

# Towards Data Mining in Large and Fully Distributed Peer-to-Peer Overlay Networks

Wojtek Kowalczyk, Márk Jelasity\*, and A. E. Eiben

Free University of Amsterdam, Department of Computer Science,  
Amsterdam, The Netherlands  
{wojtek, jelasity, gusz}@cs.vu.nl

**Abstract.** The Internet, which is becoming a more and more dynamic, extremely heterogeneous network has recently become a platform for huge fully distributed *peer-to-peer overlay networks* containing millions of nodes typically for the purpose of information dissemination and file sharing. This paper targets the problem of analyzing data which are scattered over a such huge and dynamic set of nodes, where each node is storing possibly very little data but where the total amount of data is immense due to the large number of nodes. We present distributed algorithms for effectively calculating basic statistics of data using the recently introduced *newscast model of computation* and we demonstrate how to implement basic data mining algorithms based on these techniques. We will argue that the suggested techniques are efficient, robust and scalable and that they preserve the privacy of data.

## 1 Introduction

With the rapid increase in the number of computers connected to the Internet and the emergence of a range of mobile computational devices which might soon be equipped with mobile IP technology, the Internet is converging to a more dynamic, huge, extremely heterogeneous network which nevertheless provides basic services such as routing and name lookup. This platform is already being used to support huge, fully distributed *peer-to-peer overlay networks* containing millions of nodes typically for the purpose of information dissemination and file sharing [8]. Such fully distributed systems generate immense amounts of data. Analyzing this data can be interesting from both scientific and business purposes. Among other applications, this environment is a natural target for distributed data mining [10].

In this paper we would like to push the concept of distributed data mining to the extreme. The motivations behind distributed data mining include the optimal usage of available computational resources, privacy and dependability by eliminating critical points of service. We will adopt the harshest possible constraints on the distribution of data and the elements of the network and demonstrate techniques which can still provide useful information about the distributed data effectively and dependably.

There are two constraints that we will adopt. The first is that all nodes are allowed to hold as few as one single data instance. This can be viewed as an extremum of horizontal data distribution. The second is another extremum: there is practically no limit on the number of nodes. The only requirement is that in principle each pair of nodes could communicate directly which holds if the nodes are on the Internet with a (not necessarily fixed) IP address.

---

\* Also in RGAI, MTA-SZTE, Szeged, Hungary

Furthermore, we will concentrate on two other very important aspects. The first is data privacy, the second is the dynamic nature of the underlying network: nodes can leave the overlay network and new nodes can join it.

To achieve our goal we will work in the newscast model of computation [6]. This model is built on a lower layer, an epidemic protocol for disseminating information and group membership [5], and it provides a simple interface for applications. The advantage of the model is that due to the robustness and scalability of the epidemic protocol it is built on, the applications of the newscast model of computation inherit this robustness and scalability and can target the kinds of distributed networks described above.

The outline of the paper is as follows. Section 2 will introduce the newscast model of computation at the level necessary to follow the paper. Section 3 will present algorithms for finding the mean value of distributed data. The motivation behind these algorithms is that they can be used for finding more sophisticated statistics (like ratios and empirical probabilities) that can serve as building blocks for decision trees or naive Bayes classification algorithms. Section 4 presents an application of the results of Section 3 for building naive Bayes models over the distributed data. Section 5 concludes the paper.

## 2 The Newscast Model of Computation

The newscast model has been developed as part of the European FP5-IST DREAM project [9]. The newscast model of computation is implemented by a probabilistic epidemic protocol for information and membership dissemination. This protocol provides a dependable, scalable and robust way of maintaining a connected overlay network and disseminating information among its members effectively. Here we do not discuss this protocol since it is not necessary for understanding the paper. The interested reader should consult [6]. Information about related work can be found in [3, 7].

During the discussion of the newscast model of computation we will make further simplifications avoiding the technical details and focusing only on those properties that we apply when developing our algorithms.

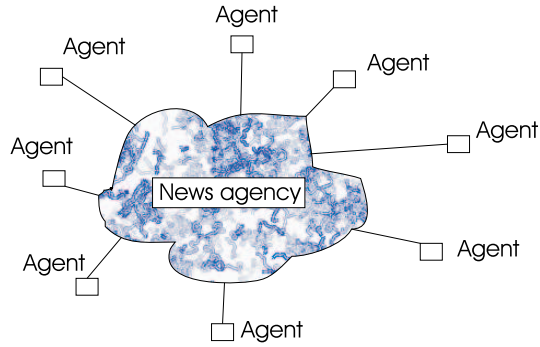
### 2.1 Specifications of the Newscast Model of Computation

The two main concepts of the model are the *collective of agents* and the *news agency*. Computation is performed by the agents that might have their own data storage, processor and I/O facilities. The agents communicate through the news agency (see Figure 1) according to a special schedule which is orchestrated by the news agency. It is very important to stress here that although the news agency plays the role of a server in the model, it is a purely *virtual* entity and the actual implementation of its functionality at the protocol level is a fully distributed peer-to-peer solution.

The communication schedule is organized into *cycles*. In each cycle, the news agency collects exactly one *news item* from all the agents. At the same time for each agent it prepares a fixed-size randomly drawn set of news items collected in the previous cycle, and delivers these sets to the agents. In the first cycle each agent receives an empty set. In the following we will denote the size of these sets by  $c$ . To realize this schedule, all agents must implement a method with the signature

```
NewItem newsUpdate(news[])
```

which is called by the news agency in each cycle. In other words, this method takes the random collection of news items as a parameter and has to return a fresh news item to be disseminated in the next cycle.



**Fig. 1.** Conceptual model of the collective of agents and the news agency.

The required bandwidth for a single agent depends on the size of a news item which is defined by the application at hand. According to the above model the generated traffic in each cycle is the transmission of the  $c$  news items and the transmission of the returned news item over a wide area network.

Even though we do not discuss the protocol here, note that since agents receive only the news content but no information about the sender, the system can stay completely anonymous so privacy is not violated. The actual protocol that implements this model can effectively act as a “remailer”, where the origin of a given item is hard to track down.

## 2.2 An Example: Finding the Maximum of a Set of Numbers

To shed some more light on how to develop applications for the model, we present an easily comprehensible yet interesting example. Let us assume that the collective contains  $n$  agents, and each agent  $i$  knows a single number  $a_i$ . The task is to find the maximum of these numbers  $a^* = \max_{i=1}^n a_i$ . The following two-liner, which will be common to all agents, will solve this problem.

```
NewsItem newsUpdate(news[]) {
    myMax = max(myMax, a, news[1], ..., news[c])
    return myMax;
}
```

where  $a = a_i$  for agent  $i$ .

It is important to note that reading the output of the algorithm is possible for *all* agents, so there is no need for a specific user terminal or service to extract the output. Although there is no signal that informs the agents that the value is found, using the theory of epidemic algorithms [1] it can be proven that all agents will hear about the final solution very quickly. The trick is that from the point of view of a true maximum value the algorithm is in fact an effective broadcasting mechanism, since all agents will keep returning it after they have seen it at least once. So the maximum value spreads exactly like an epidemic, “infecting” a quickly growing number of agents. Let us assume that  $p_i$  is the probability that a given agent is not infected in cycle  $i$ . The probability that a given agent is not infected in cycle  $i + 1$  is given by

$$p_{i+1} = p_i p_i^c$$

since it had to be uninfected in cycle  $i$  and none of its  $c$  samples in the news update must be infective. The initial value  $p_0 = (1 - 1/n)$ . It is clear that  $p_i$  decreases extremely fast.

### 3 Calculating Basic Statistics

Let us consider a system of  $n$  agents that form a newscast network, and let each agent store one number—its own value. Our objective is to program these agents in such a way, that they will collectively find, within very few cycles, the mean of all values (or a good approximation of it). In this section we will present three algorithms for this task: basic averaging (BA), systematic averaging (SA), and cumulative averaging (CA). These algorithms, although based on the same idea, have different properties with respect to convergence speed, accuracy and adaptivity.

The ability of finding the mean is central for implementing some basic data mining algorithms within the newscast framework. In 4 we will demonstrate how the process of finding the mean can be adopted for finding other statistics, like conditional probabilities, information gain, Gini index, etc. – the key elements for building various classification procedures like Naive Bayes and decision trees.

To simplify the statistical analysis of the behavior of our algorithms we will assume that  $c = 2$ , i.e., that news that are distributed by the news agency always consist of 2 news items. It should be noticed that in practice the value of  $c$  is usually much bigger than 2 (e.g., in our experiments we used  $c = 20$ ) which yields much faster convergence rates than our theoretical bounds.

#### 3.1 Basic Averaging

Probably this is the simplest algorithm for finding the mean. During the first cycle (when no news are available) every agent publishes its own value. In this way the news agency gets a copy of all values to be averaged. Next, all agents switch to the “averaging mode”: whenever they receive news they calculate the average of all news items and publish it. More formally, agent’s behavior – the `newsUpdate(news[])` function (where `news[]` refers to the list of news items, each of them being a single number) – is defined as follows:

```
NewsItem newsUpdate(news[]) {
    if (news[] is empty) return own value;
    else return the average of elements in news[];
```

The rationale behind the algorithm is based on the following observation: if we are given a set of numbers and replace two of them by their average then the overall mean will not change, but the variance will decrease. Therefore, in every cycle the news agency receives a collection of numbers that (on average) has the same mean as the mean of the original set, but the variance will be getting smaller and smaller. As a matter of fact, the variance is dropping exponentially fast with the number of cycles: every cycle reduces the variance by factor 2. Indeed, in a single cycle  $n$  pairs of numbers are drawn at random (we assumed  $c = 2$ ), and consequently each pair is averaged. This can be modeled by a random variable  $(X + Y)/2$ , where  $X$  and  $Y$  are independent random variables that take values in  $V$  – the set of values kept by the news agency – with each value having the same chance. Clearly, we have:  $E[(X + Y)/2] = E[X] = E[Y] = E[V]$  and  $Var((X + Y)/2) = Var(X)/4 + Var(Y)/4 = Var(V)/2$ , where  $E[.]$  denotes the expected value (mean) and  $Var(.)$  the variance of a random variable (so we are misusing a bit the notation, as  $V$  is not a random variable).

As said earlier, the newscast model that we are working with is an idealization of the real model that works in a more unpredictable way. In particular, it is not realistic to expect that all the agents get or send their news items simultaneously. But even if the agents acted on news in a sequential way (i.e., instead of processing  $n$  pairs of numbers in one step, the agents would average pairs of numbers one after another), the algorithm still converges to the mean exponentially fast. We will show that after  $k$  iterations of the “averaging operation” the variance drops to  $(1 - 1/n)^k$  of its initial value, thus a single cycle (of  $n$  iterations) reduces it approximately by factor  $e \approx 2.71$ . Let us note that the averaging operator does not change the mean.

**Lemma 1.** *Let  $V$  be a finite set of values and let  $W$  denote a set that is obtained from  $V$  by replacing two randomly selected elements by their averages. Then the variance of  $W$ ,  $Var(W)$ , viewed as a random variable, has the expected value  $E[Var(W)] = (1 - 1/n)Var(V)$ . Consequently, the expected reduction of variance after  $k$  iterations of the averaging operation is  $(1 - 1/n)^k$ .*

**Proof.** Let  $\mu$  denote the mean value of  $V$ ,  $n$  the number of elements of  $V$  and let  $x$  and  $y$  denote two randomly selected elements of  $V$ . We have:

$$\begin{aligned} n(Var(V) - Var(W)) &= (x - \mu)^2 + (y - \mu)^2 - 2((x + y)/2 - \mu)^2 = \\ &= x^2 + y^2 - 0.5(x + y)^2 = 0.5(x - y)^2. \end{aligned}$$

Thus, replacing  $x$  and  $y$  by random variables  $X$  and  $Y$  that model the selection of  $x$  and  $y$ , we have:

$$\begin{aligned} nE[Var(V) - Var(W)] &= 0.5E[(X - Y)^2] = \\ &= 0.5(E[X^2] + E[Y^2] - 2E[X]E[Y]) = E[V^2] - \mu^2 = Var(V), \end{aligned}$$

therefore  $E[Var(W)] = (1 - 1/n)Var(V)$ .

### 3.2 Systematic Averaging

The BA algorithm has one drawback: the lack of adaptivity. Sometimes we would like the system to dynamically adjust the output value (in our case: the estimate of the mean) in response to a changing situation: a modification of agents’ own values, changes of the number of agents that form the network, temporary faults in communication channels, etc. The systematic averaging algorithm achieves adaptivity by constantly propagating agents’ current values and temporal averages through the news agency. Therefore, any change in the incoming data will quickly affect the final result.

Let us fix a small positive integer  $d$ , e.g.,  $d = 15$ , that will control the depth of the propagation process. The SA algorithm works with news items that are vectors of  $d + 1$  numbers. The first element of a news item  $\mathbf{x}$ ,  $x_0$ , will always be an agent’s value (we will call it a 0-order estimate of the mean),  $x_1$  will be the average of two 0-order estimates (we will call it a 1-order estimate),  $\dots$ ,  $x_d$  will be the average of two estimates of order  $d - 1$  (and will be called an estimate of order  $d$ ). In this way consecutive elements of  $\mathbf{x}$  will be “balanced”: they will be averages of  $1, 2, 4, \dots, 2^d$  of original values. Clearly, the result this propagation is represented by  $x_d$ .

The systematic averaging algorithm, when applied to news items  $\mathbf{a}[]$  and  $\mathbf{b}[]$  processes the estimates from left to right:

```

NewsItem NewsUpdate({a[], b[]}){
    create a news item c[d];
    c[0]= current value of the agent
    for (i=1; i<=d; i++)
        c[i]+=(a[i-1]+b[i-1])/2;
    return c[]; }

```

Using the same argument as above we can show that the SA algorithm reduces the variance of the input data exponentially fast. Moreover, the system reacts to changes in the input data within  $d$  iterations.

### 3.3 Cumulative Averaging

Both algorithms, BA and SA, reduce variance exponentially fast. Unfortunately, due to randomness that is involved in the sampling mechanism of the newscast engine, the output values might still be different from the true mean. Our third algorithm, cumulative averaging, CA, solves this problem by running two processes in parallel: in one process agents update their estimates of the mean of the incoming data, in the other one the mean of these estimates is collectively calculated (by BA procedure). More precisely, news items consist of two numbers: the private value of an agent and the current estimate of the mean. Each agent is counting and summing up all incoming private values (first process) and returning the average of the incoming estimates and its own estimate (an average of three numbers). We will leave further implementation details to the reader. The reader can also verify that local estimates of means tend to the true mean (with the increasing number of cycles), so it is guaranteed that the whole algorithm also converges to it.

### 3.4 Experimental Setup

For the purpose of simulation we used the actual newscast model instead of the idealized model presented in the introduction. This is very useful in illustrating that the intuitions and the mathematical analysis based on the idealized model provide a practical approximation when working in the newscast model. To gain experimental data on the behavior of our system we performed runs with various number of agents (10000, 20000, and 50000), and different data sets. For each case we executed 100 independent runs with cache size 20 and terminated after 50 cycles. The data sets included *Gaussian* (where the value of each agent is drawn independently from a Gaussian distribution), *half-half* (where half of the agents hold the value 0, the other half has value 1), and *peak*, where all but one agents has the value 0. In fact, this latter data set is sufficient to show general properties on the "averaging power" of the system.

To prove this statement observe that all the algorithms discussed above use only linear operations on the data and that the algorithms are completely insensitive for the ordering of the agents. Therefore, the output of an algorithm, that is, the estimation  $m$  of the mean, can be identified as

$$m = \sum_{i=1}^n w_i a_i, \quad \sum_{i=1}^n w_i = 1 \quad (1)$$

being the value of an arbitrarily selected agent at termination (in our case after 50 cycles). The weights  $w_i$  ( $i \in \{1, \dots, n\}$ ) stand for the relative impact of agent  $i$  and they are clearly independent from the value set  $a_i$ . In case of perfect averaging the weights should be equal, but because of the stochastic nature of our algorithms they are random numbers with identical distribution. Therefore, it is sufficient to consider one of these weights to

gain information on how well the algorithms calculate averages. To obtain samples on one single weight we define the peak data set as  $a_1 = n, a_i = 0 \ i > 1$ . In this case we get

$$m = \sum_{i=1}^n w_i a_i = w_1 a_1 \quad (2)$$

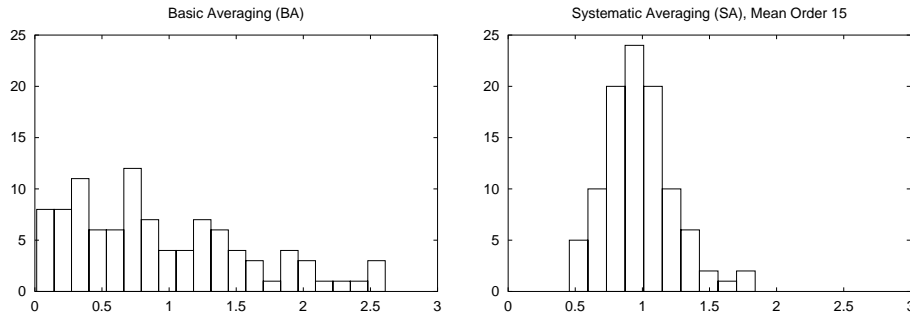
meaning that the true average is  $1/n \cdot n = 1$ . This allows us to draw general conclusions on the "averaging power" of our algorithms solely based on experiments with the peak data set. Furthermore, it is possible to determine the statistical behavior on all other value sets by Equation (1). For example, for an even  $n$  the values  $a_1 = 0, \dots, a_{n/2} = 0, a_{n/2+1} = 1, \dots, a_n = 1$  we have the following formula.

$$m = \sum_{i=1}^n w_i a_i = \sum_{i=n/2+1}^n w_i \quad (3)$$

Here the mean  $m = 1/2$  provided that the algorithm at hand is correct and standard deviation  $\sigma/\sqrt{n/2}$  where  $\sigma$  is the standard deviation of the common weight distribution.

### 3.5 Experimental Results

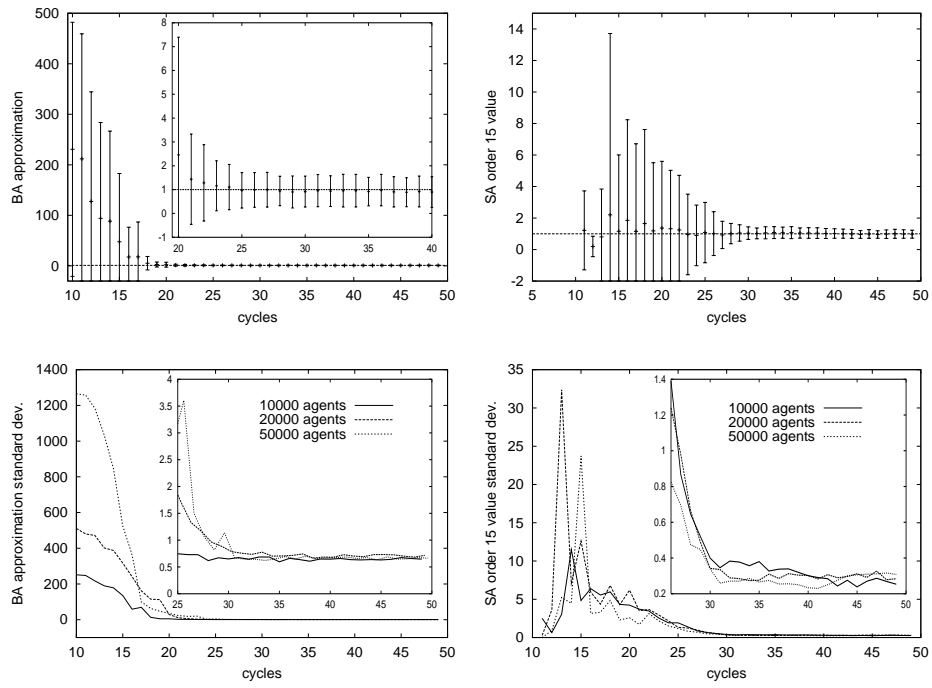
For the reasons explained in the previous section, here we concentrate on results obtained with the "peak" value set. Figure 2 illustrates the output distribution of different algorithms in the 50th cycle. Figure 3 shows the mean and standard deviation of the



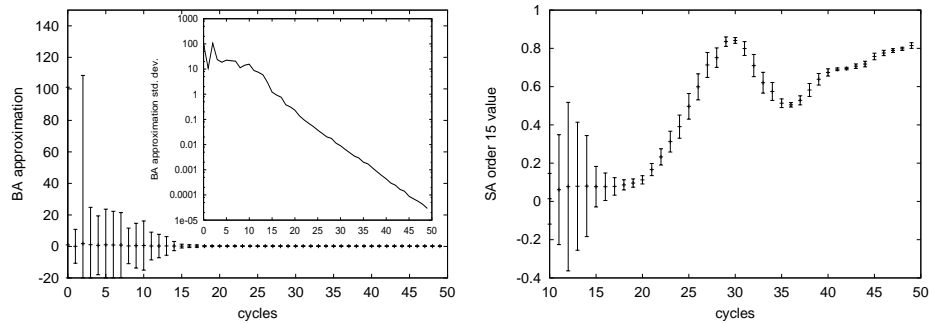
**Fig. 2.** Histograms of the converged output in the 50th cycle collected from 100 independent runs with 10000 agents. The empirical mean and standard deviation is 0.935 and 0.656 for BA and 0.98 and 0.265 for SA respectively.

same output distribution as a function of time (that is, cycles) and also as a function of  $n$ , the number of agents. We can see that in cycle 30 both algorithms stabilize at the final approximation.

All the figures mentioned so far describe statistics over multiple runs from the point of view of a fixed agent which holds  $a_i = n$ . Figure 4 shows statistics of the approximations of the agent during a single run. As it can be seen very clearly, the BA algorithm converges quickly while the SA algorithm is more unstable and some variance is always maintained in the system due to the constant propagation of lower order values. Recall



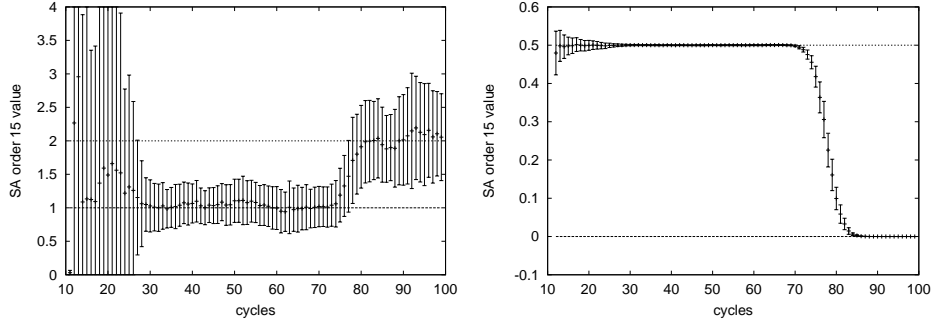
**Fig. 3.** Statistics of the approximation of average of the agent holding  $a_1 = n$  as a function of time (cycles) collected from 100 independent runs. Insets show the tails magnified. The upper plots correspond to runs with 10000 agents.



**Fig. 4.** Statistics collected from a single run with 10000 agents over the approximations of the individual agents.

however that the SA algorithm has a smaller variance when statistics are taken over many independent runs as we have seen above, so in this sense it is more stable. In any case, these results justify our claim that it does not matter which agent we chose to observe the behavior of the system since the variance within the system is relatively minimal.

This feature of constantly propagating lower order values in the SA algorithm comes handy when it comes to dynamic systems. Experiments were conducted using a scenario in which after 50 cycles half of the agents are shut down and we examine what happens with the approximation of average. Figure 5 shows the results. Note that the experiment



**Fig. 5.** Statistics of 50 independent runs with 10000 agents when the agents holding values  $a_i, i > n/2$  are removed in cycle 50. The left plot corresponds to value set  $a_1 = n, a_i = 0 \ i > 1$  and the right plot to the value set  $a_1 = 0, \dots, a_{n/2} = 0, a_{n/2+1} = 1, \dots, a_n = 1$ .

with the values  $a_1 = 0, \dots, a_{n/2} = 0, a_{n/2+1} = 1, \dots, a_n = 1$  results in a much more accurate solution as predicted by the Equation 3. The approximately 30 cycles delay is due to the time necessary to propagate the changed values to the 15th order approximation.

#### 4 An Illustrative Example: Naive Bayes

A central problem in data mining is classification: given some observations  $\mathbf{x}_1, \dots, \mathbf{x}_r$ , represented here by vectors of fixed length  $p$ , with their class labels,  $y_1, \dots, y_r$ , one wants to build a classification procedure that assigns labels to new observations that are not labeled. This classification procedure might have a form of a decision tree, a regression formula, a description of a joint probability distribution, etc., [4]. In this paper we will focus on a very simple, yet powerful, classification procedure called Naive Bayes. Additionally, we will assume that all attributes (vector elements) are discrete and take values in  $V = \{v_1, \dots, v_k\}$ ; class labels are assumed to belong to  $\{c_1, \dots, c_m\}$ .

The Naive Bayes procedure finds  $p(y = c_l | \mathbf{x})$ , for  $l = 1, \dots, m$  with help of some probability estimates that are easy to find. The class with the highest probability is chosen as the label for  $\mathbf{x}$ . Indeed, if we assume that attributes are conditionally independent with respect to the class attribute (it is a naive assumption therefore the name: Naive Bayes), the probabilities  $p(y = c_l | \mathbf{x})$  can be expressed in terms of  $p(x_i = v_j | y = c_l)$  and  $p(y = c_l)$ , for  $i = 1, \dots, p, j = 1, \dots, k$ , and  $l = 1, \dots, m$ , where  $x_i$  denotes the  $i$ -th coordinate of  $\mathbf{x}$  (the value of the  $i$ -th attribute):

$$p(y = c_l | \mathbf{x})p(\mathbf{x}) = p(y = c_l) \prod_{i,j} p(x_i = v_j | y = c_l).$$

The term  $p(\mathbf{x})$  can be eliminated as we know that  $\sum_{l=1}^m p(y = c_l | \mathbf{x}) = 1$ . Clearly, given the data, all the probabilities that we need can be expressed by ratios:

$$p(y = c_l) = \frac{\text{number of observations with label } c_l}{\text{number of all observations}}, \text{ and}$$

$$p(x_i = v_j | y = c_l) = \frac{\text{number of observations with label } c_l \text{ s.t. } x_i = v_j}{\text{number of all observations with label } c_l}.$$

Therefore, to implement the Naive Bayes procedure in the newscast model we only have to know how to calculate ratios of some counts. More precisely, let us consider  $n$  agents that form a newscast network and let each agent store two numbers  $a_i$  and  $b_i$ , for  $i = 1, \dots, n$ . We are interested in estimating the value of  $r = (\sum a_i) / (\sum b_i)$ . Once we know how to determine  $r$  we know how to calculate all the conditional probabilities we need. Fortunately, the ratio  $r$  can be expressed as a combination of two means:  $r = (\sum a_i / n) / (\sum b_i / n)$ . Therefore, any algorithm that was described in the previous section, after a slight modification (we have to estimate several means at the same time), can be immediately used for finding the Naive Bayes classifier for data that is arbitrarily distributed among the agents.

Let us note that most statistics that are used by other classification algorithms are defined in terms of ratios (or probabilities) that have the same form as described above. For example, information gain, gain ratio, Gini index and  $\chi^2$  statistics that are used by decision tree inducers: ID3, C4.5, CART and CHAID, respectively, [4]. Consequently they can be implemented within the newscast framework.

We have implemented the Naive Bayes algorithm using BA as the base averaging method. As we had to maintain estimates of all  $m + mkl$  means, we had to represent news items by vectors of length  $m + mkl$  and to run BA on all coordinates at the same time. To test the accuracy of our algorithm we have generated several data sets with  $m = 2, k = 3, l = 5$ , and the number of records varying between  $10^3$  and  $10^6$ . Next, we produced several Naive Bayes models using a newscast simulator with 5000 nodes and various distributions of the data along the nodes. Finally we compared the results to the ‘‘correct’’ models that were calculated in the conventional way. In most cases, although the model parameters were slightly different, we couldn’t notice any difference in the classification rate. This is exactly what we expected: the Naive Bayes procedure is known to be very robust—usually slight perturbations of model parameters don’t have much influence on the classification rate, [2].

## 5 Summary and Conclusions

The main contribution of this paper is the theoretical and experimental evidence for the feasibility of a novel approach to distributed data mining. The particular type of distributed data mining task we handle constitutes of seeking a model for data spread over a number of sites (here, agents). The challenge is twofold. Firstly, the number of agents can be extremely large (here, up to 50000) and the amount of data per agent can be very small (here, one single value). Secondly, the data might change on-the-fly so the system should be able to adjust the model to these changes automatically. We have reduced the general data mining task to calculating averages demonstrating that it forms the basis for ‘‘real’’ data mining algorithms, such as Naive Bayes or decision trees.

The technical approach we follow is based on the newscast model of computation. We have designed, implemented, and executed algorithms fitting into this model naturally inheriting its main properties: robustness, scalability, and efficiency. For some of these algorithms we have proved theoretical properties on convergence speed and also provided

experimental data to show the systems behavior from various perspectives, such as the “averaging power” (Figures 2 and 3), convergence behavior (Figure 4), and adaptivity in case of changing the data set on-the-fly (Figure 5). Additional experiments with a Naive Bayes algorithm gave further confirmation on the power of our approach.

## References

1. A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database management. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC'87)*, pages 1–12, Vancouver, Aug. 1987. ACM.
2. P. Domingos and M. J. Pazzani. Beyond independence: Conditions for the optimality of the simple bayesian classifier. In *International Conference on Machine Learning*, pages 105–112, 1996.
3. P. T. Eugster, R. Guerraoui, S. B. Handurukande, A.-M. Kermarrec, and P. Kouznetsov. Lightweight probabilistic broadcast. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'01)*, Göteborg, Sweden, 2001.
4. D. Hand, H. Manilla, and P. Smyth. *Principles of Data Mining*. The MIT Press, Cambridge, Massachusetts, London, England, 2001.
5. M. Jelasity, M. Preuß, M. van Steen, and B. Paechter. Maintaining connectivity in a scalable and robust distributed environment. In H. E. Bal, K.-P. Löhner, and A. Reinefeld, editors, *Proceedings of the Second IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2002)*, pages 389–394, Berlin, Germany, 2002. IEEE, IEEE Computer Society.
6. M. Jelasity and M. van Steen. Large-scale newscast computing on the Internet. Technical Report IR-503, Vrije Universiteit Amsterdam, Department of Computer Science, Amsterdam, The Netherlands, Oct. 2002. <http://www.cs.vu.nl/globe/techreps.html>.
7. A.-M. Kermarrec, L. Massoulié, and A. J. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems*, 2003. To appear.
8. D. S. Milojevic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-peer computing. Technical Report HPL-2002-57, HP Laboratories Palo Alto, 2002.
9. B. Paechter, T. Bäck, M. Schoenauer, M. Sebag, A. E. Eiben, J. J. Merelo, and T. C. Fogarty. A distributed resource evolutionary algorithm machine (DREAM). In *Proceedings of the 2000 Congress on Evolutionary Computation (CEC 2000)*, pages 951–958. IEEE, IEEE Press, 2000.
10. B.-H. Park and H. Kargupta. Distributed data mining: Algorithms, systems, and applications. In N. Ye, editor, *The Handbook of Data Mining*. Lawrence Erlbaum Associates, Inc., 2003.