Novel document detection for massive data streams using distributed dictionary learning

S. P. Kasiviswanathan
G. Cong
P. Melville
R. D. Lawrence

Given the high volume of content being generated online, it becomes necessary to employ automated techniques to separate out the documents belonging to novel topics from the background discussion, in a robust and scalable manner (with respect to the size of the document set). We present a solution to this challenge based on sparse coding, in which a stream of documents (where each document is modeled as an m-dimensional vector y) can be used to learn a dictionary matrix A of dimension $m \times k$, such that the documents can be approximately represented by a linear combination of a few columns of A. If a new document cannot be represented with low error as a sparse linear combination of these columns, then this is a strong indicator of novelty of the document. We scale up this approach to handle millions of documents by parallelizing sparse coding and dictionary learning, and by using the alternating-directions method to solve the resulting optimization problems. We conduct our experiments on high-performance computing clusters with differing architectures and evaluate our approach on news streams and streaming data from Twitter®. Based on the analysis, we share our insights on the distributed optimization and machine architecture that can help the design of exascale systems supporting data analytics.

1. Introduction

The high volume of content generated in social media (such as blogs and Twitter**) has propelled such media to the forefront as sources of trending topics and breaking news. In particular, Twitter, with more than 450 million posts (tweets) a day, has often featured news events even before the traditional news outlets. On Twitter, it is possible to find the latest updates on diverse topics, such as natural disasters [1] and the 2009 post-election protests in Iran [2]. Beyond timely news reporting, identifying such emerging topics has many practical applications, such as in marketing, finance, disease control, and national security. The applications for marketing and public relations are particularly compelling, given that 19% of all tweets [3] and 32% of blog posts [4] discuss products or brands. Businesses need to monitor emerging topics in order to stay abreast of what consumers are saying about their products.

Given the vast amount of content being generated online, it becomes necessary to employ automated techniques to detect genuine breaking news that is relevant to the user. The key challenge in automating this process is being able to separate out the documents belonging to novel topics from the voluminous background discussion (noise). The first step in this process is to detect novel documents in a stream of text, where a document is considered novel if it is "unlike" documents seen in the past. Recently, this has been made possible by dictionary learning, which has emerged as a powerful data representation framework. In dictionary learning, each data point y is represented as a sparse linear combination Ax of dictionary columns, where A is the dictionary and x is a sparse vector [5, 6]. A dictionary learning approach can be easily converted into a novel

Digital Object Identifier: 10.1147/JRD.2013.2247232

0018-8646/13/$5.00 © 2013 IBM

document detection method as follows. Let A be a dictionary representing all documents until time $t-1$. For a new data document y arriving at time t, if one does not find a sparse combination x of the dictionary columns such that $y \approx Ax$, then y clearly is not well represented by the dictionary A and is hence novel compared with documents in the past. At the end of timestep t, the dictionary is updated to represent all the documents until time t.

We adopt such an approach in [7], where we present a sequential (batch) dictionary learning approach for detecting novel documents and topics. We used an L_1-penalty on the error $y - Ax$ (instead of squared loss commonly used in the dictionary learning literature), as the L_1-penalty has been found to be more effective for text analysis [8]. Although this dictionary learning approach is conceptually simple and empirically performs very well at detecting novel documents, it is computationally inefficient. As the number of documents increases to hundreds of millions a day, updating the dictionary becomes prohibitive even with efficient optimization techniques.

In order to truly scale to the size of Twitter, here we present a distributed algorithm for novel document detection. This algorithm updates the dictionary in a distributed manner, unlike the algorithm described in [7], which uses a sequential dictionary update technique. To update the dictionary in parallel over N processors, we solve a *global consensus problem*, where a single global variable is updated (in this case, the dictionary A), with the objective term split into N parts and each processor solving one part. We efficiently solve the resulting optimization problems using the Alternating Directions Method of Multipliers (ADMM) technique [9].

We demonstrate the effectiveness of our distributed approach on online news streams as well as Twitter datasets. In order to understand the impact of different machine architectures, we run experiments on two high-performance clusters: a cluster with 512 Intel processors, as well as a Blue Gene*/P cluster with 1,024 processors (more details about specifications of these clusters are provided in Section 5). We experimented with datasets ranging from thousands to millions of documents and show that even with a dataset of 1.2 million tweets, our algorithm takes only 6.8 hours to finish on the 512-processor Intel cluster (more details on datasets and metrics used in the evaluation are provided in Section 5). Our distributed algorithm also scales almost linearly as a function of the number of processors. We find that by doubling the number of processors on the Intel cluster, the algorithm's running time decreases by a factor of 1.7, which is close to the ideal drop factor of 2. In these runs, the computation time dominates the time spent in communication by a factor of 2:1. On the Blue Gene/P cluster, our results are less impressive because of the speed and memory limitations of the Blue Gene/P compute nodes, and this informs us about the design of future architectures that are more suited for this kind of data analytics.

For the news stream dataset, we show that our distributed algorithm is quantitatively better at detecting novel documents than algorithms published previously (including that of [7]). For the unlabeled Twitter dataset, we perform a qualitative evaluation of our algorithm. Despite Twitter data having a low signal-to-noise ratio (i.e., most tweets are banal), which makes the task of identifying "relevant" novel documents particularly challenging, we show that our algorithm (along with some simple additional processing) manages to identify relevant novel documents on the subject of interest.

Related work

Several existing approaches to identify "hot topics" are based on the frequency of mention of terms (i.e., by detecting bursts of activity) [10–13]. Although high-frequency terms may be a good indicator of popularity, they do not identify breaking news. Instead, by comparing the relative frequency of terms and phrases in the current time period to the occurrences in the past, this is likely to identify more topical phrases. Our work differs from the above approaches by going beyond frequency counts, to identifying novel clusters of similar documents, which provides a richer characterization of topics.

Extensive research has been conducted in the areas of document clustering and topic modeling, e.g., the techniques of Latent Dirichlet Allocation [14] and Probabilistic Latent Semantic Analysis [15]. Although clustering and topic modeling techniques (including the dynamic ones) can find sets of posts expressing cohesive patterns of discussion, they are not guaranteed to identify clusters that are also novel or informative compared with previously appearing topics. An alternative approach, known as First Story Detection (FSD) [16], focuses on detecting when a document discusses a previously unseen event. While first story detection by itself is very valuable for broadcast news, it is less effective in social media given the low signal-to-noise ratio. In the space of social media, many posts may be considered as "first stories," solely because they are very different from previous posts. A more detailed discussion of these above techniques can be found in [7], where we also show that a dictionary learning approach outperforms an FSD-based approach for the task of novel document/topic detection.

Notation

Vectors are always column vectors. The transpose of a matrix Z is denoted by $Z^\top$. The norm of matrix used is the entry-wise norm

$$\|Z\|_1 = \sum_{ij} |z_{ij}| \text{ and } \|Z\|_F^2 = \sum_{ij} z_{ij}^2.$$

For arbitrary real matrices, the standard inner product is defined as $\langle Y, Z\rangle = \mathrm{Tr}(Y^\top Z)$, where Tr() stands for trace of a matrix. For a scalar r in $\mathbb{R}$, let $\mathrm{sign}(r)$ be 1 if $r > 0$, -1 if $r < 0$, and 0 if $r = 0$. Define the soft-thresholding operator as

$$\mathrm{soft}(r, T) = \mathrm{sign}(r) \times \max\{|\mathrm{r}| - T, 0\}.$$

The operators *sign* and *soft* are extended to a matrix by applying them to every entry in the matrix. We use 0 to denote a matrix of all zeros (often with a subscript to denote the dimension of the zero matrix).

Organization

In Section 2, we review the background with respect to dictionary learning and show it can be used to detect novel documents. In Section 3, we present our distributed algorithm for novel document detection. We use the technique of ADMM (Alternating Directions Method of Multipliers) to solve the optimization problems. In Section 4, we review the basic ADMM approach and apply it in our setting. In Section 5, we present an experimental evaluation of our approach on different datasets. We conclude in Section 6.

2. Novel document detection using dictionary learning

Dictionary learning background

Dictionary learning concerns the problem of estimating a collection of basis vectors over which a given data collection can be accurately reconstructed, often with sparse encodings. It may be formulated in terms of uncovering low-rank structure in the data using matrix factorizations, possibly with sparsity-inducing priors. It falls into a general category of techniques known as *matrix factorization*. These are also closely related to probabilistic topic models (e.g., probabilistic latent semantic indexing and latent Dirichlet allocation) for textual datasets [14, 15]. Classic dictionary learning techniques [5, 17, 18] for sparse representation (see [6] and references therein) consider a finite training set of signals $P = [p_1, \ldots, p_n] \in \mathbb{R}^{m \times n}$ and optimize the empirical cost function

$$f(A) = \sum_{i=1}^{n} l(p_i, A),$$

where $l(,)$ is a loss function such that $l(p_i, A)$ should be small if A is "good" at representing the signal p_i in a sparse fashion. Here, $A \in \mathbb{R}^{m \times k}$ is referred to as the *dictionary*. The value k is referred to as the size of the dictionary. Most dictionary learning formulations deal with squared loss [5, 6, 18], with a few results on other loss functions such as Kullback-Leibler (KL) divergence [19] and total variation distance [20].

In this paper, we use an L_1-loss function with an L_1-regularization term, and our loss function is

$$l(p_i, A) = \min_x \|p_i - Ax\|_1 + \lambda\|x\|_1,$$

where λ is the regularization parameter. Here, $x \in \mathbb{R}^k$ is known as the *sparse code*. Therefore,

$$\sum_{i=1}^{n} l(p_i, A) = \min_X \|P - AX\|_1 + \lambda\|X\|_1.$$

Here, $X \in \mathbb{R}^{k \times n}$ is the matrix of sparse codes. We define the problem of dictionary learning as that of minimizing the empirical cost $f(A)$. In other words, dictionary learning is the following optimization problem

$$\begin{aligned} \min_A f(A) &= f(A, X) \\ &= \min_{A,X} \sum_{i=1}^{n} l(p_i, A) \\ &= \min_{A,X} \|P - AX\|_1 + \lambda\|X\|_1. \end{aligned}$$

For maintaining interpretability of the results, we would additionally require that the A and X matrices be non-negative. We also add scaling constraints on A. The above optimization problem is, in general, non-convex. However, if one of the variables, either A or X is known, the objective function with respect to the other variable becomes a convex function (in fact, a linear function). A common way to solve the above equation is via alternative minimization, where one variable is fixed and other is updated. This iterative alternative minimization is the core idea behind most algorithms for dictionary learning [5, 6, 18]. However, these algorithms are centralized and need to access the whole of the dataset in every iteration; thus, these algorithms cannot efficiently deal with large datasets.

Novel document detection task

Our problem setup is similar to that published previously [7]. We assume documents arrive in streams. Let

$$\{P_t : P_t \in \mathbb{R}^{m_t \times n_t}, t = 1, 2, 3 \ldots\}$$

denote a sequence of streaming matrices, where each column of P_t represents a document arriving at time t. Here, P_t represents the term-document matrix observed at time t. In order to represent documents in our data stream, we use the conventional vector space model with TF-IDF (Term Frequency-Inverse Document Frequency) term-weighting [21] used in the information retrieval literature. Terms in document are statistically weighted using standard measures *Term Frequency* and *Inverse-Document Frequency*. We use the streaming TF-IDF scheme proposed by Akcora and Bayir [22]. The t may be at any granularity; for example, it could be the day that the document arrives. We use n_t to represent the number of documents arriving at time t. We

normalize P_t such that each column (document) in P_t has a unit L_1-norm. For simplicity in exposition, we will assume that $m_t = m$ for all t. This is without loss of generality because as new documents arrive and new terms are identified, we can expand the vocabulary and zero-pad the previous matrices so that at the current time t, all previous and current documents have a representation over the same vocabulary space.

We use the notation $P_{[t]}$ to denote the term-document matrix obtained by vertically concatenating the matrices $P_1, \ldots, P_t$, i.e., $P_{[t]} = [P_1|P_2|\ldots|P_t]$. Let N_t be the number of documents arriving at time $\leq t$, then $P_{[t]}$ is in $\mathbb{R}^{m \times Nt}$.

Problem goal

Under this setup, the goal of novel document detection is to identify documents in P_t that are "dissimilar" to the documents in $P_{[t-1]}$.

Sparse coding to detect novel documents

Let $A_{t-1} \in \mathbb{R}^{m \times k}$ represent the dictionary matrix after time $t-1$, where dictionary A_{t-1} is a good basis to represent the documents in $P_{[t-1]}$. The exact construction of the dictionary is described later. Now, consider a document y in $\mathbb{R}^m$ appearing at time t. We say that it admits a *sparse* representation over A_{t-1} if y could be "well" approximated as a linear combination of few columns from A_{t-1}. Modeling a vector with such a sparse decomposition is known as *sparse coding*.

In most practical situations, it may not be possible to represent y as $A_{t-1}x$, e.g., if y has new words that are absent in A_{t-1}. In other words, there could be no such x such that $y = A_{t-1}x$. In such cases, one could represent $y = A_{t-1}x + e$, where e is an unknown noise vector. Note that $y = A_{t-1}x + e$ has a trivial solution of $x = 0$ and $e = y$. The error vector e captures new terms introduced in the corpus and/or mismatch in the frequencies of the existing terms. Since there are few such new words, this vector is generally sparse. Our sparse coding formulation

$$l(y, A_{t-1}) = \min_{x \geq 0} \|y - A_{t-1}x\|_1 + \lambda \|x\|_1 \tag{1}$$

naturally takes into account both the error (with the $\|y - A_{t-1}x\|_1$ term) and the complexity of the sparse decomposition (with the $\|x_1\|$ term). It is quite easy to transform Equation (1) into a linear program. Hence, it can be solved using a variety of methods. In our experiments, we use the ADMM [9] to solve Equation (1). ADMM has recently gathered significant attention in the machine learning community because of its wide applicability to a range of learning problems with complex objective functions [9].

We can use sparse coding to detect novel documents as follows. For each document y arriving at time t, we perform the following. First, we solve Equation (1) to check whether y could be well approximated as a sparse linear combination of the columns of A_{t-1}. If the objective value of $l(y, A_{t-1})$ is "big," then we mark the document as *novel*; otherwise, we mark the document as *non-novel*. Because we have normalized all documents in P_t to unit L_l-length, the objective values are in the same scale. Note that this proposed algorithm could have both false positives (documents marked as novel when they are not) and false negatives (documents marked as non-novel when are they are novel), but with a good "quality" dictionary, both of these mistakes could be reduced.

Choice of the penalty function

An L_2-penalty is often imposed on the error. In fact, in the presence of isotopic Gaussian noise the L_2-penalty on $e = y - A_{t-1}x$ gives the x that is the maximum likelihood estimate [23, 24]. However, for text documents, the noise vector e rarely satisfies the Gaussian assumption, as some of its coefficients contain large, impulsive values. For example, in fields such as politics and sports, a certain term may become suddenly dominant in a discussion [7, 8]. In such situations, imposing an L_1-penalty on the error has been shown to perform better than an L_2-penalty (e.g., other work [23–26] has shown that imposing an L_1-penalty leads to more robust and better face-recognition algorithms). Our L_1-formulation is inspired by these above results and a recent result of ours [8], where we show that in the context of novel document detection, imposing L_1-penalty results in a better scheme than imposing an L_2-penalty.

Dictionary update step

First, we require the dictionaries and sparse codes to be non-negative for interpretability purposes. To prevent the dictionary from having arbitrarily large values (which could lead to small values for the sparse codes), we also require that each column of the dictionary have an L_1-norm less than or equal to 1. Let $\mathcal{A}$ be the convex set of matrices defined as

$$\mathcal{A} = \{A \in \mathbb{R}^{m \times k} : A \geq 0, \forall j = 1, \ldots, k \|A_j\|_1 \leq 1\},$$

where A_j is the j-th column of A. (A set of matrices is convex if for any two matrices in the set, its convex combination is also in the set.)

Adding this constraint, the dictionary learning step becomes

$$[A_t, X_{[t]}] = \arg\min_{A \in \mathcal{A}, X \geq 0} \|P_{[t]} - AX\|_1 + \lambda \|X\|_1. \tag{2}$$

Note that for each column i of the dictionary matrix, this projection onto the convex set $\mathcal{A}$ can be done in $O(s_i \log(m))$ time, where s_i is the number of non-zero elements in the i-th column of the dictionary matrix, using the projection onto the L_1-ball algorithm of Duchi et al. [27].

Algorithm 1 Basic scheme for novel document detection using dictionary learning.

At each timestep t

Input: $P_{[t-1]}, P_t = [p_1, \ldots, p_{n_t}]$, A_{t-1}, λ, ζ

Sequential Novel Document Detection Step:
For $j = 1$ to n_t do
Sparse Coding: $x_j = \arg\min_{x \geq 0} \| p_j - A_{t-1}x \|_1 + \lambda \| x \|_1$ (solved using ADMM, Section 4)
If $\| p_j - A_{t-1}x_j \|_1 + \lambda \| x_j \|_1 > \zeta$
Mark p_j as novel

Sequential Dictionary Learning Step:
Set $P_{[t]} = [P_{[t-1]} | p_1, \ldots, p_{n_t}]$
Set $X_{[t]} = \begin{bmatrix} X_{[t-1]} & \\ & x_1, \ldots, x_{n_t} \end{bmatrix}$
Set $A_{t-1} = A_t$
While not converged do
Sparse Coding: $X_{[t]} = \arg\min_{X \geq 0} \| P_{[t]} - A_t X \|_1 + \lambda \| X \|_1$ (solved using ADMM, Section 4)
Dictionary Update: $A_t = \arg\min_{A \in \mathcal{A}} \| P_{[t]} - AX_{[t]} \|_1$ (solved using ADMM, Section 4)

Size of the dictionary

Ideally in our application setting, dynamically changing the size (k) of the dictionary with t would lead to a more efficient and effective sparse coding. However, in our theoretical analysis, we make the simplifying assumption that k is a constant independent of t. The problem of designing an adaptive dictionary whose size automatically increases or decrease over time is an interesting research challenge.

Sequential algorithm for detecting novel documents

Before explaining our distributed approach, we first describe a slightly modified version of the algorithm (**Algorithm 1**), introduced by us in [7] for detecting novel documents. Algorithm 1 at time t performs two steps: (a) sequential novel document detection and (b) sequential dictionary learning. The sequential novel document detection does sparse coding (as described previously) on all documents in P_t with dictionary A_{t-1}, and ζ is the threshold parameter used for deciding whether a document is novel or not. In the sequential dictionary learning step, the algorithm uses an alternative minimization procedure to solve Equation (2). We use the dictionary A_{t-1} as a "warm-start" in the dictionary update step (i.e., the new dictionary is initialized to A_{t-1} before the start of the loop). We use $\Pi_{\mathcal{A}}$ to denote the projection onto the nearest point in the convex set $\mathcal{A}$.

Even though conceptually simple, Algorithm 1 is computationally inefficient. The bottleneck comes in the dictionary learning step. As t increases, so does the size of $P_{[t]}$, so solving Equation (2) becomes prohibitive even with efficient optimization techniques. To achieve computational efficiency, in [7], we solved an approximation of Equation (2) where in the dictionary learning step, we only update the values for A and not X. In particular, define (recursively) $\hat{X}_{[t]} = [\hat{X}_{[t-1]} | x_1, \ldots, x_{n_t}]$ where x_j are coming from the novel document detection step at time t. Then,

Dictionary Learning Step in [7] :

$$A_{t+1} = \arg\min_{A \in \mathcal{A}} \| P_{[t]} - A\hat{X}_{[t]} \|_1 .$$

This leads to faster running times, but because of the approximation, the quality of the dictionary degrades over time, and the performance of the algorithm decreases. In this paper, we propose a new distributed dictionary learning algorithm and show that this distributed algorithm scales up to massive datasets.

3. Distributed algorithm for detecting novel documents

In this section, we present a distributed algorithm for novel document detection. **Algorithm 2** summarizes the distributed procedure. Let N denote the number of processors. We label them $0, \ldots, N-1$. In the following, we assume that all the N processors are identical in terms of computation power, memory, etc. We use the SPMD (single-program, multiple-data) model of computation. In the SPMD model, the task is split and run simultaneously on multiple processors with different inputs in order to obtain faster execution time. For message passing and synchronization, we use MPI (Message Passing Interface).

One of the processors (e.g., processor 0) is called the root processor and plays a special role in the algorithm.

Algorithm 2 Distributed algorithm for novel document detection.

At each timestep t

Input: $P_t = \left[p_1, \ldots, p_{n_t}\right], A_t, \lambda, \zeta$

Initialization Step (done only by the (root) processor 0):

Split P_t into N pieces: $P_t = \left[P_t^{(1)} \middle| \ldots \middle| P_t^{(N-1)} \middle| P_t^{(0)}\right]$, where $P_t^{(i)}$ (for $i = 1, .., N-1$) contains the columns $(i-1) \times \text{floor}(n_t/N) + 1$ to $i \times \text{floor}(n_t/N)$ of P_t, and $P_t^{(0)}$ contains the remaining columns of P_t.

Send $P_t^{(i)}$ to processor i

Distributed Novel Document Detection Step:

Each processor i (in parallel) (for $i = 0, \ldots, N-1$):

For each document (column) p_j in $P_t^{(i)}$ do

Sparse Coding: $x_j = \arg\min_{x \geq 0} \| p_j - A_{t-1}x \|_1 + \lambda \| x \|_1$ (solved using Algorithm 4) (3)

If $\left\| p_j - A_{t-1}x_j \right\|_1 + \lambda \left\| x_j \right\|_1 > \zeta$

Mark p_j as novel

Distributed Dictionary Learning Step:

Each processor i (in parallel) (for $i = 1, \ldots, N$):

Set $P_{[t]}^{(i)} = \left[P_{[t-1]}^{(i)} \middle| P_t^{(i)}\right]$

Set $A_{t-1} = A_t$

While not converged do

Sparse Coding: $X_{[t]}^{(i)} = \arg\min_{X \geq 0} \| P_{[t]}^{(i)} - A_t X \|_1 + \lambda \| X \|_1$ (solved using Algorithm 4) (4)

Dictionary Update: $A_t = \arg\min_{A^{(0)}, \ldots, A^{(N-1)}, A \in \mathcal{A}} \sum_{i=0}^{N-1} \| P_{[t]}^{(i)} - A^{(i)} X_{[t]}^{(i)} \|_1$ (5)

subject to $A^{(i)} - A = 0$ for $i = 0, .., N-1$ (solved using Algorithm 5)

The root processor receives the input P_t and distributes the documents in P_t to all the remaining processors. The load on a processor will depend on the number of documents it obtains. In this paper, we use a simple load-splitting scheme, where each processor obtains approximately n_t/N documents. We denote the sub-matrix of P_t allocated to processor i by $P_t^{(i)}$. Thus, each $P_t^{(i)}$ matrix has approximately n_t/N columns. In the following, we use superscript (i) to denote that the variable is associated with i-th processor.

Distributed novel document detection step

Let us assume that each processor has a copy of A_{t-1}; then, the novel document detection step is simple: each processor uses the sparse coding formulation [Equation (1)] to decide whether any of the documents allocated to it is novel or not (using the same threshold ζ). The dictionary learning step will ensure that after each time t, all the processors have the same dictionary.

Distributed dictionary learning step

In this step, we update the dictionary in a distributed manner. Our dictionary learning formulation requires us to solve

$$[A_t, X_{[t]}] = \arg\min_{A \in \mathcal{A}, X \geq 0} \|P_{[t]} - AX\|_1 + \lambda \|X\|_1.$$

The alternative minimization formulation for solving this above optimization problem is to repeatedly do the following two updates (until convergence):

$$X_{[t]} = \arg\min_{X \geq 0} \|P_{[t]} - A_t X\|_1 + \lambda \|X\|_1 \tag{6}$$

$$A_t = \arg\min_{A \in \mathcal{A}} \|P_{[t]} - AX_{[t]}\|_1. \tag{7}$$

Consider the distributed setting. First construct (recursively), at each processor i,

$$P_{[t]}^{(i)} = \left[P_{[t-1]}^{(i)} \middle| P_t^{(i)}\right].$$

Note that each $P_{[t]}^{(i)}$ is a sub-matrix (contains few columns) of $P_{[t]}$, and, together, $P_{[t]}^{(0)}, \ldots, P_{[t]}^{(N-1)}$ contain all the columns of $P_{[t]}$. Define the following:

$$X_{[t]}^{(i)} = \arg\min_{X \geq 0} \left\| P_{[t]}^{(i)} - A_t X \right\|_1 + \lambda \|X\|_1. \tag{8}$$

It follows that the columns of $X_{[t]}^{(0)}, \ldots, X_{[t]}^{(N-1)}$ together contain all the columns of $X_{[t]}$ (of Equation (6)). Thus, in

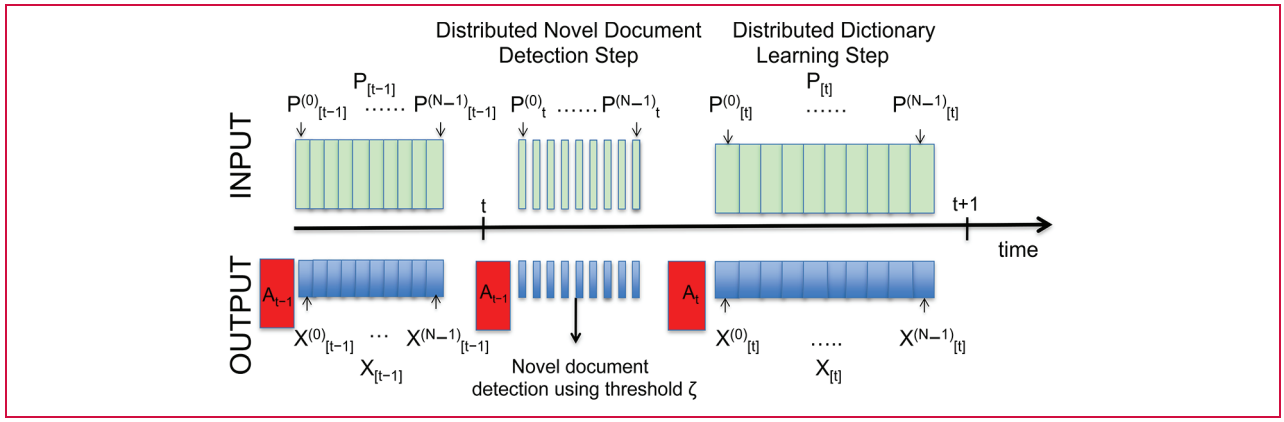

Figure 1

Schematic of distributed novel document detection. At time t, we have dictionary A_{t-1}, and the input is $P_{[t]}$. First, $P_{[t]}$ is divided into N parts and distributed across N processors, where each processor performs (in parallel) a novel document-detection step. After this, the processors together take part in a distributed dictionary update step to generate the new dictionary A_t.

the distributed setting, each processor i solves Equation (8) for updating the sparse codes.

The update for the dictionary is more involved in the distributed setting. Since each processor only has access to $P_{[t]}^{(i)}$ and not the whole of $P_{[t]}$, they cannot solve Equation (7) directly. We need an approach for updating a single global variable (A_t), with the objective term split into N parts and each processor having one part. Solving such a *global consensus* problem is well-studied in the literature [28]. In our setting, the global consensus problem is

$$A_t = \arg\min_{A^{(0)},\ldots,A^{(N-1)},A\in\mathcal{A}} \sum_{i=0}^{N-1} \left\| P_{[t]}^{(i)} - A^{(i)} X_{[t]}^{(i)} \right\|_1$$
$$\text{such that} \quad A^{(i)} - A = 0 \text{ for } i = 0,\ldots,N-1.$$

This ensures that after the dictionary learning step, each processor has the same dictionary. **Figure 1** shows a schematic of our algorithm.

In the next section, we discuss how we could use the method of alternating directions method of multipliers to efficiently solve the optimization problems arising in Algorithm 2. As mentioned earlier, ADMM is a general technique for solving optimization problems that is especially well-suited when the input is large [9].

Alternating directions method of multipliers

We start with a brief review of the general framework of ADMM. Let $p(x) : \mathbb{R}^a \to \mathbb{R}$ and $q(y) : \mathbb{R}^b \to \mathbb{R}$ be two convex functions, $F \in \mathbb{R}^{c\times a}$ and $G \in \mathbb{R}^{c\times b}$, and $z \in \mathbb{R}^c$. Consider the following optimization problem:

$$\min_{x,y} p(x) + q(y) \text{ such that } Fx + Gy = z,$$

where the variable vectors x and y are separate in the objective and are coupled only in the constraint.

Algorithm 3 Basic ADMM scheme.

ADMM Update Equations

Solves: $\min_{x,y} p(x) + q(y)$ such that $Fx + Gy = z$

For $j = 1, 2, \ldots$, convergence

$x_{\{j+1\}} = \arg\min_x L(x, y_{\{j\}}, \rho_{\{j\}})$

$y_{\{j+1\}} = \arg\min_y L(x_{\{j+1\}}, y, \rho_{\{j\}})$

$\rho_{\{j+1\}} = \rho_{\{j\}} + \gamma\varphi(z - Fx_{\{j+1\}} - Gy_{\{j+1\}})$

The augmented Lagrangian [29] for the above problem is given by

$$L(x,y,\rho) = p(x) + q(y) + \rho^\top(z - Fx - Gy) + \frac{\varphi}{2}\|z - Fx - Gy\|_2^2, \quad (9)$$

where $\rho \in \mathbb{R}^c$ is the Lagrangian multiplier, and $\varphi > 0$ is a penalty parameter. ADMM utilizes the separability form of Equation (9) and replaces the joint minimization over x and y with two simpler problems. The ADMM first minimizes L over x, then over y, and then applies a proximal minimization step with respect to the Lagrange multiplier ρ. The entire ADMM procedure is summarized in **Algorithm 3**. The $\gamma > 0$ is a constant. The subscript $\{j\}$ denotes the j-th iteration of the ADMM procedure. The ADMM procedure has been proved to converge to the global optimal solution under quite broad conditions [30].

ADMM equations for sparse coding

Consider the sparse coding formulations arising in Equations (3) and (4) of Algorithm 2. The optimization

Algorithm 4 Solving sparse coding using ADMM.

Solves: $\arg\min_{X\geq 0} \|P-AX\|_1 + \lambda\|X\|_1$

$X_{\{1\}} = 0_{k\times n}$, $E_{\{1\}} = P$, $\rho_{\{1\}} = 0_{m\times n}$

For $j = 1, 2, \ldots,$ convergence

$E_{\{j+1\}} = \text{soft}(P - AX_{\{j\}} + \rho_{\{j\}}/\varphi, 1/\varphi)$

$G = A^\top(AX_{\{j\}} + E_{\{j+1\}} - P - \rho_{(i)}/\varphi)$

$X_{\{j+1\}} = \max\{X_{\{j\}} - \kappa G - (\lambda\kappa)/\varphi, 0\}$

$\rho_{\{j+1\}} = \rho_{(j)} + \gamma\varphi(P - AX_{\{j+1\}} - E_{\{j+1\}})$

Return X at convergence

problem (ignoring subscripts) arising in these equations can be written as follows:

$$\min_{X\geq 0} \|P - AX\|_1 + \lambda\|X\|_1$$
$$\text{with the matrices } P \in \mathbb{R}^{m\times n}, A \in \mathbb{R}^{m\times k}, \text{ and } X \in \mathbb{R}^{k\times n}. \quad (10)$$

The above optimization problem could be rewritten as

$$\min_{X\geq 0, E} \|E\|_1 + \lambda\|X\|_1 \text{ subject to } E = P - AX.$$

The augmented Lagrangian for this optimization problem is

$$L(A, E, \Delta) = \min_{X\geq 0, E} \|E\|_1 + \lambda\|X\|_1 + \langle\Delta, P - AX - E\rangle + \frac{\varphi}{2}\|P - AX - E\|_F^2.$$

It is quite easy to adapt the ADMM updates outlined in Algorithm 3 to update values for X. The entire derivation is presented in [7], and we are reproducing them here (**Algorithm 4**) for completeness. Here, φ, κ, and γ are parameters for the algorithm. The following theorem from [7] proves the convergence of Algorithm 4 for an appropriate setting of the parameters.

Theorem 1 [7]: Let $\kappa, \gamma > 0$ satisfy $\kappa\Psi_{\max}(A) + \gamma < 2$, where $\Psi_{\max}(A)$ is the maximum eigenvalue of $A^\top A$. Then, for any fixed $\varphi > 0$, the sequence $(E_{\{j\}}, X_{\{j\}}, \rho_{\{j\}})$ generated by Algorithm 4 starting from any $(E_{\{1\}}, X_{\{1\}}, \rho_{\{1\}})$ converges to $(\tilde{E}, \tilde{X}, \tilde{\rho})$, where $(\tilde{E}, \tilde{X}, \tilde{\rho})$ is the optimum solution to Equation (10).

ADMM equations for dictionary update

Consider the dictionary update formulations arising in Equation (5) of Algorithm 2. This optimization problem (ignoring subscript t) is of the following form:

$$\min_{A^{(0)},\ldots,A^{(N-1)},A\in\mathcal{A}} \sum_{i=0}^{N-1} \|P^{(i)} - A^{(i)} - X^{(i)}\|_1 \text{ subject to } A^{(i)} - A = 0 \text{ for } i = 0, \ldots, N-1.$$

This is known as the global consensus problem, because the constraint is that all the local variables $(A^{(i)})$ should agree, i.e., be equal. ADMM for Equation (5) can be derived directly from the augmented Lagrangian

$$L\left(A^{(0)}, .., A^{(N-1)}, A, \Delta\right) = \arg\min_{A^{(0)},\ldots,A^{(N-1)},A\in\mathcal{A}} \sum_{i=0}^{N-1} \|P^{(i)} - A^{(i)} - X^{(i)}\|_1 + \left\langle\Delta, A^{(i)} - A\right\rangle + \frac{\varphi}{2}\|A^{(i)} - A\|_F^2$$

The resulting ADMM equations are summarized in **Algorithm 5**. The second and the third steps of the loop in the algorithm are simple closed form updates. The first and third step of the loop are performed independently for each $i = 0, \ldots, N-1$ on a different processor. We now concentrate on solving the first step (Equation (11) of Algorithm 5). Define

$$A^{(i)}_{\{j+1\}} = \arg\min_{B\in\mathcal{A}} \|P^{(i)} - BX^{(i)}\|_1 + \left\langle\Delta^{(i)}_{\{j\}}, B - A_{\{j\}}\right\rangle + \frac{\varphi}{2}\|B - A_{\{j\}}\|_F^2.$$

The above optimization problem could be equivalently written as follows:

$$A^{(i)}_{\{j+1\}} = \arg\min_{B\in\mathcal{A}} \left\|\left(P^{(i)}\right)^\top - \left(X^{(i)}\right)^\top B^\top\right\|_1 + \left\langle\Delta^{(i)}_{\{j\}}, B - A_{\{j\}}\right\rangle + \frac{\varphi}{2}\|B - A_{\{j\}}\|_F^2,$$

which in turn is equivalent to

$$A^{(i)}_{\{j+1\}} = \arg\min_{B\in\mathcal{A}} \left\|\left(P^{(i)}\right)^\top - \left(X^{(i)}\right)^\top B^\top\right\|_1 + \frac{\varphi}{2}\left\|B - A_{\{j\}} + \frac{\Delta^{(i)}_{\{j\}}}{\varphi}\right\|_F^2.$$

Substituting $H = (P^{(i)})^\top - (X^{(i)})^\top B^\top$, we obtain

$$A^{(i)}_{\{j+1\}} = \arg\min_{H,B\in\mathcal{A}} \|H\|_1 + \frac{\varphi}{2}\left\|B - A_{\{j\}} + \frac{\Delta^{(i)}_{\{j\}}}{\varphi}\right\|_F^2 \text{ subject to } H = \left(P^{(i)}\right)^\top - \left(X^{(i)}\right)^\top B^\top.$$

The augmented Lagrangian $L(H, B, \Delta_1)$ of the optimization problem is

$$\min_{H,B\in\mathcal{A}} \|H\|_1 + \frac{\varphi}{2}\left\|B - A_{\{j\}} + \frac{\Delta^{(i)}_{\{j\}}}{\varphi}\right\|_F^2 + \left\langle\Delta_1, \left(P^{(i)}\right)^\top - \left(X^{(i)}\right)^\top B^\top - H\right\rangle + \frac{\varphi_1}{2}\left\|\left(P^{(i)}\right)^\top \left(X^{(i)}\right)^\top B^\top - H\right\|_F^2.$$

We can now apply Algorithm 3 to solve this augmented Lagrangian. First, keeping B and Δ_1 fixed, the H that

Algorithm 5 Distributed dictionary update using ADMM.

Solves: $\arg\min_{A^{(0)},\ldots,A^{(N-1)},A\in\mathcal{A}} \sum_{i=1}^{N} \| P^{(i)} - A^{(i)}X^{(i)} \|_1$ subject to $A^{(i)} - A = 0$ for $i = 0,..,N-1$

$A_{\{1\}} = 0$, $\Delta^{(i)}_{\{1\}} = 0$ for all $i \in \{0,..,N-1\}$

For $j = 1, 2, \ldots,$ convergence

$$A^{(i)}_{\{j+1\}} = \arg\min_{B\in\mathcal{A}} \| P^{(i)} - BX^{(i)} \|_1 + \langle \Delta^{(i)}_{\{j\}}, B - A_{\{j\}} \rangle + \frac{\varphi}{2} \| B - A_{\{j\}} \|^2_F \quad \text{(see below)} \quad (11)$$

$$A_{\{j+1\}} = \frac{1}{N} \sum_{i=0}^{N-1} \left(A^{(i)}_{\{j+1\}} + \frac{1}{\varphi} \Delta^{(i)}_{\{j\}} \right)$$

$$\Delta^{(i)}_{\{j+1\}} = \Delta^{(i)}_{\{j\}} + \varphi (A^{(i)}_{\{j+1\}} - A_{\{j+1\}})$$

Return $A_{\{j+1\}}$ at convergence

Solves: $\arg\min_{B\in\mathcal{A}} \| P^{(i)} - BX^{(i)} \|_1 + \langle \Delta^{(i)}_{\{j\}}, B - A_{\{j\}} \rangle + \frac{\varphi}{2} \| B - A_{\{j\}} \|^2_F$

$\Delta_{1_{\{r\}}} = 0$

For $r = 1, 2, \ldots,$ convergence

$$H_{\{r+1\}} = \text{soft}((P^{(i)})^\top - (X^{(i)})^\top B^\top_{\{r\}} + \frac{\Delta_{1_{\{r\}}}}{\varphi_1}, \frac{1}{\varphi_1})$$

$$B_{\{r+1\}} = \Pi_{\mathcal{A}}((\varphi I_{m\times m} + \varphi_1 X^{(i)}(X^{(i)})^\top)^{-1}(\varphi_1 (X^{(i)})T + \varphi S))$$

where $S = A_{\{j\}}{}^\top - (\Delta^{(i)}_{\{j\}})^\top/\varphi$ and $T = (P^{(i)})^\top - H_{\{r+1\}} + \Delta_{1_{\{r\}}}/\varphi_1$

$$\Delta_{1_{\{r+1\}}} = \Delta_{1_{\{r\}}} + \gamma\varphi_1((P^{(i)})^\top - (X^{(i)})^\top B^\top_{\{r\}} - H_{\{r\}})$$

Return $B_{\{r+1\}}$ at convergence

---

minimizes $L(H, B, \Delta_1)$ could be obtained by solving

$$\arg\min_H \|H\|_1 + \left\langle \Delta_1, \left(P^{(i)}\right)^\top - \left(X^{(i)}\right)^\top B^\top \right\rangle + \frac{\varphi_1}{2} \left\| \left(P^{(i)}\right)^\top - \left(X^{(i)}\right)^\top B^\top - H \right\|^2_F$$

$$\equiv \arg\min_H \|H\|_1 + \frac{\varphi_1}{2} \left\| \left(P^{(i)}\right)^\top - \left(X^{(i)}\right)^\top B^\top + \frac{\Delta_1}{\varphi_1} \right\|^2_F$$

The H that minimizes this optimization problem has a closed form update:

$$\text{soft}\left(\left(P^{(i)}\right)^\top - \left(X^{(i)}\right)^\top B^\top + \frac{\Delta_1}{\varphi_1}, \frac{1}{\varphi_1} \right)$$

Now, keeping H and Δ_1 fixed, the B that minimizes $L(H, B, \Delta_1)$ could be obtained by solving

$$\arg\min_{B\in\mathcal{A}} \frac{\varphi}{2} \left\| B - A_{\{j\}} + \frac{\Delta^{(i)}_{\{j\}}}{\varphi} \right\|^2_F + \left\langle \Delta_1 \left(P^{(i)}\right)^\top - \left(X^{(i)}\right)^\top B^\top - H \right\rangle + \frac{\varphi_1}{2} \left\| \left(P^{(i)}\right)^\top - \left(X^{(i)}\right)^\top B^\top - H \right\|^2_F$$

$$\equiv \arg\min_{B\in\mathcal{A}} \frac{\varphi}{2} \left\| B^\top - A^\top_{\{j\}} + \frac{\left(\Delta^{(i)}_{\{j\}}\right)^\top}{\varphi} \right\|^2_F + \frac{\varphi_1}{2} \left\| \left(P^{(i)}\right)^\top - \left(X^{(i)}\right)^\top B^\top - H + \frac{\Delta_1}{\varphi_1} \right\|^2_F.$$

Let us denote $A^\top_{\{j\}} - (\Delta^{(i)}_{\{j\}})^\top/\varphi = S$ and $(P^{(i)})^\top - H + \Delta_1/\varphi_1 = T$. Substituting for S and T in the above equation, we obtain the following minimization problem for updating B:

$$\arg\min_{B\in\mathcal{A}} \frac{\varphi}{2} \|B^\top - S\|^2_{\text{F}} + \frac{\varphi_1}{2} \left\| T - \left(X^{(i)}\right)^\top B^\top \right\|^2_F.$$

The above optimization problem is similar to Tikhonov-regularized least squares problem (i.e., ridge regression) [9] and has a closed form update given by:

$$\Pi_{\mathcal{A}} \left(\left(\varphi I_{m\times m} + \varphi_1 X^{(i)} \left(X^{(i)}\right)^\top \right)^{-1} \left(\varphi_1 \left(X^{(i)}\right) T + \varphi S \right) \right),$$

where $\Pi_{\mathcal{A}}$ denotes the projection onto the nearest point in the convex set $\mathcal{A}$, and $I_{m\times m}$ denotes an identity matrix of dimension $m \times m$. Finally, for a fixed H and B, the Δ_1 can be updated as $\Delta_1 + \gamma\varphi_1((P^{(i)})^\top - (X^{(i)})^\top B^\top - H)$.

The entire procedure is summarized in Algorithm 5. Similar to Theorem 1, one could show the convergence

for Algorithm 5. Note that at processor i, each step of the iteration of Algorithm 5 (which is the bottleneck computation in our scheme) takes $O(|P^{(i)}|mk + m^3)$ time, where $|P^{(i)}|$ is the number of documents in $P^{(i)}$. The space requirement at each processor is $O(m|P^{(i)}|)$.

Experimental evaluation

We evaluate the proposed model and algorithm on a traditional topic detection and tracking (TDT) dataset comprising streaming news stories, as well as Twitter datasets collected from internal IBM public relations campaigns. First, we present a quantitative evaluation of our distributed algorithm (Algorithm 2) on the TDT dataset (see the section "Experiments on news streams"). The results show the efficacy of Algorithm 2 in detecting novel documents. We discuss the results on the Twitter dataset in the section "Experiments on Twitter." The Twitter datasets are much larger than the TDT dataset and are more relevant for our target application in social media. However, the Twitter datasets are unlabeled, and therefore, we primarily use them for qualitative evaluation and to understand the scaling behavior of our distributed algorithm.

Implementation details

All reported results are based on a C implementation using MPI for message passing and synchronization (using the MPI_Barrier routine). The main synchronization happens in the loop of Algorithm 5, where we use the MPI_Barrier routine after the first step to ensure that each processor i has finished computing $A^{(i)}_{\{j+1\}}$, before we update $A_{\{j+1\}}$ in the next step. Sparse matrix operations were implemented using a publicly available CSparse package [31]. The total execution time of the algorithm is the time taken for the slowest processor to finish. The measured running times are wall-clock times. We stem the words (e.g., reduce inflected words to their root form) while creating the TF-IDF matrices.

Our implementation uses a simple load-balancing scheme, where we try to assign an almost equal number of documents to each processor. A more involved load-balancing scheme would be to assign documents such that each processor is assigned almost the same number of non-zero entries (this is useful because running times of most sparse matrix operations depend on the number of non-zero entries). Such a scheme would be useful if there are large discrepancies in the number of non-zero entries in document vectors (e.g., when documents have a wide variation in the number of distinct words they contain). However, the datasets we tested upon the participating documents had almost the same number of non-zero entries in their document vectors, at which point both of the above load-balancing schemes perform equally well.

We use a communication scheme based on a binary tree topology for computing the mean $A_{\{j+1\}}$ in Algorithm 5. In this scheme, a processor p having processor l as its left child (l is null for a leaf processor p) and processor r as its right child (r is null for a leaf processor p) does the following:

a. It obtains the values of $A^{(l)}_{\{j+1\}}$,$\Delta^{(l)}_{\{j\}}$ from its left child and the values $A^{(r)}_{\{j+1\}}$, $\Delta^{(r)}_{\{j\}}$ from its right child.
b. It computes the sum $A^{(l)}_{\{j+1\}} + A^{(r)}_{\{j+1\}} + A^{(p)}_{\{j+1\}}(1/\varphi)\Delta^{(l)}_{\{j\}} + (1/\varphi)\Delta^{(r)}_{\{j\}} + (1/\varphi)\Delta^{(p)}_{\{j\}}$ and passes this value to its parent processor (if one exists).

Therefore, processor 0, which is the root of this binary tree, has the sum $\sum_{i=0}^{N-1}(A^{(i)}_{\{j+1\}} + (1/\varphi)\Delta^{(i)}_{\{j\}})$, which when divided by N gives $A_{\{j+1\}}$. The dictionary $A_{\{j+1\}}$ is then broadcasted by processor 0 to all the other processors.

Execution environment

We performed our main experiments on a high-performance cluster running Intel machines available at IBM (which we refer to as the Intel cluster in the following text). Each compute node in the cluster has twelve 2.93-GHz cores and 24 GB of memory. These nodes are connected using an InfiniBand** high-speed interconnect. We also did some evaluation of our distributed algorithm on a Blue Gene/P cluster. Each compute node of the Blue Gene/P cluster contains four PowerPC* 450 processor cores, running at 850 MHz and with 2 GB of memory.

Parameters

The regularization parameter λ is set to 0.1, which yields reasonable sparsities in our experiments. We used fixed ADMM parameters for simplicity: $\varphi = \varphi_1 = 5$, $\gamma = 1.89$, and $\kappa = 1/5$ (these values are set as in [7], where we used empirical experiments to obtain values). We set the size of the dictionary to $k = 200$.

Performance evaluation on labeled datasets

For performance evaluation, we assume that documents in the corpus have been manually identified with a set of topics. For simplicity, we assume that each document is tagged with the single most dominant topic that it associates with, which we call the *true topic* of that document. We call a document y arriving at time t *novel* if the true topic of y has not appeared before the time t. Thus, at time t, given a set of documents, the task of novel document detection is to classify each document as either novel (positive) or non-novel (negative). For evaluating this classification task, we use the standard AUC (area under the receiver operating characteristic curve) measure [21].

Experiments on news streams

Our first dataset is drawn from the National Institute of Standards and Technology (NIST) TDT (TDT 2 [32]) corpus, which consists of news stories in the first half of 1998. In our

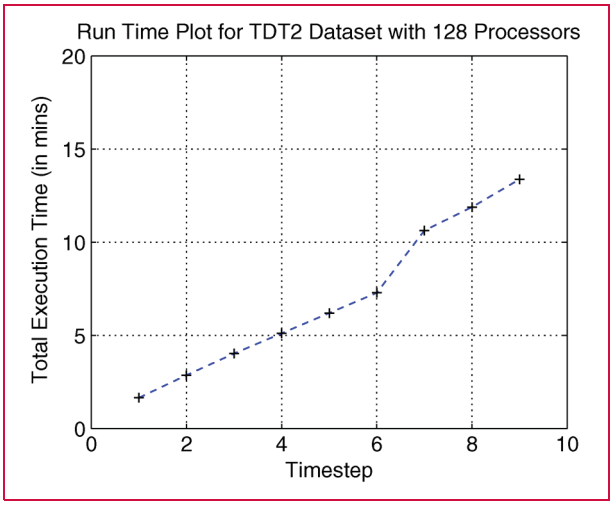

Figure 2

Plot of the running time for Algorithm 2 (our distributed novel document-detection algorithm) on the TDT2 dataset. In each timestep, we add 1,000 more documents to an already existing set of documents. The results were obtained by running the algorithm on an Intel cluster of 128 processors. The total execution time was 13.36 minutes.

evaluation, we used a set of 9,000 documents representing more than 19,528 terms and distributed into the top 30 TDT 2 human-labeled topics over a period of 27 weeks.

The purpose of this evaluation was to measure the efficacy of our distributed algorithm (Algorithm 2) for detecting novel documents. We introduce the documents in groups. At timestep 0, we introduce the first 1,000 documents, and these documents are used for initializing the dictionary. In each subsequent timestep t in $\{1, \ldots, 8\}$, we provide Algorithm 2 with a new set of 1,000 documents, and the "goal" of the algorithm is to predict novel documents in this set of newly introduced documents. In **Figure 2**, we present the running time of Algorithm 2 for this dataset on a 128-processor Intel cluster. Because we are adding 1,000 documents in each timestep t, the running time of the algorithm increases as t increases. The total execution time was 13.36 minutes on this 128-processor Intel cluster. This experiment on a single processor (using Algorithm 1) takes 12.01 hours, so one notices that the distributed Algorithm 2 achieves a massive speedup compared with the centralized Algorithm 1.

In **Table 1**, we present novel document-detection results for those timesteps in which at least one novel document was introduced (this is to ensure that AUC numbers are non-zero). The average AUC achieved by Algorithm 2 is 0.797, which is marginally better than the average AUC (0.788) achieved using the algorithms of [8] on the same dataset. (However, with a better parameter tuning it, should be possible to match up these AUC numbers, as both the sequential and the batch algorithms are solving the same optimization problems.)

Table 1 Area-under-curve (AUC) numbers for the TDT2 dataset for timesteps where novel documents were introduced. The description of the dataset is given in the section "Experiments on news streams."

Timestep	*AUC* (for our distributed algorithm)	*AUC* (for sequential algorithm from [8])
1	0.871157	0.815
2	0.723058	0.704
5	0.765735	0.764
6	0.883541	0.898
8	0.739251	0.760
Average	0.796548	0.788

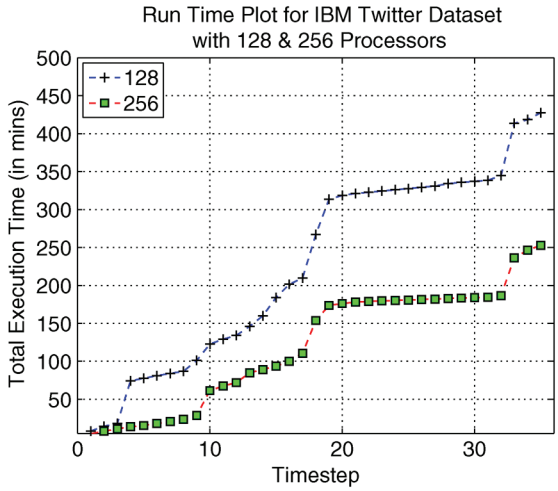

Figure 3

Running time plot of Algorithm 2 (our distributed novel document-detection algorithm) on the IBM Twitter dataset. Each timestep represents a day starting from March 11, 2012, to April 14, 2012. The results are from running the algorithm on an Intel cluster of 128 and 256 processors. The total execution time was 7.12 hours with 128 processors and was 4.21 hours with 256 processors.

Experiments on Twitter

We used several different Twitter datasets. Our first Twitter dataset contained 398,767 tweets collected over a period of 41 days (from March 5, 2012 to April 14, 2012), which was released as part of an IBM Crunch Day competition, during which teams analyzed tweets. The vocabulary has 7,894 unique terms. We use the tweets from March 5 to 10 to initialize the dictionary. Subsequently, at each timestep, we give as input to Algorithm 2 all the tweets from a given day (for a period of 35 days from March 11 to April 14). Since this dataset is unlabeled, we performed a qualitative evaluation

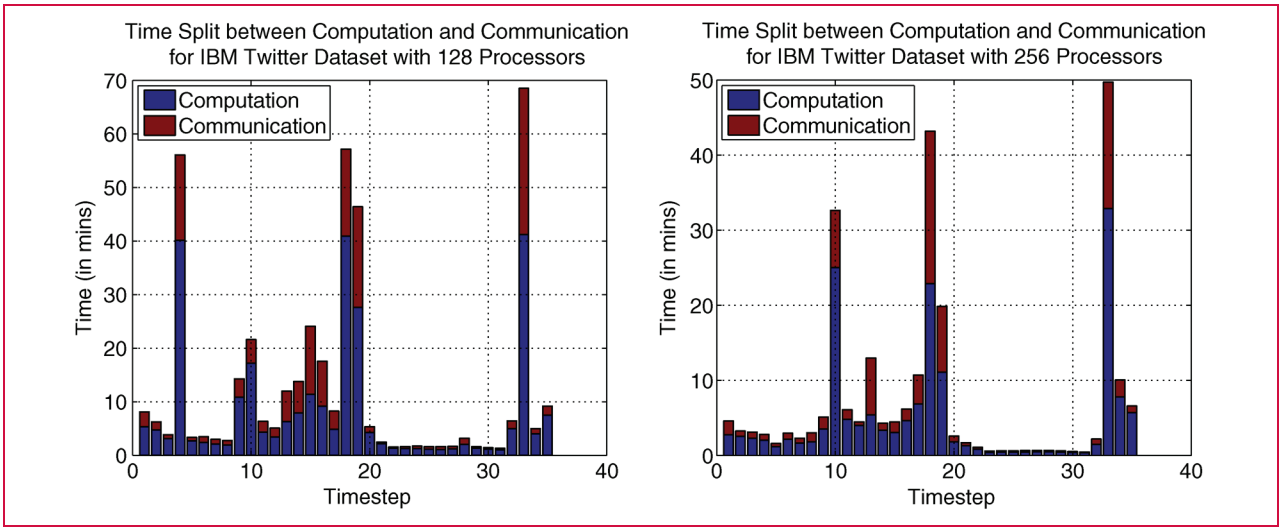

Figure 4

Computation and communication time. The plot on the left shows the time split between computation and communication time for the IBM Twitter dataset execution on a 128-processor (Intel) cluster. The plot on the right shows the same for a 256-processor Intel cluster.

of our distributed novel document-detection approach. We also used this dataset to understand the scaling (as a function of number of processors) of Algorithm 2.

Our second dataset is from an application of monitoring Twitter for marketing and public relations for smartphone and wireless providers. We used the Twitter decahose (a feed of 10% of all tweets, selected at random and delivered in real time) to collect a 10% sample of all tweets (posts) from September 22, 2011, to October 13, 2011. From this, we filtered the tweets relevant to *smartphones* using a scheme presented in [33] that utilizes the Wikipedia** ontology to perform the filtering. Our dataset comprises 1,222,745 tweets over these 22 days with a vocabulary size of 28,428 unique terms. We used this dataset to understand the scaling of Algorithm 2.

Because of the size of these datasets, the centralized Algorithm 1 does not complete in a reasonable time (under 24 hours). Hence, using a distributed algorithm is essential for scaling the novel document detection procedure to massive datasets.

Results on an IBM Twitter dataset

Figure 3 shows the execution time of the algorithm as it steps through the 35 timesteps (each timestep representing one day from March 11 to April 14). For this dataset (containing 398,767 tweets), Algorithm 2 had an overall execution time of 7.12 and 4.21 hours on the Intel cluster with 128 and 256 processors, respectively. Thus, in doubling the number of processors, the execution time decreased by a factor of approximately 1.7. **Figure 4** shows a split of the execution spent in computation and communication. For the 128-processor run, the overall execution time of 7.12 hours was split as 4.72 hours in computation and 2.4 hours in communication. For the 256-processor run, the overall execution time of 4.21 was split as 2.77 hours in computation and 1.43 hours in communication. One notices that in both the cases, the computation time dominates communication time by a factor of almost 2:1.

Figure 5(a) shows the scaling performance of the algorithm for data for a week (containing 41,289 tweets from March 5 to 11, 2012) chosen from the IBM Twitter dataset. (The reason for selecting a sample is to make the evaluation feasible, even on configurations with a small number of processors.) We evaluated the total execution time for this sample for different processor configurations (on the Intel cluster), ranging from 32 to 512 processors. On the 32-processor configuration, the algorithm took 15.51 minutes to complete, whereas with 512 processors, the algorithm took only 2.21 minutes to complete. Overall, as we doubled the number of processors, we noticed that the execution time decreases by approximately 1.69 (on average over all doubling steps).

Table 2 shows a representative set of novel tweets identified by Algorithm 2. In each timestep, instead of thresholding by ζ, we take the top 10% of tweets measured in terms of the sparse coding objective value and run a dictionary-based clustering, described in [7] on it. Further post-processing is performed to discard small clusters and to choose a representative tweet for each cluster. Using this completely automated process, we are able to detect breaking news and trending topics relevant to IBM, such as the launch of the IBM PureSystems* family of products, the

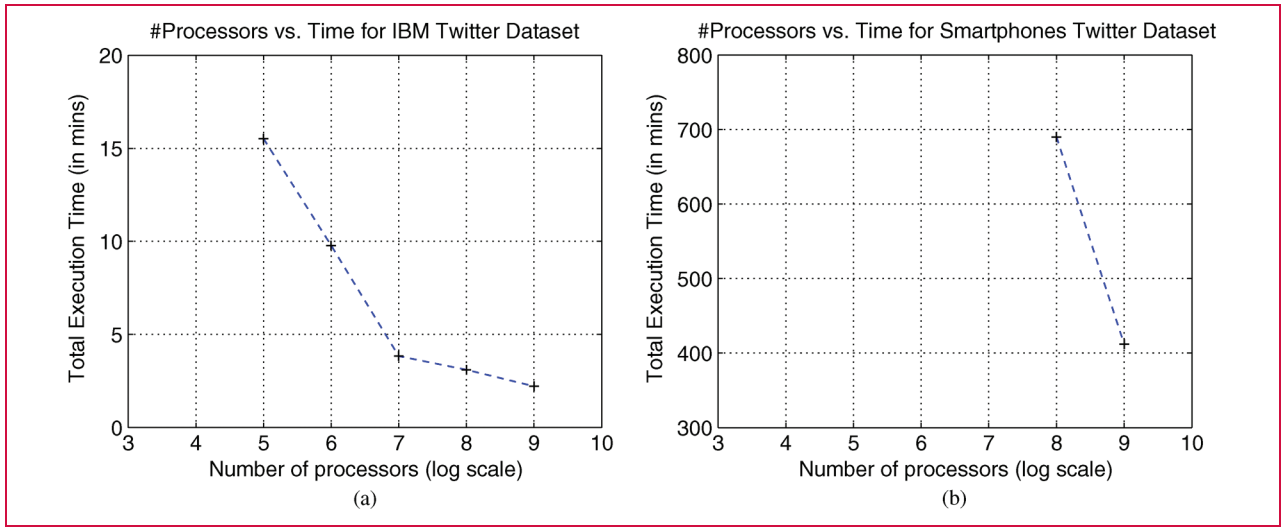

Figure 5

Twitter results. (a) The scaling result on a subset of 41,289 tweets chosen from the IBM Twitter dataset (corresponding to tweets in one week). The *x*-axis is binary log of the number of processors used. (b) The scaling result on the smartphones Twitter dataset, which contains 1,222,745 tweets.

Table 2 Sample novel tweets detected by Algorithm 2 (along with some post-processing) on the IBM Twitter dataset. Note that the algorithm produces novel tweets for each day between March 11 and April 14, but here we only report novel tweets for 3 days.

Date	*Sample novel tweets for IBM Twitter dataset*
March 28, 2012	IBM CEO potential first female Augusta member: If the Augusta National Golf Club keeps with tradition, Ginni Ro…, http://t.co/ij34dicU
	IBM's reinvention should inspire flat pharma businesses, http://t.co/TCnt2iyw
	How will patterns change the way you run your business? Attend the 4/11 broadcast event #expertsystems #ibm, http://t.co/VR1aBuSw
April 05, 2012	IBM partners with Brazil billionaire Batista's EBX: US computer giant IBM has signed a deal…, http://t.co/Rwc7UNDK
	IBM, Microsoft go big with Big Data technology offerings, http://t.co/o30wG2pF
	A thousand years of math history in a new app: IBM rolled out a free iPad app on Thursday that offers a digital…, http://t.co/PuukdI3E
April 11, 2012	Meet Jeff Jonas, the latest IBM Fellow with no college degree, http://t.co/yNI1jW57
	IBM targets rivals with info tech maintenance product: http://t.co/vW2LXuz3
	@IBM and @IBMecosystem set the stage for the next era of simplified computing with #IBMPureSystems: http://t.co/2IWAGqae

popularity of the Minds of Mathematics iPad** app released by IBM, and controversy over sexism in the Augusta national golf club involving the first female chief executive officer of IBM.

Results on smartphones Twitter dataset

Figure 5(b) shows the performance of Algorithm 2 on this dataset. The overall execution time was 11.5 and 6.86 hours on the Intel cluster with 256 and 512 processors, respectively. Because of the size of this dataset, it was not possible to finish the algorithm in a reasonable time on configurations with 128 or smaller number of processors. As with the IBM Twitter dataset, we noticed that as we doubled the number of processors, the execution time decreases by a factor of approximately 1.7. For the 256-processor run, the overall execution time of 11.5 hours was split as 7.48 hours in computation and 4.02 hours in communication. For the 512-processor run, the overall execution time of 6.86 was split as 4.74 hours in computation and 2.12 hours in communication. One again notices that the computation time dominates communication time by a factor of almost 2:1.

Comments about execution on Blue Gene/P cluster and lessons learned

We also studied the performance of Algorithm 2 on the Blue Gene/P cluster. For the smartphones Twitter dataset, the algorithm took 20.85 hours to finish when run on a configuration with 1,024 processors. Therefore, as a platform for running our distributed novel document detection algorithm, Blue Gene/P performs worse than the Intel cluster. There are multiple reasons for this. For example, the processors at each compute node are less powerful in Blue Gene/P than for the Intel cluster. In addition, each compute node in Blue Gene/P has a smaller-sized cache (and memory) than the compute nodes in the Intel cluster, resulting in more time spent on memory fetches. These results hint that a configuration with a smaller number of stronger compute nodes with good memory hierarchy is more suited to our distributed novel document-detection algorithm than a configuration with a larger number of weak compute nodes.

Conclusion

In this paper, we presented a distributed dictionary learning approach to novel document detection. Our experimental evaluations showed that this distributed algorithm can scale up to detect novel documents in large streams of texts. Our implementation is not completely optimized, and it is conceivable that a more efficient implementation of the algorithm would be much faster. Further improvements in the running time could be achieved by using the online dictionary learning algorithms introduced in [8].

There are several directions for future work based on the proposed framework. As mentioned earlier, while the current work uses fixed-sized dictionaries, using adaptive dictionaries, for which the size changes are based on the set of active or emerging topics, may be more desirable in social media applications. Another possible source of improvement may result from using a dual augmented Lagrangian technique in ADMMs, rather the primal augmented Lagrangian technique that we use here, as it is known that the dual augmented Lagrangian performs marginally better than the primal one for L_1/L_1 problems [25]. In addition, apart from the target application of novel document detection, our distributed dictionary learning algorithm could have broader applicability to other tasks using text and beyond, e.g., signal processing.

Acknowledgments

We thank Vikas Sindhwani and Arindam Banerjee for many discussions on the dictionary learning approach for novel document detection, and Stephen Boyd for a discussion on distributed ADMMs.

References

1. M. Mendoza, B. Poblete, and C. Castillo, "Twitter under crisis: Can we trust what we RT?" in *Proc. 1st Workshop Social Media Anal.*, 2010, pp. 71–79.
2. Z. Zhou, R. Bandari, J. Kong, H. Qian, and V. Roychowdhury, "Information resonance on Twitter: Watching Iran," in *Proc. 1st Workshop Social Media Anal.*, 2010, pp. 123–131.
3. B. J. Jansen, M. Zhang, K. Sobel, and A. Chowdury, "Twitter power: Tweets as electronic word of mouth," *J. Amer. Soc. Inf. Sci. Technol.*, vol. 60, no. 11, pp. 2169–2188, Nov. 2009.
4. P. Melville, V. Sindhwani, and R. Lawrence, "Social media analytics: Channeling the power of the blogosphere for marketing insight," in *Proc. Workshop Inf. Netw.*, 2009, pp. 17–21.
5. M. Aharon, M. Elad, and A. Bruckstein, "The K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation," *IEEE Trans. Signal Process.*, vol. 54, no. 11, pp. 4311–4322, Nov. 2006.
6. J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online learning for matrix factorization and sparse coding," *J. Mach. Learn. Res.*, vol. 11, no. 1, pp. 19–60, Mar. 2010.
7. S. P. Kasiviswanathan, P. Melville, A. Banerjee, and V. Sindhwani, "Emerging topic detection using dictionary learning," in *Proc. 21st ACM Int. Conf. Inf. Knowl. Manage.*, 2011, pp. 745–754.
8. S. P. Kasiviswanathan, H. Wang, A. Banerjee, and P. Melville, "Online L1-dictionary learning with application to novel document detection," in *Proc. 26th Annu. Conf. Adv. Neural Inf. Process. Syst.*, 2012. [Online]. Available: http://www-users.cs.umn.edu/~banerjee/papers/12/onlinedict.pdf
9. S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, 2011.
10. J. Kleinberg, "Bursty and hierarchical structure in streams," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2002, pp. 91–101.

11. G. Fung, C. Pui, J. Yu, P. Yu, and H. Lu, "Parameter free bursty events detection in text streams," in *Proc. Int. Conf. Very Large Data Bases*, 2005, pp. 181–192.
12. M. Mathioudakis and N. Koudas, "TwitterMonitor: Trend detection over the Twitter stream," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2010, pp. 1155–1158.
13. A. Angel, N. Sarkas, N. Koudas, and D. Srivastava, "Dense subgraph maintenance under streaming edge weight updates for real-time story identification," *Proc. VLDB Endowment*, vol. 5, no. 6, pp. 574–585, Feb. 2012.
14. D. Blei, A. Ng, and M. Jordan, "Latent Dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Mar. 2003.
15. T. Hofmann, "Probabilistic latent semantic analysis," in *Proc. Conf. Uncertainty Artif. Intell.*, 1999, pp. 289–296.
16. J. Allan, *Topic Detection and Tracking: Event-based Information Organization*. Heidelberg, Germany: Springer-Verlag, 2002.
17. B. Olshausen and D. Field, "Sparse coding with an overcomplete basis set: A strategy employed by V1?" *Vis. Res.*, vol. 37, no. 23, pp. 3311–3325, Dec. 1997.
18. H. Lee, A. Battle, R. Raina, and A. Ng, "Efficient sparse coding algorithms," in *Proc. Neural Inf. Process. Syst.*, 2007, pp. 201–208.
19. D. Lee and H. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, Oct. 1999.
20. Y. Zhang, A. d'Aspremont, and L. E. Ghaoui, "SparsePCA: Convex relaxations, algorithms and applications," in *Handbook on Semidenite, Cone and Polynomial Optimization: Theory, Algorithms, Software and Applications*. New York, NY, USA: Springer-Verlag, 2011.
21. C. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge, U.K.: Cambridge Univ. Press, 2008.
22. C. G. Akcora, A. B. Murat, M. Demirbas, and H. Ferhatosmanoglu, "Identifying breakpoints in public opinion," in *Proc. 1st Workshop Social Media Anal.*, 2010, pp. 62–66.
23. J. Wright, A. Yang, A. Ganesh, S. Sastry, and Y. Ma, "Robust face recognition via sparse representation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 2, pp. 210–227, Feb. 2009.
24. A. Y. Yang, S. S. Sastry, A. Ganesh, and Y. Ma, "Fast L1-minimization algorithms and an application in robust face recognition: A review," in *Proc. Int. Conf. Image Process.*, 2010, pp. 1849–1852.
25. J. Yang and Y. Zhang, "Alternating direction algorithms for L1-problems in compressive sensing," *SIAM J. Sci. Comput.*, vol. 33, no. 1, pp. 250–278, 2011.
26. J. Wright and Y. Ma, "Dense error correction via L1-minimization," *IEEE Trans. Inf. Theory*, vol. 56, no. 7, pp. 3540–3560, Jul. 2010.
27. J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra, "Efficient projections onto the l1-ball for learning in high dimensions," in *Proc. Int. Conf. Mach. Learn.*, 2008, pp. 272–279.
28. D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1989.
29. J. Nocedal and S. J. Wright, *Numerical Optimization*. New York, NY, USA: Springer-Verlag, 1999.
30. P. Combettes and J. Pesquet, "Proximal splitting methods in signal processing," in *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*. New York, NY, USA: Springer-Verlag, 2009, pp. 185–212.
31. T. A. Davis, *Direct Methods for Sparse Linear Systems*. Philadelphia, PA, USA: SIAM, 2006.
32. TDT2 Dataset Description. [Online]. Available: http://www.itl.nist.gov/iad/mig/tests/tdt/1998/
33. V. Chenthamarakshan, P. Melville, V. Sindhwani, and R. D. Lawrence, "Concept labeling: Building text classifiers with minimal supervision," in *Proc. Int. Joint Conf. Artif. Intell.*, 2011, pp. 1225–1230.

Received August 3, 2012; accepted for publication September 14, 2012

Shiva Prasad Kasiviswanathan *GE Global Research Center, Camino Ramon, CA 94583 USA (kasivisw@gmail.com).* Dr. Kasiviswanathan is a Research Scientist in the machine learning group at GE research. Before joining GE, he was a post doc in the machine learning group at the IBM Research - Thomas J. Watson Research Center. His research mainly focuses on machine learning, data privacy, algorithm design, and high-performance computing. He obtained his Ph.D. degree from Pennsylvania State University in 2008. Before joining IBM, he spent several years as a post doc at the Los Alamos National Laboratory. He has published approximately 25 articles in top computer science conferences and journals, including *Symposium on Theory of Computing* (STOC), *Foundations of Computer Science* (FOCS), *Symposium on Discrete Algorithms* (SODA), *International Conference on Machine Learning* (ICML), *Neural Information Processing System* (NIPS), and *International Conference on Knowledge and Management* (CIKM).

Guojing Cong *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (gcong@us.ibm.com).* Dr. Cong is a Research Staff Member at the IBM T. J. Watson Research Center. His research interests are in performance analysis and tuning for high-performance computing systems and in high-performance large-scale graph analysis. He works with colleagues to develop systematic tuning approaches for the Defense Advanced Research Projects Agency (DARPA) High Productivity Computing Systems project. The performance analysis software is used on IBM supercomputers such as the pSeries* clusters and the Blue Gene* systems. Dr. Cong also designed fast parallel algorithms for graph problems on supercomputers. For several fundamental graph problems, he presented the first implementations that achieved almost linear parallel speedups on symmetric multiprocessors and distributed-memory platforms. He publishes extensively in his research areas and holds several patents. Dr. Cong received his Ph.D. degree from the University of New Mexico in 2004. He is a senior member of Institute of Electrical and Electronics Engineers (IEEE).

Prem Melville *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (pmelvil@us.ibm.com).* Dr. Melville received his Ph.D. degree in computer science at the University of Texas at Austin and is currently a Research Scientist in the Machine Learning Group at IBM Research. His research interests lie in machine learning and data mining. More specifically, he has worked on active learning, ensemble methods, active feature-value acquisition, sentiment analysis, dual supervision, rank aggregation, recommender systems, text classification, and applications of data mining to social media analytics, business analytics, and e-commerce. Dr. Melville has served on the organizing committees of International Conference on Knowledge and Management (CIKM) 2012, International Conference on Machine Learning (ICML) 2011, Knowledge Discovery and Data Mining (KDD) 2010, and Web Search and Data Mining (WSDM) 2010. He also serves on the Editorial Board of *Data Mining and Knowledge Discovery*. He co-organized the first workshop on Budgeted Learning at International Conference on Machine Learning 2010, and the first workshop on Social Media Analytics (SOMA 2010) at KDD, and for Mining and Learning with Graphs (MLG 2011). Together with his colleagues, he received the Best Application Paper Award at KDD 2010 and has won KDD Cup 2009, KDD Cup 2008, and the INFORMS Data Mining Contest 2008.

Richard D. Lawrence *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (ricklawr@us.ibm.com).* Dr. Lawrence is a Research Staff Member and Manager, Machine Learning, in the Business Analytics and Mathematical Science organization within IBM Research. He received his B.S. degree from Stanford University in chemical engineering and his Ph.D. degree from University of Illinois in nuclear engineering. Dr. Lawrence joined IBM Research in 1990 and has managed research efforts spanning parallel computing, applied optimization, predictive modeling, and machine learning. He has received seven IBM Outstanding Technical Achievement Awards for his work in scalable data mining, customer modeling, and social media analytics, and he has authored more than 60 technical papers in these research areas.