Mach Learn (2017) 106:243–276
DOI 10.1007/s10994-016-5596-2

An empirical study of on-line models for relational data streams

Ashwin Srinivasan[1] · **Michael Bain**[2]

Received: 11 January 2014 / Accepted: 18 October 2016 / Published online: 3 December 2016

Abstract To date, Inductive Logic Programming (ILP) systems have largely assumed that all data needed for learning have been provided at the onset of model construction. Increasingly, for application areas like telecommunications, astronomy, text processing, financial markets and biology, machine-generated data are being generated continuously and on a vast scale. We see at least four kinds of problems that this presents for ILP: (1) it may not be possible to store all of the data, even in secondary memory; (2) even if it were possible to store the data, it may be impractical to construct an acceptable model using partitioning techniques that repeatedly perform expensive coverage or subsumption-tests on the data; (3) models constructed at some point may become less effective, or even invalid, as more data become available (exemplified by the "drift" problem when identifying concepts); and (4) the representation of the data instances may need to change as more data become available (a kind of "language drift" problem). In this paper, we investigate the adoption of a stream-based on-line learning approach to relational data. Specifically, we examine the representation of relational data in both an infinite-attribute setting, and in the usual fixed-attribute setting, and develop implementations that use ILP engines in combination with on-line model-constructors. The behaviour of each program is investigated using a set of controlled experiments, and performance in practical settings is demonstrated by constructing complete theories for some of the largest biochemical datasets examined by ILP systems to date, including one with a million examples; to the best of our knowledge, the first time this has been empirically demonstrated with ILP on a real-world data set.

Keywords Inductive Logic Programming · Data streams · Online learning

Editors: Gerson Zaverucha and Vítor Santos Costa.

✉ Ashwin Srinivasan
ashwin.srinivasan@wolfson.oxon.org; ashwin@goa.bits-pilani.ac.in

[1] Department of Computer Science and Information Systems, BITS Pilani, K.K. Birla Goa Campus, Goa, India

[2] School of Computer Science and Engineering, UNSW, Sydney 2052, Australia

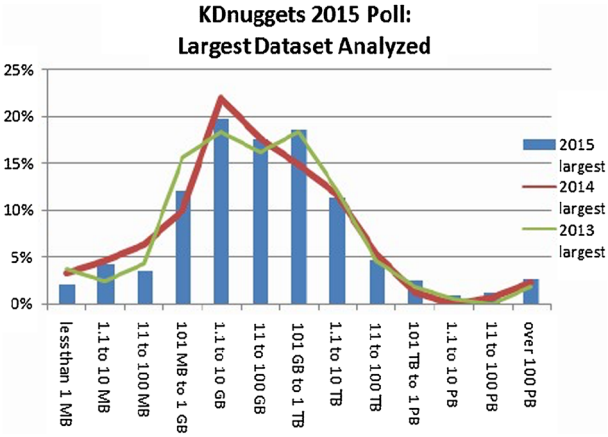


Fig. 1 Results of a poll on the largest sized datasets analysed (from KDnuggets, August 2015—used with permission)

1 Introduction

Can an Inductive Logic Programming (ILP) system process millions of data instances effectively and efficiently? It is not unreasonable to ask in return: why is this an important question? A recent survey of current data analysis practice[1] suggests that the largest datasets routinely analysed 2013–2015 were normally up to around a terabyte or so, with some much larger in size (see Fig. 1). It is not clear what form the analysis took, but in ILP terms, this is substantially larger than the biggest problems considered so far.[2]

The point of course, is not that ILP engines should be able tackle very large datasets because other people are doing it, nor is it the case that all significant analysis problems are ones with large amounts of data ((Hand et al. 1994) for example, has a collection of real-life problems characterised by small amounts of data). Instead, the main issues are these: (1) very large datasets, once the province only of large internet companies or scientific areas such as experimental physics, are now created or captured routinely in almost every field of scientific, industrial and social endeavour, and, importantly, many of the more interesting questions now being asked of this data are relational; (2) as the size and diversity of data generated increases, human comprehension is becoming a bottleneck (Muggleton and Michie 1997). Machine-learning methods that can extract information and convey them to us in a manner to provoke insight are expected to play an important role over the next decade (Kelly and Hamm 2013). Since its inception, the construction of human-comprehensible theories from relational data has been a primary motivation for ILP. By employing a highly expressive subset of first-order logic, and by making explicit provisions for incorporating domain-specific knowledge ILP systems should be a natural choice for addressing the second problem, providing they are capable of handling the first problem (that is, data volume).

We envisage at least four kinds of problems that very large datasets present for ILP:

(1) It may not be possible to store all of the data, even in secondary memory;

[1] http://www.kdnuggets.com/2015/08/largest-dataset-analyzed-more-gigabytes-petabytes.html.

[2] It seems that a synthetic "Michalski trains" data set of 1.25 million examples was the largest ever used by an ILP system (VFILP) (Cardoso and Zaverucha 2006).

(2) Even if it were possible to store the data, it may be impractical to contruct an acceptable model using partitioning techniques that repeatedly perform expensive coverage or subsumption-tests on the data;
(3) Models constructed at some point may become less effective, or even invalid, as more data become available (an example of this is the "drift" problem when identifying concepts); and
(4) The representation of the data instances may need to change as more data become available (a kind of "language drift" problem).

In this paper, we focus on discrimination tasks, which is perhaps the most common type of problem addressed by ILP, and seek to engineer a general-purpose ILP-based system that can meet the following requirements: (R1) It should be able to construct (good) discriminatory models for the data in an efficient manner—informally speaking, by efficient we mean approximately linear in the number of data instances seen; (R2) It should be able to handle indefinite amounts of relational data; and (R3) It should be able to track distributional changes in the data, resulting in changes in the model and in the representation of the data.

We design the system using as building blocks:

– A simple transducer-like extension to an ILP system that defines the processing of examples arriving in a stream;
– The feature-construction ("propositionalisation") abilities of a general-purpose ILP system to construct new relational features of examples on-demand (this is different to the usual approach); and
– Well-established on-line model construction algorithms that have been shown to have an exceptionally robust capacity to construct models even in potentially infinite dimensional feature-spaces and when concepts can drift.

We provide empirical evidence, using a range of controlled datasets, that an ILP system can be devised to satisfy the requirements R1–R3. In addition, the program's performance in practical settings is demonstrated by constructing complete theories for some of the largest biochemical datasets examined by ILP systems to date. Although ILP methods of learning from streams have been developed before (Cardoso and Zaverucha 2006; Lopes and Zaverucha 2009; Dries and De Raedt 2010), to the best of our knowledge the approach in this paper is the first in ILP to have been shown to satisfy requirements R1–R3.

The rest of the paper is organised as follows. Section 2 outlines the framework for stream-based on-line learning that is used in the paper. In Sect. 3 we introduce our approach to relational data stream analysis. Section 4 describes experiments with synthetic data sets, and some large real-world datasets. Section 5 discusses previous and related work. Section 6 concludes the paper.

2 On-line learning with streaming data

Data streams—dating back at least to the 1960s (Landin 1965)—are useful as a (data) model for dealing both with data that is generated continuously, and with large datasets that have to be stored in secondary memory. The difficulties with streaming data analysis arise from the fact that an analysis algorithm neither has the ability to select the ordering of data instances in the stream, nor is it feasible to store them (beyond perhaps a small quantity in an internal buffer). Desirable features (Bifet et al. 2010) of a suitable analysis algorithm are: (a) It processes each instance only once (or perhaps a few times), and must update its model incrementally as each example is processed; (b) Its time complexity must be near-linear in the number of data

instances (that is, it cannot take more than a small constant amount of time processing each instance); (c) It must be memory-efficient; and (d) It must be capable of "anytime" prediction (that is, a model can be used to predict at any point in the data sequence). Substantial efforts have been invested into both supervised and unsupervised model-construction that adhere to these requirements. We refer the reader to Gama (2010) for a very readable text on the area; and to Aggarwal (2007) for some key work.

It is both natural and efficient to adopt an on-line approach to the analysis of a data stream. In this setting a model is updated as elements of the data stream are received. The general form of a *mistake-driven* on-line stream learning algorithm is shown in Algorithm 1.

1 **Algorithm** `online_learner(S)`
  Input: A stream S of example data.
  Output: A model M.
2 Let the initial mistakes $m_0 = 0$, and the initial model be M_0
3 **for** $t = 1, 2, \ldots$ **do**
4 Get new example $e_t = \langle \mathbf{x_t}, y_t \rangle$ from S
5 Predict $y'_t = predict(M_{t-1}, \mathbf{x_t})$
6 **if** *prediction is incorrect (that is,* $y'_t \neq y_t$*)* **then**
7 $m_t = m_{t-1} + 1$
8 $M_t = update(M_{t-1}, e_t)$
9 **else**
10 $m_t = m_{t-1}$
11 $M_t = M_{t-1}$
12 **return** M_t

Algorithm 1: Mistake-driven On-line Learner (based on Carvalho and Cohen 2006).

Note the following about this algorithm: (1) examples are observed as a sequence of data items; (2) at any instant t, there is a model M_t that has made m_t mistakes over t examples; and (3) two auxiliary functions $predict$ and $update$ are used to determine the prediction of example e_t and the (new) model M_t. This mistake-driven approach can be modified to perform the model-updates irrespective of the prediction, or to perform the model-prediction only using a certain category of "focus" examples. In such cases, it may be useful to provide the $update$ function with the prediction y'_t.

Traditionally, the construction of models on data streams (for example, Domingos and Hulten 2000), assumes data instances to be vectors from a pre-defined, finite feature-space. However, there is at least one on-line feature-based learner that does not require a fixed set of features, and has proved to be remarkably robust to both very large numbers of irrelevant features and drifting distributions. Winnow (Littlestone 1988) learns a linear threshold discriminator, similar to a Perceptron, although with some key differences that make it more appropriate for ILP. Algorithm 2 shows the main details.

Comparison with Algorithm 1 shows that the definition of the *predict* function in Algorithm 1 is at line 6 of Algorithm 2. The *update* function is from lines 9 to 12. At any instant t the algorithm represents each data instance using a Boolean vector $\mathbf{x_t}$. Surprisingly, following the work of Blum (1992) on an *infinite-attribute* framework for Winnow-like algorithms, there is no requirement that each data instance use the same set of features. Thus the $\mathbf{x_t}$ could all be of different dimensionality (care has to be taken of course to ensure that the naming of features remains consistent).[3] Associated with each $\mathbf{x_t}$ is a class value y_t. In this paper we will assume $y_t \in \{0, 1\}$.

[3] This is not the same as using a sparse-vector representation (as in e.g., Langford et al. 2009, which enables learning from high-dimensional data). In that setting, the complete set of features is known beforehand, but each data instance is represented by only those features that have some non-default value. Usually, the default

Algorithm `winnow(`S`)`
Input: A stream S of example data.
Output: A weight vector W.
Let the initial mistakes $m_0 = 0$
Given N Boolean features, initialise an N-dimensional weight vector w_0 such that all its components $w_{1,0} = w_{2,0} = \cdots = w_{N,0} = 1$
for $t = 1, 2, \ldots$ **do**
  Get new example $e_t = \langle \mathbf{x_t}, y_t \rangle$ from S
  $y'_t = 1$ if $\mathbf{w_{t-1}} \cdot \mathbf{x_t} \geq n$ otherwise $y'_t = 0$.
  if $y'_t \neq y_t$ **then**
    $m_t = m_{t-1} + 1$
    if $y_t = 1$ **then**
      $\forall i$ s.t. $x_{i,t} = 1$ $w_{i,t} = 2w_{i,t}$, and $\forall i$ s.t. $x_{i,t} = 0$ $w_{i,t} = w_{i,t-1}$
    else
      $\forall i$ s.t. $x_{i,t} = 1$ $w_{i,t} = w_{i,t}/2$, and $\forall i$ s.t. $x_{i,t} = 0$ $w_{i,t} = w_{i,t-1}$
  else
    $m_t = m_{t-1}$, and $\mathbf{w_t} = \mathbf{w_{t-1}}$
return w_t

Algorithm 2: The Winnow algorithm, a mistake-driven on-line learner (Littlestone 1988).

Informally, in the infinite-attribute learning framework (Blum 1992) we assume that the target concept can be defined using a finite set of features, and that each instance is represented by a finite set of features, of which some subset are *relevant* to the definition of the target concept and the remainder are *irrelevant*. Since in ILP the set of possible features may be infinitely large, such a framework is clearly potentially a suitable choice for relational feature construction, particularly when features can be inductively constructed, i.e., *learned* from data, given the appropriate background knowledge.

Winnow adopts a multiplicative weight-update scheme for online learning. Algorithm 2 shows the variant known as "Winnow 2". The original formulation (Winnow 1) simply set weights to 0 instead of halving them when a mistake was made for $y_t = 0$. More general versions multiply by some factor α (the promotion rate) and divide by some factor β (the demotion rate). It has been found useful to allow features not to have zero weight in the infinite-attribute framework (this makes it possible, for example, to recover from changes in the concept).

This simple algorithm has been shown to satisfy all the desirable requirements listed above, including the ability to recover from changes in concepts (Blum 1997) (apart from the ability to handle indefinite amounts of relational data, which is the focus of this paper). Importantly, it also has some well-understood theoretical properties. For example, it is known that if that target concept is a disjunction of k features, then Winnow will make $O(k \log n)$ mistakes, where n is the number of features. It can also be shown that if the number of mistakes is bounded by some M, then Winnow will terminate within $\frac{M}{\epsilon} \ln\left(\frac{M}{\delta}\right)$ examples for some small ϵ, δ such that the probability that the resulting model has error greater than ϵ is at most δ. That is, a concept class learnable within mistake-bounded learning is PAC-learnable (Littlestone 1988).

Footnote 3 continued
value is 0, and an instance is represented by those features with non-zero values. During computation, all other features for the instance are taken to have the value 0. Dynamically adding features, e.g., as in text classification (Katakis et al. 2006), is handled by the infinite-attribute setting.

3 Relational data stream analysis

In this paper, we will be focusing on discriminatory tasks in a relational setting. That is, data instances are tuples either belonging to a pre-specified relation R (used here to denote a set of tuples), or not belonging to it. We are also given the predicate definitions of a set of "background" relations, and the predictive ILP task is to construct a model—used here in the data-analytic sense—for the data instances using the background relations. Ideally, this means that we would want to identify a function f, defined in terms of the background relations, that correctly returns "yes" for every tuple that belongs to R and "no" otherwise. In practice, the ILP engine will attempt to find a function that serves as a (good) approximation) to f.

We use the term *relational data stream* in the sense introduced by Dries and De Raedt (2010). That is, the data stream consists of relational tuples, and we wish to construct models in the sense just described. The straightforward way to adopt the methods described in the previous section to a relational data stream is to have an ILP engine examine a sample of the data—either on-line or off-line—and construct a fixed set of features using techniques developed under the umbrella of "propositionalisation" (see below). From this point on, standard on-line techniques for stream processing can be invoked. We denote this as *fixed-attribute* model construction in this paper, and use it as the baseline for comparison. Its limitations are clear enough: the sample chosen will have to be sufficiently representative of the data; and there is no opportunity to alter the feature-set were the data distribution to change.

In this section, we first describe a procedure for feature-construction by an ILP system, and then use this with on-line learning to construct models for relational data streams in both fixed- and infinite-attribute settings.

3.1 Relational feature construction

ILP-based feature construction has been well-studied; see, e.g., Kramer et al. (2001). Briefly, a feature f—more correctly, a feature function—is a Boolean function of a relational instance x, defined relative to background knowledge B. This usually includes syntactic constraints in the form of a language specification $\mathcal{L}$ and may also include semantic constraints I. The basic relational feature construction approach used by the Aleph ILP engine (Srinivasan 1999) is shown in Algorithm 3.

In this paper, we will be concerned with discriminatory tasks (classification problems). For these, we assume the ILP system is provided with examples of the form $Class(a, b)$ where a is a relational data instance, and b is some class value.

For simplicity, we will assume data instances are drawn from some set $\mathcal{X}$ and the classes are drawn from a set $\mathcal{Y}$. Then, in Step 10 of Algorithm 3, each of the h_j are of the form $Class(x, c) \leftarrow Cp_j(x)$, where x is a variable denoting data instances and c is some class value from the set of classes $\mathcal{Y}$.[4] Here, we adopt terminology from Ratnaparkhi (1996) 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

[4] We note that in general x is not restricted to a single object and can consist of arbitrary tuples of objects. For example, $Class(\langle m_1, m_2\rangle, true)$ might denote that the toxicity of molecule m_1 was higher than that of molecule m_2. Thus, clauses in Step 10 of Algorithm 3 would be $h_j : Class(\langle x_1, x_2\rangle, c) \leftarrow Cp_j(\langle x_1, x_2\rangle)$.

1 **Algorithm** `fconstruct`$(B, I, \mathcal{L}, E)$
  Input: Background knowledge B, semantic constraints I, language constraints $\mathcal{L}$ and examples E
  Output: A set of features F.
2 $i = 0$
3 $N_i = 0$
4 **while** $N_i \leq F_{max}$ **do**
5 increment i
6 Randomly choose a non-redundant relational instance $e \in E$
7 Let $\bot_{\mathcal{L}}(B, e)$ be the most specific clause in $\mathcal{L}$ that logically entails e, given background knowledge B
8 Let H_i be a set of "good" clauses in the subsumption lattice bounded by the empty clause $\top$ and $\bot_{\mathcal{L}}(B, e)$ s.t. every element in H_i is consistent with I and $|H_i| \leq (F_{max} - N_{i-1})$
9 **for** $h_j \in H_i$ **do**
10 Convert h_j to a feature f_j
11 $F_i = \bigcup\{f_j\}$
12 $N_i = N_{i-1} + |F_i|$
13 $F = \bigcup F_i$
14 **return** F

Algorithm 3: An algorithm to construct features from relational examples.

two literals. In Step 10 of Algorithm 3 clause $h_j : Class(x, c) \leftarrow Cp_j(x)$, is converted to a feature f_j using a one-to-one mapping as follows: $f_j(x) = 1$ iff $Cp_j(x) = 1$ (and 0 otherwise). There are some additional points about this procedure that require clarification:

- Programs like Aleph also attempt to ensure that the features constructed are relevant. As described in Specia et al. (2009), the notion of relevance of a feature in a set $F = \{f_1, \ldots, f_k\}$ is a probabilistic one. In a feature-based representation, data instances are elements of the set $F_1 \times \cdots \times F_k$ (where F_i is the set of values of the i^{th} feature: here all the F_i are $\{0, 1\}$). With some abuse of notation, let us take examples to be elements of $F_1 \times \cdots \times F_k \times \mathcal{Y}$. Then, following John et al. (1994), we can distinguish between the *strong relevance* of a feature $f \in F$, if $Pr(Y|F) \neq Pr(Y|F - \{f\})$; and its *weak relevance* if there is some $F' \subset F$ such that $f \in F'$ and f is strongly relevant in F'. That is, $Pr(Y|F') \neq Pr(Y|F' - \{f\})$. The feature-constructor within Aleph attempts to identify features that are at least weakly relevant. We do not describe the mechanisms Aleph employs for this here.
- "Good" clauses are those within the ILP system's hypothesis language that satisfy the given constraints on features. For example, suppose in some biochemical domain we have that $f(m) = TRUE$ for any molecule m if the molecule has 3 fused benzene rings; otherwise $f(m) = FALSE$. The task of the ILP engine is to find definite clauses encoding such a predicate, given definitions in background knowledge of general cyclic structures (like benzene rings), functional groups (like methyls, alcohols, *etc.*), and so on. Constraints can include: performance settings, such as thresholds for precision and recall; syntactic constraints, such as the number and types of literals allowed in the bodies of definite clauses; and semantic constraints encoding domain knowledge. Details of constraints used in experiments in the paper are in Sect. 4.2.
- Features correspond to definite clauses in a hypothesis space bounded above by $\top$ and below by $\bot_{\mathcal{L}}(B, x)$, and the feature construction uses the "learning-from-entailment" setting. For classification problems of the kind studied here, examples often consist of class labels of single objects (like individual molecules). In this case, it is sufficient to consider feature-construction in the more restricted "learning-from-interpretations"

setting. However, this is not a requirement of the procedure described in Algorithm 3, or of the on-line learners in the next section that use this procedure.

This specific use of clauses constructed by an 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 augmenting any existing features with ILP-constructed relational ones can substantially improve the predictive power of a statistical model (see, for example, Joshi et al. 2008; Saha et al. 2012; Specia et al. 2009; Ramakrishnan et al. 2007).

3.2 On-line model construction using relational features

When considering how to apply ILP feature construction for stream learning with Winnow we have two algorithms: one based on the *fixed-attribute* setting, and one on the *infinite-attribute* setting. In both cases, relational feature construction is *inductive* and is performed as described in Sect. 3.1.

1 **Algorithm** `rel_model_finite_attr`$(B, I, \mathcal{L}, S, E_0)$
  Input: Background knowledge B; feature constraints I; language restriction $\mathcal{L}$; a stream S of relational example data; and a sample of pre-classified relational instances E_0.
  Output: Set of features F and a model M.
2 Let the initial mistakes $m_0 = 0$, and the initial model be M_0
3 Let $F = fconstruct(B, I, \mathcal{L}, E_0)$
4 **for** $t = 1, 2, \ldots$ **do**
5 Get new example $e_t = \langle \mathbf{x_t}, y_t \rangle$ from S
6 $\mathbf{x'_t} = feature\_vector(e_t, B, F)$
7 Predict $y'_t = predict(M_{t-1}, \mathbf{x'_t})$
8 **if** *prediction is incorrect (that is,* $y'_t \neq y_t$*)* **then**
9 $m_t = m_{t-1} + 1$
10 $M_t = update(M_{t-1}, \mathbf{x'_t}, y'_t, y_t)$
11 **else**
12 $m_t = m_{t-1}$
13 $M_t = M_{t-1}$
14 **return** $\langle F, M_t \rangle$

Algorithm 4: Single-Pass Relational Stream Modelling (Fixed-Attribute Space). A procedure for on-line learning from relational data in a fixed-attribute setting.

Algorithm 4 assumes a *fixed-attribute* representation, in which all instances are represented by a fixed-set of features that are obtained before stream-processing commences. The set of attributes is constructed before stream-based modelling commences, using a sample of pre-classified instances. This could be obtained by sampling the stream S, but we have chosen to provide it as a parameter E_0 to the procedure.

We assume a feature-constructor function *fconstruct* (line 3) is available—in this paper, this is provided by an ILP system (for example, as in Algorithm 3). The function $feature\_vector$ (line 6) converts the current example into a feature-vector representation using a set of features. The functions $predict$ at line 7 and $update$ at line 10 are as in Winnow (Algorithm 2).

In contrast, in Algorithm 5 the features are obtained while processing the data stream: each data instance is represented by a finite number of features, but the complete set of features is not known beforehand. This is the so-called *infinite-attribute* setting described in Blum (1992).

1 **Algorithm** `rel_model_infinite_attr` $(B, I, \mathcal{L}, S)$
  Input: Background knowledge B; feature constraints I; language restriction $\mathcal{L}$; and a stream S of relational example data.
  Output: Set of features F and a model M.
2 Let the initial mistakes $m_0 = 0$, and the initial model be M_0
3 $F_0 = \emptyset$
4 **for** $t = 1, 2, \ldots$ **do**
5 Get new example $e_t = \langle \mathbf{x_t}, y_t \rangle$ from S
6 $\mathbf{x'_t} = feature\_vector(e_t, B, F_{t-1})$
7 Predict $y'_t = predict(M_{t-1}, \mathbf{x'_t})$
8 **if** *prediction is incorrect (that is,* $y'_t \neq y_t$*)* **then**
9 $m_t = m_{t-1} + 1$
10 $F = fconstruct(B, I, \mathcal{L}, \{e_t\})$
11 $F_t = F_{t-1} \cup F$
12 $\mathbf{x''_t} = feature\_vector(e_t, B, F_t)$
13 $M_t = update(M_{t-1}, \mathbf{x''_t}, y'_t, y_t)$
14 **else**
15 $m_t = m_{t-1}$
16 $F_t = F_{t-1}$
17 $M_t = M_{t-1}$
18 **return** $\langle F_t, M_t \rangle$

Algorithm 5: Single-Pass Relational Stream Modelling (Infinite-Attribute Space). A procedure for on-line learning from relational data in an infinite-attribute setting.

Note that, in this setting, feature construction is invoked on each incorrect prediction, enabling the addition of a bounded set of new features to the representation. The procedure is more involved than in the fixed-attribute setting, because the (initially empty) feature set is expanded on each mistake (lines 10 and 11). The functions *predict* and *update* are as before.

3.3 Implementation

To implement our approach all that is needed is to couple an ILP system capable of relational feature construction with an on-line stream learning method. For the experiments in this paper both of these components are implemented within the Aleph ILP system (Srinivasan 1999), but that is not a requirement of our approach. In this section we outline the main components of our implementation. Prolog code extracts from this Aleph implementation illustrating the various aspects of our approach are in Appendix 1.

At the top-level we need to implement an approach for handling relational data streams that must process the examples appearing in a stream. Essentially this involves two main components: code to construct relational features (in our approach this is the principal ILP component, and realises Algorithm 3); and code to construct a model using an on-line or stream learning method (in our approach this is realised using two different on-line model constructors—see Sect. 4.2.4).

In our method this top-level is implemented by a user-defined predicate for the Aleph ILP system called "`aleph_stream/1`" which is described in more detail in Appendix 1. Note that the implementation of top-level processing needs to take into account the setting for the model-construction algorithm being used. For example, if the on-line learner is Winnow, processing examples on the stream may be done in either the fixed-attribute or the infinite-attribute setting. In the infinite-attribute setting Aleph feature construction

is invoked when prediction on the current stream element is incorrect, whereas in the fixed-attribute setting Aleph feature construction occurs prior to initialisation of stream learning.

It should also be noted that the modularity of our implementation within a general-purpose ILP system such as Aleph confers a number of advantages when it comes to extending our approach. Firstly, since the relational feature construction implementation is agnostic regarding the model-construction method being used, our approach can interface to *any* on-line learning method, whether implemented within Aleph or by invoking an externally implemented method. Secondly, via the same modular interface (refer to the predicate "`aleph_model/1`" in Appendix 1) we can also enable other useful computation on streams, such as change-detection. Finally, the use of Aleph enables the output of such methods to be employed as constraints for the model-construction process, for example, by restricting relational feature construction to situations in which the data stream appears to show concept drift.

4 Experimental evaluation

Both the components used to develop our approach—the ILP-based feature constructor, and the on-line model constructor—are not new, but surprisingly, not much is known about their combination to construct models for relational streams.[5] It is not known, for example: (a) how the procedures in Algorithms 4 and 5 compare against each other as techniques for model construction from streams of relational data; and (b) whether effective and efficient on-line learning is possible when ILP-based feature construction is performed during stream analysis. Concern stems from the fact that methods in both $construct\_features$ and $feature\_vector$ can be computationally expensive. In this section we report on experimental studies that provide an empirical basis for answers to these questions. Since this empirical study forms the main part of the paper it contains considerable detail, but to improve coherence we present it as a single section.

4.1 Aims

We would like to determine whether the procedures in Algorithms 4 and 5 satisfy the requirements in Sect. 1, rephrased here in the form of questions for three empirical studies, as follows.

4.1.1 Time

Do the implementations construct effective discriminatory models in an efficient manner, from relational data streams?

4.1.2 Space

Can the implementations handle indefinite amounts of relational data?

[5] On one view Algorithms 4 and 5 are adaptations to the on-line setting of ILP-based propositionalisation methods, and are related to the learning from interpretations approach of Lopes and Zaverucha (2009); our results can be seen as adding to what is known from that work.

4.1.3 Drift

Can the implementations track changes in the concept? From now on, we will refer to the experiments as Time, Space and Drift. For brevity, we will sometimes refer to the implementations as constructing Fixed-Attribute models or Infinite-Attribute models. We expect the experiments to yield insight into the conditions under which models are better.

4.2 Materials

4.2.1 Data and background knowledge

This comprises two categories: *Synthetic*, to enable controlled experiments; and *Real*, for exploration of novel, large, real-world ILP data sets and to enable comparison with previous work in ILP.

4.2.2 Synthetic

We use the "Trains" problem posed by R. Michalski for controlled experiments. Four datasets of sizes varying from about 100,000 examples to 1 million examples are obtained for randomly drawn target concepts (see Sect. 4.3 below). For this we use S.H. Muggleton's random train generator[6] that defines a random process for generating examples.

4.2.3 Real

We report results from uncontrolled experiments conducted using four real-world datasets: (a) *Malaria*. This is a Prolog-representation of about 10,000 anti-malarials, obtained from screening an industrial database. The task is to discriminate highly active compounds from less-active ones; (b) *HIV*. This is a Prolog-representation of the atom-bond structure of molecules in the NCI-HIV dataset consisting of approximately 50,000 compounds. The task is to discriminate active HIV inhibitors from the rest; (c) *Zinc*. This is a free database of commercially available compounds for virtual screening[7]. We use a subset of the "clean-drug-like" dataset. The data contains 1 million compounds, with the task of discriminating those that target normal proteins (called the "usual" compounds in the zinc database) with a pH value in the range 6–8), from others that target rare or metallo-proteases; and (d) *Cora*. The Cora dataset comes from a citation database of research papers in computer science (McCallum et al. 2000; Bilenko and Mooney 2003). It was first addressed as a relational learning task by Kok and Domingos (2005) and has since become somewhat of a benchmark dataset in ILP. In Kok and Domingos (2005) the task was formulated as an *entity resolution* problem, where essentially the goal is to determine whether two structured citation strings refer to the same paper. The dataset contains 1295 citations to 112 papers. This gives approximately 840,000 pairs of citations to be deduplicated (order within a pair is not important). Of these 31,429 are true duplicates, i.e., they refer to the same paper.

A summary of all datasets used is in Table 1. There some issues to be noted that arise with the real data sets. For 'Malaria' the costs of misclassification are not uniform. Misclassification of highly active molecules (+) is more expensive than that of less active molecules (−). The cost of false negatives is therefore higher than that of false positives (in Faruquie et al.

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

[7] http://zinc.docking.org/.

Table 1 Synthetic (top) and real (bottom) datasets used for experimental evaluation

Dataset	Examples
Trains_125 K	125,000
Trains_250 K	250,000
Trains_500 K	500,000
Trains_1 M	1,000,000

Dataset	Examples	Class distribution
Malaria	≈10,000	≈ 15% (+)
HIV	≈50,000	≈ 4% (+)
Zinc	1,000,000	≈ 42% (+)
Cora	≈840,000	≈ 4% (+)

Class distributions in the synthetic data vary, since target concepts are drawn at random

2013 this is estimated at about 10:1). The extremely skewed distribution of the HIV dataset, along with the inherent interest in HIV inhibition suggests that, once again, the cost of false negatives is higher than that of false positives. The purpose of testing on the Zinc data is to examine model construction on real-world data at sizes that have not been routinely analysed by ILP systems. It is not yet evident that there is, in fact, any model that can discriminate molecules in the pH 6–8 (+) from molecules outside this range (−). The class distribution of the Cora dataset is also skewed towards the negative class. Since the Cora dataset is included to enable comparison with previous work, only the default Aleph settings for class rebalancing were used. For each dataset we assume that ILP-specific input information in the form of domain-specific background knowledge B, feature constraints I and language restrictions $\mathcal{L}$ is available, as required by Algorithms 3, 4 and 5.

In the case of all three biological datasets, the description of a data instance consists of 3 components: (a) bulk properties of the molecule (like molecular weight); (b) the atoms in the molecule along with their types; and (c) the bonds between atoms. The background knowledge provided is of generic definitions of functional groups (methyl groups, alcohols, and the like), and cyclic structures (aromatic rings, 5-membered rings and so on). Their occurrence in any given molecule is then computed using the usual logical inference machinery employed by the ILP system. For the Cora dataset the data format follows the representation of Kok and Domingos (2005). In this representation[8] background predicates denote the relation between a pair of citations in terms of the similarities of the text fields for author, title, and the venue and year of publication. These are discretized at the levels of 0, 20, 40, 80 and 100% common words. The foreground predicates are positive (respectively negative) examples of duplicate (resp. non-duplicate) citations.

In these experiments the following feature constraints were applied: the `minacc` (precision) and `minpos` (recall) parameters were set to default values (0.75 and 2, respectively). Only definite clauses with at least one literal in the clause body were considered as candidate features. On biological data, clauses containing only literals for bulk properties (i.e., propositional literals) were not considered. No semantic constraints (i.e., those encoding domain knowledge) were used.

[8] Available for download at https://dtai.cs.kuleuven.be/ACE/doc.

4.2.4 Algorithms and machines

The data generator for controlled experiments 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 relational data object (nominally, a "train": see Michalski 1983).

All experiments here are conducted using the Aleph ILP system (Srinivasan 1999). The latest version of this program (available from the authors) includes support for on-line model construction with streaming data. Feature-construction by the ILP engine is coupled to two different on-line model constructors: Winnow (implemented within Aleph), and Hoeffding Trees (Domingos and Hulten 2000) (implemented within the MOA toolbox Bifet et al. 2010). For Drift experiments, a variant of Hoeffding Trees designed to handle concept drift called Adaptive Hoeffding Trees (Bifet and Gavalda 2007) is also investigated. Of these, only Winnow is used within the infinite-attribute setting, and both Winnow and Hoeffding Trees are used for model-construction in the fixed-attribute setting.

The experiments were conducted on an Intel Core i7 laptop computer, using VMware virtual machine running Fedora 13, with an allocation of 2GB for the virtual machine. The Prolog compiler used was Yap, version 6.1.3.[9]

4.3 Method

We distinguish between controlled experiments ("Time", "Space" and "Drift") that use synthetic datasets, and an uncontrolled experiment ("Real Data") that uses datasets from real-world applications. The former allow us to investigate the properties of on-line model construction with relational data, by varying some important problem or data characteristics. Experiments with the latter give us some evidence on the question of whether we should expect behaviour observed under controlled conditions with synthetic data to hold in practice, and enables a comparison with previous work.

In all cases, we assume the ability to generate a data stream, in which each data instance contains sufficient information to describe the relational structure of the instance. Further, for the fixed-attribute approach, we will assume that we have access to a set of pre-classified instances, from which the set of features are constructed (E_0 in Algorithm 4).

The following details apply specifically to Experiments 1 and 2 (Time and Space). To examine whether the methods are able to construct good models efficiently, we vary the problem and data characteristics along the following dimensions: **Target Concept** ($Target$)—we distinguish between $simple$ targets (requiring 1–4 features) and $complex$ targets (requiring 8–12 features); and **Data Stream** ($Data$)—we distinguish 4 different data stream lengths, ranging from 125,000 entries to 1,000,000 entries (see Sect. 4.2).

For simple targets, the number of features K is chosen randomly from the range 1–4. For complex concepts, the number of features K is randomly chosen from the range 8–12. K features are then randomly constructed using the ILP engine, and their disjunction constitutes the target concept.

For the fixed-attribute setting a fixed set of features are obtained using a stratified sample of 500 instances each from the "positive" and "negative" class. From these, the ILP engine constructs at most 5000 features. These features are then used to construct models using Winnow and on-line Hoeffding Tree builders in the MOA toolbox.

For the infinite-attribute setting, at most 25 new features are allowed for each data instance. This choice has no specific theoretical basis. The thresholds for the weighted linear combi-

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

nation of features is correspondingly equal to the number of features allowed for each data instance (that is, 25).

For either setting, the effectiveness of model construction is estimated by its predictive accuracy, and efficiency by the time taken to process all instances in the data stream. These estimates are average values of these quantities over the specified number of repetitions R. The predictive accuracy is measured on a "test set" of 1000 pre-classified data instances that are different to ones in the data stream (this is the hold-out performance Gama 2010). Thus, for each assignment of values to $(Target, Data)$ we will denote performance by the pair (A, T), the estimated accuracy and time taken for that assignment. Performance (A_1, T_1) will be said to be better than (A_2, T_2) if $A_1 > A_2$, or $A_1 = A_2$ and $T_1 < T_2$. In practice, we will take $>$ to mean "significantly greater", and $=$ to mean "approximately equal".

4.3.1 Experiment 1: Time

The method is as follows:

1. Repeat R times:

   For each combination of values for $(Target, Data)$ do:
   i. Randomly draw a concept C from $Target$.
   ii. Generate a data stream S using entries in $Data$. Classify each data instance as $+$ or $-$ using the target concept.
   iii. Construct a model using the on-line procedures in Algorithms 4 and 5 and the stream S
   iv In each case, record the predictive accuracy of the model constructed after having processed all elements in the data stream; and the total time to process all elements in the data stream.

2. Compute average values for predictive accuracy and time for stream data analysis; and compare performance.

For the Time, Space and Drift experiments the number of repetitions R is 10.

4.3.2 Experiment 2: Space

Since on-line learning processes data one-instance-at-a-time, we expect storage requirements to be bounded by the maximum complexity of any data instance. This does not change with the size of the data stream.

For the infinite-attribute setting, the number of features constructed can, however, grow linearly with the number of instances in the stream (since any one instance can be represented by up to n features). In addition, the number of features for either infinite- or fixed-attribute settings can increase with target complexity. We therefore vary the problem and data characteristics as in Experiment 1, namely: simple and complex targets $(Target)$; and varying data stream lengths $(Data)$, and assess changes in storage requirements as follows:

1. Repeat R times:

   For each combination of values for $(Target, Data)$ do:
   i Randomly draw a concept C from $Target$.
   ii Generate a data stream S using entries in $Data$. Classify each data instance as $+$ or $-$ using the target concept.

iii Construct a model using the on-line procedures in Algorithms 4 and 5 and the stream S
iv In each case, record the number of features constructed after having processed all elements in the data stream.

2. Compute average values of the number of features and compare performance.

Procedures and choices used for the generation of target concepts are the same as in Experiment 1.

4.3.3 Experiment 3: Drift

To investigate performance when there is concept (and possibly language) drift, we vary problem and data characteristics along two dimensions, **Target Concept** ($Target$) and **Shift Size** ($Shift$). As with the Time and Space experiments, we distinguish between $simple$ targets (now restricted to 4 features) and $complex$ targets (restricted to 8 features).

We distinguish 3 different shifts of the target concept. $Small$ shift denotes a small change in the target concept (1 literal for simple targets, and 2 literals for complex targets, denoting a Jaccard similarity of 0.6). A $large$ shift in the target concept denotes a bigger change in the target concept (2 literals for simple targets, 4 literals for complex targets, denoting a Jaccard similarity of 0.4). An $extreme$ shift in the target concept denotes an entirely different target concept (a 4 literal change for simple targets, and an 8 literal change for complex targets, denoting a Jaccard similarity of 0). Shifts of target concept in the stream are simulated by switching from one concept to the next. The method is as follows:

1. Repeat R times:

   For each combination of values for ($Target$, $Shift$) do:
   i. Randomly draw a concept C from $Target$
   ii. Obtain a concept C' from C by replacing literals determined by $Shift$
   iii. Let $Data$ consist of $2N$ instances
   iv. Generate a data stream S from entries in $Data$. The first N instances are classified as $+$ or $-$ using C and the subsequent N instances are classified $+$ or $-$ using C'
   v. Construct a model using the procedures in Algorithms 4 or 5
   vi. In each case, record the predictive accuracy of the model constructed after having processed all elements in the data stream.
   vii. For each method, if C and C' are both identified correctly then record $success$, otherwise record $fail$

2. Estimate the probability of concept recovery of each approach using the proportion of successes

Here N is 1000. That is, for the first 1000 instances, the target concept is C and for the subsequent 1000 instances, the target concept is C'. For the purpose of this experiment, we will assume a target has been correctly identified if the predictive accuracy is at least 95%.

4.3.4 Experiment 4: Real data

For these uncontrolled experiments, we clearly have no control over the target concept, or the size of the data stream. We have two principal aims.

The first is a straightforward comparison of the performance of the two methods of relational stream modelling, in one case on a novel, large dataset. Specifically:

For each real dataset D:

1. Generate a data stream S consisting of the training instances in D, along with their classifications into $+$ or $-$
2. Construct a model using the on-line procedures in Algorithms 4 and 5 using the B, I and $\mathcal{L}$ relevant for D
3. Record the predictions of the on-line models on test data

Compare performances of the on-line models from each algorithm

The following additional details are relevant for this experiment. As with the synthetic data, E_0 will consist of 1000 instances consisting of stratified samples of 500 instances each of positive and negative instances. Settings for feature construction are the same as that used for synthetic data.

It has been common practice in similar ILP applications (for example, the mutagenesis or carcinogenesis datasets: Srinivasan et al. 1996, 1997) to pre-compute for each data instance e all elements of the Herbrand base that are entailed by the background knowledge B and the instance e (in effect, a form of memoing or caching, used for efficiency). We continue to do this here, although we note that this is impractical for data streams with very large numbers of instances. Pre-computation can however be done for instances in E_0, if this is small (as is the case here): this may benefit the construction of fixed-attribute models.

For simplicity, in all three biological datasets, a misclassification cost ratio of 10:1 is used for the ratio of costs of false negatives to false positives. While the motivation for this for Malaria and HIV have already been noted, it is not evident that it is needed for Zinc. We take the position with the Zinc data that it is more important to classify correctly molecules that bind to usual targets. The misclassification cost is communicated to the on-line learner using a ratio of 10:1 of Winnow promotion and demotion rates. For the Hoeffding Trees, we employ a search procedure through a discretised space of its parameters to estimate the settings that will yield the highest true-positive rates.

As before, performance will be denoted by the pair (A, T). Given the strong emphasis we have adopted on avoiding false negatives, we will take A to be the model's true-positive rate (sometimes also called the *recall* or *sensitivity* of the model). For completeness, we will also provide the model's overall accuracy.

Our second aim is to evaluate performance of our on-line model construction approach on a reasonably large ILP benchmark dataset. For this we used the Cora dataset (Kok and Domingos 2005). We downloaded the dataset in ACE format from https://dtai.cs.kuleuven.be/ACE/doc. In this representation all background predicates are given as ground relations. As downloaded, this dataset is already divided into five sets of approximately equal training and test set splits (variation in split sizes is due to the need to ensure no set of examples from a pair of true duplicates was split across training and test sets) and we adopted this experimental setup. Although Kok and Domingos (2005) applied statistical relational learning using Markov Logic Networks to learn the *probability* that a pair of citations refer to the same paper, in our approach we learn a classifier. Accordingly, we followed the setting used in the VFILP (Very Fast ILP) approach (Cardoso and Zaverucha 2006) and report accuracy and time.

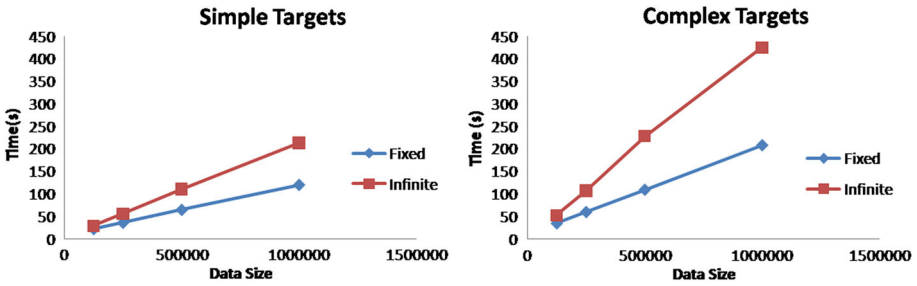


Fig. 2 Plots of results from Experiment 1: Time

4.4 Results and discussion

In this section results and discussion are given for each experiment in turn. Some summarisation and additional results are also included.

4.4.1 Experiment 1: Time

The plots in Fig. 2 show the principal performance trends in the fixed- and infinite-attribute settings. Since the features for both Winnow and Hoeffding-based model constructors are the same in the fixed-attribute setting, the plotted results from the Time experiments are simply labelled "Fixed" and "Infinite". The values plotted are averages over 10 repetitions.

For both fixed- and infinite-attribute settings the time taken appears to increase linearly with data-size. For the fixed-attribute setting this includes the time taken to construct features.[10] Actual values for the fixed-attribute setting are lower than those for the infinite-attribute setting (that is, the fixed-attribute setting appears to have a lower scaling constant than the infinite-attribute setting). Our results appear to indicate that models with Hoeffding trees are constructed faster than Winnow-based models.[11]

In Table 2 we give all runtimes from Experiment 1 for the fixed-attribute setting (both Winnow and Hoeffding trees), and the infinite-attribute setting (Winnow only); normalised times show the relative performance on both simple and complex target concepts. In all cases, accuracy is measured on an independent test set. Times reported for the fixed-attribute setting include time taken for: feature-construction, feature-testing and model-construction.

4.4.2 Experiment 2: Space

As for Experiment 1, the plots in Fig. 3 show the principal performance trends in the space requirements for the algorithms evaluated. Plots were produced by the same method as for Fig. 2.

Space requirements, measured in terms of the number of features needed, appears to be constant with increases in data-size for both fixed- and infinite-attribute settings (provided there is no concept-drift; see below). The actual value again appears to be lower for the fixed-attribute setting.

[10] Includes in all cases the time to evaluate the features and to construct the model.

[11] Some caution has to be adopted here, since the two implementations are on significantly different platforms (Fixed Winnow and Infinite Winnow are Prolog-based implementations within the Aleph ILP engine; Fixed Hoeffding and Adaptive Hoeffding are hybrid implementations, in which feature-construction is done by Aleph, and model-construction by an implementation in the MOA toolbox, written in Java).

Table 2 Summary of results from Experiment 1: Time

Model	Data	Accuracy (%)	Time (s)	Time (rel)
(a) Simple targets				
FH	125 K	100.0 (0.0)	10.9 (2.9)	1.0
	250 K	100.0 (0.0)	13.3 (3.0)	1.2
	500 K	100.0 (0.0)	18.1 (3.9)	1.7
	1 M	100.0 (0.0)	27.7 (5.5)	2.5
FW	125 K	100.0 (0.0)	22.3 (3.9)	1.0
	250 K	100.0 (0.0)	36.2 (6.0)	1.6
	500 K	100.0 (0.0)	64.7 (10.5)	2.9
	1 M	100.0 (0.0)	119.9 (21.1)	5.4
IW	125 K	100.0 (0.0)	28.9 (18.9)	1.0
	250 K	100.0 (0.0)	56.0 (36.4)	1.9
	500 K	100.0 (0.0)	110.4 (74.0)	3.8
	1 M	100.0 (0.0)	212.7 (135.8)	7.2
(b) Complex targets				
FH	125 K	99.8 (0.8)	14.5 (5.5)	1.0
	250 K	99.8 (0.8)	19.6 (4.6)	1.4
	500 K	99.8 (0.8)	29.9 (5.5)	2.1
	1 M	99.8 (0.8)	50.1 (10.4)	3.5
FW	125 K	99.8 (0.8)	34.6 (5.5)	1.0
	250 K	100.0 (0.0)	59.9 (11.2)	1.9
	500 K	100.0 (0.0)	109.4 (24.2)	3.1
	1 M	100.0 (0.0)	208.6 (49.9)	6.0
IW	125 K	100.0 (0.0)	52.3 (8.8)	1.0
	250 K	100.0 (0.0)	107.1 (12.1)	2.1
	500 K	100.0 (0.0)	228.2 (28.1)	4.4
	1 M	100.0 (0.0)	424.8 (77.4)	8.1

FH fixed Hoeffding, *FW* fixed Winnow, *IW* infinite Winnow. Time (rel) is normalised as a multiple of Time (s) on the smallest data set. An entry N in the Data column refers to the dataset Trains_N, as listed in the paper. The numbers tabulated are average values obtained from 10 repetitions. Standard deviations are in brackets

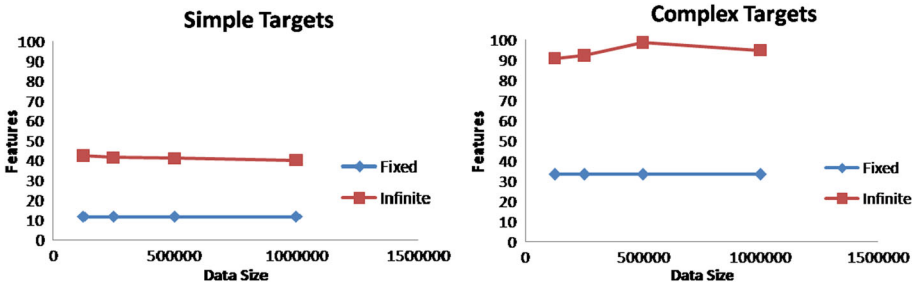
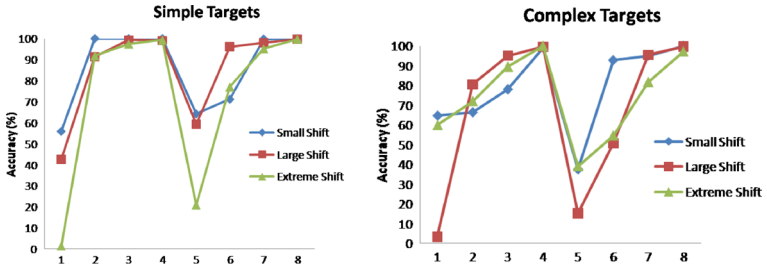


Fig. 3 Plots of results from Experiment 2: Space

4.4.3 Experiment 3: Drift

The key aspects of the graphs and tabulations of Fig. 4 are as follows. Only the infinite-attribute setting appears to recover reliably from shifts in target concepts on these data sets.



(a)

Shift	Model (after 2 epochs)		
	Features	Mistakes	Worst Case
Small	111	28	700
Large	130	31	775
Extreme	139	39	975

(b) Simple targets

Shift	Model (after 2 epochs)		
	Features	Mistakes	Worst Case
Small	150	77	1925
Large	163	35	875
Extreme	176	43	1025

(b) Complex targets

Shift	Feature Increase	
	Median	Mean (s.d.)
Small	29	25 (21)
Large	50	48 (22)
Extreme	66	72 (16)

(c) Simple targets

Shift	Feature Increase	
	Median	Mean (s.d.)
Small	38	34 (24)
Large	51	54 (17)
Extreme	75	61 (35)

(c) Complex targets

Model	Recovery Probability					
	Simple Targets			Complex Targets		
	Small	Large	Extreme	Small	Large	Extreme
Fixed Hoeffding	0.6 (0.2)	0.2 (0.1)	0.1 (0.1)	0.5 (0.2)	0.1 (0.1)	0.4 (0.2)
Adaptive Hoeffding	0.7 (0.1)	0.2 (0.1)	0.2 (0.1)	0.5 (0.2)	0.1 (0.1)	0.4 (0.2)
Fixed Winnow	0.2 (0.1)	0.1 (0.1)	0.6 (0.2)	0.1 (0.1)	0.4 (0.2)	0.6 (0.2)
Infinite Winnow	1.0 (0.0)	1.0 (0.0)	1.0 (0.0)	1.0 (0.0)	1.0 (0.0)	1.0 (0.0)

(d)

Fig. 4 Results from the Drift experiments. Graphs (a) show examples of recovery from concept drift by the on-line Winnow-based model constructor in the infinite-attribute setting. The graphs show two successive learning *epochs*, with time-stamps 1–4 and 5–8 on the X-axis each spanning 1000 data instances. A shift in the target concept occurs at $x = 5$, resulting in a drop in predictive accuracy, but the on-line learners recover by $x = 8$. Tables (b) summarise results after two epochs for models constructed in the infinite-attribute setting. "Features" is the number of features in the model, "Mistakes" is the number of mistakes during training, and "Worst case" is the theoretical maximum possible number of features (each value is the median of three replicates). Note that actual numbers of features are substantially lower than worst-case maximum in all cases. Tables (c) show median increases in number of features from one epoch to the next to achieve concept recovery in the infinite-attribute setting. Table (d) has estimates of the probability of recovery from a concept-shift. Concept-shifts range from small to extreme. Recovery is said to occur if the learner, having learned the original concept, also learns the new concept. The fraction tabulated is the average value (with associated standard error) of the probability of recovery

The price to pay is an increase in the number of features (that is, constant space can no longer be guaranteed).

Some further details regarding the results in Fig. 4 should be mentioned at this point. The graphs in Fig. 4a are learning curves measured in terms of the hold-out performance of on-line learners in the infinite-attribute setting when subject to a change of target concept between two learning *epochs*. The X-axis of each graph is marked by ordinal values simply representing progressively greater numbers of data instances: the values $\{1, 2, 3, 4\}$ span 1000 data instances, denoting the first epoch, as do the values $\{5, 6, 7, 8\}$, denoting the second epoch. The learning curves therefore show predictive accuracy after every 250 instances for each combination of target complexity and shift size.

The tables in Fig. 4b compare the actual number of features in each learned model after completion of two epochs with the theoretical maximum for a mistake-driven on-line learner in the infinite-attribute setting. Let F denote the number of features in the model, p denote the number of mistakes during training, and n denote the maximum number of features added to the model following each mistake (here $n = 25$). Then the maximum possible number of features that could be added to a model during learning is np. In Fig. 4b the values shown are the median numbers of features and mistakes observed in the experiment. Clearly the worst-case values (np) are substantially higher than observed values (F) in all cases. In particular, we can see that the number of features actually added as a percentage of the worst-case number ranges between 7.8% (small shift, complex targets) and 18.6% (large shift, complex targets). The tables in Fig. 4c illustrate recovery from concept shift after the first epoch in terms of the number of features added to the model (recall that no features are ever removed, although irrelevant features have their weights reduced arbitrarily close to zero). Since the mean is affected by some extreme values, the changes are better represented by the median. Additionally, we can see that the number of features added after the first epoch, i.e., to recover from the concept shift, as a percentage of the total number of features added ranges from 25.3% (small shift, complex targets) to 47.5% (extreme shift, simple targets).

From the experimental results we estimate in Fig. 4d the probability of recovery for different learners: two Winnow-based learners, in the fixed and infinite-attribute settings; and two stream-based learners from the MOA toolbox, Fixed Hoeffding (described above) and Adaptive Hoeffding, a method specifically designed to address concept-drift. By definition, recovery is said to occur if the learner, having learned the original concept, also learns the new concept. Here a concept is taken to be "learnt" if the accuracy on a holdout set—the predictive accuracy—is at least 95%. The fraction tabulated is the average value of the probability of recovery, estimated from 10 repetitions.

We note that given the design of the experiments, we can expect results to hold only on comparable hypothesis spaces. The results do not tell us, for example, that a data stream of $2N$ instances classified by a complex target would take twice as long to model as a data stream of N instances classified by a simple target. These observations lend empirical support to the following claims:

When there is no concept-drift: There exist on-line learners within both the fixed- and infinite-attribute settings that can use ILP-constructed features to identify models with high predictive accuracy in an efficient manner (that is, with time complexity that scales linearly with the number of instances processed);

When there is concept-drift: There exists at least one on-line learner within the infinite-attribute setting that can use ILP-constructed features to identify models that reliably track changing concepts.

The reader may be concerned at this point whether the synthetic problems are "too simple". That is, large-sized data streams may be irrelevant for the Trains problems, since targets could possibly be identified with substantially smaller subsets of the data. This is indeed so, but only under some specific circumstances. First, the representation language (here, Boolean functions of the features) must include the target concept. Secondly, the data must not be generated by an adversary. If features are identified inductively (as is done here) then the first condition cannot be guaranteed. If the data generation process is not under the experimenter's control (as is the case with observational data), then the second condition cannot be guaranteed. It is quite possible therefore for the Trains to pose substantial difficulties.

To see this, assuming that the target concept is in the hypothesis space (the first assumption above), we are able to quantify the difficulty that can be posed by the Trains to a Winnow-like learner. It has been shown that the number of examples needed for PAC-identification of a target concept is $\frac{M}{\epsilon}\ln(\frac{M}{\delta})$. Here M is the theoretical mistake bound computed for the learner, and PAC identification ensures that it will identify a concept with accuracy A of at least $(1-\epsilon)\times 100$ (%) with probability P at least $(1-\delta)$.

Experimental results shown in Table 8 yield estimates of $M = 24$ and $M = 129$ for simple and complex targets respectively. The number of instances needed in the worst case for almost certain identification ($P \geq 0.999$) of a very nearly correct model ($A \geq 99.9\%$) is approximately 240,000 for simple concepts and approximately 1.5 million for complex concepts. This gives some indication of the true nature of the difficulty of the synthetic problems, when no assumptions are made about the ordering of the examples.

4.4.4 Experiment 4: Real data

How might the on-line ILP engine perform in real-world settings? Tables 3 and 4 show the performance on the real datasets used here. We only tabulate the results obtained with the fixed-attribute setting here, since the experiments with synthetic data suggest that this may be the most effective approach if there is no concept drift. As described elsewhere in the paper, for the biological datasets we focus on the true-positive rate, since false negative errors are costlier than false positives. The values shown are true-positive rates on an independent test-set. For Hoeffding trees, the results are from a greedy search for good parameter values for model-constructors. On the Cora dataset we focus on predictive accuracy, as reported by Cardoso and Zaverucha (2006).

The results in Table 3 for the biological datasets confirm that both kinds of model constructor are able to find models that perform better than a predictor that simply returns the majority class. As with the synthetic data, Hoeffding Tree models are constructed faster (the same caveat expressed there applies here). The Winnow models appear to construct models with better true-positive rates; however, this may be an artifact of the search procedure for parameter values employed when constructing Hoeffding Trees. We caution against a misinterpretation of the runtimes reported across the biological datasets in Table 3. On face-value, it appears that the runtimes do not reflect the 1:5:100 scaling in the sizes of these datasets. However, this is due to the incomparable hypotheses-spaces of the three tasks (what we can expect is linear scaling *within* each task.)

The results on the Cora dataset in Table 4 are primarily for comparison purposes. Experiments on the Cora dataset in Cardoso and Zaverucha (2006) used propositionalization (Zelezny and Lavrač 2006) to construct relational features that were then used with feature selection and propositional learning. In Cardoso and Zaverucha (2006) progressive sampling (Provost et al. 1999) gave slightly better performance than the use of Hoeffding-bound sampling (Hulten et al. 2003) (96 vs. 94% accuracy), whereas selecting the 500 best

Table 3 Results on the real-world biological datasets

Model	True-positive rate (%)		
	Malaria	HIV	Zinc
(a) True-positive rates, with accuracies			
Fixed	22.2	96.7	24.9
Hoeffding	(80.4)	(95.2)	(60.0)
Fixed	54.2	100.0	60.5
Winnow	(63.6)	(95.2)	(52.9)
Majority	16.6	3.0	42.5
Class	(83.4)	(97.0)	(58.0)
Model	Time (s)		
	Malaria	HIV	Zinc
(b) Time			
Fixed	2872	166	4846
Hoeffding			
Fixed	2876	199	9126
Winnow			
Majority	–	–	–
Class			

In (a) the true positive rate, or recall, is the value of most interest, since the objective in each domain is to avoid misclassification of active molecules, i.e., those classified positive; for completeness we also include the overall accuracy in brackets. As a point of reference we also include true positive rates and accuracies for "Majority Class", i.e., the classifier that predicts the majority class (in each domain here, this is the negative class). In (b), the number in brackets denotes the actual time taken to construct a model (for the Hoeffding tree, this includes the times for constructing a feature-vector table). The ratio of the data sizes is approximately 1:5:100

Table 4 Results on the Cora citation deduplication dataset

Cora dataset	Fixed Winnow	$VFILP_1$	$VFILP_2$
Features constructed/selected	162 (66.4)	n/a	500
Feature construction time (s)	14.3 (11.9)	n/a	n/a
Test set accuracy	0.987 (0.011)	0.96	0.96
Total time (s)	246.5 (117.5)	n/c	n/c

Shown in the left column are sample means (standard deviations) of the main performance values for our Fixed Winnow variant. Shown in other two columns are the two best performing variants of VFILP (Cardoso and Zaverucha 2006) on this dataset, which we will refer to here as $VFILP_1$ and $VFILP_2$. Both variants use propositionalization (relational feature construction) (Zelezny and Lavrač 2006) and progressive sampling Provost et al. (1999), using as propositional learners either C4.5 Quinlan (1993) or the Very-Fast Decision Tree (VFDT) algorithm (Domingos and Hulten 2000). $VFILP_1$ uses all features constructed (number not available – n/a) whereas $VFILP_2$ uses only the best-performing 500 selected using the Hoeffding bound. Owing to differences in the platforms runtimes are not comparable—n/c. Refer to Cardoso and Zaverucha (2006) for further details

features rather than using all features did not affect the predictive accuracy on this data set. Using Aleph with no propositionalization in the study of Cardoso and Zaverucha (2006) as a baseline for comparison gave an accuracy of only 8%.

Table 5 Subset of features learned on a synthetic data set, ranked by weight

Feature no.	Weight	Feature
35	64.0	class(A,B):-has_car(A,C),shape(C,ellipse),load(C,rectangle,2)
1	32.0	class(A,B):-has_car(A,C),load(C,rectangle,1)
2	32.0	class(A,B):-has_car(A,C),load(C,rectangle,1),has_car(A,D)
5	16.0	class(A,B):-has_car(A,C),closed(C),load(C,rectangle,1)
21	16.0	class(A,B):-has_car(A,C),load(C,utriangle,3)
…	…	…
7	8.0	class(A,B):-has_car(A,C),long(C),load(C,rectangle,1)
…	…	…
4	4.0	class(A,B):-has_car(A,C),short(C),load(C,rectangle,1)
…	…	…
27	1.0	class(A,B):-has_car(A,C),shape(C,hexagon),load(C,circle,2)
…	…	…

Compared with the results on the Cora dataset in Cardoso and Zaverucha (2006) our results show a reduction in the number of features used (162 from 500) and a slight increase in predictive accuracy (98.7 vs. 96%). Owing the incompatibility of platforms due to hardware improvements over the ensuing period runtimes are not comparable.

In summary, we can conclude that the Winnow model appears to achieve at least equivalent performance in comparison to at least one previous ILP approach to scalable learning on relational data.

4.5 Supplemental Issues

We turn now to some less obvious questions.

4.5.1 Comprehensibility

Since relational feature construction is based on the results of ILP search, feature definitions for a learned model can be inspected as for any ILP theory. Shown in Table 5 are some of the top-ranked weights from a Winnow model; here the first feature matches the target concept. Other features with positive weights ≥ 2 are partly correct. Table 5 suggests that learned models can be humanly-comprehensible.

4.5.2 Drift recovery

The results in Fig. 4 show that the infinite-attribute setting is capable of reliably tracking changes in concepts. Experiments with Winnow have however shown that it is capable of tracking concepts even in the fixed-attribute setting. The poor performance of the fixed-attribute Winnow classifier therefore requires more investigation. The effect of two kinds of influences have not been examined in Fig. 4. First, does the rate of concept-change have an effect on recovery? For example, if the concept changed after a longer time, will the fixed-attribute approach recover? (In effect, this is the size of the *context* sets in Gama 2010) Some light on this question is shed in Table 6a. These tabulations suggest that recovery with

Table 6 Concept-recovery in the fixed-attribute setting

Shift	Recovery probability	
	$N = 1000$	$N = 2000$
(a) Concept changes once every N instances		
Small	0.2 (0.1)	0.7 (0.1)
Large	0.1 (0.1)	0.4 (0.2)
Extreme	0.6 (0.2)	0.3 (0.1)
Shift	Recovery probability	
	Without detector	With detector
(b) Using a perfect change detector ($N = 1000$)		
Small	0.2 (0.1)	0.7 (0.1)
Large	0.1 (0.1)	0.5 (0.2)
Extreme	0.6 (0.2)	0.3 (0.1)

In (a) the concepts change at different rates. In (b) a "perfect" change detector is used (*i.e.* the model construction process is told the rate of change). The meaning of recovery probability is as before, namely the probability that the concept identified is approximately the same as the target concept. This is taken to be the case if the predictive accuracy of the concept identified is at least 95%. For simplicity, we have only shown results here with "Simple" target concepts. The model is the Fixed Winnow model. Standard deviations are in brackets, and the estimates are from 10 repetitions

a fixed-attribute Winnow model may be possible if: (a) the concept-change is small; and (b) the rate of concept-change is not too high.

Perhaps the difficulties for the fixed-attribute system arise because the model-constructor has to detect changes automatically. Could we do better if the data analysis system were equipped with a change-detector? Various detectors are discussed in Gama (2010): without reference to any one of these, it is useful to see what happens in the fixed-attribute setting if perfect detection were possible. Table 6b shows that matters do improve somewhat. The probability of concept-recovery with a perfect detector is about the same that without, but without needing the slower rates of changes required for the latter. For extreme-shifts of concepts though, fixed-attribute models continue to perform poorly. This is not altogether unexpected.

4.5.3 Small streams

There is nothing in the streaming data model that restricts it to large numbers of data instances. Does the claim continue to hold with small relational data streams? Again, the answer is "yes", although the relative performances are somewhat different. From Table 7 clearly it is still possible to obtain good theories in an efficient manner. However, now the superiority of the fixed-attribute setting is less apparent, since predictive accuracy gains are marginal, and time for model construction is substantially more (in the fixed-attribute setting, the time to construct features dominates the time to update the model). Thus, if time is a constraint, the infinite-attribute setting may be better for small streams.

Table 7 Results with small data streams

Model	Data	Acc. (%)	Time (s)
(a) Simple targets			
	Trains_125	99.5 (0.7)	8.4 (2.7)
Fixed	Trains_250	99.7 (0.7)	8.4 (2.7)
Winnow	Trains_500	99.9 (0.2)	8.4 (2.7)
	Trains_1 K	100.0 (0.0)	8.5 (2.7)
	Trains_125	97.5 (5.4)	0.20 (0.2)
Infinite	Trains_250	99.2 (1.7)	0.20 (0.2)
Winnow	Trains_500	99.4 (0.9)	0.30 (0.3)
	Trains_1 K	99.9 (0.1)	0.40 (0.3)
Strategy	Data	Acc. (%)	Time (s)
(b) Complex targets			
	Trains_125	99.2 (1.0)	9.3 (5.9)
Fixed	Trains_250	99.7 (0.4)	9.3 (5.9)
Winnow	Trains_500	99.8 (0.3)	9.3 (5.9)
	Trains_1 K	99.9 (0.1)	9.5 (5.8)
	Trains_125	92.4 (7.4)	0.4 (0.1)
Infinite	Trains_250	95.5 (6.9)	0.5 (0.3)
Winnow	Trains_500	97.7 (2.0)	0.7 (0.3)
	Trains_1 K	99.2 (1.0)	1.0 (0.4)

Here the suffixes "125, 250, 500, 1 K" stand for 125, 250, 500 and 1000 data instances respectively. As before, accuracies are on an independent test set, and standard deviations are in brackets

4.5.4 Fixed and variable costs

In both the fixed- and infinite-attribute settings, we can distinguish 3 distinct tasks: (1) feature-construction; (2) feature-testing, or conversion of relational data into a feature-vector; and (3) model-construction. In addition, it is useful to partition the overall time for stream-processing into: *fixed* costs, that are once-off; and *variable* costs, that depend on the size of the data stream. For the fixed-attribute setting, feature-construction time is a fixed cost, feature-testing and model-construction times are variable costs. For the infinite-attribute setting, there are no fixed costs, and all three tasks contribute to the variable cost. It may be important keeping these distinctions in mind when examining runtimes. It is possible, for example, for fixed costs to dominate variable costs in the fixed attribute-setting. If this happens, the scaling of runtime with data size will only be observed if the fixed-costs are discounted (this happens with the Hoeffding trees: see Fig. 5).

4.5.5 Mistake bounds

Does the hybrid ILP-Winnow engine satisfy the mistake bounds (maximum number of mistakes) predicted for the Winnow algorithm? We note first that an upper bound on the number of mistakes made by Winnow for a concept defined by a disjunction of r features (as is done here) is $2 + 3r(1 + \log(n))$, where n is number of features used to describe a data instance. A proof of this for a fixed-set of attributes is in Littlestone (1988) and for the infinite-attribute setting is in Blum (1992). However, it is not evident that this bound holds when features

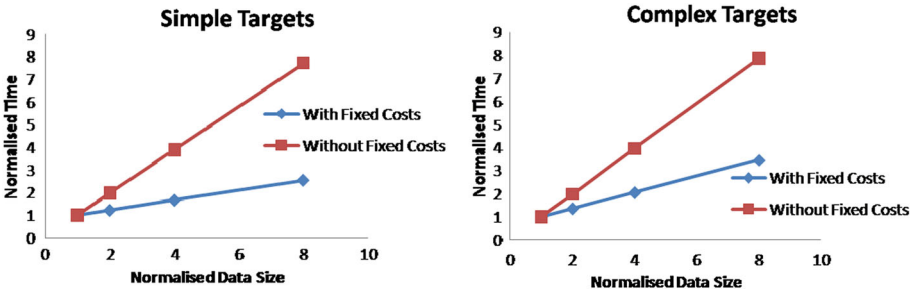


Fig. 5 Scaling with and without fixed costs. With the use of Hoeffding-tree model constructors, fixed costs (the cost for feature-construction) dominate. This means that the scaling of runtime with data size will only be observed if fixed costs are discounted. Here data sizes are progressively doubled (from 125,000 to 1,000,000 instances), and this is reflected once the fixed-cost effects are removed

Table 8 Comparison of the mistake-bound predicted by theoretical analysis for Winnow and the actual mistakes made by the ILP-Winnow engine

Trial (R)	Pred.	Act.
(a) Simple targets		
1	18	2
2	26	5
3	25	5
4	11	8
5	29	6
6	14	5
7	10	3
8	20	6
9	58	4
10	20	2
(b) Complex targets		
1	110	5
2	98	4
3	102	12
4	119	5
5	132	139 (*)
6	127	7
7	110	39
8	112	9
9	196	20
10	178	13

The data stream consists of the 1 million instances denoted "Trains_1M"; and the values shown are for the ILP-Winnow engine in the fixed-attribute setting. The row marked with a "*" denotes an instance where the Winnow bound is exceeded

are constructed inductively, as is done here. For both fixed, and infinite-attribute settings, the analysis assumes that the final set of features will contain the r features that define the target concept. When good features are identified from data, this cannot be guaranteed. It is therefore possible for the ILP-Winnow engine to make more mistakes than the theoretical bound. Fortunately, it appears that this may not be a routine phenomenon. The tabulation in Table 8 shows the predicted and actual mistakes for each repetition with the largest data

stream (Trains_1M) with fixed-attribute model construction. From these data, it appears that the probability of exceeding the Winnow bound may be no more than 1 in 10 (for simple or complex concepts).

5 Related work

As noted above, streams as a model for data and computation has early roots (Landin 1965). Hence it is not surprising that there is an extensive literature on learning from data streams (Aggarwal 2007; Gama 2010). Learning from potentially infinitely many examples, where the target concept and its representation may be evolving, and accurate prediction may be required at any time, presents challenges that have been addressed in a number of different ways that relate to our work. The first attempt to use propositionalization in combination with an algorithm capable of on-line learning in the context of scaling up ILP approaches was (Cardoso and Zaverucha 2006).

Although stream learning and incremental learning are not the same thing, it is worth noting that many early machine learning systems were incremental, and this was also the case for ILP. For example, two of the earliest implemented approaches to learning within a Logic Programming formalism (Sammut 1981; Shapiro 1981) used a strategy of generalising or specialising the theory being learned in response to a succession of input data, that is, a stream.

Incremental construction of theories is also the starting point for the introduction of learning from relational streams in Dries and De Raedt (2010), in which theories are constructed and updated within the learning-from-interpretations setting as each new interpretation is seen. This appears to be the first work in ILP to explicitly propose stream learning, in this case for clausal discovery. However, unlike our approach Dries and De Raedt (2010) assumes the target for discovery does not change, that is, it does not handle concept drift.

Algorithms that learn using some form of gradient descent update rule can be used for on-line learning by adopting the incremental or *stochastic* version of this approach, making adjustments to their parameters in response to feedback on prediction of a stream example by the current model (Guyon and Stork 2000). Although these algorithms are able, under certain conditions, to arbitrarily closely approximate the model that would be learned by the corresponding batch method (Bottou 1998), they assume that examples are randomly sorted and are not designed to handle concept drift. However, guarantees can be obtained for *multiplicative* weight update algorithms (Littlestone 1988; Freund and Schapire 1997) for on-line learnability.

Since propositional on-line learners such as Winnow (Littlestone 1988) treat each example only once they are inherently scalable to large datasets, and when extended to the infinite-attribute setting (Blum 1992) are scalable to very high dimensionality data as well. They also have theoretical guarantees of learning many concept classes within a bounded number of prediction errors or "mistakes", and often these bounds translate to PAC results. The Winnow algorithm can adapt to changes in the target concept, that is, concept drift, by demoting previously used features. Winnow is the main component in SNoW, a propositional learner which has been used with relational features, for example, to learn information extraction patterns from text (Roth and Yih 2001), although this was not in a streaming setting.

The PAC-learning framework is the basis of Valiant's robust logic (Valiant 2000), where first-order rules consist of features in a linear model learned using multiplicative weight updates. This is perhaps the work that is closest to ours; however, the propositional learning

stage applies to vectors of features denoting first-order literals, rather than clauses defined relative to general background knowledge as in our approach. Robust logic has been applied, for example, to a large-scale text dataset, on which relational rules were learned from a natural language corpus of half a million sentences (Michael and Valiant 2008). In Katakis et al. (2006) a heuristic method is described which is similar in motivation to Winnow, in that it is able to add new features (e.g., words to a vocabulary in text applications) and applies feature selection on an incremental basis. These properties are designed to handle concept drift. However, unlike Winnow with relational features, this method has no theoretical guarantees, does not use relational features and has only been tested on small datasets. Importantly, there are no guarantees provided that the incremental feature selection will converge to the set of relevant features, i.e., those defining the target concept, and hence be able to learn reliably (in terms of regret minimization) in general.

Within ILP several other attempts to address the scalability issue have been made; most relevant to our work are those focusing on efficient propositionalisation and sampling. In ILP there have been a number of approaches in which propositionalization occurs *during* the learning process, as in the infinite-attribute setting of our work. The first paper to describe such an approach appears to be Alphonse and Rouveirol (2000) who termed it "lazy propositionalisation". Following this were the approaches of Kramer (2001), Popescul and Ungar (2004), and Landwehr et al. (2006) in which this was termed "dynamic propositionalization". However, none of these feature construction approaches were used with a stream learning algorithm.

Sampling techniques from propositional learning were shown to improve the efficiency of standard ILP search without loss of accuracy in Srinivasan (1999). A combination of top-down and bottom-up search based on the "lazy" construction of a bottom clause was used to scale up an ILP system to handle a large background theory of approximately 600,000 facts (Tang et al. 2003). In the VFILP approach (Cardoso and Zaverucha 2006) propositionalization was used: either with progressive sampling (Provost et al. 1999) and C4.5 (Quinlan 1993); or with Hoeffding-bound sampling (Hulten et al. 2003) and the Very-Fast Decision Tree (VFDT) algorithm (Domingos and Hulten 2000). An empirical comparison of our relational streaming approach with the results for VFILP on the Cora benchmark dataset in Sect. 4.4 shows that they have similar performance.

As an alternative to learners using the linear models formalism for on-line learning, a tree-based approach was introduced by Domingos and Hulten (2000) that uses sampling to enable efficient incremental learning. A key property of this approach is that by an application of Hoeffding bounds the trees learned can be guaranteed to asymptotically closely approximate a tree learned by a batch approach. Hoeffding trees were lifted to a first-order representation based on TILDE (Blockeel and De Raedt 1998) in a system named HTILDE (Lopes and Zaverucha 2009) which was shown empirically to learn more efficiently on large data sets than TILDE. This approach to stream learning was then further extended in HTILDE-RT to learn relational regression trees (Menezes 2011). However, a technical difficulty prevents a direct empirical comparison to the techniques reported here. The implementation in Lopes and Zaverucha (2009), although designed for stream-based learning, uses the machinery provided by the ACE system,[12] which requires all examples to be stored in main memory This is in conflict with one of the driving motivations of this paper. Nevertheless, the encouraging results in Lopes and Zaverucha (2009) and our own findings here with use of Hoeffding trees suggests a promising way forward.

[12] http://dtai.cs.kuleuven.be/ACE/.

The Hoeffding tree approach (Domingos and Hulten 2000) has been generalised to the multi-relational learning task where the algorithm is given an extensional database and can learn attributes representing queries that are relevant to classifying instances of the target (Hulten et al. 2003). This does not, however, allow the use of general background knowledge for feature construction, in the manner we have done here.

6 Concluding remarks

Can an ILP system analyse millions of data instances effectively and efficiently? The experiments we have reported in this paper suggest that an on-line ILP system that uses a simple multiplicative-update algorithm can answer this question in the affirmative. Whether a fixed- or infinite-attribute setting should be employed depends on whether or not there is concept-drift. The results also suggest that if there is no concept-drift, an ILP-based system that builds tree-based models using on probabilistic sample bounds will also work well. The technique we have employed is in principle, applicable to indefinitely large datasets, and can be extended to perform unsupervised learning and the analysis of time-series.

There are a number of ways in which the work here can be extended. Renewed interest in on-line learning has resulted from the applicability of stochastic gradient descent (SGD) methods for a range of learning algorithms—see, e.g., Bottou (1998). This has allowed the development of efficient update rules for a number of classification, regression and clustering methods that perform some kinds of convex optimisation. It is now possible, for example, to devise very fast on-line variants of techniques like logistic regression and support-vector machines. The development of a general-purpose technique to allow the incorporation of SGD or a similar approach into on-line ILP systems will thus allow us to address a broader range of problems using more model-construction tools.

On the extension of existing methods, we can see three possibilities immediately. First, we have already mentioned the work of Lopes and Zaverucha (2009): it should be possible also to devise a true on-line version of this, which will effectively allow the construction of first-order Hoeffding trees in an infinite attribute space. Second, support-vector ILP has already been examined within the ILP setting (Muggleton et al. 2006): it is of interest to see how to obtain an on-line version of this, perhaps by employing SGD. Thirdly, and somewhat tangentially, we see a straightforward role for techniques like MapReduce for the parallel construction of feature-vectors. The use of MapReduce in Srinivasan et al. (2012) did not look at its use in ILP-based feature evaluation. Finally, all the results in this paper have been focused on a single stream of data. Encouraging results have been reported on relational models for forecasting with multiple streams of data (Ikonomovska and Dzeroski 2011), but no work appears to have been done on models for discrimination with multiple relational data streams. It remains to investigated whether classification problems can be handled simply as a special case of multi-stream regression modelling: the natural extension to the work here is a version of SNoW (Sparse Network of Winnows)[13] equipped with ILP feature-constructors.

Finally, by demonstrating the feasibility of analysing large datasets like Zinc, we hope to open the 'big data' door for ILP-based methods. This will only happen if even larger problems are addressed, in which important advantages of ILP like the use of background-knowledge and discovery of relational features are found to be beneficial. We believe the empirical investigation of the kind reported here will help clarify directions for future work of this nature.

[13] http://cogcomp.cs.illinois.edu/software/doc/snow-userguide.

Acknowledgements We thank the anonymous reviewers for detailed comments that have helped substantially to improve the paper. A.S. would also like to thank Ravi Kothari at IBM Research India for useful initial discussions on the streaming model. A.S. holds visiting positions at the School of Computer Science and Engineering, UNSW and at the Department of Computer Science, University of Oxford. Some part of this work was supported by a Ramanujan Fellowship from the Department of Science and Technology, Government of India. M.B. was supported in part by the Australian Government's Cooperative Research Centre on Smart Services.

Appendix: Implementation

Both the infinite and fixed-attribute approaches to on-line model construction can be implemented within the streaming mechanism provided by the Aleph ILP system (Srinivasan 1999). The principal technique employed by Aleph is to call a user-defined predicate `aleph_stream/1` with each each element of the data stream. `aleph_stream/1` is called by the `induce/1` predicate, which is the main interface provided by Aleph to its generalisation procedures. Here is a snippet of the code (we do not show book-keeping details):

```
% code to generalise instances in a stream
induce(stream):-
    setting(stream,Stream),             % stream is provided as an Aleph setting
    repeat,
        read(Stream,Example),
        (Example = end_of_file ->
            set(end_of_stream,true),
            aleph_stream(Example);      % aleph_stream is user-defined
            aleph_stream(Example),
            fail).

% code to construct features
induce(features):-
    <implementation of fconstruct function given in the text>

% code to construct a model (or update one if it already exists)
induce(model):-
    setting(model_type,ModelType),
    construct_model(ModelType).

construct_model(Model):-
    <implementation of model constructors like Winnow>
```

Both the fixed- and infinite-attribute settings can be implemented using the `aleph_stream` predicate. Here is an example of an implementation of model-construction in the infinite-attribute setting:

```
aleph_stream(Example):-
   process(Example).

process(Example):-
   correctly_predicted(Example),
   !.
process(Example):-
   induce(features),   % construct new features
   induce(model).      % update model
```

Assuming that features have been constructed beforehand, here is a definition of `process/1` (above) that implements model construction using the fixed-attribute setting:

```
process(Example):-
   correctly_predicted(Example),
   !.
process(Example):-
   induce(model).      % update model
```

Finally, it is possible to invoke any on-line model-constructor external to Aleph, using Aleph's user-defined `aleph_model/1` predicate:

```
construct_model(user):-    % called by induce(model) with parameter model_type=user
    !,
    aleph_model(Model),
    <book-keeping for storing Model>


aleph_model(Model):-
    <user-defined code that returns Model>
```

This modular implementation using Aleph's user-defined predicate "`aleph_model/1`" enables calls to external implementations of online learners to be made via user-defined code interfacing both to propositional methods such as SGD (stochastic gradient descent) and the MOA toolkit (Bifet et al. 2010), as well as relational online learners such as HTILDE (Lopes and Zaverucha 2009) for classification and HTILDE-RT (Menezes 2011) for regression tasks. Since the latter methods are relational tree-based methods extended for on-line learning to use the Hoeffding bound they can therefore be invoked in the same way as, e.g., Hoeffding trees in MOA. Software tools for online stochastic gradient descent include the SGD library[14] and Vowpal Wabbit.[15]

The mechanism within Aleph for customised model-construction allows us to specify several different ways of stream-processing in a (reasonably) declarative manner. It is possible, for example, use the `aleph_model/1` predicate to perform sliding-window computations over dependent instances in the following manner:

```
process(Example):-
   join_queue(buf,Example),    % store instance at the end of a queue (buf)
   induce(model),              % construct model, here by calling aleph_model/1
   serve_queue(buf,_).         % remove first element from queue (buf) to slide window

aleph_model(Model):-
   queue_to_list(buf,Examples),    % get all instances in a queue (buf)
   compute_model(Examples,Model),  % user-defined computation (e.g., median)
   update_model(model,Model).      % store model for any-time prediction
```

The ability to compute stream statistics such as medians over sliding windows has uses in, for example, monitoring and adapting to change (Aggarwal 2007, Bifet and Gavalda 2007, Gama 2010).

Finally, in addition to model-construction in the fixed- and infinite-attribute settings, it is straightforward to implement extensions that are useful in on-line learning. For example, a measure of control can be introduced into the generation of new features. We may, for instance, only want to construct new features if we are fairly sure of a shift in concept. The Page-Hinckley test is one way to detect such a change (Gama 2010), which can be incorporated as a guard on the induction of features. More generally, we envisage a set of constraints that have to be satisfied by an example before initiating feature-discovery. This can be implemented by rewriting the second clause for `process/1` (above) as:

[14] http://leon.bottou.org/projects/sgd.

[15] http://hunch.net/~vw.

```
process(Example):-
   focus(Example), !,  % constraints, e.g., Page-Hinckley test, to be satisfied
   induce(features),   % construct new features
   induce(model).      % update model
process(Example):-
   induce(model).      % only update the model (do not construct new features)
```

References

Aggarwal, C. (2007). *Data streams: Models and algorithms*. New York: Springer.

Alphonse, E., & Rouveirol, C. (2000). Lazy propositionalisation for relational learning. In W. Horn, (Ed.), *ECAI-2000: Proceedings of 14th European conference on artificial intelligence* (pp. 256–260).

Bifet , A, & Gavaldà, R. (2007). Learning from time-changing data with adaptive windowing. In *Proceedings of the seventh SIAM international conference on data mining* (pp. 443–448).

Bifet, A., Holmes, G., Kirkby, R., & Pfahringer, B. (2010). MOA: Massive online analysis. *Journal of Machine Learning Research*, *11*(2010), 1601–1604.

Bilenko, M., & Mooney, R. (2003). Adaptive duplicate detection using learnable string similarity measures. In *KDD-03: Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 39–48).

Blockeel, H., & De Raedt, L. (1998). Top-down induction of first order logical decision trees. *Artificial Intelligence*, *101*((1–2)), 285–297.

Blum, A. (1992). Learning boolean functions in an infinite attribute space. *Machine Learning*, *9*, 373–386.

Blum, A. (1997). Empirical support for winnow and weighted-majority algorithms: Results on a calendar scheduling domain. *Machine Learning*, *26*, 373–386.

Bottou, L. (1998). Online learning and stochastic approximations. In D. Saad (Ed.), *Online learning in neural networks* (pp. 9–42). Cambridge: Cambridge University Press.

Cardoso, P., & Zaverucha, G. (2006). Comparative evaluation of approaches to scale up ILP. In *Short papers of the 16th international conference on inductive logic programming (ILP 2006)* (pp. 37–39). Santiago de Compostela: UDC Press.

Carvalho, V., & Cohen, W., (2006). Single-pass online learning: Performance, voting schemes and online feature selection. In *KDD-2006: Proceedings of 12th international conference on knowledge discovery and data mining*.

Domingos, P., & Hulten, G. (2000). Mining high-speed data streams. In *KDD2000: Proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 71–80). ACM.

Dries, A., & De Raedt, L. (2010). Towards clausal discovery in stream mining. In *Inductive logic programming: 19th international conference, ILP 2009, Leuven, Belgium, July 02–04, 2009. Revised papers*, Vol. 5989 of *lecture notes in computer Science* (pp. 9–16).

Faruquie, T., Srinivasan, A., & King, R. (2013). Topic models with relational features for drug design. In F. Riguzzi, & F. Železný (Eds.), *Proceedings of the 22nd Interbational conference on inductive logic programming*, number 7842 in LNAI, pp. 45–57, Berlin: Springer.

Freund, Y., & Schapire, R. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, *55*(1), 119–139.

Gama, J. (2010). *Knowledge discovery from data streams*. Boca Raton: CRC Press.

Guyon, I., & Stork, D. (2000). Linear discriminant and support vector classifiers. In A. Smola, P. Bartlett, B. Schölkopf, & D. Schuurmans (Eds.), *Advances in large margin classifiers* (pp. 147–169). MIT Press.

Hand, D., Daly, F., Lunn, A., McConway, K., & Ostrowski, E. (1994). *A handbook of small data sets*. London: Chapman and Hall.

Hulten, G., Domingos, P., & Abe, Y. (2003). Mining massive relational databases. In *Proceedings of IJCAI-2003 workshop on learning statistical models from relational data* (pp 53–60).

Ikonomovska, E., & Dzeroski, S. (2011). Regression on evolving multi-relational data streams. In *Proceedings of the 2011 joint EDBT/ICDT Ph.D. Workshop, Uppsala, Sweden*.

John, G., Kohavi, R., & Pfleger, K. (1994). Irrelevant features and the subset selection problem. In W. Cohen, & H. Hirsh (Eds.), *Machine learning: Proceedings of the 11th International conference*. Morgan Kaufmann.

Joshi, S., Ramakrishnan, G., & Srinivasan, A. (2008). Feature construction using theory-guided sampling and randomised search. In F. Železný, & N. Lavrač (Eds.), *Proceedings of the 18th international conference on inductive logic programming*, number 5194 in LNAI, pp. 140–157. Berlin: Springer.

Katakis, I., Tsoumakas, G., & Vlahavas, I. (2006). Dynamic feature space and incremental feature selection for the classification of textual data streams. In *Proceedings of the ECML/PKDD-2006 International workshop on knowledge discovery from data streams* (pp. 107–116).

Kelly, J., & Hamm, S. (2013). *Smart machines: IBM's Watson and the Era of cognitive computing*. New York: Columbia University Press.

Kok, S., & Domingos, P. (2005). Learning the structure of Markov logic networks. In *Proceedings of the 22nd international conference on machine learning* (pp. 441–448).

Kramer, S. (2001). Demand-driven construction of structural features in ILP. In C. Rouveirol, & M. Sebag (Eds.), *ILP 2001: Proceedings of 11th international conference on inductive logic programming*, number 2157 in LNAI. Berlin: Springer.

Kramer, S., Lavrac, N., & Flach, P. (2001). Propositionalization approaches to relational data mining. In S. Dzeroski, & N. Lavrac (Eds.), *Relational data mining* (pp. 262–286). New York: Springer.

Landin, P. (1965). A correspondence between ALGOL 60 and Church's lambda notation. *Communications of the ACM*, *8*(2), 89–101.

Landwehr, N., Passerini, A., De Raedt, L., & Frasconi, P. (2006) K-Foil: Learning simple relational kernels. In Y. Gil, & R. Mooney (Eds.), *AAAI-2006: Proceedings of 21st national conference on artificial intelligence* (pp. 389–394).

Langford, J., Li, L., & Zhang, T. (2009). Sparse online learning via truncated gradient. *Journal of Machine Learning Research*, *10*, 777–801.

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

Lopes, C., & Zaverucha, G. (2009). HTILDE: Scaling up relational decision trees for very large databases. In *Proceedings of 24th annual ACM symposium on applied computing (SAC 2009)* (pp. 1475–1479). ACM.

McCallum, A., Nigam, K., Rennie, J., & Seymore, K. (2000). Automating the construction of internet portals with machine learning. *Information Retrieval*, *3*, 127–163.

Menezes, G. (2011). HTILDE-RT: Um Algoritmo de Aprendizado de Árvores de Regressão de Lógica de Primeira Ordem Para Fluxos de Dados Relacionais. Master's thesis, Universidade Federal do Rio de Janeiro.

Michael, L., & Valiant, L. (2008). A first experimental demonstration of massive knowledge infusion. In *KR-08: Proceedings of eleventh international conference on principles of knowledge representation and reasoning* (pp. 378–388).

Michalski, R. S. (1983). A theory and methodology of inductive learning. In R. Michalski, J. Carbonnel, & T. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (pp. 83–134). Palo Alto, CA: Tioga.

Muggleton, S., & Michie, D. (1997). Machine intelligibility and the duality principle. In H. Nwana, & N. Azarmi (Eds.), *Software agents and soft computing*, Vol. 1198 of *lecture notes in computer science* (pp. 276–292). Springer.

Muggleton, S., Lodhi, H., Amini, A., & Sternberg, M. (2006). Support vector inductive logic programming. In D. Holmes & L. Jain (Eds.), *Innovations in machine learning, studies in fuzziness and soft computing* (Vol. 194, pp. 113–135). Berlin: Springer.

Popescul, A., & Ungar, L. (2004). Dynamic feature generation for relational learning. In *3rd international workshop on multi-relational data mining*.

Provost, F., Jensen, D., & Oates, T. (1999) Efficient progressive sampling. In *KDD-99: Proceedings of the fifth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 23–32).

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Mateo, CA: Morgan Kaufmann.

Ramakrishnan, G., Joshi, S., Balakrishnan, S., & Srinivasan, A. (2007). Using ILP to construct features for information extraction from semi-structured text. *ILP*, *2007*, 221–224.

Ratnaparkhi, A. (1996). A maximum entropy model for part-of-speech tagging. In *Proceedings of the conference on empirical methods in NLP conference, EMNLP-96*.

Roth, D., Yih, W. (2001). Relational learning via propositional algorithms: An information extraction case study. In B. Nebel (Ed.), *Proceedings of the seventeenth international conference on artificial intelligence (IJCAI-01)* (pp. 1257–1263). Morgan Kaufmann.

Saha, A., Srinivasan, A., & Ramakrishnan, G. (2012). What kinds of relational features are useful for statistical learning? In F. Riguzzi, & F. Zelezny (Eds.), *ILP 2012*, volume LNAI 7842 (pp. 209–224). Springer.

Sammut, C. (1981). *Learning concepts by performing experiments*. PhD thesis, Department of Computer Science, University of New South Wales, Sydney, Australia.

Shapiro, E. (1981). An algorithm that infers theories from facts. In A. Drinan (Ed.), *IJCAI-81: Proceedings of the 3rd international joint conference on artificial intelligence* (pp. 446–451). Los Altos, CA: Morgan Kaufmann.

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

Srinivasan, A. (1999). The Aleph manual: Version 4 and above.

Srinivasan, A., & King, R. (1996). Feature construction with inductive logic programming: A study of quantitative predictions of biological activity aided by structural attributes. In S. Muggleton (ed.), *ILP'96: Proceedings of 6th inductive logic programming workshop*, volume LNAI 1314 (pp. 89–104).

Srinivasan, A., King, R., Muggleton, S., & Sternberg, M. (1997). Carcinogenesis predictions using ILP. In N. Lavrac, & S. Dzeroski (Eds.), *ILP-97: Proceedings of 7th international workshop on inductive logic programming*, volume 1297 of *Lecture notes in computer science* (pp. 273–287). Springer.

Srinivasan, A. (1999). *The Aleph Manual*. Available at http://www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/.

Srinivasan, A., Faruquie, T., & Joshi, S. (2012). Data and task parallelism in ILP using MapReduce. *Machine Learning*, *86*(1), 141–168.

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

Tang, L., Mooney, R., & Melville, P. (2003). Scaling up ILP to large examples: results on link discovery for counter-terrorism. In *Proceedings of the KDD-2003 workshop on multi-relational data mining* (pp. 107–121).

Valiant, L. (2000). Robust logics. *Artificial Intelligence*, *117*(2), 231–253.

Zelezny, F., & Lavrač, N. (2006). Propositionalization-based relational subgroup discovery with RSD. *Machine Learning*, *62*, 33–63.