Noname manuscript No.
(will be inserted by the editor)

Consensus-Based Large-Scale Selection of Features

Haimonti Dutta · Ashwin Srinivasan

Received: date / Accepted: date

Abstract Insert your abstract here. Include keywords, PACS and mathematical subject classification numbers as needed.

Keywords First keyword · Second keyword · More

1 Introduction

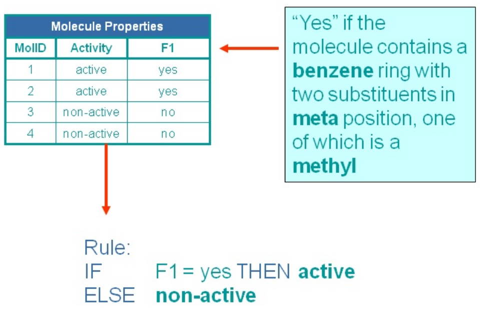
The field of Inductive Logic Programming (ILP) has made steady progress over the past two decades, in advancing the theory, implementation and application of logic-based relational learning [**?**]. A characteristic of this form of machine-learning is that data, prior knowledge and hypotheses are usually—but not always—expressed in a subset of first-order logic, namely logic programs. Side-stepping for the moment the question "why logic programs?", it is evident that settling on some variant of first-order logic allows the construction of tools that enable the automatic construction of descriptions that use relations (used here in the formal sense of a truth value assignment to n-tuples). There are at least two kinds of tasks where some form of relational learning would appear to be necessary:

Haimonti Dutta
The Center for Computational Learning Systems (CCLS)
Columbia University
New York, NY 10115.
E-mail: haimonti@ccls.columbia.edu

Ashwin Srinivasan
Department of Computer Science
IIIT, Delhi.
E-mail: ashwin@iiitd.ac.in

(a) The construction of models of systems that explicitly require relations. Examples are graphical models, grammars, recursive definitions, and so on.
(b) The identification of functions (again used formally, in the sense of being a uniquely defined relation) whose domain is the set of instances in the data. An example is the construction of new "features" for data analysis based on existing relations ("$f(m) = y$ if a molecule m has 3 or more benzene rings fused together otherwise $f(m) = n$").

Feature-construction in (b) would seem to be no more than just a special form of relation-learning as decribed in (a) above. In principle, this is correct. However, the features are not intended to constitute a stand-alone description of a system's structure. Instead, their purpose is to enable different kinds of data analysis to be performed better. These may be constructing models for discrimination, joint probability distributions, forecasting, clustering, and so on. If a logic-based relational learner like an ILP engine is used to construct these relational features, then each feature is formulated as a logical formula. A measure of comprehensibility will be retained in the resulting models that use these features (see Fig. 1).



Molecule Properties

MolID	Activity	F1
1	active	yes
2	active	yes
3	non-active	no
4	non-active	no

"Yes" if the molecule contains a **benzene** ring with two substituents in **meta** position, one of which is a **methyl**

Rule:
IF F1 = yes THEN **active**
ELSE **non-active**

Fig. 1 Feature discovery with relational learning. If we knew feature F1, it is easy to construct a model for active molecules using any machine learning program (rule at the bottom). What we are talking about here is discovering the definition of F1 (box on the right), given relational descriptions of the molecules m1–m4. Once done, we may be able to construct better models for the data.

The approach usually, but not always, separates relational learning (to discover features) and modelling (to build models using these features). There will of course be problems that require the joint identification of relational with features and models—the emerging area of statistical relational learning (SRL), for example, deals with the conceptual and implementation issues that arise in the joint estimation of statistical parameters and relational models. It would appear that separate construction of features and statistical models would represent no more than a poor man's SRL. Nevertheless, there is now a growing body of research that suggests that augmenting any existing features with ILP-constructed relational ones can substantially improve the predictive power of a statistical model [**?**]. There are thus very good practical reasons to persist with this variant of statistical and logical learning for data analysis.

There are however shortcomings with the approach which limit its applicability to large-scale problems. First, the set possible relational features is usually not finite This has led to an emphasis on syntactic and semantic restrictions constraining the features to some finite set. In practice this set is still very large, and it is intractable to identify an optimal subset of features. ILP engines for feature-construction therefore employ some form of heuristic search, with no specific guarantees associated with the result. Second, much still needs to be done to scale ILP-based feature discovery up to meet modern data and model requirements. This includes the abilities to discover features using very large datasets not all stored in one place, and only in secondary memory; from relational data arriving in a streaming mannery; and from data which do not conform easily to expected patterns.

This paper addresses some of these limitations: not in full measure, but for a class of models and loss-function. Specifically, we describe how a network of computational nodes can be used to construct optimal (or near-optimal) linear models for discrimination. Each node has access to some (but not all) relational features constructed by an ILP engine. Using a simple consensus-based algorithm, all nodes in the network converge on the optimal weights for all the features. The setting is naturally amenable to parallelisation, and empirical results obtained using the approach are encouraging. The rest of the paper is organised as follows. Section 2 shows an example of what we expect to achieve, using a simple synthetic dataset. Section 4 is a short introduction to ILP and its use in constructing features. Section 5 formulates the problem as one of consensus-based model construction. Section 6 presents an iterative procedure for constructing models in a network of nodes capable of exchanging information about their local models. The section includes a proof of convergence and termination for linear models. The algorithm is evaluated empirically on a number of standard ILP benchmarks in Section 7. Section **??** describes related work and open issues. Section **??** concludes the paper.

2 Consensus-based Feature Selection for ILP Applications

We motivate the task of consensus-based feature selection using the train spotting problem, originally proposed by Ryzhard Michalski [22]. Given two sets of trains: eastbound and westbound, the problem is to determine decision rules to distinguish between train sets, using properties of the carriages and load

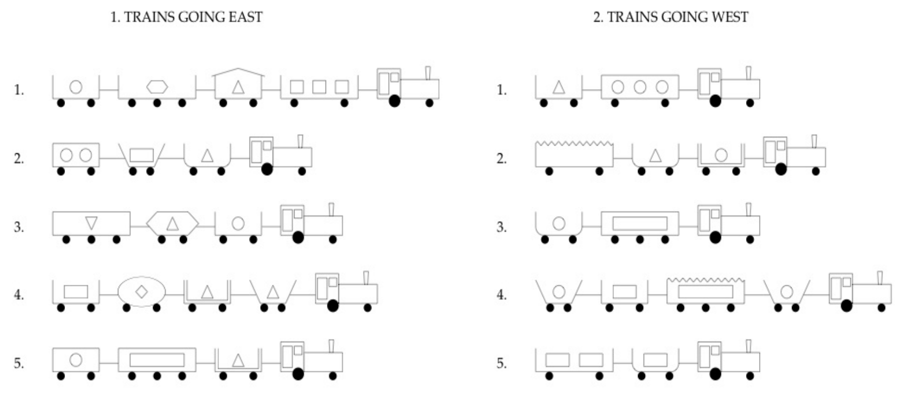


Fig. 2 The train spotting dataset. There are two sets of trains: Eastbound and Westbound. Descriptors for the trains include: number, shape and lengths of the car, shape of load, and so on. The task is to determine decision rules to distinguish between them.

carried (Figure 2). For example, a decision rule can be:

$$
\begin{aligned}
&[NCARS = 4]\\
&[IF(CAR_1, CAR_2)][INFRONT(CAR_2, CAR_3)][INFRONT(CAR_3, CAR_4)]\\
&[CAR - SHAPE(CAR_1) = ENGINE][CAR - SHAPE(CAR_2) = U - SHAPED]\\
&[CAR - SHAPE(CAR_3) = OPEN - TRAPEZOID]\\
&[CAR - SHAPE(CAR_4) = RECTANGLE][LN(CAR_1) = LONG]\\
&[LN(CAR_2) = SHORT][LN(CAR_3) = SHORT][LN(CAR_4) = SHORT] \Rightarrow [D = EAST].
\end{aligned}
$$

where the features are: (1) $NCARS$: number of cars, (2) $CAR-SHAPE(car)$: shape of a car, (3) $LN(car_i)$:length of a car, (4)$INFRONT(car_i, car_j)$: predicate indicating that car_i is infront of car_j.

We assume that trains can be adequately described by background predicates and that 10 trains shown in the figure are denoted by $t1, t2, \cdots, t10$. The classification of trains can be encoded by logical statements of the form $Class(t1, eastbound), Class(t2, eastbound) \cdots, Class(t10, westbound)$. All features are assumed to be boolean valued as is common in ILP applications [34].

Suppose 500 such features are randomly generated by an ILP engine for the train spotting problem. Our task is to find the subset of features that result in the best model. An exhaustive search in the space of 2^{500} features is clearly intractable. Are we, however, able to partition the feature space, such that compute nodes in a distributed environment could individually work with subsets, *gossip* with neighboring nodes to exchange information and eventually agree on a set of *consensus* features? The algorithm for distributed feature estimation, described in Section 5, takes a step in that direction.

3 Related Work

ILP Applications Existing literature on discovering a subset of discriminative features from inductive logic programming applications can be classified as follows: strategies that pose the problem as a (a)discrete optimization problem that is solved optimally [15,26,21] or heuristically [19,32,24,10,33,31,35] (b) convex optimization problem with sparsity inducing regularizers that solves it optimally [17,25] (c) computes all relational features that satisfy some quality criterion by systematically and efficiently exploring a prescribed search space [28,18,3,2,29,1,30,4,14].

Other Domains

4 Feature Discovery

5 Problem Statement

Let M denote an $n \times m$ matrix with real-valued entries. This matrix represents a dataset of n tuples of the form $X_i \in \mathbb{R}^m, 1 \leq i \leq n$. Assume, without loss of generality, this dataset has been vertically distributed over k sites $S_1, S_2, \cdots, S_k$ i.e. site S_1 has m_1 attributes, S_2 has m_2 attributes and so on, such that $|m_1| + |m_2| + \cdots + |m_k| = |m|$, where $|m_i|$ represents the number of attributes at site S_i[1]. Let M_1 denote the $n \times m_1$ matrix representing the dataset held by S_1, M_2 denote the $n \times m_2$ matrix representing the dataset held by S_2 and so on. Thus, $M = M_1 : M_2 : \cdots : M_k$ denotes the concatenation of the local datasets.

The goal is to learn a linear discriminative function over the data set M. The global function to be estimated is represented by $f_g = MW_g^T$ where W_g is assumed to be a $1 \times m$ weight vector. If only the local data is used, at site S_1, the local function estimated would be $f_1 = M_1 W_1^T$. At site S_2, the local function estimated would be $f_2 = M_2 W_2^T$. The goal is to describe a de-centralized algorithm for computing the weight vectors at sites $S_1, \cdots S_k$ such that on termination $W_1 \approx W_g[1 : m_1], W_2 \approx W_g[1 : m_2], \cdots W_k \approx W_g[1 : m_k]$ where $W_g[1 : m_i]$ represents the part of the global weight vector for the attributes stored at that site S_i. Clearly, if all the datasets are transferred to a central location, the global weight vector can be estimated. Our objective is to learn the function in the decentralized setting assuming that transfer of actual data tuples is expensive and may not be allowed (say for example due to privacy concerns). The weights obtained at each site on termination of the algorithm will be used for ranking the features.

[1] In the more general setting, Site S_i has a random subset of features $m_i \subset m$.

6 Distributed Feature Estimation Algorithm

6.1 Building Blocks: Distributed Scalar Product Computation

Several distributed scalar product computation protocols have been studied in literature([12],[13], [37]). The goal is to ensure that each node in the distributed environment has the scalar product of the local vectors of all other nodes (possibly securely, when privacy needs to be preserved). We introduce a two-party scalar product computation protocol that the Distributed Feature Estimation Algorithm (described in 1) relies on.
Problem 1: Site S_1 has n instances with features $X_1^i, \cdots, X_p^i$ and weight vectors $W_1^i, \cdots, W_p^i$. Site S_2 has n instances with features $X_{p+1}^i, \cdots, X_m^i$ and weight vectors $W_{p+1}^i, \cdots W_m^i$ respectively. Site S_1 and S_2 need to obtain the scalar product $\sum_{i=1}^{m} X_i^j W_i^j, 1 \leq j \leq n$ for all the n instances.

Consider the following naive solution: Site S_1 sends features and weights for all its n instances to Site S_2; Site S_2 then computes the required scalar product. Clearly, in this case the communication cost involved is large i.e. $O(n \times p)$ and all of the data tuples need to be exchanged.

A better solution would be if Site S_1 sent only the local scalar product for all n instances, i.e. $\sum_{i=1}^{p} X_i^j W_i^j, 1 \leq j \leq n$. The communication cost in this case would be $O(n)$. Site S_2 on receiving the scalar product from S_1 adds it to its own local estimate i.e. $\sum_{i=1}^{p} X_i^j W_i^j + \sum_{i=1}^{m-(p+1)} X_i^j W_i^j, 1 \leq j \leq n$. Site S_2 has effectively computed $\sum_{i=1}^{m} X_i^j W_i^j, 1 \leq j \leq n$ for all the n instances. This additive nature of the protocol makes it amenable for use in distributed settings.

6.2 Assumptions

We state the following assumptions under which the Distributed Feature Estimation algorithm operates:

Model of Distributed Computation The distributed algorithm evolves over discrete time with respect to a "global" clock[2]. Each site has access to a local clock or no clock at all. Furthermore, each site has its own memory and can perform local computation (such as computing the gradient on its local attributes). It stores f_i, which is the estimated local function. Besides its own computation, sites may receive messages from their neighbors which will help in evaluation of the next estimate for the local function.

Communication Protocols Sites S_i are connected to one another via an underlying communication framework represented by a graph $G(V, E)$, such that each site $S_i \in \{S_1, S_2, \cdots, S_k\}$ is a vertex and an edge $e_{ij} \in E$ connects

[2] Existence of this clock is of interest only for theoretical analysis.

```
Input: $n \times m_i$ matrix at each site $S_i$, $G(V, E)$ which encapsulates the underlying
        communication framework, $T$ : no of iterations
Output: Each site $S_i$ has $W_i \approx W_g[1 : m_i]$
for $t = 1$ to $T$ do
    (a) Site $S_i$ computes $M_i W_i^T$ locally and estimates the loss function;
    (b) Site $S_i$ gossips with its neighbors $S_j \in \{N_i\}$ and obtains $M_j W_j^T$ for each
    neighbor;
    (c) Site $S_i$ locally updates its function estimate as
    $J_i^t = \alpha_{ii}(M_i W_i^T) + \alpha_{ji}(M_j W_j^T)$ ;
    (d) Update the local weight vectors using stochastic gradient descent as follows:
    $\frac{\partial L_p}{\partial W_i} = -X_p(Y_p - J_i^t(W_i^t(p)))$
end
```

Algorithm 1: Distributed Feature Estimation

sites S_i and S_j. Communication delays on the edges in the graph are assumed to be zero. It must be noted that the communication framework is usually expected to be application dependent. In cases where no intuitive framework exists, it may be possible to simply rely on the physical connectivity of the machines, for example, if the sites S_i are part of a large cluster.

6.3 Algorithm Description

Algorithm 1 describes how the weights for attributes will be estimated using a consensus-based protocol. There are two main sub-parts of the algorithm: (1) Exchange of local function estimate and (2) Local update based on stochastic gradient descent. Each of these sub-parts are discussed in further detail below. Furthermore, assume that $J : R^m \rightarrow [0, \infty]$ is a continuously differentiable nonnegative cost function with a Lipschitz continuous derivative.

Exchange of Local Function Estimate: Each site locally computes the loss based on its attributes and then gossips with the neighbors to get information on other attributes. On receiving an update from a neighbor, the site re-evaluates J_i by forming a component-wise convex combination of its old vector and the values in the messages received from neighbors i.e. $J_i^{t+1} = \alpha_{ii}(X_i W_i^T) + \alpha_{ji}(X_j W_j^T)$. It is interesting to note that $\alpha_{il}, 0 \leq \alpha_{il} \leq 1$, is a non-negative weight that captures the fraction of information site i is willing to share with site l. The choice of α_{il} may be deterministic or randomized and may or may not depend on the time t [20]. The $k \times k$ matrix A comprising of $\alpha_{il}, 1 \leq i \leq k, 1 \leq l \leq k$ is a "stochastic" matrix such that it has non-negative entries and each row sums to one. More generally, this reflects the state transition probabilities between sites. Figure 3 illustrates the state transition between two sites S_i and S_j.

Another interpretation of the diffusion of J_i amongst the neighbors of i involves drawing analogies from Markov chains – the diffusion is mathematically identical to the evolution of state occupation probabilities. Further-

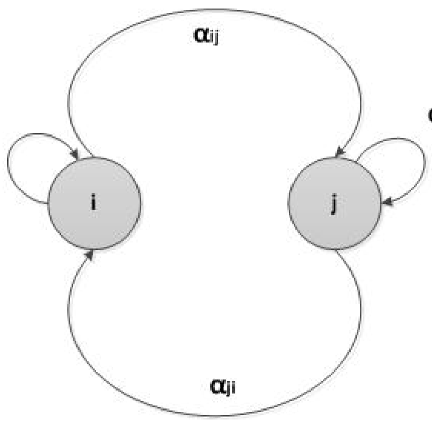


Fig. 3 State Transition Probability between two sites S_i and S_j

more, a simple vector equation can be written for updating J_i^t to J_i^{t+1} i.e. $J_i^{t+1} = A(i)(J_i^t)_{N_i}$ where $A(i)$ corresponds to the row i of the matrix A and $(J_i^t)_{N_i}$ is a matrix that has $|N_i|$ rows (each row corresponding to a neighbor of Site S_i) and n columns (each column corresponding to all the instances). More generally, $\mathcal{J}^{t+1} = A\mathcal{J}^t$ where $\mathcal{J}^{t+1}$ is a $k \times n$ matrix storing the local function estimates of each of the n instances at site k and A is the $k \times k$ transition probability matrix corresponding to all the sites. It follows that $lim_{t\to\infty}A^t$ exists and this controls the rate of convergence of the algorithm.

We introduce the notion of *average function estimate* in the network $\mathbf{J_i^t} = \sum_i \frac{J_i^t}{k}$ which allocates equal weight to all the local function estimates and serves as a baseline against which individual sites S_i's can compare their performance. Philosophically, this also implies that each local site should at least try to attain as much information as required to converge to the average function estimate. Since $\sum_i \alpha_{ij} = 1$, this estimate is invariant.

The A matrix has interesting properties which allow us to show that convergence to $\mathbf{J_i^t}$ occurs. One such property is the Perron-Frobenius theory of irreducible non-negative matrices. We state the theorem here for continuity.

Theorem 1 ***Perron-Frobenius*** *[38] Let A be a positive, irreducible matrix such that the rows sum to 1. Then the following are true:*

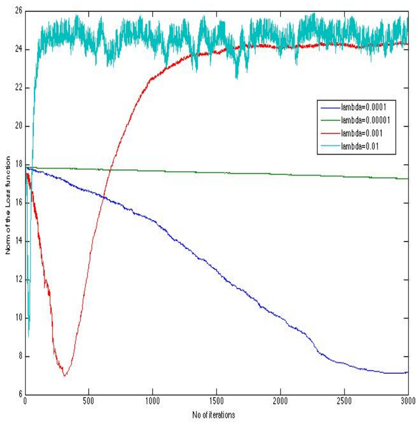
1. *The eigenvalues of A of unit magnitude are the k-th roots of unity for some k and are all simple.*
2. *The eigenvalues of A of unit magnitude are the k-th roots of unity if and only if A is similar under a permutation to a k cyclic matrix*
3. *all eigenvalues of A are bounded by 1.*

Since the eigenvalues of A are bounded by 1, it can be shown that J_i^t converges to the average function estimate $\mathbf{J_i^t}$ if and only if -1 is not an eigen value [38]. Let $\lambda_n \leq \lambda_{n-1} \leq \cdots \leq \lambda_2 < \lambda_1 = 1$ be the eigenvalues of A with $\lambda_1 = 1$. Also assume that $\gamma(A) = \max_{i>1}|\lambda_i|$. It can be shown that

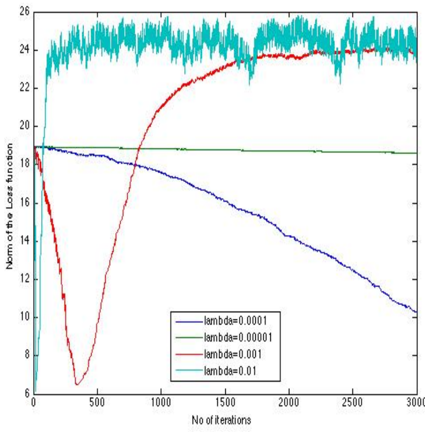
$\| J_i^{t+1} - \mathbf{J}_i^t \|^2 \leq \gamma^2 \| J_i^t - \mathbf{J}_i^t \|$. If $\gamma = 1$, then system fails to converge [38], [11].

Local Stochastic Gradient update is done as follows: $W_i^{t+1} = W_i^t - \eta_i^t s_i^t$ where $s_i^t = \frac{\partial L_i^t}{\partial W_i^t}$ is the estimated gradient, W_i^t is the weight vector and η_i^t is the learning rate at node i at time t. Let $s_i^t = \frac{\partial J_i^t}{\partial W_i^t} + \mathcal{W}_i^t$, where $\mathcal{W}_i^t$ is unit variance white noise independent for each site S_i. Then, it can be easily seen that $E[\| \mathcal{W}_i^t \|^2 |W_i^t] \leq M(\| \nabla J_i^t \|^2)$, for some constant M.

An empirical example: Consider the train spotting example introduced in Section 2. Assume that the 500 randomly generated features[3] are partitioned amongst two sites, such that each contains 250 features. A distributed computing environment is simulated using the peersim simulator[4]. The distributed feature selection algorithm was run for a fixed number of iterations varying the learning rate of the algorithm. Figures 4(a) and 4(b) show how the norm of the loss function varies with the number of iterations at each site eventually leading to convergence.



(a) Site 1



(b) Site 2

6.4 Convergence

The proof of convergence of the algorithm has several sub-parts: (1) In the distributed setting, the process of information exchange between k sites can

[3] The Aleph software `http://www.cs.ox.ac.uk/activities/machlearn/Aleph/aleph.html` was used to generate the features.

[4] http://peersim.sourceforge.net/

be modeled as a non-stationary Markov chain. A non-stationary Markov chain is weakly ergodic if the dependence on the state distribution vanishes as time tends to infinity. Thus convergence of information exchange between sites needs to be established. This follows from the idea introduced by Tsitsiklis et al. [36]. It is based on proving convergence of a sequence of products of stochastic matrices (the A matrix in our formulation) and the conclusions follow from well-known results on weak ergodicity of non-stationary Markov chains. (2) In a given iteration of the optimization, one needs to establish how good the approximated distributed optimization is compared to its "ideal" central counterpart. This follows from the rich theory of approximate convex functions. We present an illustrative example and the basic notations and assumptions which eventually lead to the proof of convergence. (3) The number of iterations T the algorithm requires to finally attain convergence in the distributed setting.

An Example: Consider two sites S_1 and S_2 with features x_1, x_2 and x_3, x_4 respectively. Site S_1 locally computes $J_i = [x_1x_2][w_1w_2]^T$ and Site S_2 computes $J_j = [x_3x_4][w_3w_4]^T$. Site S_2 sends $\alpha_{ji}J_j$ to S_1 and S_1 sends $\alpha_{ij}J_i$ to S_2. The local function estimates are updated to $J_i^{t+1} = \alpha_{ii}J_i + \alpha_{ji}J_j$ and $J_j^{t+1} = \alpha_{jj}J_j + \alpha_{ij}J_i$. Both J_i and J_j are convex functions [9] and J_j^{t+1} and J_i^{t+1} are simply perturbations of convex functions by bounded functions. This observation allows us to use the rich theory of Approximately Convex Functions ([16],[27]) to show that the proposed algorithm will eventually converge.

Definition 1 ***Perturbation of Convex Functions by Bounded Functions ([27]) :*** *Let $\delta \geq 0$ and $J : R^m \to R$ be of the form $J = g + h$, where $g : R^m \to R$, is convex and $\| h \| \leq \delta$ i.e. $|h(x)| \leq \delta, \forall x \in m$. Then, for all $x, y \in m$ and $t \in [0, 1]$, $f(tx + (1 - t)y) = tf(x) + (1 - t)f(y) + 2\delta$. More generally, $x_1, x_2 \cdots, x_n \in R^m$ and $t_1, \cdots, t_n \in [0, 1]$ with $t_1 + \cdots + t_n = 1$, $f(t_1x_1 + \cdots + t_nx_n) \leq t_1f(x_1) + \cdots + t_nf(x_n) + 2\delta$.*

Definition 2 ***δ-convex function:*** *Let $\delta \geq 0$. Then a function $J : R^m \to R$ is called δ-convex if $f(tx + (1 - t)y) \leq tf(x) + (1 - t)f(y) + \delta, (x, y \in R^m, t \in [0, 1])$*

Theorem 2 ***(Hyers and Ulam [16])*** *Let $J(x)$ be δ-convex on the open convex set $G \subset R^m$ and B be any closed bounded convex subset of G. Then, there exists a convex function $\phi(x)$ on B such that $|\phi(x) - f(x)| \leq k_m\delta$ where k_m is a positive constant that depends only on the dimensionality m and $k_m = \frac{m^2+3m}{4m+4}$.*

The above theorem can also be interpreted as follows: Assume that $J : R^m \to R$ is δ-convex and is of the form $J = g + h$, where $g : R^m \to R$, is a convex function and $h : R^m \to R$, is a bounded function such that $\| h \| \leq k_m\delta$ where k_m depends only on the dimensionality m as illustrated above.

Note that f_g is defined over $\mathbb{R}^m$ while f_i is applied on a subset of $\mathbb{R}^m$. Application of Hyers and Ulam theorem above, to each site S_i clearly shows that $|f_i - f_g| \leq k_m\delta$. This helps to characterize how good the local approximation is at each site when compared to the hypothetical centralized algorithm.

Termination criteria: The distributed algorithm is run until T iterations. The usual practice with stochastic gradient descent algorithms is to run them until the loss at each site is within a certain user-defined threshold. We follow the same principle here, but note that for the distributed algorithm, each site runs stochastic gradient descent locally and independently of any other site. This implies that the same example may not be chosen at each site and this affects the termination criteria.

How many iterations are required for convergence? Parameter updates performed on the cost function $J_k^t(W_k^t)$[5] at site S_k are based on the sample $X_p = (X_p^1, \cdots, X_p^{m_k}), p \in \{1, \cdots, n\}$ chosen from M_k. $L_k^t(W_k^t)$ can be expressed as a very large sum of terms i.e. $L_k^t(W_k^t) = \frac{1}{n}\sum_{p=1}^{n} L_k^t(X_p, W_k^t)$. The parameter update using Stochastic Gradient Descent (SGD) can be written as

$$W_k^{t+1} = W_k^t - \frac{1}{t}\phi_t \frac{\partial L_k^t}{\partial W_k^t}(X_p, W_k^t) \tag{1}$$

where ϕ_t is an appropriately chosen positive semi-definite matrix.

Simple batch algorithms where the gradient is estimated over *all* the training examples, converge linearly to the optimum W_k^* of the empirical cost. In contrast, when using an SGD algorithm, it can be shown that under mild assumptions, the optimization converges to a local minimum of the cost [7, 5,6]. It appears that SGD-based algorithms converge to the general area of the optimum as fast as batch algorithms, but tend to wobble around in that region due to the noisy estimation of the gradient. It has been shown that $\mathbf{E}(W_k^t - W_k^*)$ converges like $\frac{1}{t}$ at best [8].

The above observation can also be explained as follows: since the measurement of the gradient is noisy, we do not have access to $L_k^t(W_k^t)$ but to a noisy variant of it $\frac{\partial L_k^t}{\partial W_k^t}(X_p, W_k^t) + \delta_k^t$ where δ_k^t is a random variable representing the noise. If we used the gradient algorithm as if no noise were present, Equation 1 can be re-written as:

$$W_k^{t+1} = W_k^t - \frac{1}{t}\phi_t\left(\frac{\partial L_k^t}{\partial W_k^t}(X_p, W_k^t) + \delta_k^t\right) \tag{2}$$

Equation 2 does not converge to a minimum of L_k^t and what may happen at best is that W_k^t converges to a local minima and moves randomly about W_k^*. The mean square distance of W_k^t from W_k^* increases with variance of W_k^t and can be made small only if η_k^t is chosen small. If η_k^t is chosen to be very small it can take a very large number of steps to reach a neighborhood of a local minima. To strike a balance, η_k^t is usually set to $\frac{1}{t}$ which allows a rapid approach to the vicinity of the W_k^* and then decreased to zero, so that the variance of W_k^t also reduces. Some typical conditions set on the step size are: $\sum_{t=1}^{\infty}\eta_k^t = \infty$ and $\sum_{t=1}^{\infty}(\eta_k^t)^2 < \infty$.

In our distributed setting, let W_k^* be the local optimum of the cost $J_k^t(W)$ and W_g^* be the global optimum of the cost $J_g^t(W)$ involving all the attributes

[5] Note that the cost function is explicitly parameterized by W to indicate that J_k^t depends on W. This is a slightly different notation than what we used in Section 5.

over the network. We would like to know how fast W_k converges to W_g^* instead of W_k^*. This implies we need to study the effect of perturbation of $J_k^t(W)$ on convergence of the algorithm to the global solution.

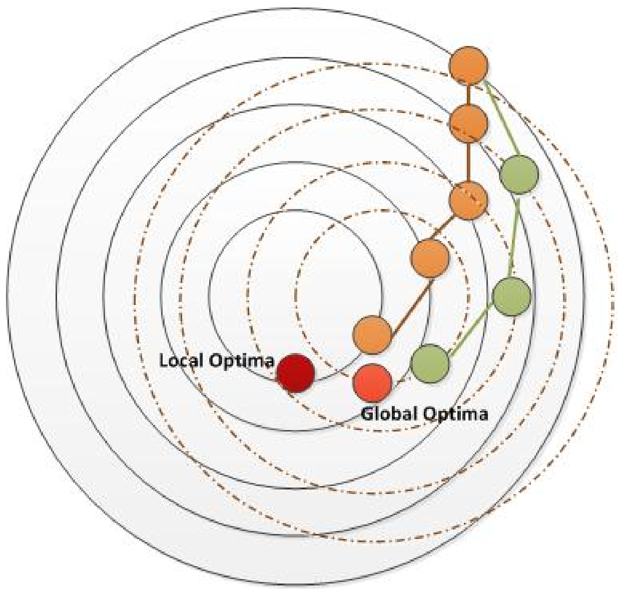


Fig. 4 Contour of the local and global optimization at a site. The brown concentric circles indicate the perturbation of the local cost to learn the global cost. The path traced by the yellow ochre colored circles indicate a possible optimization path if only the local cost were considered. The green colored circles show the path if the global cost were to be optimized.

Bottou and Le Cun ([8],[23]) show that

$$W_t^* = W_{t-1}^* - \frac{1}{t}\hat{H}_n^{-1}\frac{\partial J}{\partial W}(X_p, W_{t-1}^*) + \mathcal{O}_u(\frac{1}{n^2}) \tag{3}$$

where empirical Hessian are defined as $\hat{H}_n = \frac{1}{n}\sum_{i=1}^{n}\frac{\partial^2}{\partial\theta\partial\theta}L(X_p, W_{t-1}^*)$

7 Experimental Evaluation

7.1 Aims

7.2 Materials

7.3 Method

8 Results and Discussion

9 Conclusion

Acknowledgements This work began when Haimonti Dutta visited the Indraprastha Institute of Information Technology, Delhi in Spring 2013. She was also supported by National Science Foundation grant, IIS-0916186. The classification of literature on discriminative feature selection in ILP applications was first suggested by Ganesh Ramakrishnan.

References

1. Agrawal, R., Srikant, R.: Mining sequential patterns. In: ICDE (1995)
2. Antunes, C., Oliveira, A.L.: Generalization of pattern-growth methods for sequential pattern mining with gap constraints. In: MLDM (2003)
3. Aseervatham, S., Osmani, A., Viennet, E.: bitspade: A lattice-based sequential pattern mining algorithm using bitmap representation. In: ICDM (2006)
4. Ayres, J., Gehrke, J., Yiu, T., Flannick, J.: Sequential pattern mining using a bitmap representation. In: KDD (2002)
5. Benveniste, A., Priouret, P., Métivier, M.: Adaptive algorithms and stochastic approximations. Springer-Verlag New York, Inc., New York, NY, USA (1990)
6. Bertsekas, D.P., Tsitsiklis, J.N.: Parallel and Distributed Computation: Numerical Methods. Athena Scientific, Belmont, MA. (1997)
7. Bottou, L.: Online algorithms and stochastic approximations. In: D. Saad (ed.) Online Learning and Neural Networks. Cambridge University Press, Cambridge, UK (1998). URL `http://leon.bottou.org/papers/bottou-98x`. Revised, oct 2012
8. Bottou, L., Le Cun, Y.: On-line learning for very large data sets. Applied Stochastic Models in Business and Industry **21**(2), 137–151 (2005)
9. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press, New York, NY, USA (2004)
10. Chalamalla, A., Negi, S., Subramaniam, L.V., Ramakrishnan, G.: Identification of class specific discourse patterns. In: CIKM, pp. 1193–1202 (2008)
11. Cybenko, G.: Dynamic load balancing for distributed memory multiprocessors. Proceedings of the Journal of Parallel and Distributed Computing **7**, pp. 279–301 (1989)
12. Du, W., Atallah, M.: Privacy-preserving cooperative statistical analysis. In: Proceedings of the 17th Annual Computer Security Applications Conference, ACSAC '01, pp. 102–. Washington, DC, USA (2001). URL `http://dl.acm.org/citation.cfm?id=872016.872181`
13. Du, W., Atallah, M.: Protocols for secure remote database access with approximate matching. In: E-Commerce Security and Privacy, *Advances in Information Security*, vol. 2, pp. 87–111 (2001)
14. Garofalakis, M.N., Rastogi, R., Shim, K.: Spirit: Sequential pattern mining with regular expression constraints. In: VLDB (1999)
15. Han, Y., Wang, J.: An l1 regularization framework for optimal rule combination. In: ECML/PKDD, year = 2009,

16. Hyers, D.H., Ulam, S.M.: Approximately convex functions. Proceedings of the American Mathematical Society **3**(5), pp. 821–828 (1952). URL `http://www.jstor.org/stable/2032186`
17. Jawanpuria, P., Nath, J.S., Ramakrishnan, G.: Efficient rule ensemble learning using hierarchical kernels. In: ICML, pp. 161–168 (2011)
18. Ji, X., Bailey, J., Dong, G.: Mining minimal distinguishing subsequence patterns with gap constraints. Knowledge and Information Systems (2006)
19. Joshi, S., Ramakrishnan, G., Srinivasan, A.: Feature construction using theory-guided sampling and randomised search. In: ILP, pp. 140–157 (2008)
20. Kempe, D., Dobra, A., Gehrke, J.: Gossip-based computation of aggregate information. In: Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on, pp. 482–491 (2003)
21. Kudo, T., Maeda, E., Matsumoto, Y.: An application of boosting to graph classification. In: NIPS (2004)
22. Larson, J., Michalski, R.S.: Inductive inference of vl decision rules. SIGART Bull. (63), 38–44 (1977)
23. Le Cun, Y., Bottou, L., Orr, G.B., Müller, K.R.: Efficient backprop. In: Neural Networks, Tricks of the Trade, Lecture Notes in Computer Science LNCS 1524. Springer Verlag (1998). URL `http://leon.bottou.org/papers/lecun-98x`
24. Nagesh, A., Ramakrishnan, G., Chiticariu, L., Krishnamurthy, R., Dharkar, A., Bhattacharyya, P.: Towards efficient named-entity rule induction for customizability. In: EMNLP-CoNLL, pp. 128–138 (2012)
25. Nair, N., Saha, A., Ramakrishnan, G., Krishnaswamy, S.: Rule ensemble learning using hierarchical kernels in structured output spaces. In: AAAI (2012)
26. Nowozin, S., Bakr, G., Tsuda, K.: Discriminative subsequence mining for action classification. In: CVPR (2007)
27. Páles, Z.: On approximately convex functions. Proceedings of the American Mathematical Society **131**(1), pp. 243–252 (2002)
28. Pei, J.: Mining sequential patterns by pattern-growth: The prefixspan approach. Journal of Machine Learning Research **16-11** (2004)
29. Pei, J., Han, J., Wang, W.: Constraint-based sequential pattern mining: the pattern-growth methods. Journal of Intelligent Information Systems (2005)
30. Pei, J., Han, J., Yan, X.F.: From sequential pattern mining to structured pattern mining: A pattern-growth approach. Journal of Computer Science and Technology **9**(3), 257–279 (2004)
31. Ramakrishnan, G., Joshi, S., Balakrishnan, S., Srinivasan, A.: Using ilp to construct features for information extraction from semi-structured text. In: ILP, pp. 211–224 (2007)
32. Saha, A., Srinivasan, A., Ramakrishnan, G.: What kinds of relational features are useful for statistical learning? In: ILP (2012)
33. Specia, L., Srinivasan, A., Joshi, S., Ramakrishnan, G., das Graças Volpe Nunes, M.: An investigation into feature construction to assist word sense disambiguation. Machine Learning **76**(1), 109–136 (2009)
34. Specia, L., Srinivasan, A., Joshi, S., Ramakrishnan, G., Graas Volpe Nunes, M.: An investigation into feature construction to assist word sense disambiguation. Machine Learning **76**(1), 109–136 (2009)
35. Specia, L., Srinivasan, A., Ramakrishnan, G., das Graças Volpe Nunes, M.: Word sense disambiguation using inductive logic programming. In: ILP, pp. 409–423 (2006)
36. Tsitsiklis, J.N., Bertsekas, D.P., Athans, M.: Distributed asynchronous deterministic and stochastic gradient optimization algorithms. In: IEEE Transactions on Automatic Control, vol. AC-31 (1986)
37. Vaidya, J., Clifton, C.: Privacy preserving association rule mining in vertically partitioned data. In: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '02, pp. 639–644. New York, NY, USA (2002)
38. Varga, R.: Matrix Iterative Analysis. Prentice Hall, Englewood Cliffs, NJ (1962)