Combining Domain Knowledge and Machine Learning for Robust Fall Detection

Mirchevska Violeta[1], Luštrek Mitja[2], Gams Matjaž[2]

(1) Result d.o.o., Bravničarjeva 11, Ljubljana, Slovenia

(2) Jožef Stefan Institute, Jamova 39, Ljubljana, Slovenia

E-mail: violeta.mircevska@result.si

Abstract: This paper presents a method for combining domain knowledge and machine learning (CDKML) for classifier generation and online adaptation. The method exploits advantages in domain knowledge and machine learning as complementary information sources. While machine learning may discover patterns in interest domains that are too subtle for humans to detect, domain knowledge may contain information on a domain not present in the available domain dataset. CDKML has three steps. First, prior domain knowledge is enriched with relevant patterns obtained by machine learning to create an initial classifier. Second, genetic algorithms refine the classifier. Third, the classifier is adapted online based on user feedback using the Markov decision process. CDKML was applied in fall detection. Tests showed that the classifiers developed by CDKML have better performance than ML classifiers generated on a one-sided training dataset. The accuracy of the initial classifier was 10 percentage points higher than the best machine learning classifier and the refinement added 3 percentage points. The online adaptation improved the accuracy of the refined classifier by additional 15 percentage points.

Keywords: prior domain knowledge, inductive machine learning, ambient intelligence, fall detection

1. INTRODUCTION

The training dataset to which machine learning is applied is often one-sided, not representing all real-life cases (Li *et al.*, 2007; Yang & Kecman, 2009). A typical example is medical studies based on laboratory samples upon which machine learning methods are applied. There is an important difference between laboratory samples that consider a limited number of clear-case scenarios and real life. For a classifier induced by machine learning to work in the general case, it must be induced using a sufficiently large and representative training dataset. Because such data is not always available, this can be partially countered by expert domain knowledge. Expert domain knowledge may be related to examples not present in the available domain dataset and thus may improve the generality and robustness of classifiers induced on such datasets. This paper addresses the problem of combining domain knowledge (DK) and machine learning (ML). It contributes a method for combining DK and ML (CDKML) for classifier generation and online adaptation.

We demonstrate our method on a fall detection task. This task is relevant for the elderly and the European Union, whose population is rapidly ageing. Predictions made by the Statistical Office of the European Communities state that the over-65 population in EU27 expressed as a percentage of the working-age population (aged between 15 and 64) will rise from 26% in 2010 to 53% in 2060 (Eurostat, 2011). This demographic change will make medical and care services scarce, increasing the need to motivate and assist the elderly to stay independent as long as possible. Innovative ICT systems can help the elderly live independently for longer and counteract reduced capabilities caused by age. One such system, developed as part of a European FP7 project, is the Confidence – Ubiquitous care system to support independent living (Confidence, 2011). Confidence aims to develop a system to monitor the health conditions of its elderly users in real-time. It detects falls and behaviour changes, including limping and physical inactivity. Confidence is based on wearable tags attached to a user reporting x, y and z tag coordinates with around 15 cm accuracy. Polls show that, with respect to privacy-violation issues, such hardware is more acceptable to users than, for example, video cameras. We used CDKML to develop the fall detection classifier in Confidence.

Confidence has three fall detection properties that make it challenging from an ML perspective. First, a representative dataset for falls is difficult to obtain because of the variety of fall types (in consultation with medical experts, we compiled a list of 18 fall types from over 40 listed in the literature), variations depending on the user, as well as ethical issues and injury dangers that prevent collecting large amounts of data from healthy persons simulating falls or, even worse, the elderly. Second, developing a classifier to suit each user in each possible circumstance from the start is difficult. Confidence detects falls as situations in which the user is lying/sitting motionless on the ground for a prolonged period of time. However, it is difficult to set a period of time to suit each user. For example, one user might never voluntarily lie or sit on the ground because of a physical disability that prevents him/her from getting up again, whereas another might exercise regularly on the living room carpet. Therefore, an online classifier adaptation is needed. Third, because of noise in the sensor data, misclassifications between similar postures occur. For example, sitting on a low chair may be misclassified as sitting on the ground. Such misclassifications of the posture directly influence the output of the fall detection model.

Motivation for developing the CDKML method lies in addressing ML shortcomings through DK. Using only initial clear-cases of the domain of interest, our method can create classifiers with improved general performance than ML classifiers induced on a one-sided dataset. The method also encompasses online classifier adaptation using information obtained from user feedback. The feedback is obtained occasionally and contains information about false negatives (i.e., the system did not detect the class of interest when there was one) or false positives (i.e., the system detected the class of interest when there was not one).

The paper is organized as follows. In Section 2, we present related work about combining DK and ML for classifier generation. In Section 3, we present the CDKML method for combining DK and ML for classifier generation and online adaptation. In Section 4, we present the experiments used to test the proposed method and obtained results. Section 5 concludes the paper.

2. RELATED WORK

Cognitive psychology research shows that human concept-learning considers both prior DK and interest concept examples (Wisniewski & Medin, 1994; Feldman, 1993; Heit, 2000). In principle, one information source offsets information missing from another source. DK influences interpreting examples. Before obtaining a considerable amount of concept examples, humans base their judgements mainly on prior DK. Conversely, examples affect DK. As the number of observed items of the interest concept increases, judgment relies increasingly on the actual observations and less on prior DK.

ML literature includes examples of concept learning using both prior DK and interest concept examples. A comprehensive overview of methods for incorporating prior DK into inductive ML is presented in (Yu, 2007). Yu categorizes these methods into four groups: (1) methods that use prior DK to prepare training examples, (2) methods that use prior DK to initiate the hypothesis or hypothesis space, (3) methods that use prior DK to alter the search objective and (4) methods that use prior DK to augment the search. The first group of methods incorporates prior DK into the training dataset used for induction by inserting virtual examples into the training dataset (Kambar, 2005; Niyogi *et al.*, 1998; Poggio & Vetter, 1992). Niyogi *et al.* (1998) showed that adding virtual examples is mathematically equivalent to incorporating the prior DK as a regulariser in function learning in certain restricted domains. In the second group, prior DK determines the part of the hypothesis space searched during induction (Zhu & Liu, 2010; Burns & Danyluk, 2000; Thrun, 1996). This is achieved by determining which part of the hypothesis space satisfies prior DK and using ML to search for a hypothesis in it, or by creating an initial hypothesis from the prior DK and using ML to refine it. The third group incorporates the DK into the inductive bias that guides the search through the hypothesis space. This is achieved by modifying the goal criterion to satisfy both DK and training examples, as in learning with constraints (Sabzekar *et al.*, 2011; Chen *et al.*, 2011; Davidson & Ravi, 2005), or by weighting the examples' influence in the training dataset (Brown *et al.*, 2000; Wu & Srihari, 2004; Wang *et al.*, 2004). The fourth group produces hypothesis candidates and adjusts the hypothesis space using DK during the on-going search (Decoste & Scholkopf, 2002; Pazzani *et al.*, 1991). In all cases, incorporating the DK aims to improve the generality of the induced ML model and/or the efficiency of the learning process.

The interactive ML field also explores methods for concept learning using both prior DK and interest concept examples. Compared to the previously described methods that incorporate DK into the ML algorithm, interactive ML is basically an iterative process of classifier generation through human-computer interaction (Benyon, 2001). Two strategies for model generation using interactive ML can be distinguished: (1) iterative improvement of a single model by refining the input information used during ML induction (Sun & Hardoon, 2010; Stumpf *et al.*, 2009; Bramer, 2005) and (2) generating multiple models to select one or several that are the most relevant from the user's perspective (Vidulin & Gams, 2011; Osei-Bryson, 2004).

The CDKML method belongs to the group of methods that use prior DK to initiate the hypothesis or hypothesis space. The domain expert determines the initial classifier and hypothesis space using DK only or using interactive ML. Genetic algorithms then refine the initial classifier using the available training dataset. Here, DK is included as a set of constraints on the classifier form (e.g., for a classifier with the form of a rule set, relations between rule parameters). Finally, the Markov decision process enables online classifier adaptation based on user feedback. DK specifies the mapping from obtained user feedback to changes of the state rewards. We are unaware of any work similar to combining the three steps. The final step is also novel.

3. CDKML METHOD

Figure 1 presents a general CDKML schema – a method for combining DK and ML for classifier generation and online adaptation. The schema contains three phases: (1) initialization, (2) refinement and (3) online adaptation. In the first phase, the domain expert specifies the hypothesis space and initial classifier. The domain expert may apply ML to the available training dataset, generating human-understandable classifiers to obtain additional insight in the interest domain (Vidulin & Gams, 2011) and include parts of these ML classifiers in the initial classifier. After determining the initial classifier, genetic algorithms refine it in the second phase, under expert supervision. The third phase adapts the classifier online using feedback information obtained from the user, who may indicate that the output class was incorrect. The adaptation is defined as a Markov decision process where user feedback is considered a reward signal from the environment. CDKML is not bound to a specific classifier form, but requires a human-understandable form. In the following subsections, we present each CDKML phase in detail.

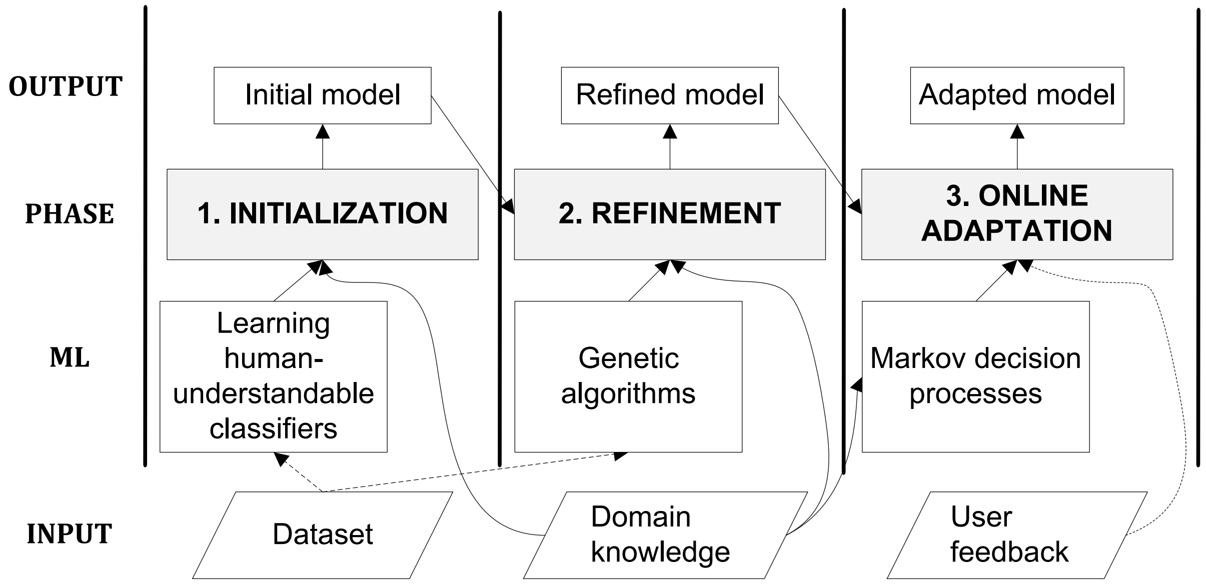


Figure 1 CDKML – method for combining DK and ML for classifier generation and online adaptation

The CDKML presentation is accompanied by examples from its application in the fall detection domain. In this specific domain, we selected the classifiers in the form of a set of disjunctive rules:

IF *condition1* AND *condition2* AND … AND *conditionN* THEN class.

We chose this classifier form because it can be constructed manually or with the help of supervised learning and modified by genetic algorithms or Markov decision processes.

3.1. Initialization

In the first CDKML phase (initialization), a domain expert defines an initial classifier. For example, in the fall detection domain, an expert may specify that if an elderly person is lying or sitting on the ground for a long period of time, then there is high probability of a fall, as elderly people are unlikely to lie or sit on the ground. Figure 2 presents an outline of this phase.

```
Function Initialization

Input: training examples Ex
Output: initial classifier $CL_{init}$
begin
    $CL_{init}$ := empty_set_of_rules

    add rules in $CL_{init}$ from domain knowledge

    // Explore human understandable ML models (e.g. decision tree, rule set)
    ML_ModelType := {decision tree, set of rules, etc.}
    for each ML_ModelType
    do
        create ML model on Ex using different initial parameters and attribute vectors
        explore patterns from the induced ML model
        add relevant rules in $CL_{init}$
    end do

end
```

Figure 2 Phase 1 – initialization

As an aid for designing the initial classifier, the expert may also examine human-understandable classifiers induced by supervised learning on the available training data. An example is presented in (Mirchevska *et al.*, 2009), where several decision trees are iteratively created to explore the space of possible classifiers. From these classifiers, the expert may obtain additional insight in the domain, modify DK, or add extracted patterns in the initial classifier.

In the fall detection example, the starting rule-based classifier contained the following rule types:

1. IF falling activity within $T1_{fall}$ seconds AND the user was lying/sitting on the ground $P1_{activity}$% of $T1_{activity}$ seconds AND the user was not moving $P1_{moving}$% of $T1_{moving}$ seconds THEN fall
2. IF falling activity within $T2_{fall}$ seconds AND the user was lying/sitting on the ground area afterwards $P2_{activity}$% of $T2_{activity}$ seconds THEN fall

3. IF the user was lying/sitting on the ground for $P3_{activity}$% of $T3_{activity}$ seconds AND the user was not moving $P3_{moving}$% of $T3_{moving}$ seconds THEN fall
4. IF the user was lying/sitting on the ground for $P4_{activity}$% of $T4_{activity}$ THEN fall

The expert specified these types of important fall patterns. However, specifying exact values for the parameters in the rules, for example specifying exact values for the parameters $P4_{activity}$ and $T4_{activity}$ in the last rule of the fall detection classifier, presented an issue for the expert, as the values may be influenced by system features, such as the noise in the sensor data or the ability of the system to correctly detect the lying/sitting posture. While the expert specified some initial values, and some values were obtained from the generated classifiers, the expert was not confident in them.

3.2. Refinement

The second CDKML phase (refinement) refines the initial classifier set from the domain expert to conform to system-related and general-user characteristics evident from the training dataset. As the rule structure in the classifier is fixed, standard rule induction methods are unsuitable for the desired training. Genetic algorithms (Eiben & Smith, 2003) thus tune the initial classifier parameters to maximize its performance on the training dataset. Figure 3 presents an outline of this phase.

We apply genetic algorithms thus: We use the Pittsburgh approach, where each individual in the population represents one possible solution. The individual is a vector containing parameters of all rules in the rule-based classifier. For example, if the rule-based classifier contains 8 rules with 4 parameters each, the individual is 32 elements long. The elements are real values inside an interval defined by the expert. The fitness function for evaluating the quality of each individual is accuracy on the training dataset. Fitness values fall within the interval [0, 1]. We thus want to tune the parameters of the rule-based classifier to the training dataset, hopefully avoiding overfitting, as the domain expert defines the structure of the rule-based classifier. We used elitism, meaning that the best individual is always transferred to the new population.

Using genetic algorithms allows constraining relations between rules and parameters within a rule. In the presented fall detection classifier, rule strictness decreases from rule type 1 to rule type 4. The first rule type requires detecting falling activity and the user to be immovable and lie/sit on the ground to detect a fall, whereas the fourth rule type requires only the user to lie/sit on the ground. The duration of lying/sitting on the ground needed for the first rule type to detect a fall should be the shortest (the combination with other evidence more quickly assures that a fall happened) and should increase toward rule type 4. The relation between the required periods of lying/sitting on the ground in the rules is expressed through constraints. Additionally, if the rule requires detecting falling activity to detect a fall, the falling activity should be detected before the person lied/sat on the ground. This relation is also represented as a constraint. The fitness of individuals representing rule-based classifiers that violate the constraints is set to minimal fitness.

```
Function Refinement

Input: initial classifier $CL_{init}$; constraints between the rule parameters of the initial
       classifier Constraints; training examples Ex; parameters of the genetic algorithms
       POPULATION_SIZE, CROSSOVER_RATE, MUTATION_RATE, STOP_CRITERION,
       MAX_ITERATIONS
Output: refined classifier $CL_{ref}$
begin
    //Create individual $I_{base}$ representing the initial classifier $CL_{init}$ using the Pittsburgh
    //approach
    $I_{base}$ := empty_vector
    for each rule r in $CL_{init}$
    do
        put the parameters of rule r in a single vector $Vec_r$
        append $Vec_r$ to $I_{base}$
    end do

    //Create initial population
    $P_{init}$ := empty_set
    add $I_{base}$ to $P_{init}$
    for i:=1 to POPULATION_SIZE
    do
        $I_i$ := create an individual by random changes of $I_{base}$
        add $I_i$ to $P_{init}$
    end do

    //Evolve population
    $I_{best}$:= Find_fittest_individual($P_{init}$, Constraints, Ex) //function defined below
    iter := 0, $P_{new}$:= $P_{init}$
    while ((accuracy($I_{best}$) < STOP_CRITERION) AND (iter < MAX_ITERATIONS))
    do
        $P_{old}$ := $P_{new}$, $P_{new}$ := empty_set, iter:= iter+1
        add $I_{best}$ to $P_{new}$ //Use elitism
        for i:=1 to (POPULATION_SIZE/2)
        do
            select two parents from $P_{old}$ by tournament selection
            crossover parents with probability CROSSOVER_RATE
            mutate individuals obtained by crossover with probability MUTATION_RATE
            add new individuals to $P_{new}$
        end do
        $I_{cur\_best}$ := Find_fittest_individual($P_{new}$, Constraints, Ex)
        if(accuracy($I_{cur\_best}$) > accuracy($I_{best}$))
        then
            $I_{best}$:= $I_{cur\_best}$
        end if
    end do

    $CL_{ref}$ := update the values of the parameters in $CL_{init}$ with the values in $I_{best}$

end



Function Find_fittest_individual

Input: population P; constraints between the parameters of the initial classifier
Constraints; training examples Ex

Output: individual $I_{result}$

begin
    for each I in P do
        if I violates Constraints
            fitness(I) := 0
        else
            fitness(I) := accuracy(I) on Ex
        end if
    end for each

    $I_{result}$ := individual in P with highest fitness value

end
```

Figure 3 Phase 2 – refinement

The genetic algorithm outputs the final general rule-based classifier. The expert should observe various classifiers generated with different input parameters and choose the best one in his/her opinion, not solely based on accuracy.

3.3. Online adaptation

The third CDKML phase (online adaptation) adapts the general rule-based classifier online using feedback obtained from a particular user. We explain the adaptation process using the example rule: "IF the user was lying on the ground for $P_{activity}$% in $T_{activity}$ THEN fall". Figure 4 presents an outline of this phase.

```
Function Initialize_MDPs

Input: refined classifier $CL_{ref}$
Output: set of MDPs
begin
    for each rule $r$ in $CL_{ref}$
    do
        create a $MDP_R(S, A, P, R)$ for $r$
        initialize the set of states $S$ to a n-dimensional state space, where n is the
        number of parameters in $r$
        initialize the set of actions $A$ to all possible parameter value changes
        initialize the transition probability $P(s, a, s')$ to be 0 or 1
        initialize the elements of the reward matrix $R$ to zero
        initialize current state $MDP_R.current$ to the parameter values of $r$
    end do

end


Function Update_rules

Input: current classifier $CL_{current}$; user feedback $UF \in$ {false positive, false negative}
accompanied with the triggering example $Ex$; penalty amount for false positive $PaFp$;
penalty amount for false negative $PaFn$
Output: adapted rule-based classifier $CL_{adpt}$
begin
    if $UF$= false positive
    then
        $R_{fp}$ := set of all rules of $CL_{current}$ that caused a false positive

        for each rule $r$ in $R_{fp}$
        do
            in $MDP_R$ reduce the utility of the current state and all states that it
            dominates by $PaFp$
            $C_{states}$ := set of neighbouring states of $MDP_R.current$ with highest utility
            $MDP_R.current$ := state from $C_{states}$ with maximum distance from $Ex$
        end do
    else // UF = false negative
        find rule $r$ in $CL_{current}$ with minimum distance from $Ex$
        in $MDP_R$ reduce the utility of the current state and all states that dominate it by
        $PaFn$
        $C_{states}$ := set of neighbouring states of $MDP_R.current$ with highest utility
        $MDP_R.current$ := state from $C_{states}$ with minimum distance from $Ex$
    end if

end
```

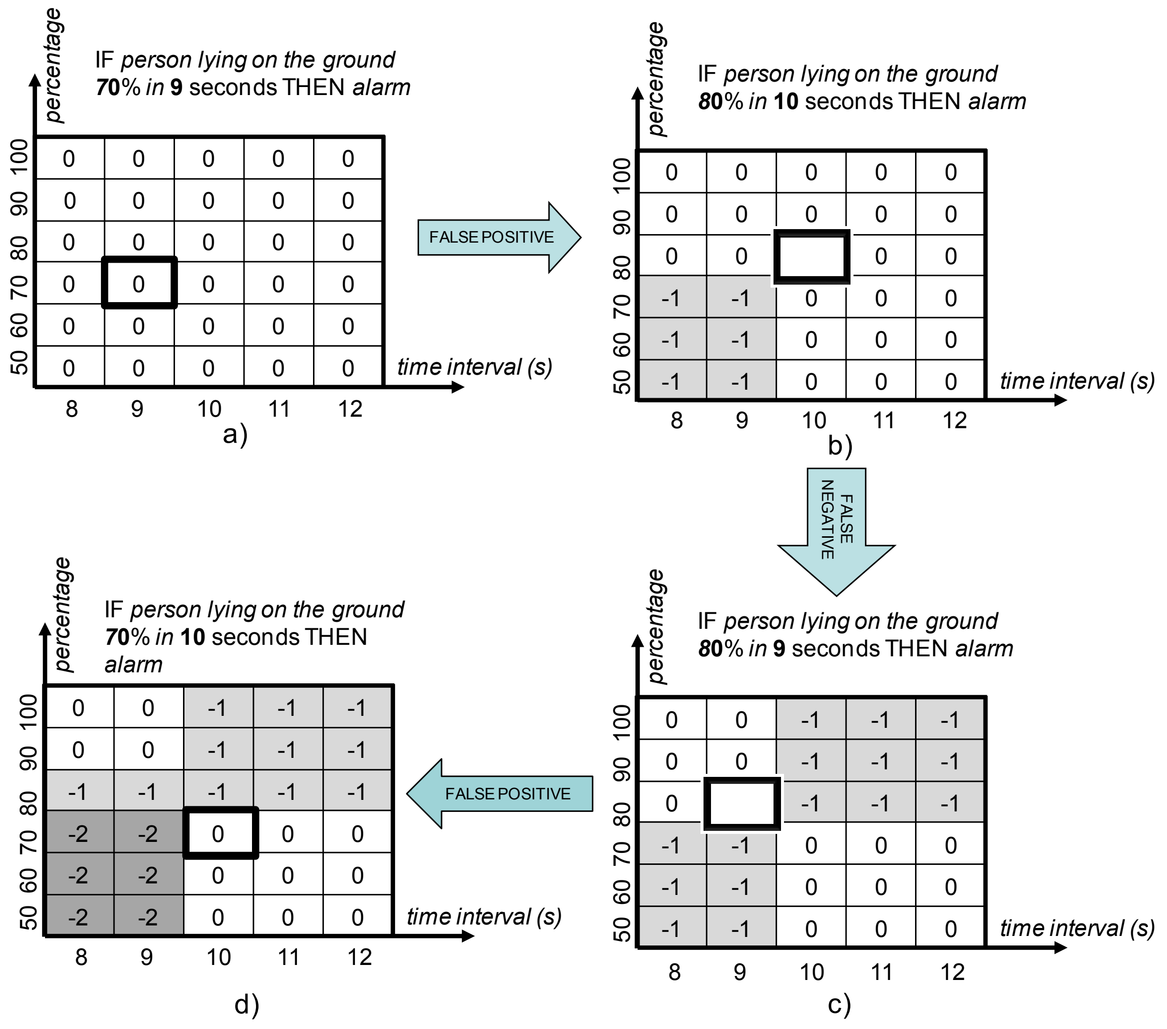
Figure 4 Phase 3 – online adaptation

The problem of adapting a rule in the rule-based classifier is defined as a Markov decision process (Russell & Norvig, 2003), *$MDP_R(S, A, P, R)$*. The state space *S* of each rule is *N*-dimensional, where *N* is the number of adjustable parameters in the rule. The example rule space is two-dimensional, with one dimension representing the set of possible percentage values and the other representing possible time interval values (Figure 5). We use discrete parameters. In each step, we can increase or decrease the value of each parameter by one unit. The set of actions *A* are combinations of such parameter value changes. Parameter value changes are deterministic; the values in the transition probability matrix *P(s, a, s')*, denoting the probability of transitioning from state *s* to *s'* when executing action *a*, are 0 or 1. The elements of the reward matrix *R* reflect the obtained user feedback (a reward signal from the environment) and may change. Translating user feedback to the appropriate state reward requires information of how each parameter influences the output of the rule, as specified by the domain expert. The MDP goal state is the combination of rule parameter values that best separates fall events from non-fall events and depends on the needs of a particular user and may change through time.

Figure 5 presents the process of adapting the example rule. Current rule parameter values (*MDP_R.current)* are highlighted with a black rectangle. First of all, the reward matrix *R* of *MDP_R* is initialized to zero for all states and *MDP_R.current* is set to the rule's values in the refined classifier (Figure 5a). We assume that, after a certain period of time, a false positive feedback is obtained. In this concrete rule, a false positive feedback reduces the current state reward and all states dominated by it (states with less strict parameter values than the current state's parameter values) by a penalty amount *paFp*, which in our example is -1, because a false positive indicates that the parameters of the rule must be made stricter (Figure 5b). After obtaining such feedback, the set of neighbouring *MDP_R.current* states with the highest utility is determined, and the new *MDP_R.current* value is the state with the maximum parameter distance from the example that caused the false positive. In the example rule, the distance from a state *s* to an example *Ex* that triggered user feedback is calculated thus:

$$dist(s, Ex) = \min_{t \in T}(\max((s_{timeInterval} - t), (s_{percentage} - Ex_{percentage}(t)))$$

where *T* is the set of possible time interval values in the rule, $s_{timeInterval}$ and $s_{percentage}$ represent the rule parameter values represented by state *s*, and $Ex_{percentage}(t)$ represents the amount of lying on the ground in time interval *t* in *Ex*. The new MDP_R*.current* value in Figure 5b has stricter values for both the time and percentage parameters. We again assume that, after a certain period of time, a false negative feedback is obtained. A false negative feedback reduces the reward of the current state and all states that dominate it (states with stricter parameter values than the current state's parameter values) by a penalty amount *paFn*, which in our example is -1, because a false negative indicates that the parameters of the rule are too strict and must be relaxed (Figure 5c). Again, the set of neighbouring MDP_R*.current* states with the highest utility is determined, and the new value of MDP_R*.current* is the state with the minimum parameter distance from the example that caused the false negative. Figure 5c presents a case where the feedback result reduced the strictness of the percentage parameter of MDP_R*.current*, while the time parameter remained unchanged. The initial state was avoided because of the negative reward received during the first false positive. Rule parameters values are adapted in this way after each obtained user feedback.



IF person lying on the ground 70% in 9 seconds THEN alarm
IF person lying on the ground 80% in 10 seconds THEN alarm
IF person lying on the ground 80% in 9 seconds THEN alarm
IF person lying on the ground 70% in 10 seconds THEN alarm
percentage
time interval (s)
FALSE POSITIVE
FALSE NEGATIVE
FALSE POSITIVE
a)
b)
c)
d)

Figure 5 Online classifier adaptation using user feedback

User feedback does not affect all rules in the rule-based classifier. If false positive feedback is obtained, only the rules that incorrectly classified the concrete example as positive are adapted. If false negative is obtained, only the rule that needs the least change to cover the concrete example is adapted.

4. EVALUATION

This section evaluates CDKML on the fall detection task. We used CDKML to build a rule-based classifier for fall detection as part of the fall detection module in the Confidence system. We first describe the fall detection module in the Confidence system. We then present the data on which the rule-based fall detection classifier was evaluated. Finally, we comment on the obtained results.

4.1 Fall detection in Confidence

This section presents the fall detection module of the Confidence system by which CDKML was evaluated.

Figure 6 presents a simplified version of the part of the Confidence system related to fall detection. Detailed system descriptions can be found in literature (Lustrek *et al.*, 2011; Kaluza et al., 2010) and a detailed description of the fall detection module in (Mirchevska *et al.*, 2010).

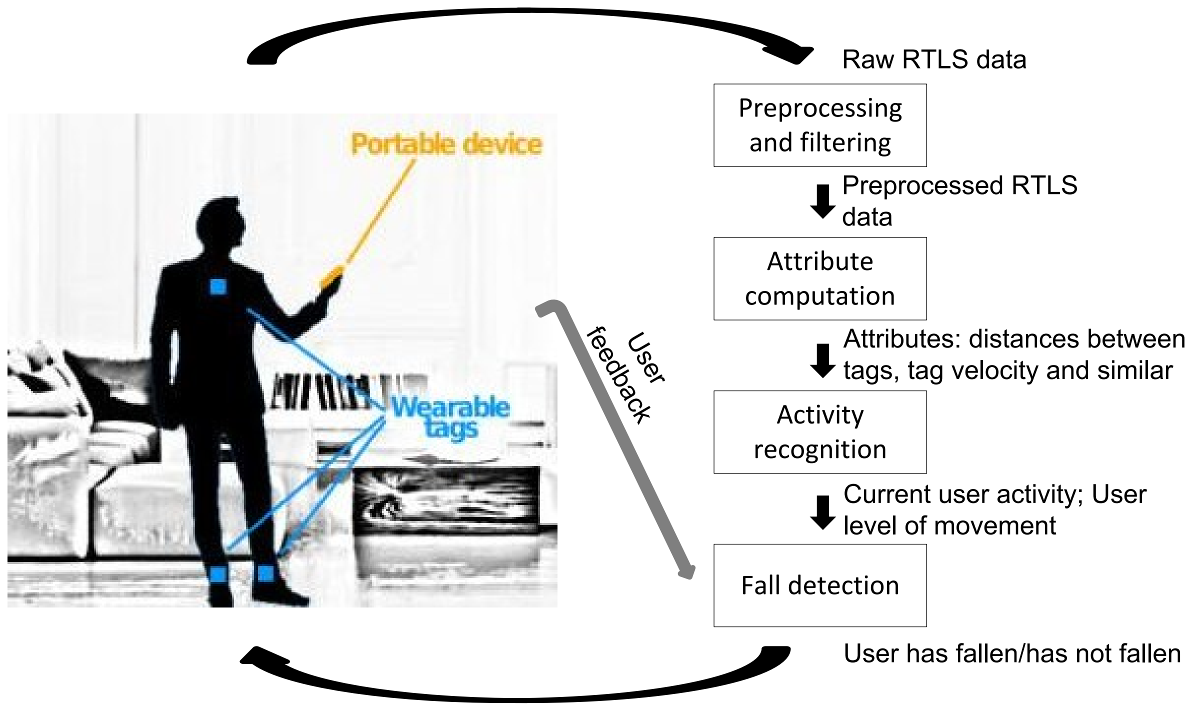

Raw RTLS data
Preprocessing and filtering
Preprocessed RTLS data
Attribute computation
Attributes: distances between tags, tag velocity and similar
Activity recognition
Current user activity; User level of movement
Fall detection
User has fallen/has not fallen
User feedback
Portable device
Wearable tags

Figure 6 Fall detection in the Confidence system

In the Confidence system, the user is equipped with wearable tags whose coordinates are detected by radio sensors. The experiments presented in this paper used the real-time localization system (RTLS) Ubisense (Ubisense, 2011) for this purpose. In a typical open-environment, the localization accuracy of Ubisense is on average about 15 cm but in practice may occasionally drop to 200 cm or more. The raw RTLS data is first preprocessed to reduce noise. The preprocessed RTLS data is then given as input to the attribute computation module. This module computes characteristics of the user's body, including tag velocity and amount of movement, and relations between body parts, including the distance between tags. The activity recognition module uses these characteristics to classify the user's activity into one of seven classes: standing, sitting, lying, standing up, going down, falling, or on all fours. Additionally, if the system detects lying or sitting, it determines whether these activities are done at appropriate places, including a bed for lying or chair for sitting, or at inappropriate places, such as on the ground. The activity recognition module's output is given as input to the fall detection module.

The fall detection module uses data concerning user activity history and user movement levels to detect falls, using the four rule types shown in Section 3.1, which mostly depend on whether an elderly person is lying or sitting at an inappropriate place (e.g., on the ground) for a long period of time, resulting in a high probability of a fall. Fall detection does not rely only on detecting the falling activity (high acceleration toward the ground), as it always lasts a very short time and is thus difficult to recognize. Compared to detecting falling activity, lying and sitting on the ground are easier to detect, which makes them convenient for fall detection. However, this approach has certain issues. Activity on all fours may be misclassified as lying on the ground. Because lying on the ground indicates a fall, such misclassifications may lead to false positives. However, activity on all fours that occurs when a person is searching for something on the ground is shorter than the period of lying/sitting on the ground that follows a fall and includes more movement. Another common misclassification occurs when a person is sitting on a low chair. Sitting on a low chair may be misclassified as sitting on the ground because of the noise in the localization system measurements and may cause false positives. However, the amount of sitting on the ground recognized when the person is sitting on a low chair should be lower than the amount of this activity recognized when the person is sitting on the ground. Therefore, the main challenge faced when developing the fall detection classifier is providing reliable and robust fall detection even in various complex real life circumstances.

4.2 Data

We designed a test scenario to investigate the generality and robustness of the developed rule-based classifiers, as well as their adaptation capabilities. The scenario (Table 1) contains two types of events: straightforward and complex events.

Straightforward events represent typical fall and non-fall events. Both fall events (1 and 2) involve high acceleration toward the ground during the falling activity. High acceleration during the falling activity is a characteristic feature of falls, and setting thresholds for it is a common way of detecting falls. The user lands lying (1) or sitting (2) on the ground after the fall. Non-fall events contain activities commonly done at home, including walking, sitting on a chair, or lying in bed (3). Additionally, searching for something on the ground on all fours or lying (4) is added as a non-fall event.

Complex events represent atypical falls and non-fall events that may be particularly easily misclassified. One type of non-fall event is lying down quickly on a bed or sitting down quickly on a chair (7). This event includes high acceleration during the lying/sitting down activity, which is a characteristic feature of falls. However, the lying/sitting that follows is on the bed/chair, enabling the rule-based classifier to differentiate falls from non-falls. The other non-fall event is sitting on a low chair (8). Five non-fall events of sitting on a low chair are present in the scenario. They differ in the position of the user's body on the chair: the user sits straight or leans forward, backward, to the left, or to the right. In complex fall events (5 and 6), the user slowly descends to the ground, trying to hold onto nearby furniture. However, after the falling activity, the user lands lying/sitting on the ground.

Table 1 Test scenario

STRAIGHTFORWARD EVENTS			COMPLEX EVENTS		
	Description	Fall		Description	Fall
1	Tripping, landing flat on the ground	Yes	5	Falling slowly (trying to hold onto furniture), landing flat on the ground	Yes
2	Falling when trying to stand up, landing sitting of the ground	Yes	6	Falling slowly when trying to stand up (trying to hold onto furniture), landing sitting on the ground	Yes
3	Normal everyday behaviour, such as walking, sitting on a chair, lying in bed	No	7	Lying down quickly on the bed / Sitting down quickly on the chair	No
4	Searching for something on the ground on all fours and lying	No	8	Sitting on a low chair	No

We selected the falls in the test scenario from a list of 18 fall types, compiled in consultation with medical personnel. The falls were demonstrated by a physician, who also provided guidance during initial recordings.

All events present in the test scenario were recorded in single recordings interspersed with short periods of walking. Each recording lasted around 20 minutes. The recordings were made by 5 healthy volunteers (3 male and 2 female), 5 times by each. Figure 7 presents the total number of fall and non-fall examples in the recorded data. The large number of non-fall events among the complex events is due to the examples of sitting on a low chair. We recorded many such examples because the adaptation (in the third phase) primarily occurred on them.

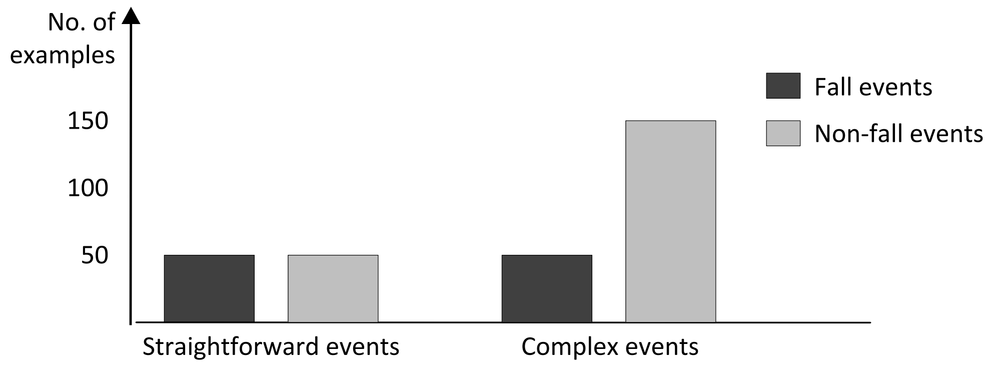


Figure 7 Total number of fall and non-fall examples in the recorded data

4.3 Results

We evaluated the first and second CDKML phases as follows: The domain expert first specified the initial classifier. Genetic algorithms then refined the initial classifier based only on examples of straightforward events (to demonstrate laboratory testing). We used leave-one-person-out evaluation, where the refined classifier was generated from examples of four people and tested on examples of the fifth, which was excluded from the training dataset. This was repeated five times, using a different person for testing each time. The accuracy of the refined classifier was tested on both straightforward and complex events of the person excluded from the training dataset, thus illustrating real-life performance, which includes both clear and complex cases. The test on the straightforward events shows how well the classifier performs on events present in the training dataset. The test on the complex events, conversely, tests the generality and robustness of the generated classifier, as the complex events are not present in the learning process.

We evaluated the online classifier adaptation part, i.e., phase 3 of CDKML, thus: The refined classifier was adapted to a concrete user using examples of both straightforward and complex concrete user events, because we wanted to test the ability of the method to learn new cases while preserving its performance on the cases present in the training dataset in phase 2 of CDKML. Four of the five concrete user scenario recordings were randomly presented one by one to the fall detection classifier. The fall detection classifier classified each event as fall or non-fall, then feedback was provided and the fall detection classifier was adapted, as necessary, before the next event. The final adapted classifier evaluation was done on the recording, which was not used in the adaptation phase.

For comparison, fall detection classifiers were induced using ML only. The attributes were the time since detecting the last falling activity, the amount of each type of activity in time intervals from 5 to 15 seconds, and the amount of user movement in this interval range. The attributes are equivalent to the parameters of the rules in the rule-based fall detection classifier. We used the following ML algorithms: decision trees (J48), rules (JRip), support vector machines (SMO), random forest (RandomForest), and Naïve Bayes (NaiveBayes). In brackets, we give the Weka implementation (Hall *et al.*, 2009) for these algorithms.

We evaluated fall detection classifier performance using two measures: accuracy on a subset of events ACC_{events} and F-measure on a subset of events FM_{events}. The accuracy on a subset of events ACC_{events} is

$$ACC_{events} = \frac{correctly\ detected\ examples\ of\ type\ E \in events}{all\ examples\ of\ type\ E \in events}$$

The F-measure on a subset of events FM_{events} is

$$FM_{events} = \frac{2 * P_{events} * R_{events}}{P_{events} + R_{events}}$$

where P_{events} is the precision and R_{events} is the recall of the classifier of events E belonging to the set *events*.

Table 2 presents the performance of the induced fall detection classifiers on straightforward events only, on complex events only, and on the whole sequence with respect to the accuracy on fall examples ACC_f, accuracy on non-fall examples ACC_{nf}, overall accuracy ACC_{all} and overall F-measure FM_{all}. Table 3 presents the accuracy of the induced classifiers on each event in the test scenario separately ACC_e. The measures were computed for each person separately, and the values in Tables 2 and 3 represent the averages. Additionally, the refinement CDKML phase was performed five times in each test run, because of the stochastic nature of the genetic algorithm and the average value was considered.

Table 2 shows that the best overall accuracy among ML classifiers was obtained by support vector machines with an ACC_{all} of 53 percentage points. The ML classifiers tended to be biased towards fall recognition. They had maximal ACC_f; however, they raised many false positives, as indicated by the low ACC_{nf} values. The overall accuracy of the initial classifier was 10 percentage points higher than support vector machines. It slightly decreased on the ACC_f, from 100 to 98 percentage points, but increased greatly on the ACC_{nf} from 30 to 46 percentage points. The refinement of the initial classifier based on straightforward-event examples contributed to a 3 percentage point increase in accuracy. The ACC_{nf} increased to 53 percentage points at the cost of a slight decrease in ACC_f, which was 93 percentage points. The adapted classifier outperformed the refined classifier in accuracy by 15 percentage points; however, as mentioned above, it had an advantage over the previous classifiers, because it obtained examples of both straightforward and complex events during learning, and the examples came from the concrete user on which the tests were made. The adapted classifier had the highest ACC_{nf}, 85 percentage points, whereas its ACC_f had 71 percentage points. The tests concerning F-measure, which compensates for uneven class distribution, also confirmed the performance improvement.

Table 2 Classifier comparison using accuracy on the fall examples (*Acc_f*), accuracy on the non-fall examples (*ACC_{nf}*), overall accuracy (*ACC_{all}*) and overall F-measure (*FM_{all}*).[1)]

CLASSIFIER			ML					CDKML		
			J48	JRip	SMO	Random Forest	Naïve Bayes	Initial classifier	Refined classifier	Adapted classifier
Training dataset			SF events	SF events	SF events	SF events	SF events	SF events	SF events	All events
Testing dataset	Straightforward events only	ACC_f	1.00	1.00	1.00	1.00	1.00	0.98	0.91	0.71
		ACC_{nf}	0.68	0.68	0.68	0.70	0.30	0.82	0.94	0.99
		ACC_{all}	0.84	0.84	0.84	0.85	0.65	0.90	0.92	0.85
		FM_{all}	0.86	0.86	0.86	0.87	0.74	0.91	0.92	0.83
	Complex events only	ACC_f	1.00	1.00	1.00	1.00	1.00	0.98	0.96	0.72
		ACC_{nf}	0.15	0.12	0.17	0.14	0.04	0.34	0.39	0.81
		ACC_{all}	0.37	0.34	0.38	0.36	0.28	0.50	0.53	0.79
		FM_{all}	0.44	0.43	0.45	0.44	0.41	0.49	0.51	0.63
	All events	ACC_f	1.00	1.00	1.00	1.00	1.00	0.98	0.93	0.71
		ACC_{nf}	0.28	0.26	0.30	0.28	0.10	0.46	0.53	0.85
		ACC_{all}	0.52	0.51	0.53	0.52	0.40	0.63	0.66	0.81
		FM_{all}	0.58	0.57	0.59	0.58	0.53	0.64	0.65	0.71

Table 3 compares the performance of the induced classifiers on each event separately. As mentioned above, the ML classifiers detected all fall events; however, they performed poorly on all non-fall events. Introducing domain knowledge to the initial classifier significantly improved the *ACC_e* on the normal behaviour non-fall event. The refinement improved *ACC_e* on the non-fall event searching on the ground. This event was included in the training data for the refinement phase, so increased performance was expected; it was achieved at the cost of neglecting certain fall events. The adapted classifier correctly recognized almost all falls after which the user lay on the ground, but it had difficulties with falls after which the user sat on the ground. Sitting on the ground is a rare event in real life. Sitting on a low chair, an event for which *ACC_e* significantly increased, is a much more common real life event. The classifier frequently confused these two activities for one another. Not only is the user's posture similar, but they can both last a long time, during which the user is immovable. Some examples of sitting on a low chair are in fact undistinguishable from falls because of the noise in the measurements of the sensors used. Adapting the fall detection classifier establishes a tradeoff between these events. As sitting on a low chair is far more frequent then falls after which a user sits on the ground in a normal sitting position,

[1)] The abbreviation SF stands for straightforward events.

misclassifications of this event are more costly. The adapted classifier is thus inclined to reduce misclassifications during sitting on a low chair at the cost of not detecting certain falls after which the user lands sitting on the ground. In any case, user immovability after falls for additional or prolonged time should enable detecting these false negatives; however, because of practical reasons, we could not capture this in the recordings.

Table 3 Classifier comparison using the accuracy on each event (*ACC$_e$*) in the test scenario

CLASSIFIER/ *ACC$_e$*		STRAIGHTFORWARD TESTS				COMPLEX TESTS			
		FALLS		NON-FALLS		FALLS		NON-FALLS	
		Tripping (1)	Falling landing sitting (2)	Normal behavi-our (3)	Searching on the ground (4)	Falling slowly (5)	Falling slowly landing sitting (6)	Lying/ Sitting down quickly (7)	Sitting on low chair(8) -avg. value
Machine Learning	J48	1.00	1.00	0.68	0.68	1.00	1.00	0.64	0.06
	JRip	1.00	1.00	0.76	0.60	1.00	1.00	0.60	0.02
	SMO	1.00	1.00	0.76	0.60	1.00	1.00	0.88	0.03
	Random Forest	1.00	1.00	0.76	0.64	1.00	1.00	0.76	0.02
	Naïve Bayes	1.00	1.00	0.12	0.44	1.00	1.00	0.20	0.01
Initial classifier		1.00	0.96	0.96	0.68	0.96	1.00	0.96	0.22
Refined classifier		0.96	0.86	0.96	0.92	0.96	0.96	1.00	0.27
Adapted classifier		0.96	0.46	1.00	0.98	0.84	0.60	1.00	0.77

5. CONCLUSION

We presented the CDKML – method for combining DK and ML for classifier generation and online adaptation. The method has three phases: initialization, refinement, and online adaptation. The domain expert specifies the initial CDKML classifier in the first phase. The expert may also use hypotheses induced by ML as help. In the second phase, genetic algorithms adjust the initial classifier to suit system- and general-user characteristics. Tests show that the classifiers developed after the first two phases are already more reliable and robust than ML classifiers built from limited examples from the domain of interest. In addition to general classifier generation, in the third phase, we presented a method for online classifier adaptation based on specific user feedback indicating incorrect system output. Tests show that, during online adaptation, the classifier is adjusted to correctly recognize events not present in the training dataset, making tradeoffs between contradictory examples based on the cost of each misclassification.

The method is suitable for domains that have limited obtainable training data and available domain knowledge. Online adaptation is essential if specific characteristics of the objects of interest must be accommodated; the user must be able to give feedback about misclassifications to allow the online adaptation. One such domain is fall detection, on which we evaluated the method. The method is suitable for modelling behaviour in general and for studies in the medical or biological fields, for which large, representative training datasets are typically difficult to obtain.

ACKOWLEDGEMENTS

This research is partly financed by the European Union, European Social Fund and partly by the European Community's Framework Programme FP7/2007–2013 under grant agreement No. 214986.

References

BENYON, D. (2001) The new HCI? navigation of information space, *Knowledge-Based Systems,* **14(8)**, 425-430.

BRAMER, M. (2005) Inducer: a public domain workbench for data mining, *International Journal of Systems Science,* **36(14)**, 909-919.

BROWN, M., GRUNDY, W., LIN, D., CRISTIANINI, N., SUGNET, C., FUREY, T., et al. (2000) Knowledge-based analysis of microarray gene expression data by using support vector machines, *Proceedings of the National Academy of Sciences,* **97**, 262-267.

BURNS, B. D., and DANYLUK, A. P. (2000) Feature Selection vs Theory Reformulation: A Study of Genetic Refinement of Knowledge-based Neural Networks, *Machine Learning - Special issue on multistrategy learning*, 89-107.

CHEN, M.-C., CHAO, C.-M., and WU, K.-T. (2011) Pattern filtering and classification for market basket analysis with profit-based measures. *Expert Systems*.

CONFIDENCE. (2011) *Confidence*, Retrieved August 23, 2011, from http://www.confidence-eu.org/

DAVIDSON, I., and RAVI, S. S. (2005) Hierarchical clustering with constraints: Theory and practice, *Proceedings of the Nineth European Principles and Practice of KDD (PKDD)*, 59-70.

DECOSTE, D., and SCHOLKOPF, B. (2002) Training invariant support vector machines machine learning, *Machine Learning,* **46**, 161-190.

EIBEN, A. E., & SMITH, J. E. (2003) *Introduction to Evolutionary Computing,* Springer-Verlag.

EUROSTAT. (2011) *Eurostat,* Retrieved August 23, 2011, from http://epp.eurostat.ec.europa.eu/tgm/table.do?tab=table&init=1&language=en&pcode=tsdde511&plugin=1

FELDMAN, R. S. (1993) *Understanding Psychology,* New York: Mc Graw-Hill.

HALL, M., FRANK, E., HOLMES, G., PFAHRINGER, B., REUTEMANN, P., and WITTEN, I. H. (2009) The WEKA Data Mining Software: An Update, *SIGKDD Explorations,* **11(1)** , 10-18.

HEIT, E. (2000) Background Knowledge and Models of Categorization, In U. Hahn, & M. Ramscar, *Similarity and Categorization,* 155-178.

KALUŽA, B., MIRCHEVSKA, V., DOVGAN, E., LUŠTREK, M., and GAMS, M. (2010) An agent-based approach to care in independent living, *Proceedings of Ambient Intelligence*, 177-186.

KAMBAR, S. (2005) *Generating synthetic data by morphing transformation for handwritten numeral recognition (with v-svm) (Master thesis).*

LI, D.-C., YEH, C.-W., TSAI, T.-i., FANG, Y.-H., and HU, S. C. (2007) Acquiring knowledge with limited experience, *Expert Systems*, **24(3)**, 162-169.

LUŠTREK, M., GJORESKI, H., KOZINA, S., CVETKOVIĆ, B., MIRCHEVSKA, V., and GAMS, M. (2011) Detecting Falls with Location Sensors and Accelerometers, *Proceedings of Innovative Applications of Artificial intelligence.*

MIRCHEVSKA, V., KALUŽA, B., LUŠTREK, M., and GAMS, M. (2010) Real-Time Alarm Model Adaptation Based on User Feedback, *Proceedings of Workshop on Ubiquitous Data Mining in ECAI 2010* , 39-43.

MIRCHEVSKA, V., LUŠTREK, M., VELEZ, I., VEGA, N. G., and GAMS, M. (2009) Classifying Posture Based on Location of Radio Tags. *Ambient Intelligence Perspectives II - Selected papers from the Second International Ambient Intelligence Forum 2009*, 85-92.

NIYOGI, P., GIROSI, F., and POGGIO, T. (1998) Incorporating prior information in machine learning by creating virtual examples, *Proceedings of the IEEE*, 2196-2209.

OSEI-BRYSON, K.-M. (2004) Evaluation of decision trees: a multi-criteria approach, *Computers and Operations Research*, 1933-1945.

PAZZANI, M., BRUNK, C., and SILVERSTEIN, G. (1991) A knowledge-intensive approach to learning relational concepts, *The Eighth International Workshop on Machine Learning*, 432-436.

POGGIO, T., and VETTER, T. (1992) *Recognition and structure from one 2D model view: Observations on prototypes, object classes and symmetries, Tech. Rep. AIM-1347.* Cambridge, MA, USA: Massachusetts Institute of Technology.

RUSSELL, S., & NORVIG, P. (2003) *Artificial intelligence A Modern Approach,* Prentice Hall.

SABZEKAR, M., YAZDI, H. S., & NAGHIBZADEH, M. (2011) Relaxed Constraints Support Vector Machine, *Expert Systems*.

STUMPF, S., RAJARAM, V., LI, L., WONG, W.-K., BURNETT, M., DIETTERICH, T., et al. (2009) Interacting meaningfully with machine systems: Three experiments, *International journal of human-computer studies,* **67**, 639-662.

SUN, S., and HARDOON, D. R. (2010) Active learning with extremely sparse labeled examples, *Neurocomputing,* **73**, 2980-2988.

THRUN, S. (1996) *Explanation-Based Neural Network Learning: A Lifelong Learning Approach.* Boston, MA: Kluwer Academic Publishers.

UBISENSE (2011) *Ubisense,* Retrieved August 23, 2011, from http://www.ubisense.net/

VIDULIN, V., and GAMS, M. (2011) Impact of higher-level knowledge on economic welfare through interactive data mining, *Aplied artificial intelligence,* **25(4)**, 267-291.

WANG, L., XUE, P., and CHAN, K. L. (2004) Incorporating prior knowledge into SVM for image retrieval, *Proceedings of the 17th International Conference on Pattern Recognition*, 981-984.

WISNIEWSKI, E. J., and MEDIN, D. L. (1994) On the Interaction of Theory and Data in Concept Learning, *Cognitive science,* **18(2)**, 221-281.

WU, X., and SRIHARI, R. (2004) Incorporating prior knowledge with weighted margin support vector machines, *Proceedings of the tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 326-333.

YANG, T., and KECMAN, V. (2009) Adaptive local hyperplane algorithm for learning small medical data sets, *Expert systems,* **26(4)**, 355-359.

YU, T. (2007) *Incorporating Prior Domain Knowledge into Inductive Machine Learning Its implementation in contemporary capital markets (PhD thesis).*

ZHU, Z., and LIU, P. (2010) Feasibility research of text information filtering based on genetic algorithm, *Scientific Research and Essays,* **5(22)** , 3405-3410.

The authors

Violeta Mirchevska

Violeta Mirchevska is a researcher at Result d.o.o., cooperating closely with the Department of Intelligent Systems at the Jožef Stefan Institute. She completed her Bachelor's in computer science and automation at the Faculty of Electrical Engineering and Information Technologies, Ss. Cyril and

Methodius University, Macedonia in 2007 and is currently a PhD candidate at Jožef Stefan International Postgraduate School, Slovenia. Her research focuses on modelling agent behaviour based on observing low-level action sequences by leveraging both domain knowledge and machine learning. The main application areas of her research are user profiling, remote health monitoring and security.

Mitja Luštrek

Dr. Mitja Luštrek is a researcher at the Jožef Stefan Institute in Slovenia. He is the head of the ambient intelligence group at the Department of Intelligent Systems. His main research area is ambient intelligence with a focus on human behavior analysis. He has experience with wearable inertial sensors and real-time locating systems, and has used artificial intelligence techniques for activity recognition and detection of anomalous behaviors. He has also worked in several other areas of artificial intelligence, including game playing, heuristic search, and machine learning in bioinformatics. He is an editor of the Informatica journal and a member of the executive board of the Slovenian Artificial Intelligence Society.

Matjaž Gams

Prof. dr. Matjaž Gams (http://dis.ijs.si/Mezi/) is a senior researcher at the Jožef Stefan Institute, Ljubljana, Slovenia. His research interests include artificial intelligence, intelligent systems, intelligent agents, machine learning, cognitive sciences, and information society. His publication list includes 500 items, 70 in scientific journals. Matjaž Gams is heading the Department of Intelligent Systems at the Jožef Stefan Institute. He is currently president of ACM Slovenia and the cofounder of the Engineering Academy, Artificial Intelligence Society, and Cognitive Sciences Society in Slovenia. He is an executive contact editor of the journal Informatica and member of the editorial board of several international journals. He headed several major applications in Slovenia, including a virtual agent for the Slovenian employment agency, an expert system controlling the quality of nearly all national steel production and a text-to-speech system in Slovenian, donated to several thousand users. Matjaž Gams also teaches several courses in computer sciences at the graduate and postgraduate levels.