Distributed Classification in Peer-to-Peer Networks

Ping Luo
Chinese Academy of Sciences
luop@ics.ict.ac.cn

Hui Xiong
Rutgers University
hui@rbs.rutgers.edu

Kevin Lü
Brunel University
kevin.lu@brunel.ac.uk

Zhongzhi Shi
Chinese Academy of Sciences
shizz@ics.ict.ac.cn

ABSTRACT

This work studies the problem of distributed classification in peer-to-peer (P2P) networks. While there has been a significant amount of work in distributed classification, most of existing algorithms are not designed for P2P networks. Indeed, as server-less and router-less systems, P2P networks impose several challenges for distributed classification: (1) it is not practical to have global synchronization in large-scale P2P networks; (2) there are frequent topology changes caused by frequent failure and recovery of peers; and (3) there are frequent on-the-fly data updates on each peer.

In this paper, we propose an ensemble paradigm for distributed classification in P2P networks. Under this paradigm, each peer builds its local classifiers on the local data and the results from all local classifiers are then combined by plurality voting. To build local classifiers, we adopt the learning algorithm of *pasting bites* to generate multiple local classifiers on each peer based on the local data. To combine local results, we propose a general form of Distributed Plurality Voting (DPV) protocol in dynamic P2P networks. This protocol keeps the *single-site validity* for dynamic networks, and supports the computing modes of both one-shot query and continuous monitoring. We theoretically prove that the condition $\mathcal{C}_0$ for sending messages used in DPV_0 is locally communication-optimal to achieve the above properties. Finally, experimental results on real-world P2P networks show that: (1) the proposed ensemble paradigm is effective even if there are thousands of local classifiers; (2) in most cases, the DPV_0 algorithm is local in the sense that voting is processed using information gathered from a very small vicinity, whose size is independent of the network size; (3) DPV_0 is significantly more communication-efficient than existing algorithms for distributed plurality voting.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications–Data Mining; C.2.4 [**Computer-communication Networks**]: Distributed Systems—*Distributed applications*

KDD'07, August 12–15, 2007, San Jose, California, USA.

General Terms

Algorithms, Experimentation

Keywords

Distributed classification, P2P networks, Distributed plurality voting

1. INTRODUCTION

Peer-to-peer (P2P) networks [9] are an emerging technology for sharing content files containing audio, video, and realtime data, such as telephony traffic. A P2P network relies primarily on the computing power and bandwidth of the participants in the network and is typically used for connecting nodes via largely ad hoc connections. Detection and classification of objects moving in P2P networks is an important task in many applications.

Motivating Examples. In a P2P anti-spam network, millions of users form a community and spam emails are defined by a consensus, particularly in a collaborative endeavor. Essentially, the P2P network allows users to share their anti-spam experiences without exposing their email content. Another example is to automatically organize Web documents in a P2P environment [21]. Initially, the *focused crawler* on each peer starts to gather the Web pages, which are related to the user-specified topic taxonomy. Based on the training data, a local predictive model can be derived, which allows the system to automatically classify these Web pages into specific topics. Once we link together a large number of users with shared topics of interest in P2P networks, it is natural to aggregate their knowledge to construct a global classifer that can be used by all members.

However, P2P networks are highly decentralized, dynamic and normally includes thousands of nodes. Also, P2P network usually does not have routers and the notion of clients or servers. This imposes several challenges for distributed classification in P2P networks. First, it is not practical to have global synchronization in large-scale P2P networks. Also, there are frequent topology changes caused by frequent failure and recovery of peers. Finally, there are frequent on-the-fly data updates on each peer.

To meet the above challenges, in this paper, we propose to build an ensemble classifier for distributed classification in P2P networks by *plurality voting* on all the local classifiers, which are generated on the local data. To this end, we first adapt the training paradigm of *pasting bites* [6, 8] for building local classifiers. Since data are not uniformly distributed on each node, the number of local classifiers re-

quired to generate can be different for different local regions. As a result, we provide a modified version of *pasting bites* to meet this requirement in P2P networks.

Next, to combine the decisions by local classifiers, we formalize the problem of Distributed Plurality Voting (DPV) in dynamic P2P networks. Specifically, we propose a general form of DPV protocol which keeps the *single-site validity* [3] for dynamic networks, and supports the computing modes of both one-shot query and continuous monitoring. Furthermore, we theoretically prove that the condition $\mathcal{C}_0$ for sending messages used in DPV_0 is locally communication-optimal to achieve the above properties. While $\mathcal{C}_0$ used in DPV_0 is only locally (not globally) communication-optimal, the experimental results show that DPV_0 is significantly more communication-efficient than existing algorithms for distributed plurality voting. Thus, this ensemble paradigm is communication-efficient in P2P networks, since neighbor-to-neighbor communication is mainly concerned in combining local outputs by DPV without any model propagations.

Finally, we have shown that our DPV algorithm can be extended to solve the problem of Restrictive Distributed Plurality Voting (RDPV) in dynamic P2P networks. This problem requires that the proportion of the maximally voted option to all the votes be above a user-specified threshold. RDPV can be used in a classification ensemble in a restrictive manner, i.e. by leaving out some uncertain instances.

2. RELATED WORK

Related literature can be grouped into two categories: ensemble classifiers and P2P data mining.

Ensemble classifiers. The main idea is to learn an ensemble of classifiers from subsets of data, and combine predictions from all these classifiers in order to achieve high classification accuracies. There are various methods for combining the results from different classifiers, such as voting and meta-learning [7]. In (weighted) voting, a final decision is made by a plurality of (weighted) votes. Effective voting [23] is also proposed to vote the most significant classifiers selected by statistical tests. Lin et al. [17] theoretically analyzed the rationale behind plurality voting, and Demrekler et al. [12] investigated how to select an optimal set of classifiers. In addition, meta-learning is used to learn a combining method based on the meta-level attributes obtained from predictions of base classifiers. The meta-learning method includes *arbiter* [7], *combiner* [7], multi-response model trees [11] using an extended set of meta-level features, meta decision trees [22] and so on.

Ensemble classifiers have been used for classification in both centralized and distributed data as follows. For centralized massive data, base-level classifiers are generated by applying different learning algorithms with heterogeneous models [19, 23], or a single learning algorithm to different versions of the given data. For manipulating the data set, three methods can be used: random sampling with replacement in bagging [5], re-weighting of the mis-classified training examples in boosting [15], and generating small bites of the data by importance sampling based on the quality of classifiers built so far [6].

For naturally distributed data, several techniques have been proposed as follows. Lazarevic et al. [16] give a distributed version of boosting algorithm, which efficiently integrates local classifiers learned over distributed homogeneous databases. Tsoumakas et al. [24] present a framework for constructing a global predictive model by stacking local classifiers propagated among distributed sites. Chawla et al. [8] present a distributed approach to pasting small bites, which uniformly votes hundreds or thousands of classifiers built on all distributed data sites. Chawla's algorithm is fast, accurate and scalable, and illuminates the classification framework proposed in the following section.

P2P data mining. With the rapid growth of P2P networks, P2P data mining is emerging as a very important research topic in distributed data mining. Approaches to P2P data mining have focused on developing some primitive operations as well as more complicated data mining algorithms [9]. Researchers have developed some algorithms for primitive aggregates, such as *average* [20, 18], *count* [3], *sum* [3], *max*, *min*, *Distributed Majority Vote* (DMV) [27] and so on. The aggregate *count*, *sum* and DMV are inherently duplicate sensitive, which requires that the value on each peer be computed only once. To clear this hurdle Paper [3] processes *count* and *sum* by probabilistic counting [14] in an approximate manner, while the DMV algorithm in [27] performs on a spanning tree over the P2P network. A main goal of these studies is to lay a foundation for more sophisticated DM algorithms in P2P systems. Pioneer works for P2P DM algorithms include P2P association rule mining [27], P2P K-means clustering [10], P2P L2 threshold monitor [26] and outlier detection in wireless sensor networks [4]. They all compute their results in a totally decentralized manner, using information from their immediate neighbors. A recent paper [21] presents a classification framework for automatic document organization in a P2P environment. However, this approach propagates local models between neighboring peers, which heavily burdens the communication overhead. It only focuses on the accuracy issue, and does not involve any dynamism of P2P networks. The experiments of this method were performed only with up to 16 peers. Thus, it is just a minor extension of small-scale distributed classification.

To the best of our knowledge, the most similar work to our DPV problem is the work on DMV [27]. To illustrate the difference, let us consider the case that a group of peers would agree on one of d options. Each peer u conveys its preference by initializing a voting vector $P^{\perp u} \in \mathbb{N}^d$ (where $\mathbb{N}$ is the set of integers), and $P^u[i]$ is the number of votes on the i-th option. DPV is to decide which option has the largest support over all peers, while DMV is to check whether the voting proportion of a specified option to all the votes is above a given threshold. Thus, DPV is a multi-valued function while DMV is a binary predicate. In addition, the advantage of DPV over DMV will be further illustrated in Section 4.

3. BUILDING LOCAL CLASSIFIERS

In this section, we introduce the method for building local classifiers on the local data in P2P networks. Specifically, the approach we are exploring is built on top of the pasting bites method, which is also known as Ivote [6]. In the Ivote method, multiple local classifiers are built on small training sets (bites) of a specific local data and the bite for each subsequent classifier relies on the plurality voting of the classifiers built so far. In other words, bites are generated from a specific local data by sampling with replacement based on the *out-of-bag error*. Also, a local classifier is only tested on the instances not belonging to its training set. This out-of-bag error estimation provides a good approximation on the

Algorithm 1 A Pasting Bites Approach for Building Local Classifier

Input: the size of each bite, N; the minimal difference of error rates between two successive classifiers, λ; the base training algorithm, $\aleph$.

Output: multiple local classifiers

1: **if** the size of local data on the peer is less than N **then**
2: Learn a classifier on the whole data by $\aleph$
3: **else**
4: Build the first bite of size N by sampling with replacement from its local data, and learn a classifier by $\aleph$
5: Compute the smoothed error, $e(k)$; and the probability of selecting a correctly classified instance, $c(k)$ (k is the number of classifiers in the ensemble so far), as follows:
$r(k) :=$ the *out-of-bag error* rate of the k aggregated classifiers on the local data.
$e(k) := p \times e(k-1) + (1-p) \times r(k)$ ($p := 0.75$, $e(0) := r(1)$).
$c(k) := e(k)/(1-e(k))$.
6: For the subsequent bite, an instance is drawn at random from the local data. If this instance is misclassified by a plurality vote of the out-of-bag classifiers (those classifiers for which this instance was not in their training set), then it is put in the subsequent bite. Otherwise, put this instance in the bite with the probability of $c(k)$. Repeat until N instances have been selected for the bite.
7: Learn the $(k+1)$-th classifier on the bite created by step 6.
8: Repeat steps 5 to 7, until $|e(k) - e(k-1)| < \lambda$
9: **end if**

generalization error, and is used as the stop condition for the training process. Indeed, Ivote is very similar to boosting, but the sizes of "bites" are much smaller than that of the original data set.

The key of the pasting bites method is to compute the *out-of-bag error* rate $r(k)$ of the k current aggregated classifiers on the local data. To compute $r(k)$, we add two data structures Λ and $\vec{V}$ on each instance of the local data. Λ records the index of the last classifier, whose training set includes this instance. $\vec{V}$ records the vote values of the instance on each class label by the out-of-bag classifiers so far. The details of this algorithm are shown in Algorithm 1. In k-th round of Steps 5 through 7, for each instance $\vec{V}$ is updated by the last built classifier only when $\Lambda \neq k-1$. Then, by plurality voting, $r(k)$ can be computed easily from the $\vec{V}$s on all the instances. The stop criteria in Step 8 satisfies if the difference of error rates between two successively generated classifiers is below λ.

Although the local classifier is only trained on a small portion of the raw data, there is still a large communication burden for model propagations when thousand or even more local classifiers participate into the classification process. In addition, local models are frequently updated caused by frequent updates of distributed data. Therefore, for distributed classification as described in the next section, each peer is responsible for maintaining its own local classifiers, which are never propagated in the rest of the network.

4. A DISTRIBUTED PLURALITY VOTING PROTOCOL FOR CLASSIFICATION IN P2P NETWORKS

In this section, we present a Distributed Plurality Voting (DPV) protocol for classification in P2P networks. Specifically, every peer will participate in the distributed voting. The final prediction is made by the local votes from all the local classifiers. The distributed plurality voting protocol is used to make sure that the results from all nodes that are reachable from one another will converge.

4.1 Problem Definition

We first give the problem definition. Following the notations in [27], we assume that a group U of peers in a P2P network, denoted by a connected graph $G(U, E)$, would like to agree on one of d options. Each peer $u \in U$ conveys its preference by initializing a voting vector $P^{\perp u} \in \mathbb{N}^d$, where $\mathbb{N}$ is the set of integers and $P^{\perp u}[i]$ is the number of votes on the i-th option. DPV is to decide which option has the largest number of votes over all peers (it may contain multiple maximally voted options), formalized as follows:

$$arg \ \max_i \sum_{u \in U} P^{\perp u}[i].$$

For distributed majority voting (DMV) [27], each peer $u \in U$ initializes a 2-tuple $\langle sum^{\perp u}, count^{\perp u} \rangle$, where $sum^{\perp u}$ stands for the number of the votes for a certain option on peer u and $count^{\perp u}$ stands for the number of the total vote on peer u. This majority voting is to check whether the voting proportion of the specified option is above a given majority ratio η. This is formalized as follows:

$$sgn \ \sum_{u \in U} (sum^{\perp u} - \eta \cdot count^{\perp u}).$$

From the above, we know that DPV is a multi-valued function while DMV is a binary predicate. Also, we can see that DMV can be converted to DPV by replacing the 2-tuple $\langle sum^{\perp u}, count^{\perp u} \rangle$ on each peer with the voting vector $\langle sum^{\perp u}, \eta \cdot count^{\perp u} \rangle$. However, DMV can only be used for 2-option DPV as a pairwise comparison. For a d-option DPV problem, pairwise comparisons among all the d options must be performed by DMV for $\frac{d \cdot (d-1)}{2}$ times, as suggested by *Multiple Choice Voting* [27], whose function is actually to rank the d options by their votes (in the rest of this paper, we refer this algorithm as $RANK$). The proposed DPV algorithm in this paper do not need to perform pairwise comparisons many times. Instead, it finds the maximally supported option directly, and thus saves a lot of communication overhead and the time for convergence. Therefore, DPV is a more general form of DMV.

Algorithm 2 $\mathcal{C}_0(P^{vu}, P^{uv}, \Delta^{uv})$

1: **for all** i^{uv} such that $i^{uv} \in \vec{i}^{uv}$ **do**
2: **for all** $j := 1, \cdots, d$ such that $j \neq i^{uv}$ **do**
3: **if** $(\Delta^{uv}[i^{uv}] - \Delta^{uv}[j]) < (P^{uv}[i^{uv}] - P^{uv}[j])$ **then**
4: **return** *true*
5: **end if**
6: **end for**
7: **end for**
8: **return** *false*

Algorithm 3 *DPV* Protocol

Input for node u: The set E^u of edges that collide with it, the local voting vector $P^{\perp u}$. $\mathcal{C}$ is the condition for sending messages.

Output: The algorithm can be used for both one-shot query and continuous monitoring. It outputs $\vec{i}^u := arg\ \max_i \sum_{vu \in N^u} P^{vu}[i]$ at each time point.

Definitions: See notation in Table 1.

Initialization: For each $vu \in E^u$, set P^{vu} to null, and send $P^{\perp u}$ over uv to v.

On failure of edge $vu \in E^u$: Remove vu from E^u.

On recovery of edge $vu \in E^u$: Add vu to E^u, and send $P^{\perp u}$ over uv to v.

On message P **received over edge** vu: Set P^{vu} to P.

On any change in Δ^{uv} **(**$uv \in E^u$**), resulting from a change in the input, edge failure or recovery, or the receiving of a message**:
for each $vu \in E^u$ **do**
  if $\mathcal{C}(P^{vu}, P^{uv}, \Delta^{uv}) = true$ **then**
    send $\Delta^{uv} := \sum_{wu \neq vu \in N^u} P^{wu}$ over uv to v
  end if
end for

4.2 The *DPV* Protocol

The purpose of *DPV* is to make sure that every node in the network converges toward the correct plurality through neighbor-to-neighbor communication. The *DPV* protocol includes a mechanism to maintain an un-directional spanning tree [28] for the dynamic P2P network, and a node is informed of changes in the status of adjacent nodes.

Because there is only one path between two nodes in a tree, the voting vector on each peer is only added once through the edges in the tree. This ensures that the DPV protocol is *duplicate insensitive*.

Table 1: Some Notations for *DPV*

Symbol	Meaning
P^{uv}	the last voting vector sent from u to v
P^{vu}	the last voting vector sent from v to u
E^u	$\{vu$: v and u are neighbors$\}$
$\perp u$	the virtual edge from u to itself
N^u	$E^u \cup \{\perp u\}$
$P^{\perp u}$	the voting vector of node u

The *DPV* protocol specifies how nodes react when the data changes, a message is received, or a neighboring node is reported to have detached or joined. The nodes communicate by sending messages that contain the voting vector. Each node u will record, for every neighbor v, the last message it sent to v, P^{uv}, and the last message it received from v, P^{vu}. For conciseness, we extend the group of edges colliding with u, denoted by E^u, to include the virtual edge $\perp u$, and the extended edge set is denoted by N^u. These related notions are listed in Table 1. Node u calculates the following functions of its messages and its own voting vector:

$$\Delta^{uv} := \sum_{wu \neq vu \in N^u} P^{wu}, \quad \Delta^u := \Delta^{uv} + P^{vu},$$

$$\Gamma^{uv} := P^{uv} + P^{vu}, \quad \vec{i}^{uv} := arg\ \max_i \Gamma^{uv}[i],$$

$$\vec{i}^u := arg\ \max_i \Delta^u[i].$$

When the local voting vector changes, a message is received, or a node connects to u or disconnects from u, The above functions, including Δ^{uv}, Γ^{uv}, $\vec{i}^{uv}$ and $\vec{i}^u$, will be recalculated. Δ^{uv} is the message when the protocol asks u to send to v when necessary. Γ^{uv} and $\vec{i}^{uv}$ are the voting vectors and the corresponding set of all maximally voted option(s) on v from the view point of u, respectively. Note that if no message is yet received from any neighbor, then $\vec{i}^u$ converges to the right result, the option(s) with the largest number of votes over all the peers.

The enforcement of the voting proposal on each node is independent from that of the immediate neighbors of this node. Node u coordinates its plurality decision with node v by making sure that P^{uv} will not lead v to a wrong result. When Δ^{uv} changes, the protocol dictates that u send v a message, Δ^{uv}, if a certain condition $\mathcal{C}_0$ satisfies. This condition is actually a boolean function of three inputs P^{vu}, P^{uv} and Δ^{uv}. In the following, we detail this condition, which is the key of the *DPV* protocol.

4.2.1 The condition for sending messages $\mathcal{C}_0$

Algorithm 2 shows the pseudo-code of $\mathcal{C}_0$ for sending messages. The inequality in Algorithm 2 says that if Δ^{uv} can result in a decrease of the voting difference between the maximally voted option i^{uv} and any other option j, Δ^{uv} must be sent. Actually, $\mathcal{C}_0$ is the more generic and extended form of the condition in DMV [27] for generic d-option DPV problems. These two conditions are equivalent only when solving 2-option voting problems. Therefore, *DPV* and *DMV* are actually equivalent for 2-option voting problems. However, *DMV* cannot solve d-option DPV problems ($d > 2$) directly. In addition, $\mathcal{C}_0$ saves much communication overhead for d-option DPV problems ($d > 2$), because it avoids multiple pairwise comparisons in *RANK*.

4.2.2 Descriptions of the *DPV* Protocol

Algorithm 3 shows the pseudo-code of the general form of the *DPV* protocol with a condition $\mathcal{C}$ as an input. DPV_0 is an instance of this algorithm, in which $\mathcal{C}_0$ is used.

To transparently deal with the frequent changes of the network topology and the inputs of voting vectors, this *DPV* protocol is designed in a way such that every peer regards itself as the root of the spanning tree. During the execution of this algorithm, each node maintains an ad-hoc solution. If the system remains static long enough, this solution will quickly converge to the exact (not approximate) answer. For the dynamic system, where nodes dynamically join or depart and the votes on each peer change over time, the ad-hoc solutions are adjusted quickly and locally to the current right answer. Therefore, this *DPV* protocol keeps the *single-site validity*, and supports the computing modes of both one-shot query and continuous monitoring for distributed plurality voting in P2P networks. Additionally, in Section 5, our experimental results also show that this protocol demonstrates

a local effect: in the most of cases, each node computes the plurality voting based on information arriving from a very small surrounding environment. Locality implies that this algorithm should be scalable to very large networks.

It can be proved that DPV_0 keeps the correctness of distributed plurality voting. When this protocol specifies that no node needs to send any message, $\vec{i}^u$ converges to the right result for all nodes. If there is a disagreement, there must be a disagreement between two immediate neighbors, in which case at least one node satisfies the inequality and sends a message. This will cause the total number of votes received to increase. This total number is bounded by the number of classifiers in the system. Hence, this protocol always reaches the right consensus in a static state. The details of this proof are presented in the Appendix.

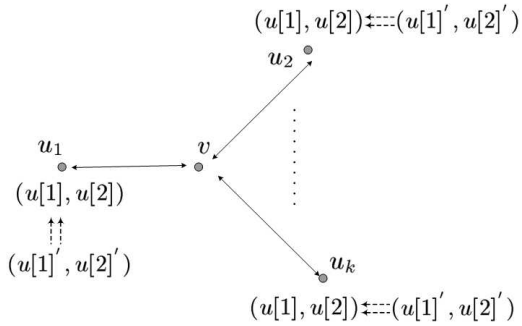


Figure 1: The error case

Algorithm 4 $\mathcal{C}_e(P^{vu}, P^{uv}, \Delta^{uv})$, $0 \leq e \leq 1$

1: **if** $\mathcal{C}_0(P^{vu}, P^{uv}, \Delta^{uv})$=*true* **then**
2: **return** *true*
3: **end if**
4: e' :=a random value in $[0, 1)$
5: **if** $e' < e$ **then**
6: **return** *true*
7: **else**
8: **return** *false*
9: **end if**

4.3 The Local Optimality in terms of Communication Overhead

In the following, we provide a formal description of the local communication optimality for the DPV_0 protocol, and then theoretically prove it. It is shown that $\mathcal{C}_0$ is the most restrictive condition for sending messages to support the computing modes of both one-shot query and continuous monitoring for distributed plurality voting in P2P networks.

DEFINITION 1. *Condition $\mathcal{C}^1$ is more restrictive than condition $\mathcal{C}^2$ denoted by $\mathcal{C}^1 \succeq \mathcal{C}^2$, if and only if, for any input case if $\mathcal{C}^1$ satisfies then $\mathcal{C}^2$ satisfies.*

DEFINITION 2. *Condition $\mathcal{C}^1$ is strictly more restrictive than condition $\mathcal{C}^2$ denoted by $\mathcal{C}^1 \succ \mathcal{C}^2$, if and only if, $\mathcal{C}^1$ is more restrictive than $\mathcal{C}^2$ and there at least exists an input case such that $\mathcal{C}^2$ satisfies and $\mathcal{C}^1$ does not.*

THEOREM 1. *For the problem of distributed plurality voting there does not exist a condition $\mathcal{C}$ which is strictly more restrictive than $\mathcal{C}_0$, such that the Algorithm 3 with $\mathcal{C}$ as the condition for sending messages still reaches the right answer for every input case.*

PROOF. We first assume that such a condition $\mathcal{C}$ exists, then construct the cases for which it reaches a wrong result. Without loss of generality, the following analysis is under the assumption of a 2-option DPV problem.

Because $\mathcal{C} \succ \mathcal{C}_0$, at least the following case Π exists: u and v are two immediate neighbors, $\mathcal{C}_0(P^{uv}, P^{vu}, \Delta^{uv})$ is *true*, $\mathcal{C}(P^{uv}, P^{vu}, \Delta^{uv})$ is *false*. We assume that $P^{uv} = (u[1], u[2])$, $P^{vu} = (v[1], v[2])$, $\Delta^{uv} = (u[1]', u[2]')$, $u[1] + v[1] > u[2] + v[2]$. Since $\mathcal{C}_0(P^{uv}, P^{vu}, \Delta^{uv})$ is *true*, $(u[1]' + v[1]) - (u[2]' + v[2]) < (u[1] + v[1]) - (u[2] + v[2])$. Then we construct the error case in Figure 1, in which v is surrounded by k nodes of $u_1, \cdots, u_k$ where

$$k = \lceil \frac{(u[1] + v[1]) - (u[2] + v[2])}{((u[1] + v[1]) - (u[2] + v[2])) - ((u[1]' + v[1]) - (u[2]' + v[2]))} \rceil.$$

In this figure the vector on the double-arrowhead side is the corresponding vector P which has been sent, the vector on the double-arrowtail side is the corresponding vector Δ which will be sent if the condition satisfies.

Initially, $P^{\perp v}$ is set as a vector (x, y) such that

$$x + k \cdot u[1] < y + k \cdot u[2],$$

and $P^{\perp u_i}$ is set as $(u[1], u[2])$ for $i = 1, \cdots, k$. Under this initial setting, all the peers converge to the state that the second vote is the maximally voted option.

After this convergence, $P^{\perp v}$ changes to $(v[1] - (k-1) \cdot u[1], v[2] - (k-1) \cdot u[2])$. Because $u[1] + v[1] > u[2] + v[2]$, v changes to the state that the first vote is the maximally voted option. Then, it sends $(v[1], v[2])$ to u_i for $i = 1, \cdots, k$. Thus, Δ^{vu_i} changes to $(v[1], v[2])$ for $i = 1, \cdots, k$.

At this time, $\Delta^{u_i v}$ on each node u_i $(i = 1, \cdots, k)$ changes to $(u[1]', u[2]')$. Because $\mathcal{C}(P^{uv}, P^{vu}, \Delta^{uv})$ is *false*, $(u[1]', u[2]')$ is not sent from u_i to v. Thus, v remains to the state that the first vote is the maximally voted option. However, the right state for the whole network is the second one, because by the initial setting the following inequality holds:

$$v[2] - (k-1) \cdot u[2] + k \cdot u[2]' \geq v[1] - (k-1) \cdot u[1] + k \cdot u[1]'.$$

It generates an error case, and follows the conclusion. □

The above proof of Theorem 1 shows that to keep the correctness of Algrithm 3, Δ^{uv} must be sent from u to v for any case, which satisfies the following conditions: u and v are two immediate neighbors, $\mathcal{C}_0(P^{uv}, P^{vu}, \Delta^{uv})$ is *true*; otherwise, it will result in the error case in Figure 1. Therefore, the conditions in Algorithm 4 for $0 \leq e \leq 1$ includes all the conditions for sending messages, which keep the correctness of Algrithm 3. In this family of conditions, $\mathcal{C}_0$ is the most restrictive one. We should mention that $\mathcal{C}_0$ is a local optimal condition, not a global one. This indicates that, for a certain input case, the total communication overhead used in $\mathcal{C}_e$ $(0 < e \leq 1)$ may be smaller than that used in $\mathcal{C}_0$. However, the careful experiments conducted in the following will show that the case mentioned above is very rare, and $\mathcal{C}_0$ is significantly more communication-efficient than $\mathcal{C}_e$ $(0 < e \leq 1)$.

4.4 Extensions of Distributed Plurality Voting

Here, we show that our *DPV* protocol can be extended to solve the problem of Restrictive Distributed Plurality Voting (RDPV) in dynamic P2P networks, which outputs the maximally voted option whose proportion to all the votes is above a user-specified threshold η. Using the notation in Section 4.1 this output is formalized as follows:

$$k \in arg \max_i \sum_{u\in U} P^{\perp u}[i] \wedge \sum_{u\in U} P^{\perp u}[k] \geq \eta \cdot \sum_i \sum_{u\in U} P^{\perp u}[i].$$

This can be used in P2P classification in a restrictive manner. In this manner the classification decision can only be made when the proportion of the maximally supported option to all the votes is not below the threshold η, and the uncertain instance is discarded.

This problem can be solved by Algorithm 3 with the following modifications: (1) the threshold η is added to the inputs of this algorithm; (2) the condition $\mathcal{M}$ for sending messages in Algorithm 5 is used; (3) the output changes to

$$i^u \in arg \max_i \sum_{vu\in N^u} P^{vu}[i] \wedge \sum_{vu\in N^u} P^{vu}[i^u] \geq \eta \cdot \sum_i \sum_{vu\in N^u} P^{vu}[i].$$

The lines through 7 to 12 in Algorithm 5 are responsible for the additional check of the additional constrain for RDPV. And this check is only applied to the current maximally voted option i^{uv}. The correctness of $\mathcal{M}$ is directly based on the correctness of $\mathcal{C}_0$, and thus the proof is omitted.

Algorithm 5 $\mathcal{M}(P^{vu}, P^{uv}, \Delta^{uv}, \eta)$

```
1: for all i^uv such that i^uv ∈ i⃗^uv do
2:   for all j := 1, ··· , d such that j ≠ i^uv do
3:     if (Δ^uv[i^uv] − Δ^uv[j]) < (P^uv[i^uv] − P^uv[j]) then
4:       return true
5:     end if
6:   end for
7:   Q^vu := (P^vu[i^uv], η · Σ_i P^vu[i])
8:   Q^uv := (P^uv[i^uv], η · Σ_i P^uv[i])
9:   Φ^uv := (Δ^uv[i^uv], η · Σ_i Δ^uv[i])
10:  if C_0(Q^vu, Q^uv, Φ^uv) = true then
11:    return true
12:  end if
13: end for
14: return false
```

5. EXPERIMENTAL EVALUATION

In this section, we present the experimental evaluation of our approach for P2P classification. Specifically, we demonstrate: (1) the performance of P2P classification in terms of accuracy, (2) the performance comparison of our DPV algorithms and *RANK* [27] in terms of communication overhead and the converge time. Three types of networks, including power-law graph [2], random graph [25] and grid, are used in our experiments. The power-law and random graphs are generated by BRITE[1] with the default parameters. For the n-node grid, the $n_1 \times n_2$ grid is generated where n_1 is the the largest common divisor of n and $n_1 \times n_2 = n$.

[1]http://www.cs.bu.edu/brite/

5.1 The Performance of P2P Classification

Due to the lack of labeled P2P data, we used the *covtype* (581012×54, 7 classes) data set, which is the largest data set in the UCI repository[2]. We first randomly divided this data set into hundreds of disjoint partitions in a way such that the size of each partition is not too small. Then, each partition was assigned to a peer in the P2P network. The P2P network used in this experiment is a power-law graph with 500 nodes. For this distributed classification, the data size s_i of peer i is set to be proportional to its number of immediate neighbors k_i, so that $s_i = s \cdot \frac{k_i}{\sum_i k_i}$ where s is the size of the original dataset. The rationale behind this is that if k_i of peer i is big, it indicates that peer i stays in this network for a long time, and thus accumulate more data. Also, we make the following initial settings to Algorithm 1: $N = 800, \lambda = 0.002$. Finally, the J48 (a variant of C4.5) from the Weka Package[3] is used as the base training algorithm and a 10-fold cross validation is performed.

Table 2: Classification Accuracy

Algorithm Type		**Accuracy**(%)	**Loss**(%)
centralized learning		80.0885	0
distributed learning	$\eta = 0$	74.7685	0
	$\eta = 0.4$	74.9171	0.3614
	$\eta = 0.5$	76.1324	4.3957
	$\eta = 0.55$	79.1231	14.5004
	$\eta = 0.6$	82.2541	24.4381

First, we compare the performance of the P2P classification and classification on the centralized data. The classification algorithm for the centralized data is the same with the local training algorithm on each peer. Next, we also evaluate the performance of this P2P classification by restrictive plurality voting, where the instance is discarded if the proportion of the maximally supported option to all the votes is below a threshold η. By changing the η values, we study the relationship between the accuracy (the fraction of the correctly classified instances) and the loss (the fraction of the discarded instances). The results of these two experiments are shown in Table 2.

As can be seen in Table 2, the accuracy of distributed learning is pretty close to that of centralized learning. Indeed, the difference is less than 6% for the most cases. This difference is mainly due to the fact that some classification patterns may only exist in the centralized data. In addition, Table 2 also demonstrates that the accuracy is increased as the increase of η values. However, this improvement is at the cost of the increase of data loss. Thus, there is a tradeoff between accuracy and loss.

5.2 A Performance Evaluation of Distributed Plurality Voting

To evaluate the performance of our DPV algorithms, we have implemented a discrete event simulator which has the capabilities in processing networks with different sizes and types. For each time unit (1 ms), this simulator checks each peer to see whether the condition for sending messages satisfies. It is assumed that each peer can only process one message in a time unit, and the delays on the immediate links are uniformly distributed between 40 ms and 1000 ms. Since

[2]http://www.ics.uci.edu/∼mlearn/MLRepository.html
[3]http://www.cs.waikato.ac.nz/ml/weka

Table 3: Average Metrics for the Power-law Graph

Algorithm	Avg. Value of $\bar{a}_n$	Avg. Rank of $\bar{a}_n$	Avg. Value of t_n (sec)	Avg. Rank of t_n
DPV_0	4.3421	1.0025	3.6039	1.0
$RANK$	15.5486	1.9975	49.3847	2.0

Table 4: Average Metrics for the Random Graph

Algorithm	Avg. Value of $\bar{a}_n$	Avg. Rank of $\bar{a}_n$	Avg. Value of t_n (sec)	Avg. Rank of t_n
DPV_0	4.2624	1.0	4.4279	1.0
$RANK$	15.7363	2.0	57.4102	2.0

the algorithms for one-shot query and continuous monitoring are performed in the same manner, we only measure the one-shot query in this experiment. For the one-shot query of plurality voting, a network topology and the voting vectors on all peers are firstly initialized and will not change during protocol execution.

We use two performance metrics: communication overhead and convergence time. In the experiment, we record the following two time points for an one-shot query: t_c (the time of convergence), which is the time when each peer reaches a right and stable result (sometimes all peers may reach the right results, however, the result on certain peer may not be stable); t_n, which is the time when no message is propagated in the network. t_c is always before t_n. In addition, we define the communication overhead of peer u as the volume of the messages it has sent. This overhead is equal to the number of received messages times the number of entries in each message[4]. The locality of a protocol is measured mainly according to the *average communication overhead* of all the nodes, denoted by $\bar{a}$. The two time points t_c and t_n corresponds to two *average communication overheads*: $\bar{a}_c$ and $\bar{a}_n$. $\bar{a}_c$ is the $\bar{a}$ consumed before t_c and $\bar{a}_n$ is the $\bar{a}$ consumed before t_n.

In the following experiments, we evaluate the performance of DPV from three aspects. First, we present a performance comparison of DPV_0 and $RANK$ algorithms. Next, we show the scalability of DPV_0. Finally, we illustrate the local optimality of DPV_0.

5.2.1 DPV_0 vs. $RANK$

For both DPV_0 and $RANK$[5], the average metrics over 2000 instances of 7-option plurality voting over 500-node networks are computed. In the following experimental data, the unit of communication overhead is the volume of each message consumed by this 7-option distributed plurality voting. Thus, each message consumed by $RANK$ has $\frac{2}{7}$ units.

We use a statistical method [13] in comparing these algorithms. This statistical method allows to give a rank to both algorithms for each instance; that is, a rank 1 is assigned to a winning algorithm and a rank 2 is assigned to the losing one. If there is a tie, the average value 1.5 is assigned to both algorithms. Let r_i^j be the rank of the j-th algorithm on the i-th of N instances, this method compares the *average ranks* of algorithms, $R^j = \frac{1}{N}\sum_{i=1}^{N} r_i^j$, by the Friedman test [13].

Tables 3, 4, and 5 show the comparison results on three network topologies: power-law graph, random graph and grid. As can be seen in these tables, DPV_0 significantly outperforms $RANK$ in terms of both $\bar{a}_n$ and t_n.

Our experimental results also indicate that the locality of DPV relates to the parameter r of the input votes: $r = \sum_u P^{\perp u}[a] - \sum_u P^{\perp u}[b]$, where a and b have the largest and the second largest number of votes. Figures 2 (a), (b), and (c) show the values of the $\bar{a}_n$ for DPV_0 at different r values, for three network topologies: power-law graph, random graph and grid, respectively. In these figures, we can see that $\bar{a}_n$ decreases as the increase of r. In addition, these figures show that in most cases ($r > 300$) $\bar{a}_n$ is always smaller than 5. This indicates that DPV_0 has a local effect in the sense that voting is processed using information gathered from a very small vicinity.

Finally, we observe that the *average rank* values of $\bar{a}_n$ for DPV_0 in Tables 3 and 5 are a little greater than 1. This indicates that there exists such a rare input case, where the $\bar{a}_n$ for $RANK$ is smaller than that for DPV_0. We also find that for both DPV_0 and $RANK$ the performance in terms of both $\bar{a}_n$ and t_n for the grid is worse than that for the power-law and random graph. A possible reason for this finding is that the *clustering coefficient* [25] of the grid is much smaller than that of the power-law graph and random graph.

5.2.2 The Scalability of DPV_0

In this experiment, we use the synthetic data to study the scalability of DPV_0. Specifically, we develop synthetic data models to simulate different problem environments for distributed plurality voting by changing model parameters. In general, there are two design issues for the generation of synthetic data models: the voting vector on each peer and the network topology.

We propose Algorithm 6 for voting vector modeling. Algorithm 6 has 3 parameters: n (the number of peers), *cov* (describing the degree of difficulty for the voting vectors), and $\vec{P}$ (the basic data vector). We first introduce this method, and then describe the meaning of the model parameters.

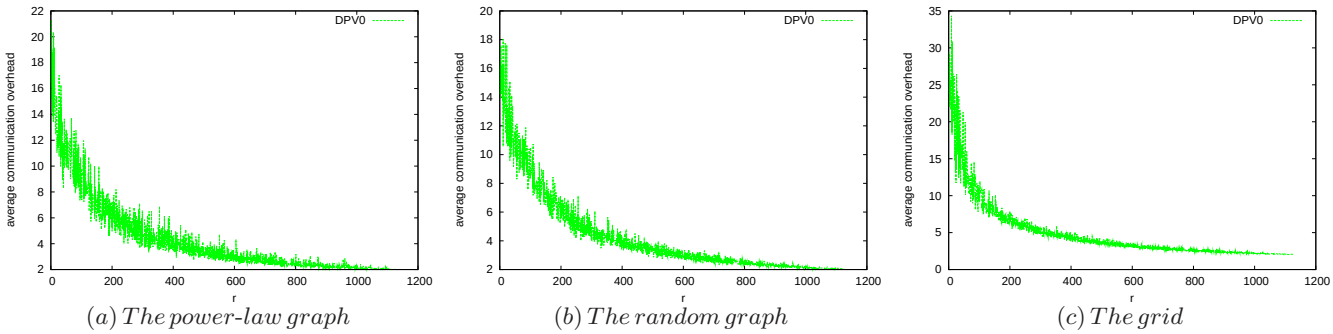
The function generateDistribution(*cov*) generates a distribution vector $\vec{D}$ by the *gamma* distribution [1], such that the bigger the value of *cov* is, the more different the values of all the entries ($\vec{D}[i] > 0$ for $i = 1, \cdots, d$) in $\vec{D}$ are. Then, the function generateOneVoteVector($\vec{D}, \vec{P}$) *randomly* generates a voting vector $\vec{P}'$, such that the bigger the value of $\vec{D}[i]$ is, the more possible that $\vec{P}'[i]$ is set by a bigger entry of $\vec{P}$, and each entry of $\vec{P}$ is used only once (the pseudo-codes

[4]Note that each message communicated by DPV contains d entries; however, every message communicated by $RANK$ contains only 2 entries. This has been considered when comparing the communication overheads of two algorithms.

[5]For RANK, we assume that the $\frac{d\times(d-1)}{2}$ pairwise comparisons are performed one after another; thus, the metric consumed by $RANK$ is the sum of the metrics of the $\frac{d\times(d-1)}{2}$ pairwise comparisons.

Table 5: Average Metrics for the Grid

Algorithm	Avg. Value of $\bar{a}_n$	Avg. Rank of $\bar{a}_n$	Avg. Value of t_n (sec)	Avg. Rank of t_n
DPV_0	4.9099	1.0215	17.1628	1.0
$RANK$	16.3738	1.9785	173.2258	2.0



(a) The power-law graph (b) The random graph (c) The grid

Figure 2: The $\bar{a}_n$ of DPV_0 at different r values.

Algorithm 6 The Voting Vector Modeling

Input: $n, cov, \vec{P}$.
Output: $E_{n\times d}$, where $E[i,j]$ is the j-th vote value on the i-th peer and $d = |\vec{P}|$.

$\vec{D}$:=generateDistribution(cov)
for all $i := 1, \cdots, n$ **do**
  $E[i]$:=generateOneVoteVector($\vec{D}, \vec{P}$)
end for
return E

for these two functions are omitted). Hence, the bigger the value of cov is, the more possible that the entry $\vec{P}'[i]$ with a much bigger distribution value $\vec{D}[i]$ is set by the much bigger entry of $\vec{P}$. As a result, the more possible that the magnitude sequences of the voting vector entries on each peer are the same; and thus, the smaller the degree of difficulty for the voting vectors is. This is the reason why cov is used to describe the degree of difficulty for the input vectors. All the parameters and their values used in the second part experiment are listed in Table 6.

Table 6: The Experimental Parameters

Parameter Name	Values
The graph type	power-law graph, random graph, grid
The number of nodes	$\{500, 1000, 2000, 4000, 8000, 16000\}$
cov	$\{0.1, 0.2, 0.4, 0.8, 1.6, 3.2\}$

The main goal of this experiment is to show that the performance of $\bar{a}_n$ for the DPV_0 algorithm is independent of the network size. These experiments are performed under the assumption that the degree of difficulty for the whole problem remains unchanged with the increase of the network size. The vote modeling method in Algorithm 6 provides a great control over the degree of difficulty for the voting vectors; however, the networks with different topologies generated by BRITE may change the degree of difficulty for the problems with the increase of the network sizes, and it is hard to control the factor of network topology. Due to this reason, not every experimental case supports the conclusion that the $\bar{a}_n$ of the DPV_0 algorithm is independent of the size of the network. However, we did observe some experimental cases as shown in Figure 3. Additionally, the results on the synthetic data also show that DPV_0 significantly outperforms $RANK$ in terms of both $\bar{a}_n$ and t_n.

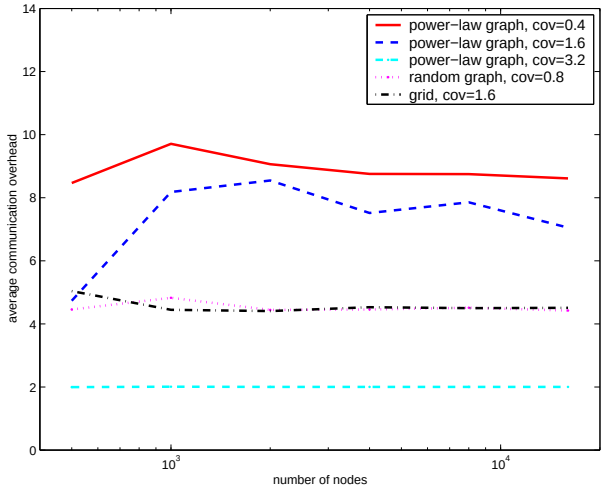


Figure 3: The $\bar{a}_n$ of DPV_0 with respect to networks with different sizes.

Table 7: $\bar{a}_n$ for the Power-law Graph

Algorithm	Avg. Value of $\bar{a}_n$	Avg. Rank of $\bar{a}_n$
DPV_0	4.3421	1.002
$DPV_{0.05}$	6.1273	2.003
$DPV_{0.1}$	8.1940	3.0001
$DPV_{0.15}$	10.7379	3.9993
$DPV_{0.2}$	13.6547	4.9953

5.2.3 The Local Optimality of DPV_0

Finally, in this experiment, we investigate the local optimality of DPV_0 by comparing it with other protocols DPV_e ($e = 0.05, 0.1, 0.15, 0.2$, DPV_e is the DPV protocol with the $\mathcal{C}_e$ as its condition for sending messages). Tables 7

Table 8: $\bar{a}_c$ for the Power-law Graph

Algorithm	Avg. Value of $\bar{a}_c$	Avg. Rank of $\bar{a}_c$
DPV_0	4.3289	1.002
$DPV_{0.05}$	6.0634	2.004
$DPV_{0.1}$	7.8860	3.0025
$DPV_{0.15}$	9.9063	4.001
$DPV_{0.2}$	12.0611	4.991

Table 9: t_n for the Power-law Graph

Algorithm	Avg. Value of t_n (sec)	Avg. Rank of t_n
DPV_0	3.6039	1.5455
$DPV_{0.05}$	4.4269	2.0488
$DPV_{0.1}$	5.5558	2.9698
$DPV_{0.15}$	6.67356	3.929
$DPV_{0.2}$	7.6662	4.507

through 10 record these comparison results for the power-law graph. Tables 7 and 8 show that DPV_0 is significantly more communication-efficient than the other protocols in terms of both $\bar{a}_n$ and $\bar{a}_c$, and the increase of e results in the increase of communication overhead. Table 9 shows that t_n increases as the increase of e, while Table 10 shows that t_c decreases as the increase of e. The above results demonstrate that:

- The increase of the value e leads to the increase of both $\bar{a}_c$ and $\bar{a}_n$ values.
- The communication overhead consumed before t_c is useful, because the increase of $\bar{a}_c$ leads to the decrease of t_c. There is a tradeoff between $\bar{a}_c$ and t_c.
- The communication overhead consumed from t_c to t_n is not very useful, because t_n still increases along the increase of e. However, the communication overhead consumed in this phase is indispensable because each peer cannot perceive its stable state.
- We did observe the case in which $DPV_{0.05}$ is more communication-efficient than DPV_0 in terms of both $\bar{a}_c$ and $\bar{a}_n$. This may indicate that $\mathcal{C}_0$ is a local optimal condition. However, such cases are very rare as shown by the *average ranks* of the the corresponding algorithms.

In general, although $\mathcal{C}_0$ is local optimal, DPV_0 is the most communication-efficient protocol in most cases. For some practical needs, we can select a bigger value of e to decrease t_c (the convergence time) at the cost of the increase of communication overhead.

6. CONCLUSIONS

In this paper, we proposed an ensemble paradigm for distributed classification in P2P networks. Specifically, we formalized a generalized Distributed Plurality Voting (DPV) protocol for P2P networks. The proposed protocol DPV_0 imposes little communication overhead, keeps the *single-site validity* for dynamic networks, and supports the computing modes of both one-shot query and continuous monitoring. Furthermore, we theoretically prove the local optimality of the protocol DPV_0 in terms of communication overhead. Finally, the experimental results showed that DPV_0 significantly outperforms alternative approaches in terms of both average communication overhead and convergence time. Also, the locality of DPV_0 is independent of the network size. As a result, our algorithm can be scaled up to large networks.

Table 10: t_c for the Power-law Graph

Algorithm	Avg. Value of t_c (sec)	Avg. Rank of t_c
DPV_0	3.0134	4.319
$DPV_{0.05}$	2.6001	3.4368
$DPV_{0.1}$	2.3781	2.7778
$DPV_{0.15}$	2.2725	2.4598
$DPV_{0.2}$	2.1487	2.0068

7. ACKNOWLEDGMENTS

This work is supported by the National Science Foundation of China (No. 60435010, 90604017, 60675010), the 863 Project (No.2006AA01Z128), National Basic Research Priorities Programme (No. 2003CB317004) and the Nature Science Foundation of Beijing (No. 4052025). Also, this research was supported in part by a Faculty Research Grant from Rutgers Business School-Newark and New Brunswick.

8. REFERENCES

[1] S. Ali, H. J. Siegel, M. Maheswaran, S. Ali, and D. Hensgen. Task execution time modeling for heterogeneous computing systems. In *Proceedings of the 9th Heterogeneous Computing Workshop*, pages 185–200, 2000.

[2] A.-L. Barabsi and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[3] M. Bawa, A. Gionis, H. Garcia-Molina, and R. Motwani. The price of validity in dynamic networks. In *SIGMOD*, pages 515–526, 2004.

[4] J. Branch, B. Szymanski, C. Gionnella, R. Wolff, and H. Kargupta. In-network outlier detection in wireless sensor networks. In *ICDCS*, 2006.

[5] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[6] L. Breiman. Pasting bites together for prediction in large data sets. *Machine Learning*, 36(2):85–103, 1999.

[7] P. Chan and S. Stolfo. A comparative evaluation of voting and meta-learning on partitioned data. In *ICML*, pages 90–98, 1995.

[8] N. V. Chawla, L. O. Hall, K. W. Bowyer, and W. P. Kegelmeyer. Learning ensembles from bites: A scalable and accurate approach. *Journal of Machine Learning Research*, 5:421–451, 2004.

[9] S. Datta, K. Bhaduri, C. Giannella, R. Wolff, and H. Kargupta. Distributed data mining in peer-to-peer networks. *IEEE Internet Computing special issue on Distributed Data Mining*, 10(4):18–26, 2006.

[10] S. Datta, C. Giannella, and H. Kargupta. K-means clustering over a large, dynamic network. In *SDM*, pages 153–164, 2006.

[11] S. Džeroski and B. Ženko. Is combining classifiers with stacking better than selecting the best one? *Machine Learning*, 54(3):255–273, 2004.

[12] M. Demrekler and H. Altincay. Plurality voting-based multiple classifier systems: statistically independent with respect to dependent classifier sets. *Pattern Recognition*, 35(11):2365–2379, 2002.

[13] J. Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7:1–30, 2006.

[14] P. Flajolet and G. N. Martin. Probabilistic counting. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, pages 76–82, 1983.

[15] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *ICML*, pages 148–156, 1996.

[16] A. Lazarevic and Z. Obradovic. The distributed boosting algorithm. In *KDD*, pages 311–316, 2001.

[17] X. Lin, S. Yacoub, J. Burns, and S. Simske. Performance analysis of pattern classifier combination by plurality voting. *Pattern Recognition Letters*, 24:1959–1969, 2003.

[18] M. Mehyar, D. Spanos, J. Pongsajapan, S. H. Low, and R. M. Murray. Asynchronous distributed averaging on communication networks. *IEEE/ACM Transactions on Networking*, 2007.

[19] C. J. Merz. Using correspondence analysis to combine classifiers. *Machine Learning*, 36(1-2):33–58, 1999.

[20] A. Montresor, M. Jelasity, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems*, 23(3):219–252, 2005.

[21] S. Siersdorfer and S. Sizov. Automatic document organization in a p2p environment. In *ECIR*, pages 265–276, 2006.

[22] L. Todorovski and S. Džeroski. Combining classifiers with meta decision trees. *Machine Learning*, 50(3):223–249, 2003.

[23] G. Tsoumakas, I. Katakis, and I. P. Vlahavas. Effective voting of heterogeneous classifiers. In *ECML*, pages 465–476, 2004.

[24] G. Tsoumakas and I. Vlahavas. Effective stacking of distributed classifiers. In *Proceedings of the 15th European Conference on Artificial Intelligence*, pages 340–344, 2002.

[25] D. Watts and S. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998.

[26] R. Wolff, K. Bhaduri, and H. Kargupta. Local l2 thresholding based data mining in peer-to-peer systems. In *SDM*, pages 430–441, 2006.

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

[28] J. Zhao, R. Govindan, and D. Estrin. Computing aggregates for monitoring wireless sensor networks. In *Proceedings of The First IEEE International Workshop on Sensor Network Protocols and Applications*, 2003.

APPENDIX

Assume a tree $T(U, E)$ with vote vector $P^{\perp u}$ at each $u \in U$. Let P^{uv}, Δ^{uv}, Γ^{uv}, i^{uv}, i^u be as defined in Section 4. For each $uv \in E$, let $[u]_v = \{w \in U : w$ is reachable from u using pathes which includes edge $uv\}$. Finally, for any subset of nodes $S \subseteq U$, let $\Delta_s = \sum_{u \in S} P^{\perp u}$.

LEMMA 1. *In a static tree $T(U, E)$ with static vote vector $P^{\perp u}$ at each $u \in U$, when reaching convergence by checking $\mathcal{C}_0$, for any $u, v \in U$, $\vec{i}^u = \vec{i}^v$.*

PROOF. In the follow we prove that this conclusion holds when $|\vec{i}^u| = 1$. When $|\vec{i}^u| > 1$ the proof is similar.
We prove an equivalent conclusion to this lemma: for any two immediate neighbors $u, v \in E$, if no messages need to be sent between u and v, then $i^u = i^v$.
If u sends Δ^{uv} to v (at the initial time u sends $P^{\perp u}$ to v), then $\Delta^u = \Gamma^u$. At this time we have $i^u = i^{uv}$.
After u sends Δ^{uv} to v, any $P^{wu} (w \in N^u)$ may be updated. Because no messages need be sent to v, this update always keeps the following inequality for any $j = 1, \cdots, d$:

$$\Delta^{uv}[i^{uv}] - \Delta^{uv}[j] \geq P^{uv}[i^{uv}] - P^{uv}[j].$$

Thus,

$$(\Delta^{uv}[i^{uv}] + P^{vu}[i^{uv}]) - (\Delta^{uv}[j] + P^{vu}[j]) \geq$$
$$(P^{uv}[i^{uv}] + P^{vu}[i^{uv}]) - (P^{uv}[j] + P^{vu}[j]),$$

$$\Delta^u[i^{uv}] - \Delta^u[j] \geq \Gamma^u[i^{uv}] - \Gamma^u[j].$$

According to the definition of i^{uv}, $\Gamma^u[i^{uv}] - \Gamma^u[j] > 0$, then $\Delta^u[i^{uv}] - \Delta^u[j] > 0$. Thus, we also have $i^u = i^{uv}$.
With the same method it can be proved that under this situation $i^v = i^{vu}$. Because $i^{uv} = i^{vu}$, the conclusion holds that $i^u = i^v$. □

LEMMA 2. *In a static tree $T(U, E)$ with static vote vector $P^{\perp u}$ at each $u \in U$, when reaching convergence by checking $\mathcal{C}_0$, for any $u \in U$ and $uv \in E$, the following inequality holds for any $j = 1, \cdots, d$, and $i^u \in \vec{i}^u$:*

$$\Delta_{[u]_v}[i^u] - \Delta_{[u]_v}[j] \geq P^{vu}[i^u] - P^{vu}[j].$$

PROOF. **By induction on** $|[u]_v|$.
Base: $|[u]_v| = 1$ which means that $[u]_v = \{v\}$. Hence, $\Delta_{[u]_v} = P^{\perp v} = P^{vu}$. Then,

$$\Delta_{[u]_v}[i^u] - \Delta_{[u]_v}[j] = P^{vu}[i^u] - P^{vu}[j].$$

Step: Assume this lemma holds for $|[u]_v| \leq k$, we will prove that it holds for $|[u]_v| = k + 1$. For any edge $wv \in E$ such that $w \neq u$, $|[v]_w| \leq k$. By the induction hypothesis,

$$\Delta_{[v]_w}[i^v] - \Delta_{[v]_w}[j] \geq P^{wv}[i^v] - P^{wv}[j],$$

$$\sum_{w \neq u}^{wv \in E} (\Delta_{[v]_w}[i^v] - \Delta_{[v]_w}[j]) \geq \sum_{w \neq u}^{wv \in E} (P^{wv}[i^v] - P^{wv}[j]).$$

Adding $P^{\perp v}[i^v] - P^{\perp v}[j]$ to the both sides of the above inequality, then

$$(\sum_{w \neq u}^{wv \in E} (\Delta_{[v]_w}[i^v]) + P^{\perp v}[i^v]) - (\sum_{w \neq u}^{wv \in E} (\Delta_{[v]_w}[j]) + P^{\perp v}[j]) \geq$$
$$(\sum_{w \neq u}^{wv \in E} (P^{wv}[i^v]) + P^{\perp v}[i^v]) - (\sum_{w \neq u}^{wv \in E} (P^{wv}[j]) + P^{\perp v}[j]),$$

which means that

$$\Delta_{[u]_v}[i^v] - \Delta_{[u]_v}[j] \geq \Delta^{vu}[i^v] - \Delta^{vu}[j].$$

Because no messages need to be sent, the following inequality holds by the condition for sending message $\mathcal{C}_0$:

$$\Delta^{vu}[i^v] - \Delta^{vu}[j] \geq P^{vu}[i^v] - P^{vu}[j].$$

According to Lemma 1, upon termination $i^v = i^u$. Hence,

$$\Delta_{[u]_v}[i^u] - \Delta_{[u]_v}[j] \geq P^{vu}[i^u] - P^{vu}[j].$$

□

THEOREM 2. *In a static tree $T(U, E)$ with static vote vector $P^{\perp u}$ at each $u \in U$, when reaching convergence by checking $\mathcal{C}_0$, for all $u \in U$, $\vec{i}^u = arg\ \max_i \Delta_U[i]$.*

PROOF. In the follow we prove that this conclusion holds when $|\vec{i}^u| = 1$. When $|\vec{i}^u| > 1$ the proof is similar.
We add a fictitious node f with $P^{\perp f} = \vec{0}$ to an arbitrary node u. Note that when u need not send a message to f, $\Delta^f = P^{uf} = \Delta^u$. Upon convergence, according to Lemma 2 for any $j = 1, \cdots, d$:

$$\Delta_{[f]_u}[i^f] - \Delta_{[f]_u}[j] \geq P^{uf}[i^f] - P^{uf}[j].$$

According to Lemma 1, $i^f = i^u$ upon convergence. Thus, $P^{uf}[i^f] - P^{uf}[j] = \Delta^u[i^u] - \Delta^u[j] > 0$. It is trivial to see that $[f]_u = U$. Thus, $\Delta_{[f]_u} = \Delta_U$. Then, the following inequality holds for any $j = 1, \cdots, d$:

$$\Delta_U[i^f] - \Delta_U[j] > 0,$$

which means that $i^f = arg\ \max_i \Delta_U[i]$. Then this conclusion holds. □