AN EFFICIENT ALGORITHM FOR CLUSTERING SHORT SPOKEN UTTERANCES

Zhu Liu

AT&T Labs - Research
200 Laurel Ave South
Middletown, NJ 07748
zliu@research.att.com

ABSTRACT

Nowadays, spoken dialogue systems which provide automated customer service at call centers become more prevalent. It is time consuming to determine a set of call types for the dialogue system by analyzing a large volume of unstructured spoken utterances. Traditional hierarchical agglomerative clustering (HAC) algorithm can bootstrap the call types in an unsupervised way, yet the time and space complexities are huge, especially for large data set. Based on our observation that spoken utterances containing less than ten terms are common in the spoken dialogue system, we proposed an efficient HAC algorithm for short utterances. By utilizing the particular properties of short utterances, we significantly reduced both the time and the space complexities of the clustering algorithm.

1. INTRODUCTION

The use of spoken dialogue systems to automate customer services in call centers is continually expanding. In one such system, unconstrained speech recognition is used in a limited domain to direct call traffic in customer call centers (Gorin et al [1]). The challenge in this environment is not only the accuracy of the speech recognition but more importantly, how the spoken utterances are mapped to the right call types in the spoken dialogue system. For example, a reasonable call type of an utterance "I need to refill this prescription." in a pharmaceutical application is "Request(Refill)".

Determining the right set of call types based on a large volume of utterances demands knowledge and understanding of the business logic and operation as well as a time consuming procedure of analyzing all the unstructured utterances. Traditional hierarchical agglomerative clustering algorithm can bootstrap the call types in an unsupervised way, yet the time and space complexities are huge, especially for large data set. Fortunately, we can improve the traditional HAC, and make it more efficient for spoken dialogue system, where the utterances are normally short.

Many clustering algorithms can be found in an excellent reviewing paper by Jain et al [2]. Guha et al [3] introduced the HAC algorithm CURE (Clustering Using REpresentatives). CURE represents a cluster by a fixed number of points scattered around it, which makes the algorithm insensitive to the outliers and more efficient for large data set. Karypis et al [4] proposed the Chameleon algorithm, a hierarchical clustering using dynamic modeling. A key feature of Chameleon algorithm is that it accounts for both interconnectivity and closeness in identifying the most similar pair of clusters. Both CURE and Chameleon are noteworthy extensions for traditional HAC, but the clustering results are normally different from those of traditional HAC. The goal of our work is to achieve the same clustering result as the traditional HAC, yet more efficiently.

Clustering techniques have been widely applied in categorizing web pages, emails, news, etc., for a long time. Franz et. al [5] investigated both unsupervised and supervised clustering in the context of NIST's Topic Detection and Tracking (TDT) project, which explores automatic techniques for locating topically related materials in newswire and broadcast news [6]. Different from these tasks, the utterances in our task typically contains less than ten terms, much smaller than web pages, emails, or newswire, which normally comprise hundreds of terms. Short documents may deteriorate the clustering performance since the extracted statistics are less reliable. Yet, they enable us to significantly increase the efficiency of the HAC algorithm.

The paper is organized as follows. The traditional HAC algorithm is briefly described in Section 2, and we propose an efficient HAC algorithm for short utterances in Section 3. Experimental results are shown and discussed in Section 4. Finally, in Sections 5, we draw out conclusions.

2. TRADITIONAL HAC ALGORITHM

In this section, we briefly describe the three issues involved in an HAC algorithm: feature extraction, utterance/cluster distance measurement, and the clustering procedure.

At the simplest level, individual words can be used as

features. Other methods for deriving features include using N-grams as features, weighting features based upon the number of times a word appears in an utterance or how unusual the word is in the corpus, and performing word stemming. In our case, we use features that are invariant to word position, word count, and word morphology, and we ignore noise words. Each utterance is converted into a feature vector, which is an array of term weights. To ease the subsequent processing, we normalize all vectors to unit norm.

The distance of two utterances is defined as the cosine distance between corresponding feature vectors. Assume x and y are two feature vectors, the distance d(x,y) between them is $d(x,y) = (x \cdot y)/(\|x\| \cdot \|y\|)$. Considering that all vectors are normalized, the utterance distance is simplified as $d(x,y) = x \cdot y$. The cluster distance is defined as the maximum distance between any pairs of two utterances, one from each cluster. The range of utterance distance is from 0 to 1, and the range of the cluster distance is the same.

The details of HAC algorithm can be found in [7]. The following is a brief description of the procedure. Initially, each utterance is a cluster on its own. Then, for each iteration, two clusters with a minimum distance value are merged. This procedure continues until the minimum cluster distance exceeds a preset threshold. The principle of HAC is straightforward, yet the computational complexity and memory requirements are huge, especially for large data sets. Assuming that there are N utterances, direct implementation of HAC requires $O(N^2)$ memory for storing the utterance distance matrix and cluster distance matrix, and the runtime complexity is $O(N^3)$.

3. IMPROVED HAC CLUSTERING ALGORITHM

We developed an efficient implementation of HAC by on-the-fly cluster/utterance distance computation and by keeping track of a list of neighboring clusters for each cluster, such that the memory usage is effectively reduced and the speed is significantly increased.

3.1. Observations for short utterances

Given that the average size of the utterances is small (10 terms) compared to the feature dimension (10,000 words), there is an efficient way to compute the distance between two utterances. The distance between two utterances can be computed by checking only the non zero terms of corresponding feature vectors. So instead of maintaining the huge utterance (resp. cluster) distance matrix, we compute the utterance (resp. cluster) distance on-the-fly, such that the memory usage is effectively reduced to O(N).

Another interesting phenomena is that when the utterances are short, most pairs of utterances share no common terms, and their distance is 1.0. Consequently, for each cluster, most of the distances from other clusters are 1.0. This means that for each cluster, we only need to track a few neighboring clusters. Instead of searching the nearest pair of clusters among all pairs of clusters ($O(N^2)$) at each iteration, we keep track of a list of neighboring clusters for each cluster, and search the nearest neighbor for each cluster. The searching complexity is reduced to O(N). The overhead is the maintenance of the neighboring cluster lists for all clusters. In later section, we will show that the maintenance for short utterances is not expensive.

3.2. Improved HAC clustering

Figure 1 shows the procedure of the proposed algorithm. For a set of N utterances $\{u_i, i = 1, ..., N\}$, we first compute their feature vectors, and normalize them. Two major steps followed are initialization and core clustering procedure. At the initialization step, each utterance u_i is assigned to a cluster C_i by itself. For each cluster C_i, the distances from the other clusters are computed and sorted to find a list of K nearest clusters $NC_i = \{NC_i^k, k = 1, ..., K\}$, and their distances $CD_i = \{CD_i^k, k = 1, ..., K\}$. Obviously, the nearest cluster to cluster C_i is cluster NC_i^1.

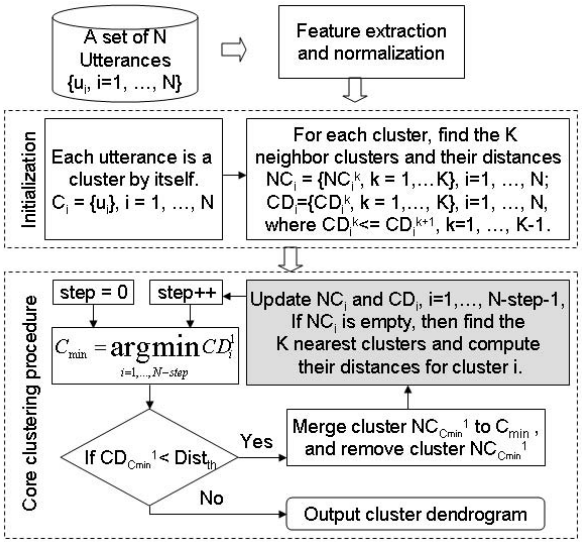

Fig. 1. Diagram of the improved HAC algorithm.

The core clustering procedure is an iterative procedure, where two closest clusters are merged at each iteration. If the distance is less than a preset threshold, we merge the two clusters, and update all neighboring cluster lists NC_i and their distances CD_i. Otherwise, the iteration terminates, and the clustering result is achieved. There are two situations while we update NC_i and CD_i for cluster C_i. If none of the merged clusters belong to the list, we don't need to do anything. For short utterances, this is true for most clus-

ters. If either one of the merged clusters was included in the neighboring cluster list, we need remove it, and compute the distance between C_i and the merged cluster. If the distance is smaller than the largest distance in CD_i, we insert the merged cluster in the right slot of the neighboring cluster list, otherwise, the merged cluster is not considered as a neighbor of cluster C_i. During the update procedure, the size of neighboring cluster list might decrease. But only when the list is empty, a new list of K nearest neighbors and their distances for the cluster will be regenerated. The end result of the algorithm is a tree of clusters called a dendrogram, which shows how the clusters are formed.

In Fig. 1, we highlight the block of neighboring cluster list update, which is most computation intensive. The complexity of list update for each cluster is determined by whether a new list of neighboring cluster need to be regenerated or not. When K is bigger, it is less probable to generate a new list of neighbors, but we need to maintain a longer list. When K is smaller, more likely, we need to regenerate a new list of neighbors, but we spend less time in maintaining a shorter list. There is a tradeoff to set the value of K. The real complexity the algorithm depends on the data, but the lower boundary is kN^2, which corresponds to the situation that there is no need to regenerate any neighboring cluster lists, but only updating the existing ones.

Depending on the distance threshold chosen in the clustering algorithm, the clustering results may either be conservative (with small threshold) or aggressive (with large threshold). In real applications, we tend to set a relatively low threshold to make sure each cluster is homogeneous. For short utterances, HAC may produce a large number of clusters since many utterances are totally different than the others. To reduce the number of tiny clusters, we merge all clusters smaller than an established minimum size into a special "other" cluster.

4. EXPERIMENTS

The data used in this paper is collected from AT&T VoiceTone spoken dialogue applications in four industrial domains, including financial, health care, insurance, and retail. Figure 2 shows the distribution of the feature vector length for a set of 4000 utterances in financial domain. 92% utterances are shorter than ten terms after feature extraction. Figure 3 shows the the effect of the size of neighboring cluster list on the time cost of the algorithm. The test is run on a data set with 10,000 utterances from financial domain, and the tested values of K are 4, 8, 16, ..., 512. The distance threshold is set to 0.99. Obviously, the curve shows the tradeoff of K in term of the computation complexity. When K is too small or too big, the complexity is high, and when K is in the middle (in this case, 128), the least complexity is achieved, which costs 123 seconds.

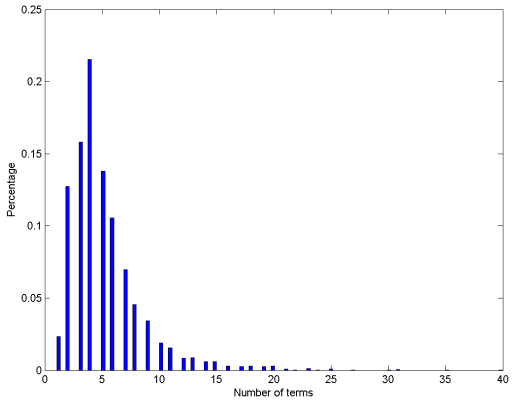

Fig. 2. Histogram of utterance vector length.

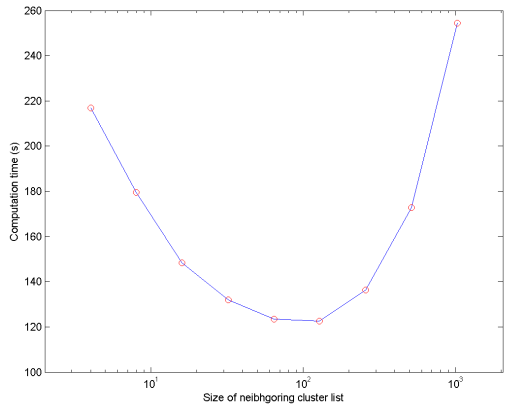

Fig. 3. Effect of the size of neighboring cluster list.

Table 1 shows the computation time and memory usage for traditional HAC and our improved algorithm. We compared them on two data sets: one contains 5,000 utterances, and the other 20,000 utterances, all from financial domain. The size of neighboring cluster list (K) is set to 128, and we set the distance threshold to 0.99. For the first data set, the direct HAC implementation requires 4 hours to complete and uses 200 MB memory, yet the improved implementation only takes 15 seconds and requires 8MB memory. For the second data set, we only provide the results for the improved implementation, and the memory usage for the direct implementation. We did not measure the computation time since it takes too long - a reasonable estimate is about 250 hours.

4.1. Clustering performance evaluation

We use the purity concept explained in [8] to evaluate clustering performance. The two measurements are the average

Implementation	Number of Utterances			
	5,000		20,000	
	Time	Memory (MB)	Time	Memory (MB)
Traditional HAC	4 Hour	200	N/A	3200
Improved HAC	15 s	8	540 s	30

Table 1. Clustering algorithm complexity.

cluster purity (ACP) and the average call type purity (ATP), as explained below. First, we define the following symbols. n_{ij}: total number of utterances in cluster i with call type j; NT: total number of call types; NC: total number of clusters; N: total number of utterances; $n_{.j}$: total number of utterances with call type j; $n_{i.}$: total number of utterances in cluster i. Then the purity of a cluster $p_{i.}$ and the the ACP can then be defined as:

$$p_{i.} = \sum_{j=1}^{N_T} n_{ij}^2 / n_{i.}^2, \qquad ACP = \frac{1}{N} \sum_{i=1}^{N_C} p_{i.} n_{i.}.$$

Similarly, the call type purity $p_{.j}$ and the ATP are calculated as:

$$p_{.j} = \sum_{i=1}^{N_C} n_{ij}^2 / n_{.j}^2, \qquad ATP = \frac{1}{N} \sum_{j=1}^{N_T} p_{.j} n_{.j}.$$

The ATP measures how well the utterances of one call type are limited to only one cluster, and the ACP measures how well the utterances in one cluster are within the same call type.

We evaluated the clustering performance for four industrial domains mentioned before. To cope with the multiple call types problem, we only consider the single call type utterances in the evaluation. Each data set consists 3000 utterances, and their call types are manually labeled. Table 2 shows the results. In the evaluation, we set the clustering distance threshold to 0.6, and the minimum cluster size to 5.

Data	N_C	N_T	ATP (%)	ACP (%)
Financial	36	335	23.2	71.2
Health care	92	133	45.6	61.3
Insurance	51	279	25.4	60.2
Retail	31	131	35.8	70.2

Table 2. Clustering performance for four applications.

Across the four applications, the ACP's are in the same range, roughly $60 - 70\%$. The ATP's are quite different among the different applications. Generally, when the number of call types is larger, the ACP will be smaller, and when the number of clusters is larger, the ATP will be smaller. For the health care and insurance applications, the numbers of call types are large, so their ACP's are small. For financial and insurance applications, the numbers of clusters are large, and their ATP's are small.

The call types for different applications are determined by analysts based on their knowledge and on the business problem to be solved. Therefore, the number of call types may not uniformly reflect the scattering in the data sets. The clusters are determined in a systematic way and they more reliably indicate the structure of the data sets. For example, the financial application has 36 call types but it does not mean that the data set is homogeneous. Actually it is not homogeneous since there are a large number of clusters.

5. CONCLUSIONS

In this paper, we proposed an improved clustering algorithm for short spoken utterances. The clustering results serve as a bootstrap for call type design in a spoken dialogue system. The time/space complexity of the proposed approach is significantly reduced, compared to traditional hierarchical agglomerative clustering, though it still achieved the same clustering outcome. The experimental result clearly demonstrate the promise of the improved algorithm.

6. REFERENCES

[1] A. L. Gorin, G. Riccardi, and J. H. Wright, "How may i help you?," *Speech Communication*, vol. 23, pp. 113–127, 1997.

[2] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Computing Surveys*, vol. 31, no. 3, pp. 264–323, 1999.

[3] S. Guha, R. Rastogi, and K. Shim, "CURE: An efficient clustering algorithm for large databases," in *SIGMOD*, Seattle, Washington, June 1998, pp. 73–84.

[4] G. Karypis, Eui-Hong Han, and V. Kumar, "CHAMELEON: Hierarchical clustering using dynamic modeling," *IEEE Computer*, vol. 32, no. 8, pp. 68–75, Aug. 1999.

[5] M. Franz, T. Ward, J. S. McCarley, and W. Zhu, "Unsupervised and supervided clustering for topic tracking," in *ACM SIGIR Conference on Research and Development in Information Retrieval*, New Orleans, Louisiana, 2001, pp. 310–317.

[6] C. L. Wayne, "Multilingual topic detection and tracking: Successful research enabled by corpora and evaluation," in *Language Resources and Evaluation Conference (LREC)*, 2000, pp. 1487–1494.

[7] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*, Prentice Hall, 1988.

[8] I. Lapidot J. Ajmera, H. Bourlard and I. McCowan, "Unknown-multiple speaker clustering using HMM," in *ICSLP*, Denver, Colorado, 2002, pp. 573–576.