Mach Learn (2018) 107:825–858
https://doi.org/10.1007/s10994-017-5672-2

Consensus-based modeling using distributed feature construction with ILP

Haimonti Dutta[1] · Ashwin Srinivasan[2]

Received: 14 December 2014 / Accepted: 14 September 2017 / Published online: 2 November 2017

Abstract A particularly successful role for Inductive Logic Programming (ILP) is as a tool for discovering useful relational features for subsequent use in a predictive model. Conceptually, the case for using ILP to construct relational features rests on treating these features as functions, the automated discovery of which necessarily requires some form of first-order learning. Practically, there are now several reports in the literature that suggest that augmenting any existing feature with ILP-discovered relational features can substantially improve the predictive power of a model. While the approach is straightforward enough, much still needs to be done to scale it up to explore more fully the space of possible features that can be constructed by an ILP system. This is in principle, infinite and in practice, extremely large. Applications have been confined to heuristic or random selections from this space. In this paper, we address this computational difficulty by allowing features and models to be constructed in a distributed manner. That is, there is a network of computational units, each of which employs an ILP engine to construct some small number of features and then builds a (local) model. We then employ an asynchronous consensus-based algorithm, in which neighboring nodes share information and update local models. This gossip-based information exchange results in the formation of non-stationary Markov chains. For a category of models (those with convex loss functions), it can be shown (using the Supermartingale Convergence Theorem) that the algorithm will result in all nodes converging to a consensus model. In practice, it may be slow to achieve this convergence. Nevertheless, our results on synthetic and real datasets suggest that in relatively short time the "best" node in the network reaches a model whose predictive accuracy is comparable to that obtained using more computational

Editors: Jesse Davis and Jan Ramon.

✉ Haimonti Dutta
haimonti@buffalo.edu

Ashwin Srinivasan
ashwin@goa.bits-pilani.ac.in

1 Department of Management Science and Systems, University at Buffalo, New York, NY 14260, USA

2 Department of Computer Sciences and Information Systems, BITS-Pilani, Goa Campus, Goa, India

effort in a non-distributed setting (the best node is identified as the one whose weights converge first).

Keywords Inductive logic programming · Consensus based learning · Stochastic gradient descent · Feature selection

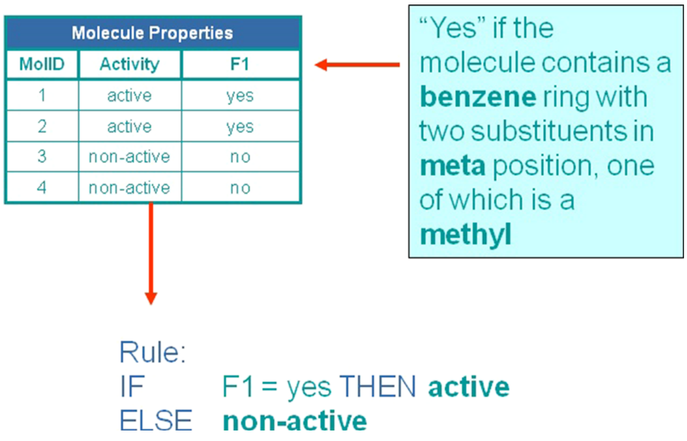
1 Introduction

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 is at least one kind of task where some form of relational learning would appear to be necessary. This is to do with 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 ("$F(m) = 1$ if a molecule m has 3 or more benzene rings fused together otherwise $F(m) = 0$"). Such 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).

The approach usually, but not always, separates relational learning (to discover features) and modeling (to build models using these features). There will of course be problems that require the joint identification of relational 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 (see, for example: Joshi 2008; Saha et al. 2012; Specia et al. 2009; Ramakrishnan et al. 2007; Specia et al. 2006). There are thus very good practical reasons to persist with this variant of statistical and logical learning for data analysis.

There are known shortcomings with the approach which can limit its applicability. First, the set of 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. Second, much needs to be done to scale ILP-based feature discovery up to meet modern "big" data requirements. This includes the abilities to discover features using very large datasets not all



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

stored in one place, and perhaps only in secondary memory; from relational data arriving in a streaming manner; and from data which do not conform to expected patterns (the concept changes, or the background knowledge becomes inappropriate). Third, even with "small" data, it is well-known that obtaining the value of a feature function for a data instance can be computationally hard. This means that obtaining the feature-vector representation using ILP-discovered features can take large amounts of time. This paper is concerned only with the first of these problems, namely how to construct models when feature-spaces are very large. The data is partitioned and placed on different processors (or nodes).

We develop a simple general-purpose consensus-based modeling technique consisting of a network of computing nodes. Each node in the network:

(a) Works with a local model that uses a small set of features;
(b) Communicates with neighboring nodes to exchange information about its model; and
(c) Eventually arrives at a consensus model and (usually bigger) set of features that represents the consensus with its neighbors.

We note straightaway that while the consensus-based approach does not provide an optimal solution to the feature-selection problem, we show that it does provide a way of distributing the computational task of feature-construction, and for a class of models, converge to a consensus solution.

Organization: Section 2 is a short introduction to ILP and its use in constructing features. The approach we intend to follow of distributed feature-construction followed by consensus-based modeling is introduced in Sect. 2.2.1. We view our approach as an instance of a more general technique that performs consensus-based model construction in a distributed setting. Section 3 presents a general iterative procedure for constructing models in a network of nodes capable of exchanging information about their local models and features. Experimental results are in Sect. 4. Section 5 presents related work; Sect. 6 discusses open issues and concludes the paper.

2 ILP

2.1 Specification

Since its formulation in the early 1990's in the form of a partial specification (Muggleton 1994), the field of Inductive Logic Programming (ILP) has grown to mean various forms of relational learning, with first-order logic as the recurring theme for the representation of inputs (domain-knowledge and data) and outputs (models or hypotheses). The original specifications in Muggleton (1994) though remain a useful way to describe the function of a class of programs that construct theories for either discriminating accurately amongst two sets of examples ("positive" and "negative"), or for describing a set of examples, without any specific goal of discrimination. In what is now known as the "learning from entailment" formulation, an ILP algorithm is taken to be one that conforms to at least the following (we refer the reader to Nienhuys-Cheng and De Wolf (1997) for definitions in logic programming).

- B is background knowledge consisting of a set of definite clauses $= \{C_1, C_2, \ldots\}$
- $\mathcal{L}$ is a language describing constraints on acceptable hypotheses
- E is a finite set of examples $= E^+ \cup E^-$ where:
  - *Positive Examples.* $E^+ = \{e_1, e_2, \ldots\}$ is a set of definite clauses denoting instances entailed by some unknown target concept T in conjunction with the background knowledge;
  - *Negative Examples.* $E^- = \{\overline{f_1}, \overline{f_2} \ldots\}$ is a (possibly empty) set of Horn clauses denoting instances consistent with $B \wedge T$; and
  - *Prior Necessity.* $B \not\models E^+$
- $H = \{D_1, D_2, \ldots\}$, the output of the algorithm given B, $\mathcal{L}$ and E, is a hypothesis about the unknown target T s.t. each D_i is consistent with $\mathcal{L}$. A hypothesis H is acceptable if the following conditions are met:
  - *Weak Sufficiency.* Each D_i in H is a definite clause that has the property $B \cup \{D_i\} \models e_1 \vee e_2 \vee \ldots$, where $\{e_1, e_2, \ldots\} \subseteq E^+$
  - *Strong Sufficiency.* $B \cup H \models E^+$;
  - *Weak Consistency.* $B \cup H \not\models \Box$; and
  - *Strong Consistency.* $B \cup H \cup E^- \not\models \Box$;

Strong Consistency ensures that H is consistent with all of the negative examples. Often, implementations do not require hypotheses to meet this requirement, as some members of E^- are taken to be "noisy". This specification is then refined to include a parameter whose value sets a lower bound on the accuracy required of each clause D_i in the theory. If the noise model extends to the positive examples, then in practice, implementations may also also not meet the Strong Sufficiency requirement.

2.2 Implementation

Given that the specifications impose fairly minimal constraints, it is not surprising that a variety of conforming (or nearly conforming) implementations have been developed. Of these, we first describe an implementation that identifies a discriminatory model (specifically, a set of classification rules), using a randomized version of a traditional greedy set-covering approach. Individual rules in the set are identified using a general-to-specific, heuristic search guided by most-specific ("bottom") clauses:

```
$ConstructModel(B, \mathcal{L}, E)$ :
1. Let $H$ be $\emptyset$
2. $Left = E^+$
3. while $Left \neq \emptyset$ do
    (a) Randomly chose an example $e \in Left$
    (b) Let $\perp_{\mathcal{L}}(B, e)$ be the most specific rule in some language $\mathcal{L}$ that logically entails $e$, given background knowledge $B$ (see Muggleton 1995)
    (c) Search for the "best" rule $h$ in the lattice ordered by the subsumption relation (Plotkin 1971) and bounded by the empty clause $\top$ and $\perp_{\mathcal{L}}(B, e)$.
    (d) $PosCover = \{e : e \in E^+ \text{ and } B \cup h \models e\}$
    (e) $Left := Left - PosCover$
    (f) Add $h$ to $B$
    (g) Add $h$ to $H$
4. done
```

Here, "best" refers to the rule with the highest value for some evaluation function. This procedure is sufficiently similar to the one followed by the classic ILP algorithm Progol, which uses a mode-based language for specifying $\mathcal{L}$, and a simple compression-based heuristic to score clauses (see Muggleton 1995 for details).

2.2.1 Feature construction

Of more direct interest here is a derivative of $ConstructModel$ that is used to identify Boolean features:

```
$ConstructFeatures(B, \mathcal{L}, E, f_{max})$ :
1. Let $F$ be $\emptyset$
2. $Left = E^+$
3. while the feature constructed is less than some maximum $f_{max}$ do
    (a) Select $e$ from $Left$
    (b) Let $\perp_{\mathcal{L}}(B, e)$ be the most specific rule in some language $\mathcal{L}$ that logically entails $e$, given background knowledge $B$
    (c) Search for "good" rules in the lattice ordered by the subsumption relation and bounded by the empty clause $\top$ and $\perp_{\mathcal{L}}(B, e)$.
    (d) For each good rule $h$
        i. Convert $h$ into a Boolean feature $f$
        ii. $PosCover(h) = \{e : e \in E^+ \text{ and } B \cup h \models e\}$
        iii. $Left = Left - PosCover(h)$
        iv. $F = F \cup f$
5. done
6. return F
```

Now a "good" rule is taken to be one that satisfies some syntactic and semantic constraints (for example, precision and recall). The reader will recognize that Strong Sufficiency is only relevant to the $ConstructModel$ procedure (that is, $ConstructFeatures$ is not attempting to obtain a hypothesis that explains all the positive examples).

The conversion of an ILP rule to a Boolean feature is straightforward. Let us assume that we are constructing rules only for some class c (usually c would be the class of positive examples) and that a data instance is denoted nominally by $\mathbf{x}$ drawn from some space $\mathcal{X}$. Then a good rule will be of the form $h_j : Class(\mathbf{x}, c) \leftarrow Cp_j(\mathbf{x})$.[1] We adopt the terminology

[1] We note that in general, for ILP, $\mathbf{x}$ need not be restricted to a single object and can consist of arbitrary tuples of objects and rules constructed by the ILP engine for a class c would more generally be $h_j : Class(\langle \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \rangle, c) \leftarrow Cp_j(\langle \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \rangle)$.

from Ratnaparkhi (1999) and $Cp_j : \mathcal{X} \mapsto \{0, 1\}$ denotes a "context predicate". A context predicate corresponds to a conjunction of literals that evaluates to $TRUE$ (1) or $FALSE$ (0) for any element of $\mathcal{X}$. For meaningful features we will usually require that a Cp_j contain at least one literal; in logical terms, we therefore require the corresponding h_j to be definite clauses with at least two literals. A rule $h_j : Class(\mathbf{x}, c) \leftarrow Cp_j(\mathbf{x})$, is converted to a feature f_j using a one-to-one mapping as follows: $f_j(\mathbf{x}) = 1$ iff $Cp_j(\mathbf{x}) = 1$ (and 0 otherwise). We will denote this function as $Feature$. Thus $Feature(h_j) = f_j$, $Feature^{-1}(f_j) = h_j$. We will also sometimes refer to $Features(H) = \{f : h \in H$ and $f = Feature(h)\}$ and $Rules(F) = \{h : f \in F$ and $h = Features^{-1}(f)\}$.

The idea of constructing propositional representations from first-order ones goes back at least to 1990, with the LINUS system (and perhaps even earlier to work in the mid 1980s done by R.S. Michalski and co-workers). This specific use of rules constructed by ILP system as Boolean features appears to have been demonstrated first in Srinivasan and King (1996).

There is now a growing body of research that suggests that ILP-constructed relational features obtained in this manner can substantially improve the predictive power of statistical models (see the section on "Related Work" later in the paper). The principal difficulty is, of course, to determine how many, and which features are worth constructing for any particular kind of statistical model. In general, the number of possible relational features can be extremely large (even when confined to the world of Boolean propositional symbols, the number of rules and hence features, is exponential in the number of symbols).

To alleviate some of the difficulties just listed with feature-construction, we consider the possibility of arriving at a consensus model, using a distributed construction of small sets of features. The idea is shown diagrammatically in Fig. 2. Each ILP engine in the figure implements.

$ConstructFeatures(B, \mathcal{L}, E, f_{max})$ for some (small) value of f_{max}. That is, each ILP engine has access to all the background knowledge and examples (we will return to this requirement later). The set of features returned by the ILP engines at a pair of nodes may or may not overlap,[2] and each node constructs a local model using the features from its ILP engine.

In this paper, we are interested in the following question: is it possible to arrive at a consensus model, starting from multiple models using different feature-sets (which may have common elements). As we shall see, when each node constructs a local linear model, the iterations of a consensus-based algorithm (described next), result in all nodes in the network exchanging feature weights and moving to different states, but finally converging on the optimal weights for all the features.[3] On the face of it, this would appear to contradict the Fischer, Lynch, Paterson (FLP) result (Fischer et al. 1985) that asserts the impossibility of reaching consensus in a distributed system. The consensus problem described in FLP involves an asynchronous system of processes, some of which may be *unreliable*. The question then is: how can the reliable processes have a consistent view of the system? Our setting is similar, in that we are concerned with an asynchronous system of processes (nodes) which must have a consistent view of data stored on them (weights of the features). The process of local model construction assigns a set of weights to the features - these can get updated by the process of communication (gossip) with other nodes in the network. The work differs from the FLP setting, however, in that we assume that there are no failures in the distributed system ensuring that the FLP result is not violated.

[2] If redundant features are produced at a node, we rely on the model constructor to be able to identify this (for example, the weights of one would be zero, in the ideal case).

[3] In practice, we need not wait for such a convergence by all nodes, of course.

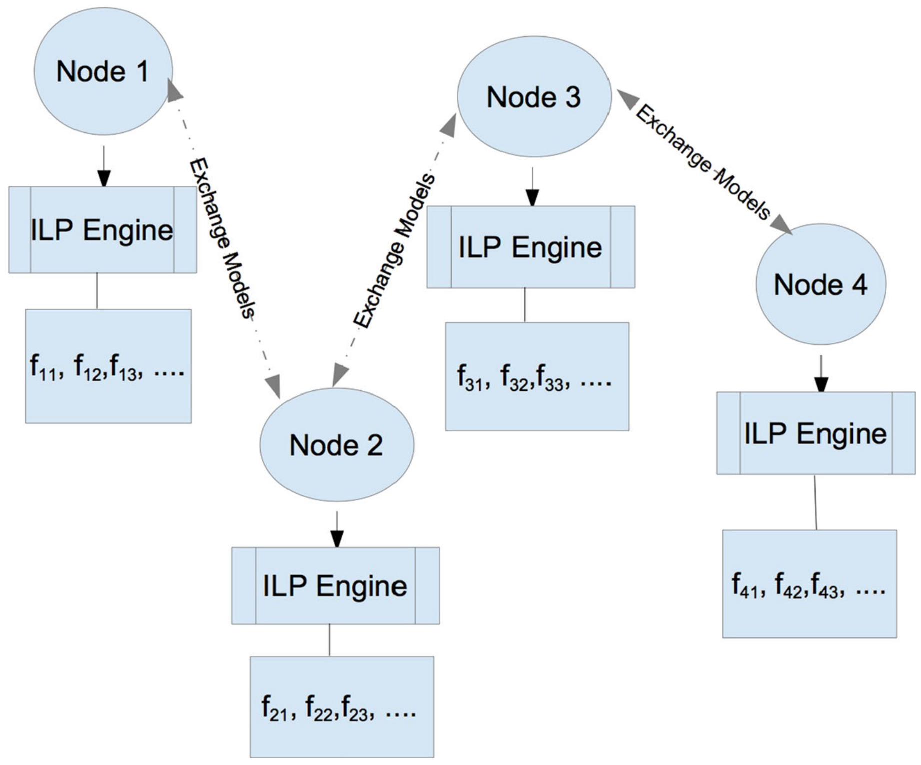


Fig. 2 An illustrative example of the consensus-based approach using Michalski's "Trains" problem (Larson and Michalski 1977). Each node has local features (for e.g., $f_1(T) = 1$, if has $\_car(T, C)$, short(C); 0 otherwise) generated from an ILP engine. In this figure, we use the notation, f_{11} indicates the first feature for Node 1, f_{21} the first feature of Node 2 and so on. It then builds local models, estimates loss and shares information with its neighbors. Eventually we would like all nodes to converge to the same model

3 An algorithm for consensus-based modeling

We use a general setting for consensus-based modeling, without any reference to ILP.

Let M denote an $n \times m$ matrix with real-valued entries. This matrix represents a dataset of n tuples of the form $X_i \in \mathbb{R}^m$, $1 \le i \le n$. 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 features, S_2 has m_2 features and so on, such that $|m_1| + |m_2| + \cdots + |m_k| = |m|$, where $|m_i|$ represents the number of features at site S_i.[4] 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.

We will restrict ourselves to learning a linear discriminative function over the data set M. The global function to be estimated is represented by $J_g = MW_g^T$ 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 $J_1 = M_1 W_1^T$. At site S_2, the local function estimated would be $J_2 = M_2 W_2^T$. 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

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

```
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:
    $\frac{\partial L_p}{\partial W_i} = -X_p(Y_p - J_i^t(W_i^t(p)))$;
    (e) If there is no significant change in the local weight vectors of one of the sites then stop
end
```
Algorithm 1: Distributed Feature Estimation (DFE) by Consensus-Based Modeling

$W_g[1 : m_i]$ 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.

3.1 Algorithm

Algorithm 1 makes the following assumptions:

1. Model of Distributed Computation. The distributed algorithm can be seen as evolving over discrete time with respect to a "global" clock. However, the existence of this clock is of interest only for theoretical analysis. Each site has access to a local clock. Furthermore, each site has its own memory and can perform local computation (such as computing the gradient on its local features). It stores J_i, which is the estimated local function. Besides its own computation, sites may receive messages from their neighbors which will help in the evaluation of the next estimate for the local function.
2. 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 sites S_i and S_j. Communication delays on the edges in the graph are assumed to be finite. 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.

Algorithm 1 describes how the weights for features 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 features and then gossips with its 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 its 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 α_{ij}, $0 \leq \alpha_{ij} \leq 1$, is a non-negative weight that captures the fraction of information site i is willing to share with

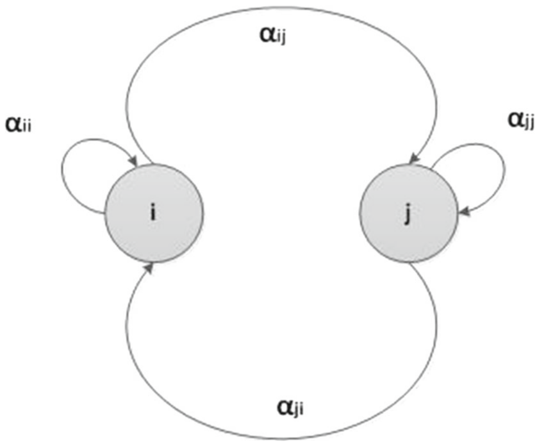


Fig. 3 State Transition Probability between two sites S_i and S_j

site j. The choice of α_{ij} may be deterministic or randomized and may or may not depend on the time t (Kempe et al. 2003). The $k \times k$ matrix A comprising of α_{ij}, $1 \leq i \leq k$, $1 \leq j \leq k$ 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. Furthermore, 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 (Varga 1962)) *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 (Varga 1962). 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 $\parallel J_i^{t+1} - \mathbf{J}_i^t \parallel^2 \leq \gamma^2 \parallel J_i^t - \mathbf{J}_i^t \parallel$. If $\gamma = 1$, then system fails to converge (Varga 1962; Cybenko 1989).
Local stochastic gradient update is done as follows: $W_i^{t+1} = W_i^t - \eta_i^t s_i^t$ where $s_i^t = \frac{\partial J_i^t}{\partial W_i^t}(X_r, W_i^t)$, $X_r \in \mathbb{R}^{m_i}$ 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.

It is evident that there are no restrictions on the features used by the DFE algorithm. The proofs of correctness and termination are in "Appendix", and we will henceforth refer to the procedure as the DFE algorithm. We now investigate empirically the performance of the algorithm when the nodes in the network use features constructed locally by an ILP engine.

4 Empirical evaluation

4.1 Aims

Our objective is to investigate empirically the utility of the consensus-based algorithm we have described. We use $Model(k, f)$ to denote the model returned by the consensus-based algorithm in Sect. 3 using k nodes in a network, each of which can call on an ILP engine to construct at most f features. In this section, we compare the performance of: $Model(N, F)$ $(N > 1)$ with $Model(1, N \times F)$. The latter effectively represents the model constructed in a non-distributed manner, with all features present at a single centralized node. For simplicity, we will call the former the $Distributed$ model and the latter the $Centralized$ model.

We intend to examine if there is empirical support for the conjecture that the performance of the $Distributed$ model is better than that of the $Centralized$ model. We are assuming that the performance of a model construction method is given by the pair (A, T) where A is an unbiased estimate of the predictive accuracy of the classifier, and T is an unbiased estimate of the time taken to construct a model. In all cases, the time taken to construct a model also includes the time taken to identify the set of features by the ILP engine and the time to compute their values. When $k > 1$, the time will also include time for exchanging information. Comparisons of pairs (A_1, T_1) and (A_2, T_2) will simply be lexicographic comparisons.

4.2 Materials

4.2.1 Data

Data for experiments are in two categories:

1. **Synthetic** We use the "Trains" problem posed by R. Michalski for controlled experiments. Datasets of 1000 examples are obtained for randomly drawn target concepts (see "Methods" below).[5] For this we use S.H. Muggleton's random train generator[6] that defines a random process for generating examples. We will use this data for controlled experiments to test the principal conjecture about the comparative performances of $Distributed$ and $Centralized$ models.

[5] We note here that we are not concerned with large numbers of examples here, since the main investigation is concerned with subsets of the feature-space, and not of the data instances.

[6] http://www.doc.ic.ac.uk/~shm/Software/GenerateTrains/.

Table 1 Dataset sizes and class distributions

Problem	Examples	% Positives
$Mut188$	188	≈ 66
$Canc330$	337	≈ 54
$DssTox$	576	≈ 38
$Amine$	686	50
$Choline$	1326	50
$Scop$	450	50
$Toxic$	886	50

The last four datasets have a 50:50 class distribution by design, since the data instances are pairwise comparisons of chemical activity. Thus, for every positive instance of the form $better(m1, m2)$, there is a negative instance of the form $\neg better(m2, m1)$

2. **Real** We report results from experiments conducted using 7 well-studied real world problems from the ILP literature. These are: Mutagenesis (King et al. 1996); Carcinogenesis (King and Srinivasan 1996); DssTox (Muggleton et al. 2008); and 4 datasets arising from the comparison of Alzheimer's drugs denoted here as $Amine$, $Choline$, $Scop$ and $Toxic$ (Srinivasan et al. 1996). The dataset characteristics are reported in Table 1. Our purpose in examining performance on the real-data is twofold. First, we intend to see if the use of linear models is too restrictive for real problems. Second, we would like to see if the results obtained on synthetic data are reflected on real-world problems. We note that for these problems predictive accuracy is the primary concern.

Language constraints We have mainly relied on the use of mode declarations to incorporate language restrictions (see Muggleton 1995 for a description of modes and their usage by a class of ILP systems). For the synthetic dataset of trains, here are some examples of mode declarations (we use the syntax introduced in Muggleton (1995):

```
:- modeh(1,train(+train)).
:- modeb(1,short(+car)).
:- modeb(1,closed(+car)).
:- modeb(1,long(+car)).
:- modeb(1,open_car(+car)).
:- modeb(1,double(+car)).
:- modeb(1,jagged(+car)).
:- modeb(1,shape(+car,#shape)).
:- modeb(1,load(+car,#shape,#int)).
:- modeb(1,wheels(+car,#int)).
:- modeb(*,has_car(+train,-car)).
```

We follow the Aleph manual (Srinivasan 1999) for the meaning of restrictions. All declarations are of the form `mode(RecallNumber,PredicateMode)`.[7] Here `RecallNumber` bounds the non-determinacy of a form of predicate call, and `PredicateMode` specifies a legal form for calling a predicate. `RecallNumber` can be either (a) a number specifying the number of

[7] ILP engines are often used to hypothesize clauses of the form $Head \leftarrow Body$, where $Head$ is a literal, and $Body$ a conjunction of literals. `modeh` specifies the form of the $Head$ literal, and `modeb` specify forms for literals that can appear in $Body$.

successful calls to the predicate; or (b) a $*$, specifying that the predicate has bounded non-determinacy. PredicateMode is a template of the form: p(ModeType, ModeType,...) where ModeType is either (a) simple; or (b) structured. A simple ModeType is one of: +T , which means that when a literal with predicate symbol p appears in a hypothesized clause, the corresponding argument should be an "input" variable of type T; -T, which means that that the corresponding argument is an "output" variable of type T; or #T, which means that the corresponding argument should have a constant of type T. A structured ModeType is of the form: f(...) where f is a function symbol, each argument of which is either a simple or structured ModeType.

Some examples of mode declarations for the $Mut188$, $Canc330$ and $DssTox$ datasets:

```
:- modeh(1,active(+drug)).

:- modeb(*,bond(+drug,+atomid,+atomid,#bondtype)).
:- modeb(*,bond(+drug,+atomid,-atomid,#bondtype)).

:- modeb(*,atm(+drug,-atomid,#element,#atmtype,-charge)).

:- modeb(1,gteq(+charge,#charge)).
:- modeb(1,lteq(+charge,#charge)).

:- modeb(*,benzene(+drug,-ring)).
:- modeb(*,carbon_5_aromatic_ring(+drug,-ring)).
:- modeb(*,carbon_6_ring(+drug,-ring)).
:- modeb(*,hetero_aromatic_6_ring(+drug,-ring)).

:- modeb(*,anthracene(+drug,-ringlist)).
:- modeb(*,phenanthrene(+drug,-ringlist)).
:- modeb(*,ball3(+drug,-ringlist)).

:- modeb(1,member(+ring,+ringlist)).
:- modeb(1,connected(+ring,+ring)).

(and so on)
```

For reasons of space, we do not show the mode declarations for the other datasets.

4.2.2 Algorithms and machines

The DFE algorithm has been implemented on a Peer-to-Peer simulator, PeerSim (Montresor et al. 2009). This software sets up the network by initializing the nodes and the protocols to be used by them. The newscast protocol, an epidemic content distribution and topology management protocol is used. Nodes can perform actions on local data as well as communicate with each other by selecting a neighbor to communicate with (using an underlying overlay network). In each communication step, they mutually update their approximations of the value to be calculated, based on their previous approximations. The emergent topology from a newscast protocol has a very low diameter and is very close to a random graph (Jelasity et al. 2004, 2005).

The ILP system used in all experiments is Srinivasan (1999). The latest version of this program (Aleph 6) is available from the second author. We use Aleph to construct features [specifically, the `induce_features` command in that program: the precise description of how this is done is in Srinivasan (1999)]. To a good approximation, the procedure is as described in Sect. 2.2.1 (some small difference arises from the Aleph implementation using a class-based upper-bound on the number of features). The Prolog compiler used is Yap[8] (version 6.2.0). The programs are executed on a dual Quad-Core AMD Opteron 2384 processors equipped with 2.7 GHz processors, 32 GB RAM, and local storage of 4 × 146 GB 15K RPM Serial attached SCSI (SAS) hard disks.

4.3 Method

For the synthetic data, classification tasks are randomly constructed for the DFE algorithm based on disjunctive concepts. "Simple" concepts have 1–4 disjuncts, and "complex" ones between 8–12 disjuncts.[9] For any one classification task a data instance x is defined as "positive" is some underlying concept is true for x. If the underlying concept is a simple concept, then the classification task is said to have a simple target; otherwise it has a complex target. Classification tasks are constructed randomly as follows:

1. A concept ("Target") is generated by:
   (a) Randomly obtaining the number of disjuncts k;
   (b) Drawing k features from a population of features; and
   (c) Defining the concept as the disjunction of the k features.
2. A binary classification task is then defined using the concept constructed, and positive and negative instance are generated randomly as training data for this classification task.

Figure 4 shows an example of the relationship between features, concepts and targets. We will refer to the Steps (a)–(c) as "randomly drawing a target concept." Of course, when given the training data, the DFE algorithm does not know the features used in the target concept. Instead, nodes in the network use the data, background knowledge, and their ILP engines to construct local features and a consensus model is obtained to discriminate between positive and negative examples. We note that it may not be sufficient simply to identify one of the k disjuncts in a Target. With sufficient data, there will be instances where each one of the disjuncts is $FALSE$. Thus, a node in the distributed setting that correctly identifies one of the features can still have a poor accuracy on the training data.

Our method for experiments is straightforward:

1. For each kind of concept ("simple" or "complex")
   (a) Randomly draw a target concept
   (b) Classify each data instance as $+$ or $-$ using the target concept
   (c) Randomly generate a network with N nodes
   (d) For each node in the network:
      i. Set the number of iterations T and initialize the learning parameter η_i for the node. It is assumed that all nodes agree on the initial choice of T and $\eta_i = \eta$.
      ii. Execute the algorithm described in Sect. 1 for T iterations and the ILP engine restricted to constructing F features
      iii. Record the predictive accuracy A of the (local) model along with the time T taken to construct the model (this includes the feature construction time,

[8] http://www.dcc.fc.up.pt/~vsc/Yap/.

[9] This distinction between simple and complex is based on results from cognitive psychology which suggest that people find it difficult to remember concepts with larger than 7 disjuncts (Michie et al. 1990).

$$F_1(x) = \begin{cases} TRUE \text{ if } (HasCar(x,y) \wedge Closed(y)) \\ FALSE \text{ otherwise} \end{cases}$$

$$F_2(x) = \begin{cases} TRUE \text{ if } (HasCar(x,y) \wedge HasTriangleLoad(y,2) \wedge HasWheels(y,1)) \\ FALSE \text{ otherwise} \end{cases}$$

$$F_3(x) = \begin{cases} TRUE \text{ if } (\ldots) \\ FALSE \text{ otherwise} \end{cases}$$

(and so on)

$$Simple_1(x) = \begin{cases} TRUE \text{ if } (F_1(x) \vee F_2(x)) \\ FALSE \text{ otherwise} \end{cases}$$
$$Complex_1(x) = \begin{cases} TRUE \text{ if } (F_1(x) \vee F_3(x) \vee F_5(x) \vee F_6(x) \vee F_{11}(x) \cdots) \\ FALSE \text{ otherwise} \end{cases}$$
$$Positive(x) = \begin{cases} TRUE \text{ if } (Simple_1(x) = TRUE) \\ FALSE \text{ otherwise} \end{cases}$$

Fig. 4 Classification problems using simple and complex concepts. The features F_1, F_2, …are functions whose values are $TRUE$, depending on the conditions in their definitions. Simple concepts have 1–4 disjuncts, and complex ones between 8–12 disjuncts. Binary classification tasks for the DFE algorithm are based on simple or complex concepts: a data instance x is defined as "positive" is some underlying concept is true for x (shown here for $Simple_1$). If the underlying concept is a simple concept, then the classification task is said to have a simple target; otherwise it has a complex target. For each classification task positive and negative instances are generated randomly and provided as training data to the DFE algorithm

and the feature computation time). The pair (A, T) is the performance of the *Distributed* model for the concept.

(e) Using a network with a single node:
   i. Execute the algorithm described in Sect. 1 for T iterations, learning parameter η, and the ILP engine restricted to constructing $N \times F$ features
   ii. Record the predictive accuracy A' of the model along with the time taken to construct the model T' (again, this includes the feature construction time and feature computation time). The pair (A', T') is the performance of the *Centralized* model for the concept.

2. Compare the performances of the *Distributed* and the Centralized models for the concepts.

The following additional details are relevant:

1. Two sources of sampling variation result with this method. First, variations are possible with the target drawn in Step 1a. Second, to ensure that both the *Distributed* and *Centralized* approaches are constructing features from the same feature-space, we employ the facility within Aleph of drawing features from an explicitly defined feature space (this is specified using a large tabulation of features allowed by the language constraints). In effect, we are performing a randomized search for good features within a pre-defined feature space. Although only "good" features are retained (see below), even after controlling for feature-spaces, sampling variations can nevertheless result for both

the *Distributed* and *Centralized* models from the step of drawing features. We report averages for 5 repetitions of draws for the target, and 5 repetitions of the randomized search for a given target.

2. A target is generated as follows. For simple targets, the number of features is chosen randomly from the range 1–4. For complex concepts, the number of features is randomly chosen from the range 8–12. Features are then randomly constructed using the ILP engine, and their disjunction constitutes the target concept.
3. As noted previously, data instances for controlled experiments are drawn from the "Trains" problem. The data generator uses S.H. Muggleton's random train generator. This implements a random process in which each data instance generated contains the complete description of a data object (nominally, a "train").
4. An initial set of parameters needs to be set for the ILP engine to describe "good" features. These include C, the maximum number of literals in any acceptable clause constructed by the ILP system; Nodes, the maximum number of nodes explored in any single search conducted by the ILP system; Minacc, the minimum accuracy required of any acceptable clause; and Minpos, the minimum number of positive examples to be entailed by any acceptable clause. C and Nodes are directly concerned with the search space explored by the ILP system. Minacc and Minpos are concerned with the quality of results returned (they are equivalent to "precision" and "support" used in the data mining literature). We set $C = 4$, Nodes=5000, Minacc=0.75 and Minpos=2 for our experiments here. There is no principled reason for these choices, other than that they have been shown to work well in the literature (Srinivasan and Ramakrishnan 2011).
5. The parameters for the PeerSim simulator include the size of the network, degree distribution of the nodes and the protocol to be executed at each node. We report here on experiments with a distributed network with $N = 10$ nodes. Each of these nodes can construct up to $F = 500$ features (per class) and the centralized approach can construct up to $N \times F = 5000$ features (per class).
6. The experiments here use the Hinge loss function. The results reported are for values of T that the stochastic gradient descent method starts to diverge.
7. The learning rate η_i remains a difficult parameter in any SGD-based method. There is no clear picture on how this should be set. We have adopted the following domain-driven approach. In general, lower values of the learning rate imply a longer search. We use three different learning rates corresponding to domains requiring high, moderate and low amounts of search (corresponding to complex, moderate or simple target concepts). The corresponding learning rates are 0.01, 0.1 and 1. We reiterate that there is no prescribed method for deciding these values, and better results may be possible with other values. The maximum number of iterations T is set to a high value (1000). The algorithm may terminate earlier, if there are no significant changes to its weight vector.
8. Since the tasks considered here are binary classification tasks, the performance of the ILP system in all experiments will be taken to be the classification accuracy of the model produced by the system. By this we mean the usual measure computed from a 2×2 cross-tabulation of actual and predicted classes of instances. We would like the final performance measure to be as unbiased as possible by the experimental estimates obtained during optimization, and estimates are reported on a holdout set.
9. With results from multiple repetitions (as we have here), it is possible to perform a Wilcoxon signed-rank test for both differences in accuracy and differences in time. This allows a quantitative assessment of difference in performance between the Distributed

Table 2 Results on synthetic data comparing *Centralized* and *Distributed* models

Model	*Simple*		*Complex*	
	Acc. (%)	Time (s)	Acc. (%)	Time (s)
(a)				
Centr.	83.3 (14.4)	0.12 (0.07)	92.0 (7.1)	0.18 (0.03)
Distr.	93.4 (10.5)	0.06 (0.07)	98.7 (1.2)	0.04 (0.03)
(b)				
Centr.	83.6 (20.1)	0.14 (0.07)	95.3 (0.6)	0.21 (0.02)
Distr.	79.9 (10.4)	0.15 (0.07)	96.6 (0.6)	0.06 (0.02)

The results in (a) are averages from repetitions across concepts; and in (b) are averages from repetitions of the feature-construction process for a randomly drawn concept. In all cases, *Distributed* denotes the model obtained with a 10 node network, each of which employs a randomized search for up to 500 good features. *Centralized* denotes the model obtained with a single node employing a randomized search for up to 5000 good features. All randomized searches draw features from the same feature-space

and the Centralized models. However, results with 5 repetitions are unreliable, and we prefer to report on a qualitative assessment, in terms of the average of accuracy and time taken.

A data instance in each of the real datasets is a molecule, and contains the complete description of the molecule. This includes: (a) bulk properties, like molecular weight, logP values etc.; and (b) the atomic structure of the molecule, along with the bonds between the atoms. For these datasets, clearly there are no concepts to be drawn, and sampling variation results solely from the feature-construction process. We therefore only report on experimental results obtained from repeating the randomized search for features. Again, estimates of predictive accuracy are obtained from a holdout set. For mutagenesis and carcinogenesis, each of the 10 computational nodes in the distributed network constructs up to 500 features, and the centralized approach constructs up to 5000 features (per class). For DssTox, we found there were fewer high precision features than the other two datasets. So the nodes in the distributed network constructs up to 50 features and the centralized node up to 500 features (per class).

4.4 Results

We present first, the main results from the experiments on synthetic data (shown in Table 2). The primary observations in these experiments are as follows: (1) On average, as concepts vary, the distributed algorithm appears to achieve higher accuracies than the centralized approach, although the differences may not be significant for a randomly chosen concept; (2) On average, as concepts vary, the time taken for model construction by the distributed approach can be substantially lower[10]; and (3) The variation in both accuracies and time with the distributed approach due to both changes in the concept, or due to repetitions of feature-construction appear to be less than the centralized approach.

[10] Although not apparent in the tabulation, the time is dominated by the time for constructing features (we present empirical results in support of this later in this section). As a result, we note that the ratio of times for the centralized and distributed approaches need not be (linearly) proportional to the number of nodes in the network. For example, the search for a large subset of good features conducted by the centralized approach may take much longer than the search for several small subsets conducted by the distributed approach.

Taken together, these results suggest that good, stable models can be obtained from the distributed approach fairly quickly, and that the approach might present an efficient alternative to a centralized approach in which all features are constructed by a single computational unit.

At this point a question could be raised on the value of the synthetic data. There are at least 3 issues here:

- First, why bother with synthetic data at all? The answer to this is that it gives us the opportunity to perform controlled experiments, of the kind that would be impossible with real-world datasets.
- Secondly, why use the “trains” problems? The trains problems are a well-known benchmark in the ILP literature, and there is an easily available simulator that is capable of generating new data instances by random draws from a known distribution (each instance is a random train with the number of carriages in the train following a multinomial distribution). There is precedence in the ILP literature of using this to test algorithms: for example, Cardoso & Zaverucha used a synthetic trains dataset of 1.25 million examples to evaluate their methods, all of which achieved 100% accuracy on the synthetic dataset.
- Thirdly, why are we getting theories with high accuracies? The targets are randomly generated k-disjuncts with different values of k (1–4 for “simple” targets and 8–12 for “complex” targets). Further, each feature is a rule restricted to Datalog without recursion, a bound on the maximum number of literals, each of which is a predicate with a fixed maximum arity, the hypothesis space is clearly bounded and therefore learnable (in the PAC-sense) with arbitrarily high accuracy and confidence, given sufficient examples. It is also known theoretically that Winnow can identify a linear threshold function for k-disjuncts making a small number of mistakes. So, with sufficiently large amounts of data (recall we use 1000 training instances here), it is not surprising that high accuracies are obtainable. What is surprising is that the accuracy on simple concepts is lower than on complex concepts. This suggests that there must be more simple concepts that are consistent with the target on the training data than complex ones.

What can we expect from the consensus-based learner on the real datasets? Results are in Table 3, and we observe the following: (1) There is a significant difference in accuracies between the distributed and centralized models on two of the datasets ($Canc330$ and $DssTox$). On balance, we cannot conclude from this that either one of the models is better;

Table 3 Results on real data comparing $Centralized$ and $Distributed$ models

Problem	Acc (%)			Time (s)		
	Centr.	Distr.	Base.	Centr.	Distr.	Base.
$Mut188$	84.3 (2.6)	76.8 (0.0)	84.6 (2.6)	1.93 (0.53)	0.42 (0.02)	–
$Canc330$	67.6 (0.5)	56.8 (0.5)	50.4 (2.8)	3.56 (0.58)	0.82 (0.18)	–
$DssTox$	53.8 (0.0)	61.6 (1.0)	64.7 (2.0)	1.94 (0.91)	0.76 (0.48)	–
$Amine$	75.5 (0.4)	76.6 (0.8)	71.4 (1.7)	7.02 (0.24)	0.87 (0.11)	–
$Choline$	72.9 (1.3)	71.2 (0.7)	52.7 (1.4)	13.00 (3.06)	1.64 (0.5)	–
$Scop$	67.2 (1.5)	69.9 (0.3)	55.1 (2.0)	11.9 (0.64)	1.20 (0.03)	–
$Toxic$	76.3 (1.3)	81.4 (0.2)	79.2 (1.4)	7.44 (3.51)	0.89 (0.49)	–

The Baseline models are the ones reported in Srinivasan and Ramakrishnan (2011) (these are cross-validation estimates, whereas the estimates for $Centralized$ and $Distributed$ models are from holdout sets). No estimates of time are reported in that paper

(2) As with the synthetic data, the time for the distributed models is substantially lower. As before, the time is dominated by the feature-construction effort. For the real data sets, it appears that it is substantially easier to get smaller subsets of good features than larger ones (as observed from the differences in the times between the distributed and centralized models); and (3) Comparisons against the baseline suggest that the use of linear models is not overly restrictive, since the models obtained are not substantially worse (predictively speaking) than the ones obtained by ILP literature in the past. The *Distributed* approach does better than *Baseline* on 5 of 7 datasets: this presents some evidence supporting the case for the former, although numbers are not large enough to claim statistical significance. Quantitatively, the differences in predictive accuracies between *Distributed* and *Centralized* are not statistically significant, although there is evidence that differences in time is statistically significant (*Distributed* is faster).

Again, taken together, these results provide support to the trends observed with the controlled experiments and suggest that the distributed approach would continue to perform at least as well as the centralized approach on real data.

4.5 Supplementary results

We turn now to some issues that have been brought out by the experimental results. We present immediate practical issues first, and then examine some more abstract questions.

4.5.1 The learning rate λ.

As with all methods based on stochastic gradient descent, the central parameter remains the learning rate λ. Many strategies have been suggested in literature to automatically adjust the learning rates. (see for example Bottou 2010; Bottou and Bousquet 2011; Darken and Moody 1990; Sutton 1992). In general, the learning rate on an iteration η_i of the algorithm here is of the form $\eta_i = \frac{B}{T^{-\alpha}}$; $B = e^{-\lambda T}$; $0 < \alpha \leq 1$. Table 4 shows the effect of varying λ for the synthetic datasets used here. These results show that the determination of λ is a tricky business, that can depend on the nature of the target theory being approximated (correctly, it is really to do with the amount of search needed). For the experiments reported above, we have used fixed values of λ based on our assessment of the search required (see the additional details in the "Methods" section). That λ is dataset dependent and needs to assigned in some domain-dependent manner appears to be an unavoidable aspect of any SGD-based method.

Table 4 Effect of the learning rate λ

λ	*Acc* (%)	*Time* (s)
(a) Simple		
0.01	92.4	0.05
0.1	92.4	0.05
1	93.4	0.06
(b) Complex		
0.01	98.7	0.04
0.1	95.6	0.05
1	77.8	0.37

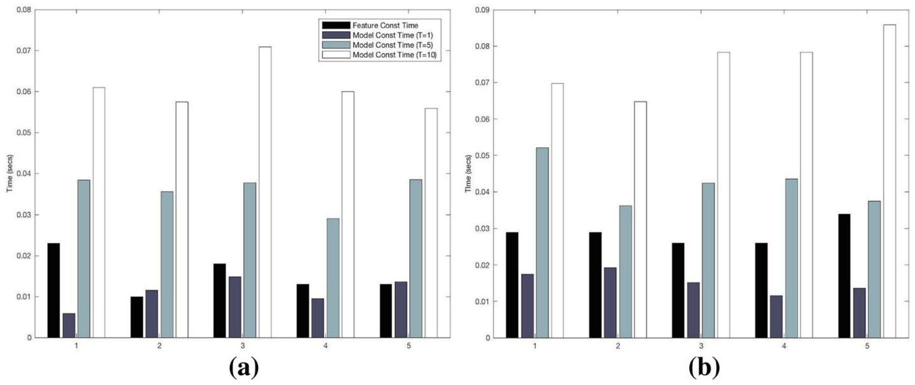


Fig. 5 Feature and model construction times for randomly drawn concepts. The plots are averages over 5 repetitions. **a** Simple, **b** Complex

4.5.2 Disaggregated time

The time estimates reported in the tabulations consist of four separate components. These are: (a) feature-construction; (b) feature-evaluation; (c) model-construction; and (d) inter-node communication. Components (a)–(c) are part of both centralized and distributed learners, and it is of some interest to examine their separate contributions. Feature evaluation is very small for both simple and complex trains ($< 0.01\%$). Feature-construction is roughly 99% of the time spent in feature engineering.

We note that all experiments reported here use the PeerSim simulator, which does have provisions for modeling the transport layer via a special protocol that provides a message sending service. There are also options for modeling latency among geographically distributed nodes in addition to churn models for nodes. None of these aspects have been explored here, and the tabulation reported here use default settings of the simulator that are designed to model a network as a random graph. It is nevertheless evident that distributed model construction is only worthwhile provided communication costs are low—there will be real networks (and corresponding simulator settings) for which distributed model-construction can be a good or a bad approach.

Figure 5 depicts the variation in the feature and model construction time repetitions of randomly drawn concepts. For model construction, the number of iterations T is varied between 1, 5 and 10. In all the cases, a single iteration of the stochastic gradient descent algorithm is good enough to obtain reasonably good accuracy in the distributed settings. Also, for small values of T, the feature construction time dominates communication costs.

4.5.3 Consensus by union of features

A natural question that emerges at this point is this: "How does the algorithm proposed here compare against one that achieves a trivial consensus simply by a union of all the features found by nodes in the network?" Such a consensus would be arrived as follows. Let us assume without loss of generality, that each node gets their own sample of features and sends it to one centralized site that is responsible for the "Union" operation. Both the test and train data will, of course, be concatenated vertically, but there are several ways to arrive at the union of features:

Table 5 Results on synthetic data for a consensus-classifier using the union of feature-sets identified at each node in a distributed network

Model	*Simple*		*Complex*	
	Acc. (%)	Time (s)	Acc. (%)	Time (s)
Union − lb.	93.42 (4.71)	0.02 (0.01)	95.66 (5.93)	0.02 (0.01)
Union − ub.		0.14 (0.05)		0.17 (0.04)

The results correspond to averages from repetitions across concepts. In all cases, "Union" denotes the model obtained by vertically concatenating data from all the nodes in network

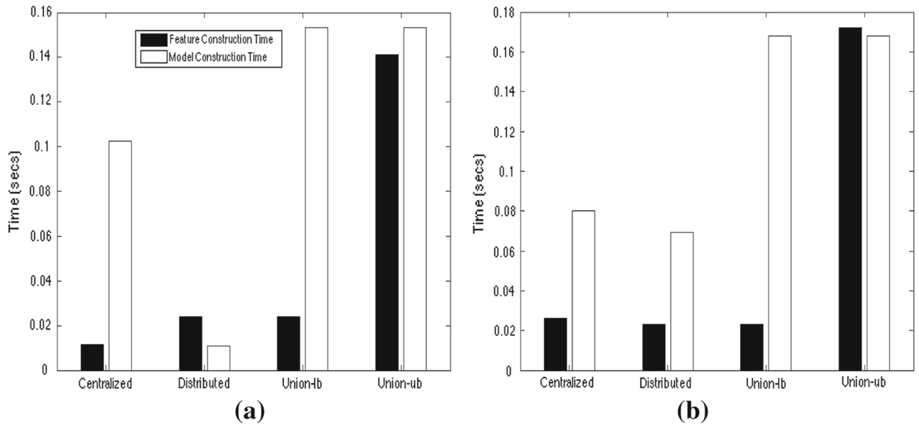


Fig. 6 Feature and model construction time for randomly drawn simple and complex targets. Lower- and upper-bounds for a consensus-classifier that uses the union of features constructed by nodes in a distributed network (Union-lb and Union-ub respectively) are compared against the centralized and distributed models. **a** Simple, **b** complex

(a) Features are generated and sent sequentially, with some predefined notion of ordering amongst nodes (node 1 goes first then node 2 and so on). This helps to provide an empirical upper-bound for the time required to compute the union of features. We assume that the upper bound is the sum of feature construction times at each distributed node. and hence referred to as *Union-ub*;

(b) Features are generated in parallel synchronously. That is, all nodes generate a set of features, starting at the same time. We refer to this as *Union-lb* corresponding to the lower bound; and

(c) Features are generated in parallel, but asynchronously. That is, nodes generate sets of features, not necessarily starting at the same time. In this case, there is no straightforward way to compute a bound on the time to compute features, since nodes can elect to commence feature-construction at any point in time.

The results presented in Table 5 provide the accuracy and time for building the model under the union operation. In general, the model(s) built on the "Union" of features does appear to have comparable performance to the distributed algorithm presented here, but this may come at a price if a sequential operation is done to obtain the union of features. Figure 6 further analyzes the time by dividing it into feature and model construction time(s).

4.5.4 Network topology

Experiments have been designed to test the effect of the network topology on the convergence rate of the algorithm using synthetic data. Assuming K to be the number of outgoing edges from a node, three different strategies of node addition are explored: (a) **Random:** K nodes are added at random; (b) **Star:** K nodes are added in a star topology; and (c) **Scale Free Network:** A scale-free network is grown using the Barabasi-Albert (BA) model ensuring that nodes have power law degree distributions. Two values of $K = 2, 8$ were used in empirical analysis. Our results indicate that there were no statistically significant difference in performance for the 10 node network(s) studied in this paper.

4.5.5 Convergence accuracy

In experiments here (both with synthetic and real data), we have observed that the predictive accuracy of the model from the distributed setting is comparable to the predictive accuracy from the non-distributed setting. Unlike gains in time which can be expected from a distributed setting, it is not evident beforehand what can be expected on the accuracy front. This is because the models constructed in the two settings can, and usually do, sample different sets of features. The results here suggest a conjecture that the consensus-based approach will always converge to a model that is within some small error bound of the model from a centralized approach with the same number of features. We have some reason to believe that this conjecture may hold in some circumstances, based on the use of Sanov's theorem (Sanov 1957) and related techniques.

5 Related work

We review related work from several areas: large scale feature selection, decentralized optimization and consensus-based learning, and discovery of feature subsets by ILP engines.
Large scale feature selection Techniques for selecting from a (large) but finite set of features of known size d[11] have been well-studied within the machine learning community, usually under the umbrella-terms of filter-based or wrapper-based methods (see for example, John et al. 1994; Liu and Motoda 1998). While most of the early work was intended for implementation on a single machine, several algorithms have been proposed to enable feature-selection from massive datasets (Garcia et al. 2006; Lopez et al. 2006; Sun 2014). Singh et al. (2009) propose a framework for handling feature selection for logistic regression by developing a new forward feature selection heuristic that ranks features by their estimated effect on the resulting model's performance. Zhao et al. (2012) describe an algorithm that selects features based on their ability to explain data variance. Zhou et al. (2014) present a framework for Parallelizable Feature Selection (PFS) which is inspired by the theory of group testing. Group testing is a combinatorial search paradigm where the goal is to identify a small subset of relevant items from a large pool of possible items. The feature selection problem in the group testing framework applies a "test" to a set of features which produces a score designed to measure the quality of the features. From the collection of test scores the relevant features are supposed to be identified. PFS has several similarities to the algorithm proposed in this paper - notably, the set of features at each node can be viewed as a collection

[11] A large feature set increases the size of the search space, making it more difficult for the learning algorithm to find a near optimal solution.

of *relevant features* for group testing; the process of local function evaluation can be mapped to the "test" required in group testing. The end product, however, in the two algorithms is fundamentally different—in PFS, all feature sets are identified in advance without knowing the scores of other tests and a final subset of features is discovered; nodes executing the DFE algorithm have updated scores of the local features after gossiping with neighbors. The updated scores help learn approximations of the global objective and nodes independently reach a consensus.

Furthermore, distributed computation is utilized to solve optimization problems that arise when exploring huge, nonlinear and multidimensional search spaces. Distributed feature selection algorithms often solve decentralized optimization problems in the constrained and unconstrained settings (seminal work of Tsitsiklis et al. 1986; Tsitsiklis 1984; Bertsekas and Tsitsiklis 1997). The convergence properties of these decentralized optimization problems naturally affect the performance of the distributed feature selection algorithms. In recent work, it has been shown that convergence properties of distributed optimization algorithms of unconstrained optimization algorithms (such as gradient descent and its stochastic variants) can be related to the network topology of the underlying distributed infrastructure by using its spectral properties (Boyd et al. 2006; Shah January 2009; Dimakis et al. 2006; Benezit et al. 2010).

Distributed optimization Learning feature subsets in distributed environments using decentralized optimization has become an active area of research (Duchi et al. 2012; Agarwal et al. 2014; Christoudias et al. 2008) in recent years. Agarwal et al. (2014) present a system and a set of techniques for learning linear predictors with convex losses on terabyte sized datasets. Their goal is to learn problems of the form $\min_{w \in R^d} \sum_{i=1}^{n} l(w^T x_i; y_i) + \lambda R(w)$ where x_i is the feature vector of the i^{th} example, w is the weight vector and R is a regularizer. The data are split horizontally and examples are partitioned on different nodes of a cluster. Duchi et al. (2012) present a dual averaging sub-gradient method which maintains and forms weighted averages of sub-gradients in the network. An interesting contribution of this work is the association of convergence of the algorithm with the underlying spectral properties of the network. Similar techniques for learning linear predictors have been presented elsewhere (Mangasarian 1995; Ryan et al. 2010; Zinkevich et al. 2010; Niu et al. 2011; Boyd et al. 2011). The algorithm presented in this paper differs from this body of literature in that the data are split *vertically* amongst nodes in the cluster thereby necessitating a different algorithm design strategy. In addition, this is a batch algorithm and hence quite different from distributed online learning counterparts (Dekel et al. 2012; Langford t al. 2009; Bottou and Bousquet 2011). Das et al. (2010) show that three popular feature selection criteria—misclassification gain, gini index and entropy can be learnt in a large peer-to-peer network. This is then combined with protocols for asynchronous distributed averaging and the secure sum protocols to present a privacy preserving asynchronous feature selection algorithm.

Discovery of feature subsets by ILP engines Existing literature on discovering a subset of interesting features from large, complex search spaces such as those by ILP engines adapt one of the following strategies:

1. Optimally (Han and Wang 2009; Nowozin et al. 2007; Kudo et al. 2004) or heuristically (Joshi 2008; Saha et al. 2012; Specia et al. 2009; Ramakrishnan et al. 2007; Specia et al. 2006; Nagesh et al. 2012; Chalamalla et al. 2008) solve a discrete optimization problem.
2. Optimally (Jawanpuria et al. 2011; Nair et al. 2012) solve a convex optimization problem with sparsity inducing regularizers;

3. Compute all relational features that satisfy some quality criterion by systematically and efficiently exploring a prescribed search space (Pei 2004; Ji et al. 2006; Aseervatham et al. 2006; Antunes and Oliveira 2003; Pei et al. 2005; Agrawal and Srikant 1995; Pei et al. 2004; Ayres et al. 2002; Garofalakis et al. 1999; Davis et al. 2005a, b; Landwehr et al. 2007; Davis et al. 2007; Džeroski 1993; Lavrac and Dzeroski 1993).

Again, much of this has been of a non-distributed nature, and usually assumes a bound on the size of the feature-space. The latter is not the case for a technique like the one proposed in Joshi (2008). This describes a randomized local search based technique which repeatedly constructs features and then performs a greedy local search starting from this subset. Since enumeration of all local moves can be prohibitively large, the selection of moves is guided by errors made by the model constructed using the current set of features. Nothing is assumed about the size of the feature-space, making it a form of vertical partitioning of the kind we are interested in. Multiple random searches can clearly be conducted in parallel (although this is not done in the paper) (Zeleny et al. 2006; Fonseca et al. 2005; Dehaspe and De Raedt 1995). As with most randomized techniques of this kind, not much can be said about the final model.

Perhaps of most interest to the work here is the Sparse Network Of Winnow (SNoW) classifiers described in Roth (1998), Carlson et al. (1999). As it stands, this horizontally partitions the data into subsets, constructs multiple linear models using Winnow's multiplicative update process, and finally uses a majority vote to arrive at a consensus classification. This would appear, on the surface to be quite different to what we propose here. Nevertheless, there are reasons to believe that this approach can be usefully extended to the setting we propose. It has been shown elsewhere that the Winnow-based approach can be extended to an infinite-attribute setting (Blum 1992). The work in this paper shows that consensus linear models are possible when convex cost functions are used. Finally, from the ILP-viewpoint, (Srinivasan and Bain 2014) shows how it is possible to construct Winnow-based models in an infinite-attribute setting using an ILP engine with a stream-based model of the data. Taken together, this suggests that a combination of the techniques we propose, and those in Carlson et al. (1999) can be used to develop linear models that can handle both horizontal partitioning of the data and vertical partitioning of the feature-space.

Learning k-disjuncts Although the technique we have described here is not specifically aimed at learning k-disjuncts, the synthetic $Simple$ and $Complex$ datasets are both of this nature. It is instructive, therefore, to note some theoretical results that have been presented in the literature on identifying such concepts. First, note that a k-term DNF formula is a disjunction of k terms, where each term is a conjunction of literals. The k-term DNF learning problem can be described as follows: Given (a) a set of Boolean variables Var_i (b) a set *Pos* of truth value assignments $p_i : Var \rightarrow \{0, 1\}$ (c) a set *Neg* of truth value assignments $n_i : Var \rightarrow \{0, 1\}$ and (d) a natural number k–the goal is to find a k-term DNF formula that is consistent with *Pos* and *Neg*. In general, this is an NP-hard problem and exhaustive algorithms are impractical from a computational standpoint. Several stochastic local search based algorithms (Rückert and Kramer 2003; Rückert et al. 2002) have been proposed for DNF learning by reducing the k-term DNF problem to the well-known SAT problems. Among the more popular algorithms are Winnow like classifiers, which are known to make $O(k\, log\, n)$ mistakes before converging on a monotone k-disjunct formula (Littlestone 1988).

6 Conclusion

A particularly effective form of Inductive Logic Programming has been its use to construct new features that can be used to augment existing descriptors of a dataset. Experimental studies reported in the literature have repeatedly shown that the relational features constructed by an ILP engine can substantially assist in the analysis of data. Models constructed in this way have looked at both classification and regression, and improvements have resulted in each case. Practical difficulties have remained to be addressed though. The rich language of first-order logic used by ILP systems engenders a very large space of possible new features. The resulting computational difficulties of finding interesting features is not easily overcome by the usual ILP-based methods of language bias or constraints. In this paper, we have introduced what appears to be the first attempt at the use of a distributed algorithm for feature selection in ILP which also has some provable guarantees of convergence. The experimental results we have presented suggest that the algorithm is able to identify good models, using significantly lesser computational resources than that needed by a non-distributed approach.

There are a number of ways in which the work here could be extended further. Conceptually, we have outlined a conjecture in the previous section that we believe is worth investigating further. If it is proven to hold, then this would be a first-of-its-kind result for consensus-based methods. In implementation terms, we are able to extend the approach we have proposed to other kinds of models that use convex loss functions, and to consider a consensus-based version of the SNoW architecture. This latter will give us the ability to partition very large datasets, and to deal with very large feature-spaces at once. It is also not required within the approach that all computational nodes draw from the same feature space (this was a constraint imposed here to evaluate the centralized and distributed models in a controlled manner). It may be both interesting and desirable for nodes to sample from different feature-spaces, or with different support and precision constraints. We note also that the new version of Apache Cassandra uses a peer-to-peer setup and it would be useful to investigate the implementation of the algorithm we have proposed on a real, distributed system with commodity components. This will allow us to validate results obtained in simulation on gains in time when using the distributed approach. Experimentally, we recognize that results on more real-world datasets are always desirable: we hope the results here will provide the impetus to explore distributed feature construction by ILP on many more real datasets.

Acknowledgements H.D. is also an adjunct assistant professor at the Department of Computer Science at IIIT, Delhi and an Affiliated Member of the Institute of Data Sciences, Columbia University, NY. A.S. also holds visiting positions at the School of CSE, University of New South Wales, Sydney; and at the Department of Computer Science, Oxford University, Oxford.

Appendix

Convergence of the DFE algorithm

The proof of convergence of the algorithm makes use of the following concept-in the distributed setting, the process of information exchange between k sites can 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 (Tsitsiklis et al. 1986). A detailed discussion regarding convergence of the algorithm is presented here.
First, we make the following assumptions about the cost function J.

Assumption 1 • $J(W^t) \geq 0$, for every $W^t \in R^m$.

- **Lipschitz Continuity of** ∇J: The function J is continuously differentiable and there exists a constant K_1 such that

$$\| \nabla J(W^{t_1}) - \nabla J(W^{t_2}) \| \leq K_1 \| W^{t_1} - W^{t_2} \|, \forall W^{t_1}, W^{t_2} \in \mathbb{R}^m. \tag{1}$$

- **Descent Lemma** (Bertsekas and Tsitsiklis 1997) If J satisfies the Lipschitz condition above, then

$$J(W^{t_1} + W^{t_2}) \leq J(W^{t_1}) + (W^{t_2})' \nabla J(W^{t_1}) + \frac{K_1}{2} \| W^{t_2} \|_2^2,$$
$$\text{for all } W^{t_1}, W^{t_2} \in \mathbb{R}^m.$$

The proof of the above Lemma is along the lines of the argument presented in Bertsekas and Tsitsiklis (1997).

Proof Let t be a scalar parameter and let $\mathcal{J}(t) = J(W^{t_1} + t \times W^{t_2})$. The chain rule for derivatives yields, $\frac{d(\mathcal{J})(t)}{dt} = (W^{t_2})' \nabla J(W^{t_1} + t \times W^{t_2})$.

$$\begin{aligned}
&J(W^{t_1} + W^{t_2}) - J(W^{t_1}) = \mathcal{J}(1) - \mathcal{J}(0) \\
&\quad = \int_0^1 \frac{d(\mathcal{J})(t)}{dt} = \int_0^1 (W^{t_2})' \nabla J(W^{t_1} + tW^{t_2}) \\
&\quad \leq \int_0^1 (W^{t_2})' \nabla J(W^{t_1}) dt + \left| \int_0^1 (W^{t_2})' \nabla J(W^{t_1} + tW^{t_2}) - \nabla J(W^{t_1}) \right| \\
&\quad \leq \int_0^1 (W^{t_2})' \nabla J(W^{t_1}) dt + \int_0^1 \| W^{t_2} \| \| \nabla J(W^{t_1} + tW^{t_2}) \\
&\qquad - \nabla J(W^{t_1}) \| dt \\
&\quad \leq (W^{t_2})' \nabla J(W^{t_1}) + \| W^{t_2} \| \int_0^1 K_1 t \| W^{t_2} \| dt \\
&\quad = (W^{t_2})' \nabla J(W^{t_1}) + \frac{1}{2} K_1 \| W^{t_2} \|^2
\end{aligned}$$

□

In our algorithm, the vector W^t is split over sites $S_1, S_2, \cdots, S_k$. The attributes at site S_i, $(1 \leq i \leq k)$ are updated according to the following equation:

$$W_i^{t+1} = W_i^t - \eta_i^t s_i^t \tag{2}$$

where η_i^t is the step size and s_i^t is the descent direction at site S_i. Let T^i be the set of times when processor i makes an update. It is assumed that $s_i^t = 0$ when $t \notin T^i$. For times $t \in T^i$, we assume that the update direction is such that the cost function decreases and s_i^t has the opposite sign from $\nabla J_i(W_i^t)$. The underlying deterministic gossip algorithm is described by:

$$W_i^{t+1} = \sum_{\{i | t \in T^i\}} \alpha_{ii} W_i^t + \sum_{\{j | t \in T^i\}} \alpha_{ij} W_j^t \tag{3}$$

where the coefficients α's are non-negative scalars.

Example 1 Let S_i and S_j be the only two sites communicating with each other. Then Eq. 3 reduces to

$$\begin{aligned}W_i^{t+1} &= \alpha_{ii} W_i^t + \alpha_{ij} W_j^t \\ &= \alpha_{ii}(W_i^{t-1} - \eta_i^{t-1} s_i^{t-1}) + \alpha_{ij}(W_j^{t-1} - \eta_j^{t-1} s_j^{t-1}) \\ &= (\alpha_{ii} W_i^{t-1} + \alpha_{ij} W_j^{t-1}) - (\alpha_{ii}\eta_i^{t-1} s_i^{t-1} + \alpha_{ij}\eta_j^{t-1} s_j^{t-1}) \\ &= \cdots \\ &= (\alpha_{ii} W_i^1 + \alpha_{ij} W_j^1) - (\alpha_{ii}(\eta_i^{t-1} s_i^{t-1} + \eta_i^{t-2} s_i^{t-2} \cdots + \eta_i^1 s_i^1) \\ &\quad + \alpha_{ij}(\eta_j^{t-1} s_j^{t-1} + \eta_j^{t-2} s_j^{t-2} + \cdots + \eta_j^1 s_j^1))\end{aligned}$$

Hence, by induction it can be shown that:

$$W_i^t = \sum_{j=1}^{k} \alpha_{ij} W_j^1 + \sum_{\tau=1}^{t-1} \sum_{j=1}^{k} \alpha_{ij} \eta_j^\tau s_j^\tau \tag{4}$$

It is also assumed that there exist positive constants K_4 and K_5 such that step-sizes η_i^t are bounded as follows: $\frac{K_4}{t} \le \eta_i^t \le \frac{K_5}{t}$. Furthermore, the following assumptions hold true:

Assumption 2 1. $\forall i, j$ and $0 \le \tau \le t, 0 \le \alpha_{ij} \le 1$.
2. For any i, j and $\tau \ge 0$, the limit of α_{ij} as t tends to infinity exists and is the same for all i and is denoted by α_i
3. There exists some $\eta > 0$ such that $\alpha_j \ge \eta$ and $\forall j \in \{1, \cdots, k\}$ and $\tau \ge 0$
4. There exists constants $A > 0$ and $\rho \in (0, 1)$ such that $|\alpha_{ij} - \alpha_j| \le A\rho^{t-\tau}, \forall t > \tau > 0$

Assumption 3 Extensions of the descent Lemma (Bertsekas and Tsitsiklis 1997) at each site:

(a) For every i and t we have,

$$s_i^t \nabla J_i(W_i^t) \le 0. \tag{5}$$

(b) There exist positive constants K_2 and K_3 such that

$$K_2 |\nabla J_i(W_i^t)| \le |s_i^t| \le K_3 |\nabla J_i(W_i^t)| \tag{6}$$

Let $\mathcal{S}(t)$ be the set of random variables defined by: $\mathcal{S}(t) = \{s_i^\tau | i \in \{1, \cdots, k\}, \tau < t\}$. The variables in $\mathcal{S}(t)$ are the only sources of randomness upto the time t at site i. The set $\mathcal{S}(t)$ is also a representation of the entire history of the algorithm upto the moment that the update directions s_i^τ are generated.

Assumption 4 Stochastic descent Lemma (Bertsekas and Tsitsiklis 1997) at each site: There exist positive constants K_6, K_7 and K_8 such that:

(a)

$$\nabla J(W_i^t)' E[s_i^t | \mathcal{S}(t)] \le -K_6 \parallel \nabla J(W_i^t) \parallel^2, \forall t \in T^i. \tag{7}$$

(b)

$$E[\parallel s_i^t \parallel^2 | \mathcal{S}(t)] \le K_7 \parallel \nabla J(W_i^t) \parallel^2 + K_8, \forall t \in T^i. \tag{8}$$

Assumption 3 implies that the expected direction of the update given the past history is in the descent direction. In Assumption 4, the presence of constant K_8 in the inequality allows the algorithm to make non-zero updates even when the minimum has been reached.

Assumption 5 Partial Asynchronism (Bertsekas and Tsitsiklis 1997)
There exists a positive integer B such that:

(a) For every i and for every $t \geq 0$ at least one of the elements of the set $\{t, t+1, \cdots, t+B-1\}$ belongs to T^i.
(b) There holds $\max\{0, t-B+1\} \leq \tau_{ij}^t \leq t$, for all i and j and $t \geq 0$.

Finally, for completeness, we introduce the notions of *martingales* and the martingale convergence theorem(s) which are required for the proofs in the appendix.

A martingale is a model of a fair game where knowledge of past events never helps predict the mean of the future winnings. In general, a martingale is a stochastic process for which, at a particular time in the realized sequence, the expectation of the next value in the sequence is equal to the present observed value even given knowledge of all prior observed values at a current time.[12] A formal definition (using measure theory Tao 2011) is given below:

Let $(\sigma, \mathcal{F}, P)$ be a probability space. A martingale sequence of length n, is a sequence $X_1, X_2, \cdots, X_n$ of random variables and corresponding sub-σ fields $\mathcal{F}_1, \mathcal{F}_2, \cdots, \mathcal{F}_n$ that satisfy the following relations:

- Each X_i is an integrable random variable which is measurable with respect to the corresponding σ-field $\mathcal{F}_i$.
- The sigma fields are increasing $F_i \subset F_{i+1}$ for every i
- For every $i \in [1, 2, \cdots, n-1]$, we have the relation, $X_i = E[X_{i+1}|\mathcal{F}_i]$ almost everywhere P.

Along the same lines,

- A *submartingale* is defined as: for every i, $X_i \leq E[X_{i+1}|\mathcal{F}_i]$ almost everywhere P and
- A *supermartingale* is defined as: for every i, $X_i \geq E[X_{i+1}|\mathcal{F}_i]$ almost everywhere P.

Martingale convergence theorem is a special type of theorem since the convergence follows from the structural properties of the sequence of random variables. The Supermartingale Convergence theorem and a variant used in proofs is presented next.
Supermartingale Convergence Theorem (Bertsekas and Tsitsiklis 1997): Let $\{Y_i\}$ be a sequence of random variables and let $\{\mathcal{F}_i\}$ be a sequence of finite sets of random variables such that $F_i \subset F_{i+1}$ for each i. Suppose that:

- Each Y_i is non-negative
- For each i, we have $E[Y_i] < \infty$
- For each i, we have $E[Y_{i+1}|\mathcal{F}_i] \leq Y_i$ with probability 1.

Then there exists a non-negative random variable Y such that the sequence of $\{Y_i\}$ converges to Y with probability 1.

An extension of the above theorem, can be stated as follows: Let $\{Y_i\}$ and $\{Z_i\}$ be two sequence of random variables. Let $\{\mathcal{F}_i\}$ be a sequence of finite sets of random variables such that $F_i \subset F_{i+1}$ for each i. Suppose that:

12 http://en.wikipedia.org/wiki/Martingale_(probability_theory).

- The random variables Y_i and Z_i are non-negative.
- There holds $E[Y_{i+1}|\mathcal{F}_i] \leq Y_i + Z_i, \forall i$ with probability 1.
- There holds $\sum_{i=1}^{\infty} E[Z_i] < \infty$.

Then there exists a nonnegative random variable Y such that the sequence $\{Y_i\}$ converges to Y with probability 1.

Proposition 1 Convergence of the Distributed Feature Estimation (DFE) Algorithm *Under the Assumptions 1-3, there exists some $\eta^0 > 0$, such that if $0 < \eta_i^t < \eta^0$, then:*

1. $\lim_{t\to\infty} J(W_i^t)$ *exists and is the same for all i with probability 1.*
2. $\lim_{t\to\infty}(W_i^t - W_j^t) = 0$ *with probability 1 and in the mean square sense.*
3. *For every i, $\lim_{t\to\infty} \nabla J(W_i^t) = 0$.*
4. *Suppose that the set $\{W|J(W) \leq C\}$ is bounded for every $C \in \mathcal{R}$; then there exists a unique vector W^* at which J is minimized and this is the unique vector at which ∇J vanishes. Then, W_i^t converges to W^* for each i with probability 1.*

Proof of Proposition 1 Without loss of generality, assume that $\eta_i^t = \frac{1}{t}, \forall i, t$.

We note that the underlying gossip protocol illustrated by Eq. 3 has a simple structure but is not easy to manipulate in algorithms primarily because we have one such equation for each i and they are generally coupled. Thus we need to keep track of vectors $W_1^t, W_2^t, \cdots, W_k^t$ simultaneously. Analysis would be simpler if we could associate one single vector $\mathcal{W}^t$ that summarizes the information contained in W_i^t's. Let $\mathcal{W}^t$ be defined as follows:

$$\mathcal{W}^t = \sum_{i=1}^{k} \alpha_i W_i^1 + \sum_{\tau=1}^{t-1} \sum_{i=1}^{k} \alpha_i \eta_i^\tau s_i^\tau \tag{9}$$

The interpretation of vector $\mathcal{W}^t$ is quite interesting in the following sense – if the sites stopped performing updates at time $\bar{t}$, but keep communicating and forming convex combinations of their states using the gossip protocol, they will asymptotically agree and the vector they agree upon is $\mathcal{W}^t$. Finally, $\mathcal{W}^{t+1} = \mathcal{W}^t + \sum_{i=1}^{k} \alpha_i \eta_i^t s_i^t$.

Define also the following:

$$b^t = \sum_{i=1}^{k} \| s_i^t \|, t \geq 1. \tag{10}$$

$$G^t = -\sum_{i=1}^{k} \nabla J(W_i^t)' \alpha_i s_i^t, \ t \geq 1. \tag{11}$$

Lemma 1.0 *(a) If $t \in T^i$, then*

$$E[G^t|S(t)] \geq K_6 \lambda \sum_{\{i|t\in T^i\}} \| \nabla J(W_i^t)' \|^2 \geq 0, \tag{12}$$

where $\alpha_{ii} > \lambda, \forall i \in \{1, \cdots, k\}$ and $\lambda > 0$.

(b) If $t \geq 1$, then

$$E[(b^t)^2|S(t)] \leq A_1 E[G^t|S(t)] + A_2 \tag{13}$$

where $A_1 = \frac{kK_7}{\lambda K_6}$ and $A_2 = k^2 K_8$.

Proof Using Assumption 2.0b, Eq. 7 and the fact that $s^i = 0, t \notin T^i$, we have:

$$\begin{aligned} E[G^t|S(t)] &= -\sum_{i|t\in T^i} \nabla J(W_i^t)\alpha_i E[s_i^t|S(t)] \\ &\geq \sum_{i|t\in T^i} K_6 \parallel \nabla J(W_i^t) \parallel^2 \alpha_i \\ &\geq \lambda K_6 \sum_{i|t\in T^i} \parallel \nabla J(W_i^t) \parallel^2 \end{aligned}$$

This proves part (a) of the Lemma. Applying Eq. 8 we obtain,

$$\begin{aligned} E[(b^t)^2|S(t)] &= E\left[\left(\sum_{i=1}^{k} \parallel s_i^t \parallel\right)^2 |S(t)\right] \\ &\leq k\sum_{i=1}^{k} E[\parallel s_i^t \parallel^2 |S(t)] \\ &\leq k\sum_{i=1}^{k}(K_7 \parallel \nabla J(W_i^t) \parallel^2 + K_8) \\ &\leq \frac{kK_7}{\lambda K_6} E[G^t|S(t)] + k^2 K_8 \end{aligned}$$

where the last inequality uses the proof in part (a). □

Lemma 2.0 *For every $t \geq 1$, we have*

$$\parallel \mathcal{W}^t - W_i^t \parallel \leq A\sum_{\tau=1}^{t-1} \frac{1}{\tau}\rho^{t-\tau} b^\tau \tag{14}$$

where $A > 0$, $\rho \in (0, 1)$

Proof Subtracting Eq. 9 from Eq. 4 we have

$$\mathcal{W}^t - W_i^t = \sum_{\tau=1}^{t-1}\sum_{j=1}^{k} \frac{1}{\tau}[\alpha_j - \alpha_{ij}]s_j^t \tag{15}$$

Furthermore, using Assumption 1.1(a) and Definition 10 we obtain,

$$\begin{aligned} \parallel \mathcal{W}^t - W_i^t \parallel &\leq \sum_{\tau=1}^{t-1} \frac{1}{\tau} \sum_{j=1}^{k} A\rho^{t-\tau} \parallel s_j^\tau \parallel \\ &\leq A\sum_{\tau=1}^{t-1} \frac{1}{\tau}\rho^{t-\tau} b^\tau \end{aligned}$$

□

Using the fact that $\mathcal{W}^{t+1} = \mathcal{W}^t + \sum_{i=1}^{k} \alpha_i \eta_i^t s_i^t$ and Assumption 1.0 (3) we obtain:

$$\begin{aligned}
J(\mathcal{W}^{t+1}) &= J(\mathcal{W}^t + \sum_{i=1}^{k} \alpha_i \eta_i^t s_i^t) \leq J(\mathcal{W}^t) + \sum_{i=1}^{k} \alpha_i \eta_i^t s_i^t \nabla J(\mathcal{W}^{t+1})' + \frac{K}{2} \parallel \sum_{i=1}^{k} \alpha_i \eta_i^t s_i^t \parallel_2^2 \\
&\leq J(\mathcal{W}^t) + \frac{1}{t} \sum_{i=1}^{k} \alpha_i s_i^t \nabla J(W_i^{t+1})' + \frac{1}{t} \sum_{i=1}^{k} \alpha_i s_i^t (\nabla J(\mathcal{W}^{t+1})' - \nabla J(W_i^{t+1})') \\
&\quad + \frac{K}{2t^2} \parallel s_i^t \parallel_2^2 \\
&\leq J(\mathcal{W}^t) - \frac{1}{t} G(t) + \frac{1}{t} \sum_{i=1}^{k} \alpha_i s_i^t K_1 \parallel \mathcal{W}^{t+1} - W_i^{t+1} \parallel + \frac{K}{2t^2} (b^t)^2 \\
&\leq J(\mathcal{W}^t) - \frac{1}{t} G(t) + \frac{K_1}{t} \sum_{i=1}^{k} \alpha_i s_i^t A \sum_{\tau=1}^{t-1} \frac{1}{\tau} \rho^{t-\tau} b^\tau + \frac{K}{2t^2} (b^t)^2 \\
&\leq J(\mathcal{W}^t) - \frac{1}{t} G(t) + \frac{K_1 A}{t} b^t \sum_{\tau=1}^{t-1} \frac{1}{\tau} \rho^{t-\tau} b^\tau + \frac{K}{2t^2} (b^t)^2 \\
&\leq J(\mathcal{W}^t) - \frac{1}{t} G(t) + K_1 A \sum_{\tau=1}^{t-1} \frac{1}{t\tau} \rho^{t-\tau} b^\tau b^t + \frac{K}{2t^2} (b^t)^2 \\
&\leq J(\mathcal{W}^t) - \frac{1}{t} G(t) + K_1 A \sum_{\tau=1}^{t-1} \rho^{t-\tau} (\frac{(b^\tau)^2}{\tau^2} + \frac{(b^t)^2}{t^2}) + \frac{K}{2t^2} (b^t)^2 \\
&\leq J(\mathcal{W}^t) - \frac{1}{t} G(t) + A_3 \sum_{\tau=1}^{t} \rho^{t-\tau} \frac{(b^\tau)^2}{\tau^2} \qquad (16)
\end{aligned}$$

where $A_3 = \frac{K_1 A}{1-\rho} + \frac{K}{2}$.

Lemma 3.0 *There holds*

$$\sum_{t=1}^{\infty} \frac{1}{t} E[G^t] < \infty \qquad (17)$$

We take expectations of both sides of the above inequality and use Eq. 13 to bound $E[(b^t)^2]$. This yeilds:

$$E[J(\mathcal{W}^{t+1})] \leq E[J(\mathcal{W}^t)] - \frac{1}{t} E[G^t] + A_3 \sum_{\tau=1}^{t} \rho^{t-\tau} \frac{1}{\tau^2} (A_1 E[G^\tau + A_2]). \qquad (18)$$

Let $t = 1, 2, \cdots, \bar{t}$ and add the resulting inequalities from Eq. 18. Then,

$$\begin{aligned}
E[J(\mathcal{W}^{\bar{t}+1})] &\leq J(\mathcal{W}^1) - \sum_{t=1}^{\bar{t}} \frac{1}{t} E[G^t] + A_2 A_3 \sum_{t=1}^{\bar{t}} \sum_{\tau=1}^{t} \rho^{t-\tau} \frac{1}{\tau^2} + A_1 A_3 \sum_{t=1}^{\bar{t}} \sum_{\tau=1}^{t} \rho^{t-\tau} \frac{1}{\tau^2} E[G^\tau] \\
&= J(\mathcal{W}^1) - \sum_{t=1}^{\bar{t}} \frac{1}{t} E[G^t](1 - A_1 A_3 \frac{1}{t} \sum_{t=1}^{\bar{t}} \rho^{t-\tau}) + A_2 A_3 \sum_{t=1}^{\bar{t}} \sum_{\tau=1}^{t} \rho^{t-\tau} \frac{1}{\tau^2} \\
&\leq J(\mathcal{W}^1) - \sum_{t=1}^{\bar{t}} \frac{1}{t} E[G^t](1 - \frac{A_1 A_3}{t(1-\rho)}) + A_2 A_3 \sum_{\tau=1}^{\bar{t}} \frac{1}{\tau^2 (1-\rho)}
\end{aligned}$$

The term $A_2 A_3 \sum_{\tau=1}^{\bar{t}} \frac{1}{\tau^2(1-\rho)}$ *is bounded since the infinite sum* $\sum_{\tau=1}^{\infty} \frac{1}{\tau^2(1-\rho)}$ *is bounded. If* $\sum_{t=1}^{\infty} \frac{1}{t} E[G^t] = +\infty$, *then the right hand side would equal* $-\infty$. *However, the left hand side is non-negative. This proves the lemma.*

Lemma 4.0 *The sequence* $\{J(\mathcal{W}^t\}$ *converges with probability 1.*

Proof Taking conditional expectation of inequality (16), conditioned on $\mathcal{S}(t)$ and using Lemma 1.0 we have,

$$E[J(\mathcal{W}^{t+1})|\mathcal{S}(t)] \leq J(\mathcal{W}^t) + A_3 \sum_{\tau=1}^{t} \rho^{t-\tau} \frac{1}{\tau^2} E[(b^\tau)^2|\mathcal{S}(t)] \tag{19}$$

Let $Z(t) = \sum_{\tau=1}^{t} \rho^{t-\tau} \frac{1}{\tau^2} E[b^{\tau 2}|\mathcal{S}(t)]$. Using Lemmas 1.0(b) and 3.0 we have:

$$\begin{aligned}\sum_{t=1}^{\infty} E[Z(t)] &= \frac{1}{1-\rho} \sum_{t=1} \infty \frac{1}{t^2} E[(b^\tau)^2] \\ &\leq \frac{1}{1-\rho} \sum_{t=1} \infty \frac{1}{t^2} (A_1 E[G^t + A_2) \\ &< \infty. \end{aligned} \tag{20}$$

Using inequalities (19) and (20), a variant of the Supermartingale theorem applies and hence $\{J(\mathcal{W}^t\}$ converges with probability 1. □

References

Agarwal, A., Chapelle, O., Dudík, M., & Langford, J. (2014). A reliable effective terascale linear learning system. *Journal of Machine Learning Research*, *15*, 1111–1133.
Agrawal, R. & Srikant, R. (1995). Mining sequential patterns. In *ICDE*.
Antunes, C. & Oliveira, A. L. (2003). Generalization of pattern-growth methods for sequential pattern mining with gap constraints. In *MLDM*.
Aseervatham, S., Osmani, A., & Viennet, E. (2006). bitSPADE: A lattice-based sequential pattern mining algorithm using bitmap representation. In *ICDM*.
Ayres, J., Gehrke, J., Yiu, T., & Flannick, J. (2002). Sequential pattern mining using a bitmap representation. In *KDD*.
Benezit, F., Dimakis, A. G., Thiran, P., & Vetterli, M. (2010). Order-optimal consensus through randomized path averaging. *IEEE Transactions on Information Theory*, *56*(10), 5150–5167.
Bertsekas, D. P., & Tsitsiklis, J. N. (1997). *Parallel and distributed computation: numerical methods*.
Blum, A. (1992). Learning boolean functions in an infinite attribute space. *Machine Learning*, *9*(4), 373–386.
Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of the 19th international conference on computational statistics (COMPSTAT'2010)*, pp. 177–187.
Bottou, L., & Bousquet, O. (2011). The tradeoffs of large scale learning. In *Optimization for machine learning* (pp. 351–368).
Bottou, L., & Bousquet, O. (2011). The tradeoffs of large scale learning. In *Optimization for machine learning* (pp. 351–368). MIT Press.
Boyd, S., Ghosh, A., Prabhakar, B., & Shah, D. (2006). Randomized gossip algorithms. *IEEE/ACM Transaction Network*, *14*(SI), 2508–2530.
Boyd, S., Parikh, N., Chu, E., Peleato, B., & Eckstein, J. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, *3*(1), 1–122.
Carlson, A., Cumby, C., Rosen, J., & Roth, D. (1999). The snow learning architecture. Technical Report UIUCDCS-R-99-2101, UIUC Computer Science Department, 5 .
Chalamalla, A., Negi, S., Venkata Subramaniam, L., & Ramakrishnan, G. (2008). Identification of class specific discourse patterns. In *CIKM*, pp. 1193–1202.

Christoudias, C. M., Urtasun, R., & Darrell, T. (2008). Unsupervised distributed feature selection for multi-view object recognition. Technical Report MIT-CSAIL-TR-2008-009, MIT.

Cybenko, G. (1989). Dynamic load balancing for distributed memory multiprocessors. *Proceedings of the Journal of Parallel and Distributed Computing*, *7*, 279–301.

Darken, C., & Moody, J. (1990). Note on learning rate schedules for stochastic optimization. In *Proceedings of the conference on advances in neural information processing systems*, pp. 832–838.

Das, K., Bhaduri, K., & Kargupta, H. (2010). A local asynchronous distributed privacy preserving feature selection algorithm for large peer-to-peer networks. *Knowledge and Information Systems*, *24*(3), 341–367.

Davis, J., Burnside, E., de Castro Dutra, I., Page D., & Costa, V. S. (2005a). An integrated approach to learning Bayesian networks of rules. In *Machine Learning: ECML 2005*, pp. 84–95.

Davis, J., Burnside, E. S., de Castro Dutra, I., Page, D., Ramakrishnan, R., Costa, V. S., & Shavlik, J. W. (2005b). View learning for statistical relational learning: With an application to mammography. In *Proceedings of the nineteenth international joint conference on artificial intelligence*, pp. 677–683.

Davis, J., Ong, I., Struyf, J., Burnside, E., Page, D., & Costa, V. S. (2007). Change of representation for statistical relational learning. In *Proceedings of the 20th international joint conference on artificial intelligence*, pp. 2719–2726.

Dehaspe, L., & De Raedt, L. (1995). Parallel inductive logic programming. Machine learning and knowledge discovery in databases. In *Proceedings of the MLnet familiarization workshop on statistics* (pp. 112–117).

Dekel, O., Gilad-Bachrach, R., Shamir, O., & Xiao, L. (2012). Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, *13*(1), 165–202.

Dimakis, A. G., Sarwate, A. D., & Wainwright, M. J. (2006). Geographic gossip: Efficient aggregation for sensor networks. In *The fifth international conference on information processing in sensor networks*, pp. 69–76.

Duchi, J., Agarwal, A., & Wainwright, M. (2012). Dual averaging for distributed optimization: Convergence analysis and network scaling. *IEEE Transactions on Automatic Control*, *57*(3), 592–606.

Džeroski, S. (1993). Handling imperfect data in inductive logic programming. In *Proceedings of the Fourth Scandinavian Conference on Artificial Intelligence*, pp. 111–125.

Fischer, J. M., Lynch, N. A., & Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, *32*(2), 374–382.

Fonseca, N. A., Silva, F., & Camacho, R. (2005). Strategies to parallelize ILP systems. In *Proceedings of the 15th international conference on inductive logic programming*, pp. 136–153.

Garcia, D. J., Hall, L. O, Goldgof, D. B. & Kramer K. (2006). A parallel feature selection algorithm from random subsets. In *Proceedings of the international workshop on parallel data mining*.

Garofalakis, M. N., Rastogi, R., & Shim, K. (1999). Spirit: Sequential pattern mining with regular expression constraints. In *VLDB*.

Han, Y., & Wang, J. (2009). An l1 regularization framework for optimal rule combination. In *ECML/PKDD*.

Jawanpuria, P., Nath, J. S., & Ramakrishnan, G. (2011). Efficient rule ensemble learning using hierarchical kernels. In *ICML*, pp. 161–168.

Jelasity, M., Guerraoui, R., Kermarrec, A., & Steen, M. (2004). The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In *Middleware 2004*, Vol. 3231, pp. 79–98.

Jelasity, M., Montresor, A., & Babaoglu, Ö. (2005). Gossip-based aggregation in large dynamic networks. *ACM Transaction on Computational Systems*, *23*(3), 219–252.

Ji, X., Bailey, J., & Dong, G. (2006). Mining minimal distinguishing subsequence patterns with gap constraints. *Knowledge and Information Systems*.

John, G. H., Kohavi, R., & Pfleger, K. (1994). Irrelevant features and the subset selection problem. In *Proceedings of the eleventh international conference on machine learning*, pp. 121–129 .

Joshi, S., Ramakrishnan, G., & Srinivasan, A. (2008). Feature construction using theory-guided sampling and randomised search. In *ILP*, pp 140–157.

Kempe, D., Dobra, A., & Gehrke, J. (2003). Gossip-based computation of aggregate information. In *Proceedings of 44th annual IEEE symposium on foundations of computer science*, pp. 482–491.

King, R. D., & Srinivasan, A. (1996). Prediction of rodent carcinogenicity bioassays from molecular structure using inductive logic programming. *Environmental Health Perspectives*, *104*, 1031–1040.

King, R. D., Muggleton, S. H., Srinivasan, A., & Sternberg, M. J. (1996). Structure-activity relationships derived by machine learning: The use of atoms and their bond connectivities to predict mutagenicity by inductive logic programming. *Proceedings of the National academy of Sciences of the United States of America*, *93*(1), 438–42.

Kudo, T., Maeda, E., & Matsumoto, Y. (2004). An application of boosting to graph classification. In *NIPS*.

Landwehr, N., Kersting, K., & Raedt, L. D. (2007). Integrating naive bayes and foil. *Journal of Machine Learning Research*, *8*, 481–507.

Langford, J., Smola, A., & Zinkevich, M. (2009). Slow learners are fast. In *Advances in neural information processing systems*, pp. 2331–2339.

Larson, J., & Michalski, R. S. (1977). Inductive inference of VL decision rules. *SIGART Bulletin*, *63*, 38–44.

Lavrac, N., & Dzeroski, S. (1993). *Inductive logic programming: Techniques and applications* (p. 10001). New York, NY: Routledge.

Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, *2*(4), 285–318.

Liu, H., & Motoda, H. (1998). *Feature selection for knowledge discovery and data mining*. Boston: Kluwer Academic Publishers.

Lopez, F. G., Torres, M. G. A., Batista, B. M., Perez, J. A. M., & Moreno-Vega, J. M. (2006). Solving feature subset selection problem by a parallel scatter search. *European Journal of Operational Research*, *169*(2), 477–489.

Mangasarian, L. (1995). Parallel gradient distribution in unconstrained optimization. *SIAM Journal on Control and Optimization*, *33*(6), 1916–1925.

Michie, D., Bain, M., & Hayes-Michie, J. (1990). Cognitive models from subcognitive skills. In M.J. Grimble J. McGhee and P. Mowforth, (Eds.), *Knowledge-based systems for industrial control* (pp. 71–99). Peter Peregrinus for IEE, London.

Montresor, A., & Jelasity, M., PeerSim. (2009). A scalable P2P simulator. In *Proceedings of the 9th international conference on Peer-to-Peer (P2P'09)*, pp. 99–100 .

Muggleton, S. (1994). Inductive logic programming: Derivations, successes and shortcomings. *SIGART Bulletin*, *5*(1), 5–11.

Muggleton, S. (1995). Inverse entailment and progol. *New Generation Computing*, *13*(3), 245–286.

Muggleton, S. H., Santos, J. C. A., & Tamaddoni-Nezhad, A. (2008). TopLog: ILP using a logic program declarative bias. *Logic Programming*, *5366*, 687–692.

Nagesh, A., Ramakrishnan, G., Chiticariu, L., Krishnamurthy, R., Dharkar, A., & Bhattacharyya, P. (2012). Towards efficient named-entity rule induction for customizability. In *EMNLP-CoNLL*, pp. 128–138.

Nair, N., Saha, A., Ramakrishnan, G., & Krishnaswamy, S. (2012). Rule ensemble learning using hierarchical kernels in structured output spaces. In *AAAI*.

Nienhuys-Cheng, S., & De Wolf, R. (1997) *Foundations of inductive logic programming*. New York: Springer.

Niu, F., Recht, B., Ré, C., & Wright, S. J. (2011). Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. *Advances in Neural Information Processing Systems*, *24*, 693–701.

Nowozin, S., Bakir, G., & Tsuda, K. (2007). Discriminative subsequence mining for action classification. In *CVPR*.

Pei, J. (2004). Mining sequential patterns by pattern-growth: The PrefixSpan approach. *Journal of Machine Learning Research*, 16-11.

Pei, J., Han, J., & Wang, W. (2005). Constraint-based sequential pattern mining: the pattern-growth methods. *Journal of Intelligent Information Systems*.

Pei, J., Han, J., & Yan, X. (2004). From sequential pattern mining to structured pattern mining: A pattern-growth approach. *Journal of Computer Science and Technology*, *9*(3), 257–279.

Plotkin, G.D. (1971). *Automatic methods of inductive inference*. PhD thesis, Edinburgh University.

Ramakrishnan, G., Joshi, S., Balakrishnan, S., & Srinivasan, A. (2007). Using ILP to construct features for information extraction from semi-structured text. In *ILP*, pp. 211–224.

Ratnaparkhi, A. (1999). Learning to parse natural language with maximum entropy models. *Machine Learning*, *34*(1), 151–175.

Roth, D. (1998). Learning to resolve natural language ambiguities: A unified approach. In *Proceedings of the innovative applications of artificial intelligence*, pp. 806–813.

Rückert, U. & Kramer, S. (2003). Stochastic local search in k-term dnf learning. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pp. 648–655.

Rückert, U., Kramer, S., & De Raedt, L. (2002). Phase transitions and stochastic local search in k-term dnf learning. In *Proceedings of the 13th European conference on machine learning*, pp. 405–417.

Ryan, M., Hall, K., & Mann, G. (2010). Distributed training strategies for the structured perceptron. In *The annual conference of the north American chapter of the association for computational linguistics*, pp. 456–464.

Saha, A., Srinivasan, A., & Ramakrishnan, G. (2012). What kinds of relational features are useful for statistical learning? In *ILP*.

Sanov, I. N. (1957). On the probability of large deviations of random variables. *Mat. Sbornik*, *42*, 11–44.

Shah, D. (2009). Gossip algorithms. *Foundations and Trends Netwroking*, *3*(1), 1–125.

Singh, S., Kubica, J., Larsen, S., & Sorokina, D., (2009). Parallel large scale feature selection for logistic regression. In *SDM*, pp. 1172–1183.

Specia, L., Srinivasan, A., Ramakrishnan, G., & Graças Volpe Nunes, M. (2006). Word sense disambiguation using inductive logic programming. In *ILP*, pp. 409–423.

Specia, L., Srinivasan, A., Joshi, S., Ramakrishnan, G., & Gracas, M. (2009). An investigation into feature construction to assist word sense disambiguation. *Machine Learning*, *76*(1), 109–136.

Srinivasan, A. & Bain, M. (2014). An empirical study of on-line models for relational data streams. Technical Report 201401, School of Computer Science and Engineering, UNSW.

Srinivasan, A. (1999). The aleph manual.

Srinivasan, A., & King, R. D. (1996). Feature construction with inductive logic programming: a study of quantitative predictions of biological activity aided by structural attributes. In *Proceedings of the sixth inductive logic programming workshop*, Vol. 1314, pp. 89–104.

Srinivasan, A., & Ramakrishnan, G. (2011). Parameter screening and optimisation for ILP using designed experiments. *Journal of Machine Learning Research*, *12*, 627–662.

Srinivasan, A., Muggleton, S. H., Sternberg, M. J. E., & King, R. D. (1996). Theories for mutagenicity: A study in first-order and feature-based induction. *Artificial Intelligence*, *85*(1–2), 277–299.

Sun, Z. (2014). Parallel feature selection based on mapreduce. In *Computer engineering and networking*, volume 277 of *Lecture Notes in Electrical Engineering*, pp. 299–306 .

Sutton, R. (1992) Adapting bias by gradient descent: An incremental version of delta-bar-delta. In *Proceeding of tenth national conference on artificial intelligence*, pp. 171–176.

Tao, T. (2011). *An introduction to measure theory*.

Tsitsiklis, J. N. (1984). *Problems in decentralized decision making and computation*. PhD thesis, Department of EECS, MIT.

Tsitsiklis, J. N., Bertsekas, D. P., & Athans, M. (1986). Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE Transactions on Automatic Control*, *31*.

Varga, R. S. (1962). *Matrix iterative analysis*.

Zelezny, F., Srinivasan, A., & Page, C. D, Jr. (2006). Randomised restarted search in ilp. *Machine Learning*, *64*(1–3), 183–208.

Zhao, Z., Cox, J., Duling, D., & Sarle, W. (2012). Massively parallel feature selection: An approach based on variance preservation. *ECML/PKDD*, *7523*, 237–252.

Zhou, Y., Porwal, U., Zhang, C., Ngo, H. Q., Nguyen, L., Ré, C., & Govindaraju, V. (2014). Parallel feature selection inspired by group testing. In *Annual conference on neural information processing systems*, pp. 3554–3562.

Zinkevich, M., Weimer, M., Smola, A. J., & Li, L. (2010). Parallelized stochastic gradient descent. In *NIPS*, Vol. 4, p. 4.