Efficient Evidence Accumulation Clustering for large datasets/big data

Diogo Silva
dasilva@academiafa.edu.pt

Academia da Fora Aérea, Portugal

October 2015

Abstract

Place abstract here. No paragraph breaks.
Keywords: Keyword1, Keyword2, Keyword3, Keyword4, Keyword5

1. Introduction

Advances in technology allow for the collection and storage of unprecedented amount and variety of data, of which there is an interest in performing automated analysis for generation of knowledge and new insights. A growing body of formal methods aiming to model, structure and/or classify data already exist, e.g. linear regression, principal component analysis, cluster analysis, support vector machines, neural networks. Cluster analysis is an interesting tool because it typically does not make assumptions on the structure of the data, which is specially interesting when no prior information about the data is known.

A vast body of work on clustering algorithms exists, but usually different methods are suited to data sets of different characteristics. Currently, there are state of the art algorithms that are more robust than "traditional" algorithms by having a wider applicability or being less dependent on input parameters. One such approach is Evidence Accumulation Clustering (EAC) [1], belonging to the wider class of ensemble methods. EAC is a state-of-the art clustering method that addresses the robustness challenge, but, currently, its computational complexity restricts its application to small data sets.

The present work is concerned with pushing the current limits of the EAC method to large datasets by addressing the challenges of scalability and efficiency without compromising robustness, using technology available in a desktop workstation. Two main approaches exist for scaling: using algorithms with better computational complexity in the EAC steps or turning to parallel and external memory computation for speeding up and addressing the space complexity. Research on using quantum clustering algorithms [2, 3] for EAC proved fruitless mainly due to its prohibitive computational complexity within the EAC context. This moved the focus of research to parallel computation, more specifically General Purpose computation in Graphics Processing Units (GPGPU), and external memory (using hard drives) solutions.

This document is structured as follows. Relevant concepts to understand the work done are presented in section 2. Since EAC is a three step method and each of these must be optimized individually, a section explaining what was done in each step exists. Finally, the implemented optimizations are tested and the results presented in section **??**.

2. Background

Place text here...

2.1. Clustering: main concepts and notation

The goal of data clustering is the discovery of the *natural grouping(s)* of a set of patterns, points or objects [4], by grouping patterns (usually represented as a vector of measurements or a point in space [5]) based on some similarity, such that patterns belonging to the same cluster are typically more similar to each other than to patterns of other clusters.

Within the clustering context, a *pattern* $\mathbf{x}$ is a single data item represented as a vector of d *features* x_i that characterize it. A *pattern set* (or data set) $\mathcal{X}$ is then the collection of all n patterns $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$. The desired clustering, typically, is one that reflects the natural structure of the data, i.e. the original ground truth labeling. *Hard* clustering (or partitional) techniques assign a class label l_i to each pattern $\mathbf{x}_i$. The whole set of labels corresponding to a pattern set $\mathcal{X}$ is given by $\mathcal{L} = \{l_1, \ldots, l_n\}$, where l_i is the label of pattern $\mathbf{x}_i$. A partition P is a collection of k *clusters*, which are

an exclusive subset of nc patterns $\mathbf{x}_i$ taken from the pattern set. A clustering *ensemble* $\mathbb{P}$ is a set of N partitions P^j of a given pattern set, each of which is composed by a set of k_j clusters C_i^j, where $j = 1, \ldots, N$, $i = 1, \ldots, k_j$. Each cluster is composed by a set of nc_i^j patterns that does not intercept any other cluster of the same partition. The relationship between the above concepts is condensed in the following expressions:

Typically, a clustering algorithm will use a *proximity* measure for determining how alike are two patterns, which can either be a *similarity* or a *dissimilarity* measure. A *distance* is a dissimilarity function d which yields non-negative real values and is also a *metric*, which means it obeys the following three properties: identity, symmetry and triangle inequality. Different proximity measures may be more appropriate in certain contexts.

Validity measures measure the quality of a partition. Several *validity measures* exist and they can placed in two main categories [6]. *External* measures use *a priori* information about the data to evaluate the clustering against some external structure. Examples of such measures include the *Consistency Index* [7] and the H-index [8]. *Internal* measures, on the other hand, determine the quality of the clustering without the use of external information about the data.

2.2. Evidence Accumulation Clustering

EAC is a clustering ensemble method. The underlying idea behind ensemble clustering is to take a collection of partitions, a *clustering ensemble*, and combine it into a single partition of better quality than any in the ensemble. The goal of EAC is to find an optimal partition P^* containing k^* clusters that is consistent with $\mathbb{P}$, robust to small variations in $\mathbb{P}$ and has a good fit with ground truth, when available. EAC makes no assumption on the number of clusters in each partition in $\mathbb{P}$. Its approach is divided in 3 steps:

1. **Production** of a clustering ensemble $\mathbb{P}$ (the evidence);
2. **Combination** of the ensemble into a co-association matrix;
3. **Recovery** of the natural clusters of the data.

In the first step, a clustering ensemble is produced. It is of interest to have variety in the ensemble with the intention to better capture the underlying structure of the data, which can be obtained by varying the number of clusters in $\mathbb{P}$ within an interval $[K_{min}, K_{max}]$. The ensemble is usually produced by random initialization of K-Means, specifying only the $[K_{min}, K_{max}]$ interval from which a random number of centroids will be picked. The choice of this interval and its influence on the EAC's properties will be a topic of discussion further ahead.

The ensemble of partitions is combined in the second step, where a non-linear transformation turns the ensemble into a co-association matrix [1], i.e. a matrix $\mathcal{C}$ where each of its elements n_{ij} is the association value between the pattern pair (i, j), which is computed as the number of co-occurrences in the same cluster. The rationale is that pairs that are frequently clustered together are more likely to be representative of a true link between the patterns [9], revealing the underlying structure of the data. The construction of the co-association matrix is at the very core of this method.

The co-association matrix is then used in the final step for obtaining the final partition. The co-association between any two patterns can be interpreted as a proximity measure. Hierarchical algorithms such as Single-Link or Average-Link have been used, since they operate over pair-wise dissimilarity matrices. However, one must convert the original similarity values to dissimilarities. Furthermore, the lifetime criteria can automatically decide the number of clusters by cutting a dendrogram in the interval corresponding to the longest lifetime. The k-cluster lifetime is defined as the range of threshold values on the dendrogram that lead to the identification of k clusters [1].

2.3. K-Means

K-Means is briefly reviewed here, since it is used by EAC. K-Means is a hard clustering algorithm that takes two initialization parameters: the number of clusters and the centroid initializations. It starts by assigning each pattern to its closer cluster based on the cluster's centroid. This is called the **labeling** step since one usually uses cluster labels for this assignment. The centroids are then recomputed based on this assignment, in the **update** step. The new centroids are the mean of all the patterns belonging to the clusters, hence the name of the algorithm. These two steps are executed iteratively until a stopping condition is met, usually the number of iterations, a convergence criteria or both. The initial centroids are usually chosen randomly, but other schemes exist to improve the overall accuracy of the algorithm.

2.4. Single-Link

Single-Link (SL) is a hierarchical agglomerative algorithm that has been used for the last step of EAC. HAC algorithms operate over a pair-wise dissimilarity matrix and output a dendrogram. The main steps of an agglomerative hierarchical clustering algorithm are [5] (1) the creation of a pair-wise dissimilarity matrix of all patterns, where each pattern is a distinct cluster singleton; (2) finding the clos-

est clusters, merge them and update the matrix to reflect this change; and, (3) repeating step 2 until all patterns belong to the same cluster. The algorithm stops when $n-1$ merges have been performed, which is when all patterns have been connected in the same cluster. The proximity measure between clusters in the second step distinguishes between the different HAC linkage algorithms, such as Single-Link , Average-Link, Complete-Link, among others. In SL, the proximity between any two clusters is the the dissimilarity between their closest patterns.

An interesting property of SL, which will be relevant further on, is its equivalence with a Minimum Spanning Tree (MST), an observation first made by [10]. In graph theory, a MST is a tree that connects all vertices together while minimizing the sum of all the distance between them. In this context, the edges of the MST are the distances between the patterns and the vertices are the patterns themselves. A MST contains all the information necessary to build a SL dendrogram. One of the advantages of using an MST based clustering is that it processes only non-zero values while a typical SL algorithm will process all pair-wise proximities, even if they are null.

2.5. Programming for GPUs with CUDA

Parallel processing with Graphics Processing Units was one of the lines of research pursued. Implemented work for the GPU used the Compute Unified Device Architecture (CUDA). This section will introduce a very brief overview of the main concepts to keep in mind for programming GPUs with CUDA.

A GPU is comprised by one or several streaming processors (or multiprocessor). Each of these processors contains several simpler processors, each of which execute the same instruction at the same time at any given time. In the CUDA programming model, the basic unit of computation is a *thread*. Threads are grouped into *blocks* which are part of the block *grid*. The number of threads in a block is typically higher than the number of processors in a multiprocessor. For that reason, the hardware automatically divides the threads from a block into smaller batches, called *warps*. The computation of one block is independent from other blocks and each block is scheduled to one multiprocessor, which means that more multiprocessors results in more blocks being processed at the same time.

Depending on the architecture, GPUs have several types of memories. Accessible to all processors (and threads) are the global memory, constant memory and texture memory, of which the last two are read-only. Blocks share a smaller but significantly faster memory called shared memory, which is a memory inside a multiprocessor to which all processors have access to, enabling inter-thread communication inside a block. Finally, each thread has access to local memory, which resides in global memory space and has the same latency for read and write operations. However, if the thread is using only single variables or constant sized arrays, it uses register space, which is very fast.

Typically, CUDA applications follow the same flow. First, the host starts by transferring any necessary data to the device memory. The next step is selecting the *kernel* (the function that will run on the GPU processors) and the thread topology (configuration of threads in a block and blocks in the grid). The set-up phase is followed by the computation phase in the GPU. Finally, the host will transfer back the results from the device.

3. Optimization of the production step of EAC

The main contribution for the optimization of the production of clustering ensemble is the implementation of a parallel K-Means for GPUs. Several parallel implementations of this algorithm for the GPU exist in literature [11, 12, 13] and all report speed-ups relative to their sequential counterparts.

The implementation of the present work followed the version in [14], which only parallelizes the labeling stage. This was further motivated from empirical data that suggested the average theoretical maximum speed-up for the labeling stage was 880, whereas for the the update phase the maximum was only 1.5. The CUDA implementation of the labeling step starts by transferring the data to the GPU (pattern set and initial centroids) and allocates space for the labels and corresponding distances. Still, several parameters are accessible to the user, such as the shape of the block of threads and grid of blocks, if data transfer should be handled by the CUDA API or optimized to minimize communication between host and device, and also how many patterns to process per thread. The computation of a label for one pattern is done by iteratively computing the distance from the pattern to each centroid and storing the label and the distance corresponding to the closest centroid to that pattern. Finally, the labels and distances are sent back to the host for computation of the new centroids. The implementation of the centroid computation starts by counting the number of patterns attributed to each centroid. Afterwards, it checks if there are any "empty" clusters, i.e. if there are centroids that are not the closest ones to any pattern. Dealing with empty clusters is important because the target use expects that the output number of clusters be the same as defined in the input parameter. Centroids corresponding to empty clusters will be the patterns that are furthest away from their centroids. Any other centroid c_i will be the mean of the patterns that were labeled i.

4. Optimization of the combination step of EAC

Space complexity is the main challenge building the co-association matrix. A complete pair-wire matrix has $O(n^2)$ complexity but can be reduced to $O(\frac{n(n-1)}{/}2)$ without loss of information. Still, these complexities are rather high when large data sets are contemplated, since it becomes impossible to fit these "conventional" co-association matrices in main memory. Two solutions in literature address this challenge. [1] approaches it by using a k-Nearest Neighbor approach, only considering associations between the k closes neighbors of each pattern. [15] approaches the problem by exploiting the sparse nature of the co-association matrix for reducing space complexity, but doesn't cover the efficiency of building a sparse matrix or the space overhead associated with sparse data structures. The effort of the present work was focused on further exploiting the sparse nature of EAC, building on previous insights and exploring the topics that literature has neglected so far.

Building a non-sparse matrix is easy and fast since the memory for the whole matrix is allocated and indexing the matrix is direct. When using sparse matrices, neither is true. In the specific case of EAC, there is no way to know what is the number of associations the co-association matrix will have which means it is not possible to pre-allocate the memory to the correct size of the matrix. This translates in allocating the memory gradually which may result in fragmentation, depending on the implementation, and more complex data structures, which incurs significant computational overhead. For building a matrix, the DOK (Dictionary of Keys) and LIL (List of Lists) formats are recommended in the documentation of the SciPy [16] scientific computation library. These were briefly tested on simple EAC problems and not only was their execution time several orders of magnitude higher than a traditional fully allocated matrix, but the overhead of the sparse data strucutres resulted in a space complexity higher than what would be needed to process large data sets. For operating over a matrix, the documentation recommends converting from one of the previous formats to either CSR (Compressed Sparse Row) or CSC (Compressed Sparse Column). Building with the CSR format had a low space complexity but the execution time was much higher than even one of the other sparse formats.

The bad performance of the above sparse formats combined with the fact that no relevant literature was found on efficiently building sparse matrices led to the design and implementation of a novel strategy for a CSR matrix specialized to the EAC context, the **EAC CSR**.

5. Optimization of the recovery step of EAC

Place text here...

5.1. Sub-section...

More text...

5.2. Sub-section...

More text...

6. Results

Place text here...

6.1. Sub-section...

More text...

Figure 1 shows the contour of pressure on the hub and blade surface planes corresponding to the baseline blade geometry.

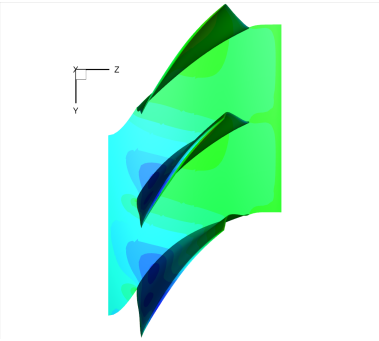

Figure 1: Pressure distribution.

As seen in Fig.1...

6.2. Sub-section...

More text...

Table 1 summarizes...

Model	C_L	C_D	C_{My}
Euler	0.083	0.021	-0.110
Navier–Stokes	0.078	0.023	-0.101

Table 1: Table caption

As seen in Tab.1...

7. Conclusions

Conclusions, future work and some final remarks...

Acknowledgements

The author would like to thank ...

References

[1] Ana N L Fred and Anil K Jain. Combining multiple clusterings using evidence accumulation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(6):835–850, 2005.

[2] Ellis Casper, Chih-Cheng Hung, Edward Jung, and Ming Yang. A Quantum-Modeled K-Means Clustering Algorithm for Multi-band

Image Segmentation. In *Proceedings of the 2012 ACM Research in Applied Computation Symposium*, volume 1, pages 158–163, 2012.

[3] David Horn and Assaf Gottlieb. The Method of Quantum Clustering. *NIPS*, (1), 2001.

[4] Anil K Jain. Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*, 31(8):651–666, 2010.

[5] a. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[6] Charu C Aggarwal and Chandan K Reddy. *Data clustering: algorithms and applications*. CRC Press, 2013.

[7] Ana Fred. Finding consistent clusters in data partitions. *Multiple classifier systems*, pages 309–318, 2001.

[8] Marina Meila. Comparing clusterings by the variation of information. *Learning theory and Kernel machines: 16th Annual Conference on Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, USA, August 24-27, 2003: proceedings*, page 173, 2003.

[9] Ana N L Fred and Anil K Jain. Data clustering using evidence accumulation. *Object recognition supported by user interaction for service robots*, 4, 2002.

[10] J. C. Gower and G. J. S. Ross. Minimum Spanning Trees and Single Linkage Cluster Analysis. *Journal of the Royal Statistical Society*, 18(1):54–64, 1969.

[11] Hong Tao Bai, Li Li He, Dan Tong Ouyang, Zhan Shan Li, and He Li. K-means on commodity GPUs with CUDA. *2009 WRI World Congress on Computer Science and Information Engineering, CSIE 2009*, 3:651–655, 2009.

[12] Mario Zechner and Michael Granitzer. Accelerating k-means on the graphics processor via CUDA. *Proceedings of the 1st International Conference on Intensive Applications and Services, INTENSIVE 2009*, pages 7–15, 2009.

[13] J Sirotkovi, H Dujmi, and V Papi. K-Means Image Segmentation on Massively Parallel GPU Architecture. pages 489–494, 2012.

[14] Mario Zechner and Michael Granitzer. K-Means on the Graphics Processor: Design And Experimental Analysis. *International Journal on Advances in System and Measurements*, 2(2):224–235, 2009.

[15] André Lourenço, Ana L N Fred, and Anil K. Jain. On the scalability of evidence accumulation clustering. *Proceedings - International Conference on Pattern Recognition*, 0:782–785, 2010.

[16] Eric Jones, Travis Oliphant, Pearu Peterson, and Others. {SciPy}: Open source scientific tools for {Python}.