An Empirical Study of On-Line Models for Relational Data Streams

Ashwin Srinivasan[1] Michael Bain[2]

[1] Department of Computer Science, IIIT, New Delhi, India
ashwin@iiitd.ac.in
[2] School of Computer Science & Engineering, UNSW, Sydney, Australia
mike@cse.unsw.edu.au

Technical Report
UNSW-CSE-TR-201401
January 2014

THE UNIVERSITY OF
NEW SOUTH WALES

School of Computer Science and Engineering
The University of New South Wales
Sydney 2052, Australia

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.

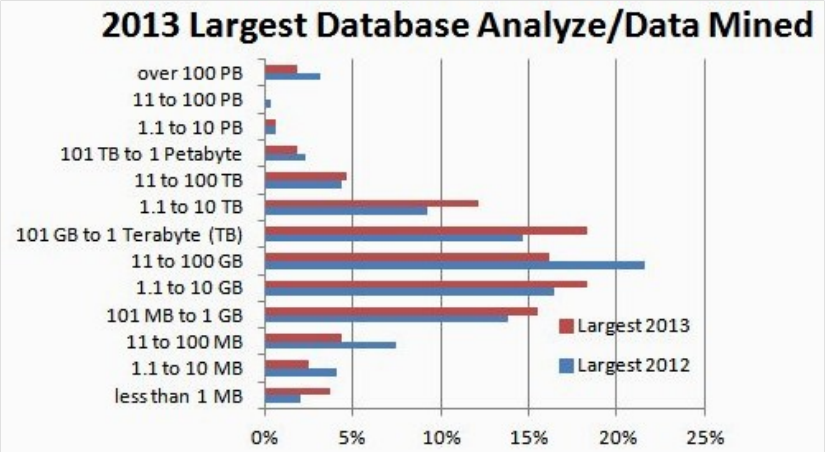


Figure 1.1: Results of a poll on the largest sized datasets analysed (from KDnuggets April 2013 – used with permission).

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 in 2013 were normally around a terabyte or so, with some much larger in size (see Figure 1.1). It is not clear what form the analysis took, but in ILP terms, this is substantially larger than the biggest problems considered so far.

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 ([12] 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 [26]. 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 [16]. 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)[2].

[1] `http://www.kdnuggets.com/2013/04/poll-results-largest-dataset-analyzed-data-mined.html`

[2] It seems, prior to our work, that two artificial relational regression data sets, each of one million examples, were the largest ever used by an ILP system (HTILDE-RT) [21].

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;
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" ptoblem).

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 — by this, 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 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 bio-chemical datasets examined by ILP systems to date.

The rest of the paper is organised as follows. Section 2 describes work in stream-based on-line learning that is relevant to the paper. In Section 3 we describe our approach to relational data stream analysis. Section 4 describes experiments with synthetic data sets, and some large real-world datasets. Section 5 describes previous and related work. Section 6 concludes the paper. Appendix A has details relevant to our implementation and experiments.

2 On-Line Learning with Streaming Data

Data streams—dating back at least to the 1960s [17]—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 [3] 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 [10] for a very readable text on the area; and to [1] 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. For example, techniques for on-line learning have been substantially bolstered recently by developments in the method of stochastic gradient descent (SGD) [5]. With SGD, an example-by-example update rule is obtained to estimate the parameters of a convex objective function. This has allowed on-line versions of some established techniques for discrimination, like support-vector machines and logistic regression.

The straightforward way to adopt these methods to a data stream of relational data 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. 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 sufficiently representative of the data; and there is no opportunity to alter the feature-set were the data distribution to change.

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 both with very large numbers of irrelevant features and drifting distributions. Winnow [19] uses a linear threshold discriminator, and a multiplicative weight-update scheme. At any instant t the algorithm represents each data instance using a Boolean vector $\vec{x_t}$. Surprisingly, following the work of [4] on an *infinite-attribute* model for Winnow-like algorithms, there is no requirement that each data instance use the same set of features. Thus the $\vec{x_t}$ could all be of different dimensionality (care has to be taken of course to ensure that the naming of features remains consistent)[3].

[3]This is not the same as using a sparse-vector representation (as in e.g., [18], 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 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 [15], is handled

This simple algorithm has be shown to satisfy all the desirable requirements just listed. 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 [19].

3 Relational Data Stream Analysis

We now examine the possibility of using two well-understood building blocks (feature-construction by an ILP system and on-line model construction) to construct models for relational data streams.

Two model-constructors for relational data resulting from using this approach are shown in Figs. 3.1 and 3.2. Both can be implemented quite simply within an existing ILP system: the details relevant to a specific implementation are in Appendix A. ILP-based feature construction has been well-studied; see, e.g., [32, 14]. Briefly, a feature f — more correctly, a feature function — is a Boolean function of a relational instance x, defined using an ILP system, background knowledge, constraints, etc., such that $f(x) = TRUE$ if the ILP system can find a definition for x in its hypothesis space, otherwise $f(x) = FALSE$.

For example, $f_1(m) = TRUE$ for any molecule m if the molecule has 3 fused benzene rings; otherwise $f_1(m) = FALSE$. The task of the ILP engine is find definitions like these, given definitions of general cyclic structures (like benzene rings), functional groups (like methyls, alcohols *etc.*) and so on. For streams we have two choices. In Fig. 3.1 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 [4]. In contrast, Fig. 3.2 assumes a *fixed-attribute* representation, in which all instances are represented by a fixed-set of features that are obtained before stream-processing commences.

Both the components used here—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.[4] It is not known, for example: (a) how the procedures in Figs. 3.1 and 3.2 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 both *construct_features* and *feature_vector* can both be computationally expensive. Next we conduct experimental studies that provide an empirical basis for answers to these questions.

by the infinite-attribute setting.

[4]We note that the procedure in Fig. 3.2 is just an adaptation to the on-line setting of the approach adopted by ILP-based "propositionalisation" methods. This is related to the learning from interpretations approach studied by [20], and the results here can be seen as adding to what is known from that work.

Single-Pass Relational Stream Modelling (Infinite Attribute Space):

rel_model_infinite_attr$(B, I, \mathcal{L}, S)$**:** Background knowledge B; feature constraints I; language restriction $\mathcal{L}$; and a stream S of relational data instances.

Return: A set of features and a model

1. Let the initial mistakes $m_0 = 0$
2. Let the initial model be M_0
3. $F_0 = \emptyset$
4. for $t = 1, 2, \ldots$:
   (a) Get new example e_t from S and its prediction y_t
   (b) $\vec{x'_t} = feature\_vector(e_t, B, F_{t-1})$
   (c) Predict $y'_t = predict(M_{t-1}, \vec{x'_t})$
   (d) If prediction is incorrect (that is, $y'_t \neq y_t$)
      i. $m_t = m_{t-1} + 1$
      ii. $F = construct\_features(B, I, \mathcal{L}, \{(e_t, y_t)\})$
      iii. $F_t = F_{t-1} \cup F$
      iv. $\vec{x_t} = feature\_vector(e_t, B, F_t)$
      v. $M_t = update(M_{t-1}, \vec{x_t})$
   (e) otherwise
      i. $m_t = m_{t-1}$
      ii. $M_t = M_{t-1}$
      iii. $F_t = F_{t-1}$
5. return $\langle F_t, M_t \rangle$

Figure 3.1: Steps of a procedure performing on-line learning from relational data streams in an infinite-attribute setting. We assume a feature-constructor function *construct_features* is available—in this paper, this will be provided by an ILP system. The function *feature_vector* converts the relational data instance into a feature-vector representation using a set of features. The functions *predict* and *update* are as before.

4 Experimental Evaluation

We would like to examine if the procedures in Figs. 3.1 and 3.2 satisfy the requirements in Section 1, rephrased here in the form of questions for 3 empirical studies.

4.1 Aims

Time. Do the implementations construct effective discriminatory models in an efficient manner, from relational data streams? **Space.** Can the implementations handle indefinite amounts of relational data? **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 Infinite-Attribute models or Fixed-Attribute models. We expect the experiments to yield insight into the conditions under which models are better.

Single-Pass Relational Stream Modelling (Fixed-Attribute Space):

$rel\_model\_finite\_attr(B, I, \mathcal{L}, S, E_0)$**:** Background knowledge B; feature constraints I; language restriction $\mathcal{L}$; a stream S of relational data instances; and a sample of pre-classified data instances E_0

Return: A set of features and a model

1. Let $F_0 = construct\_features(B, I, \mathcal{L}, E_0)$
2. Let the initial model be M_0
3. for $t = 1, 2, \ldots$:
   (a) Get new example e_t from S and its prediction y_t
   (b) $\vec{x_t} = feature\_vector(e_t, B, F_{t-1})$
   (c) Predict $y'_t = predict(M_{t-1}, \vec{x'_t})$
   (d) If prediction is incorrect (that is, $y'_t \neq y_t$)
      i. $m_t = m_{t-1} + 1$
   (e) otherwise
      i. $m_t = m_{t-1}$
   (f) $M_t = update(M_{t-1}, \vec{x_t})$
   (g) $F_t = F_{t-1}$
4. return $\langle F_t, M_t \rangle$

Figure 3.2: Steps of a procedure performing on-line learning from relational data in a fixed-attribute setting. Here, a fixed set of attributes are 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.

4.2 Materials

Data and Background Knowledge

This breaks down into two categories:

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 "Methods" below). For this we use S.H. Muggleton's random train generator[5] that defines a random process for generating examples.

Real. We report results from uncontrolled experiments conducted using three 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; and and (c) *Zinc.* This is a free database of commercially available compounds for virtual screening[6]. 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

[5] `http://www.doc.ic.ac.uk/~shm/Software/GenerateTrains/`
[6] `http://zinc.docking.org/`

Dataset	Examples
Trains_125K	125,000
Trains_250K	250,000
Trains_500K	500,000
Trains_1M	1,000,000

Dataset	Examples	Class Distribution
Malaria	$\approx 10,000$	$\approx 15\%(+)$
HIV	$\approx 50,000$	$\approx 4\%(+)$
Zinc	1,000,000	$\approx 42\%(+)$

Figure 4.1: Synthetic (left) and real (right) datasets used for experimental evaluation. Class distributions in the synthetic data vary, since target concepts are drawn at random.

(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.

A summary of all datasets used is in Fig. 4.1. There some issues 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 [8] 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 (−).

In all three cases, 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 occurence in any given molecule is then computed using the usual logical inference machinery employed by the ILP system.

In addition, we will assume that ILP-specific input information in the form of domain-specific background knowledge B, feature constraints I and language restrictions $\mathcal{L}$. Appendix A contains examples of both synthetic and real data instances and other inputs provided.

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 [23]).

All experiments here are conducted using the Aleph ILP system[7]. The latest version of this program (available from the authors) includes support for online model construction with streaming data. Feature-construction by the ILP

[7] `http://www.cs.ox.ac.uk/activities/machlearn/Aleph/aleph.html`

engine is coupled to two different on-line model constructors: Winnow (implemented within Aleph), and Hoeffding Trees [7] (implemented within the MOA toolbox [3]). For Shift experiments, a variant of Hoeffding Trees designed to handle concept drift called Adaptive Hoeffding Trees [2] 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. Aleph-specific implementation details are in Appendix A.

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[8].

4.3 Method

We distinguish between controlled experiments ("Time", "Space" and "Drift") that use synthetic datasets, and an uncontrolled experiment ("Real Data") that use 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.

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 Fig.3.2).

Experiment 1: Time

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 Section 4.2). Our method is as follows:

1. Repeat R times:

   For each combination of values for $(Target, Data)$ do:

   i. Randomly draw a concept C from $Target$. Generate a

   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 Fig. 3.1 and 3.2 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.

We refer the reader to Appendix A for additional details.

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

Experiment 2: Space

Since on-line learning processes 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 ($Data$). Our method for assessing changes in storage requirements is as follows:

1. Repeat R times:

   For each combination of values for $(Target, Data)$ do:

   i. Randomly draw a concept C from $Target$. Generate a

   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 Fig. 3.1 and 3.2 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.

We adopt the same procedures and choices as Experiment 1 for generation of target concepts and refer the reader to the details in Appendix A pertaining to Experiment 1 for details.

Experiment 3: Drift

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

We simulate shifts in the target concept by switching from one concept to the next. Our 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 on-line procedures in Figs. 3.1 and 3.2 (the details of the data used to obtain features for the latter are in Appendix A)
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 (or very closely) 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%.

Experiment 4: Real Data

For uncontrolled experiments, we clearly have no control over the target concept, or the size of the data stream. Our principal focus here therefore is a straightforward comparison of performance of the two methods of relational stream modelling. 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 Fig. 3.1,3.2 using the B, I and $\mathcal{L}$ relevant for D
3. Record the predictions of the on-line models on test data

- Compare the performances of the on-line models

We refer the reader to Appendix A for additional details.

4.4 Results and Discussion

The principal findings from Time, Space and Drift experiments are shown in Figs. 4.2 and 4.3. Table 4.1 suggests that learned models can be humanly-comprehensible. Complete tabulations on synthetic data for Time and Space experiments are in Appendix A (Figs. A.1–A.2).

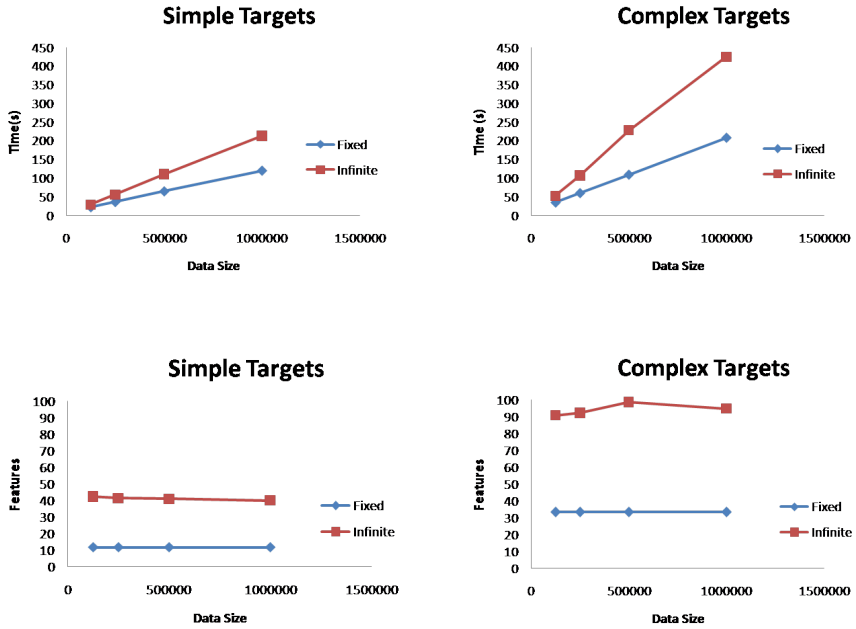


Figure 4.2: Results from the Time and Space experiments. The graphs show the principal performance trends of in the fixed- and infinite-attribute settings. Here "(W)" denotes the Winnow-based model constructor, and "(H)" the Hoeffding-tree based model constructor. The features for both Winnow and Hoeffding-based model constructors are the same in the fixed-attribute setting. So the results from the Space experiments are simply labelled "Fixed" and "Infinite". The values plotted are averages over 10 repetitions. The complete tabulations are in Appendix A.

Table 4.1: A subset of features and their weights learned on a synthetic data set. Shown 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.

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)
...		

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)

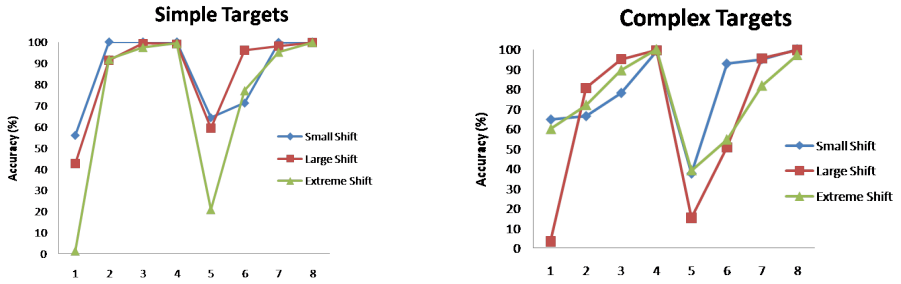
(a)

Shift	Feature Increase
Small	29
Large	50
Extreme	66

(b) Simple targets

Shift	Feature Increase
Small	38
Large	51
Extreme	75

(b) Complex targets



(c)

Figure 4.3: Results from the Drift experiments. Table (a) has estimates of the probability of recovery from a concept-shift. Concept-shifts range from small to extreme (the quantity in braces is the associated standard error). 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. Adaptive Hoeffding refers to a version in the MOA toolbox specifically designed to address concept-drift. Table (b) shows the median increase in the number of features for the infinite-attribute setting in order to achieve concept recovery. The graphs in (c) show an example of recovery by the on-line models. The X-axis are ordinals simply representing progressively greater numbers of data instances (the values $\{1, 2, 3, 4\}$ span 1000 data instances, as do the values $\{5, 6, 7, 8\}$). A shift in the concept occurs at $x = 5$ resulting in a drop in predictive accuracy. In the example shown, the model-builder recovers by $x = 8$ (the graphs shown use the Winnow-based model constructor).

The main aspects from the graphs and tabulations are these:

Time. 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[9]. 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.[10]

Space. 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.

Drift. Only the infinite-attribute setting appears to recover reliably from shifts in target concepts on these data sets. The price to pay is an increase in the number of features (that is, constant space can no longer be guaranteed).

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

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

[10]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).

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 Fig. A.4 in Appendix A 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 identification ($A \geq 99.9\%$) 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.

How might the on-line ILP engine perform in real-world settings? Fig. 4.4 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, 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.

The results confirm that both kinds of model constructors 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 real datasets in Fig. 4.4. On face-value, it appears that the runtimes do not reflect the 1:5:100 scaling in the sizes of the real datasets. However, this is due to the incomparable hypotheses-spaces of the three real-world tasks (what we can expect is linear scaling *within* each task.)
We turn now to some less obvious questions.

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 Fig. A.3 (Appendix A), it is evident that 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.

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 [19] and for the infinite-attribute setting is in [4]. However, it is not evident that this bound holds when features are

Model	True-Positive Rate (%)		
	Malaria	HIV	Zinc
Fixed Hoeffding	22.2 (80.4)	96.7 (95.2)	24.9 (60.0)
Fixed Winnow	54.2 (63.6)	100.0 (95.2)	60.5 (52.9)
Majority Class	16.6 (83.4)	3.0 (97.0)	42.5 (48.5)

(a) Predictive accuracies

Model	Time (s)		
	Malaria	HIV	Zinc
Fixed Hoeffding	2872	166	4846
Fixed Winnow	2876	199	9126
Majority Class	–	–	–

(b) Time

Figure 4.4: Results from the Real experiments. In (a) the number in braces is the overall accuracy. "Majority Class" is the classifier that predicts the majority class (here, the negative class). In (b), the number in braces 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.

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 routine phenomenon. The tabulation in Fig. A.4 in Appendix A 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).

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. 4.5).

5 Related Work

As noted above, streams as a model for data and computation has early roots [17]. Hence it is not surprising that there is an extensive literature on learning from data streams [1, 10]. Learning from potentially infinitely many examples, where the target concept and its representation may be evolving, and accurate predic-

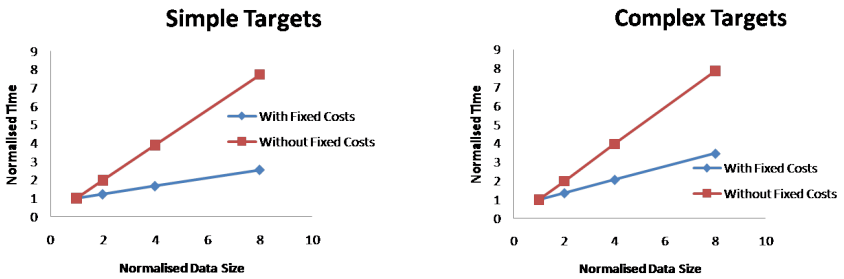


Figure 4.5: Scaling with and without fixed costs. With the use of Hoeffding-tree model constructors, fixed costs (the cost for feature-construction) dominate. This means that the scaling of runtime with data size will only be observed if fixed costs are discounted. Here data sizes are progressively doubled (from 125,000 to 1,000,000 instances), and this is reflected once the fixed-cost effects are removed.

tion may be required at any time, presents challenges that have been addressed in a number of different ways that relate to our work.

Many early machine learning systems were incremental, assuming that examples were given one-at-a-time, and this was also the case for ILP. For example, the two earliest implemented approaches to learning within a Logic Programming formalism [29, 31] used a strategy of generalising or specialising the theory being learned in response to a succession of input data, that is, a stream. In the case of Marvin [29], this was partly due to motivations from cognitive psychology [6], whereas MIS [31] was developed within the framework of Inductive Inference [11], which is inherently incremental. However, ILP moved towards batch learning as a way to obtain increased efficiency, in order to handle larger datasets—at the time, in the hundreds of examples [24, 28]. This move was a recognition that incremental learning operators, often requiring large amounts of search which was largely repeated for successive examples, were too computationally expensive for large streams. An additional advantage of efficient batch learners was to enable the adoption of probabilistic data-driven evaluation functions for generalisation or specialisation searches [28, 27], rather than *ad hoc* heuristics as typically had been adopted for incremental theory-driven approaches. These aspects of ILP search and evaluation remain in our approach to relational feature constuction, but within an online learning setting where they can be more tightly controlled.

Algorithms that learn using some form of gradient descent update rule can be used for online 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. Although these algorithms are able, under certain conditions, to arbitrarily closely approximate the model that would be learned by the corresponding batch method [5], 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 [19, 9] for online learnability.

Beyond ILP it had become clear more broadly in machine learning that, in addition to large numbers of instances, large numbers of attributes were

becoming common in real-world applications. This led to the development of *attribute-efficient* learners [19, 37] in computational and statistical learning theory. While both settings are attractive due to their sound theoretical basis, most attention was focused on kernel methods [30], Unfortunately, since these solve an optimization problem in the dual of the feature space they do not scale well to datasets with large numbers of instances.

Online learners such as Winnow [19] are inherently scalable to large datasets, and when extended to the infinite attribute setting [4] 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 algorithm can adapt to changes in the target concept, that is, concept drift, by demoting previously used features.

Valiant's proposal for robust logic [36], where first-order rules are comprised of features in a linear model learned using multiplicative weight updates, is perhaps the work that is closest to ours; however, the propositional learning stage applies to vectors of features denoting literals rather than clauses defined relative to background knowledge. The general framework has been used in applications to large-scale data such as text-mining [22] on a natural language corpus of half a million sentences.

As an alternative to learners using the linear models formalism for online learning, a tree-based approach was introduced by Domingos and Hulten [7] that uses sampling to enable efficient incremental learning. An important advantage of this approach is that by application of Hoeffding bounds the trees learned can be guaranteed to asymptotically closely approximate a tree learned by a batch approach. This representation was lifted to first-order trees for both classification [20] and regression [21]; the latter shows that the same asymptotic properties will hold for first-order trees as in the propositional setting. However, a technical difficulty prevents a direct empirical comparison to the techniques reported here. The implementation in [20], although designed for stream-based learning, uses the machinery provided by the ACE system[11], 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 [20] and our own findings here with use of Hoeffding trees suggests a promising way forward. The Hoeffding tree approach [7] 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 [13]. 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

[11] `http://dtai.cs.kuleuven.be/ACE/`

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 on on-line learning has resulted from the discovery of stochastic gradient descent (SGD) methods. 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 theincorporation of SGD 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 possibilites immediately. First, we have already mentioned the work of [20]: 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 [25]: 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 [33] did not look at its use in ILP-based feature evaluation.

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.

Bibliography

[1] C. Aggarwal. *Data Streams: Models and Algorithms.* Springer, New York, 2007.

[2] A. Bifet and R. Gavaldà. Learning from Time-Changing Data with Adaptive Windowing. In *Proc. of the Seventh SIAM Intl. Conference on Data Mining*, pages 443–448, 2007.

[3] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. MOA: Massive Online Analysis. *Jnl. Mach. Learn. Res.*, 11 (2010):1601–1604, 2010.

[4] A. Blum. Learning Boolean Functions in an Infinite Attribute Space. *Machine Learning*, 9:373–386, 1992.

[5] L. Bottou. Online Learning and Stochastic Approximations. In D. Saad, editor, *Online Learning in Neural Networks*, pages 9–42. Cambridge University Press, Cambridge, 1998.

[6] J. S. Bruner, J. J. Goodnow, and G. A. Austin. *A Study of Thinking*. John Wiley and Sons, New York, 1956.

[7] P. Domingos and G. Hulten. Mining High-Speed Data Streams. In *KDD2000: Proc. of the Sixth ACM SIGKDD Intl. Conference on Knowledge Discovery and Data Mining*, pages 71–80. ACM, 2000.

[8] T. Faruquie, A. Srinivasan, and R. King. Topic Models with Relational Features for Drug Design. In F. Riguzzi and F. Železný, editors, *Proc. of the 22nd Intl. Conference on Inductive Logic Programming*, number 7842 in LNAI, pages 45–57, Berlin, 2013. Springer.

[9] Y. Freund and R. Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[10] J. Gama. *Knowledge Discovery from Data Streams*. CRC Press, 2010.

[11] E.M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[12] D. Hand, F. Daly, A. Lunn, K. McConway, and E. Ostrowski. *A handbook of small data sets*. Chapman and Hall, 1994.

[13] G. Hulten, P. Domingos, and Y. Abe. Mining Massive Relational Databases. In *Proc. of IJCAI-2003 Workshop on Learning Statistical Models from Relational Data*, pages 53–60, 2003.

[14] S. Joshi, G. Ramakrishnan, and A. Srinivasan. Feature Construction Using Theory-Guided Sampling and Randomised Search. In F. Železný and N. Lavrač, editors, *Proc. of the 18th Intl. Conference on Inductive Logic Programming*, number 5194 in LNAI, pages 140–157, Berlin, 2008. Springer.

[15] I. Katakis, G. Tsoumakas, and I. Vlahavas. Dynamic feature space and incremental feature selection for the classification of textual data streams. In *Proc. of the ECML/PKDD-2006 Intl. Workshop on Knowledge Discovery from Data Streams*, pages 107–116, 2006.

[16] J. Kelly and S. Hamm. *Smart Machines: IBM's Watson and the Era of Cognitive Computing*. Columbia University Press, New York, 2013.

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

[18] J. Langford, L. Li, and T. Zhang. Sparse Online Learning via Truncated Gradient. *Journal of Machine Learning Research*, 10:777–801, 2009.

[19] N. Littlestone. Learning Quickly When Irrelevant Attributes Abound: A New Linear-Threshold Algorithm. *Machine Learning*, 2:285–318, 1988.

[20] C. Lopes and G. Zaverucha. HTILDE: Scaling Up Relational Decision Trees For Very Large Databases. In *Proc. 24th Annual ACM Symposium on Applied Computing (SAC 2009)*, pages 1475–1479. ACM, 2009.

[21] G. Menezes. 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, 2011.

[22] L. Michael and L. Valiant. A First Experimental Demonstration of Massive Knowledge Infusion. In *KR-08: Proc. Eleventh Intl. Conference on Principles of Knowledge Representation and Reasoning*, pages 378–388, 2008.

[23] R. S. Michalski. A theory and methodology of inductive learning. In R. Michalski, J. Carbonnel, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 83–134. Tioga, Palo Alto, CA, 1983.

[24] S. Muggleton and C. Feng. Efficient induction of logic programs. In S. Muggleton, editor, *Inductive Logic Programming*, pages 281–298. Academic Press, London, 1992.

[25] S. Muggleton, H. Lodhi, A. Amini, and M. Sternberg. Support Vector Inductive Logic Programming. In D. Holmes and L. Jain, editors, *Innovations in Machine Learning*, volume 194 of *Studies in Fuzziness and Soft Computing*, pages 113–135. Springer, Berlin, 2006.

[26] S. Muggleton and D. Michie. Machine Intelligibility and the Duality Principle. In H. Nwana and N. Azarmi, editors, *Software Agents and Soft Computing*, volume 1198 of *Lecture Notes in Computer Science*, pages 276–292. Springer, 1997.

[27] S. Muggleton, A. Srinivasan, and M. Bain. Compression, Significance and Accuracy. In D. Sleeman and P. Edwards, editors, *ML-92: Proc. of the Ninth Intl. Workshop on Machine Learning*, pages 338–347, San Mateo, CA, 1992. Morgan Kaufmann.

[28] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, 1990.

[29] C. Sammut. *Learning Concepts by Performing Experiments*. PhD thesis, Department of Computer Science, University of New South Wales, Sydney, Australia, 1981.

[30] B. Schölkopf and A. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA., 2002.

[31] E. Shapiro. An Algorithm that Infers Theories from Facts. In A. Drinan, editor, *IJCAI-81 :Proc. of the 3rd Intl. Joint Conference on Artificial Intelligence*, pages 446–451, Los Altos, CA., 1981. Morgan Kaufmann.

[32] S.Kramer, N. Lavrac, and P. Flach. Propositionalization approaches to relational data mining. In *Relational Data Mining*, pages 262–286. Springer-Verlag, New York, 2001.

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

[34] A. Srinivasan, R. King, S. Muggleton, and M. Sternberg. Carcinogenesis Predictions Using ILP. In N. Lavrac and S. Dzeroski, editors, *ILP-97: Proc. 7th Intl. Workshop on Inductive Logic Programming*, volume 1297 of *Lecture Notes in Computer Science*, pages 273–287. Springer, 1997.

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

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

[37] V. Vapnik. *Statistical Learning Theory.* Wiley, New York, NY., 1998.

A Experimental Details

A.1 Implementation

Both the infinite and fixed-vector approaches to on-line model construction can be implemented within the streaming mechanism provided by the Aleph ILP system. The principle technique employed by Aleph is to call a user-defined predicate *aleph_stream*/1 with each each element of the data stream. Here is an implementation of the infinite-stream setting using this predicate:

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

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

We can introduce a measure of control into the generation of new features. For example, we may only want to construct new features if we are fairly sure of a shift in concept. The Page-Hinkley test is one way to detect such a change [10], which can be incorporated as a guard on the induction of features. More generally, we envisage a set of constraints that have to be satified before initiating feature-discovery. The second clause for *process*/1 can be rewritten as:

```
process(Example):-
        focus(Example), !,  % constraints eg. Page-Hinkley test satisfied by the example
        induce(features),   % construct new features
        induce(model).      % update model
process(Example):-
        induce(model).      % only update the model (do not construct new features)
```

Assuming a fixed set of features have already been constructed beforehand, the fixed-attribute setting follows by removing the $induce(features)$ call in the definition. Here, for example, is a definition that allows a invoking an external model-constructor (like MOA) with a set of features already known to Aleph:

```
:- set(model_type,user).   % calls aleph_model/1 to construct model

aleph_stream(Example):-
   process(Example).

aleph_model(Model):-
    [statements to invoke external model constructor]

process(end_of_file):-
    !,
    induce(model).          % construct model, here by calling aleph_model/1
process(Example):-
   correctly_predicted(Example), !.
process(Example):-
        write_feature_vector(Example). % write feature-representation of instance
```

Using mechanisms within Aleph for customised model- and feature-construction, we are able to specify several different ways of stream-processing in a (reasonably) declarative manner. It is possible, for example, to perform sliding-window computations over dependent instances in the following manner:

```
:- set(model_type,user).   % calls aleph_model/1 to construct model

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)

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

A.2 Method

The following additional details are relevant for Experiments 1 and 2 (Time and Space):

1. For experiments here, the number of repetitions R is 10.
2. For simple targets, the number of features K is chosen randomly from the range 1 to 4. For complex concepts, the number of features K is randomly chosen from the range 8 to 12. K features are then randomly constructed using the ILP engine, and their disjunction constitute the target concept.
3. The ILP engine allows either a random or systematic strategy for generating features [14]. We use a random strategy. The ILP engine also allows the specification of the kind of features (see [14]). We specify this to be the *definite* category, which is most general possible.
4. 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 combination of features is correspondingly equal to the number of features allowed for each data instance (that is, 25).
5. 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.
6. 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 R repetitions. 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 [10]). Thus, for each assignment of values to $(Target, Data)$, we will denote performance by the pair (A, T), denoting 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".

The following supplementary details are relevant to Experiment 4 (Real Data):

1. 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.
2. It has been a common practice in similar ILP applications (for example, the mutagenesis or carcinogenesis datasets: [35, 34]) 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 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.
3. For simplicity, in all cases, 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.

4. 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.

A.3 Results

The remainder of this appendix includes additional results from the empirical evaluation.

Runtimes. In Figure A.1 we give runtimes on the controlled experiments 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.

Additional features. In Figure A.2 we tabulate the increase in features that occurs in the infinite-attribute setting as Winnow learns to adjust to the drift in the target concept, for both simple and complex targets under three sizes of drift.

Small streams. In Figure A.3 we provide the results for the experiments discussed above on small streams.

Predicted and actual mistakes. In Figure A.4 we compare the theoretical mistake bounds for Winnow with the actual results obtained.

Model	Data	Accuracy (%)	Time (s)	Normalised Time
Fixed Hoeffding	125K	100.0	10.9(2.9)	1.0
	250K	100.0	13.3(3.0)	1.2
	500K	100.0	18.1(3.9)	1.7
	1M	100.0	27.7(5.5)	2.5
Fixed Winnow	125K	100.0(0.0)	22.3(3.9)	1.0
	250K	100.0(0.0)	36.2(6.0)	1.6
	500K	100.0(0.0)	64.7(10.5)	2.9
	1M	100.0(0.0)	119.9(21.1)	5.4
Infinite Winnow	125K	100.0(0.0)	28.9(18.9)	1.0
	250K	100.0(0.0)	56.0(36.4)	1.9
	500K	100.0(0.0)	110.4(74.0)	3.8
	1M	100.0(0.0)	212.7(135.8)	7.2

(a) Simple targets

Model	Data	Accuracy (%)	Time (s)	Normalised Time
Fixed Hoeffding	125K	99.8(0.8)	14.5(5.5)	1.0
	250K	99.8(0.8)	19.6(4.6)	1.4
	500K	99.8(0.8)	29.9(5.5)	2.1
	1M	99.8(0.8)	50.1(10.4)	3.5
Fixed Winnow	125K	99.8(0.8)	34.6(5.5)	1.0
	250K	100.0(0.0)	59.9(11.2)	1.9
	500K	100.0(0.0)	109.4(24.2)	3.1
	1M	100.0(0.0)	208.6(49.9)	6.0
Infinite Winnow	125K	100.0(0.0)	52.3(8.8)	1.0
	250K	100.0(0.0)	107.1(12.1)	2.1
	500K	100.0(0.0)	228.2(28.1)	4.4
	1M	100.0(0.0)	424.8(77.4)	8.1

(b) Complex targets

Figure A.1: Results on synthetic data of using on-line model construction with ILP-constructed features. Models employ either the fixed- or infinite-attribute setting. In the former, an ILP system constructs a fixed number of features from a pre-classified sample of data instances. In the latter, features are constructed "on-demand" as and when instances are misclassified with the existing set of features. An entry N in the Data column refers to the dataset Trains_N, as listed in the paper. 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. The numbers tabulated are average values obtained from 10 repetitions. Standard deviations are in braces.

Shift	Feature Increase
Small	25(21)
Large	48(22)
Extreme	72(16)

(a) Simple targets

Shift	Feature Increase
Small	34(24)
Large	54(17)
Extreme	61(35)

(a) Complex targets

Figure A.2: Mean increase in the number of features with shift in concepts. The results are for models in the infinite-attribute setting. The quantity in brackets is the standard deviation. The mean is affected by some extreme values, and the changes are better represented by the median. Here the median values are: 29, 50, 66 (simple targets); and 38, 51, 75 (complex targets).

Model	Data	Acc. (%)	Time (s)
	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_1K	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_1K	99.9(0.1)	0.40(0.3)

(a) Simple targets

Strategy	Data	Acc. (%)	Time (s)
	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_1K	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_1K	99.2(1.0)	1.0(0.4)

(b) Complex targets

Figure A.3: Results with small data streams. Here the suffixes "125, 250, 500, 1K" stand for 125, 250, 500 and 1000 data instances respectively. As before, accuracies are on an independent test set, and standard deviations are in braces.

Trial (R)	Pred.	Act.
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

(a) Simple targets

Trial (R)	Pred.	Act.
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

(b) Complex targets

Figure A.4: Comparison of the mistake-bound predicted by theoretical analysis for Winnow and the actual mistakes made by the ILP-Winnow engine. 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.