Imperial College London

Department of Computing

MEng Individual Project

Learning mobile user behaviours

Andreas Markitanis
am907@doc.ic.ac.uk

Supervisor:
Dr. Alessandra Russo

Co-Supervisor:
Dr. Emil Lupu

June 21, 2011

Abstract

With the emergence of ubiquitous computing, innovations in mobile phones have changed the way users lead their lives. With nearly 73% [BBC] of the world population owning a mobile phone, its use defines certain rules and behaviours. Minimising user interaction requires development of adaptive systems that autonomously respond to changes by identifying patterns in past usage data. The use of machine learning to learn behaviours, based on empirical data, allows for autonomous user management with either minimal or no user intervention at all. We explore the possibilities of extracting user behaviour rules and patterns from past usage data. Such rules can be deployed to define or adapt existing policy rules that manage pervasive systems. Specifically, this project aims at investigating the feasibility of using Inductive Logic Programming (ILP) in the automated task of extraction of user behaviour rules through data mining and acquisition. This is a challenging task for various reasons. Firstly, to our knowledge, it is the first attempt of using ILP in tasks that in the past have been carried out using conventional data mining techniques. Furthermore, the datasets often carry a substantial amount of noise and are not evenly/normally distributed. Lastly, the data may often be not rich enough to allow the computation of expressive rules. The challenge is to investigate to what extent ILP and the recently developed system TAL can be adapted and customised to perform the learning of meaningful user behaviour rules in a way that addresses the above obstacles. After an experimental investigation with the Reality Mining dataset, a framework is set up for the automation and extraction of rules. In conjunction with this framework, we present *ULearn*, a mobile phone application developed to gather and process quantitative data autonomously. The learning results, that are presented to the user in Prolog and English, can ultimately be "enforced", refined, by the means of integrity constraints, or selected for theory revision.

Acknowledgements

I would like to thank both my supervisors, Dr. Alessandra Russo and Dr. Emil Lupu for their continuous support, guidance and enthusiasm they have rendered throughout the entire duration of this project. They have both constantly remained confident in my ability to deliver, for ideas and inspiration. I would also like to express immense gratitude to Domenico Corapi for the endless conversations we had, his assistance and advice on how best to move forward. Without him, this work would not have been possible. Finally, I would like to thank my friends and especially my family for their continuous financial and moral support throughout my four years at Imperial College.

Contents

1 Introduction

In this chapter, we are going to explain the problem tackled in this project, the aspects we need to take into account and why it is difficult to address. Section 1.1 explains the problem and motivation behind the project and the challenges we face in solving it. Section 1.2 outlines what this project is effectively trying to achieve along with a brief overview of the steps we intend on undertaking to find a solution. Section 1.3 lists our contributions towards the project and finally, Section 1.4 provides a summary of how this report is organised.

1.1 Motivation

In the past few years, companies such as Apple and Samsung have really managed to develop cutting-edge mobile systems that have revolutionised the way we use mobile phones. Often, the complexity of these systems prevent the user from utilising the device to its maximum. To facilitate the user, such systems must be able to continuously adapt to the user's preferences and behaviour while requiring as little help as possible from the user. Policies, governing the choices in behaviour of a system, are increasingly becoming more popular for implementing adaptive systems that promote the flexibility to manage such devices.

Extensive research in this field has shown that, while these frameworks work as expected, they require disclosure of private information that the user might not be comfortable with or in favour of. An alternative approach, far less intrusive, is to implicitly collect information about a particular user through their actions and make use of past usage data for rule-based knowledge acquisition to define previously unknown relations. These need not necessarily be in the form of logic programs. Instead, any programming language capable of representing rules would be sufficient (e.g Lisp). Adaptive systems could then benefit from these (logic) programs by taking the appropriate action when certain conditions are met.

To provide an accurate solution to the problem we need to address some of the challenges that come attached with it. To say that there exist noise-less datasets, it would be a lie. Noise in

data often plays a major role in the degradation of performance of learning systems in general and is important that it is addressed properly. Data are often not normally/evenly distributed. In multi-relational data mining problems this has a negative impact since patterns are usually identified through the "completeness" of data. Therefore, data distribution is key to solving the problem. Lastly, the domain of the problem we are targeting, although not infinite, is very large and a large amount of rich data is required to compute expressive rules.

According to [Wei91], pervasive systems should "weave themselves into the fabric of everyday life until they are undistinguishable from it". However, until recently, the majority of such systems have been unable to minimally involve the user in taking decisions on their behalf. Firstly, such systems require policy frameworks to govern certain actions but often prove to be impractical as users are always asked to specify their preferences in advance. Secondly, it is impossible to predict every possible case. In addition, users will find it hard to express the way they want to share information by means of rules. Therefore, manually defining a human-driven specification of these policies becomes part of a largely unrealistic process.

Adaptive systems belong to a class known as context-aware computing [SAW94].They are practically used, successfully, in network security (e.g., intrusion detection systems), search engines (e.g., feature extraction), pattern recognition (e.g., audio-visual recognition problems) and computer games. Another form of adaptability in the context of mobile devices, directly related to our problem, is that of privacy. There has been a huge increase of interest in adapting the user's privacy settings through the use of privacy policies.

Nevertheless, users are still concerned with what information their mobile devices can expose. Even if we grant the user's consent, the implicit collection of data is a major concern for most users. For this reason, the development of privacy management frameworks becomes essential. Combining the two, and their respective solutions, together results in a powerful platform. That is, the problem of extracting user behaviour rules finds a direct application in the context of privacy management research. A joint research project [PRI] between the Open University and Imperial College London, aims to develop a Privacy Rights Management (PRM) framework that will enable users to specify an appropriate level of privacy in pervasive systems. However, such frameworks need to be flexible enough to allow the users to interact with the device as they wish. So far, development of privacy management frameworks mostly focussed on representation and analysis of policy specifications as opposed to learning such policies. [Cra02, AHK+03].

As previously mentioned, we cannot expect the user to define such policies with great confidence. Recent efforts, without the explicit need of the user's intervention, include that of [BRL07] which suggest an analysis and learning of privacy policies using Inductive Logic Programming (ILP). In such scenarios, ILP is preferred than traditional machine learning techniques because of its powerful expressibility. ILP learns target relations, expressed in logic programs notation, that are easily understood by users and are amenable to automated analysis [BRL07, CRR+08]. Additionally, declarative rules are easily translated into English language rules using a pre-defined mapping of words and sentences thus enabling the system to involve the user for validation of the learned outcomes.

Learning rules, is only a preliminary step but the results should be promising. The real challenge, which still lies as a main research open question is the development of frameworks that allow the learning of privacy policy rules from past usage and user behaviour data in an automated fashion.

The motivation behind this project is twofold. We are interested in overcoming the aforementioned challenges/problems and provide a solution that is correct and accurate and requires as

little user input as possible. To do this we must first investigate the fundamentals of logic, inductive and abductive logic programming that will provide us with the right theory and correct mechanisms to learn.

1.2 Objectives

This project sets out to explore the benefits of using alternative methods to adapt systems through learning. More specifically, we look at different ways of applying Inductive Logic Programming (ILP) to learn user behaviours of mobile phones. We aim to develop a learning framework for the automated extraction of raw data into a learning file which is then used for the computation of rules. On top of that, we aim to complement it with an Android application enabling us to perform real user evaluation to validate our results.

The novelty behind the project is the fact that we use ILP to learn such rules, an attempt that has only been tried using conventional data mining and other machine learning techniques in the past. We also examine the effects of two other machine learning techniques to gain a deeper understanding of the hidden structure in the data and if worthwhile for an improved background knowledge in an ILP setting, in the face of uncertainty.

The real challenge is to explore whether ILP can be used over arbitrary real (large) set of data that contain a large amount of noise and are not normally distributed. We then evaluate the results based on empirical data obtained by our own developed mobile phone application. This validation will enable us to draw a conclusion based on the results we obtain.

This requires, as a first step, the definition of a domain model that is suitable to the problem of learning mobile user behaviours. This entails the definition of appropriate background knowledge, language bias and training data. This setting enables us to both apply ILP using TAL [CRL10] (a non-monotonic inductive programming tool developed at Imperial College) as well as other statistical machine learning algorithms that may prove beneficial to the precision of the overall learning process.

To improve accuracy, it is necessary to proceed with experimental analysis by tweaking the learning parameters and perform rule refinement with the use of integrity constraints. For refining rules, we also involve the users by taking their feedback into account. Finally, a rule revision strategy for theory revision will be considered by identifying situations of over-generalisation or lack of coverage of existing rules. After having defined a formal learning framework, we begin development of our Android application as a means of implementing a learning rule system on a mobile phone for empirically evaluating the learning outcomes and their effectiveness.

1.3 Contributions

The overall vision behind this project is to assess the applicability of ILP for learning mobile user behaviours. To our knowledge, this is something that has never been explored before. The final outcome of this project is a fully tested learning framework and a complete Android application called *ULearn* (refer to Chapter 5 for more details) that enables rich data acquisition on a user's mobile phone, accumulates the data into an ILP learning file which is in turn sent to a back-end server for processing.
The outcome is learned rules which are displayed back to the user. These rules are circumstances

under which the user accepts or rejects calls. The user can then choose to enforce any of the rules i.e., allow the mobile phone to automatically accept or reject phone calls whenever all of the conditions of the enforced rule are satisfied. The application can also perform rule refinement by excluding pairs of conditions that are inappropriate to the user. The following list summarises the main contributions of this project:

- The definition of a mobile user behaviour domain.
- The definition of background knowledge and appropriate language bias in the context of accepting/rejecting calls.
- The combination of the two to form an ILP learning framework for learning mobile user behaviours, able to perform cross validation and ROC analysis.
- An application of our framework to a large existing dataset. Our framework is also applied on our own collected dataset, allowing for a detailed comparison of the two.
- A proof-of-concept Android application that collects user data on a continuous basis and prompts the results to the user.
- The adjustment of our language bias and TAL's learning parameters to determine the combination of which achieves maximum performance.

1.4 Report Structure

The report is further organised as follows:

- Chapter 2, outlines the relevant work in the fields of ILP and statistical machine learning methods that can be coupled with ILP, to some extent, successfully.
- Chapter 3 describes the main aspects of this project, that is, our formalism of a learning framework. It includes the definition of a domain appropriate to our problem and consequently the definition of the background knowledge and encoding of language bias. It explains how the framework is applied to learn certain rules and demonstrates how this framework has assisted in the automated process of extracting data into a learning file fed as input to TAL for the computation of rules.
- Chapter 4 demonstrates the applicability of our learning framework to a large, existing dataset, available online. A number of experiments are carried out to get the best results possible. We also explain the difficulties we encountered trying to fit the data to our needs and describe ways in which we have solved these difficulties.
- Chapter 5 focusses on our design and programming considerations which we make use during the implementation phase. We also present the specification and development of a graphical user interface application on the Android platform and its integration with the back-end capable of performing the learning tasks.
- Chapter 6 illustrates our results, how we evaluated them and the comparisons we made to ascertain whether learning user behaviours in terms of rules is helpful enough to minimise user intervention in pervasive devices.
- Lastly, Chapter 7 summarises the achievements of this project. It also highlights various limitations that we encountered along the way followed by suggestions of future work that can be undertaken to extend our system further.

2 Background

This chapter introduces the motivating work for this project. We examine the key ideas taken from machine learning that are necessary to understand the process we employ throughout. We begin with a brief introduction to machine learning and inductive learning followed by some essential notation, necessary to understand the nature of logic programming, and more specifically that of Inductive Logic Programming (ILP). We then take a closer look at TAL, the tool that will be used throughout this project, the background work related to it together with its performing search capabilities and its learning parameters. We consider a couple of statistical machine learning methods that we believe could provide a deeper understanding of the data and finally, we examine tools we make use of in this project.

2.1 Machine Learning

Machine Learning is an area of Artificial Intelligence that enables the development of techniques which allow machines to learn. Tom Mitchell [Mit97] defines learning to include any computer program that improves its performance at some task through experience. More formally, Mitchell defines a well-posed learning problem as follows:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

In order to design a learning system we must be able to represent the type of knowledge to be learned together with a learning mechanism. The type of knowledge to be learned is often problem dependent. Once we establish this type of knowledge, we need to define a function or a program that provides a mapping between an observed instance and the result of a new instance.

Hence we can define a machine problem as a problem consisting of:

1. A *target function* $V : D \rightarrow C$ representing the problem to be solved. D is the input space and can be thought of as the training data and unobserved instances that need to be classified. C is the output space and more informally the set of classes that each instance can be classified under. Examples include choosing the best legal move in a board game, identifying people, or preventing credit card fraud.
2. The *training data* (positive and negative) relevant to the definition of the target function. Each example is represented as a tuple (d, c) where $d \in D$ and $c \in C$.
3. A *learning algorithm* that finds a *hypothesis* that best fits the training data.

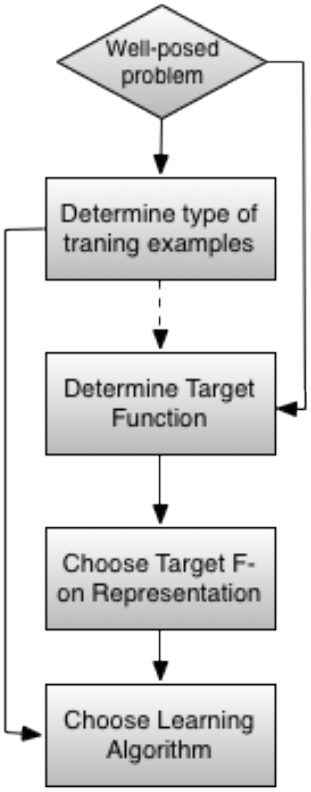


Figure 2.1: The design of a machine learning system [Pan10]

Figure 2.1, above, illustrates the process of designing a machine learning system. The two side arrows are only there to place emphasis on the relation between the learning problem with the target function and between the type of examples with the choice of a learning algorithm. As we can see, once a well-posed problem has been defined, we can determine an appropriate target function which represents the problem to be solved. Ideally we would want to find the exact function, V, but machine learning algorithms only learn an approximation, V'. The representation function V' to be learned should be as close an approximation of V as possible and should require a small amount of training data to be learned. Finally, the learning algorithm is selected based on the training examples, the number of classification types and the availability of a *learner*.

2.1.1 Inductive Machine Learning

Inductive inference within machine learning enables new instances of a hypothesis/concept to be classified accordingly. We make a strong assumption here in that the best hypothesis regarding unseen instances is the hypothesis that best fits the observed training data. Without *a priori* assumptions regarding the target concept, a system is unable to classify new, unseen instances. The training data itself are not sufficient enough to find a solution. Generally, *inductive learning* requires some form of prior assumptions or *inductive bias.* We can think of this as a policy by which the learning system will generalise beyond the observed training examples to infer the classification of new instances [Alp04].

More formally,

Given

- An inductive learning algorithm L
- A set of possible examples L_E
- A set of hypotheses L_H (or concepts)
- An unknown target function $f : E \mapsto Y$
- A set of training examples $E_h = \{(e_1, f(e_1), ..., (e_n, f(e_n)\}$ where each $e_i \in L_E$
- A loss function $loss(h, E)$

Find

- A hypothesis $h \in L_h$ that minimises the loss function.
- Y represents the domain of the classification and characterises the setting of the learning task. In *concept learning* $Y \in \{true, false\}$.

Once training is complete, we can use L to classify a new instance x_i. If $L(x_i, E_h)$ ($true$ or $false$) denotes the classification that L assigns to x_i, then we can describe this *inductive inference step* performed by L as follows:

$$(E_h \wedge x_i) \succ L(x_i, E_h)$$

where the notation $y \succ z$ denotes that z is inductively inferred from y. At times, the classification of $L(x_i, E_h)$ might not follow deductively from the training data E_h and the description of the new instance x_i. But if we have some form of background knowledge (assumptions), B, together with the training data then we can classify the new instance deductively. This is defined as inductive or semantic *bias* of L, as a set of additional assumptions. Inductive bias is defined formally as:

$$(B \wedge E_h \wedge x_i) \vdash L(x_i, E_h)$$

where the $y \vdash z$ denotes that z follows deductively from y (i.e., that z is provable from y).
The above formula can be thought of as a form of *inverse entailment* (more in Section 2.2) where the background knowledge together with the negative examples model the negation of the hypothesis. So the above formula can also, loosely, be written as follows:

$$B \wedge \overline{E} \models \overline{H}$$

Example 2.1.
Suppose that we see a swan for the first time and it is white. The next time we see another

swan, it is also white. Even though logically it does not follow that the next time we see a swan will be white, it's an assumption we make anyway. To be able to say that all "swans are white" we need an extra assumption of the sort, "all swans have the same colour".
From the above definitions, B = "all swans have the same colour", x_i is a swan with colour we haven't determined yet and D_c are all the swan observations in our training set. Then our learning algorithm L together with the training set D_c can deduce that the colour of the unobserved swan is white.

2.2 Inductive Logic Programming (ILP)

Inductive Logic Programming is concerned with the development of techniques and tools for relational data mining [Dze03].

This section introduces the different kinds of logical reasoning there exist and, briefly explain the concept of ILP before we introduce some essential notation and terminology required to understand the theory behind ILP.

2.2.1 Logical Reasoning

There are three methods of logical reasoning: *deduction, induction* and *abduction.*

Definition 2.1.
Given a rule $R : \beta \leftarrow \alpha$, read as α therefore β or β if α, then for each method:

- **Deduction:** determining β (i.e. $\alpha \wedge R \rightarrow \beta$) using R and its *antecedent* α to make a conclusion,
- **Induction:** determining rule R through learning by using a large number of examples of α and β and,
- **Abduction:** using rule R and its *consequent* β to assume that the *antecedent* α could explain β (i.e. $\beta \wedge R \rightarrow \alpha$).

From the above definitions, it follows that abduction can be regarded as the reverse of deduction.

Consider the following example from [KKT93]:

$$\begin{aligned} shoes\_are\_wet &\leftarrow grass\_is\_wet \wedge walked\_on\_grass \\ grass\_is\_wet &\leftarrow rained\_last\_night \\ grass\_is\_wet &\leftarrow sprinkler\_was\_on \end{aligned}$$

If we walked on the grass and we know that it rained last night, we can deduce that the grass is wet and our shoes will be wet. This is a form of *deductive reasoning.* On the other hand, if we knew that our shoes were wet, we can only be sure that either the grass was wet or that it rained last night. Trying to determine an explanation with respect to the observation(s) is *abductive reasoning.*

2.2.2 ILP in concept

Inductive Logic programming [DM94] is a machine learning technique which combines inductive learning and *logic programming.* It uses inductive learning to explore the *hypothesis search space*

and to make inference from past observations. ILP systems develop predicate descriptions from *examples* and *background knowledge*. Such systems generate rules that are able to distinguish between positive and negative examples. Both input and output are described as logic programs. ILP searches for a hypothesis that generalises the concept we are trying to learn from the training data and some background knowledge.

Before we give a formal definition of ILP, we first need to understand the necessary terms and notation in the context of logic programming.

2.2.3 Logic Programming: notation and terminology

Logic programs represent information about a particular relation in terms of rules and assertions. The simplest logic program is a fact called an *atom*. A very simple example representing the fact that "Helen is the daughter of Tom" can be written as follows:

$$daughter(helen, tom).$$

We can extend this simple logic program to include rules which represent properties relations. We can encode a rule which says that "Helen is a daughter of Tom, if Helen is female and Tom is the parent of Helen" as the following logical rule:

$$daughter(helen, tom) \leftarrow female(helen), parent(tom, helen)$$

The expression on the left-hand side of the left arrow is said to be the *head* of the rule and the terms on the righthand side are said to be the *body* of the rule. The procedural reading of the $daughter(X, Y)$ rule reduces to the problem of finding out if X is female, and if Y is the parent of X.

Such statements are called clauses. Before we continue, we define some logic programming notation.

Following Prolog conventions [SS86], *variables* are denoted by capital case letters (e.g. X, Y, Z), *function symbols* are denoted by a finite sequence of lower case letters (e.g. f, g, h, age), *predicate symbols* are denoted by a finite sequence of lower case letters (e.g. $p, q, married, greater\_than$), *propositional constants* are either $true$ *or* $false$ and *constants* are denoted by a finite sequence of letters (e.g. Bob, Louise). Predicates take on values of *True* or *False* and functions may take on any constant as their value. From the definitions of each symbol above, we can build up expressions which are defined below.

Definition 2.2. Term
A *term* is either a variable or a constant. If f is an n-ary function symbol and $t_1, ..., t_n$ are terms, then $f(t_1, ..., t_n)$ is a term (e.g., $Tom, X, age(Tom)$).

Definition 2.3. Atom
An *atom* contains no logical connectives and has no propositional structure. It is the simplest expression in logic (e.g., $p(X, f(3, Y)$ assuming that variables are universally quantified).

Definition 2.4. Literal
A *literal* is either an atom (e.g. A) or its negation (e.g. $\neg A$). A positive literal is an atom (e.g., $greater\_than(20, 5)$). A negative literal is the negation of an atom (e.g., $\neg greater\_than(5, 20)$).

Definition 2.5. Clause
A *clause* is a *formula* of the form

$$\forall X_1, ..., X_n(l_1 \vee ... \vee l_m)$$

where each l_i is a literal and $X_1, ..., X_n$ are all variables occurring in $l_1 \vee ... \vee l_m$.

In logic programming, clauses are usually written as the implication of a *head* from a *body*. In its simplest form, the *body* is a conjunction of positive and negative literals while the *head* is a single positive literal. The clause is written as follows:

$$h \leftarrow b_1, ..., b_m$$

Each element of the body of the rule is called a *condition* or an *antecedent*.

A *Horn clause* is a clause containing at most one positive literal. It's an expression of the form

$$H \leftarrow (L_1 \wedge ... \wedge L_n)$$

A *definite clause* is a Horn clause that has exactly one positive literal. A Horn clause without a positive literal is called a *goal* or a *headless rule* and is of the form $\leftarrow b_1, ..., b_n$.

A *normal clause* is a clause of the form $h \leftarrow b_1, ..., b_n$ where h is a positive literal and literals in the body can either be positive or negative.

A *normal logic program* is a finite set of normal clauses.

Definition 2.6. Ground term/atom/literal/formula
A *ground term* is a term that does not contain any variables. A *ground atom* is an atom that does not contain any variables. A *ground literal* is a literal constructed from a ground atom. A *ground formula* is a formula where no variable occurs in it.

Definition 2.7. Complementary atom/literal
If L is an atom, then its complement or complementary literal is given by $\neg L$. If L is a negative literal, of the form $\neg L$, its complement is given by L.

Definition 2.8. Theory
A *theory* consists of a finite set of conjunctive clauses (e.g., $\{C_1, ..., C_n\}$). A *logic program* is made up of a finite set of conjunctive *Horn clauses*.

A *substitution* is any function that replaces variable by terms. For example, the substitution $\{x/tom,\ y/5\}$ replaces the variable x by the term *tom* and replaces the variable y by the term 5. Given a substitution θ and a literal L we write $L\theta$ to denote the result of applying substitution θ to L. A *unifying substitution* for two literals L_1 and L_2 is any substitution θ such that $L_1\theta = L_2\theta$.

Atom and Clause Subsumption
Given a substitution $\theta = \{v_1/t_1, ..., v_n/t_n\}$ and formula F then $F\theta$ is formed by replacing every variable v_i in F by t_i. Atom A *subsumes* atom B, $A \succeq B$, iff there exists a substitution θ such that $A\theta = B$. Clause C subsumes clause D, $C \succeq D$, iff there exists a substitution θ such that $C\theta \subseteq D$.

Example 2.2, below, illustrates the concept of generalisation and subsumption.

Example 2.2.

Daffy Duck can fly	$can\_fly(daffy)$
All ducks can fly.	$can\_fly(x)$
	$can\_fly(x) \succeq can\_fly(daffy),\ \theta = \{x/daffy\}$

Models and Entailment

- Model: M is a model of formula (atom/literal/clause/theory) F iff F evaluates to True given M.
- Entailment: C is more general than D iff $C \models D$.
- Relative Entailment: C is more general than D w.r.t B iff $B, C \models D$.

Example 2.3, below, illustrates the concept of generalisation with an additional assumption. C is more general than D with respect to B since $B, C \models D$. At first this may seem counter-intuitive but this is generalisation as *relative entailment*.

Example 2.3.

C : Andreas lives in Nicosia	$lives(andreas, nicosia)$
D : Andreas lives in Cyprus	$lives(andreas, cyprus)$

$$\text{B} : lives(x, cyprus) \leftarrow lives(x, nicosia).$$

For additional semantic and syntactic definitions , we refer the reader to [LMR92, Mug08].

2.2.4 ILP definition

Given background knowledge B and a set of evidence E, $E = E^+ \wedge E^-$ (positive and negative evidence respectively), as logic programs, the aim of an ILP system is to find a hypothesis H such that the following conditions hold:

- **Prior Satisfiability:** $B \wedge E^- \not\models \square$
- **Consistency (Posterior Satisfiability):** $B \wedge H \wedge E^- \not\models \square$
- **Prior Necessity:** $B \not\models E^+$
- **Completeness (Posterior Sufficiency):** $B \wedge H \models E^+$

In practice these conditions are not always checked against the computed hypotheses.

2.2.5 ILP task

An ILP task [Lav98, CRL10] is formally defined as $< E, B, S >$, where E, B are defined as above and S is a set of clauses referred to as the language bias [Blo10]. The theory $H \in ILP < E, B, S >$, called hypothesis, is an inductive solution for $< E, B, S >$ if $H \subseteq S$, H is consistent with B and $B \cup H \models E$. $ILP < E, B, S >$ represents the set of all inductive solutions for the given task.

Example 2.4, below, illustrates the usage of each term and how a hypothesis can be constructed in an ILP setting. The set B represents the background knowledge, E^+ the positive training examples and E^- the negative training examples.

We know the relation *grandfather* in terms of $father(X, Y)$ and $parent(X, Y)$ and some additional facts all of which are logic programs in B, the background knowledge.

Example 2.4.

$$B = \begin{cases} grandfather(X,Y) \leftarrow father(X,Z),\ parent(Z,Y) \\ father(andy, dina) \leftarrow \\ mother(dina, helen) \leftarrow \\ mother(dina, andreas) \leftarrow \end{cases}$$

$$E^+ = \begin{cases} grandfather(andy, helen) \leftarrow \\ grandfather(andy, andreas) \leftarrow \end{cases} \qquad E^- = \begin{cases} grandfather(andreas, andy) \leftarrow \\ grandfather(helen, andreas) \leftarrow \end{cases}$$

Adding positive and negative examples concerning the relationship between grandfathers and their grandchildren, also as logic programs, allows us to infer a relationship such as the following:

$$H = parent(X,Y) \leftarrow mother(X,Y).$$

The hypothesis can be constructed by navigating the search space effectively and by adding relevant language bias to minimise the computational effort searching the hypothesis space. It is noteworthy to say that the resulting hypothesis is not a result of the background knowledge, B and the negative evidence, E^- (prior satisfiability) but instead is an explanation of E^+ with respect to B (posterior sufficiency). More details about navigating the search space can be found in Section 2.3.2.

2.2.6 Non-monotonic ILP

Normal ILP differs from *non-monotonic* ILP in that the former targets the problem of inductive construction of a general theory from examples and background knowledge and classifies new instances based on rules, whereas the latter aims at representing incomplete knowledge [Sak01].

In general, non-monotonic ILP does not guarantee that the newly-learned hypothesis preserves the consequence of the previously learned hypothesis. Referring back to Example 2.2, consider the default assumption that "birds can fly". Consequently, if an animal is known to be a bird and nothing else is known we can assume that it can fly. However, the default assumption will have to be retracted if we find out, at a later point in time, that this animal is a penguin. We know that penguins can't fly and therefore what was true before no longer holds anymore. The addition of this new fact caused the non-monotonic reasoner to retract the previous conclusion.

2.2.7 Evaluation and relevance

Learning techniques such as artificial neural networks, are said to mimic human learning by minimising the cost function using some form of gradient descent. However, unlike neural networks, ILP systems output rules which are easily understood by people and are suitable for automated analysis. We believe that the use of an ILP system will be most appropriate to

compute user behaviour rules from data which could then be easily processed and translated to the user for system validation.

However, presently, most ILP systems [DM94] require that all mode and type information be provided by the user but these could potentially be derived automatically from analysis of the background knowledge. In addition, ILP systems do not cater for noisy training data and they have rather limited facilities for handling numerical data [BM95] as opposed to other machine learning techniques. Finally, a large hypothesis space makes it intractable to calculate all acceptable hypotheses, even though acceptable solutions exist. TAL, an in-house tool developed at Imperial College has managed to overcome some of these limitations and is described in detail below.

2.3 Top-directed Abductive Learning (TAL)

Until recently, there have been no implementations of ILP systems which overcome the completeness problems but also the expressiveness and learning in the context of normal logic programs. Most systems limit the expressiveness of the language as well as the possible outcome of the learning. TAL [CRL10], a tool developed here at Imperial College London, is the first system that allows background theories and hypotheses to be normal logic programs.

In particular TAL handles negation during the learning process and has the ability to learn non-monotonic hypotheses. TAL is complete, meaning that it is able to find a solution if one exists. Moreover, it allows expressive language bias specifications, performs non-observational and multiple predicate learning and makes use of constraint solving techniques. It can also be used to perform theory revision, can use configurable heuristics in the search some of which are outlined in more detail in Section 3.6.

Its main difference from available ILP systems lies in its applicability to derive rules by handling large datasets and perform a complete search over a large search space and infinite domains. This project relies heavily on TAL and its suitability for this project is exactly for this reason; the ability to handle a very large domain whereby we can learn mobile user behaviours.

In order to understand how TAL works, it is necessary to introduce Abductive Logic Programming, the semantics and the theory behind it.

2.3.1 Abductive Logic Programming (ALP)

Abductive Logic Programming [KKT93] is concerned with computing a set of hypotheses that together with some knowledge, T, implies the given *observation*, E. It is a form of non-monotonic reasoning because each explanation is consistent with one state of a knowledge base and may become inconsistent with the addition of new information. In fact, multiple explanations is an important property of abductive reasoning. In contrast with ILP, ALP systems construct hypotheses in the form of ground literals of some predicates (*abducibles*) whereas ILP systems compute rules that are able to discriminate between positive and negative examples.

We can restrict abduction by making use of *integrity constraints*. Integrity constraints are useful when there is a perfectly valid explanation to reject irrelevant abducible explanations. Referring back to the example in Definition 2.1, if we add a constraint rule in theory T, which says that "it is impossible that it rained last night and the sprinkler was also on" (i.e. ←

$rained\_last\_night \wedge sprinkler\_was\_on$) then one of the previously candidate explanations would no longer be a candidate.

Definition 2.9. ALP task
Given a set of sentences T (a theory presentation), a sentence G and a set I of integrity constraints, then we can represent an abductive task as the problem of finding a set of sentences $\triangle$ (abductive explanation for G) such that:

- $T \cup \triangle \models G$,
- $T \cup \triangle$ satisfies I,
- $T \cup \triangle$ is consistent.

More formally, an ALP tasks is defined as $< g, T, A, I >$ where the theory T (a set of rules) is a normal logic program, A is a set of abducible facts, I is a set of integrity constraints and g is the *ground goal*[1].
$\triangle \in ALP < g, T, A, I >$ is an *abductive solution* for the ALP task, if $\triangle \subseteq A$, $T \cup \triangle$ is consistent, $T \cup \triangle \models g$ and $T \cup \triangle \models I$.
$ALP < g, T, A, I >$ denotes the set of all abductive solutions for the given ALP task.

Sometimes we wish to reduce the number of possible explanations. This can be achieved by imposing some restriction requirements. These requirements can be the following:

- An explanation must be *basic*. This effectively says that an explanation should not be explained by another. This can easily be avoided by not allowing abducible predicates to appear as heads of a rule R in theory T.
- An explanation must be *minimal*. This means that a possible explanation should not be subsumed by a smaller set.
- An explanation must satisfy all of the integrity constraints.

Example 2.5.

$$T = \begin{cases} citizen(x) \leftarrow born\_in(x, usa). \\ citizen(x) \leftarrow born\_in(x, y), y \neq usa, resident(x, usa), naturalised(x). \\ citizen(x) \leftarrow born\_in(x, y), y \neq usa, mother(y, x), citizen(y), registered(x). \\ mother(mary, john). \\ citizen(mary). \end{cases}$$

$$A = \begin{cases} born\_in(x, usa). \\ born\_in(x, y), \neq usa. \\ resident(x, usa). \\ naturalised(x). \\ registered(x). \end{cases}$$

$$I = \{\ false \leftarrow resident(john, usa).$$

$$G = \{\ citizen(john).$$

$$\triangle = \begin{cases} born\_(john, usa). \\ born\_(john, Y), Y \neq usa, registered(john). \end{cases}$$

[1]The goal does not necessarily need to be ground with the ALP procedure used in TAL

Example 2.5 [SSK+86, KC03], above, illustrates how using the following clauses together with the five abducible predicates, "is born in the USA", "is resident of the USA", "is naturalised", "is registered" and the integrity constraint $false \leftarrow resident(john, usa)$, the goal $citizen(john)$ has two abductive solutions, one of which is $born\_in(john, usa)$ and the other of which is $born\_in(john, Y), Y \neq usa, registered(john)$. The potential solution of becoming a citizen by residence and naturalisation fails because it violates the integrity constraint.

Evaluation and relevance

Abductive Logic Programming can be used to deal with incomplete theories and negation as failure offering wider expressiveness of the language and better outcome of the learning. In this project, abductive logic programming is used through TAL, discussed in detail in Section 2.3.2, which translates an ILP task into an ALP problem whose, in turn, solution is translated back into a solution of the original ILP problem. TAL relies on the semantics of the underlying abductive proof procedure [KKT93] and allows us to deal with negation and learn non-monotonic hypotheses.

2.3.2 TAL

We have seen in Section 2.2 and 2.3.1 how an ILP and an ALP task is defined and their set of solutions respectively. TAL translates an ILP task into an ALP one which is eventually translated back into a solution of the original problem. This allows TAL to handle negation within the learning process and learn non-monotonic hypotheses using the semantics of ALP. Since the space of possible solutions is very large there is a need to restrict the hypothesis search space.

TAL provides three ways to impose constraints on the space by defining:

1. Some background knowledge B.
2. Some languages bias, S.
3. Some integrity constraints, I.

The bias is useful in cases where we know how to structure our hypothesis in which case the hypothesis space will decrease considerably. The background knowledge is usually defined as a normal logic program whereas the language bias is specified using a set of mode declarations, M, to explicitly define the *head* and *body* declarations. Integrity constraints are used at times where pairs of literals are not meant to appear together in the body of a rule. This prevents TAL from searching that part of the search space and therefore speeding up the process of learning.

To limit the search space, TAL imposes an upper bound on the inductive solution by setting a mapping M, of mode declarations into a top theory $\top$. The set of M mode declarations is transformed into the top theory $\top$ as shown in Listing 2.1. With the theory defined, Prolog attempts to select clauses using the standard selection rule, in order of appearance [SS86].

Learning parameters

TAL uses a number of parameters to suit the learner's needs and these can be specified in the learning file using `option(name, value).` We list the most important ones here and a brief

explanation for each.

option(max_body_literals, N)
Specifies $N \in [0, inf[$ the maximum number of body literals allowed in a rule.

option(max_num_rules, N)
Specifies $N \in [1, inf[$ the maximum number of rules allowed in a solution.

option(max_depth, N)
Specifies $N \in\in [1, inf[$ the maximum depth of the proof procedure.

option(timeout, MS)
Specifies $MS \in \{false\} \cup [1, inf[$ the timeout for the learning task in milliseconds. $false$ disables the timeout.

option(number_of_solutions, N)
Specifies $N \in [1, inf[$ the number of solutions required before the task terminates. This parameter is used in the termination component of the heuristics, so the effect is dependent on the specific implementation.

option(strategy, ST)
Specifies $ST \in \{progol, breadth, ...\}$ the strategy used. Some of the strategies are explained in Chapter 3.

option(single_seed, B)
Specifies $B \in \{true, false\}$ if the single seed strategy is used or not.

option(single_seed_ratio, N)
Specifies $N \in [1, inf[$ the ratio for the single seed case. Intuitively each seed is taken to represent N examples.

option(solution_pool, N)
Specifies $N \in [1, inf[$ the number of solutions that are traced in the learning process. This is the number of solutions that appear in the final solution file and also the solutions that are considered in the post learning phase of the single seed case.

option(xvalidation_folds, N)
Specifies $N \in]-inf, inf[$ the number of folds used for cross validation. If less than 1 cross validation is not performed.

option(ic_check, B)
Specifies $B \in \{true, false\}$ if the integrity constraints are checked for each hypotheses. See Section 2.3.1 for details.

option(cover_loop, true)
Specifies $B \in \{true, false\}$ if the cover loop approach is to be used.

option(loop_treshold, N)
Specifies $N \in \{-1, 0\}$ and is used only if the cover loop approach is used. It specifies that solutions should only be considered if they score above N.

All of the options refer to the learning over a seed in the case of single seed learning.

The search in TAL is performed by an abductive logic programming system. Therefore each state in the derivation process corresponds to a full abductive state which includes a set of goals, a set of integrity constraints and a set of abducibles. Of these states, some correspond to the derivation of an inductive solution, in some cases already derived in other parts of the

tree and in other cases brand new. These states are called *partial hypothesis states*. All of these states are linked to a score as a tuple of (phs, s). phs is 0 for all the states that are not partial hypothesis states and set to a different number $phs_{strategy}$ otherwise. This number depends on the particular strategy adopted and is usually set to 1. s depends on the strategy and only changes in partial hypothesis states.

The reason why a pair is used is to allow strategies to evaluate the score only whenever the current open states are only partial hypothesis states, thus choosing what solution to refine whenever no abductive steps can be performed. In Chapter 3, we talk about the different heuristic strategies we opted and in Chapter 6 we explain how these parameters were altered to achieve maximum performance.

Top Theory and learning file

Example 2.6, below, demonstrates how TAL transforms the learning file in Listing 2.2, further below, consisting of the background knowledge, the mode declarations and the training examples, M, into top theory $\top$. Since no head of clause in B unifies with the positive examples, the derivation uses one of the rules defined in $\top$.

Example 2.6. Learning the family relation of $uncle(X, Y)$.

$$M = \begin{cases} m1 & : & \text{modeh(uncle(+male, +person))} \\ m2 & : & \text{modeh(brother(+person, +person))} \\ m3 & : & \text{modeb(male(+person)} \\ m4 & : & \text{modeb(parent(+person, +person)} \\ m5 & : & \text{modeb(brother(+person, -person))} \end{cases}$$

$$E^+ = \begin{cases} uncle(tim, ann) \leftarrow \\ uncle(bob, ann) \leftarrow \end{cases} \qquad E^- = \begin{cases} uncle(betty, ann) \leftarrow \\ uncle(susan, ann) \leftarrow \\ uncle(joyce, ann) \leftarrow \end{cases}$$

$$B = \begin{cases} brother(X, Y) \leftarrow male(X),\ parent(Z, X),\ parent(Z, Y),\ X = / = Y. \\ male(bob) \leftarrow \\ male(tim) \leftarrow \\ male(tom) \leftarrow \\ female(ann) \leftarrow \\ female(susan) \leftarrow \\ female(betty) \leftarrow \\ female(joyce) \leftarrow \\ parent(tom, mary) \leftarrow \\ parent(tom, bob) \leftarrow \\ parent(tom, tim) \leftarrow \\ parent(mary, ann) \leftarrow \end{cases}$$

Each head in $\top$ checks the typed arguments defined, associates itself with a rule id and attempts to unify with one of of the *body* literals, using the standard Prolog selection rule. For example, $m1$ is the head declaration for $uncle(+male, +person)$. Consequently in $\top$, the first clause is

that of $uncle(X, Y)$ and is defined by unifying X to *male*, Y to *person*, and the *body* literal to one of the three possible clauses.

The selection of the *body* literal from the rule results in three derivation branches in the search tree; one for each of the three *body* clauses whose head unifies with it. The derivation process continues until a rule is computed which is consistent with all the positive and negative examples. The rule computed under the top theory, shown in Listing 2.1, below, is as follows:

$$uncle(X, Y) \leftarrow male(X), parent(Z, Y), brother(X, Z))$$

```
1  uncle(X,Y) ← male(X), prule(RId, [(m1, [], [])]),
2      rule_id(RId), person(Y),
3      body(RId, [X,Y], [(m1, [], [])]).
4
5  brother(X,Y) ← person(X), person(Y),
6      prule(RId, [(m2, [], [])]),
7      rule_id(RId), [X,Y], [(m2, [], [])]).
8
9  body(RId, Inputs, Rule) ← append(Rule, [(m3, [], Links)], NRule),
10     prule(RId, NRule), link_variables(X, Inputs, Links),
11     male(X), person(X), body(RId, Inputs, NRule).
12
13 body(RId, Inputs, Rule) ← append(Rule, [(m4, [], Links)], NRule),
14     prule(RId, NRule), link_variables([X,Y], Inputs, Links),
15     person(X), person(Y), body(RId, Inputs, NRule).
16
17 body(RId, Inputs, Rule) ← append(Rule, [(m5, [], links)], NRule),
18     prule(RId, NRule), links_variable([X,Y], Inputs, Links),
19     person(X), person(Y), append(Inputs, Y, Outputs),
20     body(RId, Outputs, NRule).
```

Listing 2.1: Top Theory $\top$ for the *uncle* example

Listing 2.2, below, shows how we can encode a learning task in TAL using Prolog to learn properties relations that were previously unknown. The file consists of a set of `options`, a set of mode declarations (head and body), our background knowledge and a set of training examples. Since this is a toy problem, the default strategy was used, in this case Progol. The values for all the other options were set to their default values as they wouldn't impact the solution of this problem.

```
1
2  /*  Options  */
3
4  option(max_body_literals, 3).
5  option(max_num_rules, 1).
6  option(max_depth, 100).
7
8  /* Mode declarations */
9
10 modeh(uncle(+male, +person)).
11 modeb(male(+person)).
12 modeb(parent(+person, +person)).
13 modeb(brother(+person, -person)).
14
15 /*  Background knowledge  */
16
17 brother(X,Y):- male(X), parent(Z, X), parent(Z, Y), X =/= Y.
```

```
18
19 male(bob).
20 male(tim).
21 male(tom).
22
23 female(ann).
24 female(susan).
25 female(betty).
26 female(joyce).
27
28 parent(tom, mary).
29 parent(tom, tim).
30 parent(mary, ann).
31 parent(tom, bob).
32
33 person(_).
34
35 /* Examples - Both positive and negative */
36
37 example(uncle(tim, ann), 1).
38 example(uncle(bob, ann), 1).
39 example(uncle(betty, ann), -1).
40 example(uncle(susan, ann), -1).
41 example(uncle(joyce, susan), -1).
```

Listing 2.2: The learning file for learning the relation *uncle*

2.4 Statistical Machine Learning

In this section, we present a subset of machine learning techniques that are concerned with applying information theory and statistics to obtain a quantitative output [HTF09] that we wish to predict based on a set of *features*. We have a *training set* of data with which we can measure the outcome for a set of objects. With the data, a prediction model, *the learner*, can be constructed which predicts the outcome for new instances.

2.4.1 Association Rule Learning

Association rule learning [AIS93] can be used as a data mining technique to infer previously unknown relations between variables in large datasets. A famous example of such an *association rule* is the following supermarket purchase rule:

$$\{bread, butter\} \rightarrow \{milk\}$$

This says that the people who buy bread and butter at a supermarket, will also buy milk. Bread and butter is said to be the *antecedent* and the milk is said to be the *consequent*.

Agrawal et al. define this problem in terms of a *transaction* set, $T = \{t_1, t_2, ..., t_n\}$ called the *database* and a set of binary attributes, $I = \{\iota_1, \iota_2, ..., \iota_n\}$, called *items*. Each transaction, t, within the database is represented as a binary vector, with $t[k] = 1$ if t satisfies item I_k, and $t[k] = 0$ otherwise. If X is a set of items in I, then t *satisfies* X if for all items I_k in X, $t[k] = 1$.

A rule is defined as an implication of the form $X \Longrightarrow Y_j$ where X is a set of some items in I and Y_j is a single item in I that is not present in X. More formally, this is defined as $X, Y_j \subseteq I$ and

$X\hat{Y} = \emptyset$. Each rule is accompanied with a confidence factor c which signifies the probability of the rule being correct. The rule $X \Longrightarrow Y_j$ is satisfied in the set of transactions T with the confidence factor $0 \leq c \leq 1$ iff at least $c\%$ of transactions in T that satisfy X also satisfy I_j.

Example 2.7, below, shows how we can encode a rule when dealing with the Reality Mining dataset.

Example 2.7.
A set of items could be $I = \{video, game, phone, signal, nosignal\}$ and example rules inferred from D could be $\{video, game\} \Longrightarrow \{nosignal\}$ and $\{phone, music\} \Longrightarrow \{signal\}$.

The algorithm

Although there exists more than one algorithms for mining association rules like Eclat [Zak00] and FP-Growth [HPY00] we maintain our focus on the Apriori [AS94] algorithm which is the best known algorithm for rule learning.

In plain English, the algorithm looks for frequent item-sets. These are item-sets whose support count is greater than a minimum support threshold (N). For each item in the database, we calculate its support and check to see whether it qualifies. If it passes, then we iteratively continue to find frequent item-sets with cardinality from 1 to k (k-item-set). Termination of the algorithm occurs where there are no more item-sets having support N.

Example 2.8, below, illustrates how the algorithm is used in practice with a simple example.

Example 2.8. Algorithm demonstration
Consider a database of transactions consisting of the sets $\{1,2,3,4\}, \{1,2\}, \{2,3,4\}, \{2,3\}, \{1,2,4\}, \{3,4\}$ and $\{2,4\}$ with minimum support count required 4 and confidence factor 70%. We begin by creating a co-occurence (frequency) table, called the support, of each member item as follows:

$$1 \rightarrow 3$$
$$2 \rightarrow 6$$
$$3 \rightarrow 4$$
$$4 \rightarrow 5$$

where $A \rightarrow B$ denotes item A having support B. We compare the candidate support with the minimum support count and remove candidates which do not qualify. In our case all item-sets pass. Our next step is to generate a 2-item-set frequency table and again compare candidate support with minimum support. In our example, the support for 2-item-sets is as follows:

$$\{1,2\} \rightarrow 3$$
$$\{2,3\} \rightarrow 3$$
$$\{2,4\} \rightarrow 4$$
$$\{3,4\} \rightarrow 3$$

In this our example only the 2-item-set $\{2,4\}$ qualifies. If more than one qualified then we would have to repeat iteratively to find frequent item-sets with cardinality from 1 to k.

$$\{1,2,3\} \rightarrow 1$$
$$\{2,3\} \rightarrow 3$$
$$\{2,4\} \rightarrow 4$$

$$\{3,4\} \rightarrow 3$$

To generate association rules from frequent item-sets we need take all of their non-empty subsets. These are $\{2\}$ and $\{4\}$. The resulting association rules are shown below each listed with their confidence:

- **R1:** $\{2\} \rightarrow \{4\}$ with confidence $\frac{Support(\{2,4\})}{Support(\{4\})} = \frac{4}{6} = 67\%$
- **R2:** $\{4\} \rightarrow \{2\}$ with confidence $\frac{Support(\{4,2\})}{Support(\{4\})} = \frac{4}{5} = 80\%$

Since our confidence factor is 70%, R1 is rejected and R2 is accepted.

Constraints

When generating rules there are some constraints that we need to adhere to:

- **Syntactic Constraints:** These constraints impose certain restrictions on the items that can appear in a rule. For example we may be interested in specific items appearing in the *antecedent* or the *consequent*. In an ILP setting this can be thought of as the language bias which sets an upper bound on the search space.
- **Support Constraints:** These constraints involve the number of transactions in T that support a rule.

Evaluation and relevance

Association rule learning makes it difficult to find frequent item-sets in a dataset because it involves searching for all the possible item combinations. This is the power set I and has size $2^n - 1$. The power set grows exponentially and so an efficient search algorithm is important to allow for the discovery of all the frequent item-sets. In addition, searching for many possible associations, for sets of items that appear to be associated, carries the risk of finding many associations which do not provide any additional value to the data because their frequency in the data is so small.

Nevertheless, this technique can provide us with strong rules discovered in datasets using different measures of interestingness. We can use these rules as background knowledge in TAL to potentially improve the ILP learning accuracy.

2.4.2 Decision Tree Learning

Decision Tree learning is a method for approximating discrete classification functions using a tree-based representation. We can make use of this method to inductively infer [Mit97] unknown relations within our dataset. The trees can also be represented as sets of *if-then* rules, making the transition to an ILP setting almost seamless.

A learned decision tree is able to classify a new instance by sorting it down the tree, beginning from the root to some leaf node which is able to classify the new instance. Every node in the tree tests some attribute value of each new instance, and the branches of each node represent a possible value for that attribute.

Figure 2.2, below, depicts a very simple decision tree for the concept *hasSignal*. In the tree we can see that there are two attributes, namely audio and vision. The former takes the values *phone, withVision, music* and the latter the values *video, game*. An example is then classified by sorting it through the tree to the appropriate leaf node, then returning the classification associated with this leaf, in our case either of $\{yes, no\}$. The tree classifies whether mobile phones with instances of these two attributes have reception or not.

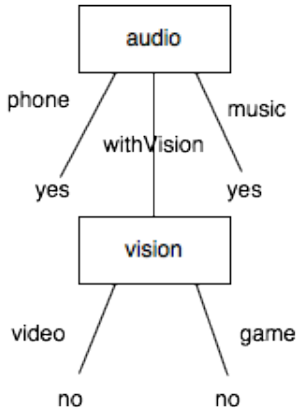


Figure 2.2: A simple decision tree for the concept *hasSignal*

The algorithm

One of the most commonly used algorithms for decision tree learning is the ID3 algorithm which employs a top-down greedy search through the space of possible solutions.

The general approach to implementing the algorithm is as follows:

1. Perform a statistical test of each attribute to determine how well it classifies the training examples when considered alone. This is often called the *information gain* and determines how well it classifies the training examples.
2. Select the attribute that performs best (the one with the highest information gain) and use it as the root of the tree.
3. Determine the node for each branch by sorting the training examples with respect to the value related to the current branch and recurse to repeat the process in steps 1 and 2 respectively.

To calculate the information gain precisely, we need to calculate its *entropy*, that characterises how pure a collection of examples is. The higher the entropy value of this collection the higher its impurity. The entropy of a collection of examples, S is calculated, relative to a *n-wise* classification, as follows:

$$Entropy(S) = \sum_{i=1}^{n} -p(x_i) log_2 p(x_i)$$

where $p(x_i)$ is the proportion of S belonging to class i. Now that we can calculate the entropy of a collection of examples, we can define a measure which signifies the effectiveness of an attribute

in classifying the training data. In other words, how well alone, an attribute can classify the data. The information gain, is the reduction in entropy caused by partitioning S according to an attribute A. Formally, the information gain can be defined, and used to calculate the value for each of attributes as follows:

$$IG(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

where $Values(A)$ is the set of all possible values for attribute A, and S_v is the subset of S for which attribute A has value v.

Example 2.9. An example calculation[2]
Suppose we are working with a set of examples, $S = \{s1, s2, s3, s4\}$ such that s_1 is classified (binary classification) as positive and the rest as negative. We want to calculate the information gain of an attribute A which can take values $\{v1, v2, v3\}$. Let $s_1 ... s_4$ take the following values:

- s_1 takes value $v2$ for A
- s_2 takes value $v2$ for A
- s_3 takes value $v3$ for A
- s_4 takes value $v1$ for A

To work out the information gain for A relative to S (i.e. $IG(A, S)$), we first need to calculate the entropy of S.

$$Entropy(S) = -\frac{1}{4} log_2(\frac{1}{4}) - \frac{3}{4} log_2(\frac{3}{4)}) = 0.5 + 0.311 = 0.811$$

Next, we need to calculate the weighted $Entropy(S_v)$ for each value of $v = v1, v2, v3, v4$, noting that the weighting involves multiplying by $\frac{|S_{vi}|}{|S|}$ where S_v is the set of examples from S which have value v for attribute A.

$$\frac{|S_{v1}|}{|S|} * Entropy(S_{v1}) = \frac{1}{4} * (-\frac{0}{1}) log_2(\frac{0}{1}) - \frac{1}{1} log_2(\frac{1}{1}) = 0$$

$$\frac{|S_{v2|}}{|S|} * Entropy(S_{v2}) = \frac{2}{4} * (-\frac{1}{2}) log_2(\frac{1}{2}) - \frac{1}{2} log_2(\frac{1}{2}) = \frac{1}{2}$$

$$\frac{|S_{v3}|}{|S|} * Entropy(S_{v3}) = \frac{1}{4} * (-\frac{0}{1}) log_2(\frac{0}{1}) - \frac{1}{1} log_2(\frac{1}{1}) = 0$$

We now have all the required values to apply the information gain formula, defined further above, as follows:

$$IG(S, A) = 0.811 - (0 + \frac{1}{2} + 0) = 0.311$$

Evaluation and relevance

Decision trees are suitable for discrete classification functions. For this reason, the ID3 learning algorithm will never search for an incomplete hypothesis space. The algorithm uses statistical properties to make decisions on training examples which makes the resulting search much less

[2]Adopted from: http://www.doc.ic.ac.uk/~sgc/teaching/v231/lecture11.html

sensitive to errors. On the other hand, every decision tree determines a single hypothesis and not a set of hypotheses. The algorithm is unable to find more than one hypothesis consistent with the data which means there could be more than one decision tree classifying the data correctly.

2.5 Tools

2.5.1 Waikato Environment for Knowledge Analysis (WEKA)

WEKA[3] [HFH+09] is a collection of machine learning algorithms for data mining tasks. A programmer can either apply the algorithms directly to a dataset suited to WEKA, adhering to the *.arff* format, or directly from Java through the use of their API[4] . WEKA contains tools for data pre-processing, classification, regression, clustering, association rules, and visualisation. It is also well-suited for developing new machine learning schemes. Our choice for WEKA was mainly based on two reasons: (1) it implements a wide range of popular machine learning algorithms suited to our needs and (2) the scope of our project relies upon an investigation of machine learning algorithms suited to our problem rather than their detailed and pedantic implementation.

For these two reasons, we think it is the best choice available. The main application is made of up of panels, the most important being:

- The *pre-processing* panel responsible for importing data. The application can either take a flat file as input or access a database query using Java Database Connectivity (JDBC) and process the result.
- The *classify* panel which is the most important panel. It enables us to apply classification and regression algorithms to the corresponding dataset and estimate the accuracy of the inferred model.
- The *associate* panel provides access to association rule learners which attempt to identify relationships between attributes.
- The *cluster* panel which gives access to clustering techniques.
- The *select* panel which provides methods for identifying the most informative attributes.

2.5.2 The Android Software Development Kit (SDK)

In this project, we make use of the Android SDK and its APIs as proof of concept. We choose the Android Platform as opposed to the iOS SDK because of the learning curve required to become experienced in Objective-C. The Android SDK provides the necessary tools and APIs to allow the development of applications on the Android platform using Java as the main programming language. We put our learning framework into practice by developing our own data collection application. Figure 2.3, on the following page depicts the Android architecture. Each layer uses the services provided by the layers below it.

Some of the SDK's main features/layers our application will use are as follows:

[3]WEKA: http://www.cs.waikato.ac.nz/ml/weka/
[4]API: Application Programming Interface

- **Android Runtime:** This layer, includes the Dalvik Virtual Machine and the core Java libraries. These slightly differ from the Java Standard Edition.
- **Activity Manager:** The Activity Manager controls the life cycle of applications and maintains the so-called activity stack for user navigation.
- **Notification Manager:** The Android Notification Manager is responsible for alerting/notifying the user for any event/appointment/alert.
- **Location Manager:** The Android Location Manager is responsible for retrieving the user's location.
- **Resource Manager:** The Android Resource Manager is responsible for loading up static resources such as images and layout.
- **SQLite:** The Android includes the lightweight SQLite database engine. It is the most widely used database engine for mobile devices and compact applications. It is most appropriately used for structured data storage.
- **Bluetooth, EDGE, 3G and WiFi:** An Android device can connect to the internet with a number of different connectivity components as provided by the Linux Kernel.
- **GPS, Light sensor:** These are hardware dependent components, meaning that they are not available in every device.

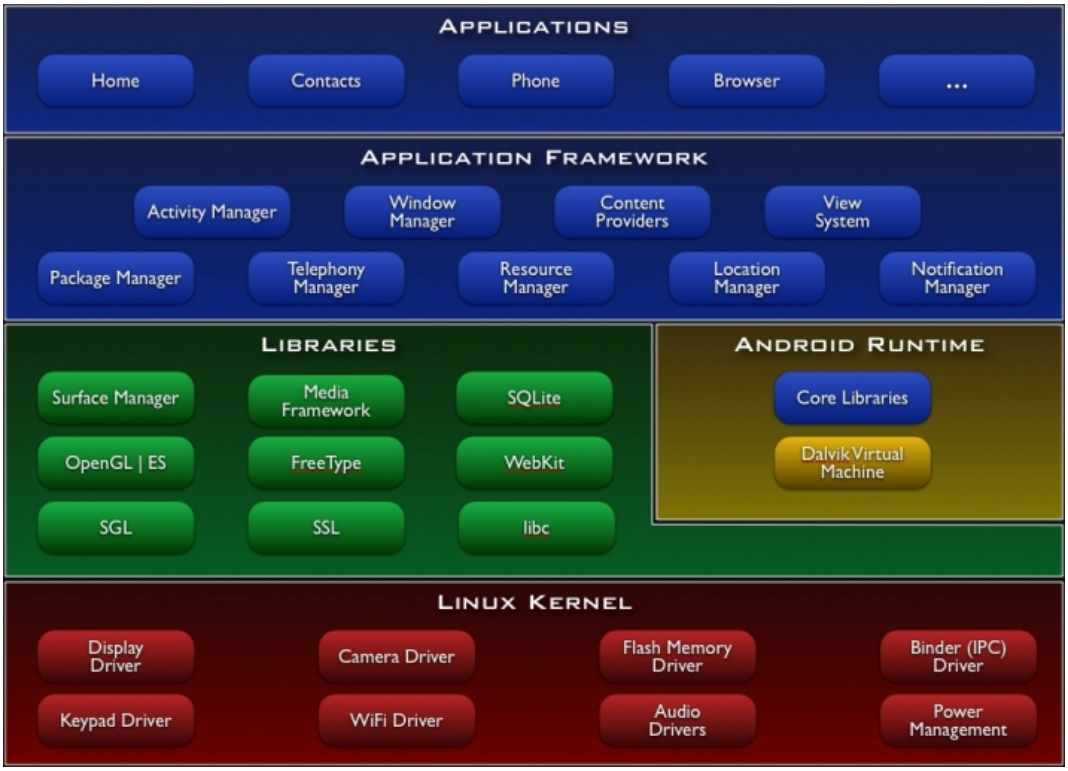


Figure 2.3: The Android Architecture [Goo11]

Activities

An *activity* represents a single screen with a user interface. Each application contains a finite set of activities which can make up the flow of the application. Even though activities work together to form a cohesive user experience, each one is independent of the others. A *ListActivity* is a special kind of activity that allows representation of lists in a user friendly manner.

Services

A *service* is a component that runs in the background to perform long-running operations or to perform work for remote processes. It has a similar notion to that of Thread or a Unix daemon. A service does not provide a user interface. Taking into account our own application, a service might log the user's location while the user is in a different application or it might fetch results over the network without blocking user interaction within an activity.

Broadcast Receivers

A *broadcast receiver* is a component that responds to system-wide broadcast announcements. Most broadcasts originate from the system, for example a broadcast announcing that there is an incoming phone call or that the device is about to be switched off. Broadcast receivers do not display a user interface per se, however using the *Notification Manager* they may create a status bar to alert the user that an event has occurred.

For more information on the Android platform, we refer the reader to [Bur08, Goo11].

2.6 Summary

We have covered a range of useful techniques that will prove useful during the design and implementation phases of our system. Asides from a quick introduction to logic programming notation and a thorough overview of inductive and abductive logic programming we have additionally focussed our interest on machine learning techniques that we believe can be coupled with ILP such as decision tree and association rule learning. To form a complete image, we have provided the reader with a thorough examination of TAL; the learning tool that forms the basis of our project.

3

A Learning Framework

In this chapter we propose and define a formal framework for learning mobile user behaviours. We give a brief overview of the steps we carry out in the project, followed by a formal definition of our domain, the background knowledge and the modelling components suitable to our problem. We explain how an appropriate language bias is formalised, what constitutes our training examples and we additionally show how the data are extracted and formatted as required by Prolog and TAL. We discuss the heuristic strategies most appropriate to our problem and how the cover loop approach performs considerably better. Further, we describe how our results are cross validated enabling us to perform a full receiver operating characteristic analysis followed by a discussion on rule refinement and revision. Lastly, we discuss whether the use of statistical machine learning has enabled us to improve learning.

3.1 Overview

Figure 3.1, below, illustrates diagrammatically the steps and processes we employ in the project. We first start off with a phase of domain modelling to allow for the definition of background knowledge and associated language bias.

We have already seen in Section 2.2 how ILP systems can be used to infer previously unknown relations. In order to be able to achieve this, we need to first understand the type of data we have, what there is available, how this information is accessed and consequently what we can learn through this type of information. More specifically, we cover how the data is defined and structured and what information can be included for each rule we wish to learn. We examine the possibility whereby this process is fully automated to speed up the entire process.

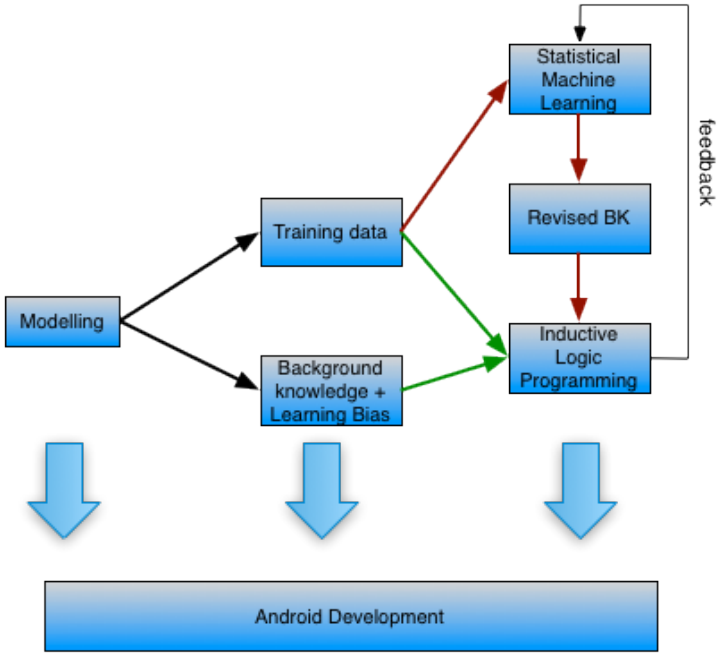


Figure 3.1: A high-level view of the project

For any learning task, we look for patterns in sets of data. In general, we have a large set of data that allows us to solve the learning problem. The data are typically in the form of a relational database (or other forms) which consists of several tables where each table corresponds to some entity.

Most data-mining techniques assume that the data are available in one table and refer to its columns as a set of attributes. For this reason, pre-processing of multiple tables is essential to form an integrated table before learning algorithms can be applied. The result is usually a propositional rule that classifies the data according to the class attribute. However, this integration usually results in loss of information which may introduce noise to the data.

In an ILP task, we are concerned with transforming this set of data into logic programs. Apart from dealing with data stored in multiple tables directly, ILP has the ability to infer relationships from other relationships by taking into account background knowledge given as a logic program. As an example, if we know how to encode the relations mother and father, we can easily define the relations grandmother, grandfather, brother and sister.

If we consider Table 3.1, which lists attributes of people classified as either rich or not rich, then we can automatically extract two rules. The first being a propositional rule and the second being a relational rule.

- **Propositional rule:** if hair == blond && smart == no then rich = Yes
- **Relational rule:** `isRich(X, Hair, Body, Pose, Smart)` ← `married_to(X, Y)` ∧ `people(Y, H, B, P, S, Ir)` ∧ `H = blond` ∧ `S = no.`

id	hair	body	pose	smart	**isRich**
1	blond	thin	arrogant	no	yes
2	brown	thin	natural	yes	no
3	blond	plump	goofy	no	yes
4	black	thin	arrogant	no	yes
5	blond	plump	natural	yes	no
...	...	...	...	...	...

Spouse1	Spouse2
1	2
2	1
3	4
4	5
...	...
...	...

Table 3.1: A relational database with two tables; People and MarriedTo

The applicability of ILP to our problem is mainly based on three things; (a) our data are not being integrated into a single table, (b) we have a very large domain and (c) we don't have a formal set of attributes to choose from. However, the usability of ILP is limited because of the difficulty in transforming the data in an automated manner over time. One of our main aims is to develop a generic and flexible framework that requires a minimal amount of change. We aim to address this issue, in the following sections.

3.2 Domain of discourse

To be able to learn mobile user behaviour rules through inductive reasoning, we first of all need to define a domain over which we are going to learn. The domain definition is necessary since writing a program in Prolog consists of listing out the facts and rules of the system which will be used to construct the domain for the problem.

The possibilities are endless. However, we can limit the domain by considering important aspects of user behaviour in general. For example, the location of the user at a certain time-point is important as it can be used in path prediction. The user's location can be inferred from GPS[1] coordinates or from the area and cell tower id the user's device is currently registered to. Using Bluetooth enabled devices we can infer proximity from repeated Bluetooth scans.

The project has mainly focussed on the definition of a domain for the computation of the rule of accepting/rejecting phone calls. This is by no means an exclusive domain, and the majority of the rules/facts can also be used for the computation of other user behaviour rules.

We refer to these behaviours as *events* since they all have a temporal element within them. Events of particular interest to us, are communication events. These are phone calls that the user accepts or rejects. For each phone call, we are particularly interested in the time-point the event takes place, its duration, the direction of the call (i.e., whether it is an incoming or an outgoing call), the contact name/id/number.

There are also other events which we, in our opinion, believe are factors in the decision-making process of accepting/rejecting calls. We define these events as follows:

(a) **Activity events:** These represent the activity of the user on the phone.

(b) **Application usage events:** These indicate the time the user is using a particular application on the phone.

(c) **Battery charging events:** These events signify the period in which the phone's battery is being charged.

[1]GPS: Global Positioning System

(d) **Device events:** Indication that Bluetooth enabled devices are scanned in the user's vicinity.

(e) **Location events:** The user's transition from one location to another.

3.3 Background Knowledge

The element of background knowledge to construct explanations is a distinctive feature of ILP. The work carried out in [QCJ93] for learning list processing logic programs suggests that the performance of an ILP system is sensitive to the type and amount of background knowledge provided. Therefore, for a large problem domain such as ours, irrelevant background knowledge may not only, unnecessarily, take more time to compute a hypothesis due to a large search space, but it may also hinder the ILP's system ability to guide the search towards a converging solution.

In Prolog, the background knowledge usually consists of *facts* and *rules*. Facts are predicate expressions that make a declarative statement about the problem domain. These are usually defined once, and for this reason they are often termed as *static* domain-specific properties. There are also *dynamic* domain-specific properties which change over time.

For our framework, we consider both type of properties. Static properties refer to some fact which remains intact throughout the learning process, such as the level of brightness which is a number between 0 and 255. On the other hand, dynamic properties are used when incremental learning is involved and refer to facts which change periodically, like newly seen device names or new contact numbers.

Rules are predicate expressions that use the logical implication (**:-**) to describe a relationship among facts. Most of our background knowledge rules in our framework are concerned with modelling relations with respect to time. This contradicts with our definition above, since our rules do not describe relationships among domain-restricted facts. The next sections describe how we construct and encode rules as well as facts to allow TAL to converge to a solution faster.

3.3.1 Modelling date and time

The concept of time plays an unprecedented role in defining behaviour. All user behaviours occur in terms of time. We answer phone calls at a particular point in time, we are at location Y at time-point X and so on. This is why all of our events are modelled in terms of a timestamp span. A timestamp span has a start-time and an end-time. In order to introduce time in our events, we start by defining basic elements of date and time. These are:

- `year/1`, `month/1` and `day_of_month/1`.
- `hour/1`, `minute/1` and `second/1`.
- `date/1`, `timex/1`.

We use `timex` instead of `time` because some Prolog solvers conflict with the word time. Listing 3.1 illustrates how they are encoded in Prolog. Each of them accepts a range of values, allowing Prolog to succeed only when those values are satisfied.

For date and time, their respective arguments are checked against their corresponding literals in the body of each rule.

```
year(Y):-
    Y in 2000..2011.

month(M):-
    M in 1..12.

day_of_month(D):-
    D in 1..31.

date([D, M, Y]):-
    year(Y),
    month(M),
    day_of_month(D).

hour(H):-
    H in 0..23.

minute(M):-
    M in 0..59.

second(S):-
    minute(S).

timex([H, M]):-
    hour(H),
    minute(M).
```

Listing 3.1: Elements of date and time in the background knowledge.

Date and time relations

The above definitions are only preliminary to enable us to define date and time relations. Date relations provide information about two time events. For instance, we might be interested in a date, say [D, M, Y], that is before or after another date, say [D1, M1, Y1]. For dates, there are three possibilities for each relation, each modelled as a different logic rule.

One where the year Y is less that year Y1, another where Y and Y1 are the same but month M is less than M1, and finally, in the case where Y, Y1 are the same and M and M1 are also the same but day D is less than D1. Similarly, we do the same for checking whether one date is after another. For example, `date_before([ 5, 6, 2011], [5, 7, 2011])` succeeds, whereas `date_after([ 5, 6, 2011], [5, 7, 2011])` fails.

In the same way, time relations provide information about two distinct time events. For time, the are two possibilities for each relation and constructed by the same argument explained above. For example, `time_before([10, 55], [23, 52])` succeeds, whereas `time_after([10, 55], [23, 52])` fails.

Having defined these, we can introduce two concepts in the context of date and time. That of before, and that of after. Listing 3.2, below, shows the final rules that we compute using the relation definitions from above. We achieve this by checking whether the two dates are either before or after the other. If they are the same, we determine whether their respective times are before or after one other.

```
before([D,M,Y],T,[D1,M1,Y1],T2):-
    timex(T),
    timex(T2),
   date_before([D,M,Y],[D1,M1,Y1]).

before([D,M,Y],T,[D,M,Y],T2):-
    date([D,M,Y]),
    timex_before(T,T2).

after([D,M,Y],T,[D1,M1,Y1], T1):-
    timex(T),
    timex(T1),
```

```
    date_after([D,M,Y],[D1,M1,Y1]).

after([D,M,Y],T,[D,M,Y], T1):-
                                        date([D,M,Y]),
                                        timex_after(T,T1).
```

Listing 3.2: Time relations

3.3.2 Duration spans

A duration span is a term we use that defines specific knowledge we can extract from our data. All of them have a starting point and an ending point, both defined in terms of time and date.

In terms of background knowledge we have specified the following:

- **Activity span:** Corresponds to the period whereby the user is doing something on the phone.
- **Application span:** Denotes the period as well as the type of application a user is using a particular mobile application. Since each application is typed, our background knowledge also consists each type of application as a fact.
- **Device span:** Represents the period for which a device is present in the user's vicinity. Each device seen by the phone is typed and is also included as background knowledge.
- **Cell span:** Indicates the period for which the user is at a certain location. All locations traversed by the user are also typed and are also used in the background knowledge.
- **On span:** Shows the period that the phone is switched on.

As explained before, there are two types of background knowledge we have. Unless we perform learning once, all spans and their respective types form the dynamic component of background knowledge which is accumulated on each data collection session. A full definition of our background knowledge definitions can be found in Appendix A.

3.4 Language bias

We have previously described the definition of language bias in Chapter 2. To remind ourselves, in short, the language bias is the structure/language of the hypotheses that a typical ILP system may return. In simple words, the bias defines a valid combination of each head of the rule with conditions that we get for the body of each rule. In TAL, the language bias is encoded as first-order logic clauses and written in terms of mode declarations which we discuss below.

3.4.1 Mode declarations

Mode declarations [Mug95] serve as a convenient way of constraining the hypothesis space and are based on the notions of types and modes. The arguments of the predicates can be typed and have an input (+), output (-) or constant mode (#). In this way, every variable that occurs

as input argument for a literal in the body must occur either as an output argument or as an input argument in the head literal of a clause.

There are two types of mode declarations. The head declarations and the body declarations. In TAL, these are identified as `modeh` and `modeb` respectively.

In TAL and our context, `modeb(weekend(+date), [name(weekend)])` is a mode declaration that is allowed to appear in the body of a rule and `modeh(accept(+date, +time, +contact))` is a mode declaration that can appear in the head of a rule. Since that is our only head declaration, then we only really have one type of rule defining the single predicate head, but with varying conditions in the body depending on our examples and background knowledge. The full list of mode declarations can be found in Appendix A.

3.4.2 Refining date and time

Having defined a formal model of date and time in Section 3.3 we can now define a more high-level view of date and time. For instance, a date can be defined in terms of a weekday or weekend. Time, on the other hand, can be defined in terms of certain time periods during the day, namely, morning, afternoon and evening.

Listing 3.3 shows how we categorise date and time into a more meaningful representation. For `weekday/1` and `weekend/1`, we first need to check that each argument in the two predicates is a list representing a date. Using our `day_of_week/4` rule, we can use these arguments as input. the rule will output `X`, which denotes a weekday if it's in the range of 1 and 5, and a 0 if the date is a Sunday or 6 if it's a Saturday. For times, we check if the list arguments are valid time points and then we check the range of hours that define `morning/1`, `afternoon/1` and `evening/1` respectively. Finally, we model the notion of answering calls before or after a particular hour of the day with the `time_before_h/2` and `time_after_h/2` predicates (not shown, due to layout purposes).

```
weekday([D, M, Y]):-
    date([D, M, Y]),
    day_of_week(D,M,Y,X),
    X in 1..5.

weekend([D, M, Y]):-
    date([D, M, Y]),
    day_of_week(D,M,Y X),
    X = 0.

weekend([D, M, Y]):-
    date([D, M, Y]),
    day_of_week(D,M,Y,X),
    X = 6.
```

```
morning([H, M]):-
    timex([H, M]),
    H #< 12,
    H #> 6.

afternoon([H, M]):-
    timex([H, M]),
    H #> 11,
    H #< 19.

evening([H, M]):-
    timex([H, M]),
    H #> 18,
    H #< 23.
```

Listing 3.3: High-level representation of date and time

For these defined predicates to be used as part of our language bias, we need to include the following:

- `modeb(weekday(+date), [name(weekkday)]).`
- `modeb(weekend(+date), [name(weekend)]).`
- `modeb(evening(+time), [name(evening)]).`
- `modeb(morning(+time), [name(morning)]).`
- `modeb(afternoon(+time), [name(afternoon)]).`

Since TAL has the ability to handle negation during the learning process we also add the negation of these (e.g., `modeb(\+afternoon(+time), [name(afternoon)]).`

3.4.3 User activity

Our definition of user activity is simple. The activity span from the background knowledge tells us the time frame whereby the user is active.

For each activity span we have from the background knowledge, we can find out whether a user is active at a particular point in time, simply by checking that this time point is after the activity start time but before the activity end time. The same principle is followed and applied for charge events, application usage events and on events (i.e., the period that the phone is switched on). Listing 3.4, below, demonstrates our formal definition in Prolog.

During the modelling phase we have made an important observation. People don't *usually* answer calls when they are in the middle of another call. We model this with the `in_call/2` predicate which checks the time span of all our phone calls. Again, negation of these predicates is handled by TAL so we only need to declare them in the mode body declarations.

```
user_is_active(D,T):-
    after(D,T,D1,T1),
    before(D,T,D2,T2),
    activityspan(D1,T1,D2,T2).

phone_charging(D,T):-
    after(D,T,D1,T1),
    before(D,T,D2,T2),
    chargespan(D1,T1,D2,T2).

phone_on(D,T) :-
    after(D, [H,M],D1,T1),
    before(D,T,D2,T2),
    onspan(D1,T1,D2,T2).

phone_on(D,T):-
    user_is_active(D,T).

user_is_using_app(D,T,A):-
    appspan(D1,T1,D2,T2,A),
    after(D,T,D1,T1),
    before(D,T,D2,T2).

in_call(D, T):-
    after(D, T, D1, T1),
    before(D, T, D2, T2),
    callspan(D1,T1,D2,T2, ....).
```

Listing 3.4: Representing user activity

3.4.4 User location and device proximity

Intuitively, we know that location, and especially the people you are with, play a crucial role in defining mobile user behaviours. We don't have information about the user's surroundings but

we have sufficient information about the user's transitions from one place to another. Listing 3.5, below, shows our definition for location in Prolog.

We define that the user is at location X, at time [D, T] if we there exists a location span such that [D,T] is after [D1, T1] but before [D2, T2] indicating the user is at location X. For device proximity, we check if there exists a device span and perform the same check as for the location. Another important aspect is to find out whether the user is at particular location by introducing a time window of two hours. For example, we say that the user has been at location X at [3, 6, 2011], [16, 13] if and only if there exists a cell span on [3, 6, 2011] and hour 16 is less than the span's end time + 2 hours, but bigger than the span's start-time.

```
nearDevice(D,T,DevID):-
    after(D,T,D1,T1),
    before(D,T,D2,T2),
    devicespan(D1,T1,D2,T2,DevID).

not_nearDevice(D,T,DevID):-
    after(D,T,D1,T1),
    before(D,T,D2,T2),
    devicespan(D1,T1,D2,T2,DevID1),
    DevID =/= DevID1.

at(D,T,CellID):-
    after(D,T,D1,T1),
    before(D,T,D2,T2),
    cellspan(D1,T1,D2,T2,CellID).

not_at(D,T,CellID):-
    after(D,T,D1,T1),
    before(D,T,D2,T2),
    cellspan(D1,T1,D2,T2,CellID1),
    CellID =/= CellID1.

user_been_in(D1,[H1,M1],C) :-
    cellspan(D1,[H2,M2],D1,[H3,M3],C),
    H1 #< H3 + 2,
    H1 #> H2.
```

Listing 3.5: Representing user location

3.5 Training examples and format definition

We classify each phone call as a positive example when its duration (in seconds) is higher than 0 and as a negative example when its duration is 0. There are two different scenarios for classification, when:

1. Incoming phone calls are our training examples.
2. Outgoing phone calls are our training examples. In this case, however, the concept of accepting/rejecting calls is somehow distorted because outgoing calls is about making phone calls. So an outgoing phone call can loosely be considered the same as an incoming call.

TAL requires training examples to be encoded as follows: `example(instance, class)` where `class` denotes that the example is positive or negative. Positive examples are identified with a number greater than zero.

Conversely, negative examples are identified with a number less than zero. The `instance` can be any of the mode head declarations defined. In our case, since we are only concerned with *observed predicate learning* we are only dealing with one type of rule and therefore there can only be one type of example.

Our format definition relies on two data sources; the Reality Mining dataset and data we directly obtained from ULearn. Extraction of information with the Reality Mining dataset is not as direct as one would expect and this is showcased in Chapter 4. Our training examples and formatting of facts with the Reality Mining dataset vary slightly with ULearn. However, the definition and format of duration spans in the background knowledge isn't any different. The spans are encoded in both cases as follows:

```
1  /* dynamic background knowledge */
2  ...
3  activityspan([15, 10, 2004], [11, 46], [15,10,2004], [11,48]).
4  activityspan([15, 10, 2004], [12, 29], [15,10,2004], [12,31]).
5  ...
6  chargespan([22, 7, 2004], [10, 53], [22, 7, 2004], [13, 38]).
7  chargespan([26, 7, 2004], [21, 52], [26, 7, 2004], [23,51]).
8  ...
9  cellspan([21, 7, 2004], [19, 49], [21, 7, 2004], [19, 53], 5119.408110).
10 cellspan([21, 7, 2004], [19, 53], [21, 7, 2004], [19, 54], 5119.407930).
11 ...
12 devicespan([19, 7, 2004], [16, 58], [19, 7, 2004], [17, 4], 961024891).
13 devicespan([19, 7, 2004], [17, 4], [19, 7, 2004], [17, 10], 971198156).
14 ...
15 on([26, 7, 2004], [15, 18], [28, 7, 2004], [9, 28]).
16 on([28, 7, 2007], [10, 8], [28, 7, 2004], [19, 2]).
17
18 device_id(961024891).
19 device_id(971198156).
20
21 /* contact 42 */
22 contact(42).
23 /* unknown contact */
24 contact(-1).
25
26 /* area id & tower id */
27 cell(5119.408110).
28 cell(5119.407930).
```

Listing 3.6: Similarities in the encoding of background knowledge

3.5.1 Reality Mining dataset

The Reality Mining dataset doesn't come with a lot of contextual information. In this case, our head of the rule is defined with three prominent arguments, namely, date, time and contact. We define the mode head declaration as `modeh(accept(+date, +timex, +contact))`. Positive and negative examples are defined respectively as follows:

```
example(accept([25, 7, 2004], [14, 38], 42), 1).
example(accept([29, 7, 2004], [10, 59], -1), -1).
```

3.5.2 ULearn

In Section 3.1 we mentioned that one of our main aims is to devise a learning framework that is partly immune to applications of different datasets. For ULearn, the changes made to the

framework were significant enough (see Chapter 6 for details) to improve learning, but small in terms of changes in the background knowledge and language bias. This shows that a transition from one set of data to another is possible with our learning framework. Our application, on the other hand, records a number of attributes when a phone call takes place. We take this to our advantage and encode the mode head declaration as follows:

```
modeh(accept(+date, +time, +contact, +volume, +vibrator, +battery_level,
+screen_brightness,+headset,+screen_status,+light_level,+battery_charging)).
```

In contrast to the Reality Mining dataset, the head declaration has been extended considerably. We think that these arguments contribute towards the user's decision in accepting/rejecting calls and therefore are key factors in the decision-making process. Listing 3.7, below, illustrates background knowledge we are only able to define using data from ULearn because of richness and availability.

```
1  example(accept([19,3,2011],[12,11],-1,7,2,46,255,0,1,10.0,0),1).
2  example(accept([19,3,2011],[14,6],323,7,2,22,255,0,0,640.0,1),-1).
3  ...
4  /* dynamic background knowledge */
5  ...
6  appspan([18, 3, 2011],[11, 00], [18, 3, 2011], [11, 10], 'skype').
7  appspan([18, 3, 2011],[11, 03], [18, 3, 2011], [11, 04], 'calendar').
8  ...
9  callspan([19, 3, 2011],[12, 11],-1,7,2,46,255,0,1,10,0).
10 callspan([19, 3, 2011],[14, 6],323,7,2,22,255,0,0,640,1).
11 ...
12 app('skype').
13 app('calendar').
14 ...
15 light_level(0.0).
16 light_level(225.0).
17 light_level(2600.0).
18 ...
19 /* static background knowledge */
20 ...
21 volume(0).
22 volume(1).
23 ...
24 vibrator(2).
25 vibrator(3).
26 ...
27 battery_level(11).
28 battery_level(12).
29 battery_level(13).
30 ...
31 screen_brightness(76).
32 screen_brightness(77).
33 ...
34 screen_status(0).
35 screen_status(1).
36 battery_charging(0).
37 battery_charging(1).
38 headset(0).
39 headset(1).
```

Listing 3.7: Background knowledge exclusive from data through ULearn

3.6 Heuristics

Heuristics are used in search-space problems to direct the search more quickly to a solution. Heuristics enable us to discard some parts of the space that would have otherwise been explored. In any ILP system, especially in large domain problems, the use of heuristics becomes essential. Moreover, when dealing with noisy datasets the task of computing expressive rules is even harder. The use of a covering loop together with a strategy performs considerably better both in terms of accuracy and richness in the rules. TAL supports the cover loop and has a number of heuristic strategies each suitable to different problem domains.

3.6.1 Strategies

A number of predefined strategies already exist within TAL. We list them below and explain each one of them briefly:

- **Breadth first search:** This strategy uses the depth of the derivation tree as its score. The learning process terminates once d solutions are found, where d is the depth of the tree and defined in the learning file as an option.
- **Progol:** This strategy is similar to the one Progol [Mug95, Mug97] uses where the score of each solution is defined as the number of positive examples covered/entailed minus the number of negative examples covered/entailed minus the complexity of the solution which is given as the number of literals in the rule i.e., #positive examples covered − #negative examples covered −#complexity. Termination is achieved after a complete search.
- **Accuracy based:** This strategy is the same as the accuracy that can be derived from a traditional confusion matrix. It is defined as the sum of true positive examples (entailed/-covered examples) and number of true negative examples (non-entailed/non-covered negative examples) over the total number of examples; i.e., $acc = (TP + TN)/(P + N)$). The accuracy based strategy does not terminate until a complete search has been carried out. It is therefore advisable, to use a timeout to denote the amount of time we want TAL to run for Progol and Accuracy based strategies.
- **Custom strategies:** This is a placeholder for strategies that can be implemented and used by the user and should be included in the file `user/custom_strategies.pl`

The breadth first search strategy is ideal for problems that have a small to medium domain. The hypothesis space will be small enough to be searched completely and therefore a solution will be obtained whether we either reach the depth or no more solutions can be found.

Both the Progol and accuracy based strategies are ideal for large domain problems. Both give good results given enough time. Nevertheless, Progol considers the complexity of a solution. Over fewer examples this strategy becomes more dominant. Even if we discard complexity we can see how one case could prefer a solution by seeing a graph of the solutions.

In the first case we would have $score = TruePositive - FalseNegative$ and in the second $score = (TruePositive + TrueNegative)/Total$. In the second case the false negatives are completely irrelevant whilst in the first case the true negatives are completely irrelevant. Both strategies do not terminate until a complete search has been performed and for a large domain problem this could mean that the search could potentially take hours or even days before it is complete.

It also does not necessarily mean that the result would be any better had the search been interrupted or stopped half-way through. The results we obtain will vary between different strategies both in terms of the solution and score. The score for the accuracy based strategy is more intuitive and understandable by the user as opposed to the score for the Progol strategy.

3.6.2 Cover Loop approach

An efficient coverage computation is crucial for the performance of an ILP system [FCS+03]. The cover loop incrementally constructs a hypothesis using one uncovered example at a time, and we repeatedly search for a rule covering as many positive examples and none of the negative examples. The positive examples covered are then deleted and the process is repeated until the algorithm terminates. TAL implements this approach and is activated by setting `option(cover_loop, true)`.

It performs best on monotonic problems, however it does not necessarily mean that its use with non-monotonic problems is completely ineffective. We start with the full set of positive examples. We specify a number of iterations, N, and at each iteration we select one of these positive examples as the example seed used for learning. The standard TAL algorithm is then used and terminates if one of the following conditions is met:

- A timeout occurs.
- The termination condition of the selected strategy is met.

This approach terminates if the score of the current solution is below a threshold we specify as `option(loop_threshold, t)` where `t` is a negative number between -1 and 0. Whenever the score given by the seed example is lower than this threshold, a new iteration is started. In this case, the current solution, which didn't score considerably well is appended to the background knowledge and the examples entailed by this solution are removed.

It is notable to say that solutions which are non-monotonic at any iteration do not have an impact on the subsequent iterations. In other words, if there are negated literals in the body of a rule, then there is no effect on the learning and the process works as a monotonic problem. However, If the solution is recursive (e.g., "accept if accept ...") or we have more than one type of rule or some form or dependency in the rules, then it will have a direct impact. The final solution is a conjunction of the best hypotheses resulted after N iterations.

3.7 Cross Validation and Receiver Operating Characteristic

3.7.1 Cross Validation

Cross validation is a concept very closely related to machine learning. It is primarily used to estimate how accurate a classifier is on newly unseen instances. Once a learning algorithm is trained using some set of examples, then it is largely assumed that it will also be able to classify the new examples correctly. However, there might be cases where learning was performed for too long which will result in overfitting or the examples were not representative of the learning problem. In such situations, the learner will classify the example(s) wrongly.

To avoid overfitting, we can use the concept of cross validation which provides us with the means to make our learning system less prone to error. In cross validation, we partition the examples into subsets (folds) such that the learning is initially performed on a single subset, while the rest are retained for subsequent use to confirm and validate the initial learning. The initial subset is often termed as the *training* set and the rest of the subsets as *validation* or *testing* sets.

In our case, our training examples are split as above. The training examples are used to compute a solution with TAL. The validation examples are then used to calculate the accuracy of the solution/classifier. The accuracy is simply calculated by the number of examples correctly classified divided by the total number of examples.

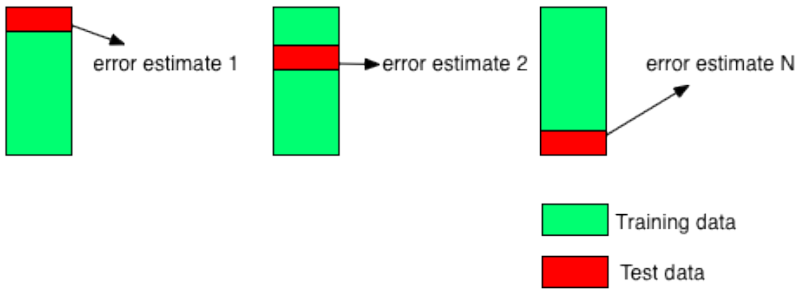


Figure 3.2: N-fold cross validation

Figure 3.2, above, depicts the cross validation process. The data is split each time into training and testing datasets. At each run, the error is calculated and is accumulated to divide by the total number of times run as the mean average error. The total error estimate is then given by the following formula:

$$\widehat{e} = \frac{1}{N} \sum_{i=1}^{N} e_i$$

In TAL, we perform cross validation as follows:

- We split our training examples (both positive and negative) into N distinct folds, randomly. We first determine how many examples each fold should consist of by dividing our number of examples by the number of folds. We then randomly select examples for each fold ensuring that those which have already been sampled are not resampled. Listing 3.8, below, demonstrates how this is achieved.
- For each fold, the TAL algorithm is run (with our specified parameters) and common performance metrics, derived from the confusion matrix, are calculated.

```
1  get_folds(Examples,N,Folds) :-
2      length(Examples, L),
3      D is integer(L/N),
4      get_foldsx(Examples,D,Folds).
5
6  get_foldsx(Examples, Dim, [Sampled | Folds]) :-
7      length(Examples, L),
8      D2 is D*2,
9      (L >= D2 ->
10         get_examples(Examples,D,Sampled,Remaining),
```

```
11          get_foldsx(Remaining,D,Folds)
12      ;
13          Sampled = Examples,
14          Folds = []
15      ).
16
17  sample_examples(Examples, NE, Sampled, Remaining) :-
18      rnd_select(Examples, NE, Sampled, Remaining).
```

Listing 3.8: Splitting the training examples into folds

To enable cross validation we use the option `option(xvalidation_folds, N)` where N is the number of validation folds.

3.7.2 Receiver Operating Characteristic analysis

A receiver operating characteristic (ROC) graph is a technique for visualising and selecting classifiers based on their performance [Faw06].

In our setting, we perform ROC analysis on each cross validation fold. Further, we are only concerned with two classes; our positive and negative examples. Thus, given a hypothesis and an instance, there are four possible outcomes:

1. If the instance is positive and classified as positive, it is counted as a *true positive* (TP).
2. If the instance is positive and classified as negative, it is counted as a *false negative* (FN).
3. If the instance is negative and classified as negative, it is counted as a *true negative* (TN).
4. if the instance is negative and classified as positive, it is counted as *false positive* (FP).

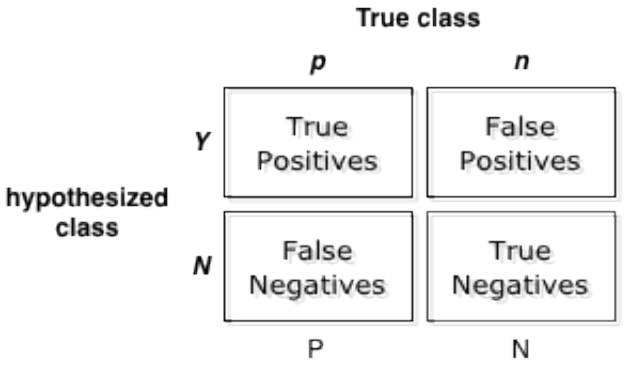


Figure 3.3: The confusion matrix for two class problems

Taking into account the above, Figure 3.3, above, shows the confusion matrix from which we can compute common performance metrics. To measure the *accuracy* (*acc*) of a rule with respect to the training examples, we simply have to sum the number of true (entailed) positive examples and the true (not entailed) negative examples and divide this sum by the total training examples.

This metric gives us how accurate our hypothesis is with respect to the examples. The *error* is simply calculated as $1 - acc$. Other popular metrics which we calculate, include, the true/false positive rate also known as recall and the positive predictive value also known as the precision rate. Listing 3.9, below, illustrates how the ROC metrics are computed for each validation fold. As part of our evaluation (see Chapter 6), we only make use of the accuracy, error and positive predictive metrics as they are the ones with a more intuitive meaning to the reader.

```
1  measures(OutTest, Performance) :-
2      ... omitted ....
3      length(ListPosExamples,Pos),
4      length(ListNegExamples,Neg),
5      ... omitted ....
6      length(ListProvedPosExamples, PPos),
7      length(ListProvedNegExamples, PNeg),
8
9       /* ROC analysis */
10
11      TruePos is PPos,
12      TrueNeg is Neg - PNeg,
13      FalsePos is PNeg,
14      FalseNeg is Pos - PPos,
15
16      /* hit-rate, recall */
17      (Pos = 0 -> Sensitivity = NaN ; Sensitivity is TruePos/Pos),
18
19      /* false-positive rate */
20      FallOut is FalsePositive/Neg,
21      ((Total is (Pos + Neg), Total = 0) -> Accuracy = NaN ;
22      Accuracy is (TruePos + TrueNeg) / (Total)),
23
24      /* true-negative rate */
25      Specificity is 1 - FallOut,
26
27      /* positive predictive value */
28      ((Temp2 is (TruePos + FalsePos), Temp2 = 0) ->
29      Precision = NaN;
30      Precision is TruePos / (TruePos + FalsePos)),
31
32      /* save in Performance list */
33      Performance = [
34         roc(totpos, Pos),
35         roc(totneg, Neg),
36         roc(tp, TruePositive),
37         roc(tn, TrueNegative),
38         roc(fp, FalsePositive),
39         roc(fn, FalseNegative),
40         roc(tpr, Sensitivity),
41         roc(fpr, FallOut),
42         roc(acc, Accuracy),
43         roc(spc, Specificity),
44         roc(ppv, Precision)].
```

Listing 3.9: Calculating the confusion matrix metrics

3.8 Rule refinement and revision: A user's perspective

Rule refinement adds flexibility to the user. Let's say a user is unhappy about the rules the system has proposed. Then with the use of integrity constraints rules can be transformed to suit the user's needs. On the other hand, if there are certain rules that the user has favoured, we can use this to our advantage to perform theory revision. The two concepts are described below together with how our framework utilises them.

3.8.1 Integrity Constraints

When the problem domain is large it is necessary to have sufficient language bias to guide the search. With the use of integrity constraints the search space can be reduced considerably relieving some of the computational efforts. There are times where certain literals in the body of a rule do not make much sense together. The notion of "sense" is sometimes dependent on the current user. In Chapter 5 we describe how we take the user's feedback into account to apply integrity constraints. We ask them to choose pairs of conditions that are personally undesirable to them. We then encode them, as required by TAL (see Listing 3.10).

TAL requires the `option(ic_check, true)` to take integrity constraints into account. Consider the following scenario. The battery level of the user's mobile is at 10% and there exists a rule that rejects phone calls from a particular contact when the battery level is at 10%. Assuming that this contact is a very important person, we would like to override the contents of this rule. The listing demonstrates how this is achieved so that the two conditions can appear separately but not together.

```
ic :-
  pr(_, R),
  member((battery_level, _, _),R),
  member((contact, _, _),R).
```

```
ic :-
  pr(_,R),
  member((battery_level,_,_),R),
  member((screen_status,_,_),R).
```

Listing 3.10: Structure of integrity constrains. The first argument in member denotes a name we assign to each mode declaration.

Evaluation of integrity constraints can be problematic. This refers to the problem of floundering, which occurs when an argument is not *ground*. In other words, a variable is required to have a value in order for the transformation to proceed, but there is nothing that makes use of this value. It is usually, but not always, associated with negation. Consider the following rule in Prolog with facts `man(a), man(b) and married(a)`:

$$bachelor(X) \text{:- } not(married(X)), man(X).$$

If we construct its derivation tree we will see that the query will give no answers. To fix this we need to make X ground by changing the order in the body. Unfortunately, the current version of TAL does not check for floundering when evaluating integrity constraints meaning that the computed rules may not be "evaluable".

3.8.2 Rule alterations

From a user perspective, rules should define part of their personal behaviours. Therefore the ability of taking control of the rules is essential. Manually, we can accomplish this by changing the arguments in the head. On a mobile phone phone such as our own, the user has the freedom to choose which arguments to include in the head of the rule. Essentially, this becomes a different ILP learning task, but to the user it is a different rule that excludes some meaningless arguments. Construction of new rules is also possible and is fully described in Chapter 5 where it is implemented.

3.8.3 User theory revision

To allow for revision of rules, learning must be incremental. As examples and background knowledge of user behaviour are continuously added, periodic revision is necessary of the rules already learned to account for the new instances that are not necessarily covered by the previously learned rules. Theory revision is concerned with the task of computing a new theory T' from theory T that correctly accounts for newly acquired examples. We make use of the algorithm presented in [CRR+08]. The algorithm involves three phases. The pre-processing phase where the user's top theory is transformed suitable for learning exceptions, the learning phase where learning is performed as before, and the post-processing phase that generates a revised user theory that is consistent with our current examples. Listing 3.11, below, illustrates the syntactical transformation of a rule during the pre-processing phase.

Pre-processing phase

We initially start with a set of rules that the user considers *revisable*. Each literal in the body of each rule is rewritten by the atom *try(i, j,* $b_{i,j}$*)* where i is the index of the rule we are considering for revision, j is the index of the body literal in the current rule and b is the body literal as from before. Additionally, the literal $\neg exception(i, h_i)$ is added to the body of the rule where i is as before and h_i is the head of the rule.

For each transformed body of each rule, the following three rules are added:

1. $try(i, j, b_{i,j}) \leftarrow use(i, j), b_{i,j}$
2. $try(i, j, b_{i,j}) \leftarrow \neg use(i, j)$
3. $use(i, j) \leftarrow \neg del(i, j)$

```
accept(D, _,C) :-
    weekend(D), C=123.

/* after pre-processing phase */
accept(D,_,C) :-
    try(1,1,weekend(D)),
    try(1,2,C=123),
    \+exception(1, accept(D,_,C)).

try(1, 1, weekend(D)) :-
    use(1,1),
    weekend(D).

try(1,2, C=123) :-
    use(1,2),
    C=123.
```

Listing 3.11: Pre-processing transformation.

Head mode declarations are added for *exception* and *del* to allow for the computation of exception cases for already existing rules and identify where body literals need to be deleted.

Learning and post-processing phase

The learning algorithm largely remains unchanged and outputs a set of hypotheses, H. There are two notable differences. First, it determines which of the revisable rules need to be revised as well as which literals need to be deleted for and second, the body of each exception rule is computed with body declarations literals that are derivable from T.

The post-processing phase generates a revised user theory consistent with the current training examples. For each $del(i, j)$ rule, we delete the corresponding body literal j from rule i in the user theory. For each exception rule in H, the rule in the user theory is replaced with n new rules, where n represents the number of body literals in the rule. Each new rule, will be of the form $h_i \leftarrow c, \neg c_n$ where c represents all the body literals present in the original rule, and c_n represents one of the body literals in the exception rule.

3.9 Discussion

An interesting point of investigation, as part of this project, is the investigation of two machine learning approaches to gain a deeper understanding of the data to enrich our background knowledge. With the use of WEKA we first encode the data in the *.arff* format as required and run the decision tree and association rule learning algorithms.

As we have previously mentioned, conventional machine learning algorithms require the data to be formatted in a relational manner with a number of attributes as features and an attribute serving as the classification attribute.

We have used the following list of attributes:

- **Date/Time:** The date and time each phone call took place.
- **Contact:** The contact number/id of each inbound/outbound call.
- **Volume:**† The volume setting at the time of each call.
- **Battery level:**† The phone's battery level at the time of each call.
- **Screen brightness:**† The phone's screen brightness level at the time of each call.
- **Headset:**† Whether the user is using the headset at the time of each call.
- **Charge status:** Whether the phone's battery is being charged during each phone call.
- **Screen status:** Whether the user is active at the time of the phone call.
- **Nearby device:** The closest device nearby at the time of the call. If there were any.
- **Location:** The phone's last beamed location at the time of the call.
- **Classification:** Yes/No depending on whether the phone is accepted/rejected. The basis of accepting/rejecting phone calls is determined by the duration of each phone call.

†Only used with the data gathered from ULearn.

In both cases, the rules we have obtained turned out to be extremely trivial/disappointing. They did not add any value to the data and at times there were no no rules at all. We believe the main reason for this lies in the primary difference between ILP and attribute-based methods. It is widely thought that ILP is useful in learning hypotheses where attribute-based methods fail [Rov05]. In particular ILP allows for capturing relationships between objects rather than only their attributes and it can be exposed to a much wider range of problems, especially problems with large domains and large datasets.

3.10 Summary

In this chapter we have mostly focussed on theory specifics and their use to define a learning framework for mobile user behaviours used in an ILP setting. More specifically, we have specified a language bias suitable to the context of accepting/rejecting calls and shown how our training examples and background knowledge is defined and formatted. We have also presented the different heuristics strategies that can be used with TAL and the ones suitable to our problem domain. We have also presented the implementation of a full cross validation process that allows us to evaluate our results based on a full ROC analysis. Our focus then shifted towards the users' needs and how their feedback can be incorporated into refining/revising rules. In the last section, we explained how we used two statistical machine learning methods to improve our learning results and we justified why we failed.

4

Application of the learning framework to existing data

In this chapter, we present the use of our learning framework detailed in Chapter 3 using an existing, widely used, dataset. We explain the necessary steps taken to extract and encode the data as defined by our framework and TAL respectively, as well as some of the challenges we have encountered due to the rigidness of the data. The data come with an unprecedented amount of noise making the task of extraction highly non-trivial.

4.1 The Reality Mining dataset

The Reality Mining dataset[1] [EPL07] currently represents the largest mobile phone experiment ever attempted in the academic world. The Reality Mining project group at MIT collected a large amount of data on human behaviour and group interactions that have been anonymised and made available to the general public. The dataset was collected using one hundred Nokia 6600 smart phones with pre-installed software developed at the University of Helsinki.

The mobile phones were given to students and staff during the academic year of 2004-2005. During this period, the information collected includes call and message logs, Bluetooth devices in proximity of approximately five meters, cell tower IDs, application usage and phone status (i.e. active or idle). The dataset tracks the movements of a hundred subjects using mobile phones pre-installed with special software that records and sends the data to researchers.

[1] The Reality Mining Dataset:http://reality.media.mit.edu/dataset.php

4.1.1 Understanding the Data

The first step towards our goal is to examine the type and structure we have available through the data. The dataset comes as a large MATLAB[2] data file (approximately 50MB) with *.mat* as its file extension.

Once the data file is loaded, each subject can be accessed with $s(n)$ where n is indexed from 1 to 106, and each number represents a random individual. Each $s(n)$ is a MATLAB structure, each containing the following:

- ***s(n).comm***: A struct array with fields for each communication event. Fields include time/date, the direction of the call (incoming or outgoing), the duration of the call in seconds (0 indicates that neither party picked-up).
- ***s(n).charge***: The date and time the phone is charging or unplugged, denoted by 1 and 0 respectively.
- ***s(n).locs***: Time-stamped tower transitions (0 indicates no signal).
- ***s(n).all_locs***: The unique set of towers seen by the subject.
- ***s(n).cellnames***: An array of towers, each mapped by a unique string the user named the location
- ***s(n).apps***: The time each application is started, and the total number of times each application is used. Applications on the device include the browser, the video and music player, the camera, the calendar et.c. As an example if we wish to see how many times the camera was used for some random subject A, then we would access this information with *s(A).apps.camera_date*. This would return an array with size $[timesUsed \times 1\ double]$ with $timesUsed$ representing the number of times the camera was used and 1 representing the time/date the camera was used every time it was used.
- ***s(n).places***: The distribution of times the subject is at home, at work, elsewhere and with no signal.
- ***s(n).device_{names, macs, date}***: The names, mac addresses of every device and time/date discovered on each Bluetooth scan.

We can already see that from the data available to us, there are some large gaps between the raw data and the form we want them to be in. Most of the events are single time-stamped, meaning that we don't know how long they last. Furthermore, a large number of subjects did not participate for a significant amount of time during the study deviating from a normal distribution and introducing inconsistencies. What's more, the data certainly do not come without noise. The custom logging software used on the phone would crash occasionally. This has lead to missing data not only because of lack of participation but also from data corruption and powered-off devices.

4.2 The need for pre-processing

For the purposes of this project, we have placed emphasis on the rule of accepting/rejecting calls. That is to say, attempt to compute a rule which determines, for a particular user, the

[2]MATLAB: The language of technical computing

conditions under which, they will accept or reject a phone call. Here, the word reject has a broader meaning; that of *not picking up* the phone since we have no information when a user actually rejects a phone call.

There are two types of phone calls based on the direction of the call. We can either pick incoming calls, outgoing calls, or a mixture of both, although the combination of the two wouldn't be very meaningful in terms of the rule computed. For example, an ILP system would find it very hard to generalise if there are hardly any negative examples to distinguish differing cases. Negative examples are identified as phone calls with 0 duration and vice versa for positive examples.

Filtering involves selecting the appropriate type of calls. Since unknown parts of the data are incomplete/inconsistent, we first identify which users have a satisfactory number for examples and that the proportion of negative examples with respect to the positive examples is maximised. For each of these users, we randomly pick one and filter through their communication events. Figure 4.1, below, shows how communication events are processed to suit our learning needs.

Having these phone calls as the basis of our learning task, we need to uncover other events that not only can take place at the same time but are somewhat correlated to use as background knowledge.

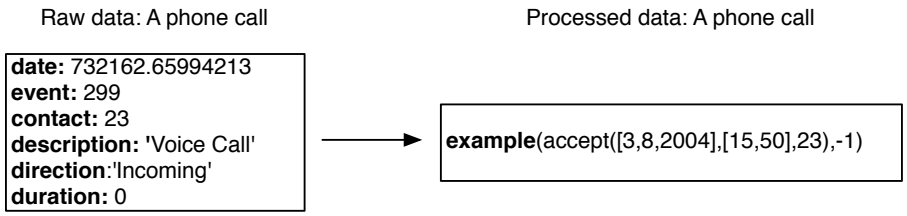


Figure 4.1: An example of raw data to processed data

As we wanted to strictly abide by our framework definition the data required a round of pre-processing. For location, we know that it represents the user roaming/transitioning from one cellular tower to the other. We use this piece of information to our advantage and iterate through all locations pair by pair and use the start-time of the second element in the pair as the end-time of the first element in the pair. All locations are also defined by the user and they can be inferred in terms of home, work or other. Because we pick a user at random, inferring the high-level location has proved problematic since not all of them are defined.

Similarly for devices, the process is iterative. At each scan, there exist a number of them. Since each device is uniquely identified, we iterate through pairs of scans. Those in the first scan which are no longer present in the second scan, are assumed to have gone out of the user's range and therefore their end-time is recorded with the minimum of all timestamps in the second scan.

We apply the same principle for the user being active. We make the following observation: If at a particular point in time, the user's phone is active, then the following event shows it as inactive. We observe that every activity event is followed by an inactivity event. This can only mean one thing; each pair of events represents the time the phone goes active and the time it becomes inactive. We also do the same for charging events and application usage events. We are now in a position to use the processed data in our learning framework.

Listing 4.1, below, shows part of the code that transforms the raw data into the format required by our framework.

```
/* users is a list of users with a high proportion of negative examples */
for j=1:size(users, 2)
  user = users(j);
  commEvents = s(user).comm;  /* retrieve phone calls*/
  columns = size(commEvents, 2);
  contacts = [], devices=[], cells=[];   /* used to store for facts */
  fid = fopen('output.txt', 'w'); /* write to a file */
  for i=1:columns /*  iterate through training data  */
    commEvent = commEvents(i);
      if strcmp(commEvent.direction,'Incoming')
         &&strcmp(commEvent.description,'Voice call')
          [contact, acceptReject] = outputCallsToFile(fid, commEvent);
          contacts = [contacts contact];
      end
  end
  /* output the rest of the events */
  cells = outputLocation(fid, s, user);
  devices = outputDevices(fid, s, user);
  outputChargingEvents(fid, s, user);
  outputActiveEvents(fid, s, user);
  outputOnEvent(fid, s, random);
  outputAppTimeStamp(fid, s, user);
  outputTypes(fid, contacts, devices, cells);
  fclose(fid);
  /* use our learning framework and copy it to a new file */
  copyfile('calls_template_matlab.pl','calls_matlab.pl')
  /* write the contents of output.txt to calls_matlab.pl, ready for TAL */
  fid = fopen('output.txt', 'r');
  A = fread(fid);
  fid2 = fopen('calls_matlab.pl', 'a');
  fwrite(fid2, A);
  fclose(fid);
  fclose(fid2);
end
```

Listing 4.1: From raw data to a learning file

4.3 Process automation and analysis

One of the most important aspects of learning mobile user behaviour is the automation of this process so that systems can autonomously operate without anyone's intervention. Using this dataset, our framework and the data mining procedures described above, we can choose an arbitrary number of users from the dataset, process the data and output corresponding learning files, use the file as input with TAL and output rules for each user. This process is also fully automated when the user uses our own developed application.

The result of this process is exhibited in Listing 4.2, below. The listing shows only results for one user as this is just an extract of results from a number of users. A full discussion and analysis of our results is given in Chapter 6.

```
Number of positive examples: 727, Number of negative examples: 176
Proportion of negative w.r.t to positive examples: 0.242091
_____New best solution_____
accept(_, B, C) :-
        C= -1,
        time_before_h(B, 9).
S:78
----------------------------

______New best solution_____
accept(_, _, C) :-
        \+C=25.
S:72
----------------------------

______New best solution_____
accept(_, _, C) :-
        \+C=2.
S:88
----------------------------
```

Listing 4.2: Rules for one user

The main idea behind automation in this project is to allow us to perform a number of experiments to find answers to three questions:

- How rich are the rules computed? Can we do better?
- How can the accuracy of the rule improve by tweaking parameters?
- Does changing the language bias affect already learned rules?

Figure 4.2, below, attempts to explain diagrammatically, the steps that need to be undertaken to provide the results outlined above.

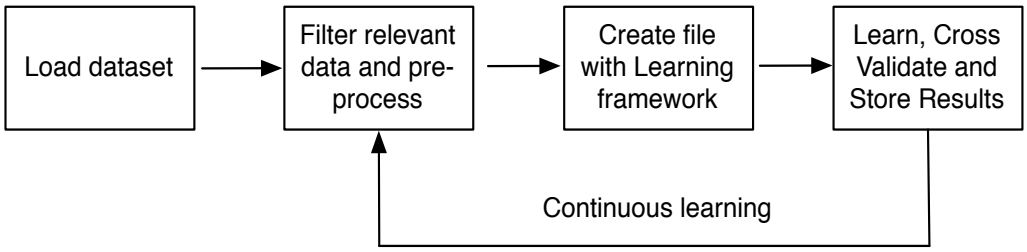


Figure 4.2: The steps involved for automating the task of learning

Most of our experiments have focussed on changing the background knowledge by widening the gaps of each span in an attempt to find a better solution. To do this we first define two rules that, given a certain time as an argument they are able to add/subtract a certain number of minutes.

These two rules are added in the background knowledge as follows:

```
1 at(D, T, CellID):-
2     location(D1, T1, D2, T2, CellID).
3     subtractTime(T1, 60, T3),
4     addTime(T2, 60, T4),
5     after(D, T, D1, T3),
6     before(D, T, D2, T4).
```

Listing 4.3: Increasing the span of location by 60 minutes

4.4 Summary

The main challenge of this case study has lied in the difficulty of trying to make sense of arbitrary and noisy data to achieve the desired result. In this chapter we have made an attempt to overcome this challenge. We have seen how the Reality Mining dataset finds an application in our learning framework, have introduced the data available to us and described how they are extracted to our suited representation. Finally, we have demonstrated how automation is possible given a dataset and a learning framework.

5

Application of the learning framework to data collected on Android

In this chapter, we give a thorough description regarding the implementation of ULearn on the Android OS[1] platform. The infrastructure we have used for the scope of this project is fully provided by the Android platform. We list and provide an explanation of the different components that make up ULearn and how they are integrated together with the back-end server to form a solid platform for presenting rules to the user. The front-end is a stand-alone application capable of recording data. Since the application is completely decoupled from the back-end server it can easily be integrated into another back-end service to perform other related tasks such as predicting user movement using GPS locations. Finally, we explain the different steps undertaken on the server that make use of the learning framework we discussed in Chapter 3.

5.1 ULearn client

The ULearn client application is implemented as an Android application. It mainly consists of five core components:

1. The *User Interface*, which allows users to initiate data collection and specify the number of hours (time window) the current session will last for. Users can view the learned rules by swiping to the right of the main screen where they can choose to enforce any of the rules and ask for theory revision or rule refinement. Users also have the ability to put their own conditions in the body of a rule. Using the Android Notification Manager the user will be notified when the time window has finished and will be asked to send the recorded data to the ULearn server.

[1] Android OS: http://www.android.com/

2. The *Data Collector Services* which are responsible for monitoring phone calls, recording the location of the user, scanning for nearby devices via Bluetooth and recording other events (more in Section 5.5) which are stored in a relational database.
3. The *Result Polling Service* which is responsible for fetching the learned rules from the server after the user sends the data for processing. The results are then translated into English using the *Rule Translator* and then presented to the user.
4. The *Phone Call Interceptor* which communicates with the Android Telephony API and automatically accepts or reject calls based on the condition(s) of the rule which the user has enforced. The enforced rule is parsed with the *Rule Parser* to allow the *Phone Call Interceptor* and the *Telephony* API to validate and verify the necessary circumstances under which the mobile device should automatically accept or reject calls.
5. The *Rule Creator*, which lists the language bias used in our learning framework, enables the user to create their own customised rule. The rule is translated both into Prolog and English and is given an accuracy of 100%. Any of the custom rules are available to view through the user interface and can equally be used for theory revision, allowing TAL to bias towards such a solution.

5.2 ULearn server

The ULearn server makes use of Apache Tomcat 7.0[2] and implements two modules:

1. The *Data Processor* which is responsible for processing the raw data when sent by the client and transforms them into learning files accordingly.
2. TAL, an abductive learning system which takes as input the processed learning files, performs the learning and outputs the solution onto the server.

5.3 Architecture and Design

Before we give an in-depth discussion of implementation of each of the different components, we present an overview of the design and system architecture.

5.3.1 System architecture

Figure 5.1 below, illustrates how the aforementioned components interact with each other to form a complete system. Both client and server have been modularised to some extent for extensibility purposes. As mentioned previously, nothing stops us from changing the back-end server with some other conventional learning system.

The single point of entry in the system is maintained by the ULearn GUI. It enables the user to initiate data collection for a specified period of time, view, enforce and/or revise rules. The GUI itself has references to a repository which maintains the user's preferences.

[2]Apache Tomcat: http://tomcat.apache.org/

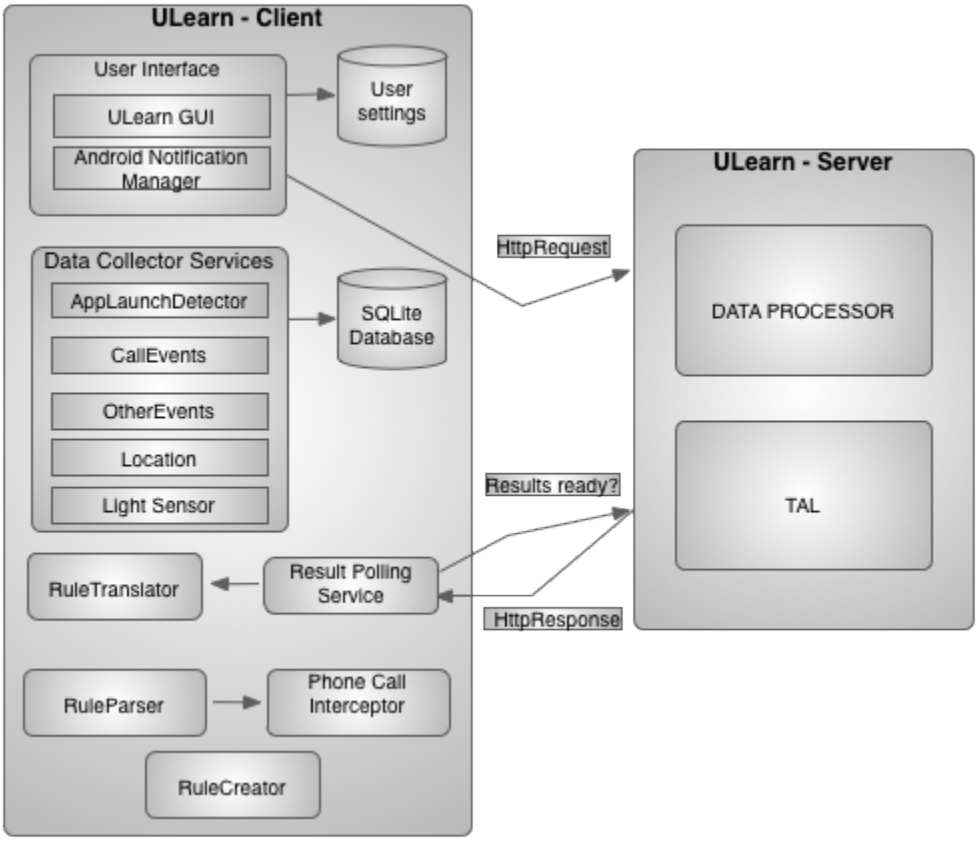


Figure 5.1: The ULearn client-server architecture

The *Data Collector Services* are services which run in the background and collect the various contextual information we are interested in. The data are stored in an SQLite[3] database on the device.

Once the specified period of time elapses, the Android Notification Manager notifies the user by asking them to send the raw data onto the server for processing and learning. An *HTTPRequest* is initiated that sends the data onto the server where the *Data Processor* takes control and prepares the learning file to be used by TAL.

In the meantime, after the user sends the data, the *Result Polling Service* polls the server at a specified time interval to determine whether TAL has finished processing the learning file. If so, the rules are sent back to the device and are translated by the *Rule Translator*. At this point, the user is notified for the last time that the results can be viewed.

To complete the architectural overview, the user may choose to enforce a rule on the phone. The *RuleParser* parses the corresponding rule according to the specifications of the *PhoneCallInterceptor*. For the rule enforced, the *PhoneCallInterceptor* monitors the call state of the phone and intercepts each phone call appropriately based on the conditions of each rule.

[3]SQLite: An ACID-compliant embedded relational database management system

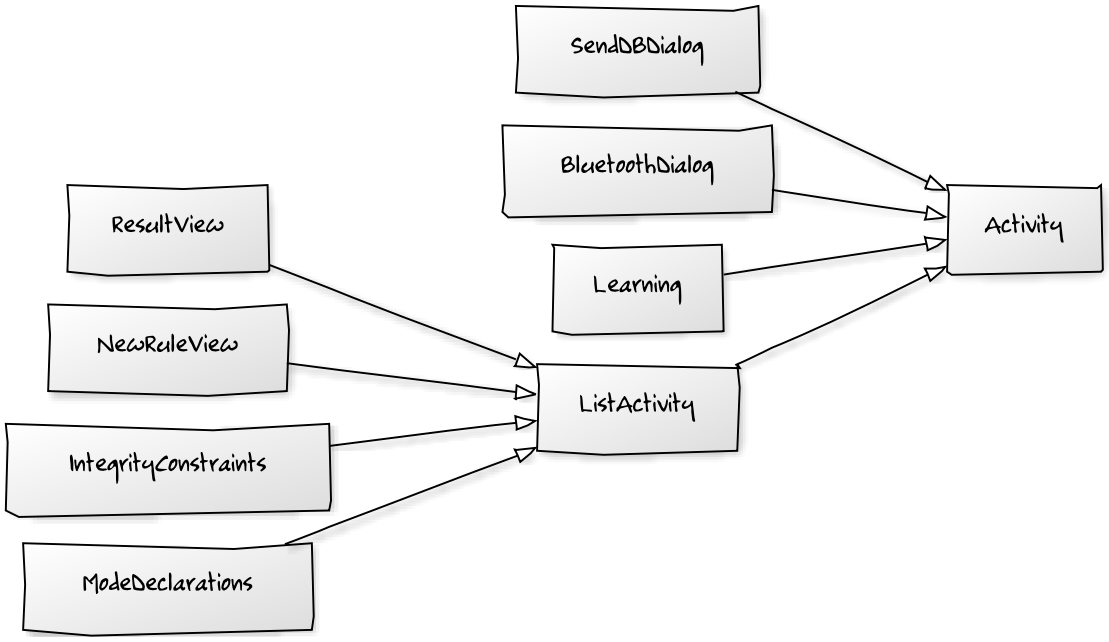


Figure 5.2: GUI Activities in ULearn Client

5.3.2 Software engineering design

We were keen to make ULearn as modular and extensible as possible. Bearing this in mind from the beginning we have chosen the appropriate modelling tools and kept them throughout the implementation phase. To model our system from a software engineering perspective we have made use of UML[4] diagrams. We start by presenting an overview of the system and its different modules. The following designs are considerations we have taken into account to build our system, along with a short description of each one.

Figure 5.2[5], above, shows the different GUI activities that there exist within our system. The single entry point of the application is the `Learning` class. From there, the user can choose to view rules. This is handled by the `ResultView` which inherits Android's `ListActivity` class which in turn inherits from Android's traditional `Activity` class. The `ResultView` is also responsible for allowing the user to enforce any of the rules.

The `IntegrityConstraints` and `ModeDeclarations` classes are responsible for allowing the user to specify integrity constraints and change the head of the rule respectively. Other `Activity` classes include various dialogues within the application that ask/notify the user for certain events. One such class is the `SendDatabaseDialog` class which is triggered when the user proceeds to send the data to the ULearn server for processing and learning. These are all handled by click and gesture events on the device.

Figure 5.3 below, shows the different classes we use to model the data of interest to us. We model each event in terms of a *duration span*. That is, each span consists of a start time and

[4]UML: Unified Modelling Language

[5]The UML diagrams were created using an online tool called yuML.me. More details can be found on their website at `http://yuml.me`

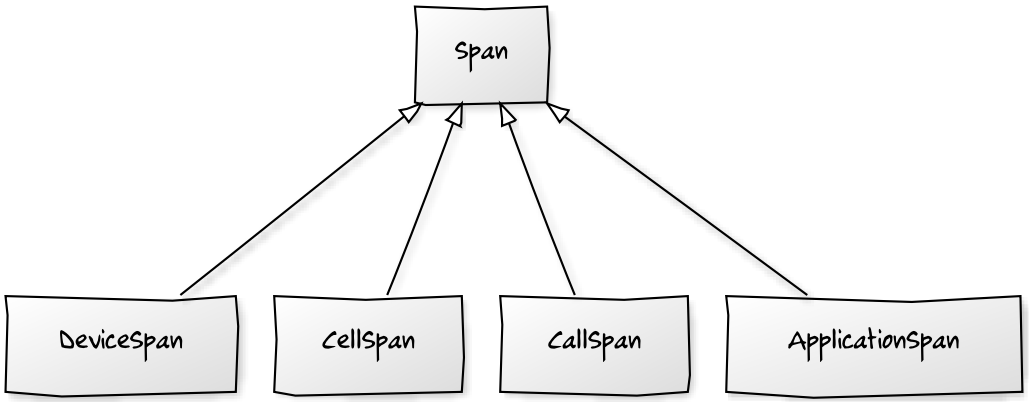


Figure 5.3: Modelling the data in Java

an end time to milliseconds precision. Some events, such as the duration of time in which the phone is active, or the amount of time the phone's battery is being charged for can be modelled by the superclass `Span`.

Other events such as device proximity and application usage, require additional information. These are modelled by the `DeviceSpan` and `ApplicationSpan` classes respectively. The former models the amount of time the user is near Bluetooth enabled devices and captures the device name and MAC[6] address of each device. The latter, on the other hand, models the amount of time the user spends using an application. In this case, additional information includes the application name as well as the type of the application being used.

The `CellSpan` class contains information about the location, in geographical and GSM[7] coordinates, of the user at a particular time. A set of `CellSpan` represents user transitions from one location to the other.

Lastly, the most important event of all is what we refer to as `CallSpan`. This models every phone call received or made by the user. For each phone call we record a number of attributes. The most important being, the direction of the call (i.e., Incoming or Outgoing), the volume setting, the duration of the call (-1 indicates a missed call), whether there is any activity at the time of call, battery charge status and level, location status and screen brightness level. The complete list of information recorded is shown in Table 5.1.

Figure 5.4, below, models the components that are necessary to perform data collection on the phone. For this reason, two `Services` are continuously running on the phone. The `BackgroundService` class has references to two additional classes; `PhoneEvents` which monitors phone call state and identity and `RecordEvents` which monitors all other events. The `AppLaunchDetector` class is responsible for monitoring the user's application activity on the phone and requires its own service (more in Section 5.5). The former has indirect references to the `DBAdapter` class, which carries out all the necessary operations to write the data in the SQLite database, whereas the latter has a direct reference to the class.

[6]MAC Address: Media access control address uniquely identifies a network device such as WiFi or Bluetooth.
[7]GSM: Global System for Mobile Communications

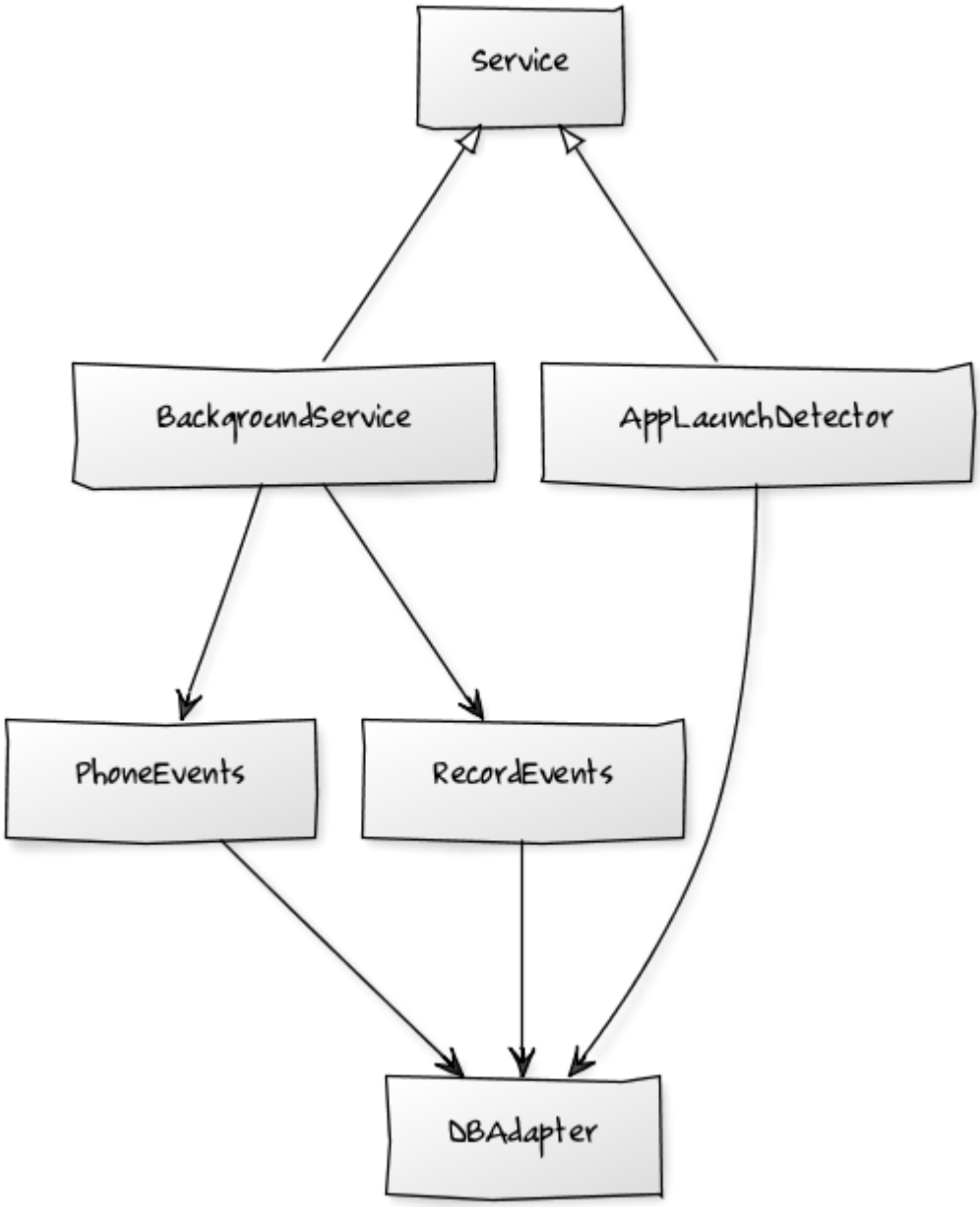


Figure 5.4: The components that allow the collection of data in ULearn

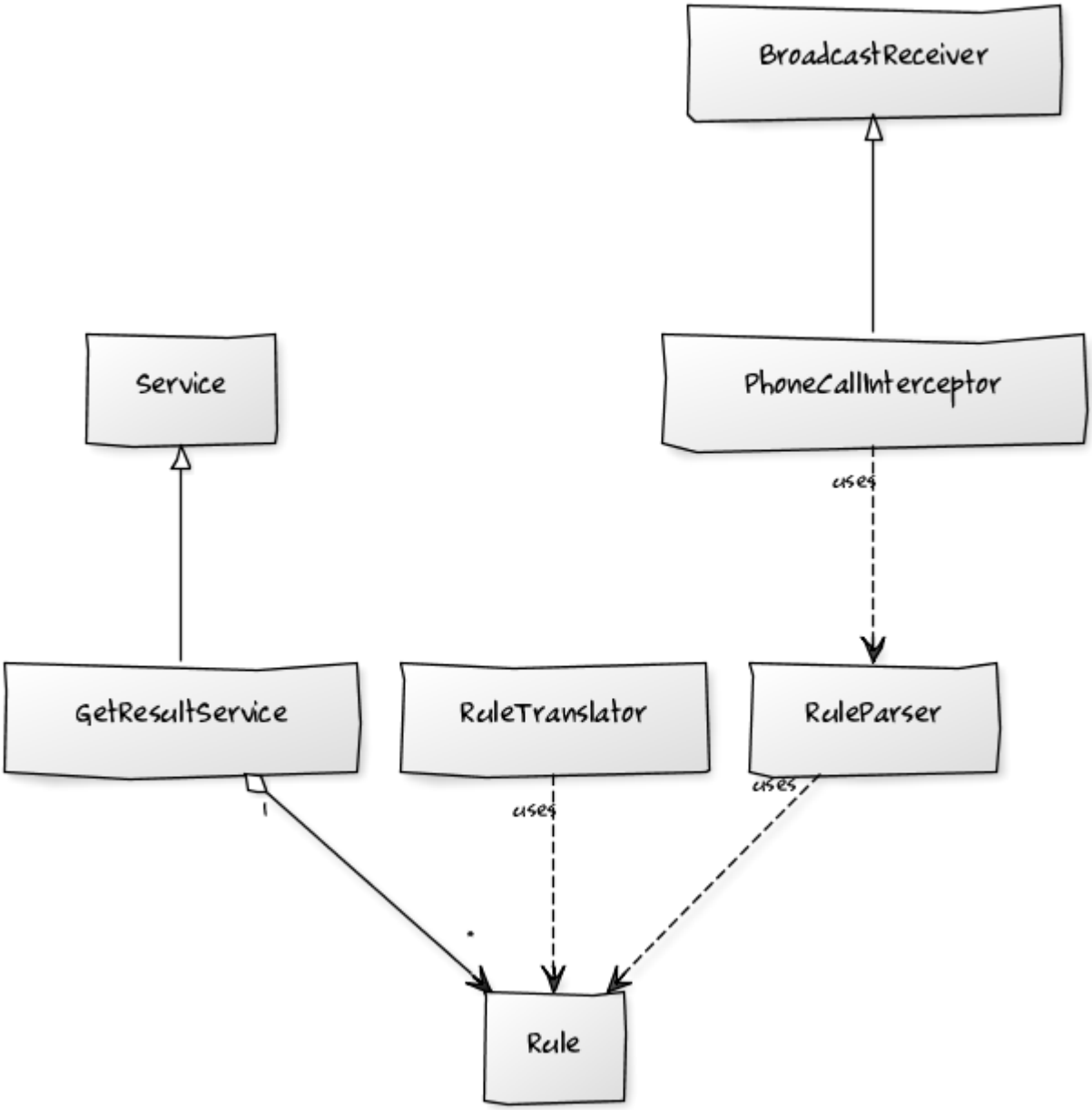


Figure 5.5: This diagram attempts to explain the interdependencies between the different components that model the process where the user is awaiting for the rules after they send the data on the server. `GetResultService`, of type `Service`, polls the ULearn server at a pre-determined interval to check whether TAL has output the rules for the specified run. If so, the service transforms them into `Rule` objects which are translated into English using the `RuleTranslator` class. If the user chooses to enforce one of the rules, the `RuleParser` class is used to parse the selected rule into conditions which are later used by the `PhoneCallInterceptor` to intercept phone calls.

5.3.3 Database design

We used Entity Relationship diagrams to model the relational database used in the ULearn Client. The ER Diagram is shown below.

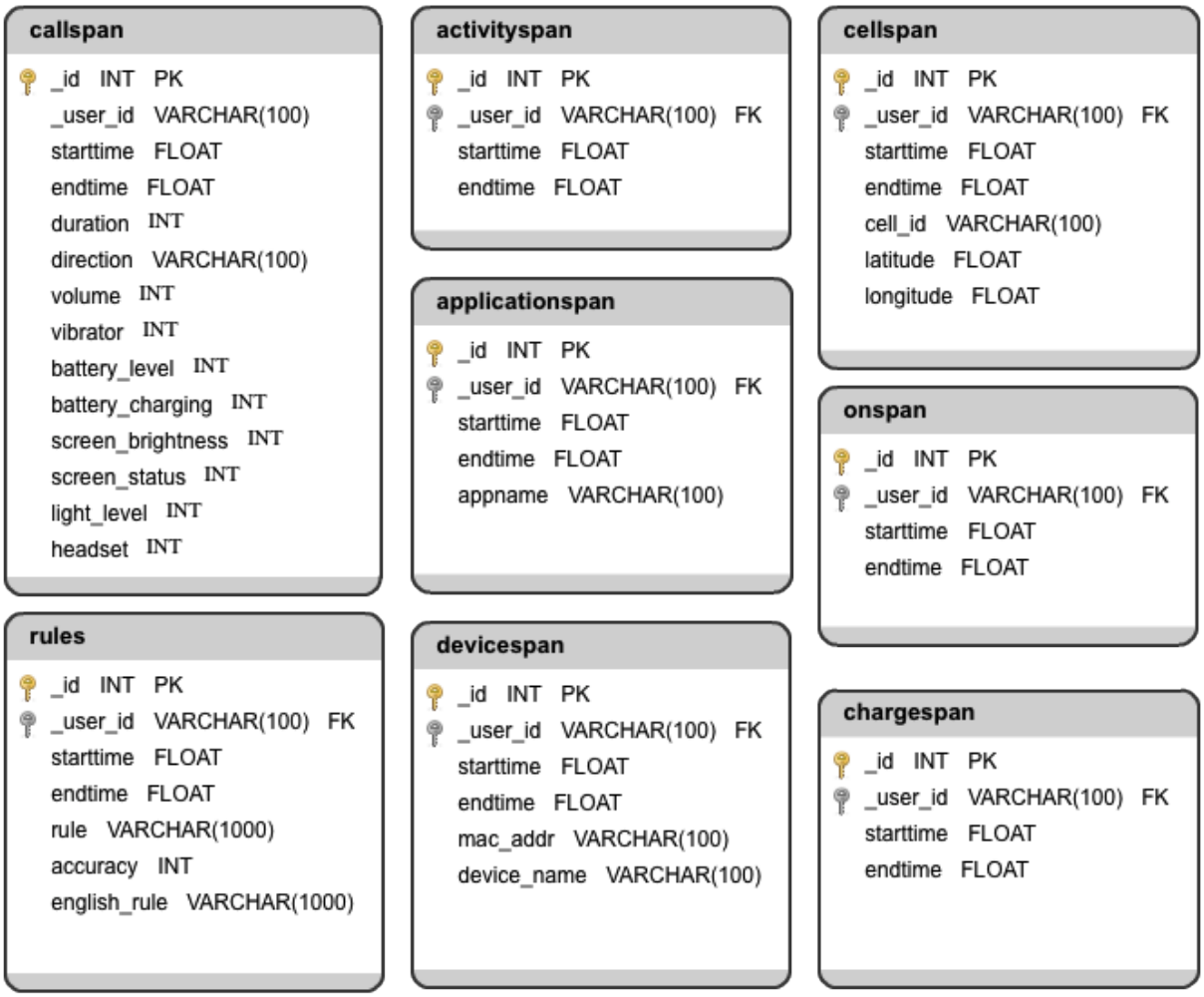


Figure 5.6: The ER diagram for the database used in our system

5.4 Graphical User Interface

To enable the user to interact with our system we have a designed a front-end user interface for ULearn. A screenshot of the interface is shown in Figure 5.7, below. This is developed in Java using Android's Widget Interface.

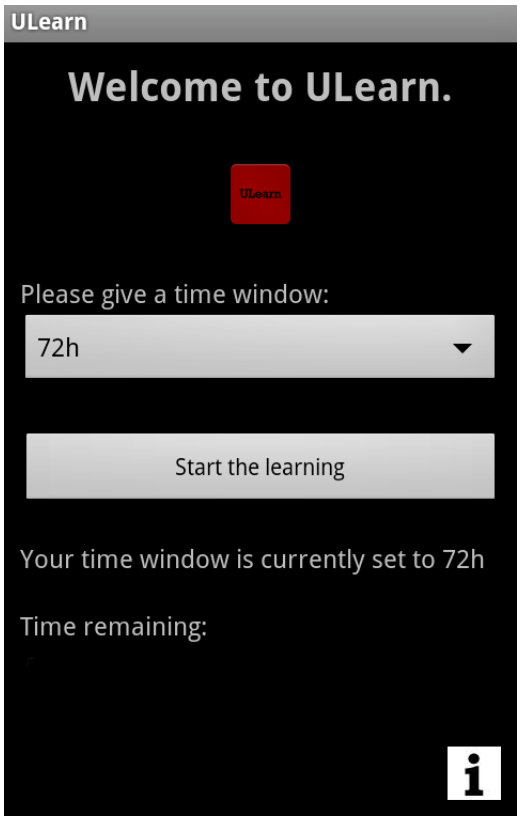


Figure 5.7: ULearn's main screen

It contains four main components that we describe in brief below.

- **ULearn view:** This is the starting point of the application. It contains the necessary controls to perform data collection, and a countdown timer which provides the remaining time of data collection.
- **Result view:** To view the results of each recording session output by TAL we store the rules in the database and can be viewed in this screen. It contains operations such as enforcing, deleting and revising rules.
- **New rule view:** This adds a new level of flexibility to the user. Users are able to create their own rules by adding any combination of conditions in the body of each rule. There are some restrictions of course. For example, a time condition, such as weekend cannot appear together with its negated form (i.e., not_weekend), otherwise they would contradict with each other.
- **Integrity Constraints view:** To enable theory revision by means of integrity constraints, in this screen, we allow the user to select pairs of *literals* to be excluded from appearing together, that would have otherwise been considered, in the *body* of the rule.
- **Mode Declarations view:** This view enables the user to change the actual *head* of the rule by excluding some of the arguments, although not all. Date & Time, as well as the Contact cannot be excluded.

These are the four main components. However, there are several other smaller components which are also considered as "Views" and these include interactive and/or information-only dialogues.

5.4.1 GUI representation

In Android, the user interface (UI) can either be designed *procedurally* or *declaratively*. This goes against the Java paradigm of GUI applications. In Java we create and manipulate user interface objects procedurally. Android gives us the option to do both. We opted for the latter, because of its clarity and minimal representation. A UI specified declaratively requires no code that is executed at run-time reducing code complexity. To represent a UI in this way, we make use of the XML mark-up language which allows entities and attributes to be specified.

```
1 <TextView android:id="@+id/ic1"
2     android:layout_width="fill_parent" android:layout_height="wrap_content"
3     android:text="IC1" />
4   <TextView android:id="@+id/ic2" android:layout_width="fill_parent"
5     android:layout_height="wrap_content"
6     android:text="IC2" />
7   <CheckBox android:id="@+id/cb" android:layout_width="wrap_content"
8    android:layout_height="wrap_content" />
```

Listing 5.1: Partial representation of the Integrity Constraints view

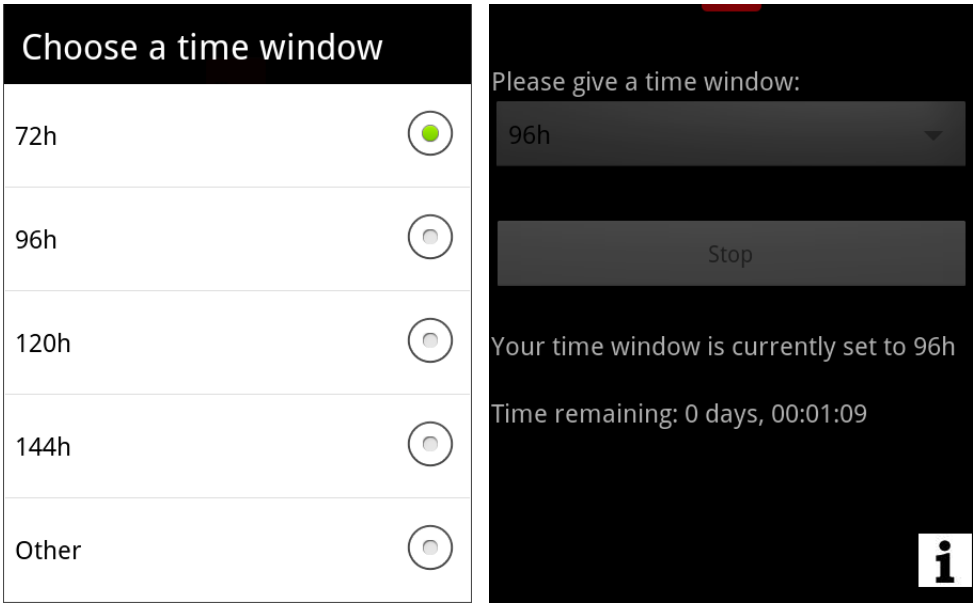
Listing 5.1, above, partially shows how we can represent an item of the Integrity Constraints view.

5.4.2 Time flexibility

We wanted to give the user the flexibility to change the time window which signifies the duration of data collection. Figure 5.12b, below, shows how users can make a custom selection. Users can choose from a minimum of 48 hours to a maximum of 999 hours, the default being 72 hours. Anything less than 48 hours is risky, especially the first time, as their usage data will not be sufficient enough to allow for the computation of meaningful rules.

5.4.3 State recovery & GUI thread synchronisation

When the user clicks on the "Start the learning" button to begin the learning process, the GUI enters a state where it is impossible to begin another session on top of the other. In addition to this, a new session cannot be started unless an old one is completely finished. A session completely finishes when the data for that session are sent to the ULearn server for processing. The application also provides a countdown timer to inform the user of the remaining time to end a session. Since we cannot control the user's actions on the mobile phone, we need to save the GUI state but also somehow keep track of the remaining time so that the timer not only resumes but is displaying the correct amount of time left. The second image in Figure 5.8b, below, illustrates the GUI state when the application is running with a remaining time countdown timer. In addition, on any unexpected event, such as when the mobile switches off, ULearn is able to recover gracefully without having to ask for the user's intervention therefore minimising intrusiveness.



(a) Selecting the appropriate time-window (b) Remaining time

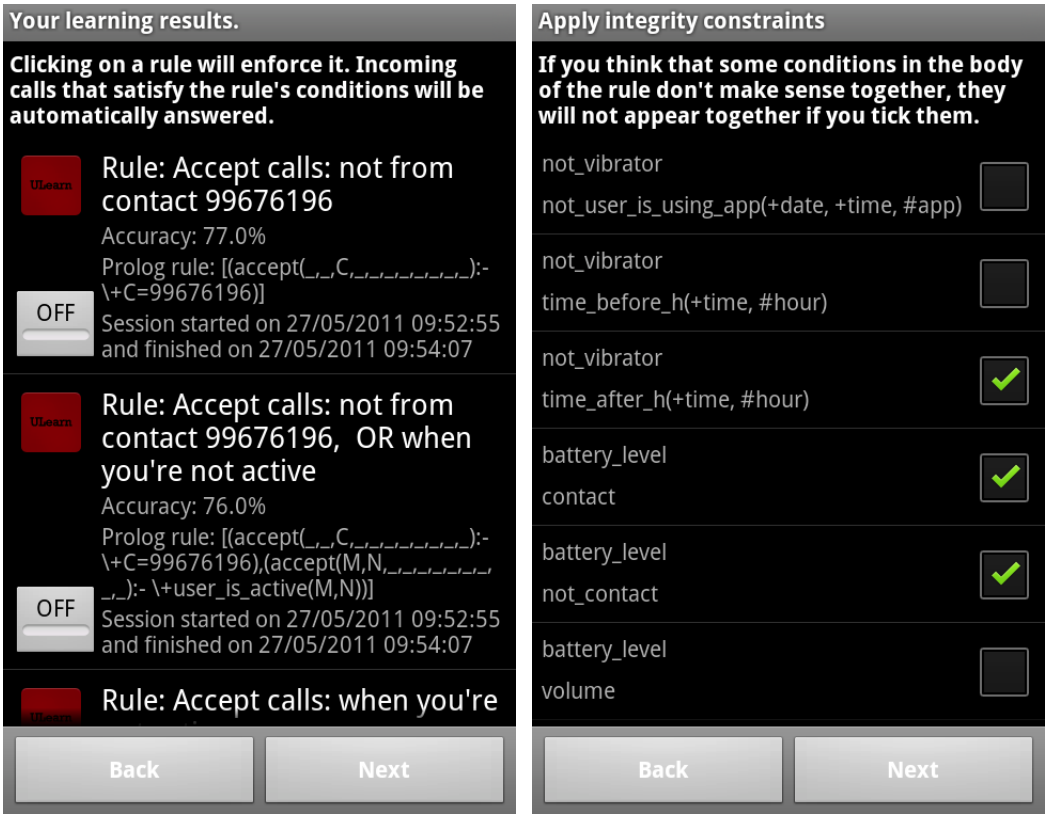
Figure 5.8: The data collection time-window

5.5 Data collector services

We wanted to minimally intervene with the user's day to day activities. Additionally, once the user initiated a new data collection session the system should still be able to capture data even when the user is using another application or the phone is in sleep/lock mode. The main responsibility of these services is to collect information from the device about hardware component settings and user activity (e.g., application usage). In this section we talk in detail about how this information is gathered and stored according to the needs of our relational database before it is processed on the ULearn server. One of the fundamental requirements of this application is to try and make the data as rich and consistent (i.e., minimal noise) as possible. This has proven challenging at times due to the restrictions imposed by the mobile phone architecture.

We have previously introduced the term *service* (refer to Section 2.5 for more details), which is Android's way of performing background tasks. Thus, our two services are implemented as dedicated threads in the context of background services. These had to be implemented as services to enable the user to freely interact with the phone (e.g., move to another application) without imposing any restrictions.

Before we set out to explain the details of the data collected, we remind ourselves that each *event* is modelled in terms of a *duration span* and has both a start-time and an end-time. From these two times, the duration for that event can be extracted. Figure 5.9, below, illustrates two additional GUI views in our application. The first shows how the results are presented to the user and the second demonstrates how integrity constraints can be specified by the user.



(a) The Result View (b) The IntegrityConstraints View

Figure 5.9: The Result and IntegrityConstraints view

5.5.1 Application usage

We thought it would be a good idea to keep track of the user's application usage. While this sounds like it should be a simple task it, unfortunately, proved to be a little more complicated than what we had originally thought. As a result, the Android SDK does not provide a way, through its API, to detect the time at which an application is launched, for obvious security reasons. To overcome this problem, we take a different approach. The Android Activity Manager logs every application launch event by its package name. For example, `com.android.camera` suggests that the user is using a the phone's camera.

Therefore, to allow our application to detect application launch events and their duration, we use Java's `Runtime` class to execute the command `logcat ActivityManager:I *:S`. Logcat, is Android's own console application that allows access to logs at run-time. We then obtain an `InputStreamReader` and continuously filter events as they arrive.

We put some thought into this to avoid duplicate entries. Like we have previously mentioned, each application consists of a finite set of activities and all reside within the same package. Consequently, if an application has displayed two activities, Android's Logcat prints the package name twice. To circumvent this, we ignore entries with the same package name within a period

of 10 milliseconds.

Furthermore, there is no way to tell that the user has left the application. What we know instead, is that the user has pressed the home key which triggers Logcat to print `com.android.launcher`. This time, to minimise data inconsistencies, we obtain the last entry in the database and update it accordingly. (i.e., the entry for which the end-time has not yet been specified).

Table 5.1, below, lists the contextual information collected via the *Data Collector* in the ULearn client.

Type	Representation	Description
Application	Application name as a String	Currently used application
Battery charging	0-1	Whether the phone is being charged
Battery level	0-100	Current battery level as a percentage
Devices	Device name, MAC address	Nearby devices
Headset	0-1	Whether a headset is plugged on the device
Light level	Number	Current light level in lux[7]
Location	Cell Tower id	Location defined by currently registered GSM tower
Location	Geographical coordinates	Collected through device's GPS
Location	Full address	Collected through Google Geocode
Ringer mode	Silent, Vibrate or Normal (0,1,2)	Current Ringer Setting
Phone status	0-1	Whether the phone is on or off
Phone in use	0,1	Whether the phone is in use
Screen brightness	0-255	Numerical representation of the screen brightness
Screen status	0-1	Whether the phone's screen is on or off
Phone calls	Date, Time, Direction, Contact etc.	Values corresponding to the time/date of the phone call for everything listed in this table

Table 5.1: Type of information collected in ULearn client

5.5.2 Battery information

We monitor two different pieces of information with regards to the phone battery. Firstly, the duration time for which the phone battery is on charge for, and secondly, the battery level. The SDK does not provide a way to query the status of these two events. Luckily for us, it provides the concept of a *broadcast receiver*, a component that responds to system-wide broadcast announcements. In other words, it provides a relatively easy way to receive information for events that we are interested in.

[9]Lux: The SI unit of illuminance and luminous emittance measuring luminous power per area

For the first task, we are interested in the `ACTION_POWER_CONNECTED` and `ACTION_POWER_DISCONNECTED` events. The former informs us that the device has been put on charge, and the latter provides the exact opposite information. The combination of the two is exactly what we need for modelling a *charge span*.

For the second task, we are interested in the `ACTION_BATTERY_CHANGED` event which is triggered every time the battery level rises or drops down. This event also comes with two additional attributes, the actual battery level and its scale (e.g., level is 23 and scale is 100).

5.5.3 Nearby devices

Modern mobile phone devices are equipped with Bluetooth technology and we can use that to our advantage to detect devices within the vicinity of the user (approximately a 10 meter radius). Our application asks the user to enable Bluetooth, as illustrated in Figure 5.10, below, at the beginning of each recording session to allow for scanning of devices.

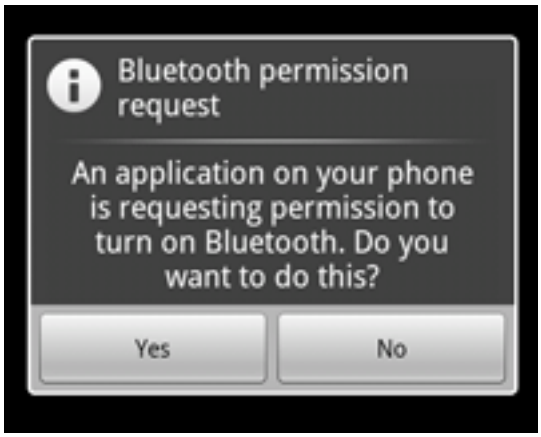


Figure 5.10: The enabling Bluetooth dialogue

We scan for devices at regular intervals. To achieve this, we perform a scan approximately every 15 minutes. As en example, if Device A and Device B are found in Scan 1, and Device B is only found in Scan 2, then Device A's end-time is updated. For the interval, we make of Java's native `Timer` class which allows us to to schedule tasks that inherit from `TimerTask`. The task makes a call to `BluetoothDiscoverer.scanforNearbyDevices()` which initiates recovery using Android's `BluetoothAdapter.startDiscovery()`.

For every device found, a broadcast event is sent and is identified by the event `ACTION_FOUND` which additionally includes the information we need for each device (i.e., name and MAC address). When the scan is complete an `ACTION_DISCOVERY_FINISHED` event is sent by the *broadcast receiver* which signifies the end of the process.

At this point, we check against the devices found in the previous scan and consider the intersection of both for the next scan. The rest of the devices are updated and this marks the *device span* of a device.

5.5.4 Location

We store location information in three different ways. We make use of GPS technology to store geographical coordinates, we also use the cell tower id provided by GSM towers and lastly, using the GPS coordinates we obtain the actual address collected using Google's Geocoding API[10].

Android provides us with the `PhoneStateListener` interface which allows us to listen for events regarding the state of the phone. We subscribe to `LISTEN_CELL_LOCATION` using Android's Telephony Manager and override the `PhoneStateListener.onCellLocationChanged()` method. This method is called every time the device registers to a different GSM tower implying that the user's location has also changed. This way, at the same time, we can also capture the longitude and latitude by implementing Android's `LocationListener` and use it as input to obtain the full address using the Geocoding API.

5.5.5 Phone status and screen information

The idea behind this is to monitor the status of the phone (i.e., is it on? or switched off?) and the current state of the screen as well as the brightness level of the screen. Using the state of the screen as information, we can make the inference that when the screen is on the user is active.

Again, through the use of a *broadcast receiver* we listen for two separate events `ACTION_SCREEN_ON` and `ACTION_SCREEN_OFF` that the system broadcasts. The former corresponds to the screen being turned on and the latter corresponds to the screen being turned off. We monitor these two events separately which allows us to determine an *activity span*. To retrieve the brightness, a system call to `Settings.System.getInt(context.getContentResolver(), Settings.System.SCREEN_BRIGHTNESS)` returns a value between 0 and 255, with 0 signifying a dark screen and 255 signifying the maximum brightness.

5.5.6 Phone state

This is arguably the most important step in the process of data collection. We observe the user's phone activity in terms of both incoming and outgoing calls. Special care is taken to ensure minimisation of inconsistencies. To allow the system to listen for call state events, we have to listen for `LISTEN_CALL_STATE` events which listen for changes to the device call state, using Android's `TelephonyManager`.

There are three operation modes for the state of the phone. Our `PhoneEvents` class extends `PhoneStateListener` and overrides the `PhoneStateListener.onCallStateChanged(int state, String incomingNumber)` method.

The `state` of the listener can be in one of three operation modes:

- `CALL_STATE_IDLE`: The phone sate is currently idle and there is no activity.
- `CALL_STATE_OFFHOOK`: The phone state is currently in off-hook mode, meaning that either an incoming or outgoing call is in place.

[10]Google Geocoding API: http://code.google.com/apis/maps/documentation/geocoding/

- `CALL_STATE_RINGING`: The phone state is set to ringing, meaning that `incomingNumber` is calling.

When the phone state is in off-hook mode, we don't know the call direction. It is vital to know the direction of each call as this portion of data would serve as our training examples. There are two possible sequences of states. The first being *ringing* → *off-hook* → *idle* and the second being *off-hook* → *idle*.

The value of `incomingNumber` is non-null only when the callback state is set to ringing. To overcome this problem we introduce two boolean flags. The first, which allows us to tell whether the phone previously rang before it is set to the off-hook state. The second, which allows us to determine whether the call is an outgoing call. In either case, we add a new entry to the database with the start-time and direction.

When the state is idle we update the last entry with the end-time as well as calculate the duration of the call. We also have to make sure that a duration of 0 is assigned to missed/unanswered calls as this how our negative examples are obtained.

Furthermore, the API would not allow us to retrieve the phone number in the case of outgoing calls. The only way to retrieve the number is through the phone log calls but the log is not updated until after the call is finished. So to get around this, although not neatly, we obtain the top-most number from the phone log a short-while after the state of phone is in idle mode and provided the call is an indeed an outgoing call.

For each of these phone calls, we also record a number of other attributes at the time of each call, some which are listed in Table 5.1.

5.5.7 Other information

In addition to the above, we record a number of other values the majority of which are query-based. These include the current light level using the phone's light sensor, the phone's ringer mode (i.e., loud, silent, vibrate) and the ringer volume of the device. The light sensor wastes a lot of battery so we are only retrieving the value at infrequent intervals to minimise batter usage.

We also wanted to infer whether the user had company (i.e., was with someone else). For example, a meeting entry in the user's calendar would suggest that they are not alone. This way we can model the behaviour of $has\_company(Date) \leftarrow calendar\_entry(Date)$. Unfortunately, we have found no way of retrieving the user's calendar entries to do this but it serves as an idea for future work in data enrichment.

5.5.8 User permissions and data anonymisation

Recording the daily behaviour of a user on a continuous basis over an extended period of time imposes remarkable privacy implications and is considered a major concern for any user. While this application has not been developed for public use, we ensure that all personal data such as phone numbers are one-way hashed using the SHA1[11] algorithm, generating unique ids used for processing and learning. Because this application is only used by the author of the thesis and his sister, data anonymisation is disabled to allow for thorough evaluation.

[11] SHA1: Secure Hash Algorithm

In the help section of the application, we provide information about the type of information that is captured and instructions on how to temporarily disable the application from logging data while it is running. Table 5.2, further below, lists the permissions, their type and a short description for each. These permissions are required for ULearn to function appropriately and have to be accepted by the user before installation.

5.6 Result Polling Service and Rule Translator

The role of the polling service is to approximately run every 15 minutes to check whether TAL has finished computation. The service is only triggered when the user sends the results to the server. When the results are fetched, the *Rule Translator* takes over and translates the results into English.

The data are sent to the server via a traditional `HttpRequest` using the `HttpFileUploader` class. Figure 5.11, below, illustrates how the user interacts with the system during this process.

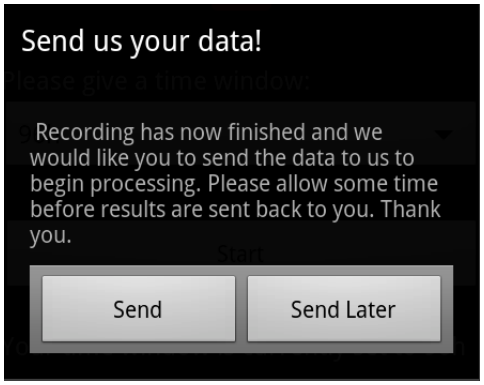


Figure 5.11: Send the data dialogue

5.6.1 Result Polling Service

We make use of Android's `AlarmManager` that allows us to set an "alarm" to check whether the results have been published on the server. On the server each user is identified by (a) the device's unique id, (b) the time the collection session started and (c) the time the collection session ended. Every 15 minutes, our `GetResults` class fires an `HttpRequest` with the parameters mentioned above. There are two cases. Either the results are sent back in a JSON[12] format, parsed and entered into the database, or the results are still not ready in which case, we reset the Android's `AlarmManager` to re-check for results 15 minutes later.

5.6.2 Rule Translator

Translation of rules into English involves a language bias mapping to English sentences. In addition, we also have to take into account the fact that the user can remove some of the arguments

[12] JSON: JavaScript Object Notation

Permission	Type	Description
ACCESS_COARSE_LOCATION	Location	Ability to retrieve Cell-ID, WiFi location
ACCESS_FINE_LOCATION	Location	Ability to retrieve GPS coordinates
ACCESS_NETWORK_STATE	Network	Allows applications to access information about the current network state
BLUETOOTH_ADMIN	Connectivity	Allows applications to discover and pair bluetooth devices
BROADCAST_STICKY	Notification	Allows an application to broadcast sticky intents
GET_TASKS	System Admin	Allows applications to get information about currently running tasks
INTERNET	Data	Ability to access the Internet
MODIFY_PHONE_STATE	Phone calls	Allows modification of the telephony state
PROCESS_OUTGOING_CALLS	Phone calls	Allows an application to monitor or abort calls
READ_PHONE_STATE	Phone calls	Ability to monitor calls
READ_CONTACTS	Address book	Access to user's address book
READ_LOGS	System Admin	Allows applications to read low-level system logs
RECEIVE_SMS	Messaging	Monitor receipts of SMS
VIBRATE	Notification	Allows access to the vibrator
WAKE_LOCK	Power	Prevent the processor from sleeping

Table 5.2: Permissions required for ULearn client

from the head of the rule and thus the translation should have to be immune to these changes. Each solution can contain more than one rule. Listing 5.2, below, shows how each solution is given by TAL.

```
1 solution(0.82, [(accept(_,B,_,_,_,_,_,_,_,_,_,_,_):-\+afternoon(B)),
2                  (accept(A,B,_,_,_,_,_,S,_,_,_,_,_):-
3                  \+user_is_active(A,B),
4                  \+S=255,
5                  not_nearDevice(A,B,'D0154A45DEFE'))]).
```

Listing 5.2: Solution output by TAL and the English translation

From the listing above we see that we first need to distinguish the number of rules contained in each solution, and for each of the rules, identify which of these arguments are "dont-cares". These are the arguments which are denoted by a single underscore and are termed as anonymous variables which can take any value.

Following that, we use Java's `Pattern` class to retrieve the body of each rule. Consider the following body in one of the rules: `C=75,user_is_active(A,B),user_is_using_app(A,B),D=78`. Splitting this string with commas would not work entirely. This is because the arguments `A,B` in `user_is_active` and `user_is_using_app` would be separated.

We need a clever regular expression that would split on a comma, that isn't followed by a closing bracket before an opening bracket. The regular expression that splits the `String` according to our needs is given in Listing 5.3 below.

As a last step, for each of the literals, we apply the mapping from our language bias as defined in Chapter 3 using our own customised dictionary. The result of the entire process with reference to Listing 5.2 is as follows:

Accept calls: not during the afternoon, OR when you are not active, when your screen is fully bright, and when you are not near device 'DevA'.

5.7 Rule Parser and Phone Call Interceptor

The *Rule Parser* is responsible for parsing the solution so that the *Phone Call Interceptor* can check against each condition and automatically accept/reject phone calls. Much of the functionality that's been developed by the *Rule Translator* is re-used and only minor changes were required to parse the solution to the standard specified by the interceptor.

The *Phone Call Interceptor* makes use of Android's `Telephony` interface. The said interface has two functions: (a) to provide call functionality and (b) to provide data services on top of the mobile network. Unfortunately, the Android SDK does not provide a public API to manipulate calls programatically but we can obtain access to this interface using reflection[13].

Using Android's *broadcast receiver* we are able to listen on the event `PHONE_STATE` without the application necessarily having to run. The event listens for both incoming and outgoing calls. The interceptor first checks whether there exists an enforced rule and then validates the current contextual conditions of the phone against the conditions defined in the rule. If the outcome of

[13]Reflection allows an executing program to examine or "introspect" upon itself and manipulate, internal properties of the program.

```
Pattern p = Pattern.compile(":-");
Matcher m = p.matcher(solution);
List<Integer> beginIndexOfEachBody = new ArrayList<Integer>();
while (m.find()) {
   beginIndexOfEachBody(m.end());
}

List<String> bodies = new ArrayList<String>();
for (int i = 0; i<numBodies; i++) {
   int start = beginIndexOfEachBody(i);
   int end = -1;
   try {
      end = solution.indexOf("),(", start);
   }catch (StringIndexOutOfBoundsException e) {
   }
   // there is only one solution so get the end of the body
   //by )]
   if (end == -1) {
      end = solution.lastIndexOf(")]");
   }
   String body = solution.substring(start, end);
   bodies.add(body);
}

for (String body : bodies) {

   String[] literals = body.split(
      "(?x),   # Verbose regex: Match a comma\n" +
      "(?!     # unless it's followed by...\n" +
      " [^(]*  # any number of characters except (\n" +
      " \\)    # and a )\n" +
      ")       # end of lookahead assertion");
}
```

Listing 5.3: Retrieval of literals in the body of the rule

```
public void onReceive(Context context, Intent intent) {
   Bundle b = intent.getExtras();
   String action = intent.getAction();
   //get the telephony interface using reflection
   ITelephony telephony = getTeleService(context);
   //has the user enforced any of the rules?
   if (interceptorIsEnabled) {
      if (action.equals(Intent.ACTION_PHONE_STATE_CHANGED)) {
         String state = b.getString(TelephonyManager.EXTRA_STATE);
         if (state.equals(TelephonyManager.EXTRA_STATE_RINGING)) {
            String number = b.getString(EXTRA_INCOMING_NUMBER);
            boolean accept = evaluateContextualConditions(number);
            if (accept) {
               telephony.answerRingingCall();
            }
            else {
               telephony.endCall();
            }
         }
      }
   }
}
```

Listing 5.4: The Phone Call Interceptor in action

these conditions is evaluated as true, we automatically accept the call, otherwise we reject it. Listing 5.4, above, shows hows the interceptor programmatically accepts or reject calls.

5.8 Rule Creator

The main task of the *Rule Creator* is to provide flexibility to the user. Using our pre-defined language bias from Chapter 3 we allow the user to choose as many literals as they like. However, since TAL will never compute a rule that is inconsistent with all of the examples, we also don't allow the user to choose the negation of an already chosen literal. In an ILP setting this preserves consistency and ensures that the rule can be later used for revision. For literals which require values, such as screen_brightness or a contact, we ask the user to input a value choosing from an appropriate range (the range is given to the user).

Due to time restrictions, however, we are unable to allow the user to add literals regarding location and/or device proximity. Firstly, the user finds it very hard to specify a location in terms of GSM coordinates. Secondly, specifying a device in terms of a network address is not an easy task for the average user. Additionally, if such a rule is enforced with the use of our *Phone Call Interceptor*, it is very hard for us to accurately check whether the current location matches the location specified by the user in the rule. Even if we cache the location, most of the times it would only be an approximation and hence the validation would fail. For devices, it becomes even harder because determining whether the corresponding device specified in the rule is nearby in near-zero time is impossible. Location and device detection typically take approximately around 5 to 15 seconds to complete - as they are asynchronous calls - making it impossible for our task; given that such a period of time does not exist when validating such conditions in real-time.

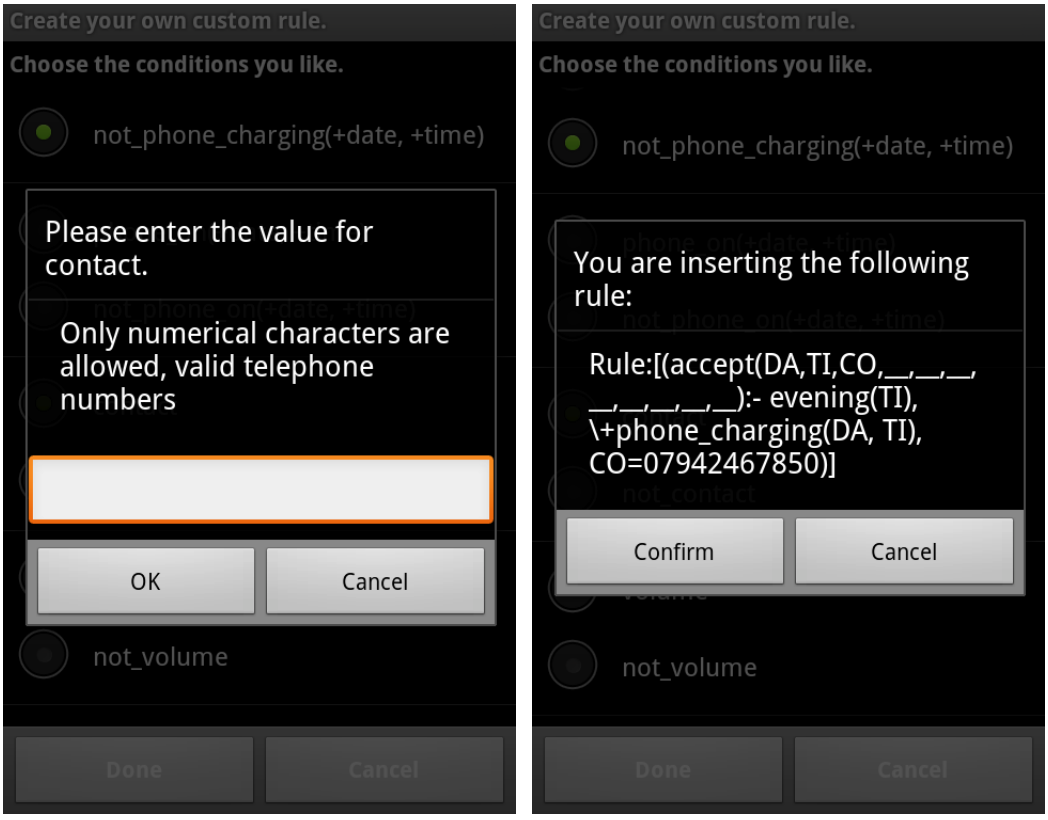
The *Rule Creator* works in two steps as follows:

- **Step 1:** We accumulate each body condition in the rule provided by the user and map it precisely according to our language bias. For conditions where a value is required, a pop-up dialog prompts the user for appropriate input.
- **Step 2:** We determine and identify the arguments to be used in the head of the rule by "merging" pairs of Strings together. Literals and their corresponding values are attached with an "=" relation and finally the rule is displayed to the user for confirmation.

Figure 5.12, below, depicts how a user would go about creating a rule. The rule may then be used as all the others, that is, for revision and enforcement. In fact, for revision it proves to be useful as TAL can potentially bias towards similar solutions.

5.9 Server

The primary reason for using a server is owed to the fact that, presently, there aren't any popular Prolog interpreters implemented/ported on the Android platform. Additionally, mobile devices are still resource constrained in terms of computational power and memory and our problem domain requires a large amount of memory to allow for the computation of rich and expressive rules.



(a) Adding values for specific literals (b) Inserting the rule into the database

Figure 5.12: Creating a new rule

Our server is implemented using Apache Tomcat. We have come to the decision of using a readily available server for powering our back-end services for two main reasons:

1. We were already aware with the technology and therefore its set up and use would be trivial.
2. The back-end server lies outside the main scope of this project and therefore its importance is questionable. Nevertheless, it is extremely scalable allowing for fast development and good performance.

Figure 5.13, below, depicts the interaction between the server and our client application. The client sends an HTTP Request to the web server. Tomcat, converts the request into an `HttpServletRequest` and dispatches it to the appropriate servlet. This object is then handled by our servlet which performs the task in question. An `HTTPServletResponse` object is generated. In turn the web server converts this object to an HTTP response and returns it to the client.

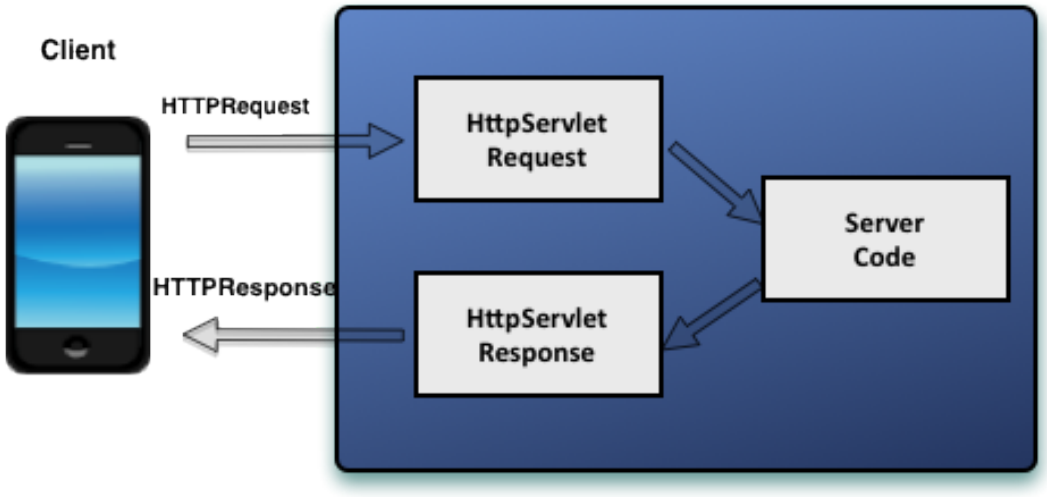


Figure 5.13: Java web Request/Response handling in Tomcat

5.9.1 Services

We have implemented two services on the server, each responsible for:

1. Uploading the data from the Android client and pipelining it to the *Data processor* for extraction.
2. Retrieving the results from TAL and sending them to the client in an appropriate format.

We have made the communication between the client to server as minimal as possible. Every request is associated with one response. The request-response protocol does not allow for any processing to take place after the response has been consumed (i.e., sent to the client). In other words, we can not inform the client that the data has been uploaded before processing of data and learning takes place.

As there is a substantial amount of data processing, we tackle the problem by creating an asynchronous task using Quartz[14]. Quartz allows us to create simple schedules to execute tasks that suit our purposes. Hence, we receive the data from the client, schedule a Quartz *job* and inform the user that the data have been sent successfully.

Our file name scheme follows a simple convention which guarantees unique identification of each user's data and learning files. Each mobile device is uniquely identified by an IMEI[15] number. Each recording session is identified by a start time and an end time which uniquely identify the period which the phone is recording data. As there could be a large number of recording sessions, each file is identified as follows: `imei_starttime_endtime`.

Using our naming convention, the `CreateLearningFile` job is responsible for retrieving the data from the server and process the data to a learning file. When processing the data into a learning file we take into account the following:

[14]Quatrz: http://www.quartz-scheduler.org/
[15]IMEI: International Mobile Equipment Identity

```
1  ProcessBuilder pb = new ProcessBuilder();
2  pb.directory(new File(talDirectory));
3
4  /* add arguments for the process and our script */
5
6  List<String> args = new ArrayList<String>();
7  args.add("/bin/sh");
8  args.add("run.sh");
9
10 /*specify file name */
11 args.add("-f");
12 args.add(fileName);
13
14 /* specify timeout */
15 args.add("-s");
16 args.add("400000");
17
18 /* specify depth */
19 args.add("-v");
20 args.add("500000");
21
22
23 pb.command(args);
24 Process process = pb.start();
25
26 /* log output to server * /
27 exhaustStreams(process);
28
29 /* block until TAL has computed a rule */
30 int termination = process.waitFor();
31
32 /* if termination was successful */
33 if (termination == 0) {
34   /* copy TAL solution using our naming convention*/
35   String command = "cp " + talDirectory+"temp/solution.txt " +
36     prefix+"results/"+userId+"_"+startTime+"_"+endTime+".txt"
37   Runtime.getRuntime().exec(command);
38 }
```

Listing 5.5: Execution of TAL on the server

1. Whether the user has specified the use of integrity constraints.
2. Whether the user has removed some of the arguments in the head of the rule, essentially creating a new learning task.
3. Whether the user has requested theory revision using a selection of previously learned rules.

With any learning file, we make use of Java's `ProcessBuilder` class which allows us to create an operating system process. Using our own developed script, we run TAL with a number of specified arguments such as the learning file, the search space depth and the time to run. Listing 5.5, above, shows part of this process.

5.10 Summary

In this chapter, we have presented an architectural overview of the system and the necessary details required for understanding, maintaining and re-engineering ULearn. We have introduced to the reader the Graphical User Interface which consists of different views and demonstrated how it interacts with the rest of the system components. A big part of the chapter was dedicated for describing the data collection process and the information we store that we use in our learning framework, discussed in Chapter 3. Another part of the chapter described the rule translating/creating process and, finally, with the aid of the Android Telephony Interface we have given an explanation of how rules are parsed so that phone calls are intercepted to suit our purpose of automatically accepting/rejecting phone calls.

6

Results & Evaluation

In this chapter we evaluate our learning framework as well as the ULearn application to determine both successes and weaknesses with a consideration of existing work. We provide the results we have obtained on both existing and newly-collected data and how our learning parameters were tweaked to achieve maximum performance in terms of rule expressiveness and richness. Additionally, we examine both quantitative and qualitative aspects of ULearn through data gathered from several experiments we carried out. Finally, an analysis of a suitability survey we produced is discussed and we explain the relevance of our results with respect to the overall vision of the work presented throughout this report.

6.1 Results

The results that follow are largely obtained on an Apple Macbook Pro Intel Core 2 Duo 2.5GHz with 4GB RAM. Our results are based on two data sources; the Reality Mining and the data we have obtained through ULearn. In this section, we present and contrast the results from both sources taking into account their similarities and differences.

6.1.1 Reality Mining

We present some of the best solutions for 4 random users. These users have been selected so that they have the highest proportion of negative examples with respect to the positive examples. This is important to allow ILP to distinguish differing cases between positive and negative examples. If the number of negative examples is limited then an ILP system may face difficulties in computing a decent hypothesis. Furthermore, each learning file is cross validated using (5 folds) and ROC analysis is performed on each fold that allows us to compute the total error estimate. For cross validation and ROC analysis the best solution, in terms of accuracy, is used. In cases where two solutions have the same accuracy, the first one in order of appearance

is used. In TAL, the lower the score achieved, the better. For the Reality Mining dataset, the Progol strategy has been used throughout. The scoring function is defined in Section 3.6.

User #5

User #5 has a total number of 544 positive examples and 89 negative examples. Average error estimate turned out to be 13.8%. Listing 6.1 illustrates the best solutions while Table 6.1 shows some of the performance metrics of the ROC analysis.

```
solution(-452,[(accept(_,_,C):-\+C=60)])
solution(-451,[(accept(_,_,C):-\+C=47)])
solution(-450,[(accept(_,B,_):-timex_before_h(B,21)),(accept(_,_,G):-\+G=60)])
solution(-450,[(accept(_,B,_):-\+evening(B)),(accept(_,_,G):-\+G=60)])

Accept calls: not from contact 60
Accept calls: not from contact 47
Accept calls: before 21:00 o'clock,  OR not from contact 60
Accept calls: except in the evening,  OR not from contact 60
```

Listing 6.1: Results for User #5

Fold	Accuracy	Error	Precision(PPV)
Fold 1	0.8640	0.1360	0.8640
Fold 2	0.9040	0.0960	0.9040
Fold 3	0.7920	0.2080	0.7920
Fold 4	0.8640	0.1360	0.8629
Fold 5	0.8837	0.1163	0.8837

Table 6.1: Performance Measure Results for User #5

User #11

User #11 has a total number of 362 positive examples and 79 negative examples. Average error estimate turned out to be 17.1%. Listing 6.2 illustrates the best solutions while Table 6.2 shows some of the performance metrics of the ROC analysis.

```
solution(-283,[(accept(_,_,C):-\+C=75)]).
solution(-281,[(accept(_,B,_):-timex_after_h(B,3)),(accept(_,_,G):- \+G=75)])
solution(-281,[(accept(_,_,C):-\+C=271)])
solution(-280,[(accept(A,_,C):-\+C=75,weekday(A)),(accept(_,_,H):-\+H=75)])
solution(-280,[(accept(_,B,C):-\+C=75,\+morning(B)),(accept(_,_,H):-\+H=75)])

Accept calls: not from contact 75
Accept calls: after 03:00 o'clock,  OR not from contact 75
Accept calls: not from contact 271
Accept calls: not from contact 75, when it's a week-day,  OR not from contact 75
Accept calls: not from contact 75, except in the morning,  OR not from contact 75
```

Listing 6.2: Results for User #11

Fold	Accuracy	Error	Precision(PPV)
Fold 1	0.7674	0.2326	0.7674
Fold 2	0.8372	0.1628	0.8372
Fold 3	0.8605	0.1395	0.8588
Fold 4	0.8372	0.1628	0.8372
Fold 5	0.8444	0.1556	0.8444

Table 6.2: Performance Measure Results for User #11

User #33

User #33 has a total number of 191 positive examples and 31 negative examples. Average error estimate turned out to be 14%. Listing 6.3 illustrates the best solutions while Table 6.3 shows some of the performance metrics of the ROC analysis.

```
solution(-154,[(accept(_,B,C):-\+C=33,\+morning(B)),(accept(_,_,H):-\+H=16)])
solution(-154,[(accept(_,B,C):-\+C=31,\+afternoon(B)),(accept(_,_,H):-\+H=16)])
solution(-154,[(accept(_,B,C):-\+C=25,timex_before_h(B,21)),(accept(_,_,H):\+H=16)])
solution(-154,[(accept(A,B,_):-user_is_active(A,B)),(accept(_,_,H):-\+H=16)])

Accept calls: not from contact 33, except in the morning, OR not from contact 16
Accept calls: not from contact 31, except in the afternoon, OR not from contact 16
Accept calls: not from contact 25, before 21:00 o'clock, OR not from contact 16
Accept calls: when you're active, OR not from contact 16
```

Listing 6.3: Results for User #33

Fold	Accuracy	Error	Precision(PPV)
Fold 1	0.8837	0.1163	0.8837
Fold 2	0.9302	0.0698	0.9302
Fold 3	0.8604	0.1396	0.8604
Fold 4	0.8372	0.1628	0.8514
Fold 5	0.7872	0.2128	0.7872

Table 6.3: Performance Measure Results for User #33

User #96

User #96 has a total number of 142 positive examples and 35 negative examples. Average error estimate turned out to be 19.2%. Listing 6.4 illustrates the best solutions while Table 6.4 shows some of the performance metrics of the ROC analysis.

```
solution(-106,[(accept(_,_,C):-\+C=200)])
solution(-104,[(accept(A,B,_):-not_nearDevice(A,B,413)),(accept(_,_,H):-\+H=200)])
solution(-104,[(accept(_,_,C):-C= -1),(accept(_,_,G):-\+G=200)])

Accept calls: not from contact 200
Accept calls: when you're not near device 413, OR not from contact 200
Accept calls: when the contact is not in your address book, OR not from contact 200
```

Listing 6.4: Results for User #96

Fold	Accuracy	Error	Precision(PPV)
Fold 1	0.8571		0.1429 0.8529
Fold 2	0.9429	0.0571	0.9429
Fold 3	0.7143	0.2857	0.7143
Fold 4	0.6857	0.3143	0.6857
Fold 5	0.8378	0.1622	0.8378

Table 6.4: Performance Measure Results for User #96

Overall, we can see that the majority of the rules have one or two literals in the body. The best solutions are always the negation of a contact. With this in mind, we conclude that the majority of users accept/reject calls based on who the caller is. The result is largely intuitive. Accuracy is the proportion of true results (both true positives and true negatives). The accuracy of such rule lies between 70% and 95%, a measure which is very promising.

On the other hand, precision is defined as the proportion of the true positives against all positive results (both true positives and false positives). In many cases, precision is equal to accuracy meaning that our results are both accurate and close to each other. Intuitively, this means that in each fold, the user's behaviour does not change much, and therefore the difference in both accuracy and precision of the results is minimal.

Figure 6.1, below, portrays the ROC space for a given set of users.

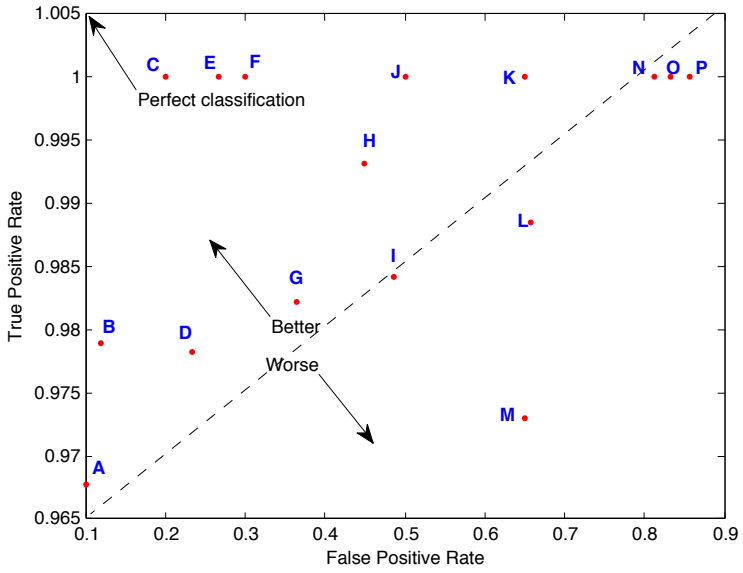


Figure 6.1: The ROC space for a set of users using the Reality Mining dataset

The upper left corner is a match for perfect classification while the lower left corner represents a classifier which has no false positives but gives no true positives. In the ROC space, a point - in our case a user behaviour - is "better" (in terms of classification) than another if it is to the northwest of the first (true positive rate is higher, false positive rate is lower, or both) [Faw06]. The diagonal line $y = x$ represents a random guess strategy. Therefore, any classifier that

appears in the lower right triangle performs worse than random guessing. Taking the above into account and the graph altogether, a user's rule classifies the examples reasonably well. There are very few cases where the solutions perform poorly.

Figure 6.2, below, depicts that the users' predictive accuracy follows a normal distribution. Both the Jarque-Bera test and Kolmogorov-Smirnov confirm that the accuracies come from a normal distribution with an unknown mean and variance with 95% confidence. On a broader scale, this indicates that our learning framework together with the TAL algorithm classify instances reasonably well.

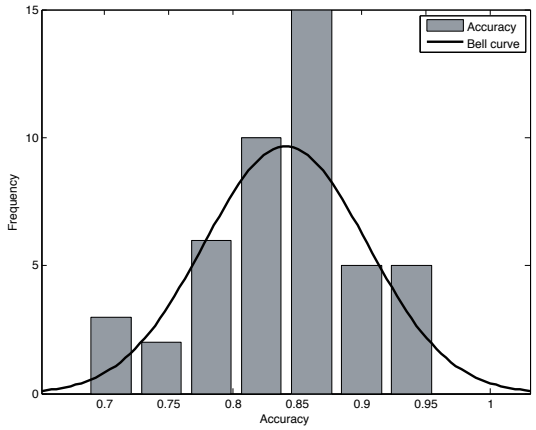


Figure 6.2: The predictive accuracy of users using the Reality Mining dataset

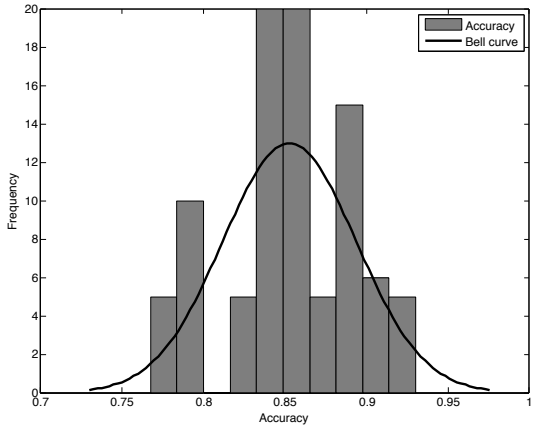


Figure 6.3: Accuracy of users over increasing time windows using the Reality Mining dataset

In addition to the above, we have run tests with increasing time windows (refer to Chapter 4 for more details) to allow more flexibility with regards to the results we obtained. Figure 6.3, above, illustrates the accuracy of users over increasing time windows. We can see that for each user the accuracy remains unchanged. From this we can conclude that the majority of users are consistent with respect to their behaviour no matter what.

6.1.2 The ULearn dataset

We collected data from two users over a period of approximately 2 months. Here we present the best solutions for both users and also show how the results immediately improve when the cover loop approach is used. For this dataset, we also list the Prolog solutions in English as a result of our translation mechanism implemented in ULearn and show by example that the translation is 100% precise. The score for each solution represents the accuracy of each rule as defined in Section 3.6.

User #1

```
solution(-0.6739,[(accept(A,B,_,_,_,_,_,_,_,_,_,_,_):-\+user_is_active(A,B))])

Accept calls: when you're not active
```

Listing 6.5: Best solution obtained without Cover Loop

```
solution(-0.7065, [
 (accept(_,_,_,_,_,F,_,_,_,_,_,_):-F=80),
 (accept(_,_,_,_,_,_,_,_,_,_,V,_):-V=225.0),
 (accept(_,_,A1,_,_,_,_,_,_,_,_,_):-A1=1200490800),
 (accept(_,_,M1,_,_,_,_,_,_,_,_,_):-M1=447515692890),
 (accept(W1,X1,_,_,_,_,_,_,_,_,_,_):-\+user_is_active(W1,X1))
])

Accept calls: when your phone's battery level is at level 80, OR when the light
level is 225.0, OR from contact 1200490800, OR from contact 447515692890,
OR when you're not active
```

Listing 6.6: Best solution obtained with Cover Loop

User #2

```
solution(-0.7065, [
 (accept(P,Q,R,_,_,_,_,_,_,_,Y,_):-\+user_is_active(P,Q),\+R=7517429133,\+Y=1280),
 (accept(A,B,C,_,_,_,_,_,_,_,_,_):-\+user_is_active(A,B),\+C=7517429133,timex_after_h(↩
     B,10))
])

Accept calls: when you're not active, not from contact 7517429133, after 10:00
 o'clock, OR when you're not active, not from contact 7517429133, not when
 your phone's light level is 1280
```

Listing 6.7: Best solution obtained without Cover Loop

```
solution(-0.7717, [
 (accept(A,B,_,_,_,_,_,_,_,_,J,_):-not_at(A,B,1071.8253461),J=225),
 (accept(_,_,Q,_,_,_,_,_,_,_,_,_):-Q=1200490800),
 (accept(_,_,C1,_,_,_,_,_,_,_,_,_):-C1=447515692890),
 (accept(_,_,_,_,_,_,_,_,_,U1,_,_):-U1=0),
 (accept(Y1,Z1,A2,_,_,_,_,_,_,_,H2,_):-\+user_is_active(Y1,Z1),\+A2=7517429133,\+H2↩
     =1280)
])

Accept calls: anywhere unless you're at 1071.8253461, when your light level is
 225, OR from contact 1200490800, OR from contact 447515692890, OR when
 your screen is off, OR when you're not active, not from contact 7517429133, not
 when your light level is 1280
```

Listing 6.8: Best solution obtained with Cover Loop

Immediately we can see the difference between the richness of rules computed with our dataset compared to the rules we have obtained using the Reality Mining dataset. We attribute the differences observed to the following reasons:

- The ULearn dataset is largely noise-less in contrast to the Reality Mining data set which comes with noise and a number of inconsistencies.
- Our learning framework has been predominantly developed prior to the development of ULearn and therefore we knew exactly how to capture data. The Reality Mining dataset required a pre-processing phase.
- The cover loop approach was non-existent during the Reality Mining phase.
- Our dataset is much richer than the Reality Mining dataset, but this does not necessarily mean richer rules, because the domain immediately becomes larger and the search for a richer hypothesis becomes even harder.

Figure 6.4, below, depics the ROC space for the ULearn dataset without the cover loop approach. It represents the classification strategy over the two sets of users. Figure 6.5, on the other hand, represents the strategy with the cover loop approach during the learning process. We can clearly see that the second diagram outperforms the first one in terms of classification accuracy.

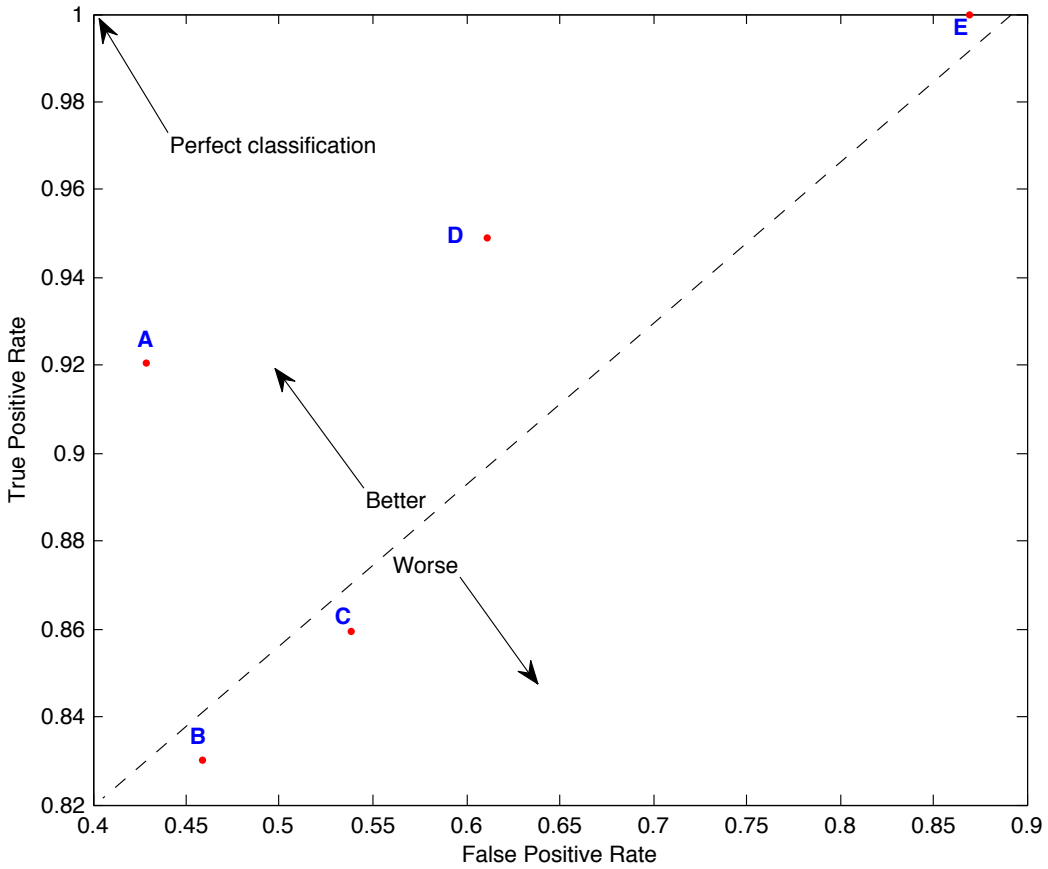


Figure 6.4: The ROC space for the ULearn dataset with cover loop disabled

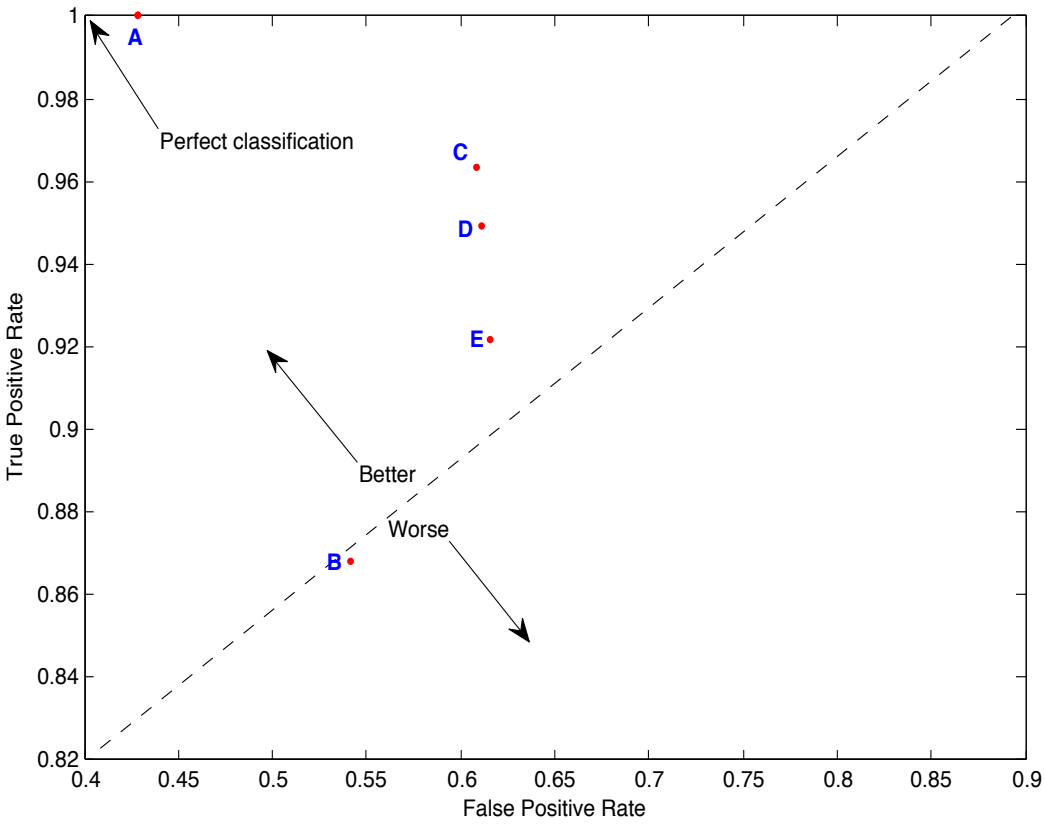


Figure 6.5: The ROC space for the ULearn dataset with cover loop enabled

6.2 Performance evaluation

In this section we assess how the performance of the TAL algorithm varies as it is used to solve problems with increasingly large domains. We assess the performance of the algorithm in terms of the results we obtain based on the language bias, learning parameters, rule revision and present the combinations of these which we believe the algorithm performs best.

Before we evaluate the performance, it is crucial we understand how complex the hypothesis search space is and why it is both a computational and memory intensive task. To illustrate how large the search can be, Figure 6.6, below, depicts a very small part of the search space for computing a relatively simple hypothesis of the following form:

$$penguin(A) \leftarrow bird(A), \neg canfly(A).$$

Domenico Corapi's PhD thesis which is due to appear in late 2011, dedicates an entire section outlining the complexity of the search space mathematically. The complexity of the hypothesis solution relies to a great extent on the learning parameters used in the system. In Chapter 2, we outlined what each parameter represents. In a subsequent section we justify why certain combinations of parameters outperform others.

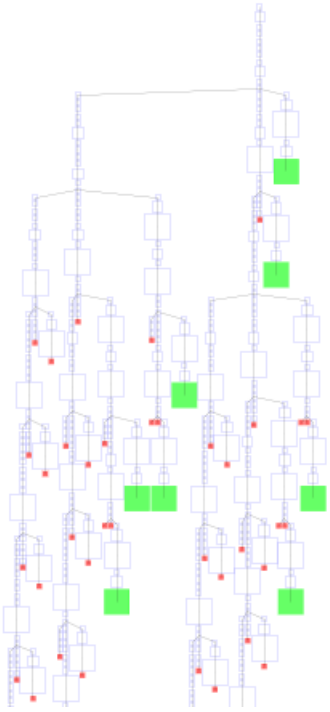


Figure 6.6: Part of the search space for computing the penguin hypothesis

6.2.1 Fixed language bias

We have run two sets of experiments to determine which combination of learning parameters performs best in the context of our domain. Table 6.5 illustrates the four settings we have used. The two sets differ in only one parameter; the cover loop approach being enabled/disabled. In all experiments, the language bias is fixed and the full set can be found in Appendix A. For a full explanation of the learning parameters, refer to Section 2.3.2.

	Setting A	**Setting B**	**Setting C**	**Setting D**
max_depth	5000	10000	50000	500000
max_num_rules	1	1	1	1
max_body_literals	4	4	4	4
timeout	5000ms	30000ms	80000ms	900000ms
number_of_solutions	1	1	1	1
strategy	progol	accuracy	progol	accuracy
single_seed	true	true	true	true
single_seed_ratio	100	50	20	10
solution_pool	1000	1000	1000	1000
xvalidation_folds	5	5	5	5
cover_loop	false/true	false/true	false/true	false/true
loop_threshold	N/A / -0.4	N/A / -0.5	N/A / -0.6	N/A / -0.7

Table 6.5: The learning parameters used in our experiments

Best solution for each setting without Cover Loop

- **Setting A:**

```
solution(-14, [
(accept(_,B,_,_,_,_,_,_,_,_,_):-
   timex_after_h(B,10))
])
```

Translates to: Accept calls after 10:00 o'clock

- **Setting B:**

```
solution(-0.6732, [
(accept(_,B,_,_,_,_,_,_,_,_,_):-
   morning(B)),
(accept(M,N,_,_,_,_,_,_,_,_,_):-
   \+user_is_active(M,N))
])
```

Translates to: Accept calls: during the morning, OR when you're not active

- **Setting C:**

```
solution(-19, [
(accept(_,B,C,_,_,_,_,_,_,_,_):-
   timex_after_h(B,10),
   \+C=99676196)
])
```

Translates to: Accept calls after 10:00 o'clock, when contact is not 99676196

- **Setting D:**

```
solution(-0.7391, [
(accept(A,B,C,_,_,F,_,_,_,_,_):-
   \+user_is_active(A,B),
   \+F=35,
   timex_after_h(B,0),\
   +C=7517429133),

(accept(Q,R,S,_,_,V,_,_,_,_,_):-
   \+user_is_active(Q,R),
   \+V=35,
   timex_after_h(R,10),
   \+S=7517429133)
])
```

Translates to: Accept calls: when you're not active, not when your phone's battery is at level 35, after midnight, not from contact 7517429133, OR when you're not active, not when your phone's battery is at level 35, after 10:00 o'clock, not from contact 7517429133

Best solution per setting with Cover Loop

- **Setting A:**

```
solution(-17, [
(accept(_,_,C,_,_,_,_,_,_,_,_):-
   C=447515692890),
(accept(M,N,_,_,_,_,_,_,_,_,_):-
   \+user_is_active(M,N))
])
```

  Translates to: Accept calls: from contact 447515692890, OR when you're not active

- **Setting B:**

```
solution(-0.7065, [
(accept(_,_,_,_,_,F,_,_,_,_,_):-
   F=80),
(accept(_,_,_,_,_,_,_,_,_,_,V,_):-
   V=225.0),
(accept(_,_,A1,_,_,_,_,_,_,_,_):-
   A1=1200490800),
(accept(_,_,M1,_,_,_,_,_,_,_,_):-
   M1=447515692890),
(accept(W1,X1,_,_,_,_,_,_,_,_,_):-
   \+user_is_active(W1,X1))
])
```

  Translates to: Accept calls: when your phone is fully charged, OR when your light level is 225.0, OR from contact 1200490800, OR from contact 447515692890, OR when you're not active

- **Setting C:**

```
solution(-21, [
(accept(_,_,C,_,_,_,_,_,_,_,_):-
   C=2088811551),
(accept(_,N,O,_,_,_,_,_,_,_,_):-
   timex_after_h(N,10),
   \+O=99676196)
])
```

  Translates to: Accept calls: from contact 2088811551, OR after 10:00 o'clock, not from contact 99676196

- **Setting D:**

```
solution(-0.7936, [
(accept(_,_,_,_,_,_,_,_,_,I,_,_):-
  I=0),
(accept(M,N,O,_,_,_,_,_,_,V,_):-
   \+user_is_active(M,N),
   \+O=7517429133,
   \+V=1280.0)
```

```
])
```

Translates to: Accept calls: when your screen is off, OR when you're not active, not from contact 7517429133, not when your light level is 1280.0

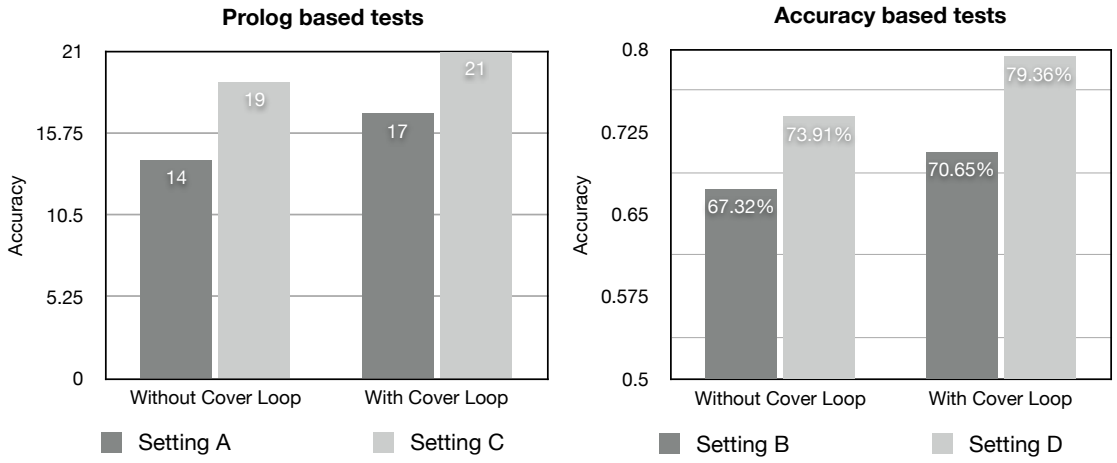


Figure 6.7: The score obtained for each learning parameter setting

Discussion

To a large extent, we can see that Setting D, with or without the cover loop being used, yields the best results. Figure 6.7, above, confirms this. This is mainly attributed to the learning parameters we have chosen for that setting. In particular, what allows for such powerful expressibility in the solution is the *max_depth* parameter which represents the depth of the abductive proof procedure. We can see this as a set of operators that rewrite a state in one or more logically equivalent states. The actual number amounts to the number of times an operator is used to obtain this hypothesis. Intuitively, some of the abductive operators result in changing the current inductive solution. If we increase the depth then we are allowing for more changes so we are exploring more solutions.

Additionally, we have selected a very long timeout. The *timeout* parameter denotes the computation time for the learning task. If the *single_seed* strategy parameter is enabled, then the *timeout* parameter is used for each seed selected. The *single_seed_ratio* parameter specifies the number of seed examples to use. For example, if we have 100 examples, and the ratio is 50, then 2 examples will be randomly picked. This means that the ILP learning task will run once but with both examples used as goals (the seeds). The final solution will result in the conjunction of the two hypotheses; one for each seed. In Setting D we used the lowest ratio. Effectively, the number of seed examples chosen was maximised. When the cover loop is enabled, this effect is duplicated, since the learning task runs at each iteration with seed examples as goals.

Nevertheless, a domain such as ours, with learning parameters similar to the ones we used for Setting D takes hours to compute. From a user perspective this is undesirable, however, the search space cannot, inevitably, get any smaller. Unless we restrict the domain or the language bias used in the hypothesis space, exploring all states remains a computationally intensive task.

On the whole, our findings suggest that the *accuracy* strategy seems to compute much richer rules as opposed to the *progol* strategy. They also lead to the conclusion that the cover loop approach finds both the most accurate and the richest solution. The use of the *loop_threshold* parameter enabled us to discard solutions with less than 70% accuracy. The threshold allows us to present to the user the most representative solutions that best describe their behaviour.

Lastly, it is worth noting that a fixed number for *solution_pool* enabled us to view a number of solutions. With a domain such as ours, there are many more available and equivalent solutions which we could have easily output.

6.2.2 Fixed learning parameters

In this section, we assess the level of impact the language bias has on a fixed set of learning parameters. The learning parameters used are the same as the ones in Setting C in Table 6.5 with two modifications. We changed the strategy to *accuracy* and increased the *max_depth* to 80000. There is no need to use Setting D or the cover loop approach, because here we are trying to assess the effect of the language bias on the learning (i.e., what language bias is most useful in defining mobile user behaviour). For brevity, we informally divide the language bias into 6 relevant categories. A full list of mode declarations used for each setting can be found in Appendix B.

	Bias A	**Bias B**	**Bias C**	**Bias D**
Date/Time	✔	✗	✔	✗
Contact	✗	✔	✔	✗
Location	✔	✗	✔	✔
User activity	✗	✔	✗	✔
Battery/Brightness/Light	✔	✗	✗	✗
Volume/Vibrator	✗	✔	✗	✔

Table 6.6: The changes made in the language bias for our experiments

Best solution for each language bias category

- **Bias A:**

```
solution(-0.5847, [
(accept(_,B,_,_,_,F,G,_,_,J,_):-
   time_after_h(B,10),
   \+F=34,
   \+G=7,
   \+J=160)
])
```

  Translates to: Accept calls: after 10:00 o'clock, not when your phone's battery is at level 34, not when screen brightness is 7, not when your light level is 160

- **Bias B:**[†]

[†]There were plenty of best equivalent solutions. Here some of the most interesting ones.

```
solution(-0.6087, [
(accept(A,B,C,_,_,_,_,_,_,_,_):-
   \+C=99676196,user_is_active(A,B)),
(accept(_,_,Q,_,_,_,_,_,_,_,_):-
   \+Q=99676196)
])
solution(-0.6087, [
(accept(_,_,C,_,_,_,_,_,_,_,_):-
   \+C=99676196,C=1614517551),
 (accept(_,_,P,_,_,_,_,_,_,_,_):-
    \+P=99676196)
])
solution(-0.6087, [
(accept(A,B,C,_,_,_,_,_,_,_,_):-
   \+C=99676196,
   \+in_call(A,B))
])
solution(-0.6087, [
(accept(_,_,C,D,_,_,_,_,_,_,_):-
   \+C=99676196,
   \+D=0)
])
```

Translates to:
Accept calls: not from contact 99676196, when you're active, OR not from contact 99676196
Accept calls: not from contact 99676196, from contact 1614517551, OR not from contact 99676196
Accept calls: not from contact 99676196, when you're not in call
Accept calls: not from contact 99676196, not when your phone is on silent

- **Bias C:**[†]

```
solution(-0.6956, [
(accept(_,B,C,_,_,_,_,H,_,_,_):-
    time_after_h(B,10),
    H=0,
    \+C=99676196,
    \+morning(B)),
(accept(_,Q,R,_,_,_,_,_,_,_,_):-
    time_after_h(Q,1),
    \+R=99676196)
])
```

Translates to: Accept calls: after 10:00 o'clock, when you don't have your headset on, not from contact 99676196, except in the morning, OR after 01:00 o'clock, not from contact 99676196

[†]There were plenty of best equivalent solutions

- **Bias D:**

```
solution(-0.6435, [
(accept(_,_,_,D,E,_,_,_,_,_,_,_):-
   \+E=0,
   \+D=0)
])
```

Translates to: Accept calls: not when your phone is off vibrate,not when your phone is on silent

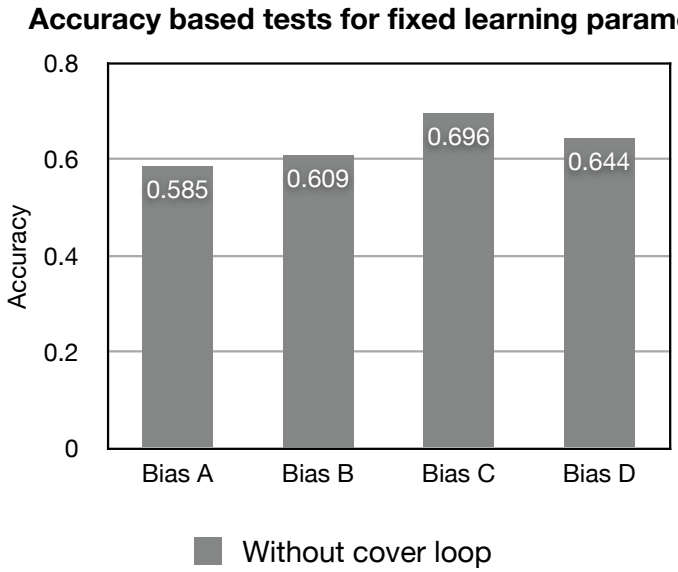


Figure 6.8: The accuracy of each solution obtained from each language bias setting

Discussion

While all different language bias settings have computed satisfying results, our experiments reveal that the concept of date/time, contact, together with user activity (Bias C) yield both the most meaningful and accurate rules. Figure 6.8 illustrates the accuracies obtained with each language bias setting. Location, and volume settings come second. This is both in terms of applicability and from a user perspective. We have learned that the recording of information such as the battery level, the screen brightness or the light level from the phone's light sensor are meaningless to the user. Even though these can be associated with a rule, they provide no significant value to the rules.

Furthermore, the accuracy of the hypotheses seems to be higher when these categories are part of the language bias. A survey we conducted validates these results to some extent. One of the questions asked users to list 4 factors that influence their decision-making process on accepting/rejecting calls. The user's current activity turns out to be the most important factor, together with the caller's identity. More details about the survey can be found in Section 6.4.

6.2.3 Rule revision

We present a subset of training examples for two consecutive recording sessions. Listing 6.9 illustrates part of this process. We show a subset of the examples that highlight the difference. In the first session, the user accepts calls from anyone, before 15:00, when the phone is off charge and when the phone is not on silent mode.

```
Session 1:
...
example(accept([18,3,2011],[11,49],447515692890,7,2,-1,30,0,1,160, 0),1).
example(accept([18,3,2011],[14,8],447515692890,5,2,-1,30,0,0,1280.0,0),1).
example(accept([19,3,2011],[15,18],-1,0,0,-1,30,0, 0,640, 1),-1).
example(accept([19,3,2011],[15,47],-1,0,0,-1,30,0,0,225,1),-1).
...

Revisable solution:

accept(_,B,_,D,_,_,_,_,_,_,K):-\+D=0,\+K=1,time_before_h(B,15).

Accept calls: not when your phone is on silent, not when your phone is charging,
before 15:00 o'clock
```

Listing 6.9: Examples and solution for Session 1

In the second session, the two new examples indicate that the user has not accepted a call from neither contact 447515692890 or contact -1 (a contact not in the user's address book) later in the evening. Additionally, the phone was neither on silent mode nor on charge when the two phone calls took place, yet they were unanswered.

There are more than two revised solutions. However, we only present two in Listing 6.10 below. The first revised solution asks us to delete all the literals from the body of the rule. Neither the volume level nor the charge status of the mobile device serve as factors in answering phone calls. A new hypothesis is computed that states that phones are being answered if the caller is 447515692890 and the call does not take place in the evening.

The second revised solution asks us to delete the first two literals from the body. The rule states that all calls are answered any time before 15:00 o'clock.

```
Session 2:
...
example(accept([20,3,2011],[19,45],447515692890,7,2,-1,30,0,1,160,0),-1).
example(accept([20,3,2011],[21,28],-1,5,2,-1,30,0,0,1280,0),-1).
...

Revised solutions:

(del(1,1)),(del(1,2)),(del(1,3)),(accept(_,B,C,_,_,_,_,_,_,_,_,_):-C=447515692890,
\+evening(B)).
Accept calls: from contact 447515692890, except in the evening

(del(1,1)),(del(1,2)),(accept(_,B,C,_,_,_,_,_,_,_,_,_):-time_before_h(B,15)).
Accept calls: before 15:00 o'clock
```

Listing 6.10: Examples and solution for Session 2

6.2.4 Observational accuracy

We have spent a large amount of time on the development of ULearn to make it as robust as possible and eliminating any possibility of down-time. While the application is recording data,

we have made no observations that the phone crashes due to any abrupt usage. Even if the phone abruptly shuts down or runs out of battery while a recording session is in place, the application picks up where it last left off thereby eliminating any inconsistencies both in terms of usage and in terms of data loss.

6.3 Usability evaluation

One of the aims of this project is to produce an application which would perform data acquisition on a mobile device, make use of our learning framework and prompt the learning results to the user. The use of ILP techniques to model mobile user behaviours has not been explored before and our application extends its implicit use to a larger audience.

The application is designed for non-technical users and as such issues of human computer interaction are important in any graphical user interface environment. To make a valid assessment of our application, we evaluate it against Nielsen's heuristics [NM90]. We describe how ULearn obeys each heuristic and score each heuristic accordingly.

6.3.1 Nielsen's heuristics

There are ten general principles for user interface design. The term "heuristics" is used because they are more in the nature of rules of thumb than specific usability guidelines. We compare our system with these defined principles as follows:

1. **Visibility of system status:** *The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.*

   Feedback is a key feature for any application that engages the user somehow. ULearn copes well with providing feedback to the users at all times. Whenever we ask the user's involvement, we provide simple informative dialogues. An example of these dialogues is seen in Figure 6.9 below.

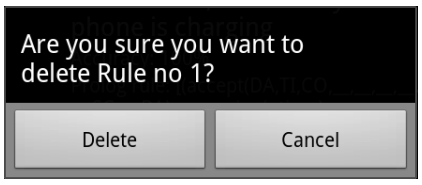


Figure 6.9: Informing the user that a rule is about to be deleted.

   Additionally, we have managed to prompt learning results on screen quickly, taking into account the computational power required for such task.

   Evaluation: 'Good'

2. **Match between system and the real world:** *The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.*

We have done our best to minimise misconceptions. We like to think that our system conveys simple and comprehensive instructions that will reduce user misinterpretation and annoyance.

Evaluation: 'Adequate'

3. **User control and freedom:** *Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.*

   Our application partially sticks to this standard. It is just as easy to exit the application, much like the way you access it. Currently, users do not have the ability to control the type and frequency of data being collected but its importance is acknowledged and is a feature which can be added in future versions. Due to time constraints, we have concentrated on the core functionality of this application; prompting rules to the users.

   Evaluation: 'Poor'

4. **Consistency and standards:** *Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.*

   ULearn does not introduce any ambiguity at any time. User confusion should not arise at all when using the system. The Android platform allows us to comply with mobile interface standards and validate against them to ensure a smooth transition from one view to the other. For example, all action controls for each associated view are performed through the use of the menu button. In cases, where actions can be performed on items of a list, a long click event pops a context menu to the user. Help/Error/Informative messages all use the same language throughout imposing a consistent environment across. An example of an Options menu is shown in Figure 6.10 below.

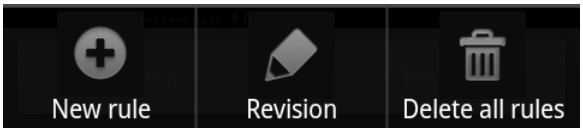


Figure 6.10: Each view is associated with an Options menu

   Evaluation: 'Good'

5. **Error prevention:** *Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.*

   The possibility of errors in the system as a result of the user is relatively small. We have taken extra care to prevent users from mishandling the system. In cases where the user's input was required, limitations have been imposed by validating the input before continuing. This prevents the users from any undesired frustration but also prevents problems from occurring in the first place making it a valuable asset in our attempt to evaluate the system.

   Evaluation: 'Good'

6. **Recognition rather than recall:** *Minimise the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.*

   The user's thinking should be effortless. We have made a tremendous effort to diminish the steps one needs to bring the system into a usable state. We believe that there are no stages in the application whereby a user has to think for performing a task. For example, when creating a new rule, users are given a list of all the options available, so the syntax or the literals do not have to be remembered. All icons in the system have text next to them, thus reducing the memory load for remembering what each icon represents.

   Evaluation: 'Good'

7. **Flexibility and efficiency of use:** *Accelerators - unseen by the novice user - may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.*

   Our system is efficient enough as it is, and hence there is no need to favour experienced users. We have not included any *accelerators* in our system as Nielsen likes to call them. Our system treats all users, both novices and experts, equally and does not provide any shortcuts.

   Evaluation: 'Good'

8. **Aesthetic and minimalist design:** *Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.*

   This is probably one of the most important principles. Keeping things simple suggests that it will appeal to the majority of a system's user-base. We have tried to avoid the use of fancy graphics and design that will boast the appearance of the system. Whilst the initial thoughts of users may be positive and enthusiastic for such dazzling effects, the purpose of the system is somehow lost and its functionality becomes insignificant.

9. **Help users recognise, diagnose and recover from errors:** *Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.*

   We have constructed clear and informative messages should an error occur. No error/warning messages that require the knowledge of an expert are passed onto the user. The system intends to foster a safe, user-friendly environment that detracts from the user any concerns which emerge from such behaviour.

   Evaluation: 'Adequate'

10. **Help and Documentation:** *Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focussed on the user's task, list concrete steps to be carried out, and not be too large.*

    Although, our application is as simple and straightforward as it can be, there exists a User Guide within the application which eliminates the gap between "thinking of knowing" and understanding. It provides detailed, yet simple, instructions on how to use the application,

taking into account the range of users; from novices to experts. It also provides details on what it does and how it is used.

Evaluation: 'Adequate'

6.4 Suitability study

We have conducted a survey[1] over a period of approximately three weeks. The main purpose for this survey is to identify whether this type of application can be rendered as a popular application that users would find it useful enough to become part of their lives. Throughout the three week period we have collected approximately 60 responses with a mixture of results, interesting enough to include in this report.

We have split the evaluation of the survey into three parts:

1. Understanding the user.
2. User confidence on data collection.
3. User confidence on mobile device autonomy.

6.4.1 Understanding the user

To evaluate the situation objectively, we first take a look at the people who have filled in the survey. From the three questions below we can see that the majority receive calls *occasionally* to *very often* and *rarely* reject phone calls. We immediately see that the benefits of a mobile application automatically rejecting phone calls are somewhat lessened.

We have also asked the users to indicate the level of importance for key factors that influence the user's decision on rejecting phone calls. We find that the user's *current activity* is the most important factor followed by *who the caller* is. The user's location also plays a considerably important factor in the decision-making process. Furthermore, the surrounding people the user is with is also another factor, but not as effective as the rest.

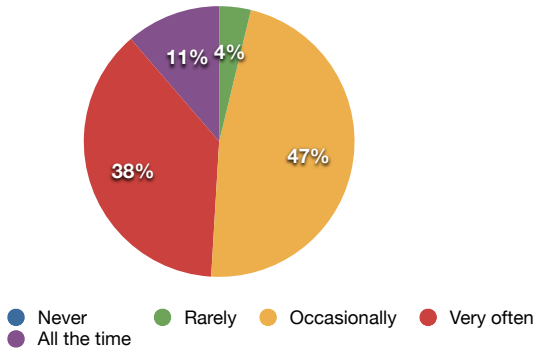


Figure 6.11: How often do you receive calls?

[1]The survey can be found online at: http://www.surveymonkey.com/s/5RZJZTK

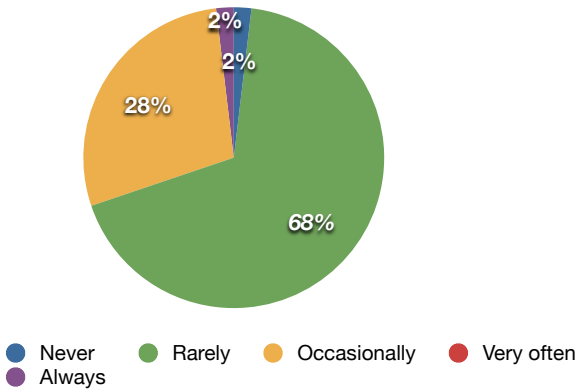


Figure 6.12: Of these phone calls, how often do you reject them?

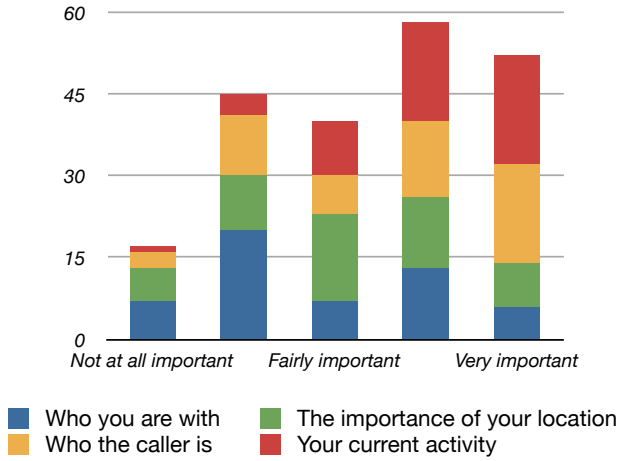


Figure 6.13: On what basis do you reject incoming phone calls? What's their importance?

Finally, our survey is biased towards the people between the age of 18 to 30. While this means we were unable to model the importance of phone calls in other age categories, the category most of our users fall in is the most accustomed one with smart phones in general.

6.4.2 User confidence on data collection

We were very keen to capture how comfortable users would be with themselves if an unprecedented amount of data was collected, with their permission. Data they might or might not have control over. The majority of the users are very concerned that data would be collected and stated that would not own such an application.

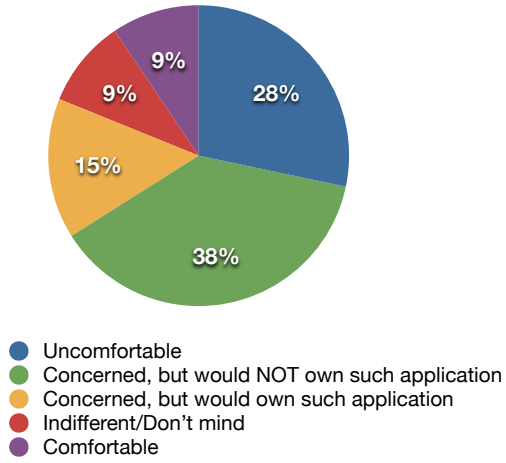


Figure 6.14: How comfortable would you be if you owned an Android/iPhone app which records your day-to-day activities implicitly? (e.g., phone activity, location, phone settings, phone call state et.c.)

6.4.3 User confidence on mobile device autonomy

We were also interested to see how such an application would cope had it been available for download. We were particularly curious to find whether the use of such an application would be prevalent for a while to become popular. Unfortunately the majority of the users stated that such an application would not be suitable to them.

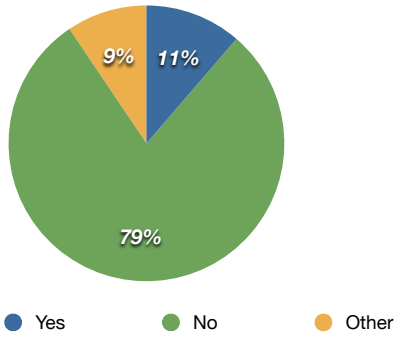


Figure 6.15: Consider an example where a mobile phone application would be able to automatically accept or reject phone calls for you based on your mobile phone behaviour. Would you use it?

Some of the survey takers made the additional notable comments:

- "I would use it given that I had accepted the people and times for which the application would reject my calls".
- "Accepts yes, but not reject".
- "Only if I could override it, if necessary".
- "I may use it, depending on location, caller and nature/purpose of call".
- "What if I don't hear the phone and it picks up?"

While this project has principally concentrated on one mobile user behaviour, it seems that if we generalise this notion and ask users of their opinion about an application executing specific tasks for them (with their consent), it turns out that the majority would use it. Some have stated that it would highly depend on the tasks.

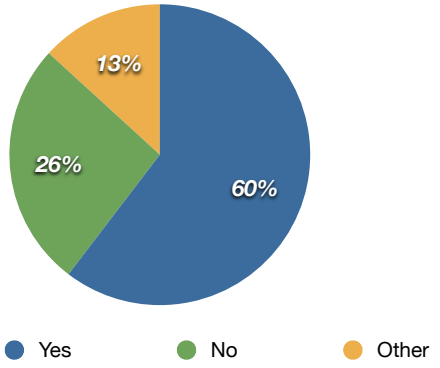


Figure 6.16: Would you use a mobile phone application which carries out specific tasks for you (with your consent)?

6.4.4 Discussion

We can see that survey we have conducted has a variety of implications. On the whole, it is notable that data acquisition and automatic execution of some tasks are major issues concerning the users' privacy considerably. While such an application seeks to minimise intrusiveness, it may be the case that there exists an inverse relationship between minimising the user's involvement but maximising user concerns over privacy and autonomy issues. A more thorough survey over a larger variety of user demographics would have certainly been preferred for a more insightful feedback. Nevertheless, as the work in the field is relatively new, there is no indication whether or not such applications can be successful in the future. This may depend on the elimination of the two major concerns mentioned above, and is simply a matter of time.

6.5 Summary

In this chapter we have evaluated and summarised our work throughout the entire duration of the project. Our results have shown that the use of ILP is applicable for defining mobile user behaviours. We have presented a number of meaningful rules with the level of accuracy obtained for each user. We have then shown how the rules differ in terms of richness from the data collected with the ULearn application. We have subsequently explained the effects of learning when using fixed learning parameters and fixed language bias and concluded with a combination which we believe achieves maximum performance. Finally, a user interface evaluation was carried out taking into account Nielsen's heuristics followed by a detailed survey about the applicability of the ULearn application in the mainstream market.

7 Conclusion

To conclude our work, we summarise our achievements both in terms of deliverables and in terms of learning. We further outline future work and extensions relevant to the project and finally draw an end with a touch of reflection. Using our learning framework and ULearn we have successfully shown that all the requirements are met:

- The development of a learning framework for learning mobile user behaviours by means of Inductive Logic Programming.
- The applicability of the framework to large, existing and inconsistent data.
- The applicability of the framework to data collected from our own developed Android application.
- The automated extraction of mobile user behaviour rules through data acquisition.
- The development of an end-to-end system able to collect, extract and process data, and present the learning results to the user to confirm or reject the proposed rules.
- The effect of revision, learning parameters and the language bias in the context of learning rich and meaningful rules.

Throughout this report we have demonstrated that ILP is applicable for defining mobile user behaviours. With the appropriate definition of a relevant domain of discourse, language bias and background knowledge our evaluation results indicate that ILP performs well when dealing both with large domains and large amount of data. We have ascertained that ILP is better at learning hypotheses where attribute-based methods fail. Our investigation on whether statistical machine learning can provide structures in the data, that would have otherwise been hidden, has established that ILP performs remarkably better when dealing with multi-relational data. As a result of the two applications of our learning framework, detailed in Chapter 4 and Chapter 5 respectively, we have shown that it is, to a great extent, immune to major changes when exposed to different datasets. However, this does not, by any means, imply that each dataset is identical; some amount of work is required to extract the data specified by our framework. Finally, TAL,

the in-house tool utilised throughout the project, has proven to be a powerful non-monotonic ILP system that tolerates noise and scales extremely well with large domains and data.

7.1 Future work

- **Enriching the background knowledge/language bias:** We have captured enough information with ULearn that we unfortunately do not use in a very interesting way. Our background knowledge can be enhanced further by defining the user's location, user's habits et.c., in terms of user-defined categories. Our system can then prompt the user for such information at the beginning of each recording session. Subsequently, some of these predicates are added to the language bias to make the language of the hypotheses even more meaningful.
- **More than one rule:** In this project, we have chiefly concentrated on one type of rule. Other rules can also be defined. This requires additional constructs in the background knowledge and language bias to allow for such rules. Some examples include application usage characteristics, how the user changes the sound settings on the phone and how the user moves between locations.
- **Location prediction:** A very interesting extension to this project is the usage of location data by means of GPS coordinates to predict the user's next location in T' time units given the user is at location X at time T. The following steps enable us to achieve this. First, we cluster location in terms of some arbitrary grouping. The clustering algorithm provides us with the centroids of the clusters which are then encoded in Prolog as facts. A predicate of the form `belongstoCluster/3` can be constructed that given a location, outputs the cluster it belongs to. Second, our positive examples consist of cases where the user is moving. Negative examples consist of cases where the user is either not moving at all, or the distance traversed is less than a desirable threshold.
- **Resampling:** Due to the time constraints, we have only been able to cross validate our results and perform ROC analysis. A potential improvement to the accuracy/richness of the rules constructed is through the use of resampling. We can achieve this with two different methods. Given a training set D of size n, *bagging* generates m new training sets Di each of size $n' <= n$ by sampling examples from D uniformly and with replacement. This kind of sample is known as a *bootstrap* sample. The m samples are then used separately in an ILP setting and their respective solutions are then combined by voting. Alternatively, we can sample a bootstrap set, test this set with our learning algorithm and increase the probability of selection for misclassified examples in further bootstrap samples. This technique is known as *boosting*.
- **Additional datasets:** So far we have only focussed on learning rules from two different datasets; one existing, one collecting. The project can benefit from exposure to additional datasets relevant to our work. This allows us to compare the datasets and rules in terms of accuracy and richness. A valid conclusion can then be drawn based on our findings.
- **Preventing concept drifting:** Mobile user behaviours unquestionably change over time in unforeseen ways. This often becomes problematic because the predictive accuracy of our rules decreases as time passes. To prevent deterioration of prediction accuracy over time we can do a number of things. We can either re-run the algorithm by using only the most recent training examples or add new background knowledge which may result in explaining

the causes of concept drift. Unfortunately, the latter requires the user's intervention and one of our main aims is to identify such behaviour with minimal intervention.

- **Use of sensors:** While we have made use of the light sensor when collecting data with the ULearn application, we haven't modelled the recorded light level data in a clever way. Using a number of mobile sensors, we can create models which enable us to uncover interesting information that is not provided by the phone itself. Such information includes, whether the user is at home (there is not too much noise), the user is outside (there is too much noise), the phone is in the user's pocket/bag (light level is very low), the phone is upside down (gyroscope information) et.c. Our language bias can then be enriched significantly and the context can be even more relevant to the user.
- **Data control:** From a user's perspective, ULearn captures almost everything triggered by the user. An interesting extension that can be added to the application is to allow the user to control what information is captured. This provides a sense of control and flexibility to the user. Because of this control, rules are customised specifically to the user's needs; a direct implication of this extension that adds a whole new dimension of possibilities.

7.2 Closing remarks

We hope that our innovative idea of learning mobile user behaviours through the use of ILP can be used as a starting point for future work in the area. Learning rules is a only a preliminary step. Our evaluation demonstrates evidence of promising rules based on results we have obtained. The primary use of such rules is within frameworks that allow the learning of privacy policy rules from past usage in an automated fashion and adapt the system to the user's needs. This poses as the real challenge and rule adaptive systems may soon emerge as the new trend in mobile devices that make people's lives easier and far more controlled.

Bibliography

[AHK+03] P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter. EPAL 1.1. 2003.

[AIS93] R. Agrawal, T. Imieliński, and A. Swami. Mining Association Rules Between Sets of Items in Large Databases. In *Proceedings of the 1993 ACM SIGMOD international conference on management of data*, SIGMOD '93, pages 207–216, New York, NY, USA, 1993. ACM.

[Alp04] E. Alpaydin. *Introduction to Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2004.

[AS94] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[BBC] Over 5 billion mobile phone connections worldwide. http://www.bbc.co.uk/news/10569081. [Accessed June 15, 2011].

[Blo10] H. Blockeel. Bias Specification Language. In C. Sammut and G.I. Webb, editors, *Encyclopedia of Machine Learning*, pages 98–100. Springer, 2010.

[BM95] I. Bratko and S. Muggleton. Applications of Inductive Logic Programming. *Commun. ACM*, 38:65–70, November 1995.

[BRL07] A.K. Bandara, A. Russo, and E.C. Lupu. Towards Learning Privacy Policies. In *Policies for Distributed Systems and Networks, 2007. POLICY '07. Eighth IEEE International Workshop on*, page 274, 2007.

[Bur08] E. Burnette. *Hello, Android: Introducing Google's Mobile Development Platform*. Pragmatic Bookshelf, 2008.

[Cra02] L. Cranor. *Web privacy with P3P*. O'Reilly, 2002.

[CRL10] D. Corapi, A. Russo, and E.C. Lupu. Inductive Logic Programming as Abductive Search. In *Technical Communications of the 26th International Conference on Logic Programming*, 2010.

[CRR+08] D. Corapi, O. Ray, A. Russo, A. Bandara, and E.C. Lupu. Learning Rules from User Behaviour. In *2nd International Workshop on the Induction of Process Models*, September 2008.

[DM94] L. DeRaedt and S. Muggleton. Inductive Logic Programming: Theory and Methods. In *Journal of Logic Programming*, pages 19/20:629–680, 1994.

[Dze03] S. Dzeroski. Multi-Relational Data Mining: An Introduction. *SIGKDD Explorations*, 5(1):1–16, 2003.

[EPL07] N. Eagle, A. Pentland, and D. Lazer. Inferring Social Network Structure using Mobile Phone Data. *PNAS*, 2007.

[Faw06] T. Fawcett. An Introduction to ROC Analysis. *Pattern Recognition Letters*, 27(8):861 – 874, 2006. ROC Analysis in Pattern Recognition.

[FCS+03] Nuno Fonseca, Vitor Santos Costa, O Silva, Nuno Fonseca, Vitor Santos Costa, O Silva, and Rui Camacho. On the Implementation of an Ilp System with Prolog, 2003.

[Goo11] Google. The Android Software Development Kit and the Developer's Guide. http://developer.android.com/, 2011. [Accessed June 08, 2011].

[HFH+09] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H. Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explorations Newsletter*, 11:10–18, November 2009.

[HPY00] J. Han, J. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, SIGMOD '00, pages 1–12, New York, NY, USA, 2000. ACM.

[HTF09] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition (Springer Series in Statistics)*. Springer, 2nd ed. 2009. corr. 3rd printing edition, February 2009.

[KC03] Robert Kowalski and Keith L. Clark. Logic programming. In *Encyclopedia of Computer Science*, pages 1017–1031. John Wiley and Sons Ltd., Chichester, UK, 2003.

[KKT93] A. C. Kakas, R. A. Kowalski, and F. Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 2(6):719–770, 1993.

[Lav98] Nada Lavrac. Computational logic and machine learning: A roadmap for inductive logic programming. *Technical Report, J. Stefan Institute*, pages 47–73, 1998.

[LMR92] J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming*. MIT Press, Cambridge, MA, USA, 1992.

[Mit97] T.M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[Mug95] S. Muggleton. Inverse Entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.

[Mug97] S. Muggleton. Learning from Positive Data. In *Selected Papers from the 6th International Workshop on Inductive Logic Programming*, pages 358–376, London, UK, 1997. Springer-Verlag.

[Mug08] S. Muggleton. Introduction to Inductive Logic Programming: Lectures 1 and 2. http://ibug.doc.ic.ac.uk/courses/machine-learning-course-395/, 2008. [Accessed Jan. 09, 2011].

[NM90] J. Nielsen and R. Molich. Heuristic Evaluation of User Interfaces. http://www.phillips.rmc.ca/courses/459-2007/lectures/03-heuristic-list.pdf, 1990. [Accessed June 15, 2011].

[Pan10] M. Pantic. Machine Learning Lecture Notes. http://ibug.doc.ic.ac.uk/courses/machine-learning-course-395/, 2010. [Accessed Jan. 08, 2011].

[PRI] Privacy Rights Management for Mobile Applications. http://primma.open.ac.uk. [Accessed Jan. 07, 2011].

[QCJ93] J. R. Quinlan and R. M. Cameron-Jones. Foil: A Midterm Report. In *In Proceedings of the European Conference on Machine Learning*, pages 3–20. Springer-Verlag, 1993.

[Rov05] M. Rovatsos. Knowledge Engineering Lecture Notes. www.inf.ed.ac.uk/teaching/courses/ke/slides/ke18-ilp-2x2.pdf, 2005. [Accessed June 11, 2011].

[Sak01] C. Sakama. Nonmonotonic Inductive Logic Programming. In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning*, LPNMR '01, pages 62–80, London, UK, 2001. Springer-Verlag.

[SAW94] Bill N. Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *IN PROCEEDINGS OF THE WORKSHOP ON MOBILE COMPUTING SYSTEMS AND APPLICATIONS*, pages 85–90. IEEE Computer Society, 1994.

[SS86] L. Sterling and E. Shapiro. *The Art of Prolog: Advanced Programming Techniques*. MIT Press, Cambridge, MA, USA, 1986.

[SSK+86] M. J. Sergot, F. Sadri, R. A. Kowalski, F. Kriwaczek, P. Hammond, and H. T. Cory. The british nationality act as a logic program. *Commun. ACM*, 29:370–386, May 1986.

[Wei91] Mark Weiser. The computer for the 21st century. *Scientific American*, 265(3):66–75, January 1991.

[Zak00] M.J. Zaki. Scalable Algorithms for Association Mining. *IEEE Trans. on Knowl. and Data Eng.*, 12:372–390, May 2000.

Appendices

A

Partial learning file

```
1
2 :- multifile option/2, builtin/1.
3 :- dynamic accept/11.
4
5 builtin(subtractTime(_, _, _)).
6 builtin(addTime(_, _, _)).
7 builtin(dow(_, _, _, _)).
8 builtin(divmod(_, _, _, _)).
9 builtin(member(_,_)).
10 builtin(user_is_active(_, _)).
11 builtin(at(_, _, _)).
12 builtin(nearDevice(_, _, _)).
13 builtin(not_nearDevice(_, _, _)).
14 builtin(not_at(_, _, _)).
15
16 /*
17  *    Learning parameters
18  */
19
20 option(max_body_literals, 4).
21 option(max_num_rules, 1).
22 option(number_of_solutions, 1).
23 option(single_seed_ratio, 50).
24 option(solution_pool, 100).
25 option(max_depth, 50000).
26 option(single_seed, true).
27 option(strategy, custom_accuracy_based).
28 option(xvalidation_folds, 5).
29 option(ic_check, true).
30
31 /*
32  *    Mode declarations
33  */
34
35 /* names and no_ground_constants have been omitted for layout reasons */
36
37 modeh(accept(+date, +timex, +contact, +volume, +vibrator, +battery_level,
38  +screen_brightness, +headset, +screen_status, +light_level, +battery_charging)).
39
40 modeb(=(+contact, #contact)).
41 modeb(\+(=(+contact, #contact))).
42 modeb(=(+volume, #volume)).
```

```
43 modeb(\+(=(+volume, #volume))).
44 modeb(=(+vibrator, #vibrator)).
45 modeb(\+(=(+vibrator, #vibrator))).
46 modeb(=(+battery_level, #battery_level)).
47 modeb(\+(=(+battery_level, #battery_level))).
48 modeb(=(+screen_brightness, #screen_brightness)).
49 modeb(\+(=(+screen_brightness, #screen_brightness))).
50 modeb(=(+headset, #headset)).
51 modeb(\+(=(+headset, #headset))).
52 modeb(=(+screen_status, #screen_status)).
53 modeb(\+(=(+screen_status, #screen_status))).
54 modeb(=(+light_level, #light_level)).
55 modeb(\+(=(+light_level, #light_level))).
56 modeb(=(+battery_charging, #battery_charging)).
57 modeb(\+(=(+battery_charging, #battery_charging))).
58
59 modeb(weekday(+date)).
60 modeb(\+weekday(+date)).
61 modeb(weekend(+date)).
62 modeb(\+weekend(+date)).
63 modeb(evening(+timex)).
64 modeb(\+evening(+timex)).
65 modeb(morning(+timex)).
66 modeb(\+morning(+timex)).
67 modeb(afternoon(+timex)).
68 modeb(\+afternoon(+timex)).
69 modeb(in_call(+date, +timex)).
70 modeb(\+in_call(+date, +timex)).
71 modeb(at(+date, +timex, #cell)).
72 modeb(not_at(+date, +timex, #cell)).
73 modeb(nearDevice(+date, +timex, #device)).
74 modeb(not_nearDevice(+date, +timex, #device)).
75 modeb(neighbourhood(+cell, #cell)).
76 modeb(user_been_in(+date, +timex, +cell)).
77 modeb(user_is_active(+date, +timex)).
78 modeb(\+user_is_active(+date, +timex).
79 modeb(phone_charging(+date, +timex)).
80 modeb(\+phone_charging(+date, +timex)).
81 modeb(phone_on(+date, +timex)).
82 modeb(\+phone_on(+date, +timex)).
83 modeb(user_is_using_app(+date, +timex, #app)).
84 modeb(\+user_is_using_app(+date, +timex, #app)).
85 modeb(timex_before_h(+timex, #hour)).
86 modeb(timex_after_h(+timex, #hour)).
87
88
89 /*
90  *  Integrity Constraints
91  *  battery_level, contact and screen_status are names we declare
92  * in the mode declarations.
93  * e.g., modeb(=(+contact, #contact), [name(conact)].
94  *
95  */
96
97 ic :-
98    pr(_, R),
99    member((battery_level, _, _), R),
100   member((contact, _, _), R).
101
102 ic :-
103    pr(_, R),
104    member((battery_level, _, _), R),
105    member((screen_status, _, _), R).
106
107
108 /*
109  *  Background knowledge
110  */
111
112 /* Dates */
113
114 year(Y):-
115    Y in 2000..2011.
116
```

```
117 month(M):-
118     M in 1..12.
119
120 day_of_month(D):-
121     D in 1..31.
122
123 date([D, M, Y]):-
124     year(Y),
125     month(M),
126     day_of_month(D).
127
128 /* computes date of the week given D, M, Y */
129 dow(G,A,Y,Isim):-
130     A < 3,
131     I is A+12,
132     M is Y-1,!,
133     dow(G,I,M,Isim).
134
135 dow(G,U,K,H):-
136     A is U+1,
137     I is K//400,
138     J is K//100,
139     T is K//4,
140     L is I+T+6+K-J,
141     P is A*26,
142     O is P//10,
143     R is L+O+G,H is R mod 7,!.
144
145 date_before([D, M, Y], [D1, M1, Y1]):-
146     date([D, M, Y]),
147     date([D1, M1, Y1]),
148     Y #< Y1.
149
150 date_before([D, M, Y], [D1, M1, Y1]):-
151     date([D, M, Y]),
152     date([D1, M1, Y1]),
153     Y = Y1,
154     M #< M1.
155
156 date_before([D, M, Y], [D1, M1, Y1]):-
157     date([D, M, Y]),
158     date([D1, M1, Y1]),
159     Y = Y1,
160     M = M1,
161     D #< D1.
162
163 date_after([D, M, Y], [D1, M1, Y1]):-
164     date([D, M, Y]),
165     date([D1, M1, Y1]),
166     Y #> Y1.
167
168 date_after([D, M, Y], [D1, M1, Y1]):-
169     date([D, M, Y]),
170     date([D1, M1, Y1]),
171     Y = Y1,
172     M #> M1.
173
174 date_after([D, M, Y], [D1, M1, Y1]):-
175     date([D, M, Y]),
176     date([D1, M1, Y1]),
177     Y = Y1,
178     M = M1,
179     D #> D1.
180
181 /* Time */
182
183 hour(H):-
184     H in 0..23.
185
186 minute(M):-
187     M in 0..59.
188
189 second(S):-
190     minute(S).
```

```
191
192 timex([H, M]):-
193     hour(H),
194     minute(M).
195
196 timex_before([H, M], [H1, M1]):-
197     timex([H, M]),
198     timex([H1, M1]),
199     H #< H1.
200
201 timex_before([H, M], [H1, M1]):-
202     timex([H, M]),
203     timex([H1, M1]),
204     H = H1,
205     M #< M1.
206
207 timex_after([H, M], [H1, M1]):-
208     timex([H, M]),
209     timex([H1, M1]),
210     H #> H1.
211
212 timex_after([H, M], [H1, M1]):-
213     timex([H, M]),
214     timex([H1, M1]),
215     H = H1,
216     M #> M1.
217
218 before([D, M, Y], T, [D1, M1, Y1], T2):-
219     timex(T),
220     timex(T2),
221     date_before([D, M, Y], [D1, M1, Y1]).
222
223 before([D, M, Y], T, [D, M, Y], T2):-
224     date([D, M, Y]),
225     timex_before(T, T2).
226
227 after([D, M, Y], T, [D1, M1, Y1], T1):-
228     timex(T),
229     timex(T1),
230     date_after([D, M, Y], [D1, M1, Y1]).
231
232 after([D, M, Y], T, [D, M, Y], T1):-
233     date([D, M, Y]),
234     timex_after(T, T1).
235
236 /* Used for adding/subtracting time */
237 divmod(D,S,Q,R) :-
238     D<S,
239     Q is 0,
240     R is S-D.
241
242 divmod(D,S,Q,R) :-
243     D>=S,
244     divmod(D-S,S,Q1,R),
245      Q is Q1+1.
246
247 /*Time T in minutes */
248 addTime([H, M], T, [H1, M1]):-
249     M + T < 60,
250     H1 is H,
251     M1 is M + T.
252
253 addTime([H, M], T, [H1, M1]):-
254     M + T >= 60,
255     DIFF is 60-M,
256     REM is T - DIFF,
257     divmod(REM,60,Q,R),
258      H1 is H+Q+1,
259      M1 is R.
260
261 subtractTime([H, M], T, [H1, M1]):-
262     M - T >= 0,
263     H1 is H,
264     M1 is M - T.
```

```
265
266 subtractTime([H, M], T, [H1, M1]):-
267     M - T < 0,
268     DIFF is T - M,
269     divmod(DIFF, 60, Q, R),
270     H1 is H-1-Q,
271     M1 is 60-R.
272
273 /*
274  *    Language bias
275  */
276
277 weekday([D, M, Y]):-
278     date([D, M, Y]),
279     dow(D,M,Y,X),
280     X in 1..5.
281
282 weekend([D, M, Y]):-
283     date([D, M, Y]),
284     dow(D,M, Y, X),
285     X = 0.
286
287 weekend([D, M, Y]):-
288     date([D, M, Y]),
289     dow(D,M, Y, X),
290     X = 6.
291
292 morning([H, M]):-
293     timex([H, M]),
294     H #< 12,
295     H #> 6.
296
297 afternoon([H, M]):-
298     timex([H, M]),
299     H #> 11,
300     H #< 19.
301
302 evening([H, M]):-
303     timex([H, M]),
304     H #> 18,
305     H #< 23.
306
307 timex_before_h([H, M], H1):-
308     timex([H, M]),
309     H #< H1,
310     member(H1, [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,0]).
311
312 timex_after_h([H, M], H1):-
313     timex([H, M]),
314     H #> H1,
315     member(H1, [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,0]).
316
317 phone_on(D, T) :-
318     after(D, [H, M], D1, T1),
319     before(D, T, D2, T2),
320     onspan(D1, T1, D2, T2).
321
322 phone_on(D, T):-
323     user_is_active(D, T).
324
325 user_is_active(D, T):-
326     after(D, T, D1, T1),
327     before(D, T, D2, T2),
328     activityspan(D1, T1, D2, T2).
329
330 phone_charging(D, T):-
331     after(D, T, D1, T1),
332     before(D, T, D2, T2),
333     chargespan(D1, T1, D2, T2).
334
335
336 at(D, T, CellID):-
337     after(D, T, D1, T1),
338     before(D, T, D2, T2),
```

```
339     cellspan(D1, T1, D2, T2, CellID).
340
341 not_at(D, T, CellID):-
342     after(D, T, D1, T1),
343     before(D, T, D2, T2),
344     cellspan(D1, T1, D2, T2, CellID1),
345     CellID =/= CellID1.
346
347 nearDevice(D, T, DevID):-
348     after(D, T, D1, T1),
349     before(D, T, D2, T2),
350     devicespan(D1, T1, D2, T2, DevID).
351
352 not_nearDevice(D, T, DevID):-
353     after(D, T, D1, T1),
354     before(D, T, D2, T2),
355     devicespan(D1, T1, D2, T2, DevID1),
356     DevID =/= DevID1.
357
358 user_is_using_app(D, T, A):-
359     appspan(D1, T1, D2, T2, A),
360     after(D, T, D1, T1),
361     before(D, T, D2, T2).
362
363 /* X is in the neighborood of Y */
364 neighbourhood(X, Y) :-
365      X #< Y + 100,
366      X #> Y - 100.
367
368 user_been_in(D1, [H1, M1], CellID) :-
369     cellspan(D2, [H2, M2], D3, [H3, M3], CellID),
370     H1 #< H3 + 2,
371     H1 #> H2.
372
373 in_call(D, T):-
374     after(D, T, D1, T1),
375     before(D, T, D2, T2),
376     callspan(D1, T1, D2, T2, _, _, _, _, _, _, _, _, _).
377
378 /*Partial output from here onwards */
379
380 /*
381  * Examples - positive and negative
382  */
383 example(accept([19,3,2011],[12,8],35799199361,7,2,47,255,0,1,10.0,0),-1).
384 example(accept([19,3,2011],[12,11],-1,7,2,46,255,0,1,10.0,0),1).
385 example(accept([19,3,2011],[13,37],07952231440,7,2,23,255,0,1,2600.0  ,0),1).
386 example(accept([19,3,2011],[14,6],07942467850,7,2,22,255,0,0,640.0,1),-1).
387
388 /*
389  *    Static background knowledge
390  */
391
392 screen_status(0).
393 screen_status(1).
394
395 battery_charging(0).
396 battery_charging(1).
397
398 volume(0).
399 volume(1).
400 volume(2).
401 volume(3).
402 volume(4).
403 volume(5).
404 volume(6).
405 volume(7).
406
407 vibrator(0).
408 vibrator(1).
409 vibrator(2).
410 vibrator(3).
411
412 headset(0).
```

```
413 headset(1).
414
415 battery_level(99).
416 battery_level(100).
417 ...
418
419 screen_brightness(1).
420 screen_brightness(3).
421 ...
422
423 /*
424 *  Dynamic background knowledge
425 */
426
427 ...
428 activityspan([3,4,2011],[11,53], [3,4,2011], [11,56]).
429 activityspan([5,4,2011],[11,58], [5,4,2011], [12,0]).
430 ...
431
432 ...
433 appspan([18,3,2011],[0,50],[18,3,2011],[1,8],'skype').
434 appspan([18,3,2011],[0,51],[18,3,2011],[0,52],'googlesearch').
435 ...
436
437 ...
438 callspan([18,3,2011],[11,49],[18,3,2011],[11,53],07942467850,7,2,-1,30,0,1,160.0,0).
439 callspan([19,3,2011],[15,47],[19,3,2011],[15,47],35799199361,5,2,-1,30,0,1,320.0,1).
440 callspan([20,3,2011],[16,19],[20,3,2011],[16,25],07952231440,5,2,-1,30,0,1,320.0,1).
441 ...
442
443 ...
444 cellspan([19,5,2011],[15,28],[19,5,2011],[15,30],1123.9497971).
445 cellspan([19,5,2011],[15,35],[19,5,2011],[15,41],651.3165).
446 cellspan([20,5,2011],[10,09],[20,5,2011],[11,38],1078.9688643).
447 ...
448
449 ...
450 chargespan([26,3,2011],[16,51],[27,3,2011], [3,1]).
451 chargespan([27,3,2011],[12,16],[27,3, 2011], [19,42]).
452 ...
453
454 ...
455 devicespan([18,3,2011],[10,29],[18,3,2011],[10, 52],'0021D17B77D7').
456 devicespan([18,3,2011],[10,51],[18,3,2011],[10, 54],'0023F172C4BC').
457 ...
458
459 ...
460 onspan([4,4,2011],[9,23],[4,4,2011],[18,1]).
461 onspan([31,3,2011],[10,11],[1,4,2011],[7,43]).
462 ...
463
464 ...
465 app('googlesearch').
466 app('skype').
467 ...
468
469 ...
470 cell(1123.9497971).
471 cell(651.3165).
472 cell(1078.9688643).
473 ...
474
475 ...
476 contact(07952231440).
477 contact(35799199361).
478 contact(07942467850).
479 ...
480
481 ...
482 device('0021D17B77D7').
483 device('0023F172C4BC').
484 ...
485
486 ...
```

```
487 light_level(225.0).
488 light_level(2600.0).
489 ...
```

Listing A.1: A partial learning file

B

Language bias settings

Language Bias A

```
1  /* Bias A: Date & Time, Location, Battery/Brightness/Light */
2
3  /* Head Declaration */
4  modeh(accept(+date, +timex, +contact, +volume, +vibrator, +battery_level,
5  +screen_brightness, +headset, +screen_status, +light_level, +battery_charging))
6
7  /* Body Declarations */
8
9  /*Date & Time */
10 modeb(weekday(+date)).
11 modeb(\+weekday(+date)).
12 modeb(weekend(+date)).
13 modeb(\+weekend(+date)).
14 modeb(evening(+timex)).
15 modeb(\+evening(+timex)).
16 modeb(morning(+timex)).
17 modeb(\+morning(+timex)).
18 modeb(afternoon(+timex)).
19 modeb(\+afternoon(+timex)).
20 modeb(time_before_h(+timex, #hour), [no_ground_constants]).
21 modeb(time_after_h(+timex, #hour), [no_ground_constants]).
22
23 /* Location */
24 modeb(at(+date, +timex, #cell)).
25 modeb(not_at(+date, +timex, #cell)).
26 modeb(nearDevice(+date, +timex, #device)).
27 modeb(not_nearDevice(+date, +timex, #device)).
28 modeb(neighbourhood(+cell, #cell)).
29 modeb(user_been_in(+date, +timex, +cell)).
30
31 /* Battery and Brightness and Light */
32 modeb(=(+battery_level, #battery_level), [no_ground_constants]).
```

```
33 modeb(\+(=(+battery_level, #battery_level)), [no_ground_constants]).
34 modeb(=(+screen_brightness, #screen_brightness), [no_ground_constants]).
35 modeb(\+(=(+screen_brightness, #screen_brightness)), [no_ground_constants]).
36 modeb(=(+light_level, #light_level), [no_ground_constants]).
37 modeb(\+(=(+light_level, #light_level)), [no_ground_constants]).
38 modeb(=(+battery_charging, #battery_charging), [no_ground_constants]).
39 modeb(\+(=(+battery_charging, #battery_charging)), [no_ground_constants]).
40 modeb(phone_charging(+date, +timex)).
41 modeb(\+phone_charging(+date, +timex)).
```

Listing B.1: Language bias A

Language Bias B

```
1  /* Bias B: Contact, User Activity, Volume/Vibrator */
2
3  /* Head Declaration */
4  modeh(accept(+date, +timex, +contact, +volume, +vibrator, +battery_level,
5  +screen_brightness, +headset, +screen_status, +light_level, +battery_charging))
6
7  /* Body Declarations */
8
9  /* Contact */
10 modeb(=(+contact, #contact), [no_ground_constants]).
11 modeb(\+(=(+contact, #contact)), [no_ground_constants]).
12
13 /* Volume/Vibrator */
14 modeb(=(+volume, #volume), [no_ground_constants]).
15 modeb(\+(=(+volume, #volume)), [no_ground_constants]).
16 modeb(=(+vibrator, #vibrator), [no_ground_constants]).
17 modeb(\+(=(+vibrator, #vibrator)), [no_ground_constants]).
18
19 /*  User Activity */
20 modeb(in_call(+date, +timex)).
21 modeb(\+in_call(+date, +timex)).
22 modeb(user_is_active(+date, +timex)).
23 modeb(\+user_is_active(+date, +timex)).
24 modeb(phone_charging(+date, +timex)).
25 modeb(\+phone_charging(+date, +timex)).
26 modeb(phone_on(+date, +timex)).
27 modeb(\+phone_on(+date, +timex)).
28 modeb(user_is_using_app(+date, +timex, #app)).
29 modeb(\+user_is_using_app(+date, +timex, #app)).
30 modeb(=(+screen_status, #screen_status), [no_ground_constants]).
31 modeb(\+(=(+screen_status, #screen_status)), [no_ground_constants]).
```

Listing B.2: Language bias B

Language Bias C

```
1  /* Bias C: Date/Time, Contact, Location, */
2
3  /* Head Declaration */
4  modeh(accept(+date, +timex, +contact, +volume, +vibrator, +battery_level,
5  +screen_brightness, +headset, +screen_status, +light_level, +battery_charging))
6
7  /* Body Declarations */
8
9  /*Date & Time */
10 modeb(weekday(+date)).
11 modeb(\+weekday(+date)).
12 modeb(weekend(+date)).
13 modeb(\+weekend(+date)).
14 modeb(evening(+timex)).
15 modeb(\+evening(+timex)).
16 modeb(morning(+timex)).
17 modeb(\+morning(+timex)).
18 modeb(afternoon(+timex)).
19 modeb(\+afternoon(+timex)).
20 modeb(timex_before_h(+timex, #hour), [no_ground_constants]).
21 modeb(timex_after_h(+timex, #hour), [no_ground_constants]).
22
23 /* Contact */
24 modeb(=(+contact, #contact), [no_ground_constants]).
25 modeb(\+(=(+contact, #contact)), [no_ground_constants]).
26
27 /* Location */
28 modeb(at(+date, +timex, #cell)).
29 modeb(not_at(+date, +timex, #cell)).
30 modeb(nearDevice(+date, +timex, #device)).
31 modeb(not_nearDevice(+date, +timex, #device)).
32 modeb(neighbourhood(+cell, #cell)).
33 modeb(user_been_in(+date, +timex, +cell)).
```

Listing B.3: Language bias C

Language Bias D

```
1  /* Bias D: Location, User activity, Volume/Vibrator */
2
3  /* Head Declaration */
4  modeh(accept(+date, +timex, +contact, +volume, +vibrator, +battery_level,
5  +screen_brightness, +headset, +screen_status, +light_level, +battery_charging))
6
7  /* Body Declarations */
8
9  /* Location */
10 modeb(at(+date, +timex, #cell)).
11 modeb(not_at(+date, +timex, #cell)).
12 modeb(nearDevice(+date, +timex, #device)).
13 modeb(not_nearDevice(+date, +timex, #device)).
14 modeb(neighbourhood(+cell, #cell)).
15 modeb(user_been_in(+date, +timex, +cell)).
16
17 /*  User Activity */
18 modeb(in_call(+date, +timex)).
19 modeb(\+in_call(+date, +timex)).
20 modeb(user_is_active(+date, +timex)).
21 modeb(\+user_is_active(+date, +timex)).
22 modeb(phone_charging(+date, +timex)).
23 modeb(\+phone_charging(+date, +timex)).
24 modeb(phone_on(+date, +timex)).
25 modeb(\+phone_on(+date, +timex)).
26 modeb(user_is_using_app(+date, +timex, #app)).
27 modeb(\+user_is_using_app(+date, +timex, #app)).
28 modeb(=(+screen_status, #screen_status), [no_ground_constants]).
29 modeb(\+(=(+screen_status, #screen_status)), [no_ground_constants]).
30
31 /* Volume/Vibrator */
32 modeb(=(+volume, #volume), [no_ground_constants]).
33 modeb(\+(=(+volume, #volume)), [no_ground_constants]).
34 modeb(=(+vibrator, #vibrator), [no_ground_constants]).
35 modeb(\+(=(+vibrator, #vibrator)), [no_ground_constants]).
```

Listing B.4: Language bias D

τά ἀγαθά κόπεις κτῶνται