A real-time semantics-aware activity recognition system

Master Thesis

Efthymiou Vasileios
Computer Science Department
University of Crete

Supervisor:
Grigoris Antoniou

January 2012
Acknowledgments

I am heartily thankful to my supervisor, Grigoris Antoniou, whose encouragement, guidance and support from the initial to the final level enabled me to finish this work. Even as an undergraduate student, I had the chance and honor to be close to a research environment - in the Institute of Computer Science, of the Foundation for Research and Technology (ICS-FORTH) at the Information Systems Laboratory (ISL) - an opportunity granted by prof. Antoniou. I wish him good luck in his new beginning and I also wish that our cooperation will continue.

This work would never exist without Antonis Bikakis and Theodore Patkos, who came up with the idea to use Case-Based Reasoning and/or Answer-Set programming in order to achieve real-time activity recognition. Even if we met only a few times on the first semester of this work, their help was more than essential. Furthermore, I owe my deepest gratitude to Antonis, who helped me take important decisions on my future studies and supported me with any possible way, whenever I needed him.

Real-time activity recognition was designed to be a part of a wider project, aiming to assist users in an Ambient Intelligent environment. For the implementation of two such systems – one for the assistance of students and teachers in a smart classroom and one for the assistance of the elderly in a “smart” home – I was lucky to cooperate with Maria Koutrakí and Dimitra Zografistou. The SWRL rules that are referred in this paper as well as the core ontology presented in Chapter 6 are products of their work. I would like to thank them for their significant support and understanding, even in times of great difficulties and disappointment.

Special thanks to Ioannis Chrysakis and Nikos Ftylitakis, who played an important role when it came to implementing this system and especially creating a demo for this system. Ioannis helped in the transition from theory to practice, offering his know-how in Ambient Intelligence and particularly in the AmI Sandbox. Nikos is also a person who simply made things work, as well as a “guide” in the AmI Sandbox.

Lastly, I offer my regards and blessings to my friends and my family and especially to my companion Efi, who supported me in any respect during the completion of the project.

Heraklion, January 28, 2012
Abstract

This paper introduces a system that exploits the semantics of ontologies, to improve the accuracy of machine-learning methods. This system is intended to perform real-time activity recognition that could assist users in an Ambient Intelligent (AmI) environment. Ontologies can offer quite rich semantics that, to our knowledge, have not been utilized yet in a system that performs real-time activity recognition, mainly due to the time-cost of loading/querying ontologies. For the utilization of the information that ontologies offer (such as hierarchies), we introduce novel methods to record them in machine learning datasets. We also present ways that make real-time activity recognition viable, by keeping execution-time and memory requirements low. Post-processing of the suggested activities is also involved to make sure that the returned results will contain as few errors as possible. Results on the experiments that we held, in order to present the performance of our system, are also provided.
Περίληψη

Σε αυτή την εργασία παρουσιάζεται ένα σύστημα που εκμεταλλεύεται τη σημασιολογία των οντολογιών, προκειμένου να βελτιώσει την ακρίβεια μεθόδου μηχανικής μάθησης. Σκοπός του συστήματος αυτού είναι να πραγματοποιεί αναγνώριση δραστηριότητας σε πραγματικό χρόνο, βοηθώντας τους χρήστες ενός περιβάλλοντος διάχυτης νοημοσύνης. Οι οντολογίες μπορούν να προσφέρουν αρκετά πλούσια σημασιολογία, που εξ όσων γνωρίζουμε, δεν έχει χρησιμοποιηθεί ακόμα σε κάποιο σύστημα που αναγνωρίζει σε πραγματικό χρόνο δραστηριότητες, κυρίως λόγω του μεγάλου χρονικού κόστους φόρτωσης/επερώτησης μίας οντολογίας. Για τη χρησιμοποίηση των πληροφοριών που προσφέρουν οι οντολογίες (όπως οι ιεραρχίες), εισάγουμε νέες μεθόδους καταγραφής τους σε «αποθήκες» δεδομένων (dataset) μηχανικής μάθησης. Παρουσιάζουμε επίσης τρόπους που κάνουν την αναγνώριση δραστηριότητας σε πραγματικό χρόνο εφικτή, κρατώντας το χρόνο εκτέλεσης και τις απαιτήσεις σε μικρή χαμηλά. Χρησιμοποιείται και ένα τελικό στάδιο επεξεργασίας των προτεινόμενων δραστηριοτήτων, εξασφαλίζοντας ότι οι λανθασμένες προβλέψεις θα είναι όσο το δυνατόν λιγότερες. Τα αποτελέσματα των μετρήσεων που έγιναν με σκοπό την εκτίμηση της ακρίβειας και του απαιτούμενου χρόνου του συστήματός μας δίνονται αναλυτικά.
# Contents

Acknowledgments ........................................................................................................ iii  
Abstract ...................................................................................................................... iv  
Περίληψη .................................................................................................................. v  
1 Introduction ............................................................................................................. 1  
   1.1 Problem description / Goals ........................................................................... 2  
   1.2 Outline .......................................................................................................... 3  
2 Background ........................................................................................................... 5  
   2.1 Terms/Concepts definition ............................................................................. 5  
   2.2 Related Work ............................................................................................... 11  
3 Solution Description ............................................................................................. 17  
   3.1 Main idea ....................................................................................................... 17  
   3.2 User input requirements .............................................................................. 18  
   3.3 Exploiting the semantics of an ontology .................................................... 20  
   3.4 Semantics covered ....................................................................................... 22  
   3.5 Reducing the dataset .................................................................................... 24  
   3.6 Caching ......................................................................................................... 25  
   3.7 Historic variant ............................................................................................. 26  
   3.8 Validation of predictions ............................................................................. 26  
   3.9 Missing prerequisites ................................................................................... 27  
   3.10 Scaling-up ................................................................................................... 28  
4 System Architecture .............................................................................................. 31  
   4.1 AmI Sandbox ............................................................................................... 31  
   4.2 Context .......................................................................................................... 32  
   4.3 Activity Recognition ..................................................................................... 32  
   4.4 User assistance ............................................................................................. 33  
   4.5 Thread architecture ...................................................................................... 34  
5 Technologies Used ................................................................................................. 37  
   5.1 Protégé .......................................................................................................... 37  
   5.2 LibSVM ......................................................................................................... 38  
   5.3 Weka ............................................................................................................. 38  
6 Applications .......................................................................................................... 39  
   6.1 Smart Classroom ......................................................................................... 40  
   6.2 Ambient Assisted Living .............................................................................. 42
7 Evaluation .................................................................................................................. 45
  7.1 Accuracy ................................................................................................................. 49
  7.2 Time ....................................................................................................................... 53
  7.3 Demo ....................................................................................................................... 53
8 Conclusions ............................................................................................................... 57
  8.1 Limitations ............................................................................................................. 57
  8.2 Future Work ........................................................................................................... 58
References .................................................................................................................... 59
List of Figures

Fig. 1: AssistedLiving.owl, a sample ontology. Squares represent classes and edges represent the subclassOf property. ................................................................. 3

Fig. 2: k-Nearest Neighbors for k=3 and k=5. When k = 3, the query point is classified as a red triangle, since 2/3 nearest neighbors are red triangles. However, when k = 5, the query point is classified as a blue square, since 3/5 nearest neighbors are blue squares. ................................................................. 8

Fig. 3: Data cannot be divided into blue and red in 2 dimensions, but they can be divided into blue and red in 3 dimensions. We only calculate their inner product in that space (using the kernel trick). .................................................................................. 9

Fig. 4: The selected hyperplane that maximizes the margin between red and green dots is the one in D. .................................................................................. 9

Fig. 5: The Markov blanket of node A in a Bayesian network. It is defined as the set of A’s parents, its children and its children’s other parents. ........................................ 10

Fig. 6: Case-Based Reasoning Cycle. ........................................................................ 11

Fig. 7: Concept based similarity functions in jCOLIBRI2 ........................................ 12

Fig. 8: Example of application of the similarity functions ........................................ 13

Fig. 9: A sample case instance, based on the sample ontology of Figure 1. Cases are stored as instances of a class (here named “Case”). Attributes are stored as the values (range) of properties that have the case class as domain (here “has-Weather”, “has-Time”, “has-Stance”, “has-Person”, “has-Season”, “has-Location”). The recognition target (here activity), is also stored in the same way as attributes (using the “has-Activity” property). ................................................................. 19

Fig. 10: The object properties of the class “Case”. Edges represent properties. Their source (the class “Case”) represents their domain and their destination (“Person”, “Season”, “Stance”, “Weather”, “Time”, “Location”, “Activity”) represents their range. ........................................................................ 19

Fig. 11: A sample GUI that demonstrates the required input ..................................... 21

Fig. 12: How Boolean attributes are valued for our final solution............................ 22

Fig. 13: Threads’ scheduling. The “Sensor Handler” thread keeps running constantly, storing the sensor data in the ontology. The “SWRL” thread runs once in a while, performing a first level of reasoning and storing its results as “Simple Events” in the ontology. Then the “Activity recognition” system is called (here for every 5 runs of the “SWRL” thread), which returns the current activity to the “User Assistance” thread. ........................................................................ 35
Fig. 14: The core ontology, used for both the “Smart-classroom” and the “Ambient Assisted Living” systems. “Simple Events” are recognized through SWRL rules and “Activities” are recognized through our activity recognition system. .................................. 39

Fig. 15: Smart Classroom Architecture. Sensor data from the classroom are stored in the ontology. Then SWRL rules perform a first level of reasoning, storing their results in the ontology. The activity recognition system uses these results and returns the current activity to the Classroom Assistant. The latter may alter the state of the classroom. ................................................................................................................................ 40

Fig. 16: Performance results. “LOO” stands for Leave-One-Out Cross Validation and “10fold” stands for 10-fold Cross Validation. “Assisted” means Assisted Living, “Type” means that the target of the recognition is the holiday type and “Category” means that the target of the recognition is the accommodation category. For these tests no data are missing. Our final approach (BNFull) always performs better than JCOLIBRI 2.0 in these tests. ........................................................................................................... 50

Fig. 17: US citizens’ income-prediction performance. BNFull is for the use of our ontology, based on this dataset. The difference is slight, but important when it comes to almost 50000 cases. ................................................................................................................. 51

Fig. 18: A screenshot from the demo. On the lower right corner is the GUI available to the user for the needs of this demo. The current activity – along with the likelihood that it is actually happening - is the last shown in the text area of the GUI. .......... 55
List of Tables

Table 1: Performance results
Table 2: UScitizens’ income results
Table 3: Detailed accuracy by class on the smart classroom domain
Table 4: Confusion matrix of smart classroom domain
Chapter 1

Introduction

 Ambient Intelligence (AmI) could be described as the branch of Artificial Intelligence in which the context of a so-called “smart” space can be modelled, processed and even altered in ways that satisfy certain needs. These needs usually include health assistance, assistance of the elderly, classroom assistance, conviviality etc. As stated in [1] “context is defined as any information that can be used to characterize the situation of an entity” and systems that are able to adapt any possible context changes are called context-aware. That makes every AmI system unique, since it is almost impossible to reproduce the same context, needs and entities in different environments. It is obvious from this description that modelling the context is probably the key factor in designing a successful AmI system.

 The most indicated approach to context-modelling is the use of ontologies. “An ontology is an explicit and formal specification of a conceptualization” (Gruber). Typically, an ontology consists of a finite list of terms and the relationships between these terms [2]. Hierarchy between these terms is one of the most important relationships that ontologies can model. Ontologies play a pivotal role not only for the semantic web, but also in pervasive computing and next generation mobile communication systems. They provide formalizations to project real-life entities onto machine-understandable data constructs [3]. This enables computers to perform the thing that they are best at: processing. In our system, machine learning (ML) algorithms use these data constructs to conceive a higher level representation of the context.

 ML can deal with uncertainty in data containing incomplete or contradictory information. This is the most common type of data that can be met at AmI systems and especially in activity recognition domains. Rule-based reasoning cannot deal with missing information and it would take a very high level of reasoning to deal with contradictory information as well. Another advantage of ML is the short reasoning time required, in comparison to rule-based reasoning, especially when it comes to large-scale datasets.

 The essence of time becomes clear, when dealing with context-aware systems, since the context keeps changing constantly and in most cases it is obligatory to “follow” these changes as they occur. Since the notion of time plays an important role in such systems, it should also be modelled and used to distinguish similar cases that occur in different periods.

 Case Based Reasoning (CBR) seems to be an efficient approach in reasoning when it comes to AmI systems. CBR is typically described as a cyclical process comprising the four REs [4]: (1) REtrieve the most similar case or cases, (2) REuse the information and knowledge in that case to attempt to solve the problem, (3) Revise the proposed solution if necessary, and (4) REtain the parts of this experience likely to be useful for future problem solving. The most distinguished advantage of CBR is that it can learn continuously by just adding new cases to the case base (the dataset in which all cases are stored). Even if our approach is not a typical CBR one (it does not use kNN), we use this notion to address the problems that arise with CBR systems, which are similar to our activity recognition system.
1.1 Problem description / Goals

The problem that we are going to solve is recognizing the (single) activity of a (single) person in real-time, by exploiting the useful information that ontologies offer. The set of possible activities is finite and predetermined, stored in the given ontology. This means that no new activities can be identified during the system’s execution. The initial dataset contains solved cases, annotated by a human. Our aim is just to choose which one of the given activities is most likely to happen at the current time. We plan to achieve this by “flattening” the given ontology, in a way that its semantics can be expressed in a ML dataset. The motivating scenario for this work was the design of a system that could be used as a part of two larger, user-assisting systems, to recognize the activities in a smart-classroom and a smart-home (for ambient assisted living).

The wide use of ontologies in modeling the context has led to the demand of reasoning based on them. Usually this kind of reasoning is the definition of rules that trigger when certain events occur. Rules, however, are domain-specific, which means that they cannot be easily generalized or even used for two similar systems that belong to the same domain. When dealing with large scale datasets (which is usually the case), this kind of reasoning becomes too time consuming to be used in real-time systems. Incomplete data is also, as mentioned above, a problem that rule-based approaches cannot solve. All these deficiencies (time cost, inability to generalize and handling missing data) are prohibitive for a real-time context-aware activity-recognition system.

Since we have rejected the use of rules for reasoning, we should find another way to somehow depict the rich expressiveness that ontologies can offer. To accomplish this we should represent as many properties that ontologies have as possible.

Most CBR tools use the k-Nearest Neighbors (kNN) algorithm. kNN is chosen due to the fact that it is easy to implement, easy to understand, easy to cross-validate, it supports lazy learning and it seems to perform pretty well in some domains. However, there are some serious drawbacks in this algorithm. kNN does not seem to perform as well as other methods, like neural networks, Support Vector Machines (SVMs) or Bayesian Networks (BNs). Apart from that, one has to choose a distance function – so as to define the nearest neighbors - and weights for the attributes. Neither of these choices is trivial, so more testing has to take place, for choosing these parameters. Even when some great distance functions are given, like the ones used in similar systems, presented later, it is not trivial even to choose one of them for each feature.

Ontologies are based on classes and properties between them. The terms denote important concepts (classes of objects) of the domain. For example, in a smart home context, persons, family members, residents, location, time and activity are some important concepts. The relationships typically include hierarchies of classes. For example, all family members are persons. Figure 1 shows a sample hierarchy for a smart home domain (used for assisted living of the elderly). The attributes chosen to depict an ontology as a dataset, must take advantage of the properties of ontologies, specifically the ability of an ontology to hold relationship information between terms.
Fig. 1: AssistedLiving.owl, a sample ontology. Squares represent classes and edges represent the subclassOf property.

1.2 Outline

Below is a small summary of the following chapters.

In Chapter 2 we present a brief explanation of notions and terms that will be used in the rest of this paper, as well as the work related to our system; mainly CBR systems that either deal with activity recognition, or with reasoning based on ontologies.

Chapter 3 is the core of our work, since it analyzes the way that we process our input and produce the desirable results. This chapter also describes the limits of this system, as well as the required input and the main ideas that led to the implementation of a new system.

Chapter 4 explains how our system is set up and what its place is in a wider ambient intelligent system. It also presents the way that processes in this wider system communicate, as well as some of the other parts of this system (user assistance, sensor handling etc).

Chapter 5 is about the technologies that we have used to design and deploy our system. These include the free software that we used to design our ontologies and run some machine learning algorithms.

Chapter 6 presents some possible applications of this system, as well as two applications that are currently developed in parallel to this system. For each of these two applications we present a motivating usage scenario and how activity recognition helps the final users.

In Chapter 7 we present the results of the experimental evaluation. We explain in details how these results occurred and how they can be interpreted. When possible,
we try to compare these results to the results of other available tools, or the results that we could acquire without the use of any reasoning.

Chapter 8 ends this paper with useful conclusions and suggestions about possible future work. It is a short summary of the most important features of this work and it also presents some more possibilities of this system that could be exploited in the future.
Chapter 2

Background

2.1 Terms/Concepts definition

To describe what ontologies are and what they can offer we will adopt some of the definitions and descriptions that exist in [2]. First of all, to define an ontology, we will repeat T.R. Gruber’s definition, later refined by R. Studer: “An ontology is an explicit and formal specification of a conceptualization”. In general, an ontology describes formally a domain of discourse.

Typically, an ontology consists of a finite list of terms and the relationships between these terms. The terms denote important concepts (classes of objects) of the domain. For example, in a university setting, staff members, students, courses, lecture theatres, and disciplines are some important concepts. The relationships typically include hierarchies of classes. A hierarchy specifies a class C to be a subclass of another class C’ if every object in C is also included in C’. For example, all faculty are staff members. Apart from subclass relationships, ontologies may include information such as
- properties (X teaches Y),
- value restrictions (only faculty members may teach courses),
- disjointness statements (faculty and general staff are disjoint),
- specifications of logical relationships between objects (every department must include at least ten faculty members).

In the context of the Web, ontologies provide a shared understanding of a domain. Such a shared understanding is necessary to overcome differences in terminology. One application’s zip code may be the same as another application’s area code. Ontologies are useful for the organization and navigation of Web sites. Many Web sites today expose on the left-hand side of the page the top levels of a concept hierarchy of terms. The user may click on one of them to expand the subcategories.

Also, ontologies are useful for improving the accuracy of Web searches. The search engines can look for pages that refer to a precise concept in an ontology instead of collecting all pages in which certain, generally ambiguous, keywords occur. In this way, differences in terminology between Web pages and the queries can be overcome.

In Artificial Intelligence (AI) there is a long tradition of developing and using ontology languages. It is a foundation Semantic Web research can build upon. At present, the most important ontology languages for the Web are RDF, RDFS and OWL, all of which are based on XML. To define these languages, we will adopt their definition and examples from W3C and [2].

Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML (ISO 8879). Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere. HTML documents do not contain structural information, that is, information about pieces of the document
and their relationships. In contrast, the XML document is far more easily accessible to machines because every piece of information is described. Moreover, some forms of their relations are also defined through the nesting structure. For example, the <author> tags appear within the <book> tags, so they describe properties of the particular book. A machine processing the XML document would be able to deduce that the author element refers to the enclosing book element, rather than having to infer this fact from proximity considerations, as in HTML. An additional advantage is that XML allows the definition of constraints on values (for example, that a year must be a number of four digits, that the number must be less than 3,000). XML allows the representation of information that is also machine-accessible.

XML Schema offers a significantly richer language for defining the structure of XML documents. There are several different schema languages in widespread use, but the main ones are Document Type Definitions (DTDs), Relax-NG, Schematron and W3C XSD (XML Schema Definitions). One of its characteristics is that its syntax is based on XML itself. This design decision provides a significant improvement in readability, but more important, it also allows significant reuse of technology. It is no longer necessary to write separate parsers, editors, pretty printers, and so on, to obtain a separate syntax, as was required for DTDs; any XML will do. An even more important improvement is the possibility of reusing and refining schemas. XML Schema allows one to define new types by extending or restricting already existing ones. In combination with an XML-based syntax, this feature allows one to build schemas from other schemas, thus reducing the workload. Finally, XML Schema provides a sophisticated set of data types that can be used in XML documents (DTDs were limited to strings only).

XML does not provide any means of talking about the semantics (meaning) of data. For example, there is no intended meaning associated with the nesting of tags; it is up to each application to interpret the nesting. Although often called a “language”, RDF (Resource Description Framework) is essentially a data model. Its basic building block is an object-attribute-value triple, called a statement. Of course, an abstract data model needs a concrete syntax in order to be represented and transmitted, and RDF has been given syntax in XML. As a result, it inherits the benefits associated with XML. However, it is important to understand that other syntactic representations of RDF, not based on XML, are also possible; XML-based syntax is not a necessary component of the RDF model. RDF extends the linking structure of the Web to use URIs to name the relationship between things as well as the two ends of the link (this is usually referred to as a “triple”). Using this simple model, it allows structured and semi-structured data to be mixed, exposed, and shared across different applications. This linking structure forms a directed, labelled graph, where the edges represent the named link between two resources, represented by the graph nodes. This graph view is the easiest possible mental model for RDF and is often used in easy-to-understand visual explanations.

RDF is domain-independent in that no assumptions about a particular domain of use are made. It is up to users to define their own terminology in a schema language called RDF Schema (RDFS). The name RDF Schema is now widely regarded as an unfortunate choice. It suggests that RDF Schema has a similar relation to RDF as XML Schema has to XML, but in fact this is not the case. XML Schema constrains the structure of XML documents, whereas RDF Schema defines the vocabulary used in RDF data models. In RDFS we can define the vocabulary, specify which properties apply to which kinds of objects and what values they can take, and describe the
relationships between objects. RDFS makes semantic information machine-accessible, in accordance with the Semantic Web vision.

The expressivity of RDF and RDF Schema that we described is deliberately very limited. RDF is (roughly) limited to binary ground predicates, and RDF Schema is (roughly) limited to a subclass hierarchy and a property hierarchy, with domain and range definitions of these properties. However, the Web Ontology Working Group of W3C identified a number of characteristic use cases for the Semantic Web that would require much more expressiveness than RDF and RDF Schema offer. The Web Ontology Language (OWL) is a Semantic Web language designed to represent rich and complex knowledge about things, groups of things, and relations between things. OWL is a computational logic-based language such that knowledge expressed in OWL can be reasoned with by computer programs either to verify the consistency of that knowledge or to make implicit knowledge explicit. OWL documents, known as ontologies, can be published in the World Wide Web and may refer to or be referred from other OWL ontologies. Ideally, OWL would be an extension of RDF Schema, in the sense that OWL would use the RDF meaning of classes and properties (rdfs:Class, rdfs:subClassOf, etc.) and would add language primitives to support the richer expressiveness required. Unfortunately, simply extending RDF Schema would work against obtaining expressive power and efficient reasoning. RDF Schema has some very powerful modeling primitives. Constructions such as rdfs:Class (the class of all classes) and rdf:Property (the class of all properties) are very expressive and would lead to uncontrollable computational properties if the logic were extended with such expressive primitives.

We decided to use OWL for our system evaluation, since it appears to be the most popular ontology language, even if its richer expressiveness is not always exploited. In our bibliography study, we found that many times RDF could be used as well to model the same information, but OWL is preferred for compatibility reasons.

Semantic Web Rules Language (SWRL) is a proposed Semantic Web language combining OWL DL (and thus OWL Lite) with function-free Horn logic, written in Datalog RuleML. Thus it allows Horn-like rules to be combined with OWL DL ontologies. A rule in SWRL has the form

\[ B_1, \ldots, B_n \rightarrow A_1, \ldots, A_m \]

where the commas denote conjunction on both sides of the arrow and \( A_1, \ldots, A_m \) can be of the form \( C(x), P(x, y), \text{sameAs}(x, y), \text{or differFrom}(x, y) \), where \( C \) is an OWL description, \( P \) is an OWL property, and \( x, y \) are Datalog variables, OWL individuals, or OWL data values.

The term machine learning is used to describe the process of designing and developing algorithms (executed by machines) that are able to evolve (learn) based on empirical data. Usually the algorithm goes through a training phase, in which it “learns” from a given dataset and then through a testing phase, in which we evaluate its behaviour on new data (that were not used for training). It becomes obvious after this definition that CBR is definitely considered as machine learning.

Machine learning can be split into three main categories; supervised, unsupervised and reinforcement. Supervised learning is used when the empirical data provided is labelled and there is a desirable output. Unsupervised learning is used when we have no information about the desired output. Reinforcement learning is based on rewards and penalties (inspired from behaviourist psychology), depending on the action of the algorithm. For the needs of this project we use a supervised approach.
initial, rather small manually-labelled dataset, on which the first learning is based and then there is the continuously growing dataset, based on real-time observations, for which we have no desired output information.

We can also further distinguish machine learning between classification and regression algorithms, depending on the attribute that is going to be “predicted” (we call this the class attribute). Classification is used when the class attribute takes discrete values and regression is used when it takes continuous values. We have implemented an algorithm that solves classification problems.

There are several machine learning algorithms, fitting several needs. The most popular approaches are decision trees, k nearest neighbours (kNNs), artificial neural networks (ANNs), Support Vector Machines (SVMs) and those based on the Bayes’ rule, such as naive Bayes and Bayesian networks (BNs). We decided to use BNs, since they are designed to work well with missing data. SVMs on the other hand were not initially designed to allow missing data, even if some later extensions that allow this have been published [5]. In [6] it is claimed that - for a specific biological domain – BNs outperform SVMs when data are missing and perform almost the same, when all data are available. Even if we have not tested our system with missing data, this is a typical phenomenon when dealing with sensors and we wanted to be prepared.

**K – Nearest Neighbour (kNN)** is a supervised learning algorithm where the result of new instance query is classified based on majority of K-nearest neighbour category. The purpose of this algorithm is to classify a new object based on attributes and training samples. The classifiers do not use any model to fit and they are only based on memory. Given a query point, we find K number of objects (or training points) closest to the query point, as depicted in Figure 2 for K = 3 and K = 5. The classification is using majority vote among the classification of the K objects. Any ties can be broken at random. K Nearest neighbour algorithm used neighbourhood classification as the prediction value of the new query instance. So in Figure 2, if we choose the value 3 for K, then we would classify our query point as a red triangle (among the 3 nearest neighbours – inside the small circle - two of them are red triangles and only one is a blue square). However if we choose the value 5 for K, then we would classify our query point as a blue square (since there are three blue squares and only two red triangles among the 5 nearest neighbours – inside the big circle).

**Support Vector Machines (SVMs)** are arguably the most important recent discovery in machine learning. They can be summarized in three main points:

![Fig. 2: k-Nearest Neighbors for k=3 and k=5. When k = 3, the query point is classified as a red triangle, since 2/3 nearest neighbors are red triangles. However, when k = 5, the query point is classified as a blue square, since 3/5 nearest neighbors are blue squares.](image)
- Map the data to a predetermined very high-dimensional space via a kernel function (Figure 3). ¹
- Find the hyperplane that maximizes the margin between the classes (Figure 4)
- If data are not separable find the hyperplane that maximizes the margin and minimizes the (a weighted average of the) outliers

Samples on the margin are called the support vectors. Even if SVMs map to a very high-dimensional space, they have a very strong bias only in that space and not in the initial space. Therefore they have the ability to generalize well. The original SVM algorithm was invented by Vladimir N. Vapnik and the current standard incarnation (soft margin) was proposed by Vapnik and Corinna Cortes in 1995 [7].

Bayesian networks (BNs), or belief networks, are probabilistic graphical models (DAGs) that represent random variables (as nodes) and their conditional dependencies

¹ We do not actually need to explicitly map the data into the high-dimensional space to solve the optimization problem. We can just find their inner product in this space, by using the kernel function. This “trick” is called the “kernel trick”.

![Diagram of SVM and Bayesian networks](image_url)
(as edges). This means that two unconnected nodes are conditionally independent. As their name reveals, BNs are based on the Bayes’ rule, which states that

\[ P(A|B) = \frac{P(B|A)P(A)}{P(B)}. \]

\( P(A|B) \) is the conditional probability that \( A \) is true, given that \( B \) is true. BNs also use the Markov condition, which states that every variable is independent of its non-descendants given its parents. The **Markov blanket** for a node \( A \) in a Bayesian network is the set of nodes composed of \( A \)'s parents, its children, and its children's other parents, as depicted in Figure 5. The Markov blanket of a node contains all the variables that shield the node from the rest of the network. This means that the Markov blanket of a node is the only knowledge needed to predict the behavior of that node. The term was coined by Pearl in 1988.

![Markov blanket diagram](image)

**Fig. 5:** The Markov blanket of node \( A \) in a Bayesian network. It is defined as the set of \( A \)'s parents, its children and its children's other parents.

If we use these properties, we can easily derive the factorization \( P(V) = \prod_i P(V_i | pa(V_i)) \), where \( V \) is a Bayesian network, \( V_i \) is a node and \( pa(V_i) \) are the parents of \( V_i \).

BNs are sometimes used to represent causality. These kinds of BNs are called Causal Bayesian Networks (CBNs) and use exactly the same logic as BNs with one difference; edges represent causal relationships instead of conditional dependencies. So, instead of using the Markov conditions, CBNs use the causal Markov condition, which states that every variable is independent of its non-effects given its direct causes. Using these semantics, one can predict the impact of external interventions from data obtained prior to intervention. For our system we have not used any causality relationships, but it might be something interesting to investigate in the future, maybe in an intention-recognition system.

**Case-Based Reasoning (CBR)** is a branch of machine learning (it solves new problems based on the solutions of similar past problems) that performs “lazy learning”. This means that no action is required until a new problem appears. When a new problem appears, CBR algorithms compare it to a given dataset (case-base) and suggest the solution that was used to solve the \( (k) \) nearest case(s), typically by using kNN. Although our system is not a typical CBR system, we use this term in order to easily explain how we implement each part of a CBR cycle (Figure 6). CBR is
described as a cyclical process comprising the four REs [4]: (1) REtrieve the most similar case or cases, (2) REuse the information and knowledge in that case to attempt to solve the problem, (3) Revise the proposed solution if necessary, and (4) RETain the parts of this experience likely to be useful for future problem solving. In §3.1 we describe how each part of this process is implemented in our approach.

Propositionalization is defined as a representation change from a relational representation to a propositional one [8]. It involves the construction of structural features from relational background knowledge. Propositionalization is a special case of constructive induction and feature construction. Propositionalizations can be either complete or partial (heuristic). In the former case, no information is lost in the process [9]; in the latter, information is lost and the representation change is incomplete. Our approach could be considered as partial propositionalization, since we do not cover all the semantics from an ontology, but only some specific properties (§3.3, §3.4). It has so far not been carefully investigated which propositionalization methods are best suited for different standard machine learning algorithms [10].

2.2 Related Work

There are several CBR tools today, most of which deal with textual CBR and many activity-recognition systems, most of which do not use ontologies. Here is a list of the most similar CBR tools/ reasoning systems that we could find.

The most popular CBR tool that supports ontologies is JCOLIBRI [11,12,13,14]. It is stated in a recent survey [15] that “The central papers in this cluster describe innovations around the jColibri CBR development environment, which is well suited for knowledge intensive CBR”. jCOLIBRI is a Java framework that helps during the development of CBR systems. As most of the CBR tools do, JCOLIBRI uses the kNN algorithm to classify a new case. The most important feature
of JCOLIBRI, in regards to our work, is that it also tries to grasp some of the semantics of an ontology. To do this – since it uses kNN as mentioned before – some distance functions have to be defined. These distance functions are chosen thoroughly and manage to grasp, to a certain degree, the hierarchy of ontologies. These distance functions, provided with JCOLIBRI, are also presented in Figure 7.

$$f_{\text{deep basic}}(i_1, i_2) = \frac{\max(\text{prof}(\text{LCS}(i_1, i_2)))}{\max_{c_i \in CN} \text{prof}(C_i)}$$

$$f_{\text{deep}}(i_1, i_2) = \frac{\max(\text{prof}(\text{LCS}(i_1, i_2)))}{\max(\text{prof}(i_1), \text{prof}(i_2))}$$

$$\text{cosine}(i_1, i_2) = \frac{\left( \bigcup_{d_i \in t(i_1)} \text{(super}(d_i, CN)) \right) \cap \left( \bigcup_{d_i \in t(i_2)} \text{(super}(d_i, CN)) \right)}{\sqrt{\left( \bigcup_{d_i \in t(i_1)} \text{(super}(d_i, CN)) \right)} \cdot \sqrt{\left( \bigcup_{d_i \in t(i_2)} \text{(super}(d_i, CN)) \right)}}$$

$$\text{detail}(i_1, i_2) = 1 - \frac{2}{1 - \left( \left( \bigcup_{d_i \in t(i_1)} \text{(super}(d_i, CN)) \right) \cap \left( \bigcup_{d_i \in t(i_2)} \text{(super}(d_i, CN)) \right) \right)}$$

Where:
- $CN$ is the set of all the concepts in the current knowledge base
- $\text{super}(c, C)$ is the subset of concepts in $C$ which are superconcepts of $c$
- $\text{LCS}(i_1, i_2)$ is the set of the least common subsumer concepts of the two given individuals
- $\text{prof}(c)$ is the depth of concept $c$
- $t(i)$ is the set of concepts the individual $i$ is instance of

**Fig. 7: Concept based similarity functions in jCOLIBRI2**

The “$f_{\text{deep basic}}$” similarity between two individuals $a$ and $b$ is defined as the depth of the most “deep” common ancestor of $a$ and $b$, divided by the greatest depth in which a concept (class) can be met in the knowledge base (ontology). The “$f_{\text{deep}}$” similarity between $a$ and $b$ uses the same numerator divided by the maximum depth between $a$’s and $b$’s depths. The “cosine” similarity between $a$ and $b$ is closer to 1 as the number of the superclasses of $a$ and $b$ that are not common for $a$ and $b$ gets closer to 0. The “detail” similarity between $a$ and $b$ is closer to 1 (but never 1, even if we compare one node to itself) as the number of common ancestors of $a$ and $b$ increases. There is also an example of how these metrics can be applied in a sample ontology presented in Figure 8, along with the similarities calculated based on each of these functions.
Another promising tool is CREEK\[16\], designed in 1991 as a PhD thesis of Agnar Aamodt. CREEK - recently renamed as AmICREEK - is not publicly available, although some case-evaluation studies have been published \[17\]. As described in \[18\] the cases built in this study were limited to nine features including location, user, role and time. These are high-level pieces of information, which must be inferred from lower granularity sensed data. This information was gathered in their user-study by a participant who followed a doctor for a number of days, noting the criteria and the corresponding situation. Kofod-Petersen[19] also noted that scalability is an issue for CBR in activity recognition due to the possible volume of cases being added to the case base.

In \[20\] some challenges are addressed in the use of CBR for context-aware AmI systems. The four main challenges are

1. Acquiring the initial cases
2. Coping with the vast number of cases
3. Knowing when to initiate a CBR cycle
4. Knowing whether a case was classified correctly

It is also described how some of these challenges could be solved in two domains (in a touristic domain and in a hospital ward) using CREEK.

The most recent and similar tool that we could find is SituRes[18], a case-based approach to recognising user activity in a smart-home environment. The algorithm used to perform CBR is a weighted 1-NN. “When the nearest neighbours are returned, their activation is compared to that of the test case. If the activation is below a threshold value we view this test case as being novel and add it to the case base automatically – we judge that the case base does not have adequate coverage in this area. The threshold value is defined externally, and is chosen based on what degree of learning is required.” Another interesting part of this system is the feature selection algorithm, which is based on Information Gain (IG). “To create a cut-down case base, the unique solutions are identified. For each of these solutions, a feature selection process is employed, whereby the features of each corresponding case are ordered based on how predictive they are of that case. This ordering is performed using the information gain algorithm. We used Ontonym to link each case feature with its related sensor and from this we could determine the location and type of this
A single case is then created based on these features, and added to the casebase. This process is repeated for each solution, leaving a casebase which contains as many cases as there are solutions; in this example the casebase is reduced from 6003 to 36 cases. This approach has a number of drawbacks, however. Although the casebase size is reduced dramatically, incorrect or inappropriate features might have far more influence should they be ranked highly. Given the limited size of the data set we used, noisy data has a high impact on accuracy, and this is magnified by the pruning process.” Even if this approach improves learning, however there is a great weakness; the accuracy of the system (the F-measure\(^\text{2}\) actually) is poor, as admitted also by the authors. SituRes is based on one of the publicly available datasets that use PlaceLab [21]. PlaceLab is a 1000-square-foot sensor-rich home environment, built by MIT as a shared research facility. Unfortunately, the PlaceLab ontology (available at the Ontonym website\(^\text{3}\)) is not in a form that our system can exploit (as described later), so we could not compare our system to SituRes. However, it is our intention to expand our system and make it capable of processing a wider range of ontologies, since PlaceLab and its datasets seem to be a milestone in this field.

SituRes is based on a framework that is specifically designed for the development of CBR systems, called FIONN [22]. Then FIONN can return the k most similar cases (applying the kNN algorithm) and it also has the ability to perform feature selection and a variety of evaluation schemes, like cross-validation and hold-out validation. Since it deals with kNN it also offers the ability to learn similarity measures. The results obtained from the feature selection module can easily be evaluated to investigate whether or not overfitting has occurred. Overfitting occurs when a prediction model works well only with the data it used for training and its performance is actually significantly lower when it comes to real data. In other words, overfitting is the problem of learning only how to cope with data used for training and the inability to generalize. An advantage of CBR is the potential to use retrieved cases to support explanation. This idea is supported in Fionn in a framework called explanation guided retrieval.

FIONN reads data in CBML form [23]. CBML is a way of representing cases for CBR in XML. The CBML specification document is written in XML Schema and all case structure documents must follow this schema exactly to be considered valid CBML. By using the CBML interfaces and parsers it is possible to totally separate case description from the underlying CBR code. Once the structure document is changed, the case parser will expect the new structure. If new similarity or adaptation rules are needed these can easily be imported into the system too. This separation of structure from the underlying application code is the main advantage of using CBML. The original goal behind CBML development was to create an open standard to facilitate case based network computing, case base storage and possible interoperability with non-CBR systems.

In [24] presented by people who had previously worked on PlaceLab, the aim was to recognize not only the current activity, but also its intensity in some cases (eg. speed if someone is running, rpm if someone is cycling etc), using five triaxial wireless accelerometers and a wireless heart-rate (HR) monitor. One interesting aspect is that they found the use of the HR monitor rather impractical, since it lags

---

\(^{2}\) F-measure is calculated as: \((2 * \text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall})\)

\(^{3}\) [http://ontonym.org/PlaceLab](http://ontonym.org/PlaceLab)
physical activity and remains altered once the activity has finished (that leads to errors at the end of an activity or at the beginning of the next one).

In his PhD thesis [25] E. Munguia Tapia (who has also worked in the PlaceLab project) demonstrates how wireless accelerometers (called MITes) can be used for real-time recognition of physical activity type, intensity, and duration and estimation of energy expenditure. Some activity recognition algorithms are described as well as options for signal segmentation (sliding windows, overlapping sliding windows and signal spotting).

COSAR[26] is a hybrid reasoning system that uses ontologies and ontological reasoning combined with statistical inferencing. In this paper a historical variant is introduced, which considers the past few cases when deciding on the current case, taking advantage of the fact that persons tend to perform the same activity for a certain lapse of time before changing activity. The statistical inferencing method used in this system is multiclass logistic regression. COSAR is an offline reasoned, meaning that it does not run in real-time.

Activity recognition is only a small part of CBR applications. CBR can be applied to a great number of fields. Actually we witness CBR in our everyday lives, when for example a plumber fixes a sink, based on his earlier experience on fixing sinks, or when we suggest a film to a friend, based on our previous experience of suggesting films of this type to this friend and his reaction.

HeyStaks[27] is a search utility that is designed to work with mainstream search engines by allowing users to organise, share and reuse their particular search experiences. The web provides a rich source of explicit and implicit experiences, which are usually lost, or are not kept in the first place for others. In this work one can find an attempt to explore a type of experience reuse, and the challenges that it presents. Each repository is called a search stak and is effectively a case base of search cases.

Music for my mood [28] is a music recommendation system based on context reasoning. Context reasoning is implemented using CBR in order to recognize the user’s mood and listening intentions, after sensing the context. Then it recommends the top 15 songs that match the user’s mood, from the songs that the user listened to in the past week.
Chapter 3

Solution Description

In this chapter we discuss how certain issues of our implementation are handled, aspects that our system covers and aspects that it does not cover (yet), our required input and things that our system contributes to the state-of-the-art knowledge-extraction methods based on ontologies.

3.1 Main idea

To improve the performance of CBR exploiting semantics, we had to design and implement a system as fast as possible, that would also be as accurate as possible. This could be seen as a trade-off between speed and accuracy, since faster systems are usually worse in terms of precision than slower, but more robust systems. However, we did not try to lower our standards for any of these two crucial fields. Our goal was to create a system faster and more accurate than any other similar system that we are aware of.

Initially we tried to implement our system using existing tools, but soon we were disappointed with their time performance. The accuracy was acceptable, but real-time systems demand faster performance. Hence, it was speed issues that led to the design and implementation of a new system. For our new system, we tried a rule-based reasoning approach (i.e. using Prolog), but we had to reject it for two reasons:

- Speed was not acceptable again, when it came to large-scale datasets
- We could not find an easy way to generalize such a system for more than one domains; the rules were too specific to be easily reused.

Therefore, we found that the use of machine learning algorithms was almost obligatory for our criteria. Even from our first tests we could see that the difference was great. Machine learning algorithms were fast enough to allow us the design of a real-time system. Thereafter, accuracy became our main target.

Since the speed problem was solved, we expected to find accuracy results similar or slightly worse than what we had seen with the slower tools. On the contrary, we were surprised when we found out that, even from our first tests, machine learning algorithms were sometimes significantly better than the other tools in terms of accuracy too. We could not stop there, however, because our goal was to create a system that would be able to exploit the semantics of an ontology and machine learning approaches did not do that. So, our last theoretical challenge was to keep the advantages that we had seen and build a system that could also grasp the information that an ontology can offer. To accomplish that, we could only change the input given to our algorithms, which is the dataset, when dealing with machine learning. In other words, we had to create feature vectors that would “encode” the semantics of an ontology.

The implementation of the four RE’s of CBR can be found in the following sections:

1. REtrieve: Retrieving the most “similar” case or rather the most possible solution is a task that machine learning algorithms can handle. The reader can
find more details on how this is accomplished in §2.1 regarding machine learning and specifically Bayesian networks.

2. REuse: Solving the activity-recognition problem is handled basically by suggesting the most possible solution, so it is almost equivalent to retrieving. Again, §2.1 can help the reader understand how this is possible.

3. REVise: There are situations in which the proposed (most possible) solution is not acceptable. In these situations we make further steps in order to find a better, more-fitting solution. If these steps fail, no solution is returned. We preferred to skip a prediction than predicting something wrong. These steps are described in §3.7.

4. RETain: §3.7 and §3.8 describe how the most possible results can be validated and saved in the dataset. Even if this step is not yet implemented, there are some ideas on how this can be done, without the use of any additional code, by using different tools.

We claim that our system can load the compatible(§3.2), given ontology (used as input), transform it to a ML dataset, recognize the current activity (from a given list of activities) and respond in real-time (within a small – depending on the system’s needs - timeframe) with an acceptable accuracy, as shown in our “Evaluation” chapter.

3.2 User input requirements

After dealing with a number of existing tools, we concluded that simplicity is the key to making such a system practical. Furthermore, we found out that many options that a user might have (such as setting weights to attributes or setting priorities), could affect the bias of the system. Therefore, we tried to make the information provided by the user as little as possible. The input that the system requires is just an acceptable ontology (details on what is an “acceptable” ontology follow) and the parts of the ontology that help in the recognition part:

- The attribute that we want to recognize (e.g. “Activity”)
- The ontology class\(^4\) that contains the cases (instances)
- The object and datatype properties of the above “Case class”

For more complex ontologies, where we would like to grasp more properties, the required input is slightly different, as we will discuss later.

Ontologies can vary in many fields, depending on their purpose, or even on their designer. For our system’s needs, we have set some requirements for an ontology that will be accepted by our system. The given ontology could contain any properties, axioms, classes or values, as long as, following our sample ontology shown in Figure 1, there are instances like the one shown in Figure 9. This means that:

- Each case is stored as an instance of a specific class (here named “Case” class)
- All case attributes, including the activity (recognition target) are given as properties, which have the above case class as their domain
- All values are direct instances (in case of object properties) of a “leaf” class (a class with no children)

\(^4\) The reader should be aware that the term “class” is used in machine learning to denote the attribute that we want to predict/recognize, whereas an ontology class is like an object in OO-programming
There must be a class with no subclasses (here named “Activity”) describing the activity (recognition target) and a property with the case class as domain and the activity class as range (here named “has-Activity”).

```
<Case rdf:ID="Case74">
  <has-Weather rdf:resource="#Stormy"/>
  <has-Time rdf:resource="#Noon"/>
  <has-Stance rdf:resource="#Standing"/>
  <has-Activity rdf:resource="#Cooking"/>
  <has-Person rdf:resource="#Nikos"/>
  <has-Season rdf:resource="#July"/>
  <has-Location rdf:resource="#Kitchen"/>
</Case>
```

Fig. 9: A sample case instance, based on the sample ontology of Figure 1. Cases are stored as instances of a class (here named “Case”). Attributes are stored as the values (range) of properties that have the case class as domain (here “has-Weather”, “has-Time”, “has-Stance”, “has-Person”, “has-Season”, “has-Location”). The recognition target (here activity), is also stored in the same way as attributes (using the “has-Activity” property).

The user should define that terms Weather, Time, Stance, Person, Season and Location describe the attributes of the problem, term Activity is used to describe the solution and each case is stored as an instance of term Case. The object properties that appear in this sample case are also shown in Figure 10. To help the user identify which parameters are required as input, we have also created a Graphical User Interface

![Graphical User Interface](image)

Fig. 10: The object properties of the class “Case”. Edges represent properties. Their source (the class “Case”) represents their domain and their destination (“Person”, “Season”, “Stance”, “Weather”, “Time”, Location”, “Activity”) represents their range.

(GUI) sample, which could be used as in Figure 11. The term Nominal values is used to distinguish attributes between those that accept nominal values and those that accept numeric values, in other words, this field distinguishes properties between object and datatype. Object properties in RDF/OWL are properties whose range is a class. For example has-Location is an object property, since it has Location as range and Location is a class. Datatype properties on the other hand have some datatype as
range. For example a property that describes the number of passengers in a ship (e.g. `hasNumberOfPassengers`) is a datatype property since its range is a number.

### 3.3 Exploiting the semantics of an ontology

The most important and innovative part of our system is the machine-learning-dataset manipulation that helps in grasping the semantics that ontologies can offer. In this paragraph we discuss how this can be achieved and we also compare some of our approaches, which led to our final solution.

Our first approach was the simplest one could find; having the data described above as input, we simply store the values of the given properties in a dataset. For the example of Figure 9 we would have a feature vector with these values:

*Cooking, Nikos, July, Stormy, Noon, Kitchen, Standing*

For convention we store the class attribute first. Our dataset contains a number of feature vectors like this, each representing a distinct case (even if the attributes have exactly the same values). A detail of great importance is that we always want to have the same number of attributes and each row should represent the same variable (missing values are accepted, but still written in the dataset as such – e.g. with a ‘?’). All these feature vectors have the same order for the attributes (i.e. `Weather` is always the 4th feature). Of course, this approach does not exploit any semantics from the ontology, but we used it as a yardstick, just to see if our following approaches will improve or deteriorate the system’s performance and in what extend.

Our second approach was going a little bit further. In this approach we tried to include some of the information that we could gain from an ontology. In order to accomplish that, we would have to add some attributes to the feature vectors that we had previously designed. The number of attributes added should be equal to all feature vectors, as explained before. We decided to include the parent value of each attribute, which would show us “where they belong”. This would make a huge difference, since - following the ontology of Figure 1 - our algorithms would now know that *Nikos* is a
Resident, July is a Summer month, Stormy is a Bad Weather, Noon means FullLight, Kitchen is located on the FirstFloor and Standing is a Stance. It is obvious to us that we now have more information about our data and it should be easier for our algorithm to recognize the current activity. However such information could "confuse" our algorithm as well, if we gave irrelevant or excessive information, so evaluation was necessary. To distinguish which values belong to the same level of hierarchy, we also included their depth. For example, as derived from Figure 1, a Resident has depth 2, Summer has depth 2, Bad (Weather) has depth 2, FullLight has depth 3, FirstFloor has depth 3 and Stance has depth 1. We could not follow the same procedure for the Activity class, since its value is supposed to be unknown to us, when we try to recognize it. To put all this data together, Case74 shown in Figure 9 would be represented in our dataset with the following feature vector:
In our third and final approach, we tried to include as much information as possible from the given ontology. Even if the second approach took advantage of some information available, it didn’t fully depict an ontology in a dataset. For example, even if Doctor and Nephew both have a depth of 3, the previous approach does not take into consideration that Nephew is a FamilyMember, whereas Doctor is a MedicalStaff. The simplest solution would be to store the whole path that leads to the desired value, i.e. for Kitchen, we would store Kitchen, FirstFloor, Indoors (Location would be always the same for the has-Location property, so it would be omitted). This however means that for the Backyard we would store Backyard, Back, Outdoors which contains one attribute less than the Kitchen, since they do not have the same depth, so we could not use this solution. Instead we tried to depict the path in a different way that maintains the same number of “sub-attributes” for each attribute of the feature vector. This was to store a Boolean value for each of the subclasses of the attributes (only if the properties they come from are object properties, which means that their range is a class and not a number). We know that this number is fixed, since the subclasses of a class are known from the beginning. As for the values of these Boolean attributes, they would be true (1) if the subclass belonged to the value’s path and false (0) else, as formulated in Figure 12.

\[
\text{if } P \in \text{properties}, C = \text{range}(P), a = \text{instance}(C), \\
\text{then } \forall D \in \text{subclasses}(C), \text{value}(D) = \begin{cases} 1, & \text{if } D \in \text{superclasses}(a) \\ 0, & \text{else} \end{cases}
\]

Fig. 12: How Boolean attributes are valued for our final solution

Case74 would now be stored in our dataset as:

Cooking,Nikos,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,July,0,0,0,1,Stormy,1,0,0,1,0,0,1,0,1,Kitchen,1,1,0,0,0,0,0,0,Standing

For example the Boolean values following Noon mean that Noon belongs to FullLight and Light and does not belong to HalfLight or Dark. The Boolean values following Kitchen mean that Kitchen belongs to FirstFloor and Indoors and does not belong to Basement, SecondFloor, Front, Back, or Outdoors. Standing is not followed by any Boolean values following, since Stance has no subclasses.

3.4 Semantics covered

The aforementioned approaches only cover the hierarchy of an ontology, namely the subclassOf property. This is of course, the most important notion that ontologies describe, but there are also some “minor” issues that could also help us improve the “mapping” between the ontology and our dataset.

There could also be a case in which the range of an objectproperty is a union of classes and not just a single class, as we have seen until now (e.g. a property contains might have a Ship as domain and Passengers or Vehicles as range). This is a case that we would like to avoid, because of the restriction that we should have a fixed number of attributes. If this restriction did not exist, we could just store the
attributes for one of the classes in this union, the one that is “active”. But since different classes lead to different number of attributes, this would not be applicable. Our suggested solution is to store the attributes for every class in the union and then keep values only for the class that is “active”. We use the word “active” to refer to the class of the union that is used in an instance, unlike the rest of the classes that are not used. E.g. if an instance of a case contains an instance of a Vehicle, then the “active” class for the property contains is the Vehicle class and not the Passenger class. For the rest of the classes and the attributes that follow them, we use unknown_value (?) and 0 accordingly. The source code for this transformation is simplified as:

```java
rangeClass = property's range;
if (rangeClass equals("DefaultOWLUnionClass")) {
    Set nested = rangeClass.getNestedNamedClasses();
    for each nestedClass in nested {
        add the nestedClass to the attributes;
    }
} else {
    //normal case (rangeClass is just one class, not a union)
    add the rangeClass to the attributes;
}
```

A usual phenomenon is the occurrence of the same property more than once in the instance of a case. For example an instance of a Case might contain many instances of Events. In this situation, we use a maximum number of occurrences for the same property and add this number of attributes to our dataset (along with any possible subclasses). For example this section of the code would start like:

```java
for (i=0; i< #given properties; i++) {
    for (j = 0; j < maxValuesOfProperty i; j++) {
        …store attributes as usually…
    }
}
```

Another minor (though sometimes it could be major) information that ontologies can offer is the information gained from the properties of classes, not necessarily those included in a case instance. If for example the range class of an objectproperty that is contained in a case instance, has some properties – either object or datatype properties – then we can also store these values as new attributes. If for instance a Person class has the properties has-Habit and has-Disease, then our previous example case would take the following form in the dataset:

Cooking,Nikos,0,0,0,0,0,0,0,0,0,0,0,1,Smoking,Alzheimer,July,0,0,0,1,Stormy,1,0,Noon,0,1,0,1,Kitchen,1,1,0,0,0,0,Standing

If the above properties have no values for a certain instance, then they could be encoded as an unknown_value (?).
3.5 Reducing the dataset

One of the four main challenges in [20] is coping with the vast number of cases, namely removing some cases occasionally to limit memory needs. In an ideal scenario, in which we would have unlimited memory, we would not care about this issue. It is highly impossible to keep an accuracy as good as the accuracy observed before the reduction. The goal is that the new accuracy will be as close to the initial one, as possible. There are several ways to reduce the dataset, but none of them is perfect, with no disadvantages. Some of the most popular solutions are:

- **Removing some of the most common cases** and keeping only some of them. The most common cases are the ones that have the same class attribute, which is the most frequent one. Many machine learning algorithms, however, take into consideration the frequency with which the values appear in the dataset, so the system would be biased against this value if we changed this frequency.

- **Removing the rarest cases**. Again, this would mean that it would be even more difficult, or impossible for some algorithms to correctly recognize the existence of these cases.

- **Removing some of the oldest cases**. This could include removing some of the rarest cases, or all of them, with the risk of permanently “forgetting” them, but could actually work well in some environments.

- **Removing random cases**. This seems to be fair enough, since no bias is applied, however, there is a chance that some (or even all if they are few) important cases will be removed, highly risking the system’s future accuracy.

In our implementation we followed a different approach, similar to the last one described above. We used WEKA’s (WEKA is described in §5.3) unsupervised Resample filter, which reduces the dataset to a percent (we have set this percent to be 50%) of its initial size, with zero bias, by removing random cases. The crucial part of this filter is that it keeps the same ratio for the distribution of cases, depending on their class attribute. This means that if we have 200 cases for example and they are split into two classes - let them be true and false with 140 (70%) of them being true and 60 (30%) of them being false - then if we resample this dataset and keep only 100 cases, we would expect to see about 70 remaining cases classified as true and 30 cases classified as false. This would preserve the distribution of classes and help classification task act similarly to how it acted before the dataset reduction. This approach, of course has some drawbacks, such as the deletion of cases that might cover some “variations” of a class; things that lead to the same conclusion (prediction), by using different observations/approaches could be lost.

Reducing the dataset is not necessarily limited to reducing the cases. Another approach would be to reduce the features of the feature vector, which would apply – as explained previously – to all the cases. We noticed that with our “full-path” approach to exploiting the semantics of an ontology, we also gather some redundant information; values that are always the same, given some other values. For example if a student can only be described as undergraduate, or postgraduate and we know that a certain student is not an undergraduate, we can safely conclude that he is a postgraduate student. In this approach we take as granted that only “leaf” classes can have instances, which means that if a class has subclasses, then instances of this class are not taken into consideration. To model this attribute reduction, we have used the following rule:
for each subclass S (not only direct) of a given objectproperty’s range class
if S only has one father F and F is marked
    add S to the attributes
else
    mark F

This removes only one, the firstly-found subclass of each “level”. It actually does not add this to the attributes, rather than “removes” it. By the term “level” we mean the siblings of the same father (only its direct subclasses). To illustrate this reduction let us consider again the sample ontology of Figure 1, and a sample case of this ontology as described in Figure 9. The attributes that we would not include in our smaller dataset, would be FamilyMember, Nephew, Nurse, Indoors, SecondFloor, Front, Winter, Light, HalfLight and Good. It is important to note that even if we remove an attribute, we can still use some of its subclasses as attributes. For example, Persons can only be FamilyMembers, MedicalStaff, or Residents (only one of these Boolean choices can be true). So we do not need to keep information about all of them. We can skip one and we chose to skip the first one: FamilyMember. However, we still need to distinguish FamilyMembers into Nephew, Spouse, Daughter, Grandson, Granddaughter, Niece, or Son. Again, only one of these Boolean attributes can be true for a single instance. So, if we know everything about all the attributes except one, we can still make the same conclusions. In this case we chose to skip the Nephew attribute.

3.6 Caching

When it comes to large-scale datasets, like the ones used in real-time activity recognition, the JenaOWLModel created by Protégé (Protégé is described in §5.1), might not be small enough to be kept in memory (cached). In this situation, the system would load an owl model each time from the ontology (.owl) file in the hard disk. This process would take place for each cycle of reasoning, namely for every single activity-case to be recognized. Of course, this procedure is too slow to allow real-time reasoning.

To avoid this drawback, a JenaOWLModel is loaded only once in memory. This can be achieved by keeping a separate repository only for the instances/individuals of the ontology. This way, the JenaOWLModel contains only the ontology and no individuals (no cases). The initial case(s) are kept in a Protégé Database Project, a kind of repository, which also allows fast insertion of new cases. These cases are loaded before any other new case. This means that they are stored in the case-base, so that they can be part of the ML process. Then, each new case is added directly to the case-base. If, after the reasoning part, the case is considered as correctly-classified, then it is also added to the instances repository (Protégé Database), so that they can be available when the system needs to start this process again. The part of code described here looks like this:

```java
Project prj = Project.loadProjectFromFile(dbProjectFileName,errors); OWLModel initialCases = prj.getKnowledgeBase(); writeInCaseBase (initialCases.getInstances);
```
JenaOWLModel owlModel = loadOntology(ontologyFile);
for each new case
writeInCaseBase (newCase, owlModel);
solve(newCase);
if newCase was solved correctly
    initialCases.add(newCase);

3.7 Historic variant

In COSAR[26] a historical variant is introduced, which considers the past few cases when deciding on the current case, taking advantage of the fact that persons tend to perform the same activity for a certain lapse of time before changing activity. We found this idea really interesting and decided to incorporate such behaviour in our system too. However, our implementation of the historic variant is slightly different. In our implementation, the historic variant is only used when the likelihood of the current prediction is lower than a certain threshold. We keep a record queue of the last few recognition results and then we set weights to these activities. The weights are set in a way that the most recent activities are more important than the least recent ones. The algorithm is like the one described here:

initialize all weights with 0
w = 1
for each position i in the history queue (starting from the oldest)
    weight[activity recognized in i] = weight[activity recognized in i] + w
    w = w + 1

If the activity with the greatest weight has likelihood above a threshold - which is lower this time – then it is returned as the predicted activity. If this threshold is not met again, then the activity recognition system does not return any result. For example, if the last 6 predicted activities (in order of occurrence) were: [bathing, bathing, bathing, cooking, eating, eating], then the weights of these activities would be:
weight[bathing]: 1 + 2 + 3 = 6
weight[cooking]: 4
weight[eating]: 5 + 6 = 11

So eating has the greatest weight, even if it appears fewer times than bathing. If the current recognized activity had likelihood below a threshold, then we would examine if eating’s likelihood is above a (lower) threshold. If it is, then eating is returned as the current activity, else no activity is returned.

3.8 Validation of predictions

A complete CBR system, retains (the fourth “RE”) the parts of this experience likely to be useful for future problem solving. This is a major issue in such a system, since this part of the system is responsible for learning. The problem here is that if we knew from the beginning the correct classification of the cases, then we would not need to
perform reasoning, but even then, we would make no incorrect predictions. So we should seek a method to grasp the true classification of our cases, in order to use these cases as an addition to our dataset. There are generally two approaches to solve this problem:

1. Asking the users. This is the easiest approach and probably the best in terms of confidence. The users cannot make a mistake in the classification. Even if they choose a class we think is wrong, it is most likely that the error is ours and not theirs, since our system aims to help the users. However, such a validation technique would not be viable, since it would be really intrusive and irritating for the users. Few people could bare a system that asks validation for every step they make. No matter how accurate such a system could become, we think that this approach is wrong. An improvement on such systems would be the limiting of confirmations it requires from the users. For example it could only ask the users when a threshold of uncertainty is reached. This would drastically eliminate the intrusiveness. If not, it would mean that this system is uncertain quite often, so there is probably a need for redesign.

2. Second level of reasoning. In real-life, when we are unsure of a person’s current activity – this means that we have a possible activity in mind, or even two possible activities – we sometimes wait for confirmation or rejection of our initial thought. We do that by observing some more actions and then decide if our intuition was correct. Other times, it is easier to ask someone else “What is this person doing?” If our opinions agree, then we become more confident of our initial thought, otherwise, we may reject it. If a similar situation occurs again in the future (including the same person), we would think that it must be the same activity that we remember in the past. If we had accepted our intuition as correct, then this should be the current activity also. If we had rejected it in the past, then we should also reject it again. With this approach we do not seek the true classification, but a classification as close to the true one as possible, since if there was a way to have the completely true classification automatically, our system would be useless.

The second approach could become as complex as designing a new system. Therefore, we decided to actually use a different system, an offline-activity-recognition reasoner[29]. This reasoner could help our system confirm or reject its predictions once in a while, and keep our dataset with as little noise as possible. Consequently, validation of predictions will not apply in real-time, but it can be scheduled once in a while (eg during the night, or once a week).

### 3.9 Missing prerequisites

Since machine learning - and particularly BNs - can cope with missing data, there might be cases or misclassifications, in which an activity is returned when some of the logical or expected prerequisites (simpler activities) either have not occurred, or have not been identified. In this case, the activity recognition system can inform the “context” that this is the situation and further mention which are the specific prerequisites that were not identified for the current activity.

To accomplish this feature, when the desired activities-to-be-recognized are given to the activity recognition system, a list of prerequisites for each activity could
be also provided. The prerequisites are the properties of a Case class, as described in §3.2. For example, if the currently recognized activity is Cooking and there is a prerequisite for this activity, like Pot_on_HotPlate, which has not been identified, the system returns a message like: “The current activity is ‘Cooking’, but ‘Pot_on_HotPlate’ is not true…” This feature aims to provide other parts of a bigger Aml system with information that could be used to warn or assist users. For the example given, the user could be informed that there is nothing on the hot plate, which is on. Otherwise, the hot plate could be automatically turned off.

3.10 Scaling-up

Even if it is highly unlikely that all these semantics covered will exist in a single ontology, we should examine how our system scales-up. Namely, what the relation between the “size” of the given ontology and the number of attributes that we store in our dataset is.

For this analysis we will assume that there are \( n \) properties in each case and in the worst case, all of them are object properties. This means that, apart from their value (range) we have to store attributes for their range-class’ subclasses (if any).

Let’s say that the first range-class (the range of the first object-property) has \( s_1 \) subclasses, the second has \( s_2 \) subclasses... and the \( n^{th} \) range-class has \( s_n \) subclasses. Let’s also represent the number of attributes that are stored for each case, given \( n \), as \( \text{atts}(n) \).

In its simplest form (covering only the hierarchy) each case in our dataset would contain one attribute for the value of each property and one attribute for each of its range-class’ subclasses:

\[
\text{atts}(n) = 1 + s_1 + 1 + s_2 + \ldots + 1 + s_n = n + \sum_{i=1}^{n} s_i
\]

If we also consider the possibility that the range of an object-property is a union of classes and not a single class, then the worst case would be that every object-property has a union of classes as range. With \( u_1 \) we represent the number of classes that exist as a range of the first property, with \( u_2 \) we represent the number of classes that exist as a range of the second property, .... with \( u_n \) we represent the number of classes that exist as a range of the \( n^{th} \) property. Each of these classes might also have subclasses. Let \( \text{sub}_{1,1} \) be the number of subclasses that the first class of the first range-union (the range of the first property) has, \( \text{sub}_{1,2} \) be the number of subclasses that the second class of the first range-union has, ..., \( \text{sub}_{1,u_1} \) be the number of subclasses that the \( u_1^{th} \) class of the first range-union has, ..., \( \text{sub}_{n,u_n} \) be the number of subclasses that the \( u_n^{th} \) class of the \( n^{th} \) range-union has. Then we would need one attribute for each one of the unions’ classes, as well as one attribute for each of these unions’ classes’ subclass:
Finally, we consider the chance that each property takes several values for each case, which means that it exists more than once in a case. For example the first property takes \( t_1 \) values for a case, the second property takes \( t_2 \) values, ..., the \( n^{th} \) property takes \( t_n \) values. Then we simply multiply each row with the appropriate \( t \):

\[
\text{atts}(n) = t_1 (1 + \text{sub}_{1,1} + 1 + \text{sub}_{1,2} + \cdots + 1 + \text{sub}_{1,n}) + \\
t_2 (1 + \text{sub}_{2,1} + 1 + \text{sub}_{2,2} + \cdots + 1 + \text{sub}_{2,n}) + \\
\cdots + \\
t_n (1 + \text{sub}_{n,1} + 1 + \text{sub}_{n,2} + \cdots + 1 + \text{sub}_{n,n}) = \\
= t_1 u_1 + t_2 u_2 + \cdots + t_n u_n + t_1 \text{sub}_{1,1} + t_2 \text{sub}_{1,2} + \cdots + t_n \text{sub}_{n,n} = \\
= \sum_{i=1}^{n} t_i (u_i + \sum_{j=1}^{u_i} \text{sub}_{i,j})
\]

The last equation could be considered as a part of the previous equations, if we think of \( n \) as the number of properties in a case instance, including duplicates.
Chapter 4

System Architecture

4.1 AmI Sandbox

The Institute of Computer Science of the Foundation of Research and Technology – Hellas (FORTH - ICS) has initiated a long RTD AmI program [30] aiming to develop pioneering human-centric AmI technologies and Smart Environments, capable of “understanding” and fulfilling individual human needs. Under this program it created a laboratory space of about 100m² comprising six rooms, aiming to provide researchers the opportunity to bring along and share their know-how and resources in order to obtain hands-on experience and experiment in a highly flexible setting. In this space, various AmI technologies and applications are installed, integrated and demonstrated, and multiple ideas and solutions are cooperatively developed, studied and tested [31].

For our system’s demonstration (on the domain presented in §6.1) we are also going to use the Smart Office prototype environment. In this environment there are a smart office, a smart board, as well as a set of sensors of many kinds. Using this environment for the implementation of the scenarios in real time we will be able to draw conclusions about the efficiency and the accuracy of the system. For the ambient assisted living domain (§6.2) we are going to use mainly services from the Home Controls and Health prototype environments. The latter includes measuring pressure and temperature of a patient lying on a bed.

AmI services are defined through a tool called Idlematic which creates and keeps service interfaces. All services communicate through FAMINE middleware, responsible for creating, connecting and consuming these services. It is CORBA-based and provides support for C++, Java, .NET, Python, and Flash/ActionScript languages.

Sensor data are handled like mouse/keyboard events in Java. Some (already implemented) services (which we found by using the Idlematic tool) can be used to catch the sensor signals and inform the user about the type of signal and what it means (by defining an event). In the AmI Sandbox there are sensors for sound, image, lighting, interaction, “smart” devices and more, including cameras, RFID readers, IRIS scanner, door-chair controllers, touch screens, projectors, PDAs, speakers and workstations. When an event occurs then it can be caught using event handlers. These event handlers are used to store the sensor data in the respective ontology.

Apart from the event handlers, Idlematic and Famine also provide the ability to change the context, by calling some methods. These are methods that interact with the environment (such as the ability to close a door, turn a device on etc). This part of the system is used mainly to assist users, rather than recognize the current activity and it is not the main focus of this work. For the activity recognition we are interested in handling events, not creating events.
4.2 Context

As described in Chapter 6, we have used ontologies to model the context, like Location, Person, Time etc. The context has to be available to any other system task that may need it. This is why the ontology is a part of our shared memory between threads. Any task that needs access to the ontology, asks permission, waits for it to be free and then gains access. While this task has access to the ontology, the other tasks have to wait for the first task to finish, if they want to gain access to the ontology. The tasks that need the ontology are the SWRL task, the event handler task and the activity recognition task. If the ontology can be divided in many sub-ontologies, then more parallelism could be achieved.

As an example of this, one could imagine two different ontologies (one modelling the activities and one modelling the sensors) being imported by a main ontology (modelling everything else). In that case, the activity recognition task would only have to share the activity ontology with the SWRL task, since they would be the only tasks to read/write on it. The event handler task would share the sensor ontology only with the SWRL task, since they would be again the only tasks to read/write on this ontology. If the ontology was not divided, then each of these tasks would have to wait for the other two to complete their access to the ontology.

4.3 Activity Recognition

For our implementation of the machine learning algorithms we use WEKA (§5.3). WEKA reads/writes files in arff format. An ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes. ARFF files have two distinct sections. The first section is the Header information, which is followed the Data information. The Header of the ARFF file contains the name of the relation, a list of the attributes (the columns in the data), and their types. An example header on the standard IRIS dataset looks like this:

```%
% 1. Title: Iris Plants Database
%
% 2. Sources:
%     (a) Creator: R.A. Fisher
%     (b) Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
%     (c) Date: July, 1988
%
@RELATION iris

@ATTRIBUTE sepalwidth NUMERIC
@ATTRIBUTE petalwidth NUMERIC
@ATTRIBUTE class {Iris-setosa,Iris-versicolor,Iris-virginica}
```

The Data of the ARFF file looks like the following:

```%
@DATA
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
%```
Consequently, we also use arff to store our case base. Each line of the data section represents a case. The Attributes that we use are the ontology properties that the user passes as arguments. If there are object properties, then their range class is found and all the instances of this class are stored as possible nominal values. If there are datatype properties, then they do not necessarily take nominal values, so they are stored as numeric attributes. String attributes are avoided, since we prefer to store them as nominal values, for reasons of learning. Boolean attributes, that are used to describe the “ontology path” of instances, can be either represented as numeric, or nominal with \{0, 1\} or \{true, false\} as their possible values.

In our domains each case represents a time-frame typically lasting 10 seconds, but generally depending on the specific domain. This means that for an hour of reasoning we need about 360 cases/lines and for a day of reasoning we need around 8640 cases/lines in our case base/arff file. Soon it becomes obvious why it is important to have some data reduction algorithms. WEKA’s Resample filter can perform this task and return a new arff file as a result.

### 4.4 User assistance

The goal of a system that incorporates activity recognition is (or should be) to assist its users. Even if our work in activity recognition is not concerned with user assistance, this is its main target, as part of a larger project. The assistance that an AmI space could provide varies depending on its nature and on the needs of its users. In this system, user assistance is implemented, like anything else, as a separate thread.

The thread responsible for assisting the user starts after the activity recognition thread has finished its task. This action is usually interference with the AmI space. It can be as simple as calling a service that turns the lights off, opens a door, or increases the TV’s volume or it could become as complicated as taking over the cooking process, or notifying a doctor about a medical emergency and the situation of the patient.

User assistance is the “final” task of the system’s cycle, since its results are not needed by any other task and no other task has to wait until it finishes. The “success” of the user assistance is highly dependent, if not equivalent to the success of activity recognition. The final two rules in §6.2 are intended to assist users. More information on user assistance will be available in the theses of Maria Koutraki and Dimitra Zografisou.
4.5 Thread architecture

The architecture of our more general system (the one that incorporates user assistance) is mainly based on parallelism. To achieve this, we make use of threads. Each task is built on a separate thread. These threads can communicate safely by sharing data (e.g. ontologies, reasoning results, input arguments), since the shared memory can only be accessed by one thread at a time.

Thus we have implemented one thread for sensor handling, one for the activity recognition task, one for running the SWRL rules and one for user assistance. Figure 13 is a graphical representation of the threads’ scheduling. The horizontal axis is an abstract representation of time. SWRL is actually a single thread that repeats the same procedure. It is depicted as multiple entities, to represent that it calls the activity recognition thread after a certain time of repetitions (5 in this example). This way, the activity recognition thread can start periodically after a relatively fixed timeframe, since the SWRL thread’s running time is stable (there are no differences in the size of the data it processes). So if, for example, SWRL takes 2 seconds to infer knowledge and it calls the activity recognition thread every 5 executions, then the latter is called almost every 10 seconds. SWRL creates (change the Boolean values of) Simple Events, so when the activity recognition thread starts, every Simple Event that occurred within the timeframe of 10 seconds is considered to be true. The sensor handler thread keeps running throughout the system execution. It is responsible for catching sensor events and storing them in the ontology. SWRL rules also write in the ontology, the simple events that take place and that will be used from the activity recognition thread. The latter, as described above, is called from the SWRL thread and when it finishes, it returns its result to the user assistance thread. This is responsible for changing the environment, according to what is expected to help the users given their current activity.

It is notable that there are times when 3 threads can run in parallel. This occurs after the SWRL thread calls activity recognition, or when activity recognition calls user assistance. Then we have sensor handler, SWRL and activity recognition/user assistance accordingly running together. That is why we need to make sure that the shared memory is safe, meaning that only one thread can access it at a time for reading/writing.\footnote{This is done by using the keyword “synchronized” in Java.}

If threads were not used, then the total execution time of a “reasoning cycle” would be significantly greater. Apart from this problem, there would also be another, probably more important drawback without the use of threads. The thread that is responsible for sensor handling as well as the one responsible for writing Simple Events to the ontology would be “inactive” for a considerable time. This means that some signals, some information about the users occurring during this time of inactivity would not be known at all. This would also dramatically decrease the credibility of such a system.
Fig. 13: Threads’ scheduling. The “Sensor Handler” thread keeps running constantly, storing the sensor data in the ontology. The “SWRL” thread runs once in a while, performing a first level of reasoning and storing its results as “Simple Events” in the ontology. Then the “Activity recognition” system is called (here for every 5 runs of the “SWRL” thread), which returns the current activity to the “User Assistance” thread.
Chapter 5

Technologies Used

To develop our system, we used some free tools that are very popular and robust. To create ontologies for testing the knowledge extraction part we used Protégé and for machine learning purposes, we used the Weka toolkit. These tools are described here in details.

5.1 Protégé

Protégé is a free, open-source platform that provides a growing user community with a suite of tools to construct domain models and knowledge-based applications with ontologies. At its core, Protégé implements a rich set of knowledge-modeling structures and actions that support the creation, visualization, and manipulation of ontologies in various representation formats. Protégé can be customized to provide domain-friendly support for creating knowledge models and entering data. Further, Protégé can be extended by way of a plug-in architecture and a Java-based Application Programming Interface (API) for building knowledge-based tools and applications. The Protégé platform supports two main ways of modeling ontologies:

- The Protégé-Frames editor enables users to build and populate ontologies that are frame-based, in accordance with the Open Knowledge Base Connectivity protocol (OKBC). In this model, an ontology consists of a set of classes organized in a subsumption hierarchy to represent a domain’s salient concepts, a set of slots associated to classes to describe their properties and relationships, and a set of instances of those classes - individual exemplars of the concepts that hold specific values for their properties.

- The Protégé-OWL editor enables users to build ontologies for the Semantic Web, in particular in the W3C’s Web Ontology Language (OWL). "An OWL ontology may include descriptions of classes, properties and their instances. Given such an ontology, the OWL formal semantics specifies how to derive its logical consequences, i.e. facts not literally present in the ontology, but entailed by the semantics. These entailments may be based on a single document or multiple distributed documents that have been combined using defined OWL mechanisms" (see the OWL Web Ontology Language Guide).

Protégé is supported by a strong community of developers and academic, government and corporate users, who are using Protégé for knowledge solutions in areas as diverse as biomedicine, intelligence gathering, and corporate modelling. We used Protégé to create our sample and experimental ontologies, as well as the Protégé OWL-API in order to facilitate the communication of our Java code with the created ontologies.
5.2 LibSVM

LIBSVM [32] is a library for SVMs, an integrated software for support vector classification, (C-SVC, nu-SVC), regression (epsilon-SVR, nu-SVR) and distribution estimation (one-class SVM). It also supports multi-class classification. Its goal is to help users from other fields to easily use SVM as a tool. LIBSVM provides a simple interface where users can easily link it with their own programs. Main features of LIBSVM include

- Different SVM formulations
- Efficient multi-class classification
- Cross validation for model selection
- Probability estimates
- Various kernels (including precomputed kernel matrix)
- Weighted SVM for unbalanced data
- Both C++ and Java sources
- GUI demonstrating SVM classification and regression
- Python, R, MATLAB, Perl, Ruby, Weka, Common LISP, CLISP, Haskell, LabVIEW, and PHP interfaces. C# .NET code and CUDA extension is available. It's also included in some data mining environments: RapidMiner and PCP.
- Automatic model selection which can generate contour of cross validation accuracy.

We used LibSVM with Java, for the initial implementation of our system.

5.3 Weka

Weka [33] is a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from your own Java code. Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes. Weka is open source software issued under the GNU General Public License.

We used Weka in order to test a variety of machine learning algorithms, before deciding to choose Bayesian Networks. Weka can include LibSVM for SVM algorithms. We found its Graphical User Interface (GUI) really helpful and time-saving. A great asset of Weka is that with the same — or almost the same — dataset and the same source code, we could test a great variety of algorithms, filters, feature selection, which significantly reduced the implementation time of our system.
Chapter 6

Applications

The reason why we designed this activity recognition system was the implementation of two Ambient Intelligent systems, which would use this activity recognition task. The two domains that these systems cover are the smart classroom and the ambient assisted living domain. As expected, our system could not be used in the same way for both domains. Therefore, we had to make some adjustments for these domains, considering each one’s specificities.

Our context ontology is divided into two hierarchical parts, upper-level ontology and low-level ontologies. The Upper Level Ontology or Core Ontology captures general features of all pervasive computing domains. It is designed in a way that can be reused in the modelling of different smart space environments, like smart homes and smart meeting spaces. The Core ontology’s context model is structured by a set of abstract entities like Computational Entity, Person, Activity, Location, Simple Event and Environment (Figure 14). All these entities are widely used, except Simple Event entity. Simple Event entity aims to capture knowledge obtained from reasoning on sensors data e.g. ‘Projector’s status is “on”’ or ‘Teacher is in front of the smart board’.

The Low Level or Domain-Specific Ontologies are based on upper level ontology and specified by the domain. In our case the domains are an intelligent classroom in a university campus and assisting the life of the elderly in a smart home. Some of the domain-specific ontologies are Person and Location ontology. All of the ontologies are expressed in OWL.

![Diagram](image)

**Fig. 14:** The core ontology, used for both the “Smart-classroom” and the “Ambient Assisted Living” systems. “Simple Events” are recognized through SWRL rules and “Activities” are recognized through our activity recognition system.
6.1 Smart Classroom

In a typical classroom, a lot of time and effort is sometimes spent on technical issues, such as lighting, projector set-up, photocopy distribution etc. This time could be replaced with “teaching time”, if all these issues were solved automatically.

There are many Smart Classroom systems that try to change the behaviour of an environment in order to improve the conditions of a class. One of them is [34] that focuses on making real-time context decisions in a smart classroom based on information collecting from environment sensors, polices and rules. Another context aware system is [35] that supports ubiquitous computing in a school classroom.

For this domain, we implemented a system that assists instructors and students in a smart classroom, in order to avoid spending time in such minor issues and stay focused on the teaching process, by also having more studying material at their disposal. To accomplish this, we have taken advantage of the benefits that ontologies and machine learning offer, by using our activity recognition system.

Scenarios in our work express the activities that can take place in a classroom e.g. lecture, exam. In order to identify these activities, a series of simpler events should be applied, in each case. After that, we undertake to assist the particular activity.

A typical scenario is “Student Presentation”. A student is giving a lecture or paper presentation in the Smart Classroom. For this scenario many students and teachers may appear in classroom. The classroom’s calendar is checked for lecture at this time and personal calendars of all these people are checked for participation in this presentation, in order to know that everyone is in the right Classroom. The lights and projector are turned on. A student stands near the display of presentation. Teachers and students should be seated. After all these minor events “Student Presentation” activity is identified. After the identification of the activity, we assist the presentation by turning off the lights and adapting the presentation file in students’ and teachers’ devices.

A simple description of a complete cycle of our system is the following, as depicted in Figure 15:

![Smart Classroom Architecture](image)

**Fig. 15:** Smart Classroom Architecture. Sensor data from the classroom are stored in the ontology. Then SWRL rules perform a first level of reasoning, storing their results in the ontology. The activity recognition system uses these results and returns the current activity to the Classroom Assistant. The latter may alter the state of the classroom.

1. Sensor data are stored in an ontology using sensor handlers. e.g. In our scenario “Student Presentation”, data from lights sensor or data from RFID sensors for people location stored into our ontology.
2. SWRL rules are used for a first level of reasoning, to create *Simple Events*. In “Student Presentation”, some simple events are: *student stands near the display of presentation, teacher sits*.

3. The *Simple Events* that occurred within a timeframe are passed to the Activity Recognition System.

4. The Activity Recognition System loads the cases and finds the current activity.

5. The result is written in the case base as a new case and also passed to the *Classroom Assistant* system.

6. Depending on the current activity, the *Classroom Assistant* changes the context. In our scenario turns of the lights and adapt the presentation’s file in students’ and teachers’ devices.

In our implementation we try to transform our scenarios for intelligent classroom into rules. This rule based approach is implemented by using SWRL (Semantic Web Rule Language). The first step is to capture data from sensors into the ontology (e.g. status of devices). After that, SWRL rules are applied on these data and save the result into the Simple Event class. Examples of rules are shown below.

**Rule: High Noise Level**

\[
\text{SmartClassroom(SmartClassroom1) } \land \\
\text{Core:Environment(SmartClassroomEnvironment) } \land \\
\text{Core:hasEnvironment(SmartClassroom1, SmartClassroomEnvironment) } \land \\
\text{Core:noise(SmartClassroomEnvironment, ?noise) } \land \text{swrlb:greaterThan(?noise, 70)} \rightarrow \text{isActivated(High-Level_Noise, true)}
\]

**Rule: High Temperature Level**

\[
\text{SmartClassroom(SmartClassroom1) } \land \\
\text{Core:Environment(SmartClassroomEnvironment) } \land \\
\text{Core:hasEnvironment(SmartClassroom1, SmartClassroomEnvironment) } \land \\
\text{Core:temperature(SmartClassroomEnvironment, ?temperature) } \land \\
\text{swrlb:greaterThan(?temperature, 30)} \rightarrow \text{isActivated(High-Level_Temperature, true)}
\]

**Rule: High Lighting Level**

\[
\text{SmartClassroom(SmartClassroom1) } \land \\
\text{Core:Environment(SmartClassroomEnvironment) } \land \\
\text{Core:hasEnvironment(SmartClassroom1, SmartClassroomEnvironment) } \land \\
\text{Core:lighting(SmartClassroomEnvironment, ?lighting) } \land \\
\text{swrlb:greaterThan(?lighting, 300)} \rightarrow \text{isActivated(High-Level_Light, true)}
\]

**Rule: Low Lighting Level**

\[
\text{SmartClassroom(SmartClassroom1) } \land \\
\text{Core:Environment(SmartClassroomEnvironment) } \land \\
\text{Core:hasEnvironment(SmartClassroom1, SmartClassroomEnvironment) } \land \\
\text{Core:lighting(SmartClassroomEnvironment, ?lighting) } \land \\
\text{swrlb:lessThan(?lighting, 220)} \rightarrow \text{isActivated(Low-Level_Light, true)}
\]

**Rule: Projector is on**

\[
\text{Projector(Projector1) } \land \text{Core:status(Projector1, "on") } \rightarrow \\
\text{isActivated(Projector_is_On, true)}
\]
Rule: **Classroom Lights Low Level**
\[ \text{Lights}(\text{ClassroomLights}) \land \text{Core:status}(\text{ClassroomLights}, "low") \rightarrow \text{isActivated}(\text{Classroom\_Lights\_Low\_Level}, \text{true}) \]

Rule: **Classroom Lights Medium Level**
\[ \text{Lights}(\text{ClassroomLights}) \land \text{Core:status}(\text{ClassroomLights}, "medium") \rightarrow \text{isActivated}(\text{Classroom\_Lights\_Low\_Medium}, \text{true}) \]

Rule: **Classroom Lights High Level**
\[ \text{Lights}(\text{ClassroomLights}) \land \text{Core:status}(\text{ClassroomLights}, "high") \rightarrow \text{isActivated}(\text{Classroom\_Lights\_High\_Level}, \text{true}) \]

Rule: **Student in front of Projector**
\[ \text{Projector}(\text{Projector1}) \land \text{Person:Student}(?s) \land \text{Core:inFrontOf}(?s, \text{Projector1}) \rightarrow \text{isActivated}(\text{Student\_inFrontOf\_Projector}, \text{true}) \]

Rule: **Professor in front of Projector**
\[ \text{Projector}(\text{Projector1}) \land \text{Person:AcademicStaff}(?a) \land \text{Core:inFrontOf}(?a, \text{Projector1}) \rightarrow \text{isActivated}(\text{Professor\_inFrontOf\_Projector}, \text{true}) \]

Rule: **Student Located in Classroom**
\[ \text{Person:Student}(?s) \land \text{Core:hasLocation}(?s, \text{SmartClassroom1}) \rightarrow \text{isActivated}(\text{Student\_LocatedIn\_Classroom}, \text{true}) \]

### 6.2 Ambient Assisted Living

Ambient assisted living is the domain in which the resident(s) of a “smart home” need assistance for their everyday routine. These residents could be elderly people, people with impairment etc. The assistance that these people may need varies depending on their situation. It could be a simple reminder of a desirable action (medication), a danger prevention (switch cooker off), an emergency alert (notify relatives that the resident needs medical help, call an ambulance) etc. For our scenario based on this domain we have mainly focused on the assistance of the elderly and particularly patients with Alzheimer’s disease, the most common form of dementia.

The system architecture is the same as in §6.1. Here are some SWRL rules that result in Simple Events, based on sensor data written in the ontology from sensor handlers. These results (the Simple Events) are also used for the activity recognition.

Rule: **Bathroom Door Closed**
\[ \text{core:status}(\text{BathroomDoor}, "off") \land \text{core:isActivated}(\text{Bathroom\_Door\_Closed}, \text{false}) \rightarrow \text{core:isActivated}(\text{Bathroom\_Door\_Closed}, \text{true}) \]

Rule: **Cooker Status**
\[ \text{core:status}(\text{CookerID}, "on") \land \text{core:isActivated}(\text{CookerOn}, \text{false}) \rightarrow \text{core:isActivated}(\text{CookerOn}, \text{true}) \]
Rule: Facing TV
\[ \text{core:facing(John, LivingRoomTV)} \land \text{core:isActivated(Facing_TV, false)} \rightarrow \text{core:isActivated(Facing_TV, true)} \]

Rule: In Front of Cooker
\[ \text{core:inFrontOf(John, CookerID)} \land \text{core:isActivated(User_In_Front_Of_Cooker, false)} \rightarrow \text{core:isActivated(User_In_Front_Of_Cooker, true)} \]

Rule: Lie in Bed
\[ \text{lying_in(John, Bed)} \land \text{core:isActivated(Lie_in_Bed, false)} \rightarrow \text{core:isActivated(Lie_in_Bed, true)} \]

Rule: TV on
\[ \text{core:status(LivingRoomTV, "on")} \land \text{core:isActivated(TV_On, false)} \rightarrow \text{core:isActivated(TV_On, true)} \]

Rule: Sitting on Sofa
\[ \text{core:sittingOn(John, Sofa)} \land \text{core:isActivated(Sitting_On_Sofa, false)} \rightarrow \text{core:isActivated(Sitting_On_Sofa, true)} \]

Here is an example of rules that aim in user assistance (that occurs after activity recognition). In this example we would like to turn off the lights, if the recognized activity is *Sleeping*.

Rule: Lights Off With Sleep Detection
\[ \text{core:activityStatus(Sleeping, true)} \land \text{core:status(Bedroom1Light, "on")} \rightarrow \text{core:startService(turn_the_lights_off, true)} \]

Rule: Set Lights Off
\[ \text{core:startService(turn_the_lights_off, true)} \land \text{core:status(Bedroom1Light, "on")} \rightarrow \text{core:status(Bedroom1Light, "off")} \]
Chapter 7

Evaluation

The ideal evaluation of our system would require a number of similar tools that are open-source or even publicly available, so that we can reproduce their experimental results. Another approach would require a number of suitable ontologies with real datasets, so that we could prove that our system performs better/worse than what we have today. Unfortunately, the evaluation of our system is a rather difficult task, for many reasons:

- **It is difficult to acquire ontologies that meet our requirements as described in Chapter 3.** Ontologies are usually designed to fulfil certain needs. Since our approach is not popular, there is no need to design such an ontology. This lack of suitable ontologies, obliges us to design new ontologies that have not been tested on other systems.

- **It is even more difficult to have a real dataset based on such an ontology.** Even if a suitable ontology is found, it is even more difficult to find datasets that are based on this ontology. Then we have to “create” a dataset on our own, with the risk of being far from realistic.

- **There are not many tools that have the same target** (we have only found one with almost the same target), so it is impossible to compare our system’s performance to other tools. When we started this project we were happy to find JCOLIBRI that we were then considering to use as our main tool. However, due to a number of limitations – described before – we decided to design a tool of our own. JCOLIBRI is publicly available and also offers two ontologies that are compatible with our implementation, as well as a GUI. We were then also happy that our tools seemed to outperform JCOLIBRI, but this is where we expected to find more similar tools, in order to compare our system with them. Unfortunately, we did not find such tools. The best we could find was just experimental results based on unavailable data, so we could not run our system on data we do not have.

This chapter presents our experimental results in comparison to JCOLIBRI, the trivial classifier, our initial implementations using SVMs and in some cases the performance we could have without the use of ontologies. We also present here the results of our smart classroom application that was described in the previous Chapter. First of all, we should introduce the methods that we used in order to measure our system’s performance.

We will use the terms *training set* and *test set* to denote the data that we use to train our models (what we know) and the data that we use to estimate the performance of our models (what we want to predict). Usually, we only have one type of data, for which we know the correct solutions. In order to test our models, we split this dataset into training and test sets in a number of ways and then estimate our system’s performance as the average of these numerous “small” tests.

For the SVM methods we did not use WEKA, but LibSVM from Java with the default SVM parameters. We present here the steps of our SVM methods’ evaluation:
1. The case base is firstly split into training and test sets, according to the type of the test.
2. Then only the training set is scaled and the scale parameters are kept. It is important that the scaling of the training data occurs after the data is split into training and test set. Otherwise, information of the test set could be passed to the training set, by defining different scaling parameters that also take the test set into consideration. This would make our model biased.
3. After that, we train a model based on our training set.
4. The test set is then scaled with the same parameters that the training set was scaled, stored in step 2.
5. In the final step a prediction is made for the test set based on the model.

To test JCOLIBRI we followed a different procedure, since no training is required (kNN is a lazy learning method). The procedure was as simple as comparing each test case with every other case belonging to the training set, in other words finding the nearest neighbour from the training set. No weights were given (this means that all weights were set to 1.0), since there is no automated/unbiased way to put weights to attributes, unless using a solution like neural networks. However, this would complicate things and also it is not required when using JCOLIBRI.

To evaluate the performance of the systems we used two Cross Validation (CV) schemes; Leave One Out Cross Validation (LOOCV) and 10-fold Cross Validation. One round of cross-validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset (the training set), and validating the analysis on the other subset (the testing set). To reduce variability, multiple rounds of cross-validation are performed using different partitions, and the validation results are averaged over the rounds. LOOCV and 10-fold CV are just two ways of dividing the dataset into training and test set.

LOOCV is quite a descriptive name… In this CV scheme all data are kept as training set, except one case that is kept as a test set. In the end, every case has been considered as a test set exactly once. This means that we have as many rounds as the number of the dataset. This is the disadvantage of this method; in large datasets, with thousands or millions of cases, this method requires thousands or millions of rounds accordingly. However it has the advantage that every single case is examined. The final accuracy is calculated as the number of correct classifications divided by the total number of cases.

10-fold CV is a similar process to LOOCV. The original dataset is partitioned into 10 subsamples. One of these subsamples is used each time as a test set and the remaining 9 are used as training set. The performance estimated is the average of the performance of each of the 10 models. For the purposes of our evaluation, the dataset is divided in 10 folds in the same way for every experiment. LOOCV can be considered as a k-fold CV, with k being equal to the total number of cases in the original dataset.

The trivial classifier is the simplest solution one could provide to a classification problem. It always classifies a case as the most common class. If we have a binary classification problem, with one class being observed in 90% of the samples, then the trivial classifier would always suggest this class as the solution and would be correct in the 90% of the cases. The trivial classifier is usually used as a comparison when no other (or few other) comparisons can be made, as in our case. As a result, there is no point in designing a system that performs worse than the trivial classifier.
**Vacation.owl** is an ontology provided with JCOLIBRI. It contains 125 vacation-package cases, called *VACATION_CASE*, each of them containing all of the following attributes:

- Category (regarding accommodation standards)
- Destination
- Price
- Persons
- Season
- Transportation
- Holiday Type (the reason of this vacation package)
- Duration

This ontology also demonstrates how our system can be used for non-activity-recognition purposes. Due to the lack of such datasets, we tried to make the most out of the available ones, so we ran two tests with this dataset. On the first test our aim was to correctly classify the *Category* of the vacation. The possible results of this test are:

- ONESTAR
- TWOSTARS
- THREESTARS
- FOURSTARS
- FIVESTARS
- HOLIDAYFLAT

On the second, we tried to recognize the reason of the vacation, namely the *Holiday Type*. The possible results of this test are:

- RECREATION
- WANDERING
- BATHING
- ACTIVE
- SKIING

**AssistedLiving.owl** is the ontology presented in Figure 1 and used as a sample throughout this work. This ontology was created for testing purposes and it is not based on real observations. It is the initial ontology that we used for our system, and we now have a richer ontology to describe this domain, as presented in the previous chapter. For this ontology we have 100 cases of activity recognition, like the one presented in Figure 9. These cases are not obtained from real data, but they were also created by us. The possible results of this test are:

- eating
- cooking
- resting
- sleeping

**US citizens’ income dataset** (also known as the “Adult” dataset or the “Census Income” dataset) is one of the most popular machine-learning datasets. It contains 48842 instances (including 3620 cases with missing values), with a mix of continuous and discrete values, which is split into 32561 training instances and 16281 test
instances. It is publicly available\(^6\) and it has received a great number of citations. This dataset and the *IRIS* dataset (used in §4.3 as an example) are the two most popular datasets in machine learning. Extraction was done by Barry Becker from the 1994 Census database. The prediction task is to determine whether a person - citizen of the USA - makes over 50 thousand dollars a year. The available attributes are the following:

- **age**: continuous.
- **fnlwgt**: continuous.
- **education**: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
- **education-num**: continuous.
- **relationship**: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
- **race**: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
- **sex**: Female, Male.
- **capital-gain**: continuous.
- **capital-loss**: continuous.
- **hours-per-week**: continuous.
- **native-country**: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinidad&Tobago, Peru, Hong, Holand-Netherlands.

In this dataset we followed a different approach to test our system. We tried to design a simple ontology that would describe this dataset and then perform a prediction task based on this ontology. Then we compare the results of this prediction task to the results of the predictions that use only the dataset. Therefore, this test does not aim to prove that our system performs better than another tool for a given ontology. It aims to prove that the use of ontologies can improve the prediction results in some cases. Our ontology is as simple as splitting *native-countries* into these belonging to the G8 (Canada, France, Germany, Italy, Japan, the United Kingdom, the United States and Russia) and these not belonging to the G8 and also split occupation into three categories; Physical, Mental and Armed. We thought that these criteria could give us information about a person’s income.

In machine-learning and particularly in decision trees, this additional piece of information is called Information Gain (IG). In general terms, the expected IG is the

\(^6\) UCI Machine Learning Repository [http://archive.ics.uci.edu/ml/datasets/Adult]
change in information entropy (uncertainty) from a prior state to a state that takes some information as given.

\[ IG(Ex,a) = H(Ex) - H(Ex | a) \]

So, in machine-learning terms, we expected that these criteria would offer high IG, and therefore improve the accuracy of this prediction task.

### 7.1 Accuracy

In Figure 16 and Table 1 we present our experimental results, after testing each of the ontologies and algorithms as described above. \textit{SVMnoDepth}, \textit{SVMParent} and \textit{SVMFull} refer to our initial approaches described in details in §3.3. \textit{BNFull} is our final approach, similar to \textit{SVMFull}, but using BNs instead of SVMs. \textit{LOOA}ssisted refers to LOOCV applied in the AssistedLiving.owl. \textit{LOO}Type refers to LOOCV applied to the task of recognizing the \textit{Holiday Type} of the vacation.owl. \textit{LOOC}ategory refers to LOOCV applied to vacation.owl for the prediction of the \textit{Category} of the vacation package and \textit{10fold}Assisted and \textit{10fold}Category refer to the same tests, run with 10-fold CV, instead of LOOCV. It is obvious that LOOCV and 10-fold CV results are almost identical for the same type of test, therefore some of them are skipped. In Figure 17 and Table 2 we present the results on the US citizens’ income prediction task. \textit{BNplain} refers to the tests that were held without the use of ontologies, in contrast to \textit{BNFull}. We also performed two tests in this dataset. The first test was performed on the test set provided with the dataset, with a model trained on the rest of the dataset (the training set). For the second test we used the training set only to perform 10-fold CV. It is important to note that for these test we have also included the cases that contain missing values.

<table>
<thead>
<tr>
<th>LOOA</th>
<th>LOOT</th>
<th>LOOC</th>
<th>10fAss</th>
<th>10fCat</th>
</tr>
</thead>
<tbody>
<tr>
<td>JCOLIBRI 2.0</td>
<td>80%</td>
<td>32%</td>
<td>39.2%</td>
<td>79%</td>
</tr>
<tr>
<td>SVMnoDepth</td>
<td>86%</td>
<td>28%</td>
<td>56.8%</td>
<td>84%</td>
</tr>
<tr>
<td>SVMParent</td>
<td>74%</td>
<td>33.6%</td>
<td>56.8%</td>
<td>72%</td>
</tr>
<tr>
<td>SVMFull</td>
<td>91%</td>
<td>42.4%</td>
<td>56.80%</td>
<td>90%</td>
</tr>
<tr>
<td>BNFull</td>
<td>95%</td>
<td>39.20%</td>
<td>53.60%</td>
<td>94%</td>
</tr>
</tbody>
</table>

**Table 1**: Performance results

<table>
<thead>
<tr>
<th>testSet</th>
<th>10fCV</th>
</tr>
</thead>
<tbody>
<tr>
<td>BNplain</td>
<td>84.21%</td>
</tr>
<tr>
<td>BNFull</td>
<td>84.63%</td>
</tr>
</tbody>
</table>

**Table 2**: UScitizens’ income results
Fig. 16: Performance results. “LOO” stands for Leave-One-Out Cross Validation and “10fold” stands for 10-fold Cross Validation. “Assisted” means Assisted Living, “Type” means that the target of the recognition is the holiday type and “Category” means that the target of the recognition is the accommodation category. For these tests no data are missing. Our final approach (BNFull) always performs better than JCOLIBRI 2.0 in these tests.
At this point it is interesting to discuss the results of the trivial classifier for each of the experiments presented above.

For the AssistedLiving.owl experiment, the trivial classifier would always predict eating, since it is the most frequent class in the dataset. This would provide an accuracy of 28 correct classifications out of 100 total classifications, namely 28%, which is significantly worse than any other value presented in Table 1 and Figure 16, regarding this test.

For the prediction of the Holiday Type in the vacation.owl dataset the trivial classifier would always suggest RECREATION as a solution, since this can be found 39 times in the 125-instance-long dataset. This would give an accuracy of 31.2%. This is far from an indifferent result. JCOLIBRI just managed to outperform the trivial classifier with an accuracy of 32% and our initial approach – SVMnoDepth – is actually outperformed by the trivial classifier, with an accuracy of just 28%!

For the prediction of the Category of the vacation package, the trivial classifier would always prefer HOLIDAYFLAT and this would give it an accuracy of 36.8% (46/125), again slightly worse than that of JCOLIBRI (39.2%), but significantly worse than our approaches (56.8%).

Finally, for the prediction of a US citizen’s income the trivial classifier would do pretty well, suggesting always that this income is below $50K, achieving an accuracy of 76.07%. Our approach, as well as the most accurate approaches recorded in the bibliography, is about 8-10% better than the trivial classifier.

Apart from these test, we also present here the experiments that were performed for the evaluation of the applications described in Chapter 6. In the absence of a real dataset for a smart classroom, we decided to create one, in order to evaluate our system. The precision of this application is mostly based on the activity recognition’s precision, since everything else is rule based. Therefore we present here the evaluation results for a dataset that we built based on our observations on a
publicly available video from a real lecture\textsuperscript{7}. Our observations – which act as sensor data – include the position of the lecturer, the lighting, the persons that speak etc. In this video the activities observed are 4:

- lecture with slides
- lecture with whiteboard
- question and
- conversation.

A 10 fold cross validation based on this dataset is illustrated in Tables 3 and 4.

<table>
<thead>
<tr>
<th>Total Number of Instances</th>
<th>326</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correctly Classified Instances</td>
<td>313</td>
</tr>
<tr>
<td>Incorrectly Classified Instances</td>
<td>13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TP Rate</th>
<th>FP Rate</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
<th>ROC Area</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.995</td>
<td>0.077</td>
<td>0.959</td>
<td>0.995</td>
<td>0.977</td>
<td>0.994</td>
<td>lecture_slides</td>
</tr>
<tr>
<td>0.955</td>
<td>0.008</td>
<td>0.977</td>
<td>0.955</td>
<td>0.966</td>
<td>0.994</td>
<td>lecture_wb</td>
</tr>
<tr>
<td>0.955</td>
<td>0.007</td>
<td>0.913</td>
<td>0.955</td>
<td>0.933</td>
<td>0.997</td>
<td>Question</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.816</td>
<td>conversation</td>
</tr>
</tbody>
</table>

Table 3: Detailed accuracy by class on the smart classroom domain

<table>
<thead>
<tr>
<th>lecture_slides</th>
<th>lecture_wb</th>
<th>question</th>
<th>conversation</th>
<th>classified as</th>
</tr>
</thead>
<tbody>
<tr>
<td>208</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>lecture_slides</td>
</tr>
<tr>
<td>4</td>
<td>84</td>
<td>0</td>
<td>0</td>
<td>lecture_wb</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>21</td>
<td>0</td>
<td>question</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>conversation</td>
</tr>
</tbody>
</table>

Table 4: Confusion matrix of smart classroom domain

Table 4 can be read as “y activity was actually classified as x activity”. It would ideally contain zeros only in the non-diagonal positions and positive values only in the diagonal positions. So it appears that no “conversation” was classified correctly. Even if the system failed to recognize any conversation - since it is actually not clear even for a human observer to distinguish conversation from other similar activities – its total accuracy is considered satisfying. For the rest of the activities (lecture_slides, lecture_wb and question) our system had an accuracy of 313 correctly classified instances out of 319 total instances (98.12%), but this number skips the classification of the 7 instances that were observed to belong to the conversation class, so it is not “real” accuracy. There is also confusion between lecture with slides and lecture with whiteboard, which was expected, since, on the particular video, sometimes both categories could be considered true. Unfortunately, our system cannot return two activities that are occurring at the same time as a result.

\textsuperscript{7} http://videolectures.net/mlss08au_hutter_isml/. Part 2
7.2 Time

The most challenging test in terms of time performance is the income prediction. It contains, as stated before, 48842 cases. This is a good number of cases to test an activity recognition system, since as stated in §4.3 for a day (24 hours) of reasoning we need around 8640 cases. This means that the US income dataset is as big as a dataset of activity recognition lasting almost 5.5 days. It also contains 18 attributes for each case, which is a satisfying number also for a typical activity recognition task, even if there are not any standards. Even if in this example the target is not activity recognition, the purpose of this test is just to measure time. As mentioned before, this dataset could not fit in memory as an owl model, so we used the caching technique described in §3.6. For the first classification, we had to load the ontology, which cost much time. So for the first classification, we had to wait for 49 seconds! This is not an acceptable time for real-time reasoning of course. However, from the second run and thereafter, we did not have to load the ontology again, so the average time for the next 10 classifications was 1046 milliseconds. This is the time needed to write a new case, classify it and return the result, after reasoning on almost 49000 cases. Without the use of the caching technique, this average time would be almost equal to the initial time (49 seconds).

We have also reproduced the same 326 cases acquired from the video of a lecture to create a 43000-case-large dataset, just to simulate the time performance of our system, ignoring accuracy. The average time performance of the activity recognition system that was executed 100 times on this dataset is 757.401 milliseconds. Similarly, the average time performance of SWRL rules that were executed 10 times is 686.215 milliseconds. Even if SWRL are not a part of the activity recognition system, their performance is crucial, since the activity recognition system takes their output as its input.

The performance of the system is measured on a rather outdated PC, with the following features:
- Intel® Pentium® 4 CPU 3.40GHz,
- 3.39GHz, 1.99 GB of RAM,
- running Microsoft Windows XP Professional with Service Pack 3
- The tests were run with NetBeans IDE 7.0.1.

7.3 Demo

In order to run a real-world demonstration of our system, we have used the AmI Sandbox, described in §4.1. The data are real, and they were not based on assumptions. Of course, the ideal evaluation of this system would require its use in a real house/classroom with people that will be its final users, but this demo is the closest that we could get. To be more accurate, the data are based on the physical presence of a human in the AmI Sandbox, who acts according to a scenario. This scenario is designed based on an initial manually created dataset. Another way to create a dataset would be to record the data of a user’s arbitrary actions and then label each “timeframe” with an activity. We preferred the manually created dataset, for time’s and human effort’s sake.
This demo is based on an ontology similar to the one presented in Chapter 6, in which Activities are recognized based on SimpleEvents. The available resources (that could be useful for us) in the AmI Sandbox at the time of the demo were:

- a hospital bed, capable to recognize whether someone is on it or not.
- four televisions, capable of returning their state (on/off, volume, channel, etc)
- window blinds, capable of returning their state (open/closed)
- two projectors, capable of returning their state (on/off)
- lights, capable of returning their intensity
- RFID readers, capable of returning if someone/something approaches them

Unfortunately, we had to build a demo using only these resources, so the activities presented are limited. These are:

- sleeping
- watchingTV
- presentation
- talkingOnPhone

The sensor data are stored in the ontology and then periodically parsed for activity recognition. The only non-obvious activity is the “talkingOnThePhone” activity, based on the above sensors. In order to recognize whether a phone is on/off we used the RFID readers; an antenna was stuck under a desk and a magnetic sticker to the earpiece of a phone. The phone was placed on top of this desk, just above the antenna. This way, when the earpiece was picked, the sticker on it was away from the antenna and we could assume that the phone was on.

The demo is recorded with the help of two videos. The first video shows a person in the AmISandbox doing some activities, following a scenario. Initially she enters the room and (after a very tiring day at work) decides to rest for a while. Therefore she lowers the window blinds and lies on bed. But after so much tension she cannot sleep and decides to stand up, turn the lights and the TV on. She watches a music show for a while, but then the phone rings. She turns the TV off, and picks the phone up. It’s her boss, asking her to review the presentation that she will have tomorrow! She decreases the lights’ level and walks into the main room. There she turns the projector on and reviews the slides. The second video is a recording of the GUI controller that appears on the screen of the laptop that this person holds. In this controller there are three toggle buttons for turning on and off the TV, the projector and the window blinds. There is also a slider for setting the lights’ intensity. In the bottom of this GUI there is a text area in which the current activity and its likelihood are printed, as well as the last activities. This is a GUI that is only created for the need of the demo, and it is not a part of the system. When the likelihood is below a threshold (70% for this demo) then no activity is returned and the text area reads “No activity recognized”. This occurs usually in the transition between two activities.

The results of the demo were really satisfying, since all activities were recognized correctly and instantly. There is only one case which is recognized falsely and that is when the phone is off, the suggested activity is still “talkingOnPhone”. This is due to the sensor signal that lags for almost 1 second. The second video starts when the person entering the room starts recording the screen, but the system was initialized before that. That explains why the text area is not initially blank, but contains some “No activity recognized” cases. Figure 18 shows a screenshot of the video, some seconds after the phone is picked up.
Fig. 18: A screenshot from the demo. On the lower right corner is the GUI available to the user for the needs of this demo. The current activity – along with the likelihood that it is actually happening - is the last shown in the text area of the GUI.
Chapter 8

Conclusions

This paper was concerned with real-time activity recognition based on semantics-aware datasets. Its main contribution is the manipulation of ontological data, so that they can be processed from ML algorithms. ML is chosen due to the fact that it can cope with incomplete or conflicting data and fast enough to enable real-time reasoning.

We have shown how other CBR tools have been used to perform activity recognition. We have also presented the challenges in CBR addressed in bibliography, for which we later suggested solutions based on our approach. The required input and the output of our system are also described in details, as well as the way that data are handled. Some implementation details are also provided to help the reader understand how this system can be used with large scale datasets and still be able to run in real-time.

This system can be used as a part of larger systems that aim to assist their users, based on the current activity. The way that this can be implemented is also described, since we have designed and presented two separate systems, for two different domains that use real-time activity recognition. These domains are a smart classroom and ambient assisted living. The architecture of each system and the way that it interacts with our system is also revealed.

The experimental evaluation demonstrated that semantics improve the performance of CBR when it comes to activity recognition. A detailed analysis of the tools and technologies used for the evaluation aims to make these results comparable and reproducible for any other similar tools. Also the experiments show that this system could be used to improve the accuracy of other ML datasets than activity-recognition datasets too.

8.1 Limitations

Even if we tried to solve every problem that we came across, there are still some problems that we could not resolve. These problems might, or might not have a solution with our approach, so a further research on them could be interesting. There might also be more limitations that we were not able to note.

Every case in our scenarios has only one solution. This means that no parallelism is achieved when it comes to activities. For example a person might be talking to the phone and watching TV at the same time. Our system would (ideally) return only one of these activities as a result.

Similarly, the returned activity is only referring to one individual. In an environment with more than one individuals (like a smart classroom) the activity can be described as a “team” or “general” activity (like slide presentation, lecture etc). This means that we have not found a way to return one activity for each of the individuals present.
Even if the historic variant described in §3.7 can help, conflict detection and resolution are not thoroughly handled. Conflicts in activity recognition are described in [29]. If this system is used as a validation technique for our system however, then conflicts (detected by the validation system) could be avoided.

Another deficiency of the presented work is the lack of a higher-level reasoning based on the returned results. This could include different level of details, when it comes to results, conclusions on the results (e.g. the user is acting strangely, in confusion etc). However, we found it impractical to perform such kind of reasoning on a real-time system.

Finally, a serious problem is the lack of flexibility when it comes to input. ML datasets usually demand a specific number of specific attributes, as explained before, so our input could not be an arbitrary number of sensor indications.

8.2 Future Work

Future work includes further experiments including the application of this activity recognition system in real-environments with real data on more domains (currently we are working on the ambient assisted living, as well as the smart classroom domains). One of the most important features that is not yet implemented is retaining the correctly classified cases in the case base. Applying this system for various tasks is also of great interest for us. Further ways to improve the accuracy of the predictions are also taken into consideration. An important challenge would be to widen the system’s input requirements, so as to enable reasoning with a greater variety of ontologies.

We would also prefer a system that can be easily adapted by a wide range of domains and ontologies. Our initial goal was to create a real-time system that will recognize the activities of users in a certain environment. However, we later realized that such a system could have more capabilities and reason about many more things than just a user’s activity. Hence we could design a system that receives the desired “reasoning target” as input.
References


