Clustering for Data Reduction: A Divide and Conquer Approach

Nicholas O. Andrews and Edward A. Fox

Department of Computer Science, Virginia Tech,
Blacksburg, VA 24060

{nandrews, fox}@vt.edu

October 16, 2007

Abstract

We consider the problem of reducing a potentially very large dataset to a subset of representative prototypes. Rather than searching over the entire space of prototypes, we first roughly divide the data into balanced clusters using bisecting k-means and spectral cuts, and then find the prototypes for each cluster by affinity propagation. We apply our algorithm to text data, where we perform an order of magnitude faster than simply looking for prototypes on the entire dataset. Furthermore, our "divide and conquer" approach actually performs *more accurately* on datasets which are well bisected, as the greedy decisions of affinity propagation are confined to classes of already similar items.

1 Introduction.

We consider the problem of data reduction, whereby we want to reduce our original data to a smaller but representative subset. This is related to feature selection in dimensionality reduction tasks, where we are looking for an $m < n$, where $m << n$. A reduced dataset might have a direct interpretation. For instance, the objects in question might be sentences, and resulting prototypes would span only the most essential (non-redundant) information, resulting in an effective summary of the document. Data reduction is also a common method of dealing with intractability, whereby we are interested in finding a small subset, called a *coreset* in computational geometry [4], that well approximates the properties of the original data.

Such reduction can be useful on a small scale for real-time applications; for example, by reducing a large number of query results to a more manageable subset, it becomes easier to find the particular topic or concept we are interested in. Furthermore, operating on this reduced set allows for further postprocessing such as more effective visualization methods. Another interesting application is facilitating the analysis and storage of large volumes of sensor data (examples include the sensors used in the Large Hadron Collider, or LADAR scanners used in autonomous navigation and obstacle avoidance). Data reduction has also been used to ease storage costs for instance-based learning [9] and for model selection algorithms [14]. This latter application is a particular motivation for this work, as model selection (that is, finding the number of clusters from the data) is mainly concerned with the shape of the data, and therefore it is possible to operate effectively on a representative subset.

Clustering, the grouping of similar objects, can be applied to data reduction as long as we enforce that each cluster be described by a representative object rather than a constructed "mean." Algorithms of this type include those using the k-centers or k-medoid heuristics. Further background information will be provided in Section 2 concerning these algorithms. A recent algorithm of this type called affinity propagation (AP) has been proposed, which, while having quadratic runtime, is more accurate than k-centers and provides a convenient way of controlling the reduction ratio based on a parameter called a *preference* [5, 6]. Properties of this algorithm as it applies to the task of data reduction will be discussed in Section 3. Unfortunately, a quadratic runtime can be problematic when dealing very large datasets, as is typical in many data mining

tasks. We are interested therefore in finding a way of performing an accurate data reduction in linear time, hopefully not at the expense of too much loss of accuracy.

Our contribution is a "divide and conquer" approach to the problem. Our idea is that it is only similar objects which are grouped together by the k-medoid heuristic, and therefore an effective sampling scheme should only present to these algorithms groups of related objects. Fortunately, this is essentially what clustering does, in the sense of the k-means heuristic, and so we apply a balanced variant of the k-means algorithm as a divide phase, and then "conquer" the output clusters by affinity propagation. We present our algorithm in Section 4, and describe our experimental validation of its effectiveness in Section 5.

2 Background.

Exact clustering is an NP-hard problem. The most popular approximate solution is the well-known and studied Lloyd's or k-means algorithm [11], which can be derived as a special case of expectation maximization. Let $\phi(x_1, x_2)$ denote the similarity between two objects from a set $x_1, x_2 \in \mathcal{X}$. Then k-means optimizes the following objective function:

$$\sum_{i=0}^{k} \sum_{x \in \mathcal{X}} \phi(x, m_i)$$

where k is the desired number of the clusters. This objective function is usually expressed with ϕ as the squared Euclidian distance, however we allow here for more general functions such as cosine similarity (which will be used later). A simple way of adapting k-means to the task of data reduction is to enforce at each iteration (or after convergence) that the means be at actual points in the dataset. That is, each mean m_i is assigned to the point x it is is most similar to (its nearest neighbor), or

$$m_i = \max_{x \in \mathcal{X}} \phi(x, m_i).$$

A different formulation known as k-medoids[1] examines the *cost* of swapping, from an initial partition, the current prototype with another potential prototype. The Partitioning Around Medoids (PAM) [8] algorithm accomplishes its clustering objective by examining the cost of swapping the current prototype with *every* other potential prototype at each iteration. This formulation has the advantage of being tolerant to outliers. However, a runtime analysis reveals that, at each iteration, the number of operations to find the best swap is of order $O(k(n-k)^2)$. Therefore, a number of sampling strategies, such as CLARANS, have been developed to scale it to larger datasets [13].

A good data reduction is one in which the reduction ratio is small and where prototypes well-describe their clusters. We take reduction ratio to be the ratio of the number of prototypes to the number of objects in the original dataset. For instance, if a sample dataset has 4 classes of 125 objects each, the ideal reduction would consist of four prototypes, one for each class. However, this assumes that each class can be described by a single prototype. While four prototypes could be found, they might not be representative of *all* objects in their respective classes. Indeed, k-medoid algorithms lack the notion of a mean and instead make local, object-wise decisions, while the k-means objective function is better suited for making such class-level decisions. We have observed that a good number of prototypes is typically much higher than the number of classes.

Figure 1 illustrates this distinction between the different heuristics (on the four class dataset); similar results have been obtained on all datasets used in Section 5. Observe how at the true number of classes (k=4) there is a sharp jump in the accuracy for k-means, while the graph of the accuracy of k-medoids remains unperturbed. We use an intuitive external measure of accuracy which is the fraction of objects whose prototype is from their own class. Perfect accuracy is trivially reached as the number of prototypes approaches the number of objects in the set, which corresponds to a 0% reduction (where we would not

[1] A *medoid* is the most central object of a cluster; it has also been called an *exemplar* or, as will be used in this paper, a *prototype*.

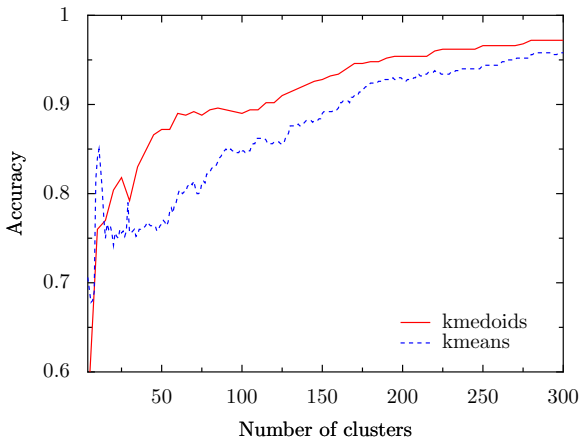

Figure 1: Accuracy as we adjust the number of clusters.

expect there to be any error). We evaluate k-means with the same metric, after assigning each mean to its nearest neighbor (i.e., discretizing the means to be at points in the dataset).

The different behavior of these algorithms can be attributed to the cost function of k-medoids, which only considers *one* other potential object at a time. On the other hand, k-means makes many swaps at each iteration and averages them in a constructed mean. As it is typically the case that a class does not contain a single centrally located object (particularly in high-dimensional data), k-medoid algorithms are less well suited for the purpose of grouping entire classes. In fact, k-means and k-medoids are quite different clustering problems, with distinct algorithms to solve them. This discussion now brings us to a different algorithm for the k-medoid problem, that has the interesting property of not requiring an explicit input for the number of prototypes saught.

3 Data reduction by affinity propagation.

Affinity propagation (AP) is a different formulation of the k-medoid objective of finding clusters that are well-described by a single prototype [6]. Rather than sequentially examining each potential prototype and calculating the cost of a swap, AP uses a message passing approach. As input, AP takes pairwise similarities (whose computation is inherently quadratric) and so called *preferences*, which are interpreted as how likely each object is of becoming a prototype. Henceforth, we will refer to the the application of k-medoid heuristic for finding prototypes, such as AP, as *prototyping*.

From this input, prototypes are identified according to two types of messages. The responsibility $r(x, m)$, sent from object $x \in \mathcal{X}$ to candidate prototype $m \in \mathcal{X}$, denotes how well-suited m is of being the prototype for x by considering all other potential prototypes m^* of x:

$$r(x, m) = \phi(x, m) - \max_{m^* \in \mathcal{X}, m^* \neq m} \{a(x, m^*) + \phi(x, m^*)\}.$$

The *availability* $a(x, m)$ of each object $x \in \mathcal{X}$ is initially set to zero. Availabilities, sent from candidate prototype m to object x, increase as evidence for m to serve as the medoid for x increases:

$$a(x, m) = \min\{0, r(m, m)\} + \sum_{x \in \mathcal{X}, x \in x, m} \max\{0, r(x, m)\}.$$

Figure 2 shows the accuracy of PAM and AP measured as in the previous section. We vary the number of clusters found with AP by adjusting the shared preference values in the range $[0, \max \phi(x_1, x_2)]$, with $x_1, x_2 \in \mathcal{X}$. This variable associated with each object determines the likelihood of it becoming a prototype.

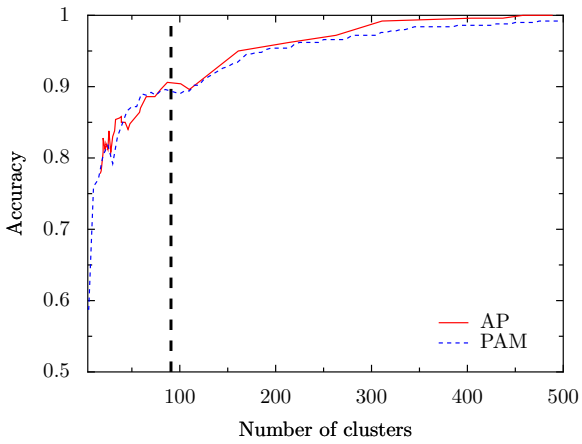

Figure 2: Accuracy as we adjust the number of the clusters.

As shown in the figure, the algorithms perform quite similarly, as indeed their objective is the same: to find clusters well described by a single prototype; of course, AP obtains such results quite faster than PAM. Frey and Dueck suggest using the median similarity for the value of shared preference, and we have observed that the accuracy at this value is usually at or near the elbow of the graph (the vertical line in Figure 2 is at this median similarity), suggesting that this is in fact a good choice.

This property of learning the number of prototypes is useful for the purpose of data reduction. Intuitively, a good choice for the number of prototypes will be at the true number of groups which can be described by a single prototype (reminiscent perhaps of the Shannon limit from information theory). That is, if the number of prototypes is below the number of groups which can be described by a single document, some objects will have to be grouped with prototypes that are not well suited to describe them; the further below this limit, the more documents will have to be grouped with less appropriate prototypes. To apply PAM for the same purpose, we would have had to guess this reduction ratio; in other words, we would have to guess the number of clusters/prototypes to supply as an argument to the algorithm.

Therefore, while AP achieves similar accuracy as PAM significantly faster, the computation of pairwise similarities as well the message passing procedure itself are still inherently of order $O(n^2)$. To be precise, the time complexity of the message passing procedure is in fact linear in the total number of similarities; similarities of 0 can be ignored without affecting the results. In the next section, we present a divide and conquer strategy for data reduction which helps scale this form of clustering to larger datasets.

4 A divide and conquer approach.

Our idea is similar in spirit to that of a recent "best of both" approach to improving k-means performance, where the authors combine batch k-means with a k-medoid style local search [7], and also to the sampling strategies for PAM mentioned in the previous section. We would like to scale k-medoid data reduction by retricting the problem space to groups of already similar objects output from a k-means clustering. If we measure accuracy as the percent of documents described by prototypes of their own class, then, ideally, restricting the input of the prototyping to pure classes will result in a pure reduction, with the added benefit of reducing the problem space significantly.

Unfortunately, in an unsupervised context, producing such an exact clustering cannot be accomplished in linear time. Furthermore, the actual number of classes is rarely known a priori. We sidestep both these problems by *overfitting* the model. That is, we supply k-means with a value of k which we guess to be greater than the true number of classes. While this is a guess, it is typically safer than when guessing the exact number of classes. The guiding factor in choosing this value of k is in fact the tradeoff between accuracy and compression. The further we increase k, the purer we would expect the clusters to become, and the

faster the prototyping phase afterwards. However, k cannot be *too* large, otherwise the search space for AP is overly restricted. This is illustrated in Figure 3: for a dataset of 500 documents, a good choice for k is at around 20 (or clusters of 30–50 documents). In fact, in our experiments, setting k to about 3% of the size of the collection has produced good results; in other words, the accuracy of the divide phase is quite insensitive to the paramater k.

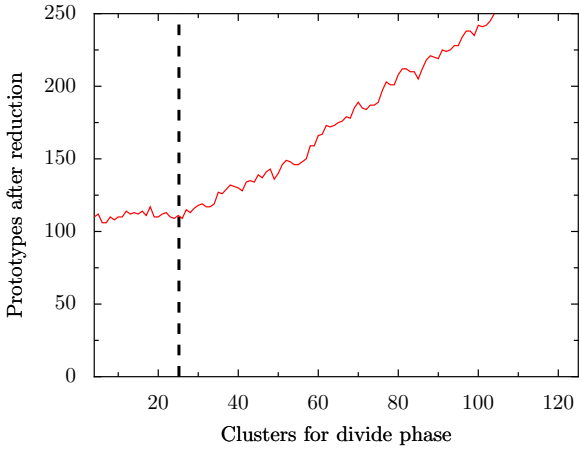

Figure 3: Reduction as we adjust the number of clusters for the divide phase. The dataset is of 500 documents.

The traditional k-means formulation provides no guarantees of producing balanced clusters. It is therefore quite possible that a single cluster contain a large portion of the entire dataset (particularly as k increases), limiting the usefulness of this approach for improving scalability. Enforcing the balancing constraint has been an area of some interest [16]. For this purpose, we opt to use a simple variant of k-means which repeatedly bisects the data until the desired value of k is reached. This formulation has the added benefit of being eminently parallelizable [10]. That is, rather than searching for k means at once, the data is divided into two clusters k times. By choosing the largest cluster as the next to bisect, the algorithm typically converges to a balanced clustering. The worst case is in fact the same as traditional k-means: for a dataset with n elements, bisecting $k$means can converge with a cluster of size $n - k$ (for k bisections) if at *each* bisection there is an empty cluster. However, in practice, the clusters are balanced enough for our purposes. If safer guarantees of balanced clusters are desired, a modification to the algorithm presented below is to continue bisections until the size of each cluster is below a certain threshold.

Suppose we supply k-means with k as a constant fraction of the input size and assuming a balanced clustering (for instance, $k = 0.03n$). As n (the size of the dataset) increases, the number of clusters increases linearly. Therefore, for the prototyping phase, we apply AP to a linearly increasing number of clusters of constant size. Theoretically, the combined algorithm therefore scales linearly with the size of the dataset. In practice, we expect performance to be slightly worse, as bisecting k-means does not provide strong guarantees of balance. In addition, AP sometimes does not converge and runs for more iterations than necessary, and the odds of this happening increase with the number of clusters. Please refer to Table 5 for actual runtimes.

Once the data is divided into relatively balanced groups, we apply AP to each cluster. On a heterogeneous collection, using the median similarity as the value of shared preference is often a good choice for the tradeoff between the quality of the reduction and the reduction ratio (as in the previous section). On the other hand, the clustering resulting from an overfitted k-means will hopefully consist of clusters of already similar items. Therefore, using the median similarity for an AP reduction will not result in a useful reduction ratio. Rather, since the clustering produces groups with reasonable guarantees of similarity, we want to describe these groups with as few prototypes as possible. Therefore, we take 0 as our value of shared preference. In other words, we initialize the message passing procedure with each item deemed "unlikely" to become a

prototype.

Algorithm 1: Divide & conquer

Data: The dataset $\mathcal{X}$
Result: A representative subset of $\mathcal{X}$
for $i \in \{1, 2, ..., k\}$ **do**
  set $m_1 = \min_{x \in \mathcal{X}} \phi(x, \frac{\sum_{x \in \mathcal{X}} x}{|\mathcal{X}|})$;
  set $m_2 = \min_{x \in \mathcal{X}} \phi(x, m_1)$;
  run $k$means($\mathcal{X}$, $\{m_1, m_2\}$);
  save cluster memberships in C_i;
  set $\mathcal{X}$ to the largest cluster;
end
for $i \in \{1, 2, ..., k\}$ **do**
  for $x_1, x_2 \in C_i$ **do**
    $s_{ij} = \phi(x_1, x_2)$;
  end
  run AP(s_{ik}) with preferences set to 0;
end

The initial assignment of the two means m_1, m_2 ensures that they are well-separated, and typically results in local optima of better quality than those produced from a random initialization. Furthermore, this initialization strategy is deterministic, and so we do not have to rerun our experiments multiple times.

4.1 Improving the divide phase by spectral cuts.

While k-means is efficient, it frequently converges to poor local maxima and is quite susceptible to the quality of the initialization. Furthermore, k-means has a tendency to get "stuck" on smaller datasets, a problem which is exacerbated in high-dimensional space [2]. Thus, while we expect the clusters to become purer as progressively more local decisions are made, this does not always occur in practice. Spectral cuts, which look at the clustering problem from a graph theoretic standpoint, have enjoyed considerable recent interest [15, 12]. One appeal of this clustering formulation is that no assumptions are made concerning the properties of the clusters; k-means, for instance, assumes that the clusters form disjoint convex sets, and is incapable of clustering datasets which are not linearly separable (e.g., two concentric rings in Euclidian space). In addition, since spectral clustering does not rely on complex initialization strategies, it produces consistent results across datasets.

All spectral clustering algorithms aim to find good cuts in the graph $G = (V, E)$, where the vertices V are the set of all objects to be clustered, and the edge set E defines the similarity between each vertex, also known as the weight between the vertices. While finding such cuts optimally is NP-complete, we will describe the common form of the approximate solution, and make specific choices based on the normalized cut approach of Shi and Malik [15]. We refer the reader to the aforementioned work for the motivations behind these choices, and for a runtime analysis.

First, we compute the weights of the edges of E based on a notion of similarity. The ϵ-neighborhood or the k-nearest neighbors can be used to produce a sparsely connected graph, however we will form the complete graph. These weights are encoded in the matrix W, where $w_{ij} = \phi(\mathbf{x}_i, \mathbf{x}_j)$. We then compose the Laplacian L of G, which is generally defined as $L = D - W$. D is the diagonal degree matrix. We instead use the normalized Laplacian, of which we find the second eigenvector (corresponding to the second smallest eigenvalue).

This eigenvector is known to have interesting properties; in particular, partitions of the eigenvector lead to good partitions of the original data. The remaining question is how to find such partitions. In the discrete case, the signs of the entries can be used to produce such a partition, where 0 would be the pivot. However, better results have been obtained in our continuous case by searching for the pivot which results in the partition of V that maximizes the normalized cut objective function of Shi and Malik, defined as

$$Ncut(A,B) = 2 - \frac{assoc(A,A)}{assoc(A,V)} + \frac{assoc(B,B)}{assoc(B,V)}$$

Here, $A, B \subset V$, and

$$assoc(A,B) = \sum_{a \in \mathbf{A}, b \in \mathbf{B}} w(a,b)$$

and $assoc(\{A,B\},V)$ are similarly defined. An incomplete search at regular intervals along the eigenvector has been found to work well, even when the intervals are quite large. It should be noted that other such criteria exist, such as the $MinMax$ cut, which work similarly well [3]. In addition, while we are interested in bisections, k-way clusterings can be achieved by clustering (using e.g., k-means) in the eigenspace spanned by the first k eigenvalues of the Laplacian.

Our new algorithm, which we will refer to as *spectral* divide & conquer, simply replaces the k-means bisections by spectral cuts. However, we show a hybrid algorithm below, which runs k-means until a certain threshold τ is reached, at which point the slower but more accurate spectral cuts are used. The idea is to exploit the efficiency of k-means for the initial splits, and then make more refined local decisions to find the leaves on which to search for prototypes.

Algorithm 2: Hybrid divide & conquer

Data: The dataset $\mathcal{X}$
Result: A representative subset of $\mathcal{X}$
for $i \in \{1, 2, ..., k\}$ **do**
  if $|\mathcal{X}| \leq \tau$ **then**
    compute the cut (A, B) of $\mathcal{X}$ optimizing $Ncut(A, B)$
  end
  else
    set $m_1 = \min_{x \in \mathcal{X}} \phi(x, \frac{\sum_{x \in \mathcal{X}} x}{|\mathcal{X}|})$;
    set $m_2 = \min_{x \in \mathcal{X}} \phi(x, m_1)$;
    run $k$means($\mathcal{X}$, $\{m_1, m_2\}$);
  end
  save memberships in C_i;
  set $\mathcal{X}$ to the largest cluster;
end
for $i \in \{1, 2, ..., k\}$ **do**
  for $x_1, x_2 \in C_i$ **do**
    $s_{ij} = \phi(x_1, x_2)$;
  end
  run AP(s_{ij}) with preferences set to 0;
end

5 Experiments.

We perform experiments on text data, where a clustering can be interpreted as a partition of the documents into topics. Under the standard vector space model, each document is represented as an m-dimensional vector, where each unique term in the corpus is a separate dimension and the values represent (weighted) term-frequencies. A corpus of n documents becomes a matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$. For each corpus, we preprocess the data by:

1. Stopword removal
2. Stemming

Table 1: Affinity propagation and our divide & conquer modification on a number of test collections.

		AP			**D&C AP**			**Spectral D&C AP**		
Corpus	Size	Time	Purity	Ratio	Time	Purity	Ratio	Time	Purity	Ratio
news	18828	-	-	-	**1072.60**	51.5	0.20	1642.31	**53.12**	0.20
sci	3949	527.63	95.95	0.214	**28.01**	80.70	0.19	165.69	**83.34**	0.19
sci/rec	7926	-	-	-	**99.90**	83.72	0.20	472.70	**85.19**	0.19
sci/comp/rec	12807	-	-	-	**340.24**	75.25	0.20	694.05	**76.29**	0.19
smart	3892	105.51	91.0	0.0761	**18.34**	**95.73**	0.20	39.19	**95.84**	0.19
sports	8580	-	-	-	**180.29**	80.1	0.22	808.05	73.14	0.22
hitech	2301	182.9	54.6	0.201	**24.67**	51.1	0.25	215.99	**52.15**	0.25
reviews	4069	308.2	71.2	0.200	**66.42**	65.8	0.23	715.71	**66.50**	0.23
la1	3204	350.0	64.2	0.184	**35.57**	60.4	0.24	433.70	59.96	0.24
la2	3075	92.5	63	0.187	**33.89**	59.3	0.23	292.24	**61.33**	0.22
tr11	414	1.2	55	0.0652	10.48	**57.73**	0.16	111.67	**65.94**	0.16
tr23	204	0.16	47.5	0.0294	11.67	**73.04**	0.21	82.33	72.06	0.20
tr41	878	6.8	49.2	0.103	11.85	**66.74**	0.21	160.38	**72.32**	0.21
tr45	690	3.2	46.4	0.0768	20.26	**65.22**	0.19	249.52	**66.96**	0.20
k1b	2340	188.25	90	0.272	**21.05**	74.53	0.20	241.51	**81.24**	0.20

3. Removal of very frequent and very infrequent words (0.5% and 99.5%)

4. *tfidf* weighing

5. L_2 normalization

As is typical with text data, the resulting term-document matrix is sparse and high-dimensional. We account for sparsity in our k-means implementation, resulting in bisections of order $O(nz \cdot k \cdot t)$, where nz is the number of non-zero values, k is the number of clusters, and t is the number of iterations until convergence. Since each document is normalized to unit-length (i.e. L_2 normalized), the clustering operates on a unit-hypersphere. The similarity ϕ between documents is therefore based on the cosine of the angle between them, computed as the vector dot product. The resulting algorithm is called *spherical* k-means for this reason.

We have used a set of standard test collections, varying in their balance (ratio of largest to smallest cluster) and number of classes, which have been fully described elsewhere and are common benchmarks of clustering performance on text [17]. In addition to those described in the cited work, we contruct some collections from the 20 newsgroups dataset[2]. The dataset used for illustrative purposes in the previous sections is a sampling of 125 documents from each of the four science classes from the 20 newsgroups dataset ("sci" in the table). We use the entire classes from the original dataset for all the constructed collections listed in Table 5. As a reminder, we define the purity of a reduction as the percent of documents grouped with prototypes of their own class, and the reduction ratio as $\frac{m}{n}$, where m is the number of prototypes after reduction and n is the original number of documents. Times are given in seconds.

As expected, the divide and conquer (D&C) approach is substantially faster on all of the larger datasets. The computation of pairwise similarities is included in the reported times for D&C, while not for affinity propagation, which is why on the smallest datasets AP completes marginally faster. Past about 8000 documents, our hardware[3] is not able to handle the $O(n^2)$ time and, especially, space requirements for AP. On the other hand, even on the entire 20 newsgroups dataset, D&C AP completes in under 20 minutes. For AP, we use the original author's C implementation[4], and our own C implementation for bisecting k-means and the hybrid algorithm. We have not modified AP for D&C, and apply the reference implementation to

[2] http://people.csail.mit.edu/jrennie/20Newsgroups/

[3] All experiments were performed on a standard laptop with 2 GB of RAM.

[4] www.psi.toronto.edu

each cluster sequentially. We expect an optimized implementation to be faster by avoiding many unnecessary allocations/deallocations.

While the efficiency of the D&C approach is consistently superior to standard AP, it is of particular note that some collections benefit significantly in terms of accuracy from the divisions. In particular, batch AP tends to be overly greedy when reducing certain collections (e.g. the tr collections), with reduction ratios below 0.1 and accuracies below 50%; here, the divide phase constrains AP, resulting in more accurate prototypes. This behavior is not entirely consistent: on the k1b collection, D&C AP has a superior reduction ratio but also one that is less accurate. Over all the collections, our D&C strategy produces quite consistant reduction ratios, with the average at a steady 0.21 ± 0.02.

These results can be attributed to the inconsistent quality of the k-means divide phase, since a perfect bisection (that is, one with entirely pure clusters) necessarily results in perfect prototypes according to our notion of accuracy. As an example, the Cornell SMART corpus consists of three well seperated classes, which bisecting k-means clusters with accuracy greater than 95%. This is reflected in the corresponding increase in accuracy for D&C on this collection, at the cost of a poorer reduction ratio. On the other hand, the k-means clustering of the TREC hitech dataset is only 38% accurate, resulting in lower accuracy values for D&C. Fortunately, errors in the initial clustering do not necessarily translate into errors in the prototyping phase, as documents from different classes will tend to be represented by different protytypes, making our method quite robust.

As for the hybrid algorithm, the results tend support the hypothesis articulated in Section 4.1 that more accurate local decisions on the leaves of the tree (if we view the divide phase as such) should result in more accurate prototypes. However, while significant in some cases (e.g. k1b, tr41), we observe that on most collections the hybrid algorithm results in marginal increases in purity, at the expense of rather more significant increases in runtime. In fact, on the sports collection there is a rather substantial *decrease* in accuracy. In general, we have seen better results from a higher values of τ (that is, applying spectral cuts higher in the bisection tree), but these obviously are accompanied by greater runtimes (we used $\tau = 256$ in our experiments).

6 Conclusions and future work.

Our experiments suggest that our divide and conquer approach is not only scalable, but typically maintains and actually surpasses the accuracy of the batch AP algorithm on certain collections. In addition, our algorithms are generally applicable, as the only requirement is a notion of pairwise similarity and we make no assumptions about the distribution of the data. In a practical implementation, the only parameters to tune are the termination conditions for the divide phase, after which to apply spectral cuts and/or affinity propagation.

For future work, it would be interesting therefore to compare these results with those obtained with different types of data. Since natural language is a notoriously difficult application, in particular because our model ignores much information such as the order of the terms, we expect our algorithm to perform more accurately in other domains.

The hybrid algorithm, while not a clear winner, may well have more potential than shown in this paper. First, it can be made more efficient by exploiting the sparse nature of term-document matrices. In particular, the similarity matrix $\mathbf{X}\mathbf{X}^T$ does not need to be explicitely computed if we use the power method to find the approximate second eigenvector [1]. Furthermore, different cut objectives might be more appropriate for data where the clusters have significant overlap, which is quite common with text data [3].

References

[1] David Cheng, Ravi Kannan, Santosh Vempala, and Grant Wang. A divide-and-merge methodology for clustering. *ACM Trans. Database Syst.*, 31(4):1499–1525, 2006.

[2] I. S. Dhillon, Y. Guan, and J. Kogan. Iterative clustering of high dimensional text data augmented by local search. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, 2002.

[3] Chris H. Q. Ding, Xiaofeng He, Hongyuan Zha, Ming Gu, and Horst D. Simon. A min-max cut algorithm for graph partitioning and data clustering. In *ICDM '01: Proceedings of the 2001 IEEE International Conference on Data Mining*, pages 107–114, Washington, DC, USA, 2001. IEEE Computer Society.

[4] Gereon Frahling and Christian Sohler. A fast k-means implementation using coresets. In *SCG '06: Proceedings of the twenty-second annual symposium on Computational geometry*, pages 135–143, New York, NY, USA, 2006. ACM Press.

[5] Brendan J. Frey and Delbert Dueck. Mixture modeling by affinity propagation. In *NIPS '05: Proceedings of 18th international conference on Neural Information Processing Systems*, Vancouver, British Columbia, Canada, 2005.

[6] Brendan J. Frey and Delbert Dueck. Clustering by passing messages between data points. *Science*, 315, 2007.

[7] Nizar Grira and Michael E. Houle. Best of both: a hybridized centroid-medoid clustering heuristic. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 313–320, New York, NY, USA, 2007. ACM Press.

[8] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: an Introduction to Cluster Analysis*. John Wiley & Sons, 1990.

[9] Terran Lane and Carla E. Brodley. Data reduction techniques for instance-based learning from human/computer interface data. In *ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning*, pages 519–526, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[10] Yanjun Li and Soon M. Chung. Parallel bisecting k-means with prediction clustering algorithm. *J. Supercomput.*, 39(1):19–37, 2007.

[11] James McQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th berkeley symposium on mathematical statistics and probability*, pages 281–297, 1967.

[12] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14, Proceedings of NIPS 2001*, 2001.

[13] Raymond T. Ng and Jiawei Han. Clarans: A method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1003–1016, 2002.

[14] Kunal Punera and Joydeep Ghosh. Clump: A scalable and robust framework for structure discovery. In *ICDM '05: Proceedings of the Fifth IEEE International Conference on Data Mining*, pages 757–760, Washington, DC, USA, 2005. IEEE Computer Society.

[15] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.

[16] Shi Zhong and Joydeep Ghosh. Scalable balanced model-based clustering. In *Proceedings of the 3rd SIAM international conference on data mining*, pages 71–83, 2003.

[17] Shi Zhong and Joydeep Ghosh. Generative model-based document clustering: a comparative study. *Knowl. Inf. Syst.*, 8(3):374–384, 2005.