
step 1: bild, focus, spiegel, welt, stern, sueddeutsche
step 2: bild, focus, spiegel, welt, stern, sueddeutsche, faz, zeit, fr-online, handelsblatt, tagesspiegel, n-tv, tagesschau, n24

This is basically exactly how a German person would have clustered national news sites.

More generally, table[3] shows two (more obscure) step1 clusters that combined into a larger step 2 cluster and clearly represent the harry potter sites relevant to the German Internet users. At step 3, the algorithm tends to become a group of fiction sites.

Table 3: two clusters at step 1 combining in step 2

cluster step 1	**cluster step 1**
zauberhogwarts-spy-loesungen.30540.n6.nabble.com	harry-potter.1001spiele.de
zauberhogwarts-spy.dreipage2.de	www.harrypotterspiele.com
loesungen-zauberhogwarts.jimdo.com	www.verzaubert.at
zauberhogwarts-spy.weebly.com	www.carlsen-harrypotter.de
harrypotter.warnerbros.com	www.harrypottershop.de
www.hp-fans.de	harrypotter-xperts.de
www.harry-potters-fanpage.dreipage2.de	www.harrypotter-xperts.de
www.zauberhogwarts.de	www.fanficparadies.de
www.hp-fc.de	www.verzaubert.at
www.jkrowling.com	harrypotter.warnerbros.de
www.pottermore.com	www.verzaubert.at
zauberhogwarts.de	

8 Practical Use of Results and further work

We see highly usable results in this work. It is definitely possible to relax the greediness and naiveness of the algorithm mostly by changing the rules and threshold at which clusters combine in step 2 and later. This would slow down the convergence (but not by much, perhaps 10 steps instead of 7) and generate more useful larger clusters.

This technique also allow us to index the cluster number in each step with each document and use it in the sharding process (as intended primarily).

These clusters would also be very good at labeling queries (because of the results we have, we can easily construct query to cluster id training sets) for further supervised intent classification (using the cluster ids as latent intent indicator).

Moreover, it is almost trivial to construct extremely efficient filters (for example for adult content) by creating groups of level 1 clusters as filter set (based on a training set). Additionally, creating user profiles as weighted set of cluster ids based on previous interactions seems promising. Also, personal page rank types calculation (where the cluster members are the restart set) could be used in combination with query classification to provide boosting factors in the ranking functions.

9 Conclusion

We covered all the requirement stated in the introduction. Given a document, we can easily identify which partitions it belongs to simply by its domain. Supervised machine learning techniques can be used to learn to classify queries into labels given by the cluster ids. We constructed the similarity matrix to be heavily influenced by co-occurences on a result page. We did end up with a large and useful number of clusters which are immediately understandable when observed. The reason the naive approach works so well is that the size of the data and the way it is collected (millions of page expressing their intent) in fact does most of the clustering upfront and the naive algorithm has an opportunity to meaingfully cluster before the greediness takes over.

References

[1] *Parallel Spectral Clustering in Distributed Systems.* Wen- Yen Chen, Yangqiu Song, Hongjie Bai, Chih-Jen Lin, E.Y. Chang, IEEE Trans. PAMI, 33(3), 2011.

[2] *R. Liu and H. Zhang*. Segmentation of 3D meshes through spectral clustering. *In Proceedings of Pacific Conference on Computer Graphics and Applications*, 2004.

[3] U. Luxburg. *A tutorial on spectral clustering. Statistics and Computing*, 17(4):395-416, 2007.