Distributed Data Mining in Peer-to-Peer Networks

Souptik Datta Kanishka Bhaduri Chris Giannella Ran Wolff
Hillol Kargupta[*]
Dept. of Computer Science & Electrical Engineering
University of Maryland Baltimore County Baltimore, MD, USA
{souptik1,kanishk1,cgiannel,hillol}@cs.umbc.edu

Abstract

Distributed data mining deals with the problem of data analysis in environments with distributed data, computing nodes, and users. Peer-to-peer computing is emerging as a new distributed computing paradigm for many novel applications that involve exchange of information among a large number of peers with little centralized coordination. Peer-to-peer file sharing, peer-to-peer electronic commerce, and peer-to-peer monitoring based on a network of sensors are some examples. This paper offers an overview of distributed data mining applications and algorithms for peer-to-peer environments. It describes both exact and approximate distributed data mining algorithms that work in a decentralized manner. It illustrates these approaches for the problem of computing and monitoring clusters in the data residing at the different nodes of a peer-to-peer network.

Keywords: Distributed data mining, peer-to-peer.

1 Introduction

Advances in computing and communication over wired and wireless networks have resulted in many pervasive distributed computing environments. The Internet, intranets, local area networks, ad hoc wireless networks, and sensor networks are some examples. These environments often come with different distributed sources of data and computation. Mining in such environments naturally calls for proper utilization of these distributed resources. Moreover, in many privacy sensitive applications different, possibly multi-party, data sets collected at different sites must be processed in a distributed fashion without collecting everything to a single central site. However, most off-the-shelf data mining systems are designed to work as a monolithic centralized application. They normally down-load the relevant data to a centralized location and then perform the data mining operations. This centralized approach does not work well in many of the emerging distributed, ubiquitous, possibly privacy-sensitive data mining applications.

Distributed Data Mining (DDM) offers an alternate approach to address this problem of mining data using distributed resources. DDM pays careful attention to the distributed resources of data, computing, communication, and human factors in order to use them in a near optimal fashion. Distributed peer-to-peer (P2P) systems are emerging as a choice of solution for a new breed of applications such as file sharing, collaborative movie and song scoring, electronic commerce, and surveillance using sensor networks. Distributed data mining is gaining increasing attention in this domain for advanced data driven applications.

This paper presents an exposure to P2P distributed data mining technology and its applications in various domains. The goal of the paper is to present a high level introduction to this field with pointers for further exploration. The paper discusses applications of P2P distributed data mining and illustrates the ideas using some exact and approximate P2P algorithms.

Section 2 introduces P2P data mining, presents the motivation, and identifies the immediate applications. Section 3 discusses some of the main challenges in P2P data mining. Section 4 presents a brief review of the related literature and offers a historical perspective. Section 5 describes several P2P data mining algorithms. Finally, Section 6 concludes this paper.

2 Peer-to-Peer (P2P) Data Mining: Why Bother?

Large-scale data analysis is likely to play a key role in the next generation of data driven collaborative problem-solving systems. Collaborations between human beings, software entities, embedded systems, and other computing devices are likely to revolutionize many application domains such as cross-domain P2P multi-organizational

[*]Also affiliated with AGNIK LLC, Columbia, MD USA.

cyber-threat detection, multi-organizational collaboration for homeland defense related applications, and unmanned border monitoring using sensor networks. Successful collaborative systems for such domains require scalable distributed data analysis capabilities from large-scale distributed, possibly multi-party, data sources.

The phrase 'Data Mining' generally portrays a picture of analyzing huge databases, mostly in the form of large tables, for useful patterns. On the other hand, P2P networks reminds us mostly of well known, file sharing networks based on a point-to-point connection, without any central server, *e.g. e-Mule, Kazaa*. With the advent of high speed network connectivity and cheap digital storage/data recording devices, these types of server-less networks are growing very fast. Collectively they already store a huge amount of widely varying data collected from different sources. If this data, distributed over large number of peers, can be integrated, it represents a very valuable data repository that, upon mining, may give very exciting and useful results. For example, had it been possible to integrate all the image files stored over all Kazaa peers, and running a content-wise clustering algorithm on the whole data, the resulting clusters would have been an excellent resource for answering image search queries. However, in practice, it is very difficult to upload all the Kazaa peers' images to a central repository where a centralized image clustering algorithm can be run. However, if the same, or close approximation of the clustering result can be achieved without moving any images from their peers, the result would likewise provide an excellent query routing resource (rout the search query to the peers having high members counts corresponding to the most relevant cluster). The same is true for topic-wise document clustering in a peer-to-peer document repository. Figure 1 shows such a case where documents stored in different peers are clustered based on three subjects (movies, baseball, hurricane) by exchanging information with other peers. Note, in a peer-to-peer clustering, some peers may not be present in the network all the time, and may join or leave the network while the clustering is in progress.

A primary goal of P2P data mining is to achieve the same (or close) data mining result as a centralization approach, without moving any data from its original location. To achieve this goal P2P algorithms must possess a unique set of characteristics including:

- the ability to efficiently scale-up (P2P systems exist today consisting of millions of peers);
- the ability to perform in a router-less network (critical for sensor networks);
- the ability to calculate the result in-network rather than collect all of the data to a central processor (which would quickly exhaust bandwidth in both sensor and peer-to-peer networks);
- the ability to function correctly in the presence of peer/edge failures and data change.

To further motivate P2P data mining, next we briefly consider other application areas.

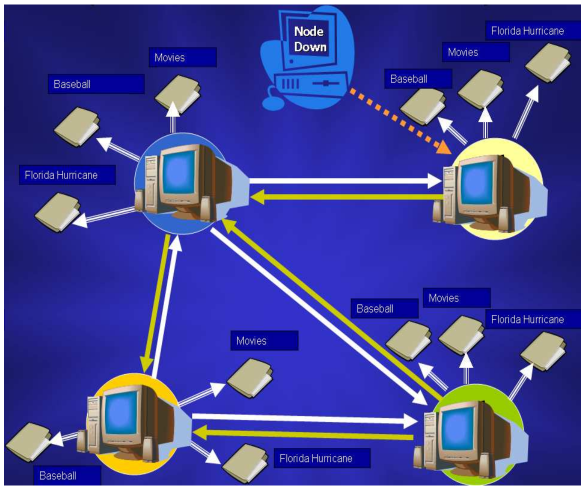

Figure 1. Document clustering in peer-to-peer file sharing network.

- **Sensor Networks:** Light-weight, inexpensive sensors with wireless communication capabilities can be easily deployed in large numbers to form sensor networks. The potential for such networks is already widely recognized. They can be deployed in hostile/difficult to reach locations to provide detailed environmental information at a fine spatial granularity. In these networks, the preservation of battery power is critical and wireless communication is significantly more energy costly than computation. Therefore, data analysis algorithms must operate in a communication-efficient fashion. Moreover, peers may need to operate in a router-less environment with no global IPs. Finally, due to the limited power and hostile environment, light-weight, wireless sensor networks are highly dynamic with peer and edge failure a common occurrence. A P2P data mining application like outlier detection or scene segmentation may be very useful in such a network. Consider sensors deployed over a deep sea bed to monitor seismic vibrations for the tsunami forecasting. These sensors form an ad-hoc peer-to-peer network amongst themselves and the central monitoring station. The user of this sensor network is is perhaps least bothered about the network details. All he likes to know is the top few vibration characteristics along with their positions periodically from the entire network to check possibility of tsunami. For this, the sensors need to communicate between themselves to decide about the highest values of vibration measured, and need to send the top-ranked values to the user. A distributed order detection algorithm that can run over a P2P network to detect the top-k values in a communication-efficient manner is useful for such an application.

- **Mobile Ad-hoc Networks (MANETs):** MANETs are getting increasing attention in many wireless application domains. Such ad-hoc networks may play a key role in defining how we communicate at work and social environments in the future. Several data rich environments (e.g. vehicular ad-hoc networks, P2P e-commerce environments) are emerging and they are likely to need data analytic supports for efficient and personalized services. Consider a simple profiling application launched via cellular phones that tries to automatically connect to MANET like network formed by different cellular phones in the vicinity and identify peers with similar interest to form a social network. A lightweight P2P classification algorithm may be very handy in such an application.

While these application-areas differ in some respects, they share the common denominator of motivating a new breed of data analysis and mining algorithms capable of operating effectively on dynamic, large-scale P2P networks. Perhaps by this time, the reader is convinced such algorithms can solve several potentially exciting problems, and have ever increasing value in the future. Next, we discuss a few novel challenges faced in developing such algorithms.

3 Challenges in a Peer-to-peer Computational Environment

The computational environment in P2P systems is drastically different from the ones for which traditional data mining algorithms were intended. To understand the requirements, consider an analogy with Internet protocols. Internet routers invest bandwidth, memory, and computation in routing protocols which yield faster communication. Likewise, peers would invest CPU, memory, bandwidth, and – in some cases (*e.g.* sensor-networks) – battery power, as well as their local data, if the result of the computation would yield them some benefit. Both systems have similar scales (of the order of millions), the same communication patterns (sparsely connected network with time varying loads), the same failure pattern, and data dynamics (data in peers constantly changes, just as latency of connections changes). Furthermore, both are necessarily open systems, which have limited control over their components.

Below are a list of operational characteristics desired for data mining algorithms developed for a P2P network.

1. *Scalability* – Scalability is without doubt the foremost requirement for a peer-to-peer algorithm. Ideally algorithms for P2P networks should be either independent of the size of the system, or at most dependent on the log of the size. Because we assume each peer also has some data, scalability with respect to the data size is equally as important. Typically the resources required by such algorithms would be dependent on the size of their output rather than on the input.

2. *C*ommunication Efficient – P2P data mining algorithms must be able to work in a communication efficient manner. Although this category somewhat overlaps with the scalability requirement, we decided to list it explicitly in order to underscore its importance. Although, many P2P systems are designed for sharing large data files (e.g. music, movies), a distributed data mining system that involves analyzing such data, may not have the luxury to frequently exchange large volume of data among the nodes in the P2P network just for data analysis. Data mining in a P2P environment should be "light-weight"; it should be able to perform distributed data analysis with minimal communication overhead.

3. *Anytimeness* – For a real-world P2P system, data at the peers often change. Thus, the algorithms have to work incrementally. This is especially right for sensor networks where generally the data is considered to be streaming. Any algorithm that needs to begin from scratch whenever the data changes is thus inappropriate for our goals. Hence, incremental algorithms are required. Furthermore, since the rate of data-change may be higher than the rate of computation in some applications, the algorithm has to be able to report a partial, ad hoc solution, at any time. These are called *anytime* algorithms.

4. *Asynchronism* – Peer-to-peer systems can be very large (number of nodes in the range of million). So any attempt to synchronize between the entire network is likely to fail due to connection latency, or limited bandwidth in case of sensor networks. Thus, any algorithm developed for peer-to-peer system should not take the route of global synchronization.

5. *Decentralization* – Although some peer-to-peer systems still use central servers for various purposes, overall, this method is considered unsuitable for most tasks. This is especially true for tasks involving large amounts of data. The algorithms need to be designed to transfer minimal data (*i.e.*, they compute in-network) and use no centralized coordination.

6. *Fault-tolerance* – The larger the system, the more frequent are failures. In some current peer-to-peer sys-

tems, it is not unusual for several peers to leave and join the system at any given moment. Thus, the failure of peers, and the subsequent loss of data and partial results they maintain is not a rare event from which the system should be able to recover, but rather a common event to which the algorithm must be robust.

4 Evolution of Peer-to-Peer Data Mining

P2P data mining is a new field. It has emerged recently from distributed data mining, motivated by the rapid growth of P2P networks. Distributed data mining is itself a very young area. It has been developed in the last 5-10 years and has resulted in many distributed versions of standard algorithms, *e.g.* association rule mining, Bayesian network learning, clustering. However, most of this work was not geared toward P2P networks as they assumed a stable network and data.

Approaches to P2P data mining have focused on developing some primitive operations as well as more complicated data analysis/mining algorithms. Researchers have developed several different approaches for computing primitive operations (average, sum, max, random sampling) on P2P networks. For example, Kempe *et al.* [5] investigate gossip based randomized algorithms. They prove that the error will go to zero in probability if the algorithm runs uninterrupted. Jelasity and Eiben [6] develop the 'newscast model' as part of the DREAM project[1]. They rely on empirical accuracy results rather than guaranteed correctness. Both of the above approaches used an epidemic model of computation. Bawa *et al.* have developed [2] an approach in which similar primitives are evaluated to within an error margin. A main goal of these works is to lay a foundation for applications and more sophisticated data analysis/mining algorithms (efficient complex algorithms can, in principle, be developed from the application of efficient primitives).

The common feature of all the approaches mentioned so far is that they all require resources that scale directly with the size of the system. This feature distinguishes these from *local algorithms*. Such algorithms [1] computed their result using information from just a handful of nearby neighbors. Even still, it is possible to make definite claims regarding correctness. The resources required by these algorithms are independent of the size of the system in many cases. The obvious benefit is their superb scalability, which make them a good fit for networks spanning millions of peers. They are also very good at adjusting to failure and changes in the input locally, so far as the output need not change. However, a disadvantage of local algorithms is the limited class of functions to which they can apply.

[1]www.dcs.napier.ac.uk/ benp/dream/private.htm

Researchers have focused on developing local algorithms for primitive operations. Mehyar *et al.* have used a graph Laplacian-based approach to compute the average over a P2P network [8]. Wolff and Schuster have developed a local algorithm for computing the majority vote over a P2P network [10]. Based on this last primitive, local algorithms have been developed for more complicated problems: K-facility location [7], association rule mining [10], and monitoring a K-means clustering [9]. Researchers have also focused on developing approximate local algorithms for complicated problems. Datta *et al.* developed an approximate local algorithm for K-means clustering [3] in a P2P network.

Data mining in wireless sensor networks (WSNs) is related to data processing over a P2P network, and this field has emerged even more recently. This is a more challenging sub-area in P2P data mining, as algorithms need to work in extremely demanding and constrained environment of sensor networks (limited energy, storage, computational power, bandwidth *etc.*). WSNs also require highly decentralized algorithms. More details about information processing in sensor network can be found at [11].

5 Algorithms in Peer-to-Peer Data Mining

Peer-to-peer networks today hold a huge amount of data that, if mined, can be a source of extremely useful information. Also, it is clear that the environment offers a unique set of challenges for the development of data mining algorithms – extending standard centralized data mining algorithms to such environment is difficult. Now we discuss some algorithms developed to address data mining primitives like sum, average over the network, as well as more complicated problems like K-means clustering. We will first introduce the concept of local algorithm, and then discuss different variants of it.

Let us formally define first what do we mean by *local algorithm*. An algorithm is local if, assuming a static network and data, there exists a constant k such that for any network size, there are inputs such that the algorithm terminates with communication expended per peer no greater than k and on the rest of the inputs the communication expended per node is on the order of network size. Local algorithms are very attractive for large scale systems like peer-to-peer systems and sensor networks because of their scalability.

They can be broadly classified under two categories.

- *Exact local algorithms:* Those which are guaranteed to always terminate with precisely the same result that would have be found by a centralized algorithm.

- *Approximate local algorithms:* Those which cannot make this accuracy guarantee.

Exact local algorithms are obviously more desirable, but are more difficult to develop (in some cases seemingly not possible). We will discuss examples in both categories and, in particular, illustrate how the classic data mining problem of K-means clustering can be addressed in each.

Next we describe an exact local algorithm for *majority voting*, and show how it can be used as a primitive for monitoring a K-means clustering [9]. Following this, we describe a approximate local algorithm offering an approximate solution for incrementally computing a K-means clustering. Before doing all of this, we briefly discuss classical K-means clustering for readers not familiar with the problem.

K-means clustering: Simply put, clustering is the grouping of similar objects (data points) in a way that minimizes intra-cluster dissimilarity while maximizes inter-cluster dissimilarity. K-means clustering is a classical clustering technique where K, the number of clusters, is fixed a priori. The goal is to divide the objects into K clusters minimizing the sum of the average distances to the centroids over all clusters. Finding an optimal clustering is hard (NP-complete), so the most common approach is to employ an iterative, greedy search as described below (data points are tuples in $\mathbb{R}^m$):

1. Place K points randomly. These represent the initial cluster centroids.

2. Assign each point to the closest centroid.

3. Recalculate the positions of each centroid (the average of all data points assigned).

4. Repeat Steps 2 and 3 until the centroids do not change. The assignment of points to the final centroids forms the clustering.

5.1 Exact Local Algorithms

All of the exact local algorithms assume that a tree topology has been overlaid on top of the network.

5.1.1 Majority Voting

This problem serves as a nice primitive upon which more complicated exact local algorithms can be developed (all material here is based on [10]). In the majority voting problem, each peer, P_i, has a number b_i either zero or one, and a threshold $\tau > 0$ (the same threshold at all peers). The goal is for the peers to collectively determine whether $\sum_i b_i$ is above $n\tau$ where n is the number of peers in the network. The approach described here can be easily extended to a more general scenario where each peer has a real number x_i and the collective goal is to decide whether $Avg(x_i) > \tau$. For simplicity we do not describe the extension but will use it later when discussing the exact local algorithm for K-means monitoring.

Peer P_i only communicates with its neighbors in the network, N_i. P_i maintains S_i, an estimate of the global sum, and C_i, an estimate of the number of nodes in the network. These estimates are based on all of the information P_i last received from its neighbors. Based on these estimates P_i believes that the majority threshold is met if $S_i - C_i\tau > 0$, otherwise it believes the majority has not been met. We call this the *threshold belief* of P_i.

Let S_{ji} denote the most recent sum estimate sent to P_i from its neighbor P_j. Likewise define C_{ji} as the most recent number of nodes estimate sent from neighbor P_j. The crux of the approach lies in deciding whether P_i needs to send a message to its neighbor P_j. If P_i can be certain that it does not have any information that will cause the threshold belief of P_j to change, then it need not send a message. If P_i cannot be certain, a message must be sent. In order for P_i to make this decision, it must estimate the sum and count of P_j based on the information it knows for certain P_j has, namely, that sent to P_j and sent from P_j: S_{ij}, C_{ij} and S_{ji}, C_{ji}.

Suppose P_i estimates that P_j believes the threshold to be met (*i.e.* $S_{ij} + S_{ji} - (C_{ij} + C_{ji})\tau > 0$). It need not send a message if its own estimate can only strengthen this belief (*i.e.* $S_{ij} + S_{ji} - (C_{ij} - C_{ji})\tau \leq S_i - C_i\tau$). In this case, P_i can be certain that it does not have any information that could change the threshold belief of P_j. Similar reasoning applies if P_i estimates that p_j does not believe the threshold to be met. If P_i decides to send a message, then it sends all of its information about the global sum excluding that sent from P_j (*i.e.* S_{ij} is set to $b_i + \sum_{\ell \neq j \in N_i} S_{\ell i}$) and likewise for the global count (*i.e.* C_{ij} is set to $1 + \sum_{\ell \neq j \in N_i} C_{\ell i}$).

This approach is naturally robust to data and network change. If P_i's data changes (*i.e.* b_i flips), then P_i recomputes S_i and C_i and applies the above conditions to all its neighbors. If a neighbor P_j drops out of the network, then P_i recomputes S_i and C_i without S_{ji} and C_{ji} and applies the above conditions to all remaining neighbors.

5.1.2 Monitoring a K-Means Clusterings

In this section we present an algorithm for monitoring a K-means clustering of data distributed over a P2P network (all material is based on [9]). The algorithm does not solve the problem of computing a K-means clustering. Doing this is an an exact local manner is quite difficult (perhaps impossible). Instead, the algorithm monitors when the centroids computed by a centralized K-means (and distributed to all nodes in the network) is no longer accurate. At this point the centralization approach is invoked. In Section 5.2.1, an *approximate* local algorithm for K-means is presented.

We begin by explaining why a monitoring algorithm is useful. In large scale applications, peer-to-peer system status data is collected as part of their daily routine. This data is often necessary to build complex models representing the state of the system. Examples for such use of global models include facility location in sensor networks [7], trust assignment in peer-to-peer file sharing [4]. It is obvious that building global models (by aggregating all the data) is difficult (if not impossible) mainly due to cost in communication and more so when the data distribution changes.

There can be two ways we can compute these models in a dynamic environment where the data distribution changes (epoch changes) interleaved with long periods of static behavior. One of the ways is to build the model *periodically*. While this scheme is simple, it has several disadvantages. If the data change is sporadic, and if the cost of monitoring is far lower than the cost of recomputing the model then this type of algorithms will unnecessary waste resources by recomputing models during stationary periods as well. Also it has the risk of being inaccurate whenever the distribution of the data changes. The other alternative is to use a *reactive* method of computing the model whenever the data distribution changes. This method not only consumes far less resources during the stationary phase but also tries to update and build a new model as soon as the data distribution changes. The *reactive* approach is thus a valid alternative to *periodic* algorithms which are in many cases inaccurate and inefficient.

The K-means monitoring algorithm has two major parts - monitoring the data distribution in order to trigger a new run of K-means algorithm and computing the centroids actually using the K-means algorithm. The monitoring part is carried out by an exact local algorithm, while the centroid computation is carried out by a centralization approach. Simply put, the local algorithm raises an alert if the centroids need to be updated. At this point data is centralized, a new run of K-means is executed, and the new centroids are shipped back to all peers. Henceforth we only focus on the monitoring part.

The monitoring phase is very tightly connected to the majority voting algorithm. First let us define the metric we use in order to determine whether the current centroids represent the data. At a given time t, each peer P_i has a multidimensional local dataset $S_{i(t)}$. It also has the current centroids computed by running centralized K-means on the complete dataset(from the last exact K-means execution). The peer computes an average of all of its data points (the average is a vector, one average value for each dimension). It then computes the component-wise difference between the average and the current centroids to get a vector, which we call $\vec{X}_i$ (the *knowledge of* P_i). Since this is a vector, a reliable measure of its value is the L2-Norm. For K-means monitoring at any peer, the problem reduces to checking whether the L2 norm of that vector is significant. In other words, the problem is: given the difference vector $\vec{X}_i$, check if $||\vec{X}_i|| \geq \epsilon$ or $||\vec{X}_i|| \leq \epsilon$, where ϵ is user-defined threshold. For the 2-D case, the shape of the L2-Norm of a vector is a circle (sum of squares of individual components) and hence the problem can be reduced to a set of majority votes to determine if $||\vec{X}_i||$ is outside or inside the circle. The first step is to approximate this figure using a set of tangent lines which are defined by a set of unit vectors. Figure 2 shows such a figure. The unit vectors are shown with arrows. Also shown in this figure are the tangent lines (the dotted lines). This shows that the circle has been approximated to a polygon. At this stage we are interested in the three cases : $||\vec{X}_i||$ is inside the circle (region (A) in Figure 2), $||\vec{X}_i||$ is outside the circle and $||\vec{X}_i||$ is in the region between the circle and polygon. The first can be solved using a simple majority voting mechanism where we check if the L2-norm of a point in $\mathbb{R}^2$ is less than ϵ. The second case however is a little tricky to solve (note that if two peers say that a point is outside the circle, the points can be on the two opposite ends and hence the sum can still be inside the circle). For this reason we use majority votes along each of these tangent lines. Thus if there are d tangent planes, there are $d+1$ majority votes, one each for checking the inside conditions and the rest for the outside. The most difficult case is however to when the $||\vec{X}_i||$ is in the region (D) in the figure. Since no peer is certain as to what to do, we have to resort to flooding in this scenario. Note that this space can be made arbitrarily small using more and more number of tangent lines. The cost is however increase in computation for each peer.

Experiments with this algorithm shows excellent accuracy and scalability. This algorithm has been simulated in a testbed of 1000 peers. We ran simulations for 500, 1000, 2000 and 4000 peers and the accuracy does not change. Also the number of messages per peer remains constant, irrespective of the size of the network. This typifies local algorithms - they are infinitely scalable.

5.2 Approximate Local Algorithms

In the previous section, we have talked about *exact local* algorithms that can be very useful in solving data mining problems in peer-to-peer networks. Although these algorithms achieve exact solutions eventually, they are limited to problems which can be reduced to threshold predicates. For example it is very difficult (perhaps impossible) to develop an exact local algorithm to compute the mean of a set of numbers distributed over the network, but not so for evaluating whether the mean lies above a threshold. In light of this, some data mining problems are very difficult, perhaps impossible, to solve with exact local algorithms – in particular, K-means clustering. In the previous section, what we described is mostly monitoring the K-means clustering. In

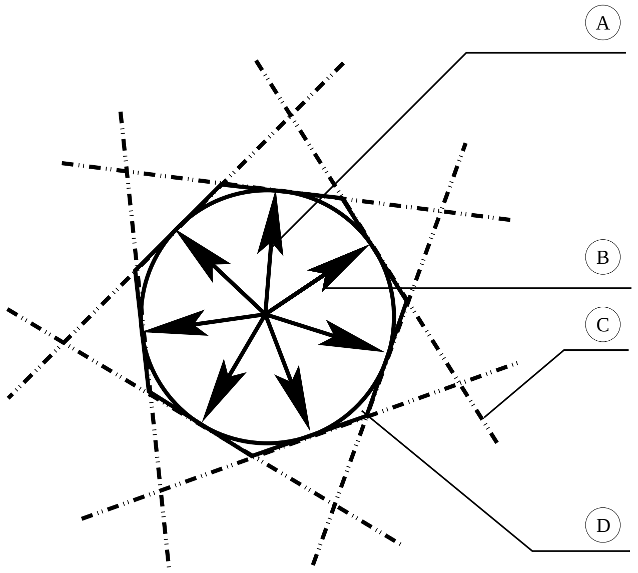

Figure 2. (A) the area inside an ϵ circle. (B) Seven evenly spaced vectors which defines the tangent planes. (C) The borders of the seven half-spaces define a polygon in which the circle is circumscribed. (D) The area between the circle and the union of half-spaces.

this section, we develop an approximate local algorithm for K-means clustering.

5.2.1 Peer-to-peer K-means Algorithm

This is an iterative algorithm based on message exchange between directly connected peers to approximately solve the K-means clustering problem in peer-to-peer network (all material here is based on []). The algorithm is initiated with a set of randomly chosen starting centroids distributed over all peers. In each iteration, each peer runs a two-step process. The first step is identical to one iteration of standard K-means where i-th peer P_i assigns each of its own point to its nearest of k cluster-centroids. Let $\{w_{j,k}^{(i)} : 1 \leq j \leq K\}$ denote the corresponding centroids (*local centroids*) and $|w_{j,k}^{(i)}|$ denote the number of points in i-th peer associated with it (*cluster counts*) on k-th iteration. P_i stores these local centroids and counts for answering queries from its neighbors. In the second step, peer P_i sends a poll message, consisting its id and current iteration number to its immediate neighbors and waits for their responses. Each response message from a neighboring peer P_a contains the locally updated centroids and cluster counts of that peer for k-th iteration. Once all immediate neighbors of P_i have responded or cease to be neighbors, P_i updates its j^{th} centroid by taking a weighted average of all the centroids received by it and its own centroid, weights being corresponding number of points in that cluster. Then it moves to the next iteration of K-means and repeats the whole process.

If the maximum change in position of the new centroid after an iteration remains above a user-defined threshold, then peer P_i goes on to iteration $k+1$. Otherwise it enters the terminated state.

Note that other than executing the above two steps, at any point, each peer also needs to respond to any polling message received from its neighbor. Suppose peer P_i receives a polling message from peer P_h during its iteration k. The message corresponds to iteration $\hat{k}$ at peer P_h. If $\hat{k} \leq k$, then P_i sends its local centroids and counts from iteration $\hat{k}$. Otherwise, this poll message is placed in a queue. P_i will check this queue at every iteration and respond to any messages it can. Any peer, P_i, can enter a terminated state at the end of iteration k if its cluster centroids stops changing significantly . In that case, P_i no longer updates its centroids or sends polling messages. However, it does respond to polling messages from peer h for iteration $\hat{k}$ by sending its local centroids and counts from iteration $min\{\hat{k}, k\}$. Note that, no peer has an explicit condition under which all activity stops. However, once a peer enters the terminated state, it no longer sends polling messages, only responses. Therefore, once all peers enter into the terminated state, all communication ceases *i.e.* the algorithm has terminated.

The algorithm can adjust itself to changing network and dynamic data in peer with simple mechanism to detect change in network/data. Any new peer joining the network can join the ongoing clustering algorithm by syncing to the ongoing minimum iteration in its neighborhood. Change in data content of any peer just reassigns the cluster centroids in that peer and move on to the next iteration.

Extensive experiments with the algorithm showed that the clustering accuracy, in comparison to centralized K-means clustering algorithm (the hypothetical case where data present in all peers are integrated and standard K-means is applied on the data as a whole), is more than 90%. The scalability of this algorithm is very good, as the average accuracy of clustering achieved remains more-or-less the same with increase in network size.

6 Conclusions

Distributed data mining is a natural choice when the data mining environment has distributed data, computing resources, and users. This paper focused on an emerging branch of distributed data mining—peer-to-peer data mining. P2P data mining applications may play a key role in the next generation of file sharing networks, sensor networks, and mobile ad hoc networks. This paper offered an exposure to the recent literature in this area. It also offered a sampler of exact and approximate P2P algorithms for clustering in such distributed environments.

Data analysis in P2P environments offers a wide spectrum of challenges for the researchers and practitioners. Designing distributed, asynchronous, decentralized algorithms offers many difficult challenges. Many algorithms are starting to emerge. However, maturing these algorithms and integrating them with real P2P applications offer additional challenges. Most of the P2P data mining algorithms rely upon asymptotic convergence properties. Finite-time behavior needs more attention. Advanced analysis of such systems such as stability must be quantified. The scope of the exact P2P algorithms is usually restricted to functions that can have a local representation in the given network. This paper discussed approximate techniques which can be a way to deal with non-local functions. We are currently exploring an ordinal framework to relax the cardinal data mining problems and develop ordinal algorithms for P2P networks.

Acknowledgments

This work was supported by the United States National Science Foundation Grant IIS-0093353.

References

[1] B. Awerbuch, A. Bar-Noy, N. Linial, and D. Peleg. Compact distributed data structures for adaptive network routing. *Proceedings of the 21st ACM Symposium on the Theory of Computing (STOC)*, 1989.

[2] M. Bawa, A. Gionis, H. Garcia-Molina, and R. Motwani. The price of validity in dynamic networks. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 515–526, 2004.

[3] S. Datta, C. Giannella, and H. Kargupta. K-means clustering over peer-to-peer networks. In *Proceedings of the 8th International Workshop on High Performance and Distributed Mining (HPDM'05). In conjunction with the SIAM International Conference on Data Mining*, 2005.

[4] S. Kamvar, M. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th international conference on World Wide Web (WWW)*, pages 640 – 651, 2003.

[5] D. Kempe, A. Dobra, and J. Gehrke. Computing aggregate information using gossip. In *Proceedings of the 44th IEEE Symposium on Foundations of Computer Science (FoCS)*, pages 482–491, 2003.

[6] W. Kowalczyk, M. Jelasity, and A. Eiben. Towards data mining in large and fully distributed peer-to-peer overlay networks. In *Proceedings of BNAIC'03*, pages 203–210, 2003.

[7] D. Krivitski, A. Schuster, and R. Wolff. A local facility location algorithm for sensor networks. In *Proc. of DCOSS'05 (to appear)*, June-July 2005.

[8] M. Mehyar, D. Spanos, J. Pongsajapan, S. Low, and R. Murray. Distributed averaging on a peer-to-peer network. In *Proceedings of IEEE Conference on Decision and Control*, 2005.

[9] R. Wolff, K. Bhaduri, and H. Kargupta. Local l2 thresholding based data mining in peer-to-peer systems. Technical Report TR-CS-05-11, Department of Computer Science, UMBC, October 2005.

[10] R. Wolff and A. Schuster. Association rule mining in peer-to-peer systems. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 34(6):2426–2438, December 2004.

[11] F. Zhao and L. Guibas. *Wireless Sensor Networks: An Information Processing Approach*. Morgan Kaufmann, 2004.