Computer Sciences Department

Badger: An Entropy-Based Web Search Clustering System with Randomization and Voting

Lidan Wang
Chloe Whyte Schulze

Technical Report #1537

October 2005

UNIVERSITY OF
WISCONSIN
MADISON

Badger: An Entropy-Based Web Search Clustering System with Randomization and Voting

Lidan Wang, Chloe Whyte Schulze
{*lidan, cschulze*}*@cs.wisc.edu*

Computer Sciences Department
University of Wisconsin, Madison
May 6th, 2005

Abstract

We have implemented and improved an entropy-based clustering algorithm. In addition to utilizing entropy as a clustering mechanism, our algorithm, Badger, uses randomization and a voting scheme to improve the quality of the resulting clusters. Using parsed web search result snippets, we have tested our algorithm and compared it against EigenCluster, a clustering meta-search engine developed by a research group at MIT. Our algorithm performs comparably to EigenCluster, but with slightly more overhead due to the extra work of the randomization step. We have found entropy to be a valid and interesting measure of document similarity and additionally we find it produces cohesive clusters.

1 Introduction

The technique of clustering has been widely used in many areas of artificial intelligence and information retrieval, however, it has only been in the last several years that it has begun to spread to web applications. Much research has been devoted to applying clustering techniques to organize web search results. The motivation behind this research is the desire to move away from the "top ten" syndrome. It has been conjectured that users rarely go beyond the first page of ranked search results, posing a problem for lesser known pages and users alike. For users who are looking for information on a broad topic or perhaps are not exactly sure what they are searching for, the top ten results will not necessarily be very helpful. The proposed answer to this problem is to present the user with clusters of returned web pages that represent different sub-categories of the queried topic. In this way, the users will be able to look at a more thorough snapshot of the returned data and hopefully this will help them pinpoint what they are looking for in a more efficient and timely manner.

The key to this solution is the quality of the clustering algorithm. Intuitively, the better the clustering, the more helpful the results will be to the user. The question is then, which clustering algorithm produces the "best" final clustering? According to research, so far, it seems a clear winner has yet to be found[7]. Each algorithm has its advantages and disadvantages, which put most of the implementations on the same level. With this in mind, we sought to develop our own algorithm that combined several different techniques and see how it compared with an existing web search clustering application.

We have implemented an entropy-based clustering algorithm that specifically applies to clustering documents and web search results. Our implementation uses the techniques of randomization and voting to improve the final clustering result. This combined technique, which has roots in machine learning, has yet to be used in other entropy-based algorithms. When comparing output, we found that our algorithm performs on par with a current

implementation from MIT called EigenCluster [3]. Additionally, we conclude that there are steps that still need to be explored in order to fully address the problem of web search result clustering.

The paper will continue as follows: in section 2 we discuss related work, section 3 describes the concept of entropy and how we utilize it. The details of our algorithm are discussed in section 4 and performance results are given in section 5. Finally we conclude in section 6.

2 Related Work

Cheng et al from MIT developed a divide and merge clustering algorithm, which they then used to drive a meta-search engine called EigenCluster [3]. The search engine takes results from other search engines such as Google, parses the results, and then returns clusters of the web page snippets under topic headings. Their aim was to combine a top-down or divide approach with a bottom-up or merge technique to create a hierarchical tree whose leaf nodes are clusters. They show that their algorithm performs competitively with other clustering methods.

Grouper, an interface to a web search engine developed at the University of Washington, is another application which seeks to improve usability of search engines by clustering results [8]. In their study, they compared the differences of user actions when faced with a ranked list versus a set of clusters. Though the authors mention that they need to perform a more in-depth user study, their initial results show that a higher percent of users click on more links with a clustered search result representation. Unfortunately, their site has been retired and so we were unable to personally assess the quality of their engine.

An interesting approach to using clustering for document browsing comes from Cutting et al in their Scatter/Gather application [4]. They propose using clustering as a way to retrieve documents, instead of using the technique as a post filter on the search results. In this way a user can incrementally search for results without being overly specific in his/her query. Though certainly an intriguing concept, it would be difficult to apply it to the vastness of the web search space.

There are currently several other clustering search engines on the web. Clusty by Vivismo [9] and SRC beta by MSRA [10] are two commercially available implementations. From casual use, they both appear to be passable implementations of a clustering search engine. Just from eying the clusters, however, it is clear that for a clustering engine to really become popular more accurate clusters need to be produced.

3 Entropy

Entropy is the measure of information and uncertainty of a random variable [6]. Let X be a random variable and $S(X)$ be the set of values X can take, and $p(x)$ be the probability function of X. The entropy of X is $E(X)$:

$$E(X) = -\sum_{x \in S(X)} p(x)\log(p(x))$$

A cluster contains a set of documents, if we assume independence of the attributes in each document, the entropy for a cluster can be computed by:

$$E(X) = E(X_1) + E(X_2) + \ldots + E(X_n)$$

We denote the goodness of clusters by a term called system entropy, which is a weighted sum of clustered entropies. The equation of system entropy is:

$$E(C) = \sum_k \frac{|P(C_k)|}{D}(E(P(C_k)))$$

4 Badger Vs. CoolCat

The clustering algorithm we implemented takes much of its influences from the CoolCat algorithm described in the paper by Barbará et al[1]. Their aim was to connect the clustering of categorical data with the well-known concept of entropy. In doing this, they showed that entropy is a comparable similarity measurement as to those used in other clustering algorithms[2]. However, we depart from CoolCat's implementation in two distinct ways. First, the application domain of Badger has been extended to clustering web search results. Secondly, we introduce randomization and voting to reduce the negative effect of order imposed on incremental document processing.

We found entropy to be an intriguing measurement because of its connotation of order, and additionally because

it has been used as a way of measuring the "goodness" of a final clustering in other clustering algorithms[7]. Below we will describe our algorithm and then discuss the deviations we have made from the CoolCat implementation.

4.1 Badger

4.1.1 Initialization Step

To begin the clustering process we must first create K initial clusters of one document each. To do this, we first randomly choose a sample set of N data points where N is a subset of all documents being clustered. The goal of the initialization step is to choose the K most dissimilar points, with which we will initialize the clusters.

After we have chosen our sample, we then compare each of the N points to every other point and create a matrix of the pairwise entropies. To find the first two points, (p1 ,p2), we select an entropy from the matrix, which maximizes the minimum pairwise entropy. We do this because we then get two points with the relative maximum difference. Once these two points are obtained they are each put into a separate cluster, (C1, C2). The process then continues: to create the **j**-th cluster we choose a point pj that maximizes the minimum entropy between the already chosen points.

Once K clusters have been created, the remaining points in N are added back to the total list of documents so that they can be clustered in the incremental step. The initialization step takes $O(N^2)$. It is important to note that the number of points chosen for N is a significant parameter in the algorithm. The number of documents chosen for N must be large enough that there is high probability that at least one member of each category in the dataset is chosen. Intuitively, this ensures the variety of the initial clusters and a final clustering of better quality. Of course, a balance must be made between choosing N to be big enough, yet not so big that it slows down the algorithm significantly.

4.1.2 Incremental Step

Once the initial clusters have been created, the remaining points are added to clusters incrementally. A document is added to a cluster such that the minimum system entropy is maintained (see Figure 0). This is done until all points are added.

```
do for each remaining document dn {
   do for each cluster Ci {
      place dn into Ci
      calculate the system entropy
   }
   place dn into Cj, where placing
   dn in Cj results in the minimum
   system entropy
}
```

Figure 0. Pseudo code of Incremental Step

Clearly the ordering of the documents impacts the resulting clustering. A document clustered early on may later not fit in its cluster, and if it had been clustered later in the set, it may have been put in a different cluster. For these reasons we have departed from using this incremental step by itself and additionally use a function described below.

4.1.3 Incremental Step with Randomization and Voting (Badger)

This method uses the above incremental step, but also introduces randomization and voting to result in a less order-biased clustering. Randomization is used in choosing the documents to put into clusters so that the order of the dataset has no impact on the resulting clusters. Voting then determines the most common clustering situation for each document. The algorithm is bootstrapped using the initialization step as before. Then the incremental step is used, however, instead of clustering the documents in order, each point is randomly chosen from the document list. This step is performed R times, and at each iteration we keep track of the cluster that each data point was placed in. The final clustering is determined by the total votes taken: each data point is put in the cluster where they had most often been placed.

In this more thorough version of the incremental step, the final clustering benefits from many earlier clustering results. Voting and randomization ensures that a document that may have been placed in an inappropriate cluster in an earlier iteration has a chance to be moved to a

better cluster. The parameter R is another important variable in the algorithm. Intuitively, we would like to choose R to be as big as possible, because that would introduce the maximum amount of randomness into the clustering. Of course making R too large would radically slow down the performance of the algorithm. As is common in the database field, the trick is to balance the performance trade-off with the quality gain. At the moment we have chosen R to be twenty.

4.2 New Features Introduced in Badger

CoolCat differs in several points from our implementation. Our technique of randomization and voting is not used by CoolCat. The authors of course noted the impact of clustering the documents in order and their solution was to pick a percentage of points that seemed poorly suited to their cluster and reclustered them. This solution, however, still suffers from the original ordering problem. We further discuss this issue in the performance section.

An additional difference between our implementation and CoolCat is the change in usage domain. CoolCat was not meant to cluster web search results. Since part of our project's aim is to do just that, we have applied our algorithm to web search data and it has generally performed well.

5 Performance

In this section we are interested in identifying how sensitive the resulting clustering is to varying each of the three parameters in our Badger algorithm. We also want to compare the performance of Badger versus EigenCluster, and the performance of Badger versus CoolCat (incremental). Lastly, we want to see how Badger works on different datasets.

5.1 Varying The Number of Clustering Systems

5.1.1 Identifying The Problem

One weak point in the original CoolCat algorithm is that the order of processing points has a definite impact on the quality of the final clusters. It is quite possible that a point added to a cluster at a given time may become unfit for the same cluster at a later time as more points are clustered. To address this issue, the CoolCat algorithm selects a fraction of the points which can be considered the worst fit for the clusters they reside in, and re-cluster these points by using the same incremental method it uses originally to cluster them. However, it is obvious that the same problem of incremental clustering will persist for these selected points too as they are being clustered; i.e., the order of processing these selected points again matters, thus it again has an effect on the quality of final clustering. As a result, there is still no guarantee that the re-processing scheme used by CoolCat will improve the quality of the clustering.

5.1.2 Our Solution

To solve this problem, we use a well known method in machine learning and expert systems — ensemble, a.k.a. expert voting. We build a set of clustering systems, where each clustering system is constructed from a different ordering of processing the documents. We then let them vote which cluster a given document should belong to. The parameter R in the Badger's algorithm denotes the number of such clustering systems we build. Intuitively, the more clustering systems, i.e. experts, the better the voting results or the quality of final clusters we should get. The following theorem in expert systems serves as a justification for using this method:

> Theorem: Let Ri(x) be the region in feature space x that classifier i classifies correctly. Then, the region of correct classification of the mixture of expert classifier R(x) belongs to union(allRi(x)).

5.1.3 Results Of Varying R

Here we ran a set of experiments with various values for R (while other parameter values are fixed throughout) to see figuratively why R is an important parameter and how it impacts the quality of clusters. The dataset is constructed from Google search results for the query *"door"*. Each search result returned by Google is a document, and the Badger algorithm clusters these documents by minimizing cluster entropies and using the ensemble scheme. Two other parameters are kept constant; K (the number of

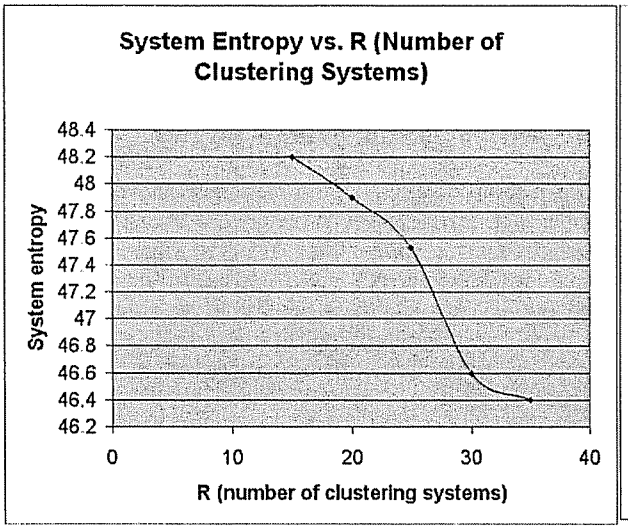

Figure 1: **System Entropy with Varying Values of R.**

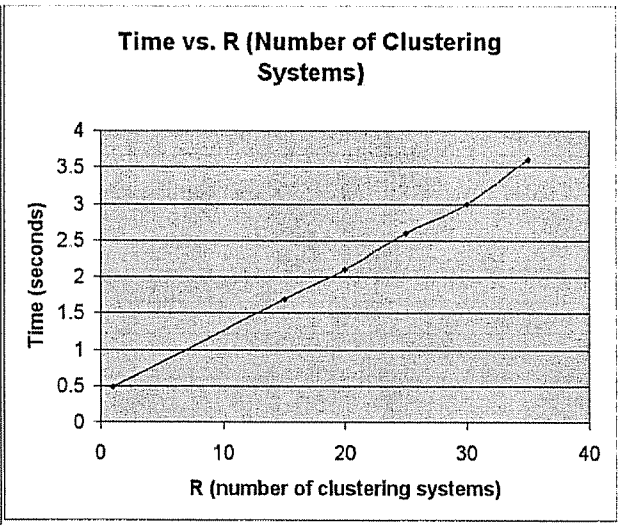

Figure 2: **Measurement of Time with Increasing Values of R.**

clusters) is 10, and N (the sample size used to initialize clusters) is half of total number of documents.

The system entropy is the weighted sum of cluster entropies in the system, and it is used as a measure of "goodness" of clustering. As we see from Figure 1, as the number of clustering systems increases, we generally get better clusters, as evidenced by the reduced system entropy. This relates to the theorem mentioned earlier, simply, more experts are better than few experts.

As Figure 2 clearly displays, there is a trade-off between runtime and the quality of clustering. We may be forced to compromise on the value for R, which results in a good (not best) clustering, but only takes a moderate amount of time.

5.2 Comparing Badger and EigenCluster

EigenCluster is another interesting clustering based web search engine. We are interested in comparing the Badger algorithm with EigenCluster in terms of time used to return results for a given query and the quality of final clusters.

The dataset used here is comprised of Google snippets returned by the query *"Wisconsin database group"*. Forty-two results are returned by this query. EigenCluster forms two main clusters for this query. The key words for Cluster 1 by EigenCluster are the database group at University of Wisconsin-Madison and other schools or departments that have associations it. The key words for Cluster 2 by EigenCluster are the research projects associated with UW-Madisons database group. Most of these results returned by Badger overlapped with results by EigenCluster in each cluster, and Badger returned a similar number of documents in each cluster as Eigen, with roughly the same overall meaning as the Eigen in each cluster. The results are displayed in Table 1.

	EginCluster	Badger
Cluster 1	18 documents	11 documents
Cluster 2	24 documents	31 documents
Time (sec)	0.04	0.39
System Entropy	30.71	33.3

Table 1: EigenCluster Vs. Badger. Dataset: query "Wisconsin database group"

EigenCluster is a good clustering algorithm because it combines hierarchical clustering with flat clustering. As a result, it consists of a top-down and bottom-up phase which results in better clustering quality, but similarly to Badger, at the cost of increased time complexity. We

	R=1	R=15	R=20	R=25	R=30	R=35
Sys. Ent.	72.7	48.2	47.9	47.53	46.6	46.4
Time (sec)	0.48	1.7	2.1	2.6	3.0	3.6

Table 2: Varying R Values (averaged over five runs)

	Badger	CoolCat
Mis-classification	1 document	5 documents
System Entropy	34.2	43.12

Table 3: Badger vs. CoolCat. Mis-Classification and System Entropy

chose the number of clustering systems to be ten in Badger, and as a result, the amount of time for Badger to return results is higher than the time taken by EigenCluster.

5.3 Comparing Badger with CoolCat

5.3.1 Dataset *"Door"* (as described in section 5.1.3)

In the CoolCat algorithm, R=1, i.e. only one clustering system is built. From Table 2 we see CoolCat system entropy is 72.2, which is much higher than the Badger algorithms entropies, which range from 46.4 to 48.2 depending on the R value.

Though we are improving the quality of clustering, it is at the cost of increased running time. This fact is again illustrated in Figure 2.

5.3.2 A Different Dataset

We manually constructed a dataset *"eclipse"*, which combines the query results returned by two queries *"eclipse IDE"* and *"eclipse lunar"*. Our goal is to see whether Badger is able to identify each member in each cluster correctly, and if there are any mis-classifications. We can compute mis-classifications by Badger because we know beforehand what the membership for each document should be.

From Table 3 we see that Badger outperforms CoolCat on this dataset. Badger only mis-classifies 1 document while CoolCat mis-classifies 5, and additionally as a result, CoolCat has a higher system entropy. This is due to the introduction of randomization and voting in the Badger algorithm.

5.4 Varying Initial Sample Size

Given a set of data, we want to determine the size of the sample such that at least one member of each cluster exists in the sample, so that they will serve as good starting points for the incremental step. A subset of the sample is used to initialize the clusters, and the subset points should be the K most dissimilar points in the sample, in order to maximize the distance/dissimilarity clusters (K is the number of clusters). One observation we have made is that the sample size, N, is dependent on the number of clusters, K. The more clusters there are, the larger sample size should it be in order to guarantee with high probability that at least one member of each cluster exists in the sample.

We ran several experiments to verify our observation. We are interested to see how various sample sizes, when other parameters are fixed (R=20, K=4), would affect the clustering outcome in terms of the resulting entropy for each cluster. The dataset we used is the Google snippets returned by query *"door"*, each snippet is considered an individual document, and our Badger algorithm clusters these documents with the expert voting scheme.

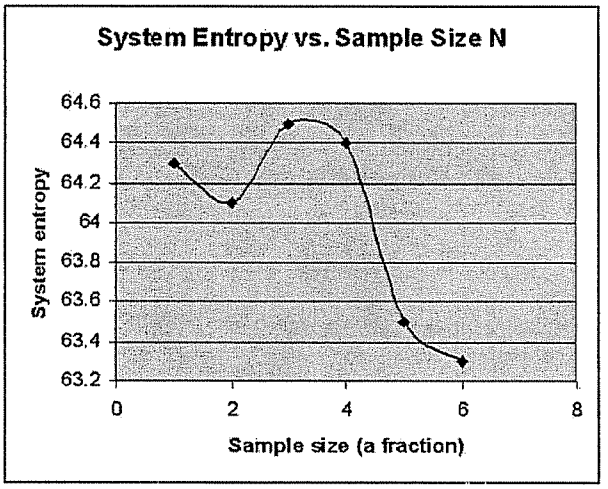

Figure 3: **System Entropy with Varying Values of N.**

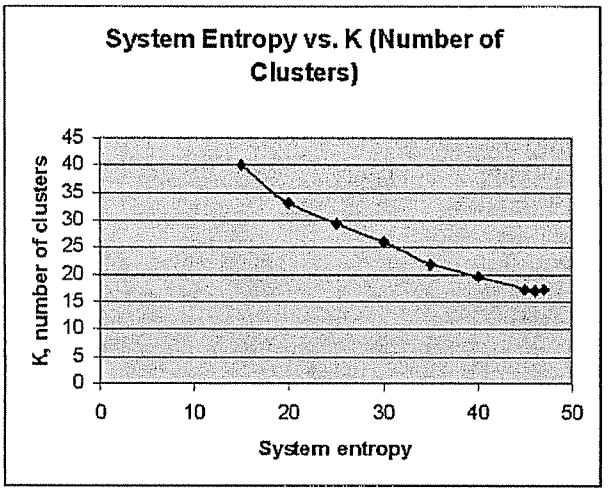

Figure 4: **System Entropy with Varying Values of K.**

From Figure 3, we see that as the sample size increases, we generally get better clusters, as evidenced by the de- crease in system entropy, which is the quantity we are try- ing to minimize. So this has confirmed our earlier conjec- ture that sample size correlates positively with the quality of final clusters.

5.5 Varying The Number of Clusters

K is the number of clusters predefined by the user. Each dataset naturally always has a K value that represents the real number of clusters that should be formed in the dataset, but we do not know what the K is for any given dataset. Here, we experiment with various K values and see how they affect the outcome of clustering. We keep all other parameters fixed. The dataset is *"door"* as described earlier in section 5.1.3, which is a very generic word with multiple meanings, and thus should result in a large number of clusters, with each cluster representing one meaning of *"door"*.

From the entropy curve in Figure 4, we can make a good guess on what the correct K value should be for the query results of *"door"*. We observe that when K=46, the system entropy reaches its minimum, and when K=47, the system entropy starts to increase again. Thus the optimal K value is 46. Examining the output snippets from Google query more carefully, we find that roughly K=46 is a good number of clusters. For example, there is a door magazine, the Double Door music hall, the girl next door, door and hardware institute, etc.

We further examined the output of the query "door" when run on EigenCluster, and found that it roughly returns around 50 clusters as well.

6 Conclusion

Badger is a clustering algorithm that utilizes entropy as a measure of document similarity and entire system ordering. Our implementation involves an initialization step to bootstrap the clusters, and then an incremental step, which uses randomness and a voting methodology to create the final clustering.

Based on the experiments we performed we can conclude that Badger has a better clustering qualify than CoolCat, but with a slight increase in time complexity. We have also seen how various parameters in Badger would affect the outcome of the clustering. Furthermore, upon comparing Badger and EigenCluster, we see that there is consistency between results returned by them, and Badger works well on finding right membership for documents as illustrated by the *"eclipse"* dataset example in section 5.3.2.

We conclude that further research needs to be done to create really meaningful clusters. The current problem with clustering web search results is it is difficult to produce clusters that encapsulate the central topic of several web pages. Many times an implementation will either create clusters that do not group documents correctly, or create topic groups that are incoherent. We have concluded that document clustering needs to move beyond the bag of words model to embrace a more complete understanding of the content of documents.

References

[1] D. Barbará, J. Couto , Y. Li. CoolCat: an entropy-based algorithm for categorical clustering. In *CIKM '02*, November 4-9, 2002, McLean, VA, USA.

[2] D. Barbará, J. Couto , Y. Li. CoolCat: an entropy-based algorithm for categorical clustering. draft, 2001

[3] D. Cheng, R. Kannan, S. Vempala, G. Wang. A Divide-and-Merge Methodology for Clustering. 2004

[4] D. Cutting, D. Karger, J. Pedersen, J. Tukey. Scatter/Gather: A Cluster-based Approach to Browsing Large Document Collections. In *15th Ann Int'l SIGIR'92/Denmark-6/92* 1992.

[5] B. Larson, C. Aone. Fast and Effective Text Mining Using Linear-time Document Clustering. 1999

[6] C.E. Shannon A Mathematical Theory of Communication. In *Bell System Technical Journal*, pages 379-423, 1948. Technical Report 00-034 1999

[7] M. Steinback, G. Karypis, V. Kumar. A comparison of Document Clustering Techniques. Technical Report 00-034 1999

[8] O. Zamir, O. Etzioni. Grouper: A Dynamic Clustering Interface to Web Search Results. http://www8.org/w8-papers/3a-search-query/dynamic/dynamic.html 1999

[9] Vivisimo. Clusty Search Engine. http://vivisimo.com/

[10] Microsoft Research Group Asia. MSRA SRC Toolbar 1.12. http://wsm.directtaps.net/default.aspx.

[11] Google. http://www.google.com.