Distributed Identification of Top-l Inner Product Elements and its Application in a Peer-to-Peer Network *

Kanishka Bhaduri, Kamalika Das, Kun Liu, Hillol Kargupta
CSEE Dept, UMBC
{kanishk1, kdas1, kunliu1, hillol}@cs.umbc.edu

Abstract

Inner product computation is an important primitive used in many techniques for feature dependency detection, distance computation, clustering and correlation computation among others. Recently, peer-to-peer networks are getting increasing attention in various applications involving distributed file sharing, sensor networks, and mobile ad hoc networks. Efficient identification of top few inner product entries from the entire inner product matrix of features in a distributed peer-to-peer network is a challenging problem since centralizing the data from all the nodes in a synchronous, communication efficient manner may not be an option. This paper deals with the problem of identifying significant inner products among features in a peer-to-peer environment where different nodes observe a different set of data. It uses an ordinal framework to develop probabilistic algorithms to find top-l elements in the inner product matrix. These l inner product entries are important in making crucial decisions about dependency or relatedness between feature pairs, important for a number of data mining applications. In this paper we present experimental results demonstrating accurate and scalable performance of this algorithm for large peer-to-peer networks and also describe a real-world application for this algorithm.

1 Introduction

Inner product is an important primitive for many machine learning and data mining applications such as classifier learning and clustering. These are used for various kinds of tasks such as information retrieval from the web, text processing, predictive modeling and the like. Traditional data mining techniques assume that all data is available at a central location. However there exist situations in which the data is inherently distributed over a large, dynamic network containing no special servers or clients, for example, peer-to-peer (p2p) networks. Some of the most important characteristics of p2p networks are decentralized control, asynchronous communication paradigm, dynamic topology and the like. A distributed scenario can be of two types: (1) homogeneous – where only a fraction of each feature is observed at every site or (2) heterogeneous – where only some of the features are observed at each site. For either scenario centralizing all the data in order to build a global model is not an appropriate solution due to the high cost of centralizing and storage requirement at the central node. Therefore, distributed algorithms are required to solve most data mining problems in p2p networks. In general, a distributed algorithm in this setting should (1) not require global synchronization, (2) be communication efficient, and (3) be resilient to moderate changes in the network topology.

Here we explore the homogeneous scenario where each node has information about only a subset of data tuples for every feature in the data set. If these features of the data set can be represented as vectors, then an inner product matrix is a matrix where each entry is the inner product between a pair of features. However, since every site only has partial information about a feature, the top-l elements of this matrix need not be the global set of top-l inner product elements. We have proposed an algorithm, which can identify the global set of top-l elements in the entire network by estimating the global inner product values of feature pairs selected by random sampling in a communication efficient manner. To the bets of our knowledge, there does not exist any algorithm in the literature that can correctly identify the top few elements in a network without communicating with every single node. As a result we, have an algorithm with bounded message complexity. We have also provided error bounds for the quality of our estimate using the theories from ordinal statistics.

The rest of the paper is organized as follows. In Section 2 we provide a motivation for the study

*A preliminary version of this work has been submitted to CIKM 2006.

of inner product computation in a distributed (p2p) setting. Section 3 points out some of the existing techniques that can be used to solve this problem and discusses their shortcomings. In Section 4 we describe our approach in details while Section 5 discusses the theory behind the approach that we have adopted for solving the distributed inner product computation problem. Section 6 analyzes the performance of our algorithm with respect to the accuracy achieved and the communication complexity. In Section 8 we discuss the experimental results while Section 9 describes an application of this framework in a real life p2p network scenario. In Section 10 we conclude the paper with some discussion on the future work.

2 Motivation

The curse of dimensionality makes data analysis significantly difficult [1] [2]. It even dictates the cost of centralization, since the latter increases with increasing dimension. There exists a number of techniques such as principal components analysis (PCA), singular value decomposition (SVD), etc. for dimensionality reduction in the centralized setting. These can be applied to reduce the dimensionality and then the inner product entries of this reduced space can be centralized to find the significant entries in the new space. However these off-the-shelf techniques do not scale well in large scale peer-to-peer networks with respect to communication, computation and storage. In many cases such as the Internet, distributed file sharing networks (*e.g.* Gnutella, Kazaa, Bit Torrents), local area networks, sensor networks and peer-to-peer networks the data is inherently distributed. Thus there exists great scope for development of distributed algorithms for performing a wide variety of tasks that are otherwise quite easily solvable, in a distributed scenario.

One such task is inner product computation. Inner product computation is a very powerful primitive in machine learning and data mining that can be used for computing Euclidean distance (clustering), information gain (classifiers, bayes net) and correlation between vectors. Inner product can also be used for computing the angle between two vectors. Now, if we consider each feature as a column vector, then the inner product between two feature vectors measures the "similarity" between them in terms of the angle between them. In other words, higher the inner product value, more is the similarity between two features and vice versa.

In a homogeneously distributed setting a naïve method for inner product computation would involve centralizing all the data tuples of every feature to a central site and then computing the inner product matrix. However, this is not a feasible solution for a large-scale distributed system such as a p2p network because of problems stated earlier. So we propose an order statistics-based approach that enables a node to identify the top-l elements in the inner product matrix. Simply speaking, our statistical tool guarantees that if the inner product between any two features is above a threshold, then it can be concluded with a certain confidence level that those two features are in the top $1-p$ percentile of the entire population of inner product entries.

In the next section we present some related work in the area of distributed inner product computation and identifying the top elements of a population.

3 Related Work

This section presents a brief overview of the literature on distributed inner product computation (Section 3.1) and top-k elements identification (Section 3.2) from a population using different techniques. The naïve approach to this problem is to centralize all the data and then find the top elements. The obvious deficiency of this approach is the bandwidth requirement for communication and storage requirement at the central node. Below we present a few of the techniques that have been proposed in the literature to deal with similar problems in a distributed setting.

Peer-to-peer data mining is a relatively new field. Clustering in p2p networks [3] [4], association rule mining [5], monitoring L2 norm [6] are some of the recent work in this area.

3.1 Distributed Inner Product Computation

Data transformations such as Fourier Transforms, Wavelet Transforms and the like can be used for doing inner product computation in an efficient manner when the data is distributed among a large number of peers. Since these transforms are orthogonal, they preserve the inner product between any two given vectors exactly. In other words, the inner product between any two vectors in the original space is the same as the inner product between those two transformed vectors in the new space. In general, however, for every vector of dimension d, there would be d coefficients in the new space. Thus there would be no savings with respect to computation or communication. We can, however, consider only a few ($k < d$) significant coefficients and still be able to get the inner product with high accuracy. Using such a technique, therefore, we can do a distributed inner product computation by simply communicating these compact coefficients instead of the original data.

Random Projection can also be used for computing inner product between vectors in a projected space. Random Projections have been used extensively in the

literature for dimensionality reduction and several other tasks (for example [7] and [8]). It preserves inner product on average (when using large random matrices) and the variance of the process is also bounded. Hence this method serves as a natural candidate for inner product computation using low communication overhead as proposed by Giannella et al. [9]. The major disadvantage of this method is that every peer in the network needs to agree on the random seed from which to generate the random matrix. This implies an expensive round of synchronization and flooding of the network.

3.2 Identifying top-k items Several techniques exist in the literature for monitoring the top elements of a population. These techniques can potentially be used for identifying the top interacting features in a static environment as well. Wolff and Schuster [5] present a *local* algorithm that can be used for monitoring the entries in a certain percentile of the population. In the paper, the authors describe a majority voting algorithm, where 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: each peer has a real number x_i and the collective goal is to decide whether $Avg(x_i) > \tau$. By a proper choice of τ (*e.g.* p^{th} percentile of the population to monitor) this technique can be used to find the all the entries of the inner product matrix that belong to the p^{th} percentile of the population. The major disadvantage of this problem is the communication complexity – a separate majority voting problem needs to be invoked for every inner product entry and thus the system will not scale well for large number of features.

Distributed Top-K Monitoring by Babcock and Olston [10] presents a way of monitoring the answers to continuous queries over data streams produced at physically distributed locations. While a naïve solution to this problem is to centralize all the data, it is not feasible in a typical streaming scenario due to the extremely high rate of data arrival. In this paper, the authors propose an incremental technique to deal with this problem. This paper assumes a central node and the top-k set is always determined by the central node. The basic algorithm is as follows. The coordinator node finds the answers to the top-k queries and distributes it to all the monitor agents. Along with it, the central node also distributes a set of constraints. These constraints allow a monitor node to validate if the current top-k set matches with what it finds from the local stream. If the validation results in true, nothing needs to be done else, the monitor agent sends an alert to the co-coordinator node. The coordinator node re-computes the top-k set based on the current data distribution and sends out both the new top-k and new set of constraints to be validated for each monitor agent. Since the paper assumes that there is a central node, this technique is not directly applicable in many inherently distributed environments *e.g.* Mobile ah-hoc networks, vehicular ad hoc networks and the like.

In the context of information retrieval, several techniques exist in the literature for top-k object identification. Balke et al. [11] propose a super peer approach for finding the top objects. The top queries are handled by the super peers and any other peer in the network can contact these super peers to get the answers to these queries. The paper also discusses ways to select these super peers so that any peer can find its closest super peer efficiently. There are also techniques which explore the retrieval algorithms taking into account the relative rankings of objects. Many of these algorithms depend on gossip based techniques for spreading the ranks of its objects [12].

In the next section we present our distributed algorithm to identify the significant inner product entries.

4 P2P Computation of Significant Inner Product Entries

In this section we describe our algorithm for doing distributed selection of top-l entries when there are k elements in the top $1-p$ percentile of the population ($l < k$). For this, we first introduce some of the notations used throughout the rest of this paper.

4.1 Notations We deal with the homogeneously partitioned data where all features are observed at every node, but each site only has a subset of the total tuples in the overall data set. We assume that there are S nodes (peers) in the network $P_1, P_2, ..., P_S$. The global set of features, which is common for all the peers, is denoted by $(a_1, a_2, ..., a_c)$. The local data set for peer P_i is denoted by $\mathbb{D}_i$ having r_i rows and c columns. Therefore, the union of the data sets of all the peers is $\cup_{i=1}^{S} \mathbb{D}_i = \mathbb{D}$. The inner product between two k-dimensional vectors $\mathbf{x}$ and $\mathbf{y}$ is defined as $\mathbf{x}.\mathbf{y} = \sum_{i=1}^{k} x_i y_i$. Now, if each feature is represented as a vector, we can define the inner products between the features in a similar fashion. Thus, the global inner product matrix A is a $c \times c$ matrix where A_{ij} corresponds to the inner product between the i^{th} and j^{th} feature. Since this matrix is symmetric about the diagonal and the diagonal entries are the inner product of the feature vectors with themselves, we consider only the upper half of this matrix. Thus we have $\frac{c^2-c}{2}$ distinct entries in the inner product matrix.

Henceforth we refer to A as the upper triangular inner product matrix. We also assume that the entries of this matrix has been indexed with a single number in a consistent fashion across all peers – *i.e.* $\{1,1\} \rightarrow \{1\}, \{1,2\} \rightarrow \{2\}, ..., \{c,1\} \rightarrow \{\frac{c^2-c}{2}\}$,. The entries of matrix A can be written as $A[1], A[2], ..., A[\frac{c^2-c}{2}]$. Now matrix A is distributed among all the peers, and let the share of peer P_k be denoted by I_k. Let the d^{th} entry of this matrix be denoted by $I_k[d]$, $\forall d \in \{1, ..., \frac{c^2-c}{2}\}$. Because of the decomposable property of inner products, (refer to Section 5.1 for details), $\sum_{k=1}^{S} I_k[d] = A[d]$.

4.2 Approach The process is started by any node in the network that decides to find the top-l entries in the distributed inner product matrix. We call this node the initiator node. Based on the desired level of confidence (q) and percentile (p) of the population to monitor, our algorithm needs to know three parameters – (1) number of ordinal samples to collect or n, (2) the number of peers to visit for estimating each sample or m, and (3) n indices of the inner product matrix corresponding to the n samples to collect. In this section we will assume that the initiator node already has the values of these parameters. It launches $m \times n$ random walks and after all these walks terminate, the samples are sent back to the initiator node. The initiator then needs to add all the samples having the same index. It then orders the n samples and the highest one is known as the *threshold*. Any inner product value greater than this threshold is expected to be in the top $1-p$ percentile of the population with the chosen confidence. Hence the approach consists of the following tasks: *sample size computation, sample collection, threshold detection* and finally *top-l inner product elements identification.*

- **Sample Size Computation:**

  The initiator P_i first selects a confidence level q and the order of population percentile p it would tolerate. Based on the algorithm described in Section 5.2, the initiator calculates the number of samples required to compute the threshold such that any inner product that is greater than this threshold is among the top $(1-p)$ percentile of the population of inner products. Let us denote this sample size as n. It also randomly generates n indices (each between $1 \leq i \leq \frac{c^2-c}{2}$) which will be used for sampling the inner product entries. Since the data is distributed among all the peers finding a global entry of the inner product matrix requires visiting all the peers and adding up the values obtained from each peer. If we choose to visit a subset m of S nodes, we can estimate the value of this inner product entry. We have used Central Limit Theorem (CLT) in order to derive the value of m in Section 5.3. After this step, the initiator peer knows the value of m, n and the actual indices of the inner product entries to be sampled.

- **Sample Collection:**

  Given the sample size of n and the number of peers to visit m, the initiator invokes $m \times n$ random walks using the protocols described in Section 5.4 to choose independent samples from the network. Since estimating one single inner product entry requires sampling m peers for the same indexed entry, each random walk carries with it the index number of the element to be sampled. Also each random walk carries the IP address and port number of the initiator node so that the terminal node of a random walk can send its inner product entry directly to the initiator node. At the end of these random walks P_i has $m \times n$ samples where there are n different indices and m inner product values for every index of the inner product.

- **Threshold Detection:**

  Once the initiator node gets all the samples, its next task is to identify the threshold. Since inner product is decomposable, for every index i, peer P_i sums up the all the m entries corresponding to the same index i. It then finds the largest of this set of inner product entries and this is the threshold.

- **Top-l inner product elements identification:**

  The above technique would give the peer a way to identify one of the items in the top-k, where there are k elements in the top $(1-p)$ percentile of the population. We can extend this to find some l of the top-k elements ($l < k$). All that a peer P_i needs to do is to launch $n \times m \times l$ random walks. Now after aggregating the results we have nl elements and for every n element we can find a threshold. Thus we will have l thresholds and each of these would belong to the top-k elements.

OrdSamp (Algorithm 4.2.1) presents the sample collection technique for a single random walk using the ordinal framework. The initiator sends a token (initialized to a value equal to the length of the random walk λ), its IP address, port number (*InitiatorNodeNum*) and the index of the element (*SampleIndex*) to sample for this random walk. When a node gets this token, it decrements its value by 1. If the value of the token becomes 0, the inner product entry indexed by *SampleIndex* is selected from the local data set and sent back to the initiator node.

Algorithm 4.2.1 Distributed selection of samples (*OrdSamp*)

Input of peer P_i: $\mathbb{D}_i$ - the local database, N_i - set of immediate neighbors of P_i, its local transition matrix p_{ij}

Output of peer P_i: Sends the sample if the random walk terminates at this peer

On receiving a message ($Token$):

$Token = Token$ - 1

Fetch $SampleIndex$

Fetch $InitiatorNodeNum$

IP Address and Port number of the initiator node

if $Token = 0$ **then**

Pick the element whose index is $SampleIndex$ from $\mathbb{D}_i$.

Send $SampleIndex$ to the $InitiatorNodeNum$.

Wait for new $Token$ messages for other random walks

else

Send $SampleIndex$, $InitiatorNodeNum$ to a neighbor selected according to the transition matrix

end if

5 Building Blocks

This section elaborates on some building blocks that are necessary to understand the distributed feature selection algorithm.

5.1 Decomposable Inner Product Computation

Let $\mathbf{x}$ and $\mathbf{y}$ be two vectors. Now inner product between two vectors is defined as:

$$\mathbf{x}.\mathbf{y} = \sum_{(x,y)\in\mathbb{D}} xy$$

Now according to our scenario, $\mathbb{D}$ is divided into S peers and the contribution of peer P_i is $\mathbb{D}_i$. We can therefore, write the inner product of those two vectors as:

$$\sum_{i=1}^{S} \sum_{(x,y)\in\mathbb{D}_i} xy = \sum_{i=1}^{S} k_i$$

where k_i is the contribution of the P_i^{th} peer towards the inner product between $\mathbf{x}$ and $\mathbf{y}$. Sampling from all the peers is infeasible especially in large systems and hence we resort to sampling from a subset of peers from S in order to estimate $\mathbf{x}.\mathbf{y}$.

5.2 Ordinal Approximation

Given a data set horizontally partitioned among peers we want to find some l of the top-k entries which are in the top $1-p$ percentile of the population. A trivial approach to this problem would be to collect the entire data set from all peers and compare all the pairwise inner products among the features. This simple approach, however, does not work in a large-scale distributed p2p environment because the network state is not stable with frequent nodes arrivals and departures, and the overhead of communication would be extremely high. Theories from order statistics, however, could relieve us from this burden by considering only a small set of samples from the entire population and producing a solution with probabilistic performance guarantees. Order statistics has been applied in a number of different fields such as classifier learning [13], genetic algorithms [14], sensor networks [15], and discrete event optimization [16]. The following part of this section discusses application of order statistics in our framework.

Let $\mathbf{X}$ be a continuous random variable with a strictly increasing cumulative density function (CDF) $F_{\mathbf{X}}(x)$ [1]. Let ξ_p be the population percentile of order p, *i.e.* $F_{\mathbf{X}}(\xi_p) = Pr\{x \leq \xi_p\} = p$, *e.g.* $\xi_{0.5}$ is called the median of the distribution. Suppose we take n independent samples from the given population $\mathbf{X}$ and write the ordered samples as $x_1 < x_2 < \cdots < x_n$. We are interested in computing the value of n that guarantees

$$Pr\{x_n > \xi_p\} > q.$$

LEMMA 5.1. *[16] Let $x_1, x_2, ..., x_n$ be n i.i.d. samples drawn from an underlying distribution. They are arranged such that $x_1 < x_2 < ... < x_n$. Then $P(x_n > \xi_p) = 1 - p^n$, where ξ_p is the p^{th} percentile of the population.*

Proof.

$$\begin{aligned} P(x_n > \xi_p) &= 1 - P(x_n \leq \xi_p) \\ &= 1 - F_n(\xi_p) \\ &= 1 - p^n \end{aligned}$$

Now if the above probability is bounded by a confidence q, we can rewrite the above equation as

$$1 - p^n > q \Rightarrow n \geq \left\lceil \frac{\log(1-q)}{\log(p)} \right\rceil. \quad (5.1)$$

For example, for $q = 0.95$ and $p = 0.80$, the value of n obtained from the above expression is 14. That is, if

[1] Following a common convention, we use upper-case bold letters to refer to random variables, lower-case regular letters to refer to their actual observed values, and lower-case bold letters to refer to vectors.

we took 14 independent samples from any distribution, we can be 95% confident that 80% of the population would below the largest order statistic x_{14}. In other words, any sample with value greater or equal to x_{14} would be in the top 20 percentile of the population with 95% confidence. The smaller the p is, the smaller the n. For detailed treatment of this subject we refer the reader to David's book [17].

5.3 Cardinal Approximation In order to derive bounds on the number of peers to sample (m) we have used the Central Limit Theorem (CLT). CLT tells us that if we take m independent samples from any population, the mean of the samples approximately follows a Normal Distribution $(\mu, \frac{\sigma}{\sqrt{m}})$, where μ and σ are the mean and standard deviation of the population. Lemma 5.2 gives a expression for determining the value of m using CLT.

LEMMA 5.2. *Let $\mathbf{X_1}, \mathbf{X_2}, ..., \mathbf{X_m}$ be m independent samples drawn from a population with mean μ and standard deviation σ. Let $\bar{\mathbf{X}}$ denote the mean of these samples. Then the value of m is bounded from below by $m \geq \frac{\Phi^{-1}(1-q)\sigma}{\epsilon}$, where Φ is cdf of the standard normal variate, q is the confidence and ϵ denotes the probability of deviation of $\bar{\mathbf{X}}$ from μ.*

Proof. Form the CLT we know that $\bar{\mathbf{X}}$ approximately follows a Normal distribution with mean μ and standard deviation $\frac{\sigma}{\sqrt{m}}$. We can write,

$$\begin{aligned} P(\bar{\mathbf{X}} - \mu \geq \epsilon) &= P(\frac{\bar{\mathbf{X}} - \mu}{\frac{\sigma}{\sqrt{m}}}) \geq \frac{\epsilon}{\frac{\sigma}{\sqrt{m}}} \\ &= P(\mathbf{Z} \geq \frac{\epsilon\sqrt{m}}{\sigma}) \\ &= 1 - P(\mathbf{Z} \leq \frac{\epsilon\sqrt{m}}{\sigma}) \\ &= 1 - \Phi(\frac{\epsilon\sqrt{m}}{\sigma}) \end{aligned} \tag{5.2}$$

Now if Equation 5.2 is bounded above by q, we get

$$\begin{aligned} 1 - \Phi(\frac{\epsilon\sqrt{m}}{\sigma}) &\leq q \\ \implies m &\geq \frac{\Phi^{-1}(1-q)\sigma}{\epsilon} \end{aligned}$$

5.4 Random Sampling Random sampling in the networks is a prerequisite to the estimation of population percentile. It can be performed by modeling the network as an undirected graph with transition probability on each edge, and defining a corresponding Markov chain. Random walks of prescribed length on this graph produce a stationary state probability vector and the corresponding random sample. The simplest random walk algorithm chooses an outgoing edge at every node with equal probability, *e.g.* if a node has degree five, each of the edges is traversed with a probability 0.2. However, it can be shown that this approach does not yield a uniform sample of the network unless the degrees of all nodes are equal (see [18] for example). Since typical large-scale Peer-to-Peer network tends to have non-uniform degree distribution, this approach will generate a biased sample in most practical scenarios. Figure 1(a) shows the non-uniform selection probability using a graph of 5000 nodes.

Fortunately, the elegant Metropolis-Hastings algorithm [19] [20] implies a simple way to modify the transition probability so that it leads to a uniform stationary state distribution, and therefore results in uniform sample. In this paper, we implement an adaptation of this classical algorithm. Next we briefly introduce the Metropolis-Hastings algorithm for random walk.

Let $G(V, E)$ be a connected undirected graph with $|V| = n$ nodes and $|E| = m$ edges. Let d_i denote the degree of a node i, $1 \leq i \leq n$. The set of neighbors of node i is given by $\Gamma(i)$ where $\forall j \in \Gamma(i)$, edge $(i, j) \in E$. Let $P = \{p_{ij}\}$ represent the $n \times n$ transition probability matrix, where p_{ij} is the probability of walking from node i to node j in one message hop ($0 \leq p_{ij} \leq 1$ and $\sum_j p_{ij} = 1$). Algorithm 5.4.1 gives the basic protocol for doing a random walk using the Metropolis Hastings Algorithm.

Algorithm 5.4.1 Metropolis-Hastings (MH)

Input of peer P_i: Its degree d_i

Output of peer P_i: A row of transition matrix p_{ij}

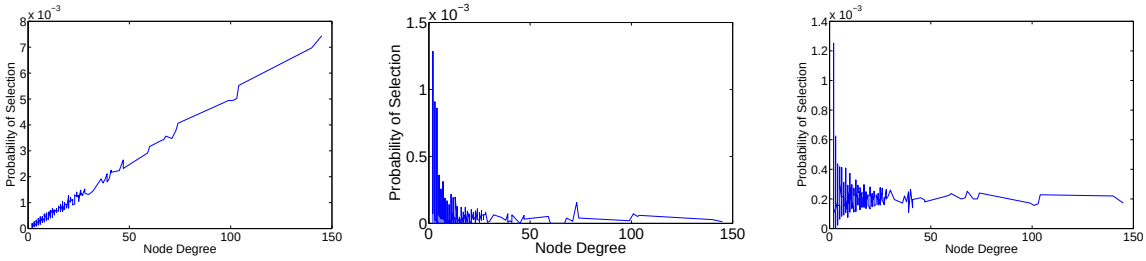
On Initialization: P_i sends out a *Degree* message to all $P_j \in \Gamma(P_i)$

On receiving a message (*Degree*): If it has received the degree information from all $P_j \in \Gamma(P_i)$ it can compute p_{ij} as follows:

$$p_{ij} = \begin{cases} 1/\max(d_i, d_j) & \text{if } i \neq j \text{ and } j \in \Gamma(i) \\ 1 - \sum_{j \in \Gamma(i)} p_{ij} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Termination: Once the degree has been calculated the peer can terminate the transition matrix calculation

This algorithm generates a symmetric transition probability matrix and is proved to produce uniform sampling via random walk [21]. Lovász [18] showed that the length of random walk (λ) necessary to reach to stationary state is of the order of $O(\log n)$. Empirical re-



(a) Simple Random Walk (b) Degree Balanced Random Walk (c) Metropolis Hastings

Figure 1: Performance of three different random walks on a power law topology of 5000 Nodes.

sults show that when the length of walk is $10 \times \log n$, this algorithm converges to uniform distribution. The network size S could be estimated using the localized estimation scheme proposed by [22]. Figure 1(c) shows the probability of selection using the Metropolis-Hastings algorithm over a simulated network with 5000 nodes. As can be easily seen, the probability of selection is uniform even for varying degree distribution. We also compared this technique with another random walk technique proposed by Orponen and Schaeffer [23] known as the Degree Balanced Random Walk (DRW). Experiments with this random walk technique (as shown in Figure 1(b) shows that the probability is uniform in this case as well. The major problem with this technique is that it requires a long length to come to a stationary distribution. In this paper, therefore, we have used the MH algorithm for collecting samples from the network.

6 Performance Analysis

In this section we analyze the error that occurs in our distributed algorithm and the message complexity.

6.1 Error Bound In our distributed algorithm there are two sources of error – (1) error due to ordinal sampling and (2) due to cardinal sampling. Let $\tilde{\mathbf{A}}_\mathbf{1}, \tilde{\mathbf{A}}_\mathbf{2}, ..., \tilde{\mathbf{A}}_\mathbf{n}$ denote the samples as found by the distributed algorithm (the subscripts correspond to the indexing scheme defined in 4.1). Note that each of these $\tilde{\mathbf{A}}_\mathbf{d}$'s are estimated by aggregating the values of the d-th indexed entry of the inner product matrix from m peers, where the value of the d^{th} indexed entry for the i^{th} peer is given by $\mathbf{I_i[d]}$ and $\bar{\mathbf{I}}[\mathbf{d}] = \frac{\sum_{i=1}^{m} \mathbf{I_i[d]}}{m}$ denotes the mean of the estimates for a fixed d. Lemma 6.1 derives the probability that the threshold *i.e.* $\tilde{\mathbf{A}}_\mathbf{n}$ belongs to the p^{th} percentile of the population.

LEMMA 6.1. *Let $\tilde{\mathbf{A}}_\mathbf{1}, \tilde{\mathbf{A}}_\mathbf{2}, ..., \tilde{\mathbf{A}}_\mathbf{n}$ be the n samples returned by the distributed algorithm. They are ordered such that $\tilde{\mathbf{A}}_\mathbf{1} < \tilde{\mathbf{A}}_\mathbf{2} < ... < \tilde{\mathbf{A}}_\mathbf{n}$. Then, $P(\tilde{\mathbf{A}}_\mathbf{n} > \xi_p) = 1 - \prod_{d=1}^{n} \Phi((\frac{\xi_p}{m} - \mu_d)\frac{\sqrt{m}}{\sigma_d})$, where μ_d and σ_d are the mean and standard deviation of the feature of the population corresponding to $\tilde{\mathbf{A}}_\mathbf{d}$, ξ_p is the population percentile of order p and $\Phi(.)$ is the area under the standard normal curve.*

Proof.

$$
\begin{aligned}
P(\tilde{\mathbf{A}}_\mathbf{n} > \xi_p) &= 1 - P(\tilde{\mathbf{A}}_\mathbf{n} \leq \xi_p) \\
&= 1 - \prod_{d=1}^{n} P(\tilde{\mathbf{A}}_\mathbf{d} \leq \xi_p) \\
&= 1 - \prod_{d=1}^{n} P(\sum_{i=1}^{m} \mathbf{I_i[d]} \leq \xi_p) \\
&= 1 - \prod_{d=1}^{n} P(\frac{\sum_{i=1}^{m} \mathbf{I_i[d]}}{m} \leq \frac{\xi_p}{m}) \\
&= 1 - \prod_{d=1}^{n} P(\bar{\mathbf{I}}[\mathbf{d}] \leq \frac{\xi_p}{m}) \\
&= 1 - \prod_{d=1}^{n} P(\bar{\mathbf{I}}[\mathbf{d}] \leq \frac{\xi_p}{m}) \\
&= 1 - \prod_{d=1}^{n} P(\frac{\bar{\mathbf{I}}[\mathbf{d}] - \mu_d}{\frac{\sigma_d}{\sqrt{m}}} \leq \frac{\frac{\xi_p}{m} - \mu_d}{\frac{\sigma_d}{\sqrt{m}}}) \\
&= 1 - \prod_{d=1}^{n} P(\mathbf{Z} \leq (\frac{\xi_p}{m} - \mu_d)\frac{\sqrt{m}}{\sigma_d}) \\
&= 1 - \prod_{d=1}^{n} \Phi((\frac{\xi_p}{m} - \mu_d)\frac{\sqrt{m}}{\sigma_d})
\end{aligned}
$$

Some explanations regarding the derivation are as follows. Step 2 follows directly from step 1. Now since $\tilde{\mathbf{A}}_\mathbf{d}$ is a sum of all the elements obtained by visiting m peers, we must have $\tilde{\mathbf{A}}_\mathbf{d} = \sum_{i=1}^{m} \mathbf{I_i[d]}\ \forall\ d$. Finally, since $\sum_{i=1}^{m} \mathbf{I_i[d]}$ is a sum of random variables we have used CLT to derive the final expression.

Hence the probability of error is $\prod_{d=1}^{n} \Phi((\frac{\xi_p}{m} - \mu_d)\frac{\sqrt{m}}{\sigma_d})$. This shows that as n increases, the error

decreases since each term of the product is $\Phi(.)$, which is the area under a unit Normal variable and is less than or equal to 1. Also as m increases, the expression inside Φ decreases and thus the overall probability of error decreases. For a special case in which all the μ_d's and σ_d's are equal to say μ and σ, the error becomes $\Phi((\frac{\xi_p}{m} - \mu)\frac{\sqrt{m}}{\sigma})^n$ – hence as n increases, the error decreases exponentially.

7 Message Complexity

The distributed algorithm that we just described launches $n \times m \times l$ parallel random walks each of length λ such that each random walk will return a single element. The coordinator node can then aggregate these samples, and come up with l thresholds. We will use this model to analyze the message complexity.

For each such a random walk, the initiator node needs to send the following four information in the message:

1. Token Number - Integer 32 bits
2. Index of the inner product entry to sample - Integer 32 bits
3. IP Address - Integer 32 bits
4. Port Number - Integer 32 bits

The message complexity for this step is : $128m \times n \times l \times \lambda = 128mnl\lambda$ bits. Now from Section 5.2, since $n = \left\lceil \frac{\log(1-q)}{\log(p)} \right\rceil$, the message complexity can be rewritten as, $128ml\frac{\log(1-q)}{\log(p)}$.

Once the random walk ends, the terminal node simply needs to send the sampled element back to the initiator node. This would need 64 bits assuming that each entry of the inner product matrix can be represented as a single double number. Thus, the overall message complexity for the entire step is: $128mnl\lambda + 64nml = \mathrm{O}(mnl\lambda)$ bits.

Note that this expression is independent of the number of features c. Hence we expect the algorithm to be scalable with respect to the number of features.

Now, considering the centralized algorithm, if each peer has a dataset $r_i \times c$, then the total message complexity for the centralized scheme can be written as : $64r_i \times c \times S = \mathrm{O}(r_i cS)$ bits.

8 Experiments and Performance Evaluation

In this section, we study the performance of the proposed distributed feature selection algorithm.

8.1 Data Generation The experimental data was synthetically generated. Each entry of the data matrix was generated randomly from a uniform distribution. Each column was generated from a different range of the uniform distribution in order to have variation in the data. The centralized data set was then uniformly split (so that each peer has the same number of tuples) among all the peers to simulate a horizontally partitioned scenario. Unless otherwise specified, each peer had 500 tuples and 500 features. The reason for the synthetic data experiments was to have more control on the parameters of the algorithm.

8.2 Network Topology and Simulator Our network topology was generated using the ASWaxman Model from BRITE [2], a universal topology generator. The generator initially assigns node degrees from a power-law distribution and then proceeds to interconnect the nodes using Waxman's probability model. Simply speaking, in this model, the probability of two nodes (i and j) being connected is given by $P(i,j) = \alpha e^{-d(i,j)/\beta L}$, where $0 < \alpha, \beta < 1$ (fixed at 0.15 and 0.2 respectively in our experiments), $d(i,j)$ is the Euclidean distance from node i to node j, and L is the maximum distance between any two nodes. Power-law random graph is often used in the literature to model large non-uniform network topologies. It is believe that p2p networks conform to such power law topologies [24]. We use the Distributed Data Mining Toolkit (DDMT) [3] developed by the DIADIC research lab at UMBC to simulate the distributed computing environment. This toolkit is build upon LEAP (Light Extensible Agent Platform) [4], which itself is an extension of JADE (Java Agent DEvelopment Framework) [5], a multi-agent systems platform. All of our algorithms were implemented in Java JDK 1.5, and the experiments were conducted on a dual-processor workstation running Windows XP with 3.00GHz and 2.99GHz Xeon CPUs and 3.00GB RAM.

8.3 Performance We have performed experiments to study the applicability of the ordinal approximation theories in our distributed environment by comparing the results returned by the centralized algorithm. By a centralized algorithm we mean centralizing the entire data set of all peers and running the ordinal approximation on this data set. We have reported four sets of experiments - (1) the quality of our estimation when monitoring increasing percentile of population, (2) the scalability of our algorithm, (3) the effect of increasing the sampling on the error, and (4) effect of finding l

[2] http://www.cs.bu.edu/brite/
[3] http://www.umbc.edu/ddm/wiki/software/DDMT/
[4] http://leap.crm-paris.com/
[5] http://jade.tilab.com/

elements in the top $1-p$ percentile of the population. We have reported both the accuracy and message complexity (in bits transferred) whenever appropriate. Unless otherwise noted we have the following default values for the different parameters: (1) S=500, (2) c=500, (3) n=15 (which comes from p=15% and q=95%), (4) $m = S/2$, (5) l=1, (6) $\lambda = 10 \times logS$, and (7) r_i (number of data rows for each peer) = 500.

8.3.1 Experiments with different percentile of population In this experiment we compared the accuracy of the distributed algorithm with the centralized one. We have experimented for three different percentile (p) values of 95, 90 and 85 for which the number of samples (n) required are 59, 29 and 19 respectively. We sampled 50% of the peers ($m = S/2$). In the graph in Figure 2 the circular points represent the actual p^{th} percentile of the population, whereas the square and the star represent the threshold for the same confidence and percentile for the centralized and distributed scenario respectively using ordinal approximation. The distance between the red stars and the green circular dots represents the error due to ordinal approximation whereas the difference between the red stars and the blue squares in the graph can be attributed to the cardinal approximation introduced in the distributed environment. We notice that in both the centralized and distributed scenario, the threshold is greater than the actual p^{th} percentile of the population. This means that there will be no false positives in ordinal estimation. Each point in the graph is an average of 100 runs of the experiment. The variance in the different runs is quite low compared to the value itself ($\sim 10^2$).

Figure 3 compares the communication of our algorithm with that of the centralized version for monitoring different percentiles of population (p). Since the number of features $c = 500$ and $S = 500$ remains constant, messages for the centralized experiments for different percentiles is the same. In the distributed scenario, the expression in Section 7 is used for finding the number of messages. In all cases, our algorithm outperforms the centralizing scheme by a factor of approximately 100. Hence in the figure the number of distributed messages is tending to zero in the higher scale.

8.3.2 Experiments with varying number of peers We have also performed a scalability test on the accuracy and message complexity of our algorithm. Figure 4 shows the results when the percentile of the population is fixed at $p = 10\%$. As can be seen from the figure, the threshold detected by both the centralized and distributed experiments using order statistics are greater than the p^{th} percentile of the population cor-

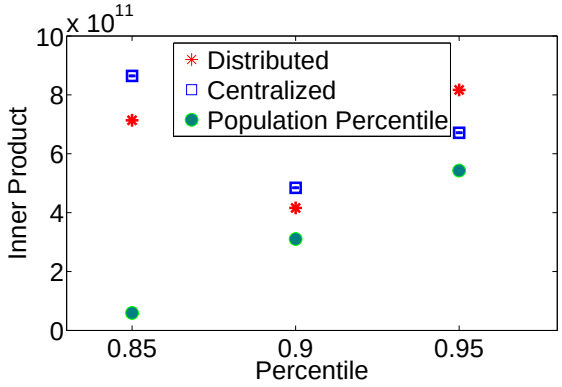


Figure 2: Relative values of the estimated highest order statistic with corresponding values of actual population percentile for varying values of population percentile.

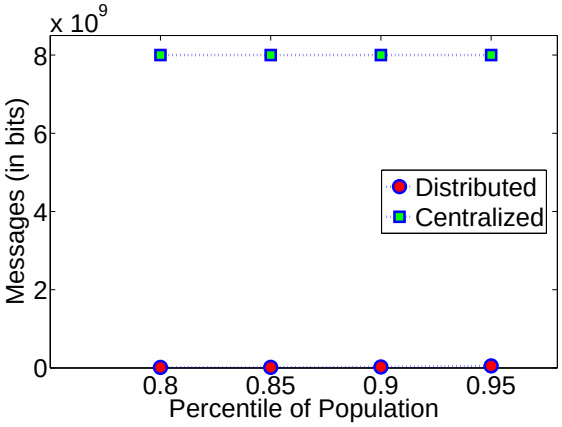


Figure 3: Message complexity with increasing percentile of population.

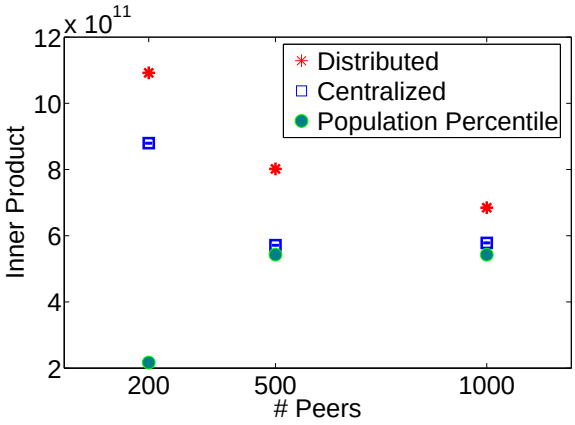


Figure 4: Relative values of the estimated highest order statistic with corresponding values of actual population percentile for varying size of the network.

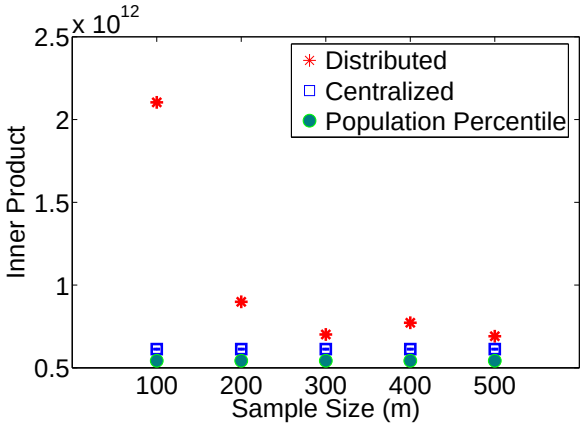


Figure 6: Variation of the threshold with changes in m.

roborating the fact that the algorithm has excellent accuracy. These results are and average of 100 runs of the experiment.

Figures 5(a) and 5(b) presents the number of messages both in the distributed and centralized scenario. Top 15% of the population was monitored in each of these cases. Figure 5(a) presents the scalability with respect to the number of peers (c was set at 500) while in Figure 5(b) the number of features was varied (S was set at 500). In both figures, the number of messages for the distributed algorithm is far less than the centralizing algorithm. Note that in Figure 5(b), the distributed messages were constant since it is invariant of the number of features.

8.3.3 Experiments with increasing m We have also done experiments to test the effect of increasing the sample size m. Figure 6 shows the effect on the threshold selection with increasing of sampling m. The trend is clear - as we increase m, the distributed threshold (red stars) approaches the centralized threshold (blue squares).

8.3.4 Experiments with increasing l In this section we present the message complexity of our distributed algorithm while monitoring l of the top-k elements in the p^{th} percentile of the population. Section 7 shows that message complexity is linearly dependent on l. Figure 7 presents the message complexity compared to the centralized scheme.

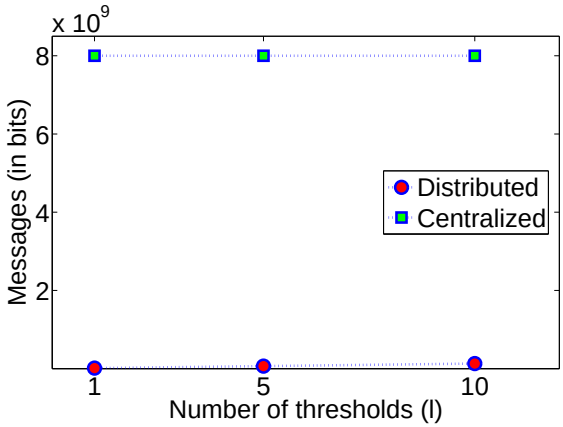
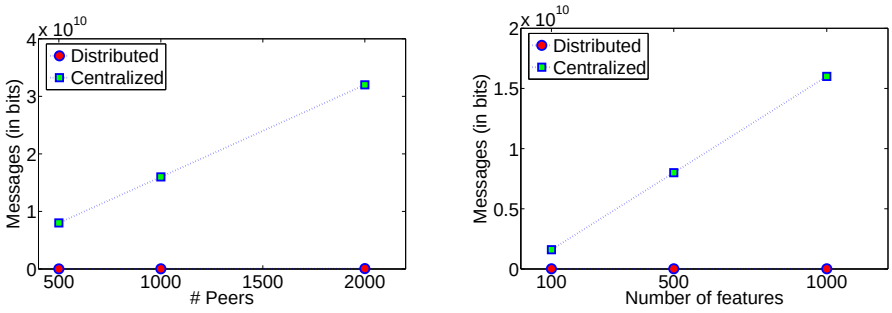


Figure 7: Message complexity with increasing number of thresholds (l).



(a) Variation with Number of Peers (b) Variation with Number of Features

Figure 5: Message complexity with increasing number of peers and features.

9 Application

An interesting application of this technique is client-side web mining. In this section we discuss how we modify our order statistics based top-l item identification technique for this application. Interested readers are referred to the paper by Liu et al. [25] for a detailed discussion on this application.

9.1 Why p2p communities and Client-side web mining ? According to Maslow's theory [26], social motive, which drives people to seek contact with others and to build satisfying relations with them, is one of the most basic needs of human beings. The tendency to have affiliations with others is visible even in virtual environments such as the World Wide Web. Many online communities like Google and Yahoo groups provide the user a place to share knowledge, and to request and offer services. Traditional web mining has spent lots of efforts on the web server side, *e.g.* to analyze the server log. We propose a framework that utilizes the client-side information, namely, the web browsing cache. In many cases the server-side web data is inaccessible to the user who generated the data – so no information about that data is available to the user. On the other hand, using the data at the source machine itself (which we call the client-side data), we can learn several interesting facts about the data and develop several systems (*e.g.* p2p community, recommender systems etc.). We define a Peer-to-Peer community as a collection of nodes in the network that share common interests. Communities can then exchange information for better query routing for example. Compared with other related work, our framework has the following specific features:

- It proposes an order statistics-based algorithm (similar to what has already been discussed) to quantify the similarity between peers over the network. This approach allows a peer to build a community with hierarchical structure.
- Any technique that creates and represents a peer's personal profile as a feature vector can be plugged into our framework.

9.2 Peer Profiles A crucial issue in forming Peer-to-Peer communities is to create peer profiles that accurately reflects a peer's interests. These interests can be either explicitly claimed by a peer, or implicitly discovered from the peer's behaviors. A peer's profile is usually represented by a keyword/concept vector. Trajkova and Gauch proposed techniques to implicitly build ontology-based user profiles by automatically monitoring the user's browsing habits [27]. Figure 8 shows a sample ontology for user profile. We point out that any approach that represents a peer's profile in a feature vector can be used in our framework. In this paper, we use the frequency of the web domains a peer has visited during a period of time as the peer's profile vector. To avoid the uncertainty of ontology matching, we expect all peers to agree on the same ontology defined by a controlled vocabulary. In this paper, this means that all peers agree on a superset of web domain names.

9.3 Similarity Measurement The goal of community formation is to find peers sharing similar interests. However, if we choose a similarity measurement Ω, and simply setup a subjective threshold such that peers with similarities greater than this threshold can be grouped together, we can't represent the essential characteristics of a social community, namely, *hierarchy*. In a social network, a person may have multi-level friends, where the first level might be family members and closest friends, the second level might be some colleagues who are not so familiar with. A person could also have indirect friends from his/her friends' social network. A Peer-to-Peer community from one peer's

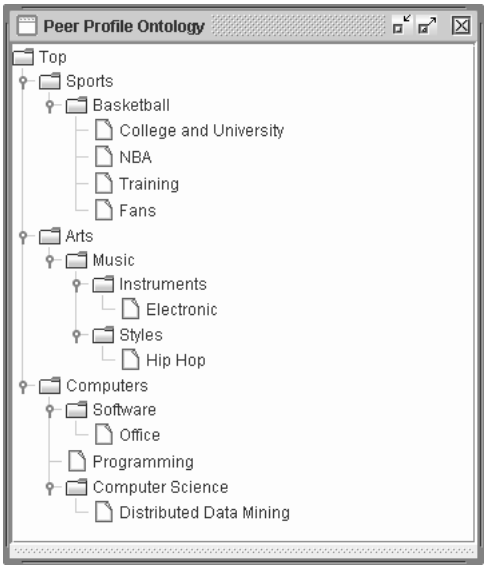


Figure 8: A sample ontology for user profile.

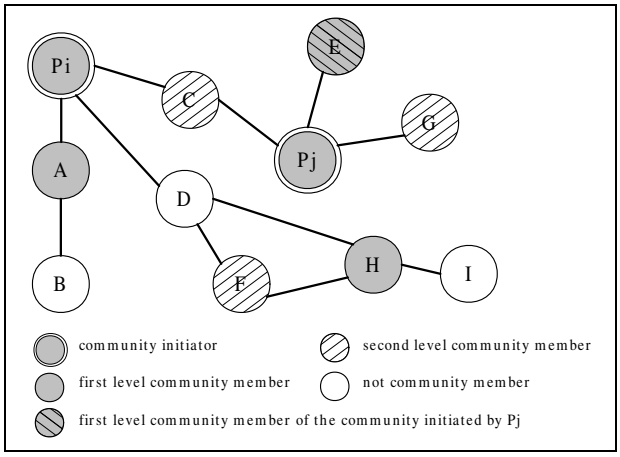


Figure 9: Example of Peer-to-Peer communities.

perspective should also have such kind of hierarchical structure. That is, some peers share more interests with this peer, and some less.

To achieve this goal, we use our order statistics-based approach which enables a peer to know how similar the other peer is to itself. In other words, our statistical measurement guarantees that if the similarity between peer P_i and P_j is above a threshold, P_i can determine with confidence level q that P_j is among the top $(1-p)$ quantile most similar peers of P_i's. As a running example, let us assume there are 5 peers $\{P_1, P_2, P_3, P_4, P_5\}$ in the network, and the similarity measures between P_1 and all other peers are $\{1, 3, 2, 4\}$, respectively, where the higher the value, the higher the similarity. If P_1 knows the similarity between her and P_5 is 4, our approach will enable P_1 to know with high confidence that P_5 is among the top 25% most similar peers of P_1's in the network, without computing all the similarity values.

Now we formally define a Peer-to-Peer community based on our above discussion.

DEFINITION 9.1. [(Ω, p, q)-P2P COMMUNITY] *A (Ω, p, q)-P2P community from peer P_i's view is a collection of peers in the network, denoted by $\mathcal{C}$, such that the similarity measures Ω between P_i and all the members in $\mathcal{C}$ are among the top $(1-p)$ quantile of the population of similarity measures between P_i and all the peers in the network, with confidence level q.*

DEFINITION 9.2. [EXTENDED (Ω, p, q)-P2P COMMUNITY] *An extended (Ω, p, q)-P2P community from peer P_i's view is the union of $\mathcal{C}$ (defined by Definition 9.1) and all the peers from the (Ω, p, q)-P2P community of each member in $\mathcal{C}$.*

These two definitions implicitly capture the hierarchical characteristics of the community. When a peer finds a similar buddy, she could compute the quantile value and determine which area this buddy belongs to. A peer could also specify a p value and only invite those belonging to top $(1-p)$ quantile area to be her community members. The community could be expanded to include members from members's community. For example, in Figure 9, Peer A, P_j, H are the first level members (with larger p) of community initiated by P_i. Peer C, F and G are the second level members (with smaller p) of community. Note that P_j is also a initiator of another community, and it has E as its first level community member. Peer A, P_j, H, E compose an extended p2p community initiated by P_i.

We use the scalar product between two profile vectors to quantify the similarity between two peers. Other similarity metrics such as Euclidean distance can also be applied in our framework without any hurdle. In the next subsection we discuss how the community is actually formed.

9.4 Community Formation Process We address the Peer-to-Peer community formation process under the assumptions that: 1) each peer can be a member of multiple virtual communities; 2) peers interact with each other by submitting or replying queries to determine the potential members of a given community; and 3) there is no super peer as a centralized authority.

The Peer-to-Peer community emerges as a peer, P_i, called community initiator, invokes a community

discovery process which consists of the following tasks: *sample size computation, quantiles estimation, member identification, member notification and acceptance,* and *community expansion.*

- **Sample Size Computation:** The initiator P_i first selects a confidence level q and the order of population quantile p it would tolerate. It can then find the sample size n as discussed in Section 5.2. Note that for this scenario a peer does not need to do a cardinal sampling since we are dealing with a special case of the distributed inner product computation here – when each peer has only one feature vector and not a matrix of local inner product elements.

- **Quantiles Estimation:** Given the sample size n, the initiator invokes n random walks using the protocols described in Section 5.4 to choose independent sample peers in the network. Whenever a new peer P_j is chosen, it replies to P_i with its address and port number, and builds an end-to-end connection with P_i. Then P_i computes the scalar product of its profile vector and P_j's profile vector. After P_i collects all the n scalar products, it finds the largest one as the threshold for quantiles of order p. These two steps are very similar to the first two steps of the algorithm discussed in Section 4.2.

- **Member Identification:** The initiator P_i composes a discovery message containing its address and port number, as well as a time-to-live (TTL) parameter defining the maximum number of hops allowed for the discovery propagation. Then the discovery message is sent to all P_i neighbors. When a peer P_j receives this message, it replies to P_i with its address and port number. P_i then invokes a scalar product computation with P_j to get the similarity value. If TTL ≥ 0, P_j forwards the discovery message to all its neighbors, except for the peer from which the message has been received. Each peer discards duplicate copies of the same discovery message possibly received.

- **Member Invitation and Acceptance:** The initiator P_i evaluates the quality of the discovered peers by comparing the similarity values with its threshold. If the similarity is above the threshold, P_i sends an invitation message to that peer. If the similarity is below the threshold, P_i still could analyze, with the same confidence level, the order of quantile that the peer belongs to; but note that this order will be lower than the preset p. Given this information, P_i can decide whether to send an invitation to a peer with less similarity. For the sake of simplicity, in our experiments, P_i will not send invitations in this circumstance. Once a peer P_j receives an invitation message, it decides whether to accept it or not by replying an acceptance message. Receiving the acceptance message, P_i records P_j in its local cache.

- **Community Expansion:** When a peer P_j accepts the invitation, it replies to the initiator an acceptance message, as well as with the member lists in its local cache. These members are from the p2p community or extended p2p community initiated by P_j. As a reward, the initiator sends the current member list in its local cache to P_j. In this way, each peer has an extended Peer-to-Peer community.

9.5 Experiments In this section, we study the performance of the proposed framework for Peer-to-Peer community formation.

9.5.1 Data Preparation We use the web domains a peer has browsed to create the profile vector. Each element of the vector corresponds to the frequency that the domain has been visited by the peer during a period of time. The data was collected from the IE history files of volunteers from UMBC and Johns Hopkins University. There are totally 97050 browsing history records in our data set, and 722 unique web domains. These records are randomly split and distributed to peers in our network simulator so that each peer can compute its own profile vector. As we have stated previously, we assume all the peers agree on the same profile ontology, *i.e.* the same set of domain names, and therefore, all the profile vectors have the same size - 722. Figure 12 shows a snapshot of a peer's profile.

9.5.2 Performance Having discussed about the data and the simulator setup we are in a position to report the experimental results.

- **Random Sampling and Quantile Estimation:** This experiment evaluates the accuracy of random sampling and quantile estimation. We chose three different p values - $80\%, 85\%$ and 90%. In all the three cases, the confidence level q was set to 95%, and the size of the network was fixed at 100 nodes. According to Equation 5.1, the number of samples, denoted by n, necessary to guarantee that the highest order statistic is within the top $(1-p)$ percentile of population is given by $14, 19$ and 29, respectively. Let P_i be the community initiator. The population can be defined as the set of all pairwise scalar products between P_i and all the other peers. Now,

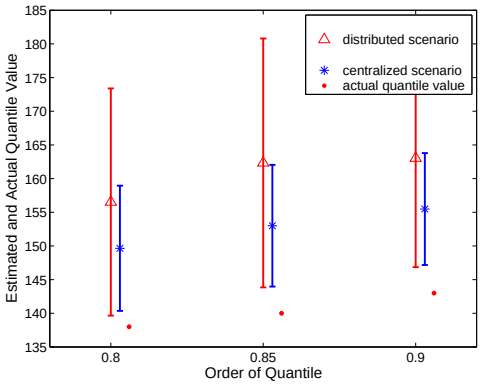


Figure 10: Estimated and actual quantile value w.r.t. the order of quantile. The results are an average of 100 independent runs.

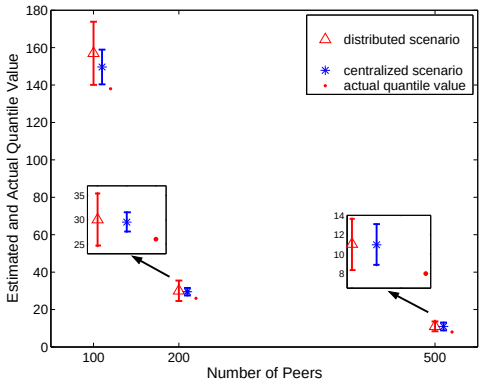


Figure 11: Estimated and actual quantile value w.r.t. the number of peers for fixed $p = 0.8, q = 0.95$. The results are an average of 100 independent runs.

if P_i wants to find similar peers who are in the top $(1-p)$ quantile of the population, it launches n random walks. The terminal peer for each random walk refers to a sample and P_i computes the scalar product between its own vector and the vector owned by the sample. P_i sorts all the n scalar products and finds the largest one as the threshold of quantile of order p. Figure 10 shows estimated threshold in the distributed experiment. To compare the results with centralized sampling, P_i first collects the pairwise scalar products between itself and all the peers in the network. P_i then performs a random sampling of size n and finds the largest scalar product. The threshold found by this approach is illustrated by the stars in Figure 10. Figure 10 also shows the actual population quantile of order p. As is evident from these results, the threshold found through random sampling and order statistics theory is above the actual population quantile. Therefore any scalar product greater than this threshold can be recognized as among the top $(1-p)$ quantile population with high confidence.

The next experiment measures the accuracy of the random sampling and quantile estimation algorithm with respect to the number of peers - $100, 200$ and 500. In each of these cases, the quantile of the population to monitor was set at 80%, and the confidence level was fixed to 95%. Figure 11 shows similar results that in all the three cases the average ordinal thresholds are greater than the actual quantiles of the population. Note that as we increase the size of the network the scalar product between any two peers decreases because the original data set is now divided into more partitions and hence each profile vector becomes more sparse.

TTL	Ave Num of Community Members
3	3
4	8
8	13

Table 1: Average number of community members found by the initiator without community expansion.

- **Community Formation:** Once the threshold is detected, the next step is to form the communities. We experimented with two community formation schemes. One is without community expansion and one is with expansion. The size of the network was fixed to be 100. Table 1 shows the average number of members found by a community initiator with respect to different TTL values. Table 2 presents the results using the community expansion scheme.

10 Conclusion

In this paper we have developed a new algorithm for efficiently identifying some user specified l entries of the inner product matrix that belong to the top $1-p$ percentile of the population. In order to achieve low communication complexity for our distributed algorithm, we have used an ordinal statistics based approach together with cardinal sampling. Ordinal statistics provides a

TTL	Ave Num of Community Members
3	7
4	12
8	17

Table 2: Average number of community members found by the initiator with community expansion.

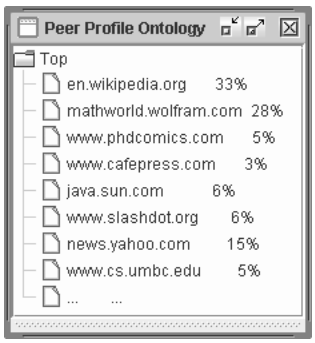


Figure 12: Snapshot of a peer's profile.

general framework for estimating distribution free confidence intervals for population percentiles. What this means is that, for any data distribution, we can use the same theory developed here in order to estimate the top-l elements. Using simple cardinal approximation is more communication intensive. Similarly, we cannot use only ordinal sampling since the inner product entries are distributed among the peers. Thus, using both, we can achieve good results. In this work we bounded both the message complexity of our algorithm and the error in our decision making. We have provided experimental results that substantiate our claims regarding accuracy and message complexity of our algorithm. Finally, we have presented a a real-life application of online p2p community formation using our technique.

As a future work, we are currently trying to extend this framework for heterogeneously distributed scenarios.

Acknowledgements

This research is supported by the United States National Science Foundation CAREER award IIS-0093353. The authors would also like to thank Phoung Nguyen who collected the web data for the experiments.

References

[1] J. H. Friedman, "On bias, variance, 0/1-loss, and the curse-of-dimensionality," *Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 55–77, 1997.

[2] C. Aggarwal, "On k-anonymity and the curse of dimensionality," in *Proceedings of the 31st VLDB Conference*, 2005.

[3] S. Datta, C. Giannella, and H. Kargupta, "K-means clustering over large, dynamic networks," in *In proceedings of 2006 SIAM Conference on Data Mining (SDM-2006)*, Bethesda, Maryland, 2006.

[4] S. Banyopadhyay, C. Giannella, U. Maulik, H. Kargupta, K. Liu, and S. Datta, "Clustering distributed data streams in peer-to-peer environments," *Information Science*, 2005.

[5] R. Wolff and A. Schuster, "Association rule mining in peer-to-peer systems," in *Proc. of the ICDM'03*, 2003.

[6] R. Wolff, K. Bhaduri, and H. Kargupta, "Local L2 thresholding based data mining in peer-to-peer systems," in *Proceedings of SIAM International Conference in Data Mining (SDM)*, Bethesda, Maryland, 2006.

[7] R. I. Arriaga and S. Vempala, "An algorithmic theory of learning: Robust concepts and random projection." in *Proceedings of the 40th Foundations of Computer Science*, 1999.

[8] K. Liu, H. Kargupta, and J. Ryan, "Random Projection-Based Multiplicative Data Perturbation for Privacy Preserving Distributed Data Mining," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 18, no. 1, pp. 92–106, January 2006. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/TKDE.2006.14

[9] C. Giannella, K. Liu, T. Olsen, and H. Kargupta, "Communication efficient construction of deicision trees over heterogeneously distributed data," in *Proceedings of the Fourth IEEE International Conference on Data Mining (ICDM04)*, 2004.

[10] B. Babcock and C. Olston, "Distributed top-k monitoring," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, 2003.

[11] W. Balke, W. Nejdl, W. Siberski, and U. Thaden, "Progressive distributed top-k retrieval in peer-to-peer networks," in *Proceedings of the 21st International Conference on Data Engineering (ICDE'05)*, 2005.

[12] F. Cuenca-Acuna, C. Peery, R. Martin, and T. Nguyen, "Planetp: Using gossiping to build content addressable peer-to-peer information sharing communities," in *International Symposium on High-Performance Distributed Computing*, 2003.

[13] K. Tumer and J. Ghosh, "Robust combining of disparate classifiers through order statistics," *Pattern Analysis and Applications*, vol. 5, pp. 189–200, 2001.

[14] H. Kargupta, "Search computational processes in evolution and preliminary development of the gene expression messy genetic algorithm," *Complex Systems*, vol. 11, pp. 189–200, 1997.

[15] M. B. Greenwald and S. Khanna, "Power-conserving computation of order-statistics over sensor networks," in *PODS*, Paris, France, June 2004.

[16] Y.-C. Ho, C. G. Cassandras, C.-H. Chen, and L. Dai, "Ordinal optimization and simulation," *Journal of Operations Research Society*, vol. 51, pp. 490–500, 2000.

[17] H. A. David, *Order Statistics*. John Wiley and Sons, Inc., 1970.

[18] L. Lovász, "Random walks on graphs.. a survey," *Combinatorics*, vol. 2, no. 80, pp. 1–46, 1993.

[19] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equations of state calculations by fast computing machines," *Journal of Chemi-*

cal Physics, vol. 21, pp. 1087–1092, 1953.

[20] W. Hastings., "Monte carlo sampling methods using markov chains and their applications," *Biometrika*, vol. 57, pp. 97–109, 1970.

[21] A. Awan, R. A. Ferreira, S. Jagannathan, and A. Grama, "Distributed uniform sampling in unstructured peer-to-peer networks," in *Hawaii International Conference on System Sciences*, 2006.

[22] K. Horowitz and D. Malkhi, "Estimating network size from local information," *The Information Processing Letters Journal*, vol. 88, no. 5, pp. 237–243, December 2003.

[23] P. Orponen and S. E. Schaeffer, "Efficient algorithms for sampling and clustering of large nonuniform networks," arXiv.org e-Print archive, Tech. Rep. cond-mat/0406048, 2004.

[24] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "A measurement study of peer-to-peer file sharing systems," in *Proceedings of Multimedia Computing and Networking (MMCN'02)*, San Jose, CA, January 2002.

[25] K. Liu, K. Bhaduri, K. Das, P. Nguyen, and H. Kargupta, "Client-side web mining for community formation in peer-to-peer environments," in *WebKDD'06*, 2006.

[26] A. H. Maslow, *Motivation and Personality*, 3rd ed. HarperCollins Publishers, January 1987.

[27] J. Trajkova and S. Gauch, "Improving ontology-based user profiles," in *Proceedings of RIAO*, Vaucluse, France, April 2004, pp. 380–389.