Journal of Ambient Intelligence and Humanized Computing (2019) 10:2125–2142
https://doi.org/10.1007/s12652-018-0769-4

ORIGINAL RESEARCH

Methodology for improving classification accuracy using ontologies: application in the recognition of activities of daily living

A. G. Salguero[1] · J. Medina[2] · P. Delatorre[1] · M. Espinilla[2]

Received: 30 October 2017 / Accepted: 16 March 2018 / Published online: 27 March 2018

Abstract
Feature construction and selection are two key factors in the field of machine learning (ML). Usually, these are very time-consuming and complex tasks because the features have to be manually crafted. The features are aggregated, combined or split to create features from raw data. In this paper, we propose a methodology that makes use of ontologies to automatically generate features for the ML algorithms. The features are generated by combining the concepts and relationships that are already in the knowledge base, expressed in form of ontology. The proposed methodology has been evaluated with three different activities of a popular dataset, showing its effectiveness in the recognition of activities of daily living (ADL). The obtained successful results indicate that the use of extended feature vectors provided by the use of ontologies offers a better accuracy, regarding the original feature vectors of the classic approach, where each feature corresponds to the activation of a sensor. Although the classic approach produces classifiers with accuracies above 92%, the proposed methodology improves those results by 1.9%, on average, without adding more information to the dataset.

Keywords Machine learning · Ontology · Feature learning · Activity recognition · Activities of daily living · Smart environments · Data-driven approaches · Knowledge-driven approaches

1 Introduction

Supervised learning is a well-known task in the field of ML that consists on inferring a function from labeled training data. The training data are usually expressed in form of vectors, where each of the components of the vector is a feature or attribute of the sample data.

Feature engineering is a key in the development of data mining applications. The success of many learning schemata, in their attempts to construct models of data, hinges on the reliable identification of a set of highly predictive features (Hall and Holmes 2003). However, the task of feature construction and selection is tedious and non-scalable (Cheng et al. 2011). Usually, the features are manually crafted from raw data. This often relies on the expert knowledge and requires spending a lot of time thinking about how the underlying raw data is best exposed to predictive modeling algorithms. This means that features need to be aggregated, combined or split to create new features. While it is possible to identify correlation of particular features, the algorithms do not attempt to generate better features during model induction (Terziev 2016).

In this paper we propose the use of ontologies in order to improve ML algorithms. The structured knowledge contained in such ontologies can be exploited to automatically extract features for general learning tasks. More precisely, we propose in this work a methodology that can be used to generate new concepts by combining those already present in the knowledge base. The new concepts can be used as new features for the ML algorithms, so the knowledge base can actually be seen as a vast feature store. While most of the new concepts might be useless, we can eventually find some of them to be relevant for the problem, which increases the prediction accuracy of the ML algorithms. Our proposal

✉ A. G. Salguero
alberto.salguero@uca.es

J. Medina
jmquero@ujaen.es

P. Delatorre
pablo.delatorre@uca.es

M. Espinilla
mestevez@ujaen.es

[1] Universidad de Cádiz, Cádiz, Spain

[2] Universidad de Jaén, Jaén, Spain

is very useful when the data have some kind of underlying structure but there is no clear idea what the relevant features are or when the feature search space is so vast that they could not be generated manually. In those situations, the results show that our proposal improves the accuracy of the classifiers.

To evaluate our proposal, we have conducted multiple experiments in the field of sensor-based activity recognition in smart environments. This kind of activity recognition is based on identifying the actions of one or more people within an intelligent environment, by using a stream of observed sensor events that depend only on the current activity (Espinilla and Nugent 2017; Alemdar and Ersoy 2017). Common activities of interest are ADL such as "bathing","sleeping" or "dinning", for instance (Ferrández-Pastor et al. 2017; Shewell et al. 2017; Gutiérrez López de la Franca et al. 2017). Usually, objects or furniture can generate sensor events indicating, for example, the use of a faucet, the opening of a door, or the use of a light switch (Korhonen et al. 2003). We can even use much more complex sensors which give us information such as the posture of the people performing the activities (Gutiérrez López de la Franca et al. 2017).

Approaches used for sensor-based activity recognition have been divided into two main kinds: data-driven approaches (DDA) and knowledge-driven approaches (KDA).DDA are based on machine learning techniques in which a preexistent dataset of user behaviors is required. A training process is carried out to build up an activity model, which is followed by a testing process to evaluate the generalization of the model in classifying unseen activities (Li et al 2014). With KDA, an activity model is developed through the incorporation of rich prior domain knowledge obtained from the application domain, using knowledge engineering and knowledge management techniques (Chen and Nugent 2009a). KDA has the advantages of being semantically clear, logically elegant, and easy to get started. In the context of KDA, ontologies for activity recognition have provided successful results. In this kind of approach, interpretable activity models are built in order to match different object names with a term in an ontology that is related to a particular activity. Some hybrid approaches have been developed (Chen et al. 2014; Rafferty et al. 2015), which take advantage of the main benefits provided by DDA and the use of ontologies. Thereby, ontological ADL models capture and encode rich domain knowledge and heuristics in an understandable and processable way by the machine.

In this paper, we propose a hybrid approach for activity recognition. The use of an ontology is proposed in order to extend the feature vectors with asserted and inferred knowledge from the ontology, improving the accuracy of classifiers based on the DDA approach. An extensive evaluation is undertaken with a popular dataset to consider the effects of the extension of feature vectors, in terms of the overall accuracy for activity recognition based on sensor data obtained from different smart environments.

The remainder of the paper is structured as follows: next section provides a brief review of ontologies and some of the concepts needed to understand our proposal are revised; some of the related works found in the literature are revised in Section 3; Section 4 proposes the methodology to extend the feature vector by using ontologies; Sect. 5 presents an empirical study where our proposed methodology is applied to a popular ADL dataset; in Sect. 6, the results obtained are analyzed and discussed; finally, in Sect. 7, conclusions and future works are presented.

2 Ontologies

In this section, some relevant concepts related to ontologies are reviewed in order to understand our proposed methodology. Ontologies are used to provide structured vocabularies that explain the relations among terms, allowing an unambiguous interpretation of their meaning. Ontologies are formed by concepts (or classes) which are, usually, organized in hierarchies (Chandrasekaran et al. 1999; Uschold and Gruninger 1996), being the ontologies more complex than taxonomies because they not only consider *type-of* relations, but they also consider other relations, including *part-of* or domain-specific relations (Knijff et al. 2013).

The main advantage of ontologies is that they codify knowledge and make it reusable by people, databases, and applications that need to share information (Knijff et al. 2013; Wei et al. 2015). Due to this, the construction, the integration and the evolution of ontologies have been critical for the Semantic Web (Horrocks 2008; Kohler et al. 2006; Maedche and Staab 2001). However, obtaining a high quality ontology largely depends on the availability of well-defined semantics and powerful reasoning tools.

Regarding Semantic Web, a formal language is OWL (Horrocks et al. 2003; Sirin et al. 2007), which is developed by the World wide web consortium (W3C). Originally, OWL was designed to represent information about categories of objects and how they are related. OWL inherits characteristics from several representation languages families, including the description logic (DL) and Frames basically. OWL is built on top of the resource description framework (RDF) and (RDFS). (RDF) is a data-model for describing resources and relations between them. RDFS describes how to use RDF to describe application and domain specific vocabularies. It extends the definition for some of the elements of RDF to allow the typing of properties (domain and range) and the creation of subconcepts

Table 1 Semantic of OWL logical operators

	DL syntax	Manchester syntax	Semantics
$\mathcal{I}$	$C_1 \sqcap C_2$	C_1 *and* C_2	$(C_1 \sqcap C_2)^I = (C_1^I \cap C_2^I)$
$\mathcal{U}$	$C_1 \sqcup C_2$	C_1 *or* C_2	$(C_1 \cup C_2)^I = (C_1^I \cup C_2^I)$
$\mathcal{C}$	$\neg C$	*not* C	$(\neg C)^I = \Delta_I \backslash C^I$
$\mathcal{S}$	$\exists R.C$	R *some* C	$(\exists R.C)^I = \{x \mid \exists y.\langle x,y\rangle \in R^I \wedge y \in C^I\}$
$\mathcal{A}$	$\forall R.C$	R *only* C	$(\forall R.C)^I = \{x \mid \forall y.\langle x,y\rangle \in R^I \rightarrow y \in C^I\}$
$\mathcal{X}$	$\leq nR.C$	R *max* n C	$(\geq nR.C)^I = \{x \mid card\ \{y.\langle x,y\rangle \in R^I \wedge y \in C^I\} \leq n\}$
$\mathcal{M}$	$\geq nR.C$	R *min* n C	$(\leq nR.C)^I = \{x \mid card\ \{y.\langle x,y\rangle \in R^I \wedge y \in C^I\} \geq n\}$

and subproperties. The major extension over RDFS is that OWL has the ability to impose restrictions on properties for certain classes.

The design of OWL is greatly influenced by DL, particularly in the formalism of semantics, the choice of language constructs and the integration of data types and data values. In fact, OWL DL and OWL Lite (subsets of OWL) are seen as expressive DL, offering a DL knowledge base equivalent ontology. They are in fact extensions of the DL "Attributive Concept Language with Complements" ($\mathcal{ALC}$). More formally, let N_C, N_R and N_O be (respectively) sets of "concept names", "role names" (also known as "properties") and "individual names". The semantics of DL are defined by interpreting concepts as sets of individuals and roles as sets of ordered pairs of individuals.

A "terminological interpretation" $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ over a "signature" (N_C, N_R, N_O) for $(\mathcal{ALC})$ consists of the following concepts:

- A non-empty set $\Delta^{\mathcal{I}}$ called the "domain".
- A "interpretation function" $\cdot^{\mathcal{I}}$ that maps:
  - every "individual" a to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
  - every "concept" to a subset of $\Delta^{\mathcal{I}}$
  - every "role name" to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$

such that semantics in Table 1 holds.

The third column in Table 1 shows the Manchester OWL Syntax equivalent expression of the corresponding DL expression, in the second column. This syntax is derived from the OWL Abstract Syntax, but is less verbose and minimizes the use of brackets. This means that it is quicker and easier to read and write by humans than DL formal syntax (Horridge et al. 2006). The *subsumption* relation is usually expressed in DL syntax using the symbol $A \sqsubseteq B$, meaning that the concept A is a subset of the concept B.

In 2009, the W3C proposed the OWL 2 recommendation in order to solve some usability problems detected in the previous version, keeping the base of OWL. OWL 2 adds several new features to OWL. Some of the new features are syntactic sugar (e.g., disjoint union of classes) while others offer new expressiveness, including: increased expressive power for properties, simple metamodeling capabilities, extended support for datatypes, extended annotation capabilities, and other innovations and minor features (Zhang et al. 2015). One of the highlights of OWL 2 is the inclusion of profiles. The profiles are subsets of OWL 2, which provide key advantages in certain situations by means of a set of restrictions. Following, the profiles are defined briefly below.

- *OWL 2 EL* The use of this profile is recommended when dealing with extensive ontologies in which relatively complex entities are used (with a large number of properties). In these cases, the fundamental problem lies in the efficiency at the time of carrying out the classification and when propagating the properties associated with the entities. The solution to this problem is to reduce the expressiveness of the OWL 2 language. Therefore, in this profile, universal quantifiers, cardinality restrictions, the disjoint operator, the complement operator, or the enumerations for more than one individual, are not allowed. Additionally, it is not possible to define irreflexive, inverse, functional, symmetric or asymmetric properties.
- *OWL 2 QL* The use of this profile is indicated for those applications in which a high interoperability between OWL and the relational database systems is required. This situation occurs when working with relatively simple ontologies (thesauri and entity-relationship or UML schemes), but with a large number of individuals. OWL 2 QL is designed to facilitate access to these individuals through some languages, such as SQL. It is not possible to use the universal quantifier, cardinality restrictions or the disjoint operator. In the same way, defining subproperties, functional properties, inverse, transitive or connecting individuals with themselves are not allowed. Neither enumerations nor keys can be used.
- *OWL 2 RL* The OWL 2 RL profile is designed to facilitate the interoperability between the inference engines and the OWL language. It is based on the same idea

with which the QL profile is developed, but in this case, the objective is to facilitate the access to the set of individuals in the form of RDF triplets, improving the efficiency when making inferences. Most OWL 2 class constructions can be used in this profile under certain limitations, in terms of their syntactic position. This profile allows the use of any of the class axioms defined in the OWL 2 specification, except for the disjoint union of classes, the negative assertions and the reflexive properties.

3 Related works

In the activity recognition process, identifying a suitable sensor-based representation for building feature vectors is a key factor (van Kasteren et al. 2011). Previous works have been focused on evaluating expert-defined representations of binary sensors, such as, raw activation, last activation or change point (Ordónez et al. 2013; Singh et al. 2017). In this section we review some works that use structured knowledge sources to generate new features for classifying tasks.

If we consider all the information that can be inferred from an ontology as the input for a task of ML, the methodology presented in this paper could be considered as an approach for the problem of *feature learning*. Feature learning involves a set of of techniques to learn features and discover representations from raw data (Bengio et al. 2013). More specifically, feature learning techniques consist on the transformation of raw data input to a representation that can be effectively exploited in ML tasks. The goal of feature learning is often to reduce the dimensionality of the dataset, selecting or aggregating features in order to produce low-dimensional versions of the original datasets (Brown et al. 2011; Espinilla et al. 2017). Unsupervised feature selection for activity recognition has been developed by Neural Networks under black-boxes approaches, where several weighting techniques, such as, auto-encoders (Oukrich et al. 2017), back-propagation (Fang et al. 2014) and, more prominently, Convolutional Neural Networks (Singh et al. 2017) are proposed. Feature selection has been demonstrated as a suitable method in activity recognition based on wearable-devices approaches (Gupta and Dallas 2014; Ordóñez and Roggen 2016; Xu et al 2016).

Instead of reducing the number of features, in this paper we propose a methodology to generate new features by using the underlying structure of data. Although the automatic generation of features can be applied to relational datasets (Kanter and Veeramachaneni 2015), most of the current works use Open Linked Data (Ristoski 2015). A framework that generates new features for a movie recommendation dataset is proposed by Cheng et al. (2011). They use that framework to construct semantic features from YAGO, a general purpose knowledge base that was automatically constructed from Wikipedia, WordNet and other semi-structured Web sources. Then, they manually define a set of the static queries in SPARQL language, which are used to add information to the original dataset, such as, budget, release date, cast, genres, box-office information, etc. A very similar approach is proposed by Ristoski et al. (2015), where an interactive system for adding new features to a multidimensional cube is described. The new features can be selected by the user from a set of related features that are automatically generated according to the features already in the cube. Although it allows the use of custom rules in SPARQL for the definition of rules that generate features, their authors also propose some basic rules for the automatic generation of features based on RDF annotations. In spite of being proposals similar to ours, it is important to underline that the high formalization of ontologies allows us to automatically generate relevant features, without the need for human interaction.

Terziev (2016) also proposes the use of ontologies to generate new features. He expands features from the origin feature in a breadth first search manner considering the rules for semantically correct paths. Only concepts on outgoing paths from the origin entity conforming to these patterns are considered as possible features in the further process. Although they plan to test their proposal with two ontologies, they are actually dealing with the underlying RDF graph of those ontologies. They do not make use of the inference mechanisms of ontologies nor the formal logic behind them. They just use the user defined relationships between concepts in the RDF graph in order to relate the original concept with the concepts in its context.

Paulheim (2012) also proposes another technique that employs user defined relations between concepts in the RDF graph of ontologies for the automatic generation of features. Its main goal is the generation of possible interpretations for statistics using Linked Open Data. The prototype implementation can import arbitrary statistics files, and uses DBpedia for generating attributes in a fully automatic fashion. Furthermore, the author argues that their approach works with any arbitrary SPARQL endpoint providing Linked Open Data. The use of the inference mechanisms of ontologies is also very limited in this work.

Another key difference of our proposal with respect to all these works is that they propose the use of external knowledge to generate new features, whereas we only consider the information in the original dataset to do so.

In the field of ontology engineering, the algorithms for Class expression learning (CEL) have been extensively developed (Böhmann et al. 2016). CEL is the most similar technique to the approach presented in this paper. They can be used to suggest new class descriptions that are relevant

Table 2 Expansion functions

Expansion function	Expansion set
$f^{\sqcap} : C_1 \times \mathcal{P}(C_2) \rightarrow \mathcal{P}(C)$	$\{c \sqcap d \mid \langle c, d\rangle \in C_1 \otimes \mathcal{P}(C_2)\}$
$f^{\sqcup} : C_1 \times \mathcal{P}(C_2) \rightarrow \mathcal{P}(C)$	$\{c \sqcup d \mid \langle c, d\rangle \in C_1 \otimes \mathcal{P}(C_2)\}$
$f^{\neg} : C_1 \rightarrow \mathcal{P}(C)$	$\{\neg c \mid c \in C_1\}$
$f^{\exists} : C_1 \times \mathcal{P}(R) \rightarrow \mathcal{P}(C)$	$\{\exists r.c \mid \langle c, r\rangle \in C_1 \otimes \mathcal{P}(R)\}$
$f^{\forall} : C_1 \times \mathcal{P}(R) \rightarrow \mathcal{P}(C)$	$\{\forall r.c \mid \langle c, r\rangle \in C_1 \otimes \mathcal{P}(R)\}$
$f^{\leq} : C_1 \times \mathcal{P}(R) \times \mathcal{P}(\mathbb{N}) \rightarrow \mathcal{P}(C)$	$\{\leq n\ r.c \mid \langle c, r, n\rangle \in C_1 \otimes \mathcal{P}(R) \otimes \mathcal{P}(\mathbb{N})\}$
$f^{\geq} : C_1 \times \mathcal{P}(R) \times \mathcal{P}(\mathbb{N}) \rightarrow \mathcal{P}(C)$	$\{\geq n\ r.c \mid \langle c, r, n\rangle \in C_1 \otimes \mathcal{P}(R) \otimes \mathcal{P}(\mathbb{N})\}$

for the problem while the ontologies are being developed. The objective of CEL algorithms is to determine new class descriptions for concepts that may be used to classify individuals in an ontology according to some criterion. More formally, given a class C, the goal of CEL algorithms is to determine a class description A such that $A \equiv C$.

Let us suppose an ontology O with sufficient number of individuals. The set of individuals in $\Delta^{\mathcal{I}}$ is the search space S. CEL algorithms search in S, trying to find a description for class A such that $A^{\mathcal{I}}$ contains the same individuals in $C^{\mathcal{I}}$. The CEL problem may be defined as a supervised learning problem but unlike the usual supervised learning problems the number of features for each instance is not fixed. They are dynamically generated as the CEL algorithm moves along the search space S. To navigate through the space S the CEL algorithms usually apply a refinement operator to existing classes in the knowledge base (Lehmann et al. 2011).

The main difference with respect to our proposal is that the result of CEL algorithms is always a DL class expression, whereas the result of the proposed methodology is a set of DL class expressions. The class expression produced by CEL algorithms is the final model for the classification of individuals. All individuals which meet the class expression are classified as positive. The rest of them are classified as negative. The expressivity of the classifier is thus, very limited, because only DL operators can be used to describe it. This restriction is necessary when we want to incorporate this knowledge to an ontology, which is usually the main purpose of CEL-based applications. The restriction is not necessary, however, when building a more general classification model which does not have to be expressed in form of DL class expression. We can use the set of class expressions as the input of a neural network and use the individuals in the ontology to train it, for example. The obtained classification model is a neural network where the class expressions are used as its features. The knowledge in the classifier cannot be incorporated to an ontology because neural networks are black boxes. Nevertheless, it is expected to achieve a better classification performance as it has more flexibility to combine the class expressions given as input.

4 Methodology

The purpose of our methodology is to generate feature vectors by using the asserted and inferred knowledge in the ontology. In this section, we first describe how class descriptions can be combined to produce new class descriptions. Afterwards, we explain the algorithms which we use to find the most relevant class descriptions, which will be used as features for the classifiers.

4.1 Class expression expansion rules

In this section, the proposed rules for generating new Class Expressions are described. All new class expressions are developed from a given one. Depending on the DL operator chosen, the Expansion Function produces different kinds of class expressions as result.

Definition 1 Expansion function. An Expansion Function f^d generates an Expansion Set of class expressions by applying a DL operator d to a given class expression.

In Table 2, the expansion functions proposed in this work to generate the new class expressions are shown. They correspond to the basic semantics of each of the operators defined in OWL, detailed in Table 1, but other expansion functions may be used. One could define, for example, an expansion function that generates class expressions by randomly combining two other given class expressions, using the different operators of OWL as possible cut points. This expansion function would be very useful in case of using genetic algorithms, for example. In the functions: (1) C and R respectively represent the set of concepts and relations, which are defined in the ontology; (2) $\otimes$ represents the Cartesian product; and (3) $\mathcal{P}(X)$ is the Power Set of X, that is, the set of all possible subsets of X.

The given functions $f^{\sqcap}$ and $f^{\sqcup}$ apply the logical operators $\sqcap$ and $\sqcup$ to the concept being expanded c and a given set of class expressions $\mathcal{P}(C_2)$. The expansion function $f^{\neg}$ applies the complement operator to produce a unique class expression as a result, where the original class expression is complemented.

Table 3 Examples of expansion functions

	C_1	$\mathcal{P}(C_2)$ *or* $\mathcal{P}(R)$	$\mathcal{P}(\mathbb{N})$	Generated expressions
$f^{\sqcap}$	*Child*	{*Person*, ∃*hasParent*.*Person*}		*Child* ⊓ *Person*
				Child ⊓ ∃*hasParent*.*Person*
$f^{\neg}$	*Child*			¬*Child*
$f^{\forall}$	*Child* ⊔ *Parent*	{*hasParent*, *hasChild*}		∀*hasParent*.(*Child* ⊔ *Parent*)
				∀*hasChild*.(*Child* ⊔ *Parent*)
$f^{\leq}$	¬*Parent*	{*hasParent*, *hasChild*}	{2,3}	≤ 2 *hasParent*.(¬*Parent*)
				≤ 3 *hasParent*.(¬*Parent*)
				≤ 2 *hasChild*.(¬*Parent*)
				≤ 3 *hasChild*.(¬*Parent*)

In the case of quantifiers operators (∀, ∃), the expansion functions produce as many class expressions as properties are received as argument. For each of them, a new class expression of the form $o\ r.c$ is generated, where: (1) o is the given quantifier operator; (2) r represents a relation in $\mathcal{P}(R)$; and (3) c is the class expression which is being expanded.

The expansion functions that use cardinality constraints also produce class expressions by combining the original concept with all relations in $\mathcal{P}(R)$. However, these expansion functions produce as many class expressions as cardinality values $\mathcal{P}(\mathbb{N})$ are set as the function argument.

Some examples for the expansion functions defined in this section are shown in Table 3. The second and third columns indicate the set of classes $\mathcal{P}(C_2)$ or relations $\mathcal{P}(R)$ to be combined with the concept in the first column. The fourth column indicates the cardinality values to be used in the case of functions such as $f^{\leq}$ or $f^{\geq}$.

4.2 Algorithm for the generation of features

In this section, we describe the algorithm for generating class expressions that will be used as features in the classifiers. In this paper, the algorithm is described in pseudo-code, although it has been implemented using the Java language. The applications are freely available under the terms and conditions of the GNU General Public License.[1]

Algorithm 1 Expansion of features.

Require:
O is the set of DL operators that will be used to generate new class expressions.
C are the concepts in the ontology.
R are the relations in the ontology. c is the concept representing all the instances that the classifier will take as input.
n is the number of features to be generated.
d is the maximum value for the cardinality constraints.

function EXPAND(O, C, R, c, n, d)
  $C \leftarrow C \setminus \{k \in C \mid k^I = \varnothing\}$
  $F \leftarrow \varnothing$
  while $\mid F \mid < n$ **do**
    $B \leftarrow$ SELECTBESTS(C)
    $B' \leftarrow B$
    while $B \neq \varnothing \wedge \mid F \mid < n$ **do**
      $b \leftarrow \{k \mid \forall x, y \in B$ SCORE(x) < SCORE(y) $\wedge\ x \neq y\}$
      $B \leftarrow B \setminus b$
      $E \leftarrow$ EXPANDCLASSEXPRESSION(b, O, B', R, d)
      for all $e \in E$ **do**
        if SATISFIABLE(e) $\wedge\ e \notin C$ **then**
          $C \leftarrow C \cup e$
          if $e \sqsubseteq c$ **then**
            $F \leftarrow F \cup e$
  return F

[1] https://sourceforge.net/p/owlmachinelearning/.

The Algorithm 1 is described as a Step-Form algorithm. First, those concepts from the ontology which do not contain any individual instances, or cannot be identified by the reasoner, are eliminated. Next, it enters into the main loop of the algorithm, which only breaks after having generated a sufficient number of new class expressions. The desired number of class expressions is a parameter of the algorithm, which has to be specified by the user in the applications that have been developed. For each iteration of this main loop, a subset with the most relevant class expressions is created. The new class expressions will be generated by combining the expressions in this set. The current implementation of the *SelectBests* function, which is responsible for performing this task, just returns the entire set of generated class expressions, without any sort of selection. The algorithm has been designed so that the selection and the combination strategies can be modified without changing the rest of the algorithm, because we plan to evaluate different bio-inspered approaches for this strategy, such as genetic algorithms and ants' colonies.

Before proceeding to expand the selected class expressions, a copy of them is made (see line 11). This copy is necessary, since the set of selected class expressions is modified in the 14 line of the algorithm. Next, each of the selected class expressions is expanded. The order in which these class expressions are expanded is really important, since the expansion process can generate a large number of class expressions and the process is stopped when sufficient number of them are generated. The order in which class expressions are expanded depends on the *Score* function, which gives a score to each of them based on its relevance. In the current implementation of the algorithm, this score is assigned according to the depth of the tree that represents the class expression. Class expressions with smaller number of operators are prioritized, favoring the generation of simpler class expressions. This method, based on depth-first search (Hopcroft and Tarjan 1974), is widely recognized as a powerful technique for searching in graphs and trees (Even 2011). Moreover, the class expressions generated in the search describe a knowledge representation of a concept, so the pruning of solutions based on the depth of the tree is recommended to prioritize simpler and shorter expresions. This reduces the overfitting, improves the performance on test data, and increases the human interpretability (Mingers 1989). More complex evaluation functions are planned for future works, for a more accurate measure of the relevance of the class expressions. However, it is important to highlight that the efficiency of this function has a great impact on global efficiency of the algorithm, because it is called at least once for each expression.

The most relevant class expression is selected in the line 13 of the algorithm and removed from the set of selected classes. The *ExpandClassExpression* function is responsible for generating all possible class expressions that are formed from the currently selected class expression. Again, the algorithm has been carefully designed to make this function easily modifiable, so we can test other expansion strategies without having to rewrite the entire algorithm. This function can generate unsatisfiable class expressions, that is, it can generate expressions with restrictions impossible to meet for any individual in the ontology. For example, given the concepts *Cat* and *Dog*, which are defined in a hypothetical ontology as *disjoint* concept, an example of unsatisfiable expression is $Cat \sqcap Dog$, because no individual instance in the ontology can be represented by such expression. Only satisfiable class expressions are added to the ontology (see line 18), previously checking that they have not been added before.

Finally, only those of the generated class expressions that successfully describe a type of the concept to be recognized are added to the set of features for the classifiers (see line 20). Following the previous example, in the case we are trying to recognize instances of animals, we only add to the feature set those class expressions that describe some king of animal, such as *eats some Thing*, avoiding individual instances that describe other types of concepts, such as *Vehicle* or *hasIngredient some Vegetable*. We have to realize that although these expressions are not included in the final set of features for the classifiers, they are added to the set of class expressions C, since they can result in relevant class expressions in combination with others.

Algorithm 2 Generation of derived class expressions.

Require:
1: c is the concept being expanded.
2: O is the set of DL operators that will be used to generate new class expressions.
3: C are the concepts that may be combined with c to generate new class expressions.
4: R are the relations in the ontology.
5: n is the maximum value for the cardinality constraints.

```
6:  function ExpandClassExpression(c, O, C, R, n)
7:      E ← ∅
8:      for all o ∈ O do
9:          if o ∈ {¬} then
10:             E ← E ∪ ¬c
11:         if o ∈ {∀, ∃} then
12:             for all r ∈ R do
13:                 E ← E ∪ o r.c
14:         if o ∈ {⊔, ⊓} then
15:             for all k ∈ C do
16:                 if c ≠ k then
17:                     E ← E ∪ k o c
18:         if o ∈ {≤, ≥} then
19:             for all i ∈ {1..n} do
20:                 for all r ∈ R do
21:                     if SimpleProperty(r) then
22:                         E ← E ∪ o i r.c
        return E
```

Algorithm 2 describes an implementation of the *ExpandClassExpression* function, which is responsible for producing new class expressions based on a given one. For each of the operators received as arguments, the algorithm applies the expansion rules detailed in Table 2. The complement operator produces a single class expression, where the original expression is complemented. Given the class expression *hasChild some Man*, for example, the line 10 of the algorithm produces the expression *not*(*hasChild some Man*).

In the case of quantifier operators, the algorithm generates as many class expressions as properties are defined in the ontology. For each of them, an expression of the form *o r.c* is generated, where: (1) *o* is the given quantifier operator; (2) *r* represents a property defined in the ontology; and (3) *c* is the class expression being expanded. For the case of the previous example and the existential quantifier, the statement in the line 13 of the algorithm generates expressions such as *hasChild some*(*hasChild some Man*), *hasParent some*(*hasChild some Man*), *hasIgredient some*(*hasChild some Man*), etc.

Logical operators construct class expressions by combining the concept to be expanded with the other selected concepts. Since $C \sqcup C \equiv C$ and $C \sqcap C \equiv C$, it only makes sense to combine different concepts with logical operators. The statement in the line of the algorithm produces class expressions such as *Woman* $\sqcup$ *Man* or *Man* $\sqcap$ (*hasChild some Woman*).

The number of class expressions being generated by cardinality constraints is virtually infinite, so the user must determine the specific cardinality values to limit the generation of them. For each possible value of cardinality and properties in the ontology, the algorithm generates a new class expression. The algorithm generates in this case expressions of the form *hasChild min* 3 *Woman* or *hasIngredient max* 2 (*hasParent some Vegetable*), for example. However, since only simple properties can be used in cardinality constraints (Motik et al. 2012), only the expressions that are formed by simple properties are actually considered.

Many of the class expressions generated by the last algorithm do not make much sense, as it has been described in the examples in this section. For that, the algorithm checks in the line 17 if the generated class expression is satisfiable before adding it to the set of generated expressions. This substantially reduces the search space for the new features, since the expansion of class expressions stops when they are not satisfiable. The reasoning mechanisms represent in this case a clear advantage against a simple brute-force search.

Finally, it is noteworthy that the proposed methodology helps to improve the accuracy of classifiers in those situations where there is no clear idea about the relevant features for the problem or when the feature search space is so large that could not be generated manually. Obviously, the information in the dataset must have some kind of structure and be able to be expressed as an ontology. The methodology can easily be applied to problems related to natural language

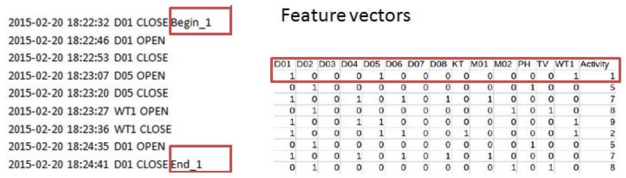


Fig. 1 Partial sensor data stream of a dataset and its computed feature vector

processing, for example. In a previous work we have applied CEL-based algorithms for the sentiment analysis of text documents (Salguero and Espinilla 2017). In that case, the ontology was formed by entities such as "Word", "Sentence" or "Adjective". The text document can be seen as a sequence of ordered words, just as it happens with the events generated by the sensors in ADL.

5 Experiment

In order to evaluate the quality of the methodology proposed in this work an experiment has been carried out, in which the dataset proposed in Ordónez et al. (2013) has been used. The objective of the experiment is to determine whether or not a particular ADL has been performed based on the sensors that have been fired during a specific period of time.

In this section we first describe the dataset that has been used in the experiment and how the sensor data stream has been transformed into feature vectors. Then, the ontology derived from the dataset, which is needed to apply our methodology, is also described. To conclude this section, the specific tasks that have been performed in the experiment are detailed.

5.1 From sensor data stream to classic feature vectors. Smart environment datasets

In this paper, the activity recognition dataset of smart environments developed by Ordónez et al. (2013) in the UC Irvine Machine Learning Repository, is used to evaluate our proposal. The dataset represents two participants performing ten ADL activities in their own homes. The activities were performed individually and this dataset is composed by two instances of data, each one corresponding to a different user and summing up to 35 days. Ten activities are classified: breakfast, dinner, leaving, lunch, showering, sleeping, snack, spare time TV and grooming. In this dataset, the number of sensors is 12, although two of them are never fired in the case of the second participant. In fact, the dataset can be actually considered as two different datasets. We decided to use the second set of activities because classifiers based on the classic approach produce perfect classifiers for most of the activities in the first dataset, so there is no room for improvement.

Usually, feature vectors generated by a smart environment are computed from the temporal sensor data stream that is discretized into a set of time windows, denoting each time window by W^k, which is limited by each activity. The set of activities are denoted by $A = \{a_1, \ldots, a_i, \ldots, a_{AN}\}$, being AN the number of activities of the dataset.

Each feature vector is denoted by F^k and has $N_S + 1$ components, being N_S the number of sensors in the dataset denoted by $S = \{s_1, \ldots, s_j, \ldots, s_{N_S}\}$. Therefore, each computed feature vector is defined by the following equation:

$$F^k = \{f_1^k, \ldots, f_j^k, \ldots, f_{N_S}^k, f_{N_S+1}^k\}$$

being $f_j^k; j = \{1, \ldots, N_S\}$ a binary value that indicates if the sensor s_j was fired at least once, 1, or was not fired 0 in this time window W^k. The last component $f_{N_S+1}^k \in A$ indicates the activity carried out in the time window W^k. An example of this process is shown in Fig. 1, which can be find in Quesada et al. (2015).

The classic feature vectors can be expanded through ontologies to improve the accuracy of the results. The ontology that has been specifically developed for this task is reviewed in the following subsection.

5.2 An ontology for the description of ADL

This section presents a brief review of the ontology developed for the description of ADL. The aim of this ontology is to provide a basic set of primitives that allow the representation of the information present in the dataset. The set of primitives should be comprehensive enough to be able to represent all the activities but should also be as brief as possible to facilitate the use of reasoners.

The ontology defines two basic disjoint concepts, *Activity* and *Event*, which respectively represent all the activities in the dataset and the activation of the sensors during these activities. In fact, the *Event* concept represents any situation reported by a sensor. It can also be used to represent the deactivation of a sensor. Each of the sensors in the dataset requires the creation of at least one subconcept of the *Event* concept to represent the events produced by that sensor. The application developed to convert the dataset into an ontology creates two subconcepts for all sensors. One of them represents the activation of the sensor and the other, which is optional, its deactivation. In the first case the suffix "_set" is appended to the end of the name of the concept, while the suffix "_clear" is appended in the second. The class $Maindoor\_set \sqsubseteq Event$, for example, represents the set of events corresponding to the activation of the sensor in the main door. The ontology also includes a property

to associate literal values to sensor events (*hasValue* datatype property). This property can be used to annotate all the changes in a sensor along time. However, this is often a bad idea for a sensor that is continually changing, since this would create a lot of individuals in the ontology and decrease the performance of reasoners. It is best to define some intervals ("Low", "Medium" and "High", for example) and register only the significant changes.

Only for this particular dataset, the type and location of the sensor has been added to the concept name because some of the sensors share the same name. For example, the class that represents the activation of the sensor of the main door is actually defined in the given ontology as $Maindoor\text{–}Magnetic\text{–}Entrance\_set \sqsubseteq Event$, although in the rest of examples of the work we will just use the name of the sensor for the sake of clarity. Actually, only the sensors of the three interior doors share the same name in the dataset. In any case, the information about the type and location of sensors is solely used to distinguish a sensor from another in the experiment.

Our proposal for representing activities is based on a list structure (see Fig. 2). *hasNext* is defined as a functional, asymmetric and irreflexive property, establishing the order of events in the activity. Because it has been defined as a functional property just one event could follow another event. The inverse property is also defined as functional, forcing an event to be directly preceded by a unique event. We defined a transitive property *isFollowedBy* as a superproperty of *hasNext*. Since this means that *hasNext* implies *isFollowedBy*, any sequence of entities linked by *hasNext* will be inferred to be a chain linked by *isFollowedBy*. *hasNext* is used to express that an event *B* immediately follows another event *A*. There is no other event between them. So event *A* has *B* as the next event in the list (*A hasNext B*) or, in other words, event *A* is followed by event *B* (*A isFollowedBy B*). If another event *C* appears after event *B*, event *A* is also followed by event *C* (*A isFollowedBy C*), but event *A* has not *C* as the next event on the list (*not A hasNext C*).

The property *hasItem* establishes the membership of an event to the list. The class description *hasItem some (Frontdoor_set and isFollowedBy some Dishwasher_set)* is a way of describing the activity *#Activity24* of the example in Fig. 2.

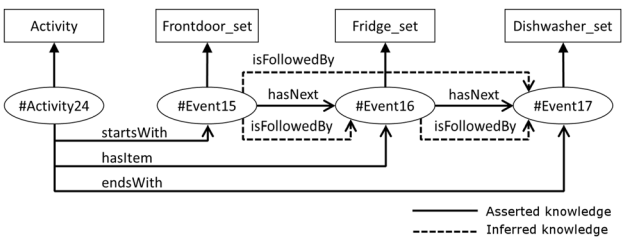


Fig. 2 Ontology example

The properties *startsWith* and *endsWith* are used to identify the first and last events in the activities. Due to open world assumption in OWL, reasoners cannot automatically infer the individuals that belong to these concepts. Therefore, it is necessary to annotate these individuals when the activities are converted to the model proposed in this paper. The class description *startsWith some (Frondoor_set and hasNext some (Fridge_set and hasNext some Dishwasher_set))*, for example, represents the activities that begin with the activation of the sensor of the front door, which is immediately followed by the activation of the fridge sensor and then by the dishwasher sensor, immediately. The activity *#Activity24* in Fig. 2 is an example of activity described by the above class description.

Figure 3 shows how the sensor data stream in Fig. 5.1 can be expressed in form of ontology by using all the properties defined above.

In OWL the same individual could be referred to in many different ways (i.e. with different URI references). Due to this, it is necessary to state that all the elements in the datasets are different individuals. For practical reasons, a functional property *hasID* is used to identify all of the individuals in the model with an unique code. In this way, the addition of new entities to the ontology is easier, without the need of asserting that all of them are different from the existing individuals.

$$\top \sqsubseteq \forall\ hasID\ Datatype\#long$$

When we want to refer to events that occur before another one we can make use of inverse properties, which have not been explicitly defined for efficiency reasons. *#Activity24* may also be described by the class description $endsWith\ hasNext^{-}\ Fridge\_set$, which describes activities ending with an event preceded by an activation of the fridge sensor.

There are others ADL ontologies available in the literature (Riboni and Bettini 2011; Chen and Nugent 2009b; Bae 2014; Okeyo et al. 2014; Villalonga et al. 2016). However, most of them describe activities from a high level of abstraction. Few of them describe activities as sequences of events produced by sensors. The ontology proposed by Noor et al. (2018) is the most similar to the ontology described in this section and it is one of the few that is available for download. To show that the methodology proposed in this work is independent of the ontology being used, part of the experiment consisted on describing the dataset by using the ontology proposed by Noor and comparing the results obtained by the ontology described in this section.

5.3 Experiment design

The objective of the experiment is to determine whether or not a particular ADL has been performed by a single person based on the sensors that have been fired during a

Fig. 3 Relations for entities in Fig. 1

Property:*startsWith*	
Domain:Activity	Range:Event
#Activity#1	D01

Property:*endsWith*	
Domain:Activity	Range:Event
#Activity#1	D01

Property:*hashItem*	
Domain:Activity	Range:Event
#Activity#1	D01
#Activity#1	D05
#Activity#1	WT1

Property:*hashNext*		
Domain:Activity	Domain:Event	Range:Event
#Activity#1	D01	D05
#Activity#1	D05	WT1
#Activity#1	WT1	D01

Property:*isFollowedBy*		
Domain:Activity	Domain:Event	Range:Event
#Activity#1	D01	D05
#Activity#1	D05	WT1
#Activity#1	WT1	D01
#Activity#1	D01	WT1
#Activity#1	D01	D01
#Activity#1	D05	D01

Table 4 Activities in the dataset

Activity	Instances
Breakfast	22
Dinner	11
Grooming	113
Leaving	38
Lunch	13
Showering	11
Sleeping	29
Snack	47
Spare time TV	116
Toileting	93
Total	493

Table 5 Activity recognition accuracy for classic approach

Activity	C4.5	SMO	VP	DT
Breakfast	94.87	95.00	**95.34**	94.73
Dinner	97.77	97.77	97.77	97.77
Grooming	95.05	**95.39**	94.11	95.12
Leaving	99.19	99.53	99.12	98.92
Lunch	97.37	97.37	97.30	97.17
Showering	100.00	100.00	99.53	100.00
Sleeping	99.39	99.39	98.45	99.39
Snack	**91.96**	90.13	90.20	90.61
Spare time TV	95.74	95.74	95.53	95.74
Toileting	98.39	98.39	98.39	98.39

Best accuracies of selected activities are in bold

specific period of time. To simplify the experiment, the time intervals always correspond to the labeled activities in the dataset.

The number of instances per activity is shown in Table 4. All the instances in the dataset have been taken as the input for all the classifiers. The instances corresponding to the activity being recognized are treated as positive individuals whereas the rest of the instances are treated as negative individuals. 10-fold cross-validations have been used to evaluate all the classifiers. The main advantage of this validation is that all the activities in the dataset are used for training and testing, avoiding the problem of considering how the dataset is divided.

For the development of the classifiers in the experiment, we have made use of six algorithms of the Weka data mining software (Witten et al. 2016). We have tried to select the most commons algorithms from most of the categories in Weka ("bayes", "funcitons", "rules" and "trees"). We discarded some algorithms, such as the Multilayer Perceptron, because they take much more time than the others to compute. The algorithms have always been run with defaults parameters.

- The Naive Bayes (NB) algorithm is a probabilistic induction algorithm that is based on the classic Bayesian classifier. It uses statistical methods for nonparametric density estimation for each predictive attribute instead of using a single Gaussian distribution, as Bayesian classifiers usually do.
- The C4.5[2] is an algorithm used to generate decision trees. It is an extension of the basic ID3 algorithm that try to address some of its issues, such as the missing data, the handling of continuous attributes or the overfitting.
- Sequential Minimal Optimization (SMO) is an iterative algorithm for the training of Support Vector Machines (SVM). It requires much less time than all the previous

[2] The Weka implementation of the C4.5 classifier is called J48.

Table 6 Subsets of DL operators used in the experiment

Operators set name	Intersection	Union	Complement	Existential quantifier	Universal quantifier	Maximum cardinality	Minimum cardinality
$\mathcal{CSM}$			✓	✓			✓
$\mathcal{IUCSAXM}$	✓	✓	✓	✓	✓	✓	✓
$\mathcal{IUCSM}$	✓	✓	✓	✓			✓
$\mathcal{S}$				✓			
$\mathcal{SCI}$	✓		✓	✓			
$\mathcal{SU}$		✓		✓			

available methods. Given a p-dimensional vector, where p is the number of features, SVM try to find the hyperplane that represents the largest separation between two given classes.

- The Voted Perceptron (VP) algorithm is an improved version of the perceptron algorithm. As well as SVM, the VP algorithm uses a kernel function to separate data. However, it is considered a simpler method to implement, and much more efficient in terms of computation time.
- The Decision Table (DT) algorithm builds decision tables with a default rule mapping to the majority class. Given an unlabeled instance, the classifier seeks for similar labelled instances. If no instances are found, the majority class of the DT is returned. Otherwise, the majority class of the dataset is returned.
- The PART algorithm can be used to generate decision list. It builds a partial C4.5 decision tree in each iteration and makes the "best" leaf into a rule.

The results obtained by these classifiers when using a classic DDA approach to solve the problem have been taken as reference to measure the efficiency of our proposal. For this purpose an application that identifies the sensors that have been fired during each of the activities has been built. The application generates a file in Weka format, following the structure presented in Section 5.1. This file contains an instance for each activity and as many features as sensors in the dataset. All the features are binary and specify if the sensor has been fired during the activity or not. Finally, it includes a class attribute, also binary, that indicates if it is the activity that the classifier is learning to identify (positive) or not (negative). Each experiment consists, therefore, in determining which combination of sensors are fired for a particular activity, such as "Breakfast", for example.

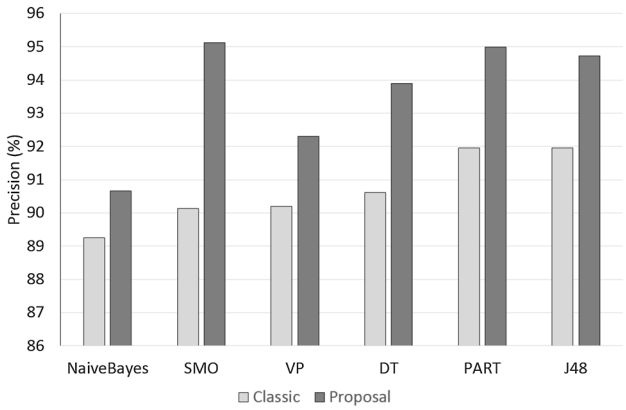
By using the Weka data mining software, we first have generated C4.5, SMO, VP and DT classifiers for all the activities in the datasets. A summary of the results obtained for all the activities of the different datasets is shown in Table 5. The three activities that are more difficult to be recognized have been taken as reference, so there is more room for improvement. More precisely, the activities chosen for the experiment have been "Breakfast", "Grooming" and "Snack", with best classification accuracies of 95.34, 95.39 and 91.96%, respectively.

An ontology has been automatically generated for describing the activities in the dataset by using the primitives presented in Sect. 5.2. Then, the Algorithm 1 has been used to generate different sets of new class descriptions, which are then used as new features by the classifiers. Five different subsets of DL operators have been used to generate five different sets of class descriptions. The specific DL operators for each subset are shown in Table 6. To relate activities to events, it is necessary to include one of the quantifiers, at least. The universal quantifier is very restrictive, because all related individuals have to meet the conditions. We have only include it in one of the sets for this reason. Cardinality restriction operators are expensive for by the reasoner, so they have only been included in some of the sets. The maximum cardinality operator is especially expensive. The intersection, union and complement operators cannot form class expressions that relate activities and sensors on their own, so they have been combined with the existential quantifier. Versions with $n = \{10, 20, \ldots, 90, 100, 200, \ldots, 1000\}$ class expressions have been generated for each of these sets of DL operators.

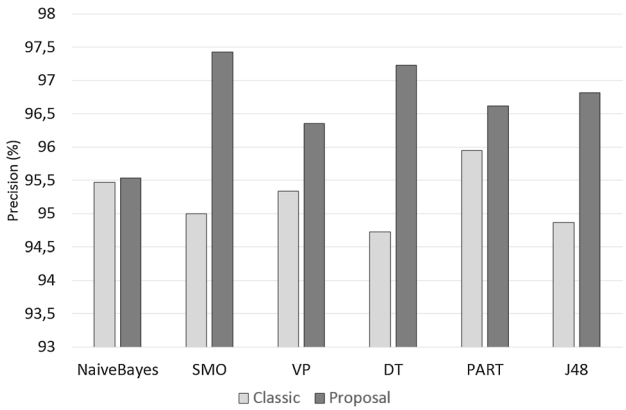
All the class descriptions are evaluated by another application and new files in Weka format are generated for each of the versions of the different subsets of DL operators. We get a file for the version with $n = 10$ generated class expressions for the $\mathcal{CSM}$ set, another file for the version with $n = 20$ class expressions and so on. Each class expression corresponds to a new feature for the classifier.

The results obtained through the methodology proposed in this work have also been compared with the results obtained with the application DL-Learner,[3] which implements several algorithms for CEL. This application just needs the ontology and the sets of individual instances representing the positive and negative individuals. This information was generated when the original dataset was translated into an ontology. In this case, the concept to be

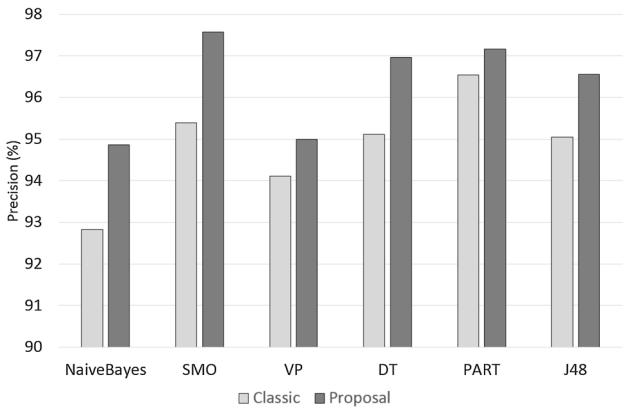
[3] http://dl-learner.org.



(a) Snack



(b) Breakfast



(c) Grooming

Fig. 4 Classifiers performance

recognized is a given activity and the result of the process is a class expression, which is composed by the instances of this activity, but not the instances of other activities. The DL-Learner application has been running for one hour for each of the three activities in search of the class expression that best matches them.

6 Results

The accuracies obtained by all the different classifiers for the selected activities are analyzed in this section. As can be seen in Fig. 4, the classifiers based on our approach clearly improve the results obtained by those using the traditional approach. This is true for the three activities analyzed and for all supervised learning algorithms used to construct the classifiers. A detailed analysis of the results shows that the improvement obtained for the "Snack" activity is the biggest. The best accuracy obtained using the classic approach is 91.96%, corresponding to the PART algorithm, while the algorithm SMO gets an accuracy of 95.12% when the methodology presented in this paper is applied. This is an increase of 3.16%, being 8.04% the best possible improvement. Improvements are also obtained for the two other activities, but the difference is not so evident. This mainly is because the algorithms using the traditional approach obtain very high accuracies (95.95 and 96.54% for the "Breakfast" and "Grooming" activities, respectively).

Regarding the learning algorithms used in the experiment, we can observe that *SMO* provides the best performance in the three cases, obtaining the best absolute results for the three activities. In addition, it is also the learning algorithm with the greater differences between the proposed and the traditional approaches. The algorithms that get worse results are Naive Bayes and VP, but they always improve the results of the traditional approach.

Table 7 shows the most relevant class descriptions found for the "Grooming" activity by the SMO algorithm. These class expressions have been found after generating six hundreds of them by using the $\mathcal{SCI}$ operator set. This is the class expressions set that results in better prediction accuracy, as shown in Fig. 6. The first column indicates the weight assigned to each feature by the algorithm. Each of the six hundred class expressions have an associated weight, but only the expressions with higher absolute values have been included in Table 7. As can be seen on the table, there are four main class expressions that can be used to identify the "Grooming" activity. All these expressions describe activities in which the sensor of the basin in the bathroom has been activated, which makes sense. The second expression does not provide more relevant information, because all activities have to start with some sensor event. It describes the same activities as the previous class description does. On the contrary, the third and the fourth class expressions does provide more relevant information. They describe class expressions in which the basin sensor has been activated but the sensors located at the bed and the door of the bedroom have not, respectively. This makes sense because the "Sleeping" activity often includes the activation of that sensors.

Table 7 Most relevant class expressions found for the activity "Grooming" by the SMO algorithm

Weight	Expression
0.4036	hasItem some Basin_set
0.4036	(hasItem some Basin_set) and (startsWith some Event)
0.4036	(not (Door-Bedroom_set)) and (hasItem some Basin_set)
0.4036	(not (Bed_set)) and (hasItem some Basin_set)
−0.4999	hasItem some (hasNext some Event)
−0.4999	startsWith some (isFollowedBy some Event)
−0.4999	startsWith some (hasNext some Event)
−0.4686	(hasItem some Toilet_set) and (startsWith some Door-Bedroom_set)
−0.9349	(hasItem some Door-Bedroom_set) and (hasItem some Toilet_set)

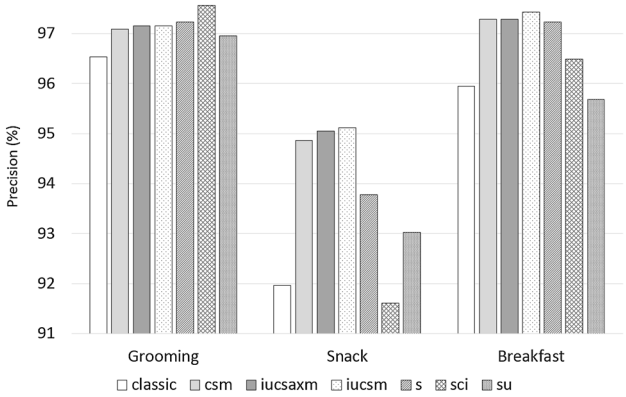


Fig. 5 DL operators performance

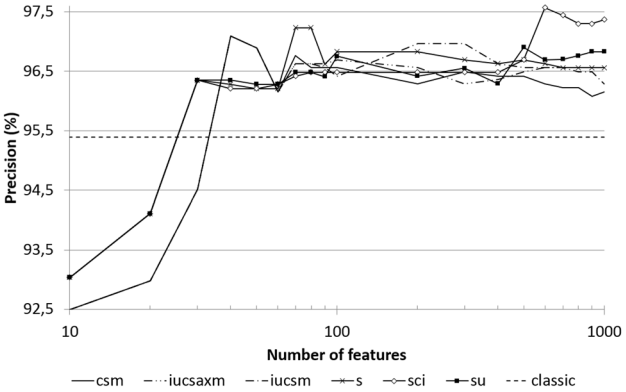


Fig. 6 SMO performance for "Grooming"

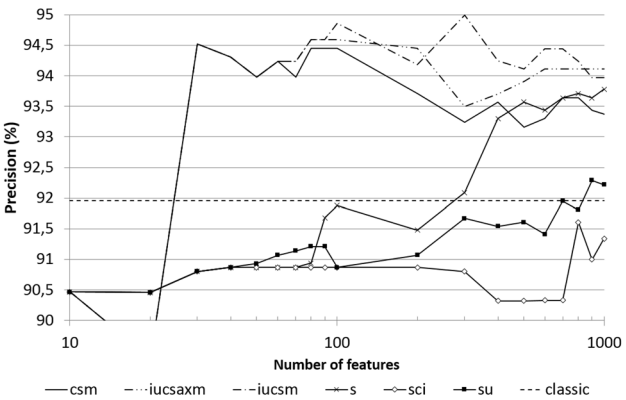


Fig. 7 PART performance for "Snack"

There are also some class expressions that can be used to discard negative instances, that is, other activities than "Grooming". In this example, activities in which both the sensor of the door in the bedroom and the sensor in the toilet have been activated have many possibilities to not be classified as "Grooming" activities. These kind of activities are described by the last two class descriptions in Table 7. The rest of the class expressions can also be used to identify negative instances, but their meaning are not so obvious. Class descriptions in fifth, sixth and seventh rows describe activities having two or more sensors activations. This also makes sense because there are many instances of the "Grooming" activity in the dataset in which the sensor of the basin is the unique sensor that has been activated, and just once for the entire activity. Therefore, an activity having multiple activations of sensors is less likely to be classified as a "Grooming" activity.

As can be seen, our proposal also takes into account the class expressions that serve to identify the negative instances. This is the main reason why our proposal achieves better results than CEL-based applications. However, and depending on the supervised learning algorithm used, with the proposal presented in this paper it will not always be possible to extract knowledge from the class expressions generated. In the case of using a black box algorithm, such as VP, it is not possible to determine the relevance of the different class expressions or their structures.

With respect to the set of DL operators that are used to generate the class expressions, there is not a clearly winner, as shown in Fig. 5. In fact, the results obtained are slightly different for each of the activities being analyzed. When the number of operators used to generate the set of class expressions is high the results are better for the activities "Snack"

and "Breakfast". For those activities, the best performance is obtained for the $\mathcal{CSM}$, $\mathcal{IUCSAXM}$ and $\mathcal{IUCSM}$ sets of operators. Good results are also obtained for the activity "Breakfast" when the set that only uses the existential quantifier to generate the class expressions is used. On the contrary, the set of operators $\mathcal{SCI}$ obtains the best results for the activity "Grooming", although the other sets of operators also obtain very good results. It should also be outlined that the improvement over the classical approach is the lower among the three activities, probably due to the inherent high precision obtained using this approach. Therefore, there is not an optimal set of operators to generate the class expressions, although the sets $\mathcal{CSM}$, $\mathcal{IUCSAXM}$ and $\mathcal{IUCSM}$ provide good results in all three cases.

We also have to realize that not all class expressions require the same amount of time to be evaluated by reasoners. The class expressions containing cardinality constraints are the ones that require most computing time, followed by the expressions containing universal quantifiers. For example, the evaluation[4] of the set with thousand classes generated with the set of operators $\mathcal{CSM}$ requires about one hour of processing in one of the nodes of the computer cluster of the University of Cadiz[5]. This is because it is the set of DL operators that generates most class expressions containing cardinality constraints. In contrast, the version with thousand classes generated with only the existential operator just requires ten seconds to be completely evaluated with the same system configuration. This makes the existential quantifier operator a very interesting option to consider, because the results obtained with just this operator are relatively good, specially in the activities "Grooming" and "Breakfast".

The developed application is actually composed by two separate applications. The first one is responsible for generating the number of class expressions specified by the user. We call this the *expansion* process, and it has to be executed only once for each DL operator set. The other application processes the list of generated class expressions to determine which of the activities in the ontology are described by each of those class expressions. We call this the *vectorization* process because it produces the set of feature vectors. Because the latter may be a high time-consuming task, we have used a multi-threaded approach. The list of class expressions is divided in blocks and they are processed in parallel by each core of the processor. The vectorization process has to be also applied to any other forthcoming activity that has to be classified according to the final classifier generated, so it has to be as much as efficient as possible.

[4] The application developed makes use of the HermiT OWL Reasoner (http://www.hermit-reasoner.com).

[5] 2×Intel Xeon E5 2670, 2.6 GHz with 128 GB of RAM.

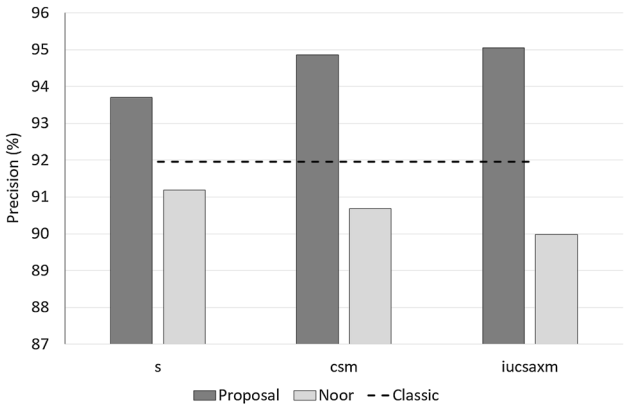


Fig. 8 Performance comparison with Noor et al. (2018) for "Snack"

The number of characteristics used in the classification clearly influences the accuracy obtained by the different learning algorithms. In Fig. 6, we show a very significant example of the behavior of these algorithms where the number of features generated vary from ten to one thousand. As can be observed, when the number of features generated is low, the results are worse than those obtained by the classic approach. This is because the class expressions generated do not provide enough information to the classifier. As the number of features available for the classifier increases, the result improves significantly. However, the results do not improve significantly from a given number of features. The results may even worsen if the number of generated class expressions is excessive, as can be observed in Fig. 7. This is due to the inclusion of irrelevant, redundant and noisy features, which result in a poorer predictive performance (Hall and Holmes 2003).

In Fig. 7, we highlight a behavior that is repeated with some frequency in the different experiments of this work: the results improve faster in the case of the sets of operators that include cardinality restrictions, while in the other sets of operators more class expressions are needed to achieve the same results.

In view of the results obtained, an adequate strategy for using the methodology proposed in this paper would be to only use the existential quantifier for the generation of class expressions at the beginning. This operator produces class expressions that can be evaluated very quickly by the reasoner, which allows us to perform a first exploratory analysis and validate the design of the ontology. Only from then on is it convenient to test the rest of the DL operators for generating class expressions. The complement, union and intersection operators should be the next to be tested, since the class expressions they generate are also relatively quick to evaluate. The universal quantifier and the maximum cardinality operator are the last ones that should be added to the analysis, since the time required to evaluate the class

expressions generated with them requires an extremely high amount of time compared to the former.

As we discussed in Sect. 5.2, we have also carried out an experiment to evaluate the results obtained by using an ontology different from the one proposed in this paper. The applications developed have been modified to generate the ontologies following the scheme proposed by Noor. In this scheme, the activities are composed of smaller time intervals. These time intervals are those which are really associated with the events generated by the sensors, in such a way that we can characterize an interval according to the sensors that are activated during it. The activities are then defined as sequences of intervals in which certain sensors are fired. More than forty thousand intervals of thirty seconds lengths are required to represent all the activities in the dataset, which makes it impossible for a reasoner to handle them. Instead, we divided each instance of each activity into three different intervals, which produces a reasonable number of individuals in the ontology.

The results obtained for the "Snack" activity, using the ontology proposed by Noor, are shown in Fig. 8. The experiment has been carried out with three subsets of DL operators and four supervised learning algorithms (C4.5, SMO, Decision Table and Voted Perceptron). As can be seen, excellent results have been also obtained when applying our methodology using Noor's Ontology. However, they are far from the results obtained when using our proposed ontology, since it has been defined *ad hoc* for machine learning purposes.

Moreover, the main lack of Noor ontology is related to its low efficiency. Only one hundred features have been generated for each set of DL operators. In the case of the set $\mathcal{CSM}$ it was only possible to generate eighty features. The version with ninety features was rejected after ten hours of execution. Only one hour is necessary to generate the same number of features with the ontology proposed in this work. Seven minutes are needed to generate ten class expressions when only the existential quantifier is used, while ten seconds are required to generate a hundred of class expressions with the ontology we propose.

The prediction accuracies reported by the DL-Learner application, have been 95.74, 97.57 and 95.54% for the "Snack", "Breakfast" and "Grooming" activities, respectively. Apparently, the CEL approach produces slightly better results than the one proposed in this paper for two of the three activities. However, it should be noted that DL-Learner makes use of its own approximate incomplete reasoning procedure for Fast Instance Checks which partially follows a closed world assumption. This means that the results produced by the DL-Learner application are not the same as those provided by reasoners that complies with the OWL standard, so the produced class expressions only have sense in the context of the DL-Learner application. As an example, the following class expression is the one that DL-Learner found to be the best description for the "Breakfast" activity.

hasItem min 4 (*Fridge_set or Microwave_set or* (*Door_Kitchen_set and* (*isFollowedBy only* (*not* (*Door_Living_set*)))))

However, when the expression is evaluated by the HermiT reasoner, only four instances are found for the activity, being two of them incorrectly classified.

7 Conclusions and future works

In this work, an ontology-based methodology to improve the accuracy of supervised learning algorithms has been proposed.

To do so, the feature vectors for the datasets are extended with asserted and inferred knowledge from the ontology that describes the dataset itself. An evaluation in the field of sensor-based activity recognition with Data-Driven Approaches has been carried out with the following six popular classifiers: C4.5, Sequential Minimal Optimization, Voted Perceptron, Naive Bayes, PART and Decision Table. Results from the evaluation demonstrated the ability of the ontology to extend the feature vectors.

More precisely, our approach has achieved an improvement of 3.16, 1.48 and 1.03% for the "Snack", "Breakfast" and "Grooming" activities of the dataset (Ordónez et al. 2013), respectively. Despite not being very high values, we consider that this is a significant improvement because the classic approach achieves high performance by itself. Following a classic approach, where each feature represents the activation of the sensors, maximum precisions of 91.96, 95.95 and 96.54% are obtained, respectively, so the maximum possible improvement are 8.04, 4.05 and 3.56%. In addition, these improvements have been obtained without considering any other additional information than the available in the dataset.

The results show that the best absolute results are achieved by the algorithm based on Sequential Minimal Optimization, obtaining this algorithm the most evident improvements with respect to the classic approach. On the other hand, we can also conclude that the algorithms Naive Bayes and Voted Perceptron provide worse results.

Regarding the set of DL operators used to generate the expressions, which configure the features in classifiers, we cannot conclude that there are significant differences among them. The set that uses the universal quantifier, the complement and the intersection of operators obtained worse results for the "Snack" and "Breakfast" activities. However, it achieved the best performance for the "Grooming" activity. In general, we can conclude that the sets of operators that include cardinality restrictions obtain better results and with less quantity of generated class expressions. Nevertheless, when this type of operators are used, the necessary time to evaluate the set of class expressions is very high. The set of expressions generated just using the existential quantifier requires much less time and the obtained results are good in general.

The main problems with the methodology often come from an inadequate design of the ontologies. Usually, they contain a lot of concepts and properties, with the aim of increasing their expressiveness. However, this causes several problems in the methodology proposed in this work. On the one hand, the performance of the reasoners worsens as the complexity of the ontology increases. On the other hand, it makes necessary to generate a greater number of class descriptions to obtain some that are relevant, since the number of class expressions in the search space grows exponentially for each concept or property that are included in the ontology. It is also important to keep in mind that reasoners usually work with an open-world assumption. It is necessary to give them the tools so that they can demonstrate, under this assumption, some axioms that people habitually give for certain, especially when working with the complement operator or the universal quantifier.

However, there is still room for the improvement in the proposal presented in this paper. On the one hand, the number of features to consider grows exponentially with every expansion process. This degrades the performance of the methodology. To overcome this situation, an heuristics that can restrict the number of features generated may be useful. This is the solution proposed by Terziev (2016), in which we are working to adapt to our proposal. On the other hand, we are currently using a naive approach for selecting the class expressions to be expanded. We are just selecting the less complex expressions. We consider that a more elaborate strategy for selecting and combining class expressions would improve the efficiency of the methodology. We plan to implement some bio-inspired approaches for this, such as genetic algorithms or ants' colonies.

Many of the related works employ external knowledge in order to generate new features. However, only the information presented in the dataset has been taken in consideration in this work. Thanks to the modular design of OWL, which greatly facilitates the interconnection among ontologies, it is not difficult to integrate information coming from external data sources into the dataset, once expressed in form of ontology. Our future work is also addressed to develop an ontology that describes the environments of the ADL experiments and helps us to interconnect the datasets with general purpose knowledge bases.

Finally, it is important to realize that the methodology proposed in this work can be used in many other domains. Although the experiment has been focused on the recognition of ADL, the proposed methodology and the applications developed can be used in many other areas without modifications. In this case, the features represent different combinations of terms or grammatical structures. We are currently applying our proposal in a research project funded by the Spanish Government which tries to predict the health problems in the labor of pregnant women, based on the changes in the values of dozen of biomarkers measured during their pregnancy. Using white-box supervised learning algorithms, we intend to find patterns in biomarkers or relations among them to predict problems in childbirth.

Acknowledgements This project has received partial support from the REMIND Project from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement no 734355 as well as from the Spanish government by research project TIN2015-66524-P.

References

Alemdar H, Ersoy C (2017) Multi-resident activity tracking and recognition in smart environments. J Ambient Intell Hum Comput 8(4):513–529. https://doi.org/10.1007/s12652-016-0440-x

Bae IH (2014) An ontology-based approach to adl recognition in smart homes. Future Gener Comput Syst 33:32–41

Bengio Y, Courville A, Vincent P (2013) Representation learning: a review and new perspectives. IEEE Trans Pattern Anal Mach Intell 35(8):1798–1828

Böhmann L, Lehmann J, Westphal P (2016) Dl-learner-a framework for inductive learning on the semantic web. Web Semant Sci Serv Agents World Wide Web 39(Supplement C):15–24. https://doi.org/10.1016/j.websem.2016.06.001

Brown M, Hua G, Winder S (2011) Discriminative learning of local image descriptors. IEEE Trans Pattern Anal Mach Intell 33(1):43–57

Chandrasekaran B, Josephson J, Benjamins V (1999) What are ontologies, and why do we need them? IEEE Intell Syst Appl 14(1):20–26

Chen L, Nugent C (2009a) Ontology-based activity recognition in intelligent pervasive environments. Int J Web Inf Syst 5(4):410–430

Chen L, Nugent C (2009b) Ontology-based activity recognition in intelligent pervasive environments. Int J Web Inf Syst 5(4):410–430

Chen L, Nugent C, Okeyo G (2014) An ontology-based hybrid approach to activity modeling for smart homes. IEEE Trans Hum Mach Syst 44(1):92–105. https://doi.org/10.1109/THMS.2013.2293714

Cheng W, Kasneci G, Graepel T, Stern D, Herbrich R (2011) Automated feature generation from structured knowledge. In: Proceedings of the 20th ACM international conference on Information and knowledge management, ACM, pp 1395–1404

Espinilla M, Nugent C (2017) Computational intelligence for smart environments. Int J Comput Intell Syst 10(1):1250–1251

Espinilla M, Medina J, Calzada A, Liu J, Martinez L, Nugent C (2017) Optimizing the configuration of an heterogeneous architecture of sensors for activity recognition, using the extended belief rule-based inference methodology. Microprocess Microsyst 52(Supplement C):381–390. https://doi.org/10.1016/j.micpro.2016.10.007

Even S (2011) Graph algorithms. Cambridge University Press, Cambridge

Fang H, He L, Si H, Liu P, Xie X (2014) Human activity recognition based on feature selection in smart home using back-propagation algorithm. ISA Trans 53(5):1629–1638

Ferrández-Pastor FJ, Mora-Mora H, Sánchez-Romero JL, Nieto-Hidalgo M, García-Chamizo JM (2017) Interpreting human activity from electrical consumption data using reconfigurable hardware and hidden markov models. J Ambient Intell Hum Comput 8(4):469–483

Gupta P, Dallas T (2014) Feature selection and activity recognition system using a single triaxial accelerometer. IEEE Trans Biomed Eng 61(6):1780–1786

Hall MA, Holmes G (2003) Benchmarking attribute selection techniques for discrete class data mining. IEEE Trans Knowl Data Eng 15(6):1437–1447

Hopcroft J, Tarjan R (1974) Efficient planarity testing. JACM 21(4):549–568

Horridge M, Drummond N, Goodwin J, Rector A, Wang HH (2006) The manchester owl syntax. In: Proc. of the 2006 OWL experiences and directions workshop (OWL-ED2006

Horrocks I (2008) Ontologies and the semantic web. Commun ACM 51(12):58–67

Horrocks I, Patel-Schneider P, Van Harmelen F (2003) From SHIQ and RDF to OWL: the making of a web ontology language. Web Semant 1(1):7–26

Kanter JM, Veeramachaneni K (2015) Deep feature synthesis: towards automating data science endeavors. In: Data science and advanced analytics (DSAA), 2015. 36678 2015. IEEE international conference on, IEEE, pp 1–10

Knijff J, Frasincar F, Hogenboom F (2013) Domain taxonomy learning from text: The subsumption method versus hierarchical clustering. Data Knowl Eng 83:54–69. https://doi.org/10.1016/j.datak.2012.10.002

Kohler J, Philippi S, Specht M, Ruegg A (2006) Ontology based text indexing and querying for the semantic web. Knowl Based Syst 19(8):744–754

Korhonen I, Parkka J, Van Gils M (2003) Health monitoring in the home of the future. IEEE Eng Med Biol Mag 22(3):66–73

Lehmann J, Auer S, Bëhmann L, Tramp S (2011) Class expression learning for ontology engineering. Web Semant Sci Serv Agents World Wide Web 9(1):71–81. https://doi.org/10.1016/j.websem.2011.01.001

Li C, Lin M, Yang L, Ding C (2014) Integrating the enriched feature with machine learning algorithms for human movement and fall detection. J Supercomput 67(3):854–865

López Gutiérrez, de la Franca C, Hervás R, Johnson E, Mondéjar T, Bravo J (2017) Extended body-angles algorithm to recognize activities within intelligent environments. J Ambient Intell Hum Comput 8(4):531–549. https://doi.org/10.1007/s12652-017-0463-y

Maedche A, Staab S (2001) Ontology learning for the semantic web. IEEE Intell Syst Appl 16(2):72–79

Mingers J (1989) An empirical comparison of pruning methods for decision tree induction. Mach Learn 4(2):227–243

Motik B, Patel-Schneider PF, Parsia B (2012) Owl 2 web ontology language. structural specification and functional–style syntax (second edition). https://www.w3.org/TR/owl2-syntax/. Accessed 30 Oct 2017

Noor MHM, Salcic Z, Kevin I, Wang K (2018) Ontology-based sensor fusion activity recognition. J Ambient Intell Hum Comput 1–15. https://doi.org/10.1007/s12652-017-0668-0

Okeyo G, Chen L, Wang H, Sterritt R (2014) Dynamic sensor data segmentation for real-time knowledge-driven activity recognition. Pervas Mob Comput 10:155–172

Ordóñez FJ, Roggen D (2016) Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. Sensors 16(1):115

Ordónez FJ, de Toledo P, Sanchis A (2013) Activity recognition using hybrid generative/discriminative models on home environments using binary sensors. Sensors 13(5):5460–5477

Oukrich N, El Bouazzaoui C, Maach A, Driss E (2017) Human activities recognition based on autoencoder pre-training and back-propagation algorithm. J Theor Appl Inf Technol 95(19):5194–5202

Paulheim H (2012) Generating possible interpretations for statistics from linked open data. Research and applications, the semantic web, pp 560–574

Quesada FJ, Moya F, Medina J, Martínez L, Nugent C, Espinilla M (2015) Generation of a partitioned dataset with single, interleave and multioccupancy daily living activities, vol 9454. Springer, Cham, pp 60–71

Rafferty J, Chen L, Nugent C, Liu J (2015) Goal lifecycles and ontological models for intention based assistive living within smart environments. Comput Syst Sci Eng 30(1):7–18

Riboni D, Bettini C (2011) Owl 2 modeling and reasoning with complex human activities. Pervas Mob Comput 7(3):379–395

Ristoski P (2015) Towards linked open data enabled data mining. In: European semantic web conference, Springer, pp 772–782

Ristoski P, Bizer C, Paulheim H (2015) Mining the web of linked data with rapidminer. Web Semant Sci Serv Agents World Wide Web 35(Part 3):142–151. https://doi.org/10.1016/j.websem.2015.06.004 (semantic Web Challenge 2014)

Salguero A, Espinilla M (2017) A flexible text analyzer based on ontologies: an application for detecting discriminatory language. Lang Resour Eval. https://doi.org/10.1007/s10579-017-9387-6

Shewell C, Medina-Quero J, Espinilla M, Nugent C, Donnelly M, Wang H (2017) Comparison of fiducial marker detection and object interaction in activities of daily living utilising a wearable vision sensor. Int J Commun Syst 30(5):e3223. https://doi.org/10.1002/dac.3223

Singh D, Merdivan E, Hanke S, Kropf J, Geist M, Holzinger A (2017) Convolutional and recurrent neural networks for activity recognition in smart environment. In: Towards integrative machine learning and knowledge extraction, Springer, pp 194–205

Sirin E, Parsia B, Grau B, Kalyanpur A, Katz Y (2007) Pellet: a practical owl-dl reasoner. Web Semant 5(2):51–53

Terziev Y (2016) Feature generation using ontologies during induction of decision trees on linked data. In: ISWC PhD Symposium

Uschold M, Gruninger M (1996) Ontologies: principles, methods and applications. Knowl Eng Rev 11(2):93–136

van Kasteren TLM et al (2011) Activity recognition for health monitoring elderly using temporal probabilistic models. ASCI

Villalonga C, Razzaq MA, Khan WA, Pomares H, Rojas I, Lee S, Banos O (2016) Ontology-based high-level context inference for human behavior identification. Sensors. https://doi.org/10.3390/s16101617

Wei T, Lu Y, Chang H, Zhou Q, Bao X (2015) A semantic approach for text clustering using wordnet and lexical chains. Expert Syst Appl 42(4):2264–2275. https://doi.org/10.1016/j.eswa.2014.10.023

Witten IH, Frank E, Hall MA, Pal CJ (2016) Data Mining: practical machine learning tools and techniques. Morgan Kaufmann, Cambridge

Xu C, Zhang X, He J (2016) Human activity recognition based on quantization on feature's classification capability **(preprints)**

Zhang F, Ma Z, Li W (2015) Storing owl ontologies in object-oriented databases. Knowl Based Syst 76:240–255. https://doi.org/10.1016/j.knosys.2014.12.020