Machine Learning Changing Concepts

Thesis submitted for the transfer to full DPhil status

Rupert Parson
Oxford University Computing Laboratory
The Wolfson Building
Parks Road
Oxford

October 12, 1998

Abstract

Machine Learning is concerned with the induction of rules or programs from a set of examples. In some cases the target to be learnt changes during the production of the examples. For instance, it is possible to learn rules about users' interests from the content of Web pages that they browse. However, users' interests may change over time, and a good learning algorithm should deal with these changes appropriately. In this thesis, a survey is made of the available literature on Concept Drift, and the related literature on Theory Revision. Work described in the thesis gives a theoretical model of the problem, based on probability arguments. A simple Markov chain is used to establish the probability that an old example is still valid after a period of time. A Bayesian argument leads to a formula for the probability that an hypothesis is the target concept, given a set of examples. A simple experiment in learning a changing concept is described, using the same artificial source of data as a published experiment. A description is made of work conducted at British Telecom as part of the EPSRC CASE award. This work looked into the learning of user interests from a log of the web pages that they browsed. It is intended that this work be extended to learn the changes in the user's interests. The thesis concludes by suggesting suitable approaches for future work; theory revision can be used to track a drifting concept incrementally, and also to form a temporal theory recording the drift of the concept.

Contents

Chapter 1

Introduction

1.1 Machine Learning

Machine learning is the study of algorithms that induce rules or programs from a set of example facts. Valiant [23] describes learning as the acquisition of a program or rules for a particular task in the absence of explicit programming. In a domain of discourse, the objects in that domain may be partitioned into different subsets, or classes of the domain. Machine learning attempts to find rules that define these classes, based on a set of examples from the domain. It is typical to deal with binary classifications, into objects labelled positive and negative. An *example* is defined to be an instance of the domain together with its classification. In this case, the set of all positively classified instances in the domain is called the *target concept* (Shapiro called this set the *intended interpretation*). In our setting, the target concept will be taken from the *target domain*, a set of subsets of the instance domain.

In most settings, the aim is to find a theory in some language L that expresses the target concept. For instance, in an Inductive Logic Programming (ILP) setting [17], the aim is to induce a theory in Horn clause logic (Prolog), which has a Herbrand model that is (or closely matches) the target concept. The target concept is regarded as being fixed. The language L should be chosen so that for any intended interpretation, it is expressive enough to form a theory that models that interpretation on the domain.

A general machine learning model is as follows; a target concept is selected from the target domain. Instances are then chosen from the instance domain, labelled appropriately according to the target concept, and are then presented to a learning algorithm. The algorithm's task is to produce a theory in the language L that explains the examples given, by having a model that matches the target concept closely.

The general requirement for a learning algorithm to be considered useful is that it should satisfy a form of convergence. Gold [?] required that the learner should eventually identify the target concept. That is, it should, after a finite number of examples, give a hypothesis that is the target concept, and its output should not change if more examples are given. This he called *identification in the limit*. Valiant [23] gave the looser requirement that, given enough examples, the learner will output an hypothesis that is as close as required to the target concept. This later became known *PAC (provably approximately correct) learnability*.

1.2 Drifting Concepts – Motivation

In the field of machine learning, only preliminary investigation has been made into the problems of learning target concepts that change with time. This is despite the fact that such problems are encountered in real world applications. For instance, the main impetus for investigating this topic came from the author's MSc thesis [**?**], in which the learning of user interests from their classifications of World Wide Web (WWW) pages was investigated.

It became apparent that the classifications that users gave to pages (interesting or not interesting) were not the same each time they made them. It is clear that this should be the case. People do discover new areas of interest, particularly when browsing through the web.

The changes of interest may be driven by a number of factors. For instance, consider "window shopping" on the WWW, in which, for a while, a user will be interested in Web pages that contain information on a type of product that he is intending to buy. After he has made a purchase, the user will probably no longer be interested in price or general information for that type of product.

Other examples include face recognition - a person's face changes as he ages; factory systems wear down through use; weather structures at a particular instant can be determined from distributed weather samples, but will change with time; price relationships fluctuate on the stock exchange.

1.3 Outline of the Thesis

In chapter 2 a background literature survey will be addressed, comparing the different approaches that have been taken for the relevant problems. Work that has been conducted in the areas of Concept Drift and Theory Revision will be described, and their inter-relationship and relevance will be discussed. Although little work by the author to extend Theory Revision is described in this thesis, the potential for its use in tracking drifting concepts is recognised, and future work will make use of it.

Preliminary theoretical investigations are described in chapter 3. A basic model for learning drifting concepts is set up, and a simple example weighting mechanism is described. A general Bayesian framework for probability-based machine learning is constructed, and this is extended to deal with drifting concepts.

An experiment in learning a changing concept is described in chapter 4. The experiment attempts to make a preliminary application of ILP to the problem. The (artificial) data source used is the same as that used by Widmer and Kubat [27] in their paper on Concept Drift.

Chapter 5 describes the work undertaken at British Telecom (BT), as part of the CASE industrial sponsorship award. The work involved applying learning techniques to a dataset supplied by BT, a log of users accesses to the WWW.

A timetable is suggested, along with a research proposal for the DPhil work in chapter 6. I shall describe the areas that I feel merit detailed investigation, and describe a timetable for doing so.

Chapter 2

Concept Drift and Theory Revision

2.1 Concept Drift

In this section a review will be made of work that has been carried out in the field of learning drifting concepts.

2.1.1 Introduction

Helmbold and Long [9], Kuh et al. [10], and Bartlett et al. [3] have made investigations into the conditions under which effective *drift tracking* is possible.

In laying the foundations for future work, their investigations gave definitions of the size of a change of concept and the speed of a drifting concept, and, by extending previous machine learning results for learning static concepts, they proved results that gave theoretical limits on the maximum tolerable speed of drift. The next section contains a more detailed description of their results.

Widmer and Kubat [27] extended these theoretical results by conducting some preliminary experiments in learning a concept that changed infrequently. They took as their concepts and learning domain the *STAGGER* concepts first used by Schlimmer and Granger [20].

Widmer and Kubat's algorithm used a window on the set of examples of a size that was varied using a simple heuristic. The heuristic was based on variables such as the current predictive accuracy of the theory generated, the size of the theory, and a few user set parameters. Varying the window size in this way allowed the algorithm to "forget" irrelevant examples when the concept changed, only seeing the last (more relevant) m examples.

2.1.2 Definitions

All the papers adopt the same definition for the size of a change of concept, based on the measure of concept error in Blumer et al.[5]. There is assumed to be a probability distribution D_X over the instance domain. The size of change between two concepts is defined to be the probability according to D_X of an instance picked at random falling in the symmetric difference of the two concepts.

Widmer and Kubat also define an interesting concept of speed of drift. They imagine that the target concept changes from one target T_1 to another T_2 as Δx examples are presented. At the start of the change, the probability that an example is classified according to T_1 is 1. At the end of the change, Δx examples later, the probability that an example is classified according T_2 is 1. The probabilities vary linearly in between, always with the probability that an example is classified according to concept T_1 equaling 1 minus the probability that the example is classified according to concept T_2. Δx is then a measure of the speed of the change. Clearly, the larger Δx, the slower the change. They do not attempt to combine the two ideas of speed and size.

2.1.3 Results

Kuh, Petsche and Rivest hoped to extend Computational Learning Theory [23] by applying a PAC-like model to learning drifting concepts. They investigated two different types of *tracking* problem, batch learning and incremental learning. In batch learning, a set of time–stamped examples, produced from a drifting concept, are presented to a learner as a set. The algorithm tries to learn the last state of the concept. In incremental learning, examples are presented to the learner one by one, and the algorithm must produce a concept that is as close as possible to the present target concept at each stage (or must correctly predict the classification of the next example). They establish a sample complexity bound and a bound on the maximum tolerable rate of drift for the case of batch learning, using the Vapnik–Chervonenkis [5] dimension of the concept class. They establish bounds on the average error rate for incremental tracking.

Helmbold and Long provide even tighter bounds for the case of incremental tracking, also using the VC–dimension, using a method that they call "minimizing disagreements". They show that if the target class is properly PAC-learnable, then there is an efficient algorithm that finds, with high probability, an hypothesis that approximately minimises the number of disagreements (on the given examples) with the target concept. They also give bounding results that are specific to particular target classes, such as halfspaces and hyperrectangles.

Widmer and Kubat's paper describes a class of varying concept learning algorithms, each enhanced to deal with the particular problems that may be encountered, such as noise, and, more interestingly, cyclic or repetitive patterns in the changing concept.

As of yet, no attempt has been made to explicitly implement a drift tracking algorithm in an Inductive Logic Programming setting.

2.2 Theory Revision

2.2.1 Introduction

A Concept Drift problem can be tackled in two ways, corresponding to different applications.

- A non-incremental, or *batch* learning algorithm receives a single finite set of examples. The examples are time-stamped to show when they were classified, and they are classified according to the state of the target concept at that time. The algorithm must generalise the examples to find an hypothesis to closely match the state of the target concept at the time of the last example's classification. The effectiveness of the algorithm is measured in terms of the size of drift that it can cope with, a bound on the error of the hypothesis for that drift, and the number of examples required to get within that bound (the sample complexity).

- An incremental learning, or *tracking* algorithm receives examples one at a time. Each example is classified according to the state of the target concept at that instant. After each example has been given, the algorithm should form an hypothesis that closely matches the present state of the target concept. Alternatively, the algorithm must try to predict the classification of the next example. The effectiveness of the algorithm can be measured in terms of the number of examples required by the algorithm to produce an hypothesis within a certain error bound, and what rate of drift the algorithm can tolerate to remain within that bound. One can measure its success (or lack of it) in terms of the *error rate* - the frequency of incorrect classification predictions for the next example.

When trying to learn a user's WWW interests from a cache log of his browsing, one is performing batch learning to learn the rules to predict interest. The set of timestamped examples of his past changing interests is taken and used to produce rules that predict as closely as possible the user's present interests.

When trying to *follow* a user's WWW interests, one must be prepared to update the rules one has derived when new data arrives. This is an incremental form of Theory Revision. The incremental approach is the appropriate one when trying to keep track of a user's interests, and is also the approach that has been the most widely researched [9, 10, 11, 27].

Theory Revision is an area of Machine Learning that is relevant to tracking changing target concepts. Theory Revision systems act incrementally. The approach assumes that a theory that approximates the target concept has already been devised, but that it may have some errors. The presentation of a new example that is incorrectly classified by the present theory forces a *revision*. Generally the change is required to be as small as possible. The present theory may need to be generalised or specialised in order to accommodate the new example. Some of the algorithms assume the introduction of a set of examples, rather than just a single example, at each revision stage.

The different approaches to Theory Revision vary with their representation languages, their generalisation and specialisation operators, and their dependency on some form of oracle or interaction with a user.

2.2.2 A Generic Theory Revision Model

A generic model of Theory Revision, that all the algorithms follow in some instantiated manner, is now given.

Given a theory T, expressed in some language $\mathcal{L}$, and given a new set of examples X, find a new theory T' such that $T' \models X$, and T' is "close" to the original T. Examples are considered one at a time in some order from X, $x \in X$. If $T \models x$, then we are done, otherwise either x is labelled positive and is not covered by T, or x is labelled negative and is covered by T. Each case is handled separately by a different operator, of generalisation and specialisation respectively.

Each aspect of the Theory Revision algorithm shall be considered in turn, and comparisons made between the approaches of three particular implementations, RUTH [1, 2], AUDREY [28] and FORTE [19].

The language $\mathcal{L}$ is generally that of universally quantified Horn-clauses, specifically Prolog (FORTE and AUDREY), and Datalog (RUTH) (a functor-free version of Prolog).

For all three algorithms, the examples are ground atoms, labelled according to their correct classification. FORTE also allows the inclusion of new background knowledge with each example. The introduction of background knowledge in this way enhances the use of inverse resolution operators in the generalisation stage. RUTH is designed to fit a theory to a set of integrity constraints (clauses of the form $p_1 \wedge \ldots \wedge p_k \rightarrow q_1 \wedge \ldots \wedge q_m, k \geq 0, m \geq 0$). Contradicted examples in the form of ground literals are hypothesised from violated constraints, and these examples are then used in the specialisation or generalisation of the database. Other examples are hypothesised as a result of the refinement operations, and these are dealt with in turn. This makes RUTH a much more powerful Theory Revision system than the other two.

2.2.3 Generalisation and Specialisation Operators

The generalisation of a theory is needed when a positive example is not presently covered by the theory. To generalise the theory, one can do either of the following.

- Add a clause to the theory.
- Delete an atom from the body of a clause in the theory.

All the generalisation operators described in the literature are special cases of these two operations.

Specialisation is used when a negative example is incorrectly covered by the present theory. To specialise a theory one can do either of the following:

- Delete a clause from the theory.
- Add an atom to the body of a clause in the theory.

The actual algorithms all use variants of the above operations. They vary with the different criteria that they use for the selection of affected clauses, and the generalisation systems used. For instance, RUTH uses the generalisation procedure of Shapiro's MIS system [21, 22], FORTE uses inverse resolution operators based on Muggleton's CIGOL system [16], and AUDREY uses a combination of abduction and Vere's maximally specific generalisation (msg) operator [24, 25].

FORTE uses a FOIL-based operator to add atoms to the body of clauses in the specialisation procedure. The other two algorithms do not have this facility. To analyse when a clause must be deleted, RUTH uses a variant of Shapiro's debugging specialization procedure. To avoid the use of an oracle, RUTH hypothesises that one of the clauses in the theory, that leads to the incorrect proof of a covered negative example, is incorrect. New literals, derived from this clause, are then added as new examples. An intelligent search strategy is used to select the incorrect clause.

This technique, whilst making RUTH more powerful, does make it less efficient. FORTE and AUDREY are based on heuristics, and are therefore more efficient.

Note that, unlike FORTE and AUDREY, RUTH guarantees completeness (proven in the paper), in that the algorithm is guaranteed to find a solution in finite time if there is one. FORTE and AUDREY are vulnerable to local maxima.

2.3 Extending The Work

The next two chapters will describe the theoretical and practical work that extends the content of the papers described above. In chapter 3 an approach strongly based on Bayesian probability will be investigated, leading to formulae for estimating the probability that an hypothesis is the target concept, given the set of examples seen so far.

Although it would be prohibitively inefficient to use this estimate as a direct means of picking the best hypothesis for a set of examples, it would be a suitable heuristic measure for guiding the search of an algorithm, such as the ILP algorithm Progol.

The work of Widmer and Kubat is extended in chapter 4. The machine learning algorithm Progol is used generate rules that not only predict the classification of the next example, but also summarize the "history" of past examples.

Chapter 3

Theoretical Results

3.1 A Markov Chain

When Learning from a target that is drifting in some well defined manner, it is possible to assign a confidence weighting to examples, that decreases the further into the past that the example was produced.

However, to establish this weighting, first the size of a concept change must be defined, from which we can infer the rate of drift. The definition adopted seems to be standard in the Concept Drift literature — the measure of a size of change is the same as the error between the two concepts, as defined in the paper on Learnability and the Vapnik-Chervonenkis dimension by Blumer et al [5].

We are given a domain of discourse, or instance domain, a set X, a distribution D_X over the domain, the concept class $\mathcal{H} \subseteq \mathcal{P}(X)$, and a distribution $D_{\mathcal{H}}$ over it.

Define, for two subsets $C_1, C_2 \subseteq X$, $\text{error}(C_1, C_2) = D_X(C_1 \nabla C_2)$, where the *symmetric difference* $C_1 \nabla C_2 = (C_1 \setminus C_2) \cup (C_2 \setminus C_1)$.

The concepts to be learned are members of the concept class $\mathcal{H}$. The initial target concept T_1 is drawn at random from the concept class according to the fixed distribution $D_{\mathcal{H}}$. A Target sequence $T_1, T_2, \ldots T_n, \ldots$ is called $\lambda - \textit{admissible}$ if, for all $i = 1 \ldots$, $T_i \in \mathcal{H}$ and $\text{error}(T_i, T_{i+1}) \leq \lambda$ ie, the probability that an instance x picked at random according to D_X is classified differently by two consecutive targets is less than or equal to λ. The change from target T_i to target T_{i+1} shall be called a *target transition*.

At each time unit i, an instance is drawn randomly and independently from the domain X according to the distribution D_X. This instance is then labelled according to whether it is in T_i: 0 for not in the target, and 1 for in the target set. The example given to the learner will therefore be a pair $(x_i, c_i), x_i \in X, c_i \in \{0, 1\}$.

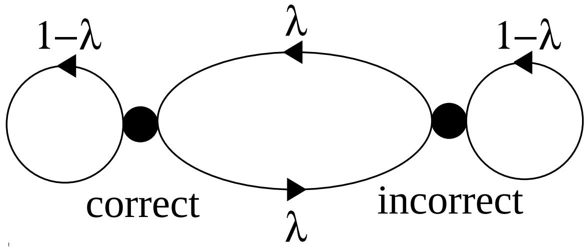

Figure 3.1: Simple Markov Chain for Constantly changing Target

3.1.1 A Measure of Confidence

For each example given, a confidence weighting for the validity of the example can be calcuated, as a function of how far in the past it was labelled, and the bound on the speed of drift, λ. In other words, it is possible to calculate the probability that the labelling of the instance is still correct for the current target concept.

For each instance, the situation can be represented by a Markov chain, with two states, (correctly and incorrectly labelled), and with probability λ of passing from one state to the other (see figure 3.1). It is assumed, for the moment, that the change is exactly λ for each target transition.

This simple Markov chain has transition matrix:

$$A = \begin{pmatrix} 1-\lambda & \lambda \\ \lambda & 1-\lambda \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & 1-2\lambda \end{pmatrix}\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$$

The initial state is $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$; (the labelling is definitely correct).

After i transitions the state is:

$$\begin{aligned} A^i\begin{pmatrix} 1 \\ 0 \end{pmatrix} &= \frac{1}{2}\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & (1-2\lambda)^i \end{pmatrix}\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ &= \frac{1}{2}\begin{pmatrix} 1+(1-2\lambda)^i \\ 1-(1-2\lambda)^i \end{pmatrix} \end{aligned}$$

Therefore Prob((x, c) correct after i target transitions) $= \frac{1+(1-2\lambda)^i}{2}$.
(Note: this is the correct value if the size of change is *exactly* λ for each presentation of an example. When the size of change is bounded by (less than or equal to) λ, a very similar calculation gives
Prob((x, c) correct after i target transitions) $\geq \frac{1+(1-2\lambda)^i}{2}$.)

Therefore, using a simple information measure, the *confidence* in a labelled example is defined to be, after i transitions, $\mathrm{conf}_\lambda(i) = 1 + \log_2(\frac{1+(1-2\lambda)^i}{2})$. This takes the value 1 for $i = 0$, and tends to 0 as $i \to \infty$.

3.2 A Bayesian Model of Learning

A model of machine learning Concept Drift, in which the speed of drift is measured probabalistically, lends itself to a simple Bayesian analysis. In the next few sections, a framework is built up, with the aim of allowing reasoning about drifting concepts. For more detailed investigations into Bayesian learning, see [6, 7, 13, 8].

3.2.1 The Model

Let the instance domain be some non-empty set X. Let the *target class* $\mathcal{H}$ be a set of subsets of X. Let C, the class value set, be $\{0, 1\}$. Let $D_{\mathcal{H},X,C}$ be a probability distribution over the target class, domain of instances, and class value set. The distribution is such that the selection of a target from a target class, and an instance from the instance domain are independent. However, the class values will of course be dependent on both the target and instance chosen. An example will be a pair, consisting of an instance, and an associated class value. The random variables x_i, c_i will represent the examples.
The general Bayesian model of batch learning is as follows.

An (initial) target hypothesis T is chosen according to the marginal distribution over the target class $D_{\mathcal{H}}$.

The random variables $\{x_i : i = 1 \ldots m\}$ take the values $\{a_i : i = 1 \ldots m\}$, chosen according to the distribution over the instance domain. These are then classified by choosing each c_i to be some value $b_i \in \{0, 1\}$ according to the marginal distribution $D_{C|T,X=a_i}$. This gives an m-sample — an m-tuple of examples. This event of obtaining this example set is denoted $E = \bigwedge_{i=1}^m (x_i = a_i \wedge c_i = b_i)$.
The aim is to find the most suitable hypothesis H for the example set E.
This can be achieved by choosing an hypothesis that maximises the posterior probability $P(Target = H|E)$ over the hypothesis space. We will be using Bayes' theorem, that for any random variables A, B, $Pr(A|B)Pr(B) = Pr(B|A)Pr(A)$, and the following lemma.

Lemma 3.1 *For any random variables $A_1, A_2, \ldots A_m, B$,*

$$Pr(\bigwedge_{i=1}^m A_i|B) = \prod_{i=1}^m Pr(A_i|B \wedge \bigwedge_{j=i+1}^m A_i).$$

Proof:

$$Pr(\bigwedge_{i=1}^m A_i|B) \quad = \quad \frac{Pr(A_1 \wedge A_2 \wedge \ldots A_m \wedge B)}{Pr(B)}$$

$$
\begin{aligned}
&= Pr(A_1|A_2 \wedge \ldots A_m)\frac{Pr(A_2 \wedge \ldots A_m \wedge B)}{Pr(B)} \\
&= \prod_{i=1}^{m} Pr(A_i|B \wedge \bigwedge_{j=i+1}^{m} A_i)
\end{aligned}
$$

by repeating m times, and assuming that $\bigwedge_{j=m+1}^{m} A_i = \Box$.
$\Box$.

The calculation of $Pr(T = H|E)$ proceeds as follows.

$$
\begin{aligned}
& Pr(T = H|E) \\
= \;& Pr(T = H| \bigwedge_{i=1}^{m} (x_i = a_i \wedge c_i = b_i)) \\
= \;& \frac{Pr(T = H \wedge \bigwedge_{i=1}^{m} (x_i = a_i \wedge c_i = b_i))}{Pr(\bigwedge_{i=1}^{m} (x_i = a_i \wedge c_i = b_i))} \\
= \;& Pr(\bigwedge_{i=1}^{m} c_i = b_i|T = H \wedge \bigwedge_{i=1}^{m} x_i = a_i)\frac{Pr(T = H \wedge \bigwedge_{i=1}^{m} x_i = a_i)}{Pr(\bigwedge_{i=1}^{m} (x_i = a_i \wedge c_i = b_i))} \\
= \;& Pr(\bigwedge_{i=1}^{m} c_i = b_i|T = H \wedge \bigwedge_{i=1}^{m} x_i = a_i)Pr(T = H)\frac{Pr(\bigwedge_{i=1}^{m} x_i = a_i|T = H)}{Pr(\bigwedge_{i=1}^{m} (x_i = a_i \wedge c_i = b_i))}
\end{aligned}
$$

Notice that the fraction on the right is constant for all H. We will denote it k_1.

$$
\begin{aligned}
= \;& k_1 Pr(T = H)Pr(\bigwedge_{i=1}^{m} c_i = b_i|T = H \wedge \bigwedge_{i=1}^{m} x_i = a_i). \\
= \;& k_1 Pr(T = H) \prod_{i=1}^{m} Pr(c_i = b_i| \bigwedge_{j=i+1}^{m} c_j = b_j \wedge T = H \wedge \bigwedge_{i=1}^{m} x_i = a_i)
\end{aligned}
$$

using the lemma.
In the following sections this equation is instantiated to various cases of machine learning.

3.2.2 Positive and Negative Data

Consider the purest case, in which instances are chosen from the entire domain, and every instance is then classified correctly according to the target concept selected. In other words, the marginal distribution over the set of classifications, given the target and the instance, is such that it biases random choice only to the correct classification of the instance.

It is then true that $Pr(c_i = b_i|T = H \wedge \bigwedge_{j=1}^{m} x_j = a_j \wedge \bigwedge_{j=i+1}^{m} c_j = b_j) = Pr(c_i = b_i|T = H \wedge x_i = a_i)$, since the i^{th} class value is dependent only on the i^{th} instance chosen and the target hypothesis that classifies it.

Hence,

$$
\begin{aligned}
Pr(T=H|E) &= k_1 Pr(T=H)\prod_{i=1}^{m} Pr(c_i=b_i|T=H\wedge x_i=a_i)\\
&= k_1 Pr(T=H)\prod_{i=1}^{m}\left\{\begin{array}{ll}\left\{\begin{array}{ll}0 & \text{if } b_i=0\\ 1 & \text{if } b_i=1\end{array}\right\} & \text{if } a_i\in H\\ \left\{\begin{array}{ll}1 & \text{if } b_i=0\\ 0 & \text{if } b_i=1\end{array}\right\} & \text{if } a_i\notin H\end{array}\right.\\
&= k_1 Pr(T=H) consistent(H,E)
\end{aligned}
$$

where $consistent(H,E)=\left\{\begin{array}{l}1, \text{if H is consistent with the example set E}\\ 0, \text{otherwise}\end{array}\right.$

Thus, any good hypothesis must be consistent. Take logs, of both sides. An Occam's Razor assumption can also be taken – that the most likely theory is the simplest (or smallest) theory using the size measure that $size(H) = -\log(Pr(T=H))$, where T is the target. This gives us:

$$\log(Pr(T=H|E)) = k_2 - size(H)$$

Thus the problem is reduced to finding the smallest or simplest *consistent* hypothesis.

3.2.3 Noisy Positive and Negative Data

Now learning from a noisy example source. Noise is modelled by supposing that the class values have been transmitted down a *binary symmetric noisy channel*, with noise coefficient (probability of a noisy transmission) λ for both types of error (positive transmitted as negative, and negative transmitted as positive). It is simple to extend to a non-symmetric channel. (For a simple look at communication theory see Dominic Welsh's book [26]).

It is then true that $Pr(c_i=b_i|T=H\wedge\bigwedge_{j=1}^{m}x_j=a_j\wedge\bigwedge_{j=i+1}^{m}c_j=b_j) = Pr(c_i=b_i|T=H\wedge x_i=a_i)$, since the probability of the i^{th} class value taking a particular value is dependent only on the i^{th} instance chosen and the target hypothesis that classifies it.

Hence,

$$
\begin{aligned}
Pr(T=H|E) &= k_1 Pr(T=H)\prod_{i=1}^{m} Pr(c_i=b_i|T=H\wedge x_i=a_i)\\
&= k_1 Pr(T=H)\prod_{i=1}^{m}\left\{\begin{array}{ll}\left\{\begin{array}{ll}\lambda & \text{if } b_i=0\\ (1-\lambda) & \text{if } b_i=1\end{array}\right\} & \text{if } a_i\in H\\ \left\{\begin{array}{ll}(1-\lambda) & \text{if } b_i=0\\ \lambda & \text{if } b_i=1\end{array}\right\} & \text{if } a_i\notin H\end{array}\right.\\
&= k_1 Pr(T=H)\lambda^d(1-\lambda)^{m-d}
\end{aligned}
$$

where $d = |\{a_i | a_i \in H \not\Leftrightarrow b_i = 1\}|$, is the number of examples which are not consistent with H.

Suppose now, however, that the value of λ is unknown. It is possible to estimate its value from the data, by assuming that for every example, the probablity of an error is λ exactly. This means that the expected number of errors in m examples is $m\lambda$.

The examples are being classified by the target T, with $T = H$. Let d be defined as above, (the number of examples not consistent with H). d can then be used to estimate a value for λ. Using Laplace correction, this gives: $\lambda = \frac{d+1}{m+2}$.

This then implies that:

$$Pr(T = H|E) = k_1 Pr(T = H) \left(\frac{d+1}{m+2}\right)^d \left(\frac{m+1-d}{m+2}\right)^{m-d},$$
$$\text{where, } d = |\{a_i | a_i \in H \not\Leftrightarrow b_i = 1\}|$$

Therefore:

$$\log(Pr(T = H|E)) = k_2 - size(H) + d \log\left(\frac{d+1}{m+2}\right) + (m-d)\log\left(\frac{m+1-d}{m+2}\right),$$
$$\text{where, } d = |\{a_i | a_i \in H \not\Leftrightarrow b_i = 1\}|$$

3.2.4 Drifting Data

For drifting data, it is not possible to make the same independence assumptions that were made before. It is necessary to calculate $Pr(c_i = b_i | T_m = H \wedge \bigwedge_{j=1}^m x_j = a_j \wedge \bigwedge_{j=i+1}^m c_j = b_j)$ explicitly, for each i.

First, it is clear that

$$\begin{aligned}
\mathcal{P} &= Pr(c_i = b_i | T_m = H \wedge \bigwedge_{j=1}^{m} x_j = a_j \wedge \bigwedge_{j=i+1}^{m} c_j = b_j) \\
&= Pr(c_i = b_i | T_m = H \wedge \bigwedge_{j=i}^{m} x_j = a_j \wedge \bigwedge_{j=i+1}^{m} c_j = b_j) \\
&= Pr(c_i = b_i | T_m = H \wedge x_i = a_i \wedge \bigwedge_{j=i+1}^{m} (x_j = a_j, c_j = b_j))
\end{aligned}$$

Notice that it is only necessary to consider the relevant examples in the conditional part – the examples a_j for which $a_j = a_i$.

$$\begin{aligned}
\mathcal{P} &= Pr(c_i = b_i | T_m = H \wedge x_i = a_i \wedge \bigwedge_{j \in \mathcal{J}_i} (x_j = a_j, c_j = b_j)) \\
&\quad \text{where } \mathcal{J}_i = \{j | a_j = a_i, j = i+1 \ldots m\}.
\end{aligned}$$

A pseudo-example $x_m = a_i$ is included, classified according to H to give c_m. (The real value and classification of x_m has not yet been given).

The probability that c_i takes a particular value depends only on the classification of a_i the next time it appears in the sequence of examples. Define $next(i)$ to be the index j of the next occurrence of a_i. ie $next(i) = \min\{j | j > r, a_j = a_i\}$. Notice that since the pseudo-example has been defined $(a_m = a_i)$, $next(i)$ is defined for all indices $1 \leq i \leq m-1$.

$$
\begin{aligned}
\mathcal{P} &= Pr(c_i = b_i | T_m = H \wedge x_i = a_i \wedge \bigwedge_{j \in \mathcal{J}_i} (x_j = a_j, c_j = b_j)) \\
&= Pr(c_i = b_i | T_m = H \wedge x_i = a_i \wedge (x_{next(i)} = a_i, c_{next(i)} = b_{next(i)})) \\
&= \begin{cases} \frac{1}{2}(1 + (1-2\lambda)^{next(i)-i}), & \text{if } c_i = c_{next_i} \\ \frac{1}{2}(1 - (1-2\lambda)^{next(i)-i}), & \text{if } c_i \neq c_{next_i} \end{cases}
\end{aligned}
$$

This leads to the result:

$$
\begin{aligned}
Pr(T_m = H | E) &= k_1 Pr(T_m = H) \prod_{i=1}^{m} Pr(c_i = b_i | T_m = H \wedge \bigwedge_{j=1}^{m} x_j = a_j \wedge \bigwedge_{j=i+1}^{m} c_j = b_j) \\
&= k_1 Pr(T_m = H) \prod_{i=1}^{m} f_\lambda(i) \\
&\quad \text{where } f_\lambda(i) = \begin{cases} \frac{1}{2}(1 + (1-2\lambda)^{next(i)-i}), & \text{if } c_i = c_{next(i)} \\ \frac{1}{2}(1 - (1-2\lambda)^{next(i)-i}), & \text{if } c_i \neq c_{next(i)} \end{cases} \\
&\quad \text{where } next_i = \min\{j | j > i, a_j = a_i\}.
\end{aligned}
$$

3.2.5 Problems With Estimating $Pr(T_m = H)$

Our learnability setting allows us to assume a prior distribution on the selection of targets by the learner.

However, in the drifting target concept model, we would have trouble estimating the probability $Pr(T_m = H)$. Presumably, we could assume that the first target, T_1, was selected according to the prior distribution over target concepts. However, with each successive target, which would be offset from the previous target by a certain amount of change, estimating the probability would become a much more complex problem.

To avoid this problem, we can introduce a new assumption at the start. We can assume a prior probability distribution on all possible concept sequences (restricted by the maximum size of drift allowed).

In this way, we can assume the algorithm will not be trying to find a suitable hypothesis concept at each stage, but will be trying to hypothesise a suitable concept sequence at each stage. The next section explains how this ties in neatly with Theory Revision.

3.3 Concept Drift and Theory Revision

Any Concept Drift situation can be viewed as a Theory Revision problem. Instead of viewing the situation as a sequence of different concepts, consider the sequence of concepts to be the target concept itself. At each stage, partial information about the sequence of concepts is received, and the theory that has been learned to explain all observations so far must be revised. The theory should approximate as closely as possible all the concepts that have been drifted through so far, and should provide a good description of the present concept. It is this idea that is used in the Concept Drift experiments in chapter 4.

It is also clearer, in this model, to see how one might create an algorithm to extrapolate the drift into the future. With a distribution over the class of all possible drift sequences, it should be possible to find the most likely sequence, and we can use this sequence to predict future concepts.

If we assume that the distribution over the sequences obeys a form of Occam's Razor, then the simplest set of rules that is consistent with the sequence so far might be hypothesised.

Chapter 4

Experiments

4.1 Previous Work

Within machine learning, surprisingly little work has been conducted to produce working algorithms to learn time-varying concepts. However, there are a few papers on this topic, as discussed above in the literature survey, most notably what of Widmer and Kubat [27].

4.2 Learning Changing Targets Monotonically

In this section we investigate the potential for learning changing targets using Progol [14]. We will use this ILP algorithm to learn rules based not only on classified instances, but on timestamped classified instances, so that Progol will not only learn predictive rules, but will also maintain a set of rules which will summarize the history of the concept changes to some degree.

The fundamental difference between the approach of the present paper and the work of Widmer and Kubat, is that instead of forgetting previous irrelevant examples, all the examples are kept, and Progol then learns time intervals as well as concepts. The rules learned will not only provide predictive rules for the present concept, but will have rules which summarize the previous target concepts.

This method has the additional advantage of potentially allowing us to establish patterns in the changes of concept, such as cyclic changes.

However, the method is not incremental (unlike W&K), and so as more examples arrive, the learning time will increase. A potential avenue of research to follow would be to consider a windowing approach for Progol, and perhaps an incremental learning style.

4.3 The Experiment

We will use the same artificial data as Widmer and Kubat to allow us to make direct comparisons to their results.

The instance space of simple children's blocks is defined by the three attributes $size \in \{small, medium, large\}$, $colour \in \{red, green, blue\}$, $shape \in \{square, circular, triangular\}$. The following three target concepts are used, each valid for 40 instances.

1. Instances 1–40. $size = small \wedge colour = red$
2. Instances 41–80. $colour = green \vee shape = circular$
3. Instances 81–120. $size = (medium \vee large)$

120 timestamped training instances are generated randomly, and labelled according to the appropriate concept. After each instance has been presented to Progol, the absolute predictive accuracy for the rules generated is calculated. The results were averaged over ten runs.

4.3.1 Interval Measures

In order to learn rules which will predict timestamped examples, we must somehow be able to provide Progol with a predicate which will formalize the notion of "after this time", "between these two times", etc. For the generalization algorithm to work, the predicate must not only be able to say when one time is before another, but also be able to generate a set of times that are before (or after) a particular given time.

A simple solution to this problem is the predefined Progol predicate '$\leq /2$'. This predicate has been written so that a call to $0 \leq X$ satisfies the following properties:

- By the $n+1^{th}$ recall, a rational number at least n will have been returned.
- Given any interval $[x, y]$, after enough recalls there will have been at least one instance in $[x, y]$.
- If X is instantiated, then the obvious and correct yes/no result is given.

However, this has several limitations in the experiment. We do not need rational numbers, but only integers. The returned values do not grow fast enough for us to be able to use a feasibly small number of recalls.
The predicate that was finally settled upon is as follows:

```
lte3(X,Y) :- not var(X), not var(Y), !, X=<Y.  lte3(X,Y) :- var(X),
not var(Y), nat(Z), X is Y-(7*Z).  lte3(X,Y) :- not var(X), var(Y),
nat(Z), Y is X+(7*Z).
```

This predicate generates values linearly in multiples of 7, allowing us to use a recall number of 6 times for this experiment. However, one must be wary of setting this sort of precedent, as there will not always be limits to the period between changes.

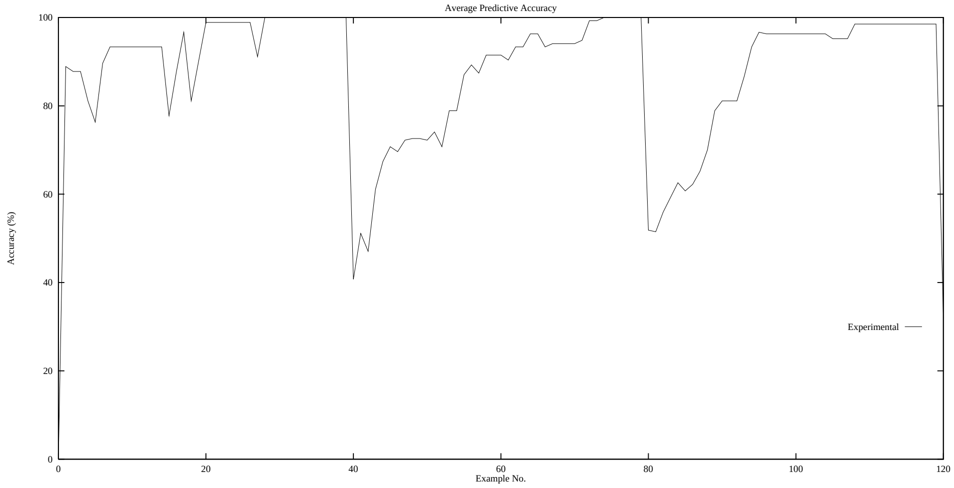

Figure 4.1: Experimental Results: Predictive Accuracy

4.3.2 Results

The Predictive Results are shown in figure 4.1. As expected, the accuracy increases up until the moment of a change, when it drops to a value equal to the % overlap between the consecutive concepts. The troughs in the first 3rd are caused by a rather interesting phenomenon. When Progol receives its first positive example, it immediately hypothesises that *everything* is true from that point in time onwards. The next negative example rectifies this over-generalisation, but the drop in accuracy is enough to produce the troughs, even when averaged over the ten trials.

It is also possible to derive results about how well Progol summarizes historical data. If Progol is given all 120 examples, and asked to generalise them, it produces rules which effectively summarize the data. These rules can be evaluated absolutely for their accuracy at all points in the past. Figure 4.2 displays the results. This historical record is the temporal theory described in section 3.3.

The historical results give some interesting results. The historical accuracy is overall very good. However it is clear that at the boundaries between changes of concept, Progol has problems. The accuracy decreases as a concept reaches the end of its "epoch", as well as increasing at the start of a new concept.

4.3.3 Theoretical Explanations for Observed Results

Theorem 4.3.1 $E[error(m,T)] \leq (1-\varepsilon)^m + \varepsilon$, *for some* $\varepsilon, 0 < \varepsilon < 1$.

Proof:

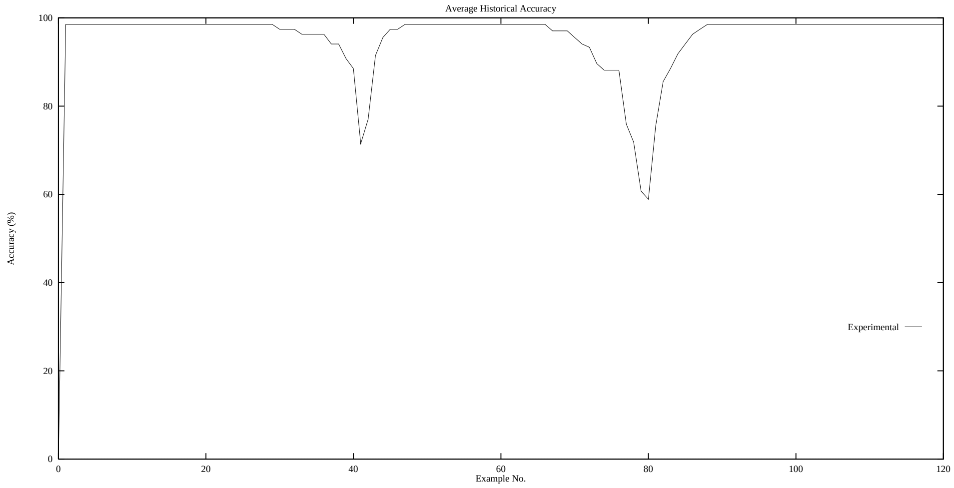

Figure 4.2: Experimental Results: Historical Accuracy

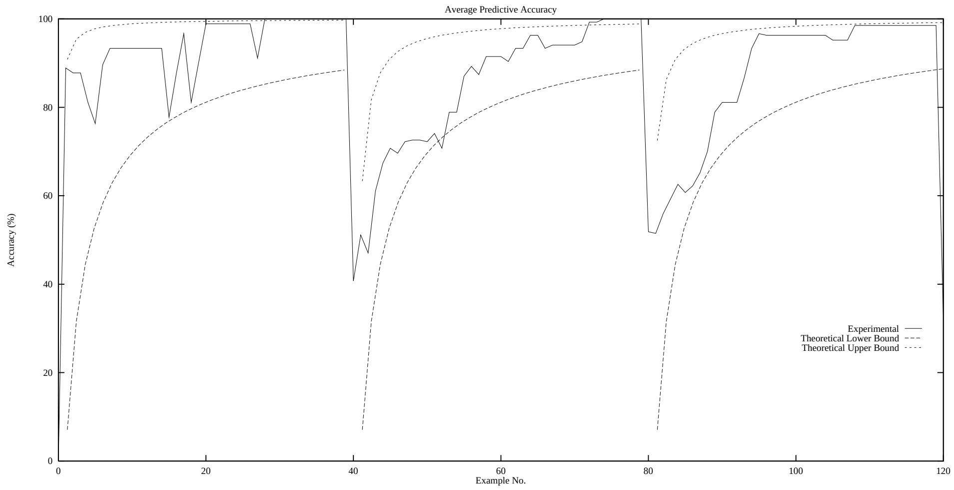

Figure 4.3: Experimental Result with Theoretical Bounds. The lower bound is from Theorem 4.3.2, and the upper bound from Remark 1

Let $\mathcal{A}$ be any learning algorithm, that takes as arguments m labelled elements of the domain X, and returns a consistent hypothesis $H \in \wp(X)$.

Pick ε, with $0 < \varepsilon < 1$. We can now divide the set X^m in two as follows: Let $\mathcal{T}_m = \{(x_1, \ldots x_m) \in X^m | error(\mathcal{A}(x_1, \ldots x_m), T) > \varepsilon)\}$, be the set of m-tuples of examples from which $\mathcal{A}$ induces an hypothesis with error greater then ε. Then $X^m = \mathcal{T}^m \cup (X^m \backslash \mathcal{T}^m)$.

Now,

$$\begin{aligned} E[error(m,T)] &= \sum_{x_1,\ldots x_m \in X^m} \Pr(x_1, \ldots x_m) error(\mathcal{A}(x_1, \ldots x_m), T) \\ &= \sum_{x_1,\ldots x_m \in \mathcal{T}^m} \Pr(x_1, \ldots x_m) error(\mathcal{A}(x_1, \ldots x_m), T) \\ &\quad + \sum_{x_1,\ldots x_m \in X^m \backslash \mathcal{T}^m} \Pr(x_1, \ldots x_m) error(\mathcal{A}(x_1, \ldots x_m), T) \end{aligned}$$

Now, by the definition of $\mathcal{T}^m$, the second summand is forced to be less than or equal to ε.

How can we put an upper limit on the first summand? It is clearly less than or equal to $\sum_{x_1,\ldots x_m \in \mathcal{T}^m} \Pr(x_1, \ldots x_m)$, as $error(\mathcal{A}(x_1, \ldots x_m), T) \leq 1$, for all $x_1, \ldots x_m$.

Let $H \in \wp(X)$ be such that $error(H,T) > \varepsilon$. Let $T \Delta H = T \setminus H \cup H \setminus T$ be the exclusive-OR of H and T.

$$\begin{aligned} \Pr_{x \in X}(x \in T\Delta H) &> \varepsilon \\ \Pr_{x \in X}(x \notin T\Delta H) &< 1 - \varepsilon \\ \Pr_{x_1,\ldots x_m \in X^m}(\forall i, x_i \notin T\Delta H) &< (1-\varepsilon)^m \\ \Pr_{x_1,\ldots x_m \in X^m}(D_X(T\Delta\mathcal{A}(x_1, \ldots x_m)) < \varepsilon) &< (1-\varepsilon)^m \end{aligned}$$

Therefore we have put a limit on the first summand. It is less than (or equal to) $(1-\varepsilon)^m$.

The result follows.

Theorem 4.3.2 $E[error(m,T)] \leq 1 + (\frac{1}{m})^{\frac{m}{m-1}} - (\frac{1}{m})^{\frac{1}{m-1}}$.

Proof:

We can minimize the value of $(1-\varepsilon)^m + \varepsilon$, for $0 < \varepsilon < 1$. Let $y = (1-x)^m + x$. Then $\frac{dy}{dx} = -m(1-x)^{m-1} + 1$, giving a local stationary point at $x = 1 - (\frac{1}{m})^{\frac{1}{m-1}}$. At this point $\frac{d^2y}{dx^2} > 0$, so this is indeed a minimum point. The result follows.

Remark 1 *It is a consequence of the* Blumer Bound *[5] that* $error(m,T) \geq \frac{\phi}{m}$, *(where ϕ is the fraction of the examples in the new concept that have the same labelling as for the previous concept), for all consistent learning algorithms, giving us an upper bound on the predictive accuracy. This curve also fits the experimental data well.*

4.4 Progol vs FLORA2

The following figure 4.4 shows the performance of Progol against the FLORA2 algorithm developed by Widmer and Kubat [27]. Although Progol appears slower at first to gain high accuracy levels, it is clear that in the second and third "epochs", which are the important and interesting time intervals, Progol is achieving higher accuracy than Flora2, although not significantly higher. For the first time interval, the algorithms are competing only at learning a static concept. Progol is burdened by the introduction of the time interval arithmetic into its background knowledge, making it produce inappropriate rules initially, as described in section 4.3.2.

Flora2 is an algorithm that is specifically designed to learn from a changing concept. Progol is a general purpose ILP learning algorithm. It is pleasing therefore that the results of the two algorithms are comparable. It is worth noting that Progol is actually doing more than Flora2 by producing the temporal concept at each stage. This would seem to support the temporal concept idea suggested in section 3.3.

4.5 Further Work

The work described above deals with a target concept that changes discretely over one instant of time. It would be interesting to extend the work to deal with concepts that are drifting, as defined in chapter 3.

The Bayes probability estimation function can be incorporated within Progol's evaluation function to improve its search when constructing hypothesis clauses. This would also allow one to parametrise the concept drift.

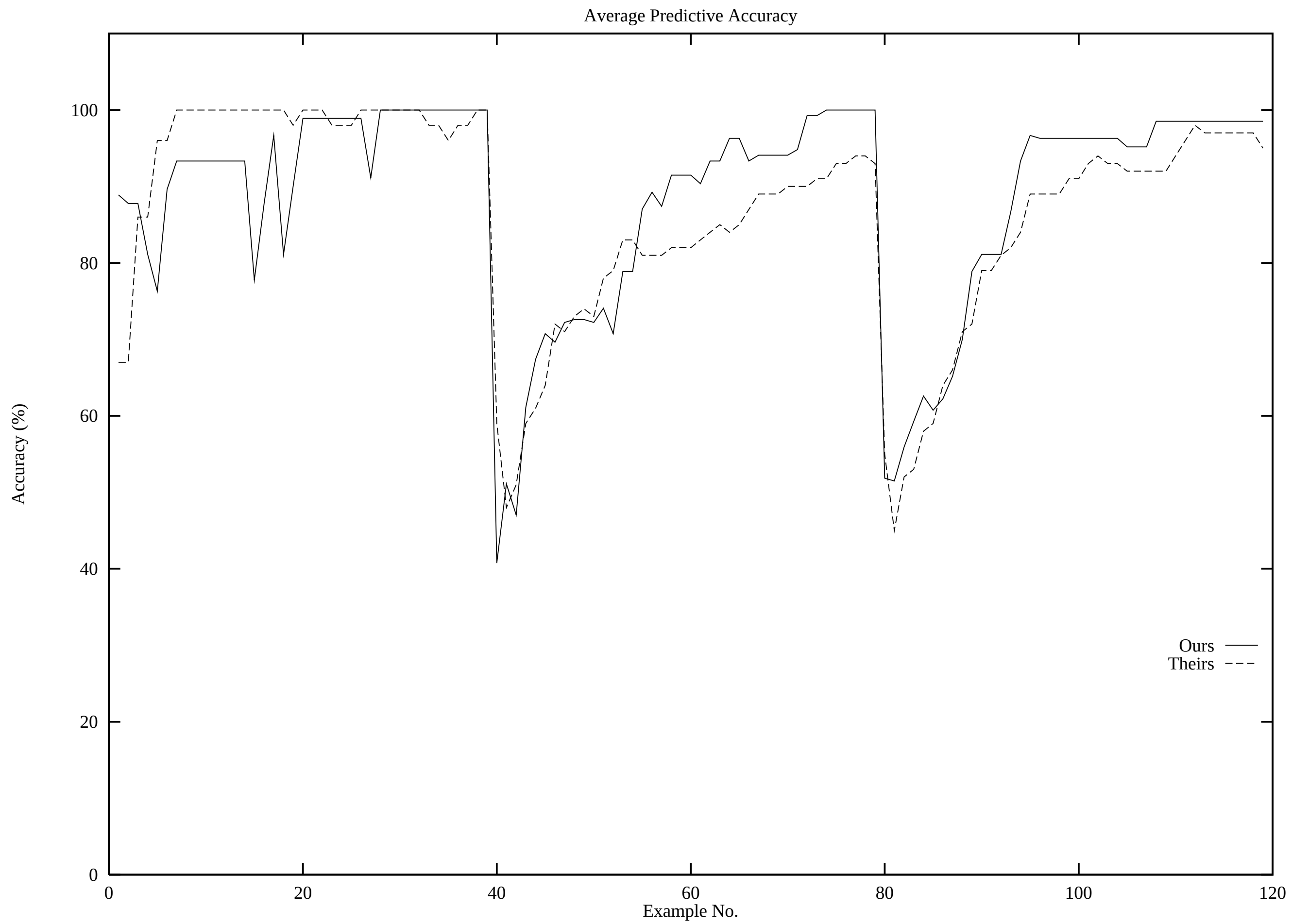

Figure 4.4: Experimental Result vs FLORA2

Chapter 5

Industrial Sponsor Visit

CASE sponsored students have the opportunity to spend 4 weeks each year on site at an industrial establishment. This allows the student to gain insight into research work that is being carried out in industry, to discuss possible applications of research work with employees, to meet with other CASE student and swap ideas and experiences, and to generally get a wider view of the work being conducted in the area. It is also intended that more practical experiments can be conducted at the site, possibly using data that the sponsors provide.

5.1 Introduction

With the advent of the internet and the information age, increasing pressure has been put on Telecommunication companies to provide services that can disseminate information efficiently and economically. Information has become a powerful asset, and procedures to find and use relevant and important information quickly and efficiently is becoming a major research area. Electronic trading of information is being facilitated by personal software agents, that act autonomously on behalf of an individual or company. These software mechanisms are designed to intercommunicate with other agents and users, and to act autonomously whilst doing so.

An obvious need for agents is the ability to learn from the information that they have gathered. This not only enables them to learn rules that allow them to predict, but also has the effect of compacting the data, reducing memory usage.

A multi-agent environment, with many agents cooperating and interacting with each other, is clearly not going to be static. The information that the agents store can be considered concepts, and the trade of a piece of information from one agent to the other is equivalent to the presentation of an example of that concept. These examples are being produced in an environment that is changing, and therefore the ability to tolerate Concept Drift is a fundamental one in this setting.

As described in the introduction, the author's MSc thesis [**?**] made a preliminary investigation into learning user interests from their classifications of Web pages. This led to the consideration that an agent that acts according its owner's interests would have to keep track of those interests, which would change with time.

British Telecom have experimented with a number of agents for the web. One that could loosely be described as an agent is the "Casper" intelligent web browsing cache. Casper is a proxy server for the internet, designed to be used by a group of like-minded individuals. The server caches accessed documents in case that they are requested again by the user, or by other members of the group. In addition, a primitive lookahead is conducted, whereby the server starts to load all the links from a page while it is being viewed. The aim is to increase document retrieval speed.

As a side effect of this process, Casper keeps a log of all accesses made through the server. These access logs can, at a first approximation, be used as a source of (positive only) examples of the interests of Casper's users. Each access is also labelled with a time and date, making them suitable for experiments in tracking a drifting concept. One major drawback is the lack of negative examples, and this restricted the work in this visit to simply learning user interest rules.

5.2 Materials

The major identifying feature of a web page is its lexical content. The word content of each page is therefore the fundamental piece of background knowledge that one must present to a learning algorithm.

Therefore, preprocessing was performed on the example pages to download the contents of each page, and extract and count the words in them. Pages that were likely to change (i.e. those called through a *cgi-bin* script) were ignored, as were any pages that were downloaded that contained no text.

It is clear that there are some words that are closely related through their semantic value. In the author's MSc thesis, an attempt was made to extract this semantic correlation by using an electronic semantic dictionary called WordNet [12]. Various relationships were defined in this dictionary, including the concept used in MSc, *hypernym*. X is a hypernym of Y if Y is a type of (or way of doing) X.

The MSc experiments were not entirely successful, and the increase in computational time incurred by including this relationship in the background knowledge was significant.

Although it does not gain any real semantic value, a more efficient step in this direction can be made by reducing all words to their root form. In other words, establish the similarity of the words *computer, compute, computation* etc, by reducing the words to the common word stem, *comput*. This should give an approximate indication of semantic similarity, through the relationship of syntactic similarity for words of the same root. To this end, all the words extracted from the Web pages were reduced to their stem forms, by using the Porter Stemming Algorithm [18].

The ILP algorithm Progol [14], implemented in C, with positive example only learning implemented [15], was used as the learning algorithm.

5.3 Methods

To learn interests from the web pages, one must distinguish between the words that appear frequently because the user is interested in that topic, and the words that appear frequently because of their high relative frequency in the English language. In other words, we must distinguish between words such as "then" and "agent", each of which may appear frequently on a page but for different reasons. To enable this distinction to be made, an English word frequency count data corpus was obtained.

5.3.1 Data Corpus of English Word Frequencies

The data corpus, compiled from a variety of different English written text sources, has a total frequency count of 17.9 million words.

Using the following formula, it was then possible to calculate the importance of a word to a particular page, by calculating the *excess information* that the word had on that page.

$$\text{ExcessInfo}(W, page) = -\log_2\left(\frac{freq(page, W)}{\sum_w freq(page, w)}\right) + \log_2\left(\frac{freq(corpus, W)}{\sum_w freq(corpus, w)}\right)$$

Once the excess information had been calculated for each word on a page, the top 20 words were then taken as being the representative words for that page. This also had the effect of reducing the size of the background knowledge representation of the pages, and thus decreasing computation times.

5.3.2 Progol

The examples given to Progol were of the form:

`interested(`*time*`,`*machine*`,`*page*`).`

And the Background Knowledge:

`hasstem(`*page*`,`*word-stem*`).`

Progol is used to generalise the examples relative to the background knowledge, to produce rules of the form:

`interested(T,`*machine*`, P) :- hasstem(P,`*word-stem*`), ...`

It is here that certain trade-offs must be made. The number of `hasstem/2` clauses in the body of the learned clause changes the generality or specificity of that clause. If all the learned clauses have too many clauses in the body, then the rules will be too specific, and will have bad coverage of the user's interests, failing to predict that he is interested in a page. If there are too few clauses in the body, then the rules will be too general, and will incorrectly classify uninteresting pages as interesting. Progol can be constrained to learn clauses that contain the correct number of literals in the body.

5.3.3 Testing Rule Accuracy

In order to test the accuracy of the rules generated by Progol, it is necessary not just to have examples of pages that the user found interesting, but also to have instance of pages that are not interesting. Without negative examples in a test set, it is perfectly possible for a set of interest rules to score 100% accuracy, and be totally inappropriate. The rule that just states that the user is interested in every page is an obvious case. Therefore, a set of 50 pages was chosen at random from among all those that had been logged at Casper by any of the users. These pages were presented to the user in question, who then classified them into positive and negative classes of interest, forming a test set for testing the accuracy of generated interest rules.

5.4 Results

Unfortunately, due to time constraints, not as many results were collected as might have been hoped. Only one user's interests were tested, and consequently, the results are not to be taken as statistically valid. However, the preliminary results were encouraging. The following is a contingency table and associated χ^2 statistical test for a particular user. The rules constructed had two `hasstem/2` clauses in the body. The rules were learned from 629 example pages - A month's worth of browsing for this particular user. The test set was the classified set of 50 pages described above. The rules derived are shown in appendix A.

	A	$\overline{A}$	
P	8	6	14
$\overline{P}$	3	33	36
	11	39	50

Overall accuracy = 82.00% ± 5.43%
$\chi^2 = 11.29$
$P(\chi^2 = 11.29) = 0.0008$

The user was generally happy with the rules generated. Some were more valid to him than others. For instance the rules:

```
interested(molokai,A) :- hasstem(A,agent), hasstem(A,softwar).
interested(molokai,A) :- hasstem(A,applet), hasstem(A,java).
interested(molokai,A) :- hasstem(A,comlab), hasstem(A,oxford).
interested(molokai,A) :- hasstem(A,ipswich), hasstem(A,martlesham).
```

made immediate sense to him, whereas:

```
interested(molokai,A) :- hasstem(A,justif), hasstem(A,preced).
interested(molokai,A) :- hasstem(A,genu), hasstem(A,line).
interested(molokai,A) :- hasstem(A,default), hasstem(A,paramet).
interested(molokai,A) :- hasstem(A,centr), hasstem(A,page).
```

did not.

This dissatisfaction does not necessarily indicate incorrectness in the rules for the purposes of predicting interest for new pages.

5.5 Comments and Suggestions for Further Work

It is clear that much work can be done in this area to extend this preliminary investigation. The original aim, of using the data as an example set of a changing concept, has not been realised.

This was partly due to the fact that the data was interpreted as being a set of positive examples of user interest, from which the problem of learning a changing concept is intrinsically more difficult. (How can one tell that a rule has stopped being valid, if there are no negative examples to contradict it?). The original assumption, that the user is interested in every page that he looks at, is clearly naïve. There are no guarantees that, just because a user opens a link, he will be interested in the contents of the page arrived at from that link.

A better approximation could perhaps be made by using the duration that a user looks at a page as an indication of interest. A page visited only briefly is not interesting. However, this ignores the possibility that this is merely a case of familiarity, and the user flicks through the page quickly, not through lack of interest, but because he knows it well.

Another aspect that has not been fully investigated, is the effect of good background knowledge about the user on the accruacy of the rules generated. No use was made of any source of information about the user, whether facts constructed by the user, or extracted from other sources, such as the user's curriculum vitae, or Web home pages.

In the author's MSc thesis, a simple Naive Bayes learning algorithm proved most effective at learning from positive and negative examples of user web page interest. It would be interesting to adjust this algorithm to try to learn from positive examples only.

For an agent to contain a correct description of the user's beliefs and intentions, the rules it contains should be validated in some way by the user in question. Progol allows one to constrain against clauses being generated during the learning process, giving an easy way to prevent the generation of rules that a user diagrees with. A process of user verification and refinement could be explored, whereby rules generated initially are marked as invalid by the user, if he disagrees with them. The original examples are again generalised by Progol, but this time with constraints against the invalid rules. This process is repeated until the user is satisfied. Of course, an ideal web browsing aid would have little or no user involvement, but an agent must not contain information that is incorrect for the user. This is an idea that needs more investigation.

Chapter 6

Conclusions, and DPhil Proposal

6.1 Conclusions

This preliminary work was intended to establish an insight into the work conducted so far in the area, to conduct simple investigations, in the process revealing the problems that need to be overcome, and to find avenues of research suitable for further work.

In extending the PAC framework of Valiant [23], it is argued that modelling Concept Drift must be approached probabalistically. A first attempt was made at tackling this by building a theoretical model based on Bayesian probability. Although initially encouraging, there is still much work to be done on this model, including sample complexity calculations and the extension to trying to learn a sequence of concepts, as opposed to just the single present concept.

The idea of trying to learn the *sequence* of concepts lead to the conclusion that Theory Revision may be an important tool when applying ILP to the problem. No attempt has been made yet to make use of Theory Revision in Learning drifting concepts; indeed, most of the work so far has been theoretical, without any attempt to instantiate the ideas by constructing an algorithm. The only attempt in this vein has been the work by Widmer and Kubat [27], in which they used a "Window Adjustment Heuristic" on the set of past examples to restrict the damage of past incorrect examples.

An attempt was made to learn the drifting concept sequences used in Widmer and Kubat's paper. The ILP algorithm PROGOL was used, and a very simple approach was used to learn rules that described the entire sequence. The approach was successful, despite its basic nature. Currently, the entire example sequence is reused each time (with the new example added). This is inefficient, and gets slower with each new example.

The experiment could be extended using a more sophisticated Theory Revision approach to learning a temporal theory. For this, the old hypothesis would be revised at each stage, by using Theory Revision operators specifically designed to cope with a concept that is growing, rather than just being corrected. These Theory Revision operators would produce a more general theory each time, but may be doing so by modifying clauses within the previous theory. At each time, the theory would be a *temporal* theory that should closely match the sequence of past concepts. However, a Theory Revision approach cannot achieve a greater accuracy than a batch approach (there is less information, in the information theory sense, available at each stage), and so one must be wary of surrendering accuracy for efficiency.

An approach to the batch learning of drifting concepts can be formulated with the use of the Bayesian estimate of chapter 3. This is distinct from the Theory Revision method described above in that the learner must only form a hypothesis based on the set of past examples, and is not expected to use new examples to revise this hypothesis. Its use would therefore be limited, perhaps to establishing an initial hypothesis, which a theory revision based system would use as a starting point for its revisions. Notice that any Theory Revision based learner can also be used to learn from scratch, by simply having the empty theory as the initial theory to be revised.

If it is not necessary to derive a record of the past concepts during the learning procedure, then a more simple Theory Revision system can be used, in which new examples simply force specialisation or generalisation of the current theory. No "historical concept" of the concept sequence is considered.

The theory of Temporal Reasoning must be investigated prior to formulating theoretical work on learning temporal concepts. This will form part of the remaining background reading. Only recently has it become apparent that this area is significant.

6.2 The Thesis

The thesis will have the following structure.

- Introduction
- Literature Survey
- Theory
- Application Description
- Algorithm Development and Implementation
- Experimental Results
- Conclusions and Further Work

The following must be addressed over the course of the next two years. Estimated completion times are given as a guide for the intended schedule.

- Calculate sample complexity results for the batch learning of drifting concepts, based on the naive Bayes estimation formula derived in chapter 3. Expected error will be analysed to give an average case sample complexity [4]. 2 months.

- Investigate Theory Revision operators and the potential for applying a specialized form of Theory Revision to the problem of incrementally tracking a drifting concept. Formulate such theory tracking operators. Are they complete (ie will a good theory always be found if there is one)? What does it mean for such operators to be correct in this setting? 3 months.

- Establish a framework for learning changing concepts in an Inductive Logic Programming setting. This would be an extension of Bayesian Inductive Logic Programming U-learnability setting described in [13]. 2 months.

- Implement a Concept Drift tracking algorithm. This may be done by extending a previously existing algorithm (Progol for example), or by beginning from scratch. Extending Progol would be the easiest route. Progol could be used for the generalisation stages of theory revision, and the Bayesian estimate can be used to optimize the search when Progol is constructing the clauses of the hypothesis. 6 months.

- Test the theoretical results in an experimental setting. Artificial data could be used at first, and if successful, learning from real data would be investigated. Accuracy would be tested by a hold-out test. Suitable sources of data must be identified. 6 months.

- Write up the thesis. This is estimated to take 6 months at the most.

One source of data that has been considered is that of an electronic diary database. A user's conception of future engagements are stored in the diary for the user's reference. These engagements would include regular commitments, which a Learner would be able to generalise to form compact rules. However, engagements do change, and hence so does the target concept for the Learning Algorithm. This is potentially an interesting and easily gathered source of data.

Appendix A

Derived User Interest Rules

```
interested(molokai,acds).
interested(molokai,acdv).
interested(molokai,A) :- hasstem(A,access), hasstem(A,introduct).
interested(molokai,A) :- hasstem(A,agent), hasstem(A,refer).
interested(molokai,A) :- hasstem(A,agent), hasstem(A,softwar).
interested(molokai,A) :- hasstem(A,algorithm), hasstem(A,ieee).
interested(molokai,A) :- hasstem(A,applet), hasstem(A,java).
interested(molokai,A) :- hasstem(A,autom), hasstem(A,updat).
interested(molokai,A) :- hasstem(A,bandwidth), hasstem(A,fibr).
interested(molokai,A) :- hasstem(A,boardman), hasstem(A,roming).
interested(molokai,A) :- hasstem(A,browser), hasstem(A,capabl).
interested(molokai,A) :- hasstem(A,browser), hasstem(A,cern).
interested(molokai,A) :- hasstem(A,browser), hasstem(A,chri).
interested(molokai,A) :- hasstem(A,browser), hasstem(A,denmark).
interested(molokai,A) :- hasstem(A,browser), hasstem(A,docum).
interested(molokai,A) :- hasstem(A,calendar), hasstem(A,ical).
interested(molokai,A) :- hasstem(A,centr), hasstem(A,martlesham).
interested(molokai,A) :- hasstem(A,centr), hasstem(A,page).
interested(molokai,A) :- hasstem(A,centr), hasstem(A,technologi).
interested(molokai,A) :- hasstem(A,christoph), hasstem(A,hillman).
interested(molokai,A) :- hasstem(A,click), hasstem(A,copyright).
interested(molokai,A) :- hasstem(A,cluster), hasstem(A,data).
interested(molokai,A) :- hasstem(A,code), hasstem(A,text).
interested(molokai,A) :- hasstem(A,comlab), hasstem(A,oxford).
interested(molokai,A) :- hasstem(A,competitor), hasstem(A,network).
interested(molokai,A) :- hasstem(A,comput), hasstem(A,enter).
interested(molokai,A) :- hasstem(A,comput), hasstem(A,page).
interested(molokai,A) :- hasstem(A,coordin), hasstem(A,multi).
interested(molokai,A) :- hasstem(A,copyright), hasstem(A,correct).
interested(molokai,A) :- hasstem(A,copyright), hasstem(A,homepag).
interested(molokai,A) :- hasstem(A,copyright), hasstem(A,insid).
```

```
interested(molokai,A) :- hasstem(A,crabtre), hasstem(A,titmuss).
interested(molokai,A) :- hasstem(A,crpo), hasstem(A,cyberoffic).
interested(molokai,A) :- hasstem(A,cyberoffic), hasstem(A,dave).
interested(molokai,A) :- hasstem(A,cyberoffic), hasstem(A,homepag).
interested(molokai,A) :- hasstem(A,data), hasstem(A,info).
interested(molokai,A) :- hasstem(A,databas), hasstem(A,overview).
interested(molokai,A) :- hasstem(A,david), hasstem(A,onlin).
interested(molokai,A) :- hasstem(A,david), hasstem(A,paul).
interested(molokai,A) :- hasstem(A,default), hasstem(A,paramet).
interested(molokai,A) :- hasstem(A,distribut), hasstem(A,parallel).
interested(molokai,A) :- hasstem(A,docum), hasstem(A,move).
interested(molokai,A) :- hasstem(A,email), hasstem(A,emporium).
interested(molokai,A) :- hasstem(A,email), hasstem(A,fraud).
interested(molokai,A) :- hasstem(A,email), hasstem(A,html).
interested(molokai,A) :- hasstem(A,email), hasstem(A,list).
interested(molokai,A) :- hasstem(A,email), hasstem(A,microsoft).
interested(molokai,A) :- hasstem(A,email), hasstem(A,softwar).
interested(molokai,A) :- hasstem(A,email), hasstem(A,telecom).
interested(molokai,A) :- hasstem(A,extens), hasstem(A,user).
interested(molokai,A) :- hasstem(A,finin), hasstem(A,jess).
interested(molokai,A) :- hasstem(A,finin), hasstem(A,multi).
interested(molokai,A) :- hasstem(A,finin), hasstem(A,string).
interested(molokai,A) :- hasstem(A,finlayson), hasstem(A,neil).
interested(molokai,A) :- hasstem(A,genu), hasstem(A,line).
interested(molokai,A) :- hasstem(A,graham), hasstem(A,paul).
interested(molokai,A) :- hasstem(A,html), hasstem(A,http).
interested(molokai,A) :- hasstem(A,html), hasstem(A,nbsp).
interested(molokai,A) :- hasstem(A,hyperspac), hasstem(A,slide).
interested(molokai,A) :- hasstem(A,hyperspac), hasstem(A,structur).
interested(molokai,A) :- hasstem(A,imageobserv), hasstem(A,isen).
interested(molokai,A) :- hasstem(A,intellig), hasstem(A,unit).
interested(molokai,A) :- hasstem(A,interfac), hasstem(A,user).
interested(molokai,A) :- hasstem(A,internet), hasstem(A,invis).
interested(molokai,A) :- hasstem(A,internet), hasstem(A,joseph).
interested(molokai,A) :- hasstem(A,internet), hasstem(A,lair).
interested(molokai,A) :- hasstem(A,internet), hasstem(A,microsoft).
interested(molokai,A) :- hasstem(A,internet), hasstem(A,multimedia).
interested(molokai,A) :- hasstem(A,internet), hasstem(A,onlin).
interested(molokai,A) :- hasstem(A,internet), hasstem(A,server).
interested(molokai,A) :- hasstem(A,internet), hasstem(A,updat).
interested(molokai,A) :- hasstem(A,introduct), hasstem(A,position).
interested(molokai,A) :- hasstem(A,ipswich), hasstem(A,martlesham).
interested(molokai,A) :- hasstem(A,java), hasstem(A,label).
interested(molokai,A) :- hasstem(A,java), hasstem(A,rotat).
interested(molokai,A) :- hasstem(A,justif), hasstem(A,preced).
interested(molokai,A) :- hasstem(A,keith), hasstem(A,paul).
```

```
interested(molokai,A) :- hasstem(A,keyword), hasstem(A,moonei).
interested(molokai,A) :- hasstem(A,kqml), hasstem(A,lenat).
interested(molokai,A) :- hasstem(A,kqml), hasstem(A,sender).
interested(molokai,A) :- hasstem(A,kqml), hasstem(A,stanford).
interested(molokai,A) :- hasstem(A,last), hasstem(A,updat).
interested(molokai,A) :- hasstem(A,laurent), hasstem(A,motorola).
interested(molokai,A) :- hasstem(A,lopez), hasstem(A,mesospher).
interested(molokai,A) :- hasstem(A,microsoft), hasstem(A,multivari).
interested(molokai,A) :- hasstem(A,microsoft), hasstem(A,newcom).
interested(molokai,A) :- hasstem(A,mors), hasstem(A,movi).
interested(molokai,A) :- hasstem(A,mors), hasstem(A,next).
interested(molokai,A) :- hasstem(A,motorola), hasstem(A,palaiseau).
interested(molokai,A) :- hasstem(A,nbsp), hasstem(A,telecommun).
interested(molokai,A) :- hasstem(A,newsgroup), hasstem(A,roming).
interested(molokai,A) :- hasstem(A,page), hasstem(A,perl).
interested(molokai,A) :- hasstem(A,page), hasstem(A,racket).
interested(molokai,A) :- hasstem(A,page), hasstem(A,server).
interested(molokai,A) :- hasstem(A,page), hasstem(A,structur).
interested(molokai,A) :- hasstem(A,request), hasstem(A,server).
interested(molokai,A) :- hasstem(A,reuter), hasstem(A,sept).
interested(molokai,A) :- hasstem(A,server), hasstem(A,updat).
interested(molokai,A) :- hasstem(A,softwar), hasstem(A,summari).
interested(molokai,A) :- hasstem(A,telecom), hasstem(A,updat).
[Total number of clauses = 100]

test(molclass)?
[False negative:]:- interested(molokai,aapb).
[False positive:]interested(molokai,aaah).
[False positive:]interested(molokai,acks).
[False negative:]:- interested(molokai,acgx).
[False negative:]:- interested(molokai,abpc).
[False negative:]:- interested(molokai,abum).
[False positive:]interested(molokai,abts).
[False negative:]:- interested(molokai,acha).
[False negative:]:- interested(molokai,ablu).

[PREDICATE interested/2]

Contingency table=    _________A_________~A
                    P|        8|        6|        14
                     |(    3.1)|(   10.9)|
                   ~P|        3|       33|        36
                     |(    7.9)|(   28.1)|
                      ~~~~~~~~~~~~~~~~~~~
                             11        39        50
[Overall accuracy= 82.00% +/- 5.43%]
```

```
[Chi-square = 11.29]
 [Without Yates correction = 13.99]
[Chi-square probability = 0.0008]
yes
[:- test(molclass)? - Time taken 0.78s]
```

Bibliography

[1] H. Adé, B. Malfait, and L. De Raedt. RUTH: an ILP theory revision system. In Zbigniew W. Raś and Maria Zemankova, editors, *Proceedings of the 8th International Symposium on Methodologies for Intelligent Systems*, volume 869 of *LNAI*, pages 336–345, Berlin, 2–4 May 1994. Springer-Verlag.

[2] H. Adé, L. De Raedt, and M. Bruynooghe. Theory revision. In S. Muggleton, editor, *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, pages 179–192, 1993.

[3] P. L. Bartlett, S. Ben-David, and S. R. Kulkarni. Learning changing concepts by exploiting the structure of the change. Technical Report CIS 9603, Technion — Israel Institute of Tchnology Computer Science Department, Haifa 32000 Israel, January 1996.

[4] S. Ben-David, B. Chor, O. Goldreich, and M. Luby. On the theory of average case complexity. *Journal of Computer and System Sciences*, 44:193–219, 1992.

[5] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the vapnik-chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, 1989.

[6] J Cussens. Bayes and psuedo-bayes estimates of conditional probabilities and their reliability. In Pavel B. Brazdil, editor, *European Conference on Machine Learning*, volume 667 of *LNAI*, pages 136–152, Vienna, 1993. Springer-Verlag.

[7] S. Džeroski, B. Cestnik, and I. Petrovski. The use of bayesian probability estimates in rule induction. *unknown*, 1992?

[8] D. Haussler, M. Kearns, and R. Schapire. Bounds on the sample complexity of Bayesian learning using information theory and the VC dimension. *Machine Learning*, 14(1):83–113, 1994.

[9] D. P. Helmbold and P. M. Long. Tracking drifting concepts by minimizing disagreements. *Machine Learning*, 14:27–45, 1994.

[10] A. Kuh, T. Petsche, and R. L. Rivest. Learning time-varying concepts. In Richard P. Lippmann, John E. Moody, and David S. Touretzky, editors, *Advances in Neural Information Processing Systems*, volume 3, pages 183–189. Morgan Kaufmann Publishers, Inc., 1991.

[11] A. Kuh, T. Petsche, and R. L. Rivest. Incrementally learning time-varying half-planes. In John E. Moody, Steven J. Hanson, and Richard P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4, pages 920–927. Morgan Kaufmann Publishers, Inc., 1992.

[12] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. Miller. *Five Papers on WordNet*. Cognitive Science Laboratory, Princeton, 1993.

[13] S. Muggleton. Bayesian inductive logic programming. In W. Cohen and H. Hirsh, editors, *Seventh Annual ACM Conference on Computational Learning Theory*, pages 371–379, San Mateo, CA, 1994. Morgann-Kaufmann.

[14] S. Muggleton. Inverse entailment and progol. *New Generation Computing*, 13:245–286, 1995.

[15] S. Muggleton. Learning from positive data. In *ILP96*, 1996.

[16] S. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 339–352, 1988.

[17] S. Muggleton and L. De Raedt. Inductive logic programming: theory and methods. *Journal of Logic Programming*, 19,20:629–679, 1994.

[18] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, July 1980.

[19] B. L. Richards and R. J. Mooney. First order theory revision. In *Proceedings of the 8th International Workshop on Machine Learning*, pages 447–451. Morgan-Kaufmann, 1991.

[20] J. C. Schlimmer and R. H. Granger. Incremental learning from noisy data. *Machine Learning*, 1(3):317–354, 1986.

[21] E. Y. Shapiro. An algorithm that infers theories from facts. *IJCAI*, pages 446–451, 1984.

[22] E. Y. Shapiro. Inductive inference of theories from facts. In J-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honour of Alan Robinson*, chapter 7. MIT Press, 1991.

[23] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.

[24] S. A. Vere. Induction of concepts in the predicate calculus. In *Fourth International Joint Conference on Artificial Intelligence*, pages 281–287, Tbilisi, USSR, 1975. Morgann-Kaufmann.

[25] S. A. Vere. Induction of relational productions in the prescence of background information. In *Fifth International Joint Conference on Artificial Intelligence*, pages 349–355, Cambridge, MA, USA, 1977. Morgann-Kaufmann.

[26] D. Welsh. *Communication and Coding Theory*, chapter 8. Oxford University Press, 1990.

[27] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23:69–101, 1996.

[28] J. Wogulis. Revising relational domain theories. In *Proceedings of the 8th International Workshop on Machine Learning*, pages 462–466. Morgan-Kaufmann, 1991.