BEHAVIOR MODELING BY COMBINING MACHINE LEARNING AND DOMAIN KNOWLEDGE

Violeta Mirchevska

Doctoral Dissertation
Jožef Stefan International Postgraduate School
Ljubljana, Slovenia, September 2013

Evaluation Board:
Prof. Dr. Bogdan Filipič, Chairman, Jožef Stefan Institute, Ljubljana, Slovenia
Prof. Dr. Vladislav Rajkovič, Member, Faculty of Organisational Sciences, University of Maribor, Kranj, Slovenia
Prof. Dr. Jurij Tasič, Member, Faculty of Electrical Engineering, University of Ljubljana, Ljubljana, Slovenia

MEDNARODNA PODIPLOMSKA ŠOLA JOŽEFA STEFANA
JOŽEF STEFAN INTERNATIONAL POSTGRADUATE SCHOOL

Violeta Mirchevska

BEHAVIOR MODELING BY COMBINING MACHINE LEARNING AND DOMAIN KNOWLEDGE

Doctoral Dissertation

MODELIRANJE OBNAŠANJA S KOMBINIRANJEM STROJNEGA UČENJA IN DOMENSKEGA ZNANJA

Doktorska disertacija

Supervisor: Prof. Dr. Matjaž Gams

Co-supervisor: Dr. Mitja Luštrek

Ljubljana, Slovenia, September 2013

Contents

Abstract

In the last two decades a range of successful machine-learning applications emerged as large amounts of archived data become available for many real-world problem domains. Credit-card fraud detection, optical character recognition and book recommendations are just a few examples. Machine learning algorithms may automatically extract comprehensive concept models solely from concept examples, finding even patterns which are too subtle to be detected by humans. However, their performance greatly depends on the quality and the completeness of the available concept examples.

The dissertation proposes a novel method, named CDKML (Combining Domain Knowledge and Machine Learning), for classifier generation in the case of scarce data. We assume there are at least two reasons for scarce data: (1) sufficient general-purpose data may be costly or otherwise difficult to obtain, possibly due to great domain variation, and (2) general-purpose data may be inappropriate for some deployments, for example, because they are user-specific. CDKML incorporates domain knowledge in the learning process for the purpose of overcoming the challenges posed by insufficient general-purpose data. Domain knowledge may contain information on a domain not captured by the available concept examples. It thus complements machine learning. For the purpose of overcoming the challenges posed by lacking deployment-specific data, CDKML utilizes user feedback. User feedback is given 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 none).

CDKML consists of three phases: initialization, refinement and online adaptation. The goal of the first two phases (initialization and refinement) is to create a general-purpose classifier under expert supervision. In the initialization phase, an expert specifies a set of patterns important for distinguishing the concept of interest. The patterns may be extracted from domain knowledge or be obtained using interactive data mining. In the refinement phase, an optimization algorithm is used for finding the most suitable general-purpose pattern-parameter values by maximizing the classifier's accuracy on the available training data. The third CDKML phase (online adaptation) uses user feedback to fine-tune the pattern-parameter values to the characteristics of a specific deployment. The online adaptation problem is formulated as a Markov decision process.

The performance of the CDKML method was evaluated on three behavior modeling tasks: behavioral cloning, posture recognition and fall detection. We describe the built classifiers in each domain and compare their performance to classifiers induced solely with machine learning. CDKML achieved higher accuracy than classical machine-learning algorithms when learning from scarce data by leveraging the available domain knowledge and user feedback.

Povzetek

Strojno učenje je vse bolj prisotno v vsakdanjem življenju, saj so za vedno več področij na voljo podatki, primerni za ta namen. Odkrivanje prevar s kreditnimi karticami, optično prepoznavanje znakov in priporočanje knjig so le nekateri primeri uspešnih aplikacij, ki so danes v širši uporabi. Algoritmi za strojno učenje gradijo modele učnih konceptov na podlagi primerov teh konceptov. Sposobni so odkriti tudi vzorce, ki so preveč subtilni, da bi jih opazili ljudje. Vendar je njihova uspešnost v veliki meri odvisna od kakovosti in popolnosti učnih primerov.

Disertacija predlaga novo metodo, imenovano CDKML (ang. Combining Domain Knowledge and Machine Learning), za gradnjo klasifikacijskih modelov za probleme, pri katerih je na voljo premalo učnih primerov. Obstajata vsaj dva vzroka za nezadostne podatke: (1) pridobitev podatkov je lahko draga ali težavna, morda zaradi velike raznolikosti domene, in (2) splošni podatki za nekatera področja uporabe niso primerni, ker so, denimo, preveč odvisni od uporabnika. CDKML nezadostnost učnih podatkov rešuje z vključevanjem domenskega znanja v učni proces. Domensko znanje lahko vsebuje informacije o domeni, ki niso zajete z razpoložljivimi učnimi primeri, in s tem dopolnjuje strojno učenje. Poleg tega CDKML predvideva sprotno prilagajanje modela posamičnemu primeru uporabe z izkoriščanjem povratnih informacij od uporabnikov. Povratne informacije izpostavljajo napačno klasificirane primere − bodisi negativne (ciljni koncept ni bil prepoznan) bodisi pozitivne (ciljni koncept je bil prepoznan, ko ga v resnici ni bilo).

Metoda CDKML obsega tri faze: začetek, izboljševanje in sprotno prilagajanje. Namen prvih dveh faz je tvoriti splošen klasifikacijski model pod nadzorom strokovnjaka. Strokovnjak v prvi fazi določi množico vzorcev, ki opredeljujejo učni koncept. Vzorce lahko oblikuje na podlagi svojega domenskega znanja, lahko pa jih pridobi tudi z interaktivnim podatkovnim rudarjenjem. Parametri vzorcev se nato izboljšajo v drugi fazi metode z uporabo optimizacijskega algoritma. Cilj te faze je najti nabor vrednosti parametrov, ki maksimizira točnost modela na učnih podatkih. Tretja faza metode prilagaja parametre vzorcev posamičnemu primeru uporabe, za kar uporabi povratne informacije od uporabnikov. Problem sprotnega prilagajanja je formuliran v obliki markovskega odločitvenega procesa.

Metodo CDKML smo ovrednotili na treh domenah modeliranja obnašanja: kloniranje obnašanja, prepoznavanje drže in zaznavanje padcev. Modele, zgrajene s CDKML, smo primerjali z modeli, zgrajenimi z običajnim strojnim učenjem. Z uporabo domenskega znanja in povratnih informacij od uporabnikov je CDKML dosegel klasifikacijsko točnost višjo kot klasični algoritmi za strojno učenje pri učenju iz omejenih podatkov.

1 Introduction

The field of machine learning (ML) is concerned with development of algorithms that enable computer programs to learn and automatically improve with experience (Mitchell, 1997). ML algorithms have been successfully applied to a wide variety of domains ranging from credit-card fraud detection based on classifiers induced from transaction examples (Chan et al., 1999) to book recommendations based on automatically extracted person's preferences from examples of his/her past purchases (Mooney and Roy, 2000) to creating helicopter control logic based on trial-and-error experience (Ng et al., 2004). The majority of ML algorithms learn concept models solely from observed examples without considering existing prior domain knowledge (DK). Archived data for many real world problem domains is growing exponentially, supported by the low-cost digital storage, providing a boost for ML. The amount and density of available data is often beyond the human processing capacity. But learning solely from examples is a disadvantage for ML in domains for which a limited amount of concept examples (capturing a subset of the possible cases) is available.

Learning with a limited amount of concept examples is illustrated in Figure 1.1. Here, the task is posture recognition aimed at distinguishing six postures: standing, sitting, lying, falling, standing up and slowly going down. A posture example is a pair $<$*attributes*, *class*$>$, where *attributes* contain information about the position and velocity of a person's body parts and the distance between them when the person is in posture *class*. Figure 1.1 presents a decision tree induced by the J48 algorithm in Weka (Hall et al., 2009) with the default algorithm parameter values and the minimum number of examples per leaf equal to 1000. The tree is induced from a dataset composed of 34707 posture examples, each of which describes a posture by 44 attributes. Each path from the tree root to a leaf represents one learned posture rule. The type of posture the rule corresponds to is presented in the leaf. Each leaf also contains information about the number of correctly classified examples by the posture rule (the first number in the brackets) and the number of incorrectly classified ones (the second number in brackets). The decision tree contains 7 rules. Let's examine the rule for standing represented by the leaf in the second row from the top of the tree. This rule states that a person is standing if the vertical distance between his/her left ankle and the chest (Distance_Z_direction_AnkleLeftToChest) is greater than 1.08 m. It is supported by 23110 examples, but it misclassifies 508 examples. This rule complies with human understanding of the standing posture, as large vertical distance between the ankles and the chest is a representative feature of standing. Let's also examine the rule for sitting represented by the leaf in the second row from the bottom of the tree. This rule states that a person is sitting if the vertical distance between his/her left ankle and the chest (Distance_Z_direction_AnkleLeftToChest) is smaller than or equal to 1.08 m, the vertical distance between his/her right ankle and the chest (Distance_Z_direction_AnkleRightToChest) is greater than 0.7 m, the total velocity of the right wrist (Velocity_total_WristRight) is smaller than or equal to 1.42 m/s, and the total distance between the right wrist and the chest (Distance_total_WristRightToChest) is smaller than or equal to 0.44 m. It is supported by 1205 sitting examples, but it misclassifies 123 examples. As humans we would not completely agree with this rule, since the total wrist velocity and the total wrist-chest distance

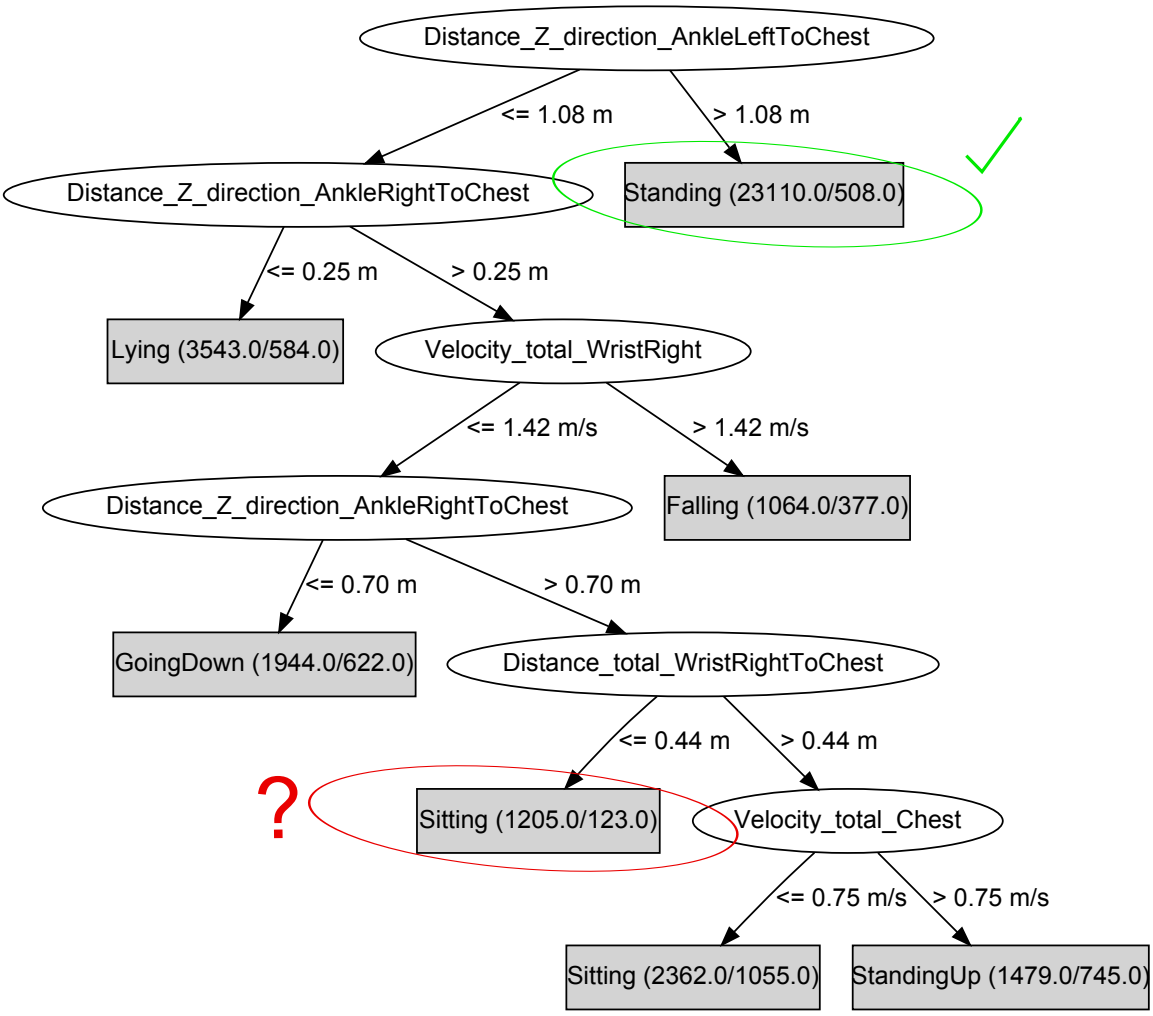

Figure 1.1: A decision tree for recognizing postures induced from a limited amount of concept examples.

are not a distinguishing feature of sitting. A person may perform fast hand moves and strengthen the arms while sitting. The reason why the rule contains this set of conditions is that in the recorded examples the people performed sitting with the hands near the waist and without moving the hands. If sitting examples in which a person strengthens the arms and/or performs fast hand moves were available to the learner, this rule would not have been present in the decision tree.

The main research problem addressed in the dissertation is: *How can a reliable classifier be created when learning from a limited amount of concept examples?* For notation consistency in the dissertation, we define four notions: a classifier, a class pattern, an evidence and class-pattern parameters.

Definition 1.1: *A classifier* C is a set of class patterns P_{class} defining a categorization, $C = \{P^i_{class}\}_i$.

The decision tree presented in Figure 1.1, for example, is a classifier for distinguishing six posture categories: standing, sitting, lying, falling, standing up and slowly going down.

Definition 1.2: *A class pattern* P_{class} is a set of evidences E supporting an object's membership to a category *class*, $P_{class} = \{E^i\}_i$.

Each rule in the decision tree is an example of a class pattern (or *a pattern* for short). The posture classifier contains 7 patterns.

Definition 1.3: *An evidence* E is a boolean function representing an atomic object characteristic, $E : \{attribute\ values\} \rightarrow \{true, false\}$.

Each rule condition is an example of an evidence. Evidences typically compare attribute's value with a constant, i.e., an evidence's parameter. The presented standing rule encompasses one evidence, Distance_Z_direction_AnkleLeftToChest $>$ 1.08 m, which compares the attribute Distance_Z_direction_AnkleLeftToChest with the constant 1.08 m. The constant 1.08 m is the evidence's parameter value.

Definition 1.4: *Class-pattern parameters* are the union of parameters present in a class-pattern's evidences.

The presented standing rule has one class-pattern parameter (or *pattern parameter* for short) whose value is 1.08 m.

When learning from a limited amount of concept examples, the learner may create a classifier from patterns which, although representative of the available examples, are not characteristic for the learned concept. Such classifier would perform poorly in real life because it does not capture the essence of the learned concept. This issue may be partially tackled by introducing DK as an additional information source in the learning process. Experts are often capable of reliably categorizing examples (e.g., human postures). They may verify a classifier's patterns and/or provide characteristic patterns from DK, but often have difficulties in specifying a complete classifier. Expert DK complements ML. On the one hand, DK may contain patterns which are not captured by the available concept examples. On the other hand, ML may extract novel patterns not present in DK solely from concept examples. Therefore, a combination of DK and ML is expected to produce classifiers with a characteristic set of concept patterns.

The dissertation proposes a novel approach to combining DK and ML, named CDKML. It is a three-phase approach to learning consisting of initialization, refinement and online adaptation.

The aim of the initialization phase is to extract a comprehensive set of concept patterns that form a classifier. It is an interactive process in which an expert examines human-understandable classifiers induced by ML and selects patterns characteristic for the learned concept. For example, in the posture-recognition task, the expert may select the presented standing pattern: IF Distance_Z_direction_AnkleLeftToChest $>$ 1.08 m THEN standing. The expert may modify the presented sitting pattern excluding and/or replacing obsolete evidences: IF Distance_Z_direction_AnkleLeftToChest $\leq$ 1.08 m AND Distance_Z_direction_AnkleLeftToChest $>$ 0.7 m AND Velocity_total_Chest $\leq$ 1.42 m/s AND Distance_XY_AnkleLeftToChest $\leq$ 0.75 m THEN sitting. The expert may also add patterns from DK.

Having the classifier's patterns, the refinement phase determines the most suitable general-purpose pattern-parameter values. Each pattern implicitly represents a class-boundary segment whose layout (e.g., position, length) is specified by the pattern's parameter values. Figure 1.2 depicts a 2D projection of the class boundaries specified by the two example patterns given in the previous paragraph. The optimal layout of the class-boundary segments greatly depends on their interconnection with the segments represented by the rest of the classifier's patterns. This interconnection is not captured in the initialization phase, where the pattern-parameter values are obtained separately either from a ML classifier or are estimated using DK. The refinement phase searches for the optimal pattern-parameter values using an optimization algorithm by maximizing the classifier's accuracy on the available concept examples. Here, DK poses constraints of the search space.

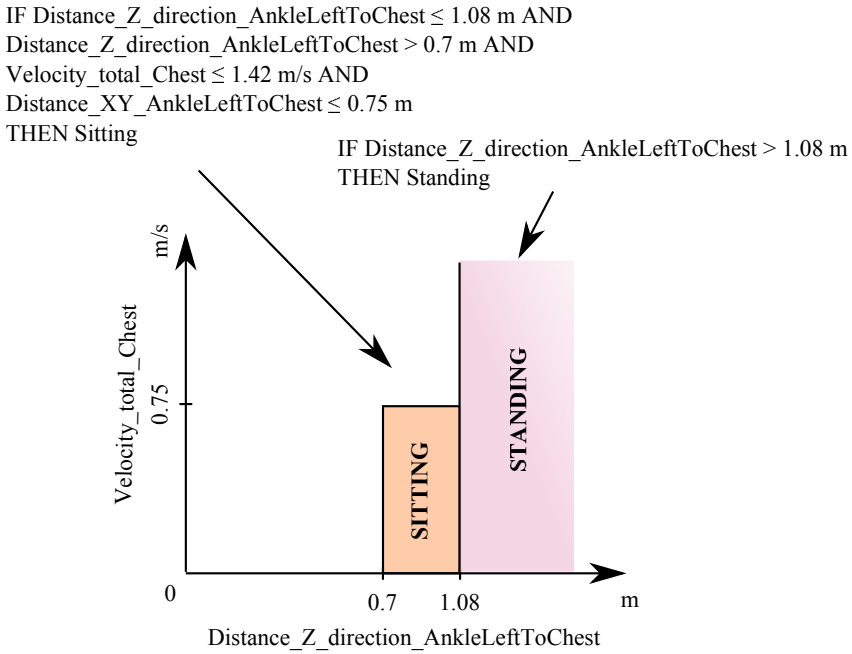

Figure 1.2: Visualization of patterns' class boundary – 2D projection.

The online adaptation aims at adjusting the pattern-parameter values to suit a particular system deployment. What are, for example, the optimal pattern-parameter values in the posture-recognition classifier for a particular person? In order to pose minimal burden to the user, the online adaptation is based on user feedback. User 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) and false positives (i.e., the system detected the class of interest when there was none). The online adaptation problem is defined as a sequential decision making problem using the Markov decision process formalism. DK specifies the mapping from user feedback to rewards (indicators of the desirability of concrete pattern-parameter values).

1.1 Hypothesis and Purpose

The hypothesis of the dissertation is that a combination of interactive data mining to extract a comprehensive set of characteristic concept patterns and optimization algorithms to determine optimal pattern-parameter values (general-purpose or deployment-specific) is needed for creation of reliable classifiers in domains for which a limited amount of concept examples is available. The purpose of the dissertation is to improve concept learning from a limited amount of concept examples.

The main dissertation goals are the following:

- Survey state-of-the-art methods for classifier creation by combining DK and ML;
- Develop a method for generating reliable general-purpose and deployment-specific classifiers in domains for which a limited amount of concept examples is available by leveraging both DK and ML;
- Apply the method to three behavior modeling domains: behavioral cloning, posture recognition and fall detection.

1.2 Scientific Contributions

This dissertation proposes a new, three-phase method, named CDKML, for extraction of reliable classifiers in domains where the training examples partially represent the domain properties, but human experts can contribute with their DK. The method and analysis related to the dissertation were published in journals and conference proceedings (Mirčevska et al., 2009; Mirchevska et al., 2013a,b). The complete bibliography is presented in Appendix A.

The main contributions of the dissertation are the following:

- A novel method, named CDKML, for classifier generation and online adaptation which leverages both DK and ML. The novelty is in the way of integration of three phases: initialization, refinement and online adaptation;
- A novel classifier adaptation based on user feedback using Markov decision processes. This, third phase of the CDKML method, is novel on its own.

As additional contributions we consider: (i) an algorithm to estimate the decision-tree hypothesis space size, (ii) an extension of the agent definition by adding the agent's role as an important agent characteristic and by modifying the agent's action representation, and (iii) an improvement in classifier accuracy in comparison to standard ML approaches on two important ambient-assisted-living subtasks: posture recognition and fall detection.

1.3 Overview of the Dissertation Structure

Chapter 2 contains a survey of the current state-of-the-art in combining DK and ML for classifier generation. Two major approaches are present in the literature. The first incorporates DK in the ML algorithm as a pre-learning step. The learning is then performed without expert engagement. The second emphasizes the importance of human-computer interaction during the whole knowledge discovery process.

Chapter 3 analyzes why a combination of expert DK and ML offers the possibility to extract reliable classifiers from a limited amount of task examples. First, we formalize ML. Second, we present the characteristics of learning tasks that would benefit from the incorporation of DK. Third, we formalize expert DK and present ways in which it may influence the learning process to improve generalization. Finally, we formalize learning using both concept examples and expert DK.

Chapter 4 describes the domains that motivated the development of the CDKML method: behavioral cloning, posture recognition and fall detection. Three main research questions arose from the motivating domains: (1) is an expert capable of selecting a comprehensive set of patterns of the learned concept, thus creating a representative concept classifier, (2) how can optimal pattern-parameter values be obtained from a training dataset, and (3) how can we leverage user feedback for online classifier fine-tuning to user needs.

The main contribution of the dissertation – the CDKML method – is described in Chapter 5. First, we present the used classifier form. Then, we formalize each of the three CDKML phases: initialization, refinement and online adaptation.

Chapter 6 describes and evaluates the classifiers created using CDKML in the three motivating domains. They are compared to five ML classifiers: decision trees (Quinlan, 1993), a set of rules (Cohen, 1995), support vector machines (Keerthi et al., 2001), random forest (Breiman, 2001) and Naïve Bayes (John and Langley, 1995).

Finally, Chapter 7 presents our conclusions from the performed study and the ideas for future work.

2 Related Work

Cognitive psychology research shows that human concept-learning considers both prior DK and concept examples (Wisniewski and Medin, 1994; Heit, 2000; Feldman, 2005). 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 judgments mainly on prior DK. Conversely, examples affect DK. As the number of observed concept examples increases, judgment relies increasingly on the actual observations and less on prior DK.

ML literature also includes examples of concept learning using both prior DK and concept examples. This chapter presents related work in this domain.

2.1 Incorporating Expert Domain Knowledge into the Learning Process of Inductive Machine Learning Algorithms

A comprehensive overview of methods for incorporating prior DK into inductive ML is presented by Yu (2007). Yu categorizes these methods into four groups, i.e., methods that use prior DK to:

- prepare training examples,
- initialize the hypothesis or hypothesis space,
- alter the search objective,
- augment the search.

In all cases, incorporating DK aims to improve the generality of the induced ML classifier and/or the efficiency of the learning process.

2.1.1 Using Domain Knowledge to Prepare Training Examples

This group encompasses approaches to enlarging the number of training examples by DK. DK serves as a source for identifying data transformation functions T that out of a valid example $(x, f(x))$ produce a valid example $(Tx,\ y_T(f(x)))$. The most commonly used are invariances to transformations in which y_T is the identity mapping. Novel training examples, called *virtual examples*, are created by applying the transformation functions T on the training examples.

Kambar (2005) presents an approach to enlarging the number of training examples in the handwritten numeral recognition domain. Morphing transformations with convex evolution are used for generating virtual examples, which represent the transition from a *source* to a *target* training numeral. The concept class of a newly generated example is determined according to its distance from the *source* and *target* numerals. Virtual examples closer to the *source* are assigned the class value of the *source*, whereas virtual examples closer to the

target obtain the class value of the *target*. The virtual examples are validated by support-vector-machine classifier created on the original training dataset with no virtual examples. If the support-vector-machine classifier outputs the same concept class as the class assigned to the virtual example according to its *source* and *target* distance, the example is put to the enlarged training dataset. Otherwise, it is deleted.

Niyogi et al. (1998) discuss creating virtual examples of objects belonging to a special, well-behaved class called linear object class. Linear objects are objects which can be represented as the weighted sum of views of other objects (their components). For example, a three-dimensional cuboid can be represented by three two-dimensional cuboids. Faces and speech also belong to the linear object class. Object transformations in this case can be computed as the weighted sum of transformed views. In order to create virtual examples, patterns of variability and class-specific deformations are learned from a representative training set of views of generic or prototypical objects of the class of interest (e.g., different views of the face of one person). These patterns are applied to novel objects' views to create virtual examples.

Niyogi et al. (1998) show that incorporating DK through virtual examples can be equivalent to incorporating DK through regularization. Although the proof was derived only for functions with radial symmetry, it mathematically confirms the benefit of incorporating DK through virtual examples.

The major drawback of DK incorporation using virtual examples is the increase in the computational cost of classifier training. Schölkopf et al. (1996) propose the Virtual SV method which preserves the advantages of the virtual examples approach without increasing the computational cost. The method bases on the observation that the support vector set contains the necessary information to solve a classification task. Support vector machine classifiers trained solely on support vectors had test performance not worse than such classifiers trained on the full dataset (Vapnik, 1995). The Virtual SV method, therefore, proposes generating virtual examples from the support vectors, termed *virtual support vectors*. The training process encompasses three steps: (1) a support vector machine is trained on the full dataset in order to extract the support vectors, (2) virtual support vectors are created by applying invariance transformations, and (3) another support vector machine is trained on the enlarged set of support vectors.

Virtual examples may also be obtained from domains related to the learning problem of interest. For the purpose of activity recognition, Zheng et al. (2009) propose an approach to using labeled examples from a source set of activities (e.g., doing laundry) to train a classifier to recognize a different, but related set of target activities (e.g., indoor cleaning). First, an activity similarity function is obtained by Web knowledge mining. Web search is used to find Web pages describing each of the source and target activities. The similarity between two activities is measured according to the similarity of the text on the Web pages describing the activities. Second, pseudo training data is generated by relabeling the source examples. The pseudo examples contain the same feature values as the source examples, but their class value is an activity in the target domain. Each pseudo example is assigned a confidence level which equals to the similarity between the source and the target activity class measured using the text similarity function. Finally, the weighted support vector machines method (Chang and Lin, 2011) is applied to the pseudo training data to obtain the classifier for predicting the target activities.

2.1.2 Using Domain Knowledge to Initialize the Hypothesis or Hypothesis Space

The hypothesis space may be partially or completely selected by DK. The learning process in this case searches a reduced, more appropriate hypothesis space.

Incorporating DK into the kernel, a non-linear generalization of inner products (Jäkel et al., 2007), used by kernel methods is a common approach to hypothesis space selection using DK. Lauer and Bloch (2008) present a review of methods for incorporating DK in the kernel used by support vector machines. The kernel may capture invariances to transformations (Decoste and Schölkopf, 2002; Pozdnoukhov and Bengio, 2004; Haasdonk et al., 2005) as well as invariances to permutations (Kondor and Jebara, 2003). In contrast to these approaches in which DK is hard-coded in the kernel, the selection of the kernel may also be formulated as an optimization problem (Wang et al., 2005). The idea is to use DK to define a quality criterion of the kernels. The best kernel is then obtained by gradient-descent search optimization in a predefined space of kernel functions. Wang et al. (2005) applied this approach of kernel selection in the domain of content-based image retrieval with relevance feedback. In this domain, the learning is performed on a small set of examples labeled by the user. The positive examples share a common concept in the user's mind, while the negative examples capture other heterogeneous concepts. Wang et al. (2005) propose selection of the kernel which tightly clusters the positive examples in the kernel space and pushes the negative examples away from the positive, scattering them at the same time. This heuristic is encoded in a kernel quality criterion and the Broyden-Fletcher-Goldfarb-Shanno (BFGS) Quasi-Newton method is used for finding the best kernel to the available training examples. Examples of the use of kernels customized by DK is also present in distance-based classification algorithms (Simard et al., 1993).

Možina (2009) proposes argument-based machine learning (ABML), an extension to classical ML which uses DK in the form of arguments to constrain the hypothesis space. Arguments are reasons supporting the membership of an example to a particular concept class (positive arguments) and statements specifying attributes that do not influence the example's class (negative arguments). Consider predicting the weather situation on a particular day based on the previous day weather situation and a training example "previous_weather = *sunny*, previous_temperature = *high*, previous_pressure = *low*, previous_humidity = *normal*, class = *rainy*". An expert may explain why the class value is rainy using the argument: "It was raining because of the low pressure on the previous day". The ABML output is selected from a reduced hypothesis space encompassing classifiers in which all proofs of a classification *class* mention at least one positive argument and none of the negative arguments.

DK may be used for creating an initial hypothesis (classifier). The motivation for initializing the hypothesis by DK is that such hypothesis would provide better search starting point, contributing to a more efficient convergence.

One of the first approaches to hypothesis initialization by DK is the knowledge-based artificial neural network (Towell and Shavlik, 1994). Domain theory in the form of non-recursive, propositional Horn clauses is firstly transformed to an artificial neural network topology. Then, the backpropagation algorithm is applied for tuning the parameter values of the neural network to the training examples. Burns and Danyluk (2000) propose two methods, INDIGENT and TNT-INDIGENT, for refinement of knowledge-based artificial neural networks using genetic algorithms. INDIGENT refines the input features of the neural network. The features specified by the domain theory and features present in decision trees induced using C4.5 (Quinlan, 1993) constitute the genotype. Each gene in the genotype is associated to an input feature; the gene value represents presence or absence of the

feature in the feature subset. A knowledge-based artificial neural network is created on each genotype feature subset. TNT-INDIGENT refines the topology of the neural network. It uses genotypes that represent an entire neural network. Both methods measure the genotype fitness by the N-fold cross validation accuracy of its corresponding neural network on the training examples.

Hu et al. (2009) propose coupling artificial neural networks with partially known relationships extracted from DK, aiming to enhance "black box" neural network learning to a semi-analytic one. In dynamic system control, for example, the partially known relationships may capture the following two properties of the input and output signals: (1) there exists a constant time delay τ between the input and the output in the system, and (2) the amplitude of the output is damped exponentially when the input signal becomes zero. The authors propose generalized-constraints neural networks which couple the artificial neural networks with partially known relationships using superposition, multiplication and composition. The parameter values of the generalized-constraints neural network model are obtained by minimizing an error function on a training dataset satisfying at the same time the partially known relationships to a certain degree of accuracy.

2.1.3 Using Domain Knowledge to Alter the Search Objective

This group encompasses approaches that incorporate the DK into the inductive bias which guides the search through the hypothesis space. This is achieved by modifying the learner's optimization problem and by introducing weights to the training errors (cost-sensitive learning).

Approaches to objective function adjustment by DK are present in the field of multi-task learning. In multi-task learning the learning task of interest (the main task) is addressed simultaneously with several other related learning tasks (extra tasks). Terms which measure the quality of the learned classifier on the extra tasks are added to the objective function. This way the learning process leverages not only information present in the training dataset, but also task-specific information hidden in the extra tasks. Jin and Sun (2008) use multi-task learning for face recognition. Face recognition is addressed together with the task of distinguishing face directions as an extra task. The training examples from the both tasks share the same feature representation. A single artificial neural network is trained to solve both tasks. The number of output-layer nodes in the artificial neural network equals the sum of distinct classes in both tasks. The error rate on distinguishing face directions is added to the objective function. Backpropagation is used to find the classifier with the best performance on both tasks.

Domain's invariance to transformations may be incorporated into the support-vector-machine optimization problem by either modifying the objective function or the constraints (Lauer and Bloch, 2008). A general framework for incorporation of transformation-invariance into the support-vector-learning optimization problem is presented by Loosli et al. (2005). Graepel and Herbrich (2004) present a formulation for support vector machines that finds an optimal separating hyperplane between trajectories. Shivaswamy and Jebara (2006) incorporate permutation invariance in support vector machines that finds an optimal separating hyperplane between sets of vectors.

Examples of altering the search objective by DK is also present in artificial-neural-network learning. One of the first such approaches is the Explanation-Based Neural Network (EBNN) algorithm (Thrun, 1996). Input to the EBNN algorithm are: (1) training examples, and (2) domain theory consisting of previously trained artificial neural networks. An example of domain theory is the knowledge-based neural network presented in Subsection 2.1.2. EBNN creates a fully connected feed-forward network by minimizing an objective function

which besides reducing the misclassification errors reduces the errors in training-example's derivatives computed using the domain theory.

Sabzekar et al. (2011) introduce a new formulation of support vector machines, Fuzzy Relaxed Constraints Support Vector Machines (fuzzy RSVM), which enables specification of training example weights in support vector machines learning. The training example weights enable introduction of knowledge about the quality of the training data. The more noisy the data, the lower the weight of the corresponding example. The training example weights also enable specification of desired class precision. The more class misclassifications are allowed, the lower the weight of the examples of the class. The example weights are incorporated using fuzzy logic in the quadratic programming problem solved by support vector learning.

2.1.4 Using Domain Knowledge to Augment the Search

This group encompasses approaches that use DK to augment the set of legal steps in the search through the hypothesis space.

One of the first such approaches is the First Order Combined Learner (FOCL) algorithm (Pazzani and Brunk, 1993), an extension of the First Order Inductive Learner (FOIL) algorithm (Quinlan, 1990). Similarly to FOIL, FOCL learns a set of first-order Horn clauses using a sequential covering algorithm. Each Horn clause is created by a general-to-specific search which starts with the most general Horn clause. Several candidate specializations are generated in each search step and the Horn clause is extended using the specialization with the highest information gain relative to the training dataset. Unlike FOIL which specializes the Horn clauses only by addition of one literal at a time, FOCL considers also addition of clauses present in domain-theory relations. Consider as example learning illegal states on a chess-board having a white king, white rook and black king using the predicates *between(X, Y, Z)*, *adjacent (X,Y)* and *equal(X,Y)*, and a domain theory relation which states that a state is illegal if a king attacks a king:

$$\begin{aligned} &illegal(white\_king_{rank}; white\_king_{file}; white\_rook_{rank}; white\_rook_{file};\\ &\quad black\_king_{rank}; black\_king_{file}) \leftarrow king\_attacks\_king(white\_king_{rank}; white\_king_{file};\\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad black\_king_{rank}; black\_king_{file}). \end{aligned}$$

$$\begin{aligned} &king\_attacks\_king(white\_king_{rank}; white\_king_{file}, black\_king_{rank}; black\_king_{file})\\ &\quad \leftarrow adjacent(white\_king_{rank}; black\_king_{rank}), adjacent(white\_king_{file}, black\_king_{file}). \end{aligned}$$

Unlike FOIL which chooses a specialization from the predicates *between(X, Y, Z)*, *adjacent (X,Y)* and *equal(X,Y)*, and their negations, FOCL considers also addition of whole relations present in the domain theory, such as $adjacent(white\_king_{rank}; black\_king_{rank})$, $adjacent(white\_king_{file}, black\_king_{file})$ in the given example.

FOIL and FOCL belong to the field of inductive logic programming (ILP), a broad category of algorithms that generates logical theories using both training examples and background knowledge (Lavrač and Džeroski, 1993). Other examples of inductive logic programming algorithms include Aleph (Srinivasan, 2013) and Progol (Muggleton, 1995). ILP needs a complex corpus of background knowledge for successful classifier learning. Because articulating the background knowledge can be difficult to non-ILP experts, it can also be generated automatically from an expert-provided explanations about why specific examples are positive or negative in a simple relevance language (Walker et al., 2011).

2.2 Interactive Data Mining

Interactive data mining also explores methods for concept learning using both prior DK and concept examples. Compared to the previously described methods, interactive data mining emphasizes the importance of human-computer interaction during classifier generation. While computers are capable of manipulating large volumes of data and performing complex operations, humans are crucial for selecting alternatives, planning and coping with unexpected situations. Zhao (2009) stresses that the learning-process success depends not only on how intelligent the user is or how efficient the algorithm is, but also on how well these two parts interact.

Active learning is a group of supervised learning methods where human-computer interaction contributes to iterative training-set improvement (Sun and Hardoon, 2010; Zhang and Sun, 2010; Dasgupta, 2011). Certain learning domains contain a large number of examples only few of which are labeled. In image classification one has access to many unlabeled images, however labeling them (e.g., as city images or landscapes) is costly as a person requires a considerable amount of time to perform this task. Active learning aims at reducing example-labeling cost by iteratively querying the user to label only examples whose label is "the most" beneficial for the learning problem. The human-computer interaction starts with classifier induction from the available labeled training examples. The classifier is then used for selecting examples to be labeled. The novel labeled examples are added to the training dataset, the classifier is reinduced and new examples are selected for labeling. The process is iterated until satisfiable classifier performance is achieved.

Stumpf et al. (2009) present user co-training, an approach to introducing user's classifier-content suggestions in the learning process. Similarly to co-training (Blum and Mitchell, 1998), user co-training employs two classifiers in the learning process each of which has its own, specific "view" on the data. It creates one ML classifier using the available labeled examples, while the second one is created purely from the provided user feedback. Unlabeled examples are used for improving the ML classifier in an iterative process consisting of addition of the most confidently classified unlabeled examples by both the ML and user-feedback classifiers to the training dataset and reinduction of the ML classifier. Stumpf et al. (2009) apply user co-training to an e-mail classification problem. The Naïve Bayes algorithm (Mitchell, 1997) is used for inducing the ML classifier. The user examines e-mail messages together with the class value assigned by the Naïve Bayes classifier and a list of keywords that according to the classifier influence the decision the most (keywords assigned the highest positive and negative weights by the Naïve Bayes algorithm). User feedback contains agreement with the provided keywords, irrelevant keyword indications and suggestions for keyword weight change. Such user feedback is transformed to a user-feedback classifier, which for each e-mail class holds a vector of words v_{class} which are designated as characteristic for the class in the user feedback. Given an unlabeled e-mail message, the user-feedback classifier assigns it to the class for which the word intersection between v_{class} and the e-mail message is the largest. The classification confidence equals the number of words in the intersection.

Visual data mining (Simoff et al., 2008) is another paradigm that emphasizes human-computer interaction in the knowledge discovery process. Humans posses visual pattern recognition skills able to detect changes in shape, color and motion of objects. Visual data mining uses data visualization as an communication channel between the human and the computer leveraging the human visual pattern recognition skills in the knowledge discovery process. Interactive decision tree construction algorithms (Liu and Salvendy, 2007; Poulet and Do, 2008) enable users to manually create decision trees. For each tree node, a visualization of the attributes' split together with numeric quality estimates are presented to the

user who selects the node split. Caragea et al. (2008) present an approach to coupling data visualization with the support vector machines algorithm. Tour-based methods are used for visualizing the separation boundary and the class structure of the support-vector-machines output. The proposed visualization guides the user in the process of selecting the output classifier from a set of candidates generated interactively by varying the attribute set and/or the input parameter values of the algorithm.

Osei-Bryson (2004) proposes usage of multi-criteria decision analysis for examination of the space of decision tree classifiers. The proposed approach empowers data mining analysts to perform a thorough experimentation and analysis of the decision-tree hypothesis space without being overwhelmed by the task of analyzing a significant number of decision trees. It uses a weighting model to compute an overall quality value of a decision tree evaluated by multiple performance criteria, such as accuracy, simplicity, stability and discriminatory power. The data mining analysts provide the intervals in which the weights belong and linear programming is used to find the weight values for which the decision tree quality is maximal. Only non-dominated decision trees are listed to the user sorted according to their quality in decreasing order.

Vidulin and Gams (2011) propose Human-Machine Data Mining (HMDM), an interactive method for extracting credible classifiers and relations in complex domains. The approach introduces a combination of human understanding and raw computing power for smart examination of parts of the hypothesis space where the most credible classifiers are. Initially, a set of classifiers are generated by human-understandable data mining algorithms (e.g., decision trees) by varying the algorithm parameter values. They are examined by the user who selects one or several interesting classifiers. The patterns in the selected classifiers are further examined to check their credibility. Two procedures are applied for this purpose: (1) remove attribute procedure which determines high quality attribute combinations, and (2) add attribute procedure which examines attribute redundancy. The classifiers and patterns that pass the credibility check are stored.

2.3 The Dissertation's Contribution in the Context of the Related Work

CDKML belongs to the group of approaches that uses prior DK to initialize the hypothesis or hypothesis space. In contrast to the other approaches in this group, it uses interactive data mining to initialize the classifier (CDKML's initialization phase) after which optimization algorithms are applied for determining the optimal general-purpose classifier's parameter values in a hypothesis space restricted by DK (CDKML's refinement phase) as well as for determining the optimal deployment-specific classifier's parameter values (CDKML's online adaptation phase).

The CDKML's initialization phase (concept pattern extraction) is primarily based on the ideas for smart examination of parts of the hypothesis space with the most credible patterns proposed by Vidulin and Gams (2011). CDKML's refinement phase (determination of the most suitable general-purpose pattern-parameter values) is primarily based on the ideas of the learning classifier systems (Holmes et al., 2002) – an approach to evolving classifiers according to their expected reward from the environment. To the best of our knowledge, the combination of the two phases and their application to domains for which a limited amount of concept examples is available is novel.

The CDKML's online adaptation phase (determination of the most suitable deployment-specific pattern-parameter values) is primarily based on the ideas of the Markov decision processes (Russell and Norvig, 2010). Markov decision processes are used in sequential decision-

making domains. The online adaptation phase resembles sequential decision-making tasks as pattern-parameter adaptation is performed in a step-by-step manner until satisfactory, deployment-specific pattern-parameter values are reached. We are unaware of any work which utilizes Markov decision processes for classifier adaptation according to user feedback.

3 Machine Learning and Expert Domain Knowledge

This chapter analyzes why a combination of expert DK and ML offers the possibility to extract reliable concept classifiers from a limited amount of task examples. First, we formalize ML. Second, we present the characteristics of learning tasks that would benefit from the incorporation of DK. Third, we formalize expert DK and present ways in which it may influence the learning process to improve generalization. We conclude the chapter by formalizing learning using both concept examples and expert DK.

3.1 Inductive Machine Learning

The dissertation addresses classification, a subclass of supervised learning (also referred to as concept learning) (Mitchell, 1997), which concerns learning an object categorization from labeled examples. *A labeled example* is a pair <*attributes*, *class*>, where *attributes* refers to a set of attributes that capture the properties of the observed example and *class* specifies the concept the example belongs to. The value of *class* is discrete. It is assumed that the class depends on the attributes, i.e., $class = f(attributes)$, where the function f (*the target function*) is unknown. The classifier inferred by classification learning (also referred to as *a hypothesis* or *a model*) is an approximation of the target function *f*. The labeled examples used for learning are called *training examples*. Classification learning aims at finding *a general classifier*, which: (1) is consistent with the training examples, i.e., it correctly predicts the class value for all training examples; the *training error* represents the fraction of training examples incorrectly classified by the learner, and (2) is a good predictor, i.e., is able to determine the correct membership of examples whose class is unknown; the *true error* represents the fraction of all concept members (even unseen ones) that is incorrectly classified by the learner.

Since a set of training examples is all information the learner has, most classification-learning algorithms formulate the learning problem as a search through *a hypothesis space*, i.e., the set of all candidate classifiers expressible by the learner's language, aimed at finding a classifier with minimum training error (Možina, 2009). This is a broad class of learners called *agnostic learners*. The term agnostic, meaning "not known", emphasizes the fact that the learning algorithm designer may have no prior knowledge about the target function. The algorithm J48 used in the introduction is one such example. Agnostic learners assume that a classifier that approximates a target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples (Mitchell, 1997). This is a fundamental assumption of inductive learning in general.

How do we select the classifier that is the best predictor when more than one classifier in the searched hypothesis space have the same training error? Each classification-learning algorithm has *an inductive bias*, a policy by which the learner generalizes beyond the training examples (Mitchell, 1997). It is a set of assumptions that, if considered, enable deductive derivation of the class y_i a classifier would assign to a new example from the available training examples and the example description x_i. The inductive bias of the algorithm J48, for example, states that smaller trees are preferred over larger ones and trees that

place attributes with high information gain on the training examples close to the root are preferred over those that do not (Mitchell, 1997).

The dissertation formally specifies the agnostic learner's task akin to the formalization used by Možina (2009). Given:

- E, a set of training examples where each $e \in E$ is a labeled example of the form $<attributes_e,\ class_e>$,
- B, an inductive bias

agnostic learners find a classifier H such that:

$$\forall e, H \wedge attributes_e \vdash class_e$$

$$B \wedge E \vdash H \tag{3.1}$$

The symbol $\vdash$ represents logical derivation.

What distinguishes successful learning from unsuccessful ones? The dissertation adopts the probably approximately correct learning framework (Mitchell, 1997) which formalizes successful learning. First, in order to be successful, a learner does not have to output a classifier with zero true error. As stated previously, more than one classifier in the hypothesis space may have the same minimum training error. Because training examples are all information the learner has, it cannot pick the right classifier with certainty in such case. Therefore, a successful learner is required to output a classifier with true error bounded by some constant ϵ, that can be made arbitrarily small. Second, in order to be successful, a learner does not have to output a classifier with true error smaller than ϵ for every sequence of randomly drawn training examples. When the training examples are drawn randomly, there is a nonzero probability that the training dataset contains accidental patterns which are not characteristic for the learned concept. In the introduction, we presented an example of such pattern. Therefore, a successful learner is required to fail to learn a good classifier (classifier with true error smaller than ϵ) with probability bounded by some constant δ, that can be made arbitrarily small. Finally, the learner must find a classifier in time polynomial in $1/\delta$, $1/\epsilon$ and the number of attributes used to describe each example.

How big is the difference between the training error and the true error of agnostic learners? The following formula provides the true-error bound of agnostic learners (Mitchell, 1997):

$$Pr[(\exists h \in H)(error_{true}(h) > error_{train}(h) + \epsilon)] \leq |H|\, e^{-2m\epsilon^2} = \delta \tag{3.2}$$

i.e., with probability $1 - \delta$

$$error_{true}(h) < error_{train}(h) + \sqrt{\frac{\ln |H| + \ln \frac{1}{\delta}}{2m}} = error_{bound} \tag{3.3}$$

where $|H|$ denotes the hypothesis space size and m is the number of training examples. The true-error bound grows proportionally with the square root of the natural logarithm of the hypotheses-space-size $|H|$ and $1/\delta$, and inversely proportional with the square root of the number of training examples m. Equation (3.3) is a general estimate of the true-error bound of agnostic learners, independent of the specific algorithm used for selecting the best classifier.

How many training examples m would an agnostic learner need in order to approximately correctly learn a target concept for a desired ϵ and δ? The minimum number of training examples needed for probably approximately correctly learning a concept can be derived from Equation (3.2):

$$m \geq \frac{1}{2\epsilon^2}(ln\,|H| + ln(1/\delta)) \tag{3.4}$$

Definition 3.1: *The training examples are of limited amount* if a target concept cannot be probably approximately correctly learned using them.

3.2 Is the Training Data Enough for Successful Learning?

This section discusses the characteristics of learning from a limited amount of training examples. First, we describe a set of error rates used for estimating a classifier's performance. Second, we present the performance of posture-recognition decision-tree classifiers induced from a limited amount of training examples. Third, we discuss what error features indicate that the training data is possibly not enough for successful learning. Finally, we estimate the number of training examples needed for probably approximately correctly learning to recognize postures in the used setting.

We estimate a classifier's performance using the following four error rates:

1. training error ($error_{train}$) – error rate on the training examples:

$$error_{train} = \frac{number\ of\ incorrectly\ classified\ training\ examples}{number\ of\ training\ examples}$$

2. separate-training-and-test-set error rate ($error_{test}$) – error rate on test examples recorded separately from the training examples:

$$error_{test} = \frac{number\ of\ incorrectly\ classified\ test\ examples}{number\ of\ test\ examples} \tag{3.5}$$

3. 10-fold-cross-validation error rate – considers dividing the training examples into ten subsets with equal number of examples. Ten iterations are performed. In each iteration $i \in [1, 10]$ the classifier is induced from the examples in 9 out of the 10 subsets, after which the classifier's test error ($error_{test}(i)$) is calculated on the examples in the 10^{th} subset which is not seen during learning. The 10-fold-cross-validation error rate ($error_{CV}$) is the average test error of the 10 iterations.

$$error_{CV} = \frac{1}{10}\sum_{i=1}^{10} error_{test}(i)$$

4. true-error bound ($error_{bound}$) – the maximum true error $error_{true}$ calculated using Equation (3.3).

The classifier's performance is analyzed using a graph which plots the four error rates per dataset size for training dataset sizes in a range [*min_set*, *full_set*]. The maximum training dataset size *full_set* is the number of available training examples. In the analysis carried out in this dissertation, we set *min_set* to 50. Depending on the domain other *min_set* values may also be used as long as the dataset-size range is large enough to capture the shape of the error curves. For each training dataset size, N training subsets are randomly drawn from the available training examples. In the analysis carried out in this dissertation, we set N to 5. Depending on the domain's error variation, other values may also be used. A classifier is created for each of the N training subsets and its four error values are computed. The average value of each of the four error rates is added to the graph.

Figure 3.1 depicts the four error rates per dataset size for decision-tree classifiers induced from posture examples with the J48 algorithm in Weka (Hall et al., 2009).

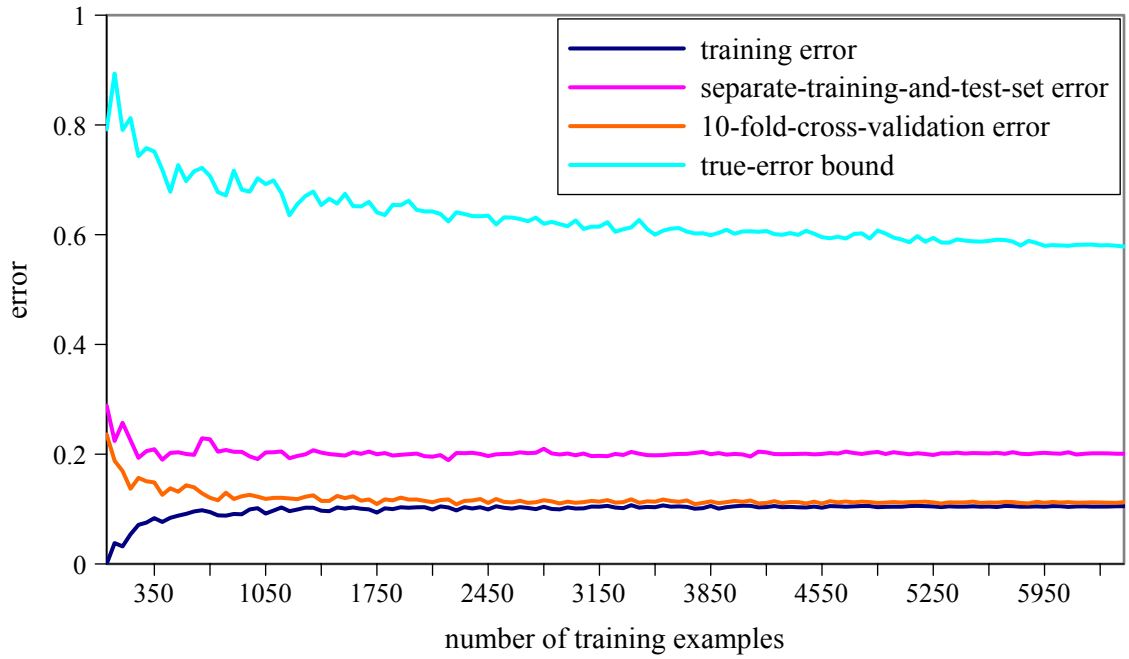

Figure 3.1: Classifier's performance estimation. Comparison of the training error, 10-fold-cross-validation error, separate-training-and-test-set error and the true-error bound of ML posture-recognition classifiers.

A posture example was a pair <*attributes*, *class*>, where *attributes* contained information about the position and velocity of a person's chest, left ankle and right ankle as well as the total, the z direction and xy direction distance between these body parts (18 attributes), while *class* was the person's posture (standing, sitting, lying, falling, slowly going down and standing up). The posture examples were recorded in two phases. The first phase contains 6435 clear-case posture examples (e.g., normal walking, going to bed, fast forward falls). The second phase contains 59652 examples which, besides clear-case posture examples, cover several kinds of falls, and examples of imitation of walking and lying of people with health problems, such as Parkinson's disease and hemiplegia.

The decision-trees were induced with the default J48 Weka algorithm parameter values (Hall et al., 2009) and the minimum number of examples per leaf equal to 2 % of the training-subset size. We tested the J48 performance with the minimum number of examples per leaf in the range from 0 % to 10 % of the training-subset size, while keeping the default Weka values for the rest of the algorithm parameters. The minimum number of examples per leaf was set to 2 % because it had the lowest 10-fold-cross-validation and separate-training-and-test-set error.

The training and 10-fold-cross-validation errors were computed using the posture examples in the training subsets which were extracted from the first phase recordings, while the separate-training-and-test-set error was computed using the posture examples in the second phase recordings.

In order to compute the true-error bound, an estimate of the decision-tree hypothesis space size is needed. Guestrin (2005) and Pichuka et al. (2007) present approaches to estimating the hypothesis space size of binary trees. Since we address multi-class problems, we extended their breadth-based approach to estimating the decision-tree hypothesis space size to multi-class problems. The approach is outlined in Algorithm 3.1:

Algorithm 3.1: Calculating the hypothesis space size of decision-tree classifiers.

DECISION_TREE_HYPOTHESIS_SPACE_SIZE(training dataset *data_training*)

```
1   Transform the continuous attributes and the multi-label discrete attributes
            in data_training to binary attributes;
2   J48 → Pruning = false;
3   J48 → Min_Num_Examples_Per_Leaf = 1;
4   classifier = J48 → buildClassifier(data_training);
5   max_leaves = classifier → Get_Num_Leaves();
6   hypothesis_space_size = 0;
7   num_classes = data_training → Get_Num_Classes();
8   num_atts = data_training → Get_Num_Attributes();
9   FOR (num_leaves=num_classes; num_leaves ≤ max_leaves; num_leaves++)
    DO
10    hypothesis_space_size +=
            NUM_DECISION_TREES(num_leaves, num_atts, num_classes);
    END DO
11  return hypothesis_space_size
```

Step 1: The continuous attributes are discretized and represented by a set of binary attributes contributing to a finite decision-tree hypothesis space size. The decision-tree hypothesis space size is infinite when learning from training examples with at least one continuous attribute because a continuous-attribute node split can be placed at an infinite number of places. This step is required because Equation (3.3) (the true-error bound estimate) holds for finite hypothesis spaces. On the presented posture-recognition task, we applied the information entropy minimization technique proposed by Fayyad and Irani (1993) to discretize the continuous attributes. Out of the 18 continuous attributes, 213 binary attributes were created. Besides the continuous attribute discretization, multi-label discrete attributes need to be converted to a set of binary attributes due to the requirements posed by the function NUM_DECISION_TREES (explained below).

Steps 2–5: Determine an upper bound of the decision-tree leaf number (*max_leaves*) in the hypothesis space. An unpruned decision tree with minimum number of examples per leaf set to 1 (*tree_unpruned*) is induced from the training examples. This decision tree has the lowest training error (zero if there are no conflicts between training examples). Because out of two decision trees which have the same training error the simpler is preferred, the leaf number of *tree_unpruned* poses an upper bound on the leaf number in the hypothesis space.

Steps 6–11: The hypothesis space size is computed. First, the number of binary attributes (*num_atts*) and the number of classes (*num_classes*) present in the training examples is retrieved. The minimum decision-tree leaf number (*min_leaves*) in the hypothesis space is set to *num_classes*, because a decision tree having less leaves than the number of classes cannot have the minimum training error. Then, the number of distinct decision trees with $num\_leaves \in [min\_leaves, max\_leaves]$ leaves, where each leaf represents one of *num_classes* classes and each node checks one of *num_atts* binary attributes is summed to obtain the hypothesis space size.

Number of distinct decision-tree classifiers with *num_leaves* leaves, where each leaf represents one of *num_classes* classes and each node checks one of *num_atts* binary attributes (Algorithm 3.2) used in Step 10 in Algorithm 3.1 is computed as follows:

Steps 1–2: Checks the validity of the input parameter values.

Algorithm 3.2: Calculating the number of distinct decision-tree classifiers with *num_leaves* leaves, where each leaf represents one of *num_classes* classes and each node checks one of *num_atts* binary attributes.

NUM_DECISION_TREES(number of tree leaves *num_leaves*, number of binary attributes *num_atts*, number of classes *num_classes*)

```
1   IF ((num_leaves == 0) OR (num_atts < (num_leaves – 1))) %invalid input
2     return 0;
    END IF
3   IF (num_leaves == 1) %only class value
4     return num_classes;
5   ELSE IF (num_leaves == 2)
6     return num_atts * num_classes * (num_classes – 1);
    ELSE
7     num_trees = 0;
8     FOR (l=1; l < num_leaves; l++)
9       num_trees += num_atts
                * NUM_DECISION_TREES(l, num_atts – 1, num_classes)
                * NUM_DECISION_TREES(num_leaves – l, num_atts – 1, num_classes);
      END FOR
10    return num_trees;
    END IF
```

Steps 3–4: If the decision tree has only one leaf (always predicts the same class), the number of distinct decision trees equals the number of classes (*num_classes*).

Steps 5–6: If the decision tree has two leaves, then the number of possible leaf class assignments equals *num_classes* $*$ (*num_classes* – 1) because the leaves must represent two different classes. The number of possible leaf class assignments is multiplied by the number of attributes *num_atts* to obtain the number of distinct decision trees.

Steps 7–10: These steps compute the number of distinct decision trees having more than two leaves, i.e., *num_leaves* > 2. There are (*num_leaves* – 1) ways of distributing *num_leaves* leaves to the left and the right root-node subtrees, [*left_leaves*, *right_leaves*] = {[1, *num_leaves* – 1], [2, *num_leaves* – 2], ..., [*num_leaves* – 1, 1]}. The product of the number of root-node attributes *num_atts*, the number of distinct left subtrees and the number of distinct right subtrees for each leaf distribution is summed to obtain the number of distinct decision trees having *num_leaves* leaves.

The decision-tree true-error bound was computed using Equation (3.3), where the decision-tree hypothesis-space size was estimated using Algorithm 3.1 and Algorithm 3.2. The parameter δ was set to 0.05. We would like to note that Equation (3.3) may lead to weak error-bounds for large hypothesis spaces (Mitchell, 1997). Tighter error bounds leveraging the Vapnik-Chervonenkis dimension of the hypothesis space have been derived (Vapnik and Chapelle, 2000); however there is no explicit formula for calculating the Vapnik-Chervonenkis dimension of decision trees (Asian et al., 2009).

An analysis of the four error curves in Figure 3.1 suggests that the posture examples used for training may not be enough for successful learning. The training error is below 0.11 for the whole set of training dataset sizes – the induced decision-tree classifiers fit the training dataset relatively well. The 10-fold-cross-validation error is also relatively low (below 0.13 for training dataset sizes greater than 650 examples), indicating that the induced

classifiers are good predictors of the patterns present in the training dataset. However, there is a gap of approximately 0.10 points between the 10-fold-cross-validation error and the separate-training-and-test-set error. The separate-training-and-test-set error slightly decreases for training dataset sizes from 50 to 350 examples, after which it remains fairly constant at approximately 0.20. This indicates that the posture-recognition dataset on which the separate-training-and-test-set error is computed contains patterns that are not present in the dataset used for training. Thus, the increase in the training dataset size does not reduce the separate-training-and-test-set error. The true-error bound slowly decreases with the increase of the training dataset size and is above 0.57 for the whole set of training dataset sizes indicating that the probability that the best decision-tree classifier in the hypothesis space has a very misleading training error is high. A large gap between the 10-fold-cross-validation and the separate-training-and-test-set error as well as high true-error bound indicate a lack of training examples.

According to Equation (3.4), approximately 860000 training examples are needed for probably approximately correctly learning to recognize postures with $\epsilon = 0.05$ and $\delta = 0.05$.

3.3 Eliciting Expert Domain Knowledge in Inductive Machine Learning

Knowledge refers to acquaintance with or understanding of a science, an art or a technique gained through experience or association used to achieve a goal (http://www.merriam-webster.com). This definition lays two important features of knowledge: (1) knowledge is very much context-dependent; the knowledge an engineer uses to build a car is not much of use to a chemist developing new skin care products, and (2) knowledge is extracted from experience, i.e., past observations of events, as well as from association, i.e., recognized links between events.

Definition 3.2: Suppose we have an attribute space $A = \mathbb{R}^d$ and a finite set of classes C. Let a concept be defined by a target function $f : A \to C$. *Complete knowledge* of a concept is a function $g : A \to C$, such that $\forall a \in A, f(a) = g(a)$.

A classifier having complete knowledge of a concept is capable of specifying the correct class value for all concept examples. In the simplest case, it may be a hash table with $|A|$ entries, which provides the correct class value for each attribute vector $a \in A$. Learning, however, aims at extracting a set of patterns capturing a concept's regularities.

Humans posses two types of knowledge: (1) tacit knowledge – thoughts, feelings and emotions which are hard to formalize and share with other people, and (2) explicit knowledge – systematic and easily communicated facts or procedures about a domain (Pyle, 2003). Pan and Scarbrough (1999) divide explicit knowledge into three groups: (1) recipe knowledge – knowledge of procedures for accomplishing a goal (e.g., the steps required to create a classifier using a particular ML algorithm), (2) functional knowledge – an extension of recipe knowledge that includes knowledge about the settings in which a particular procedure is appropriate (e.g., knowing that when the numeric output is linearly dependent on a numeric input, linear regression should be applied), and (3) theoretical knowledge – an extension of functional knowledge that includes understanding of how each procedure step is performed and why it is appropriate in a given setting (e.g., knowledge of the learning process of a ML algorithm).

Definition 3.3: *Expert domain knowledge* is the understanding of a target concept that an expert possesses that is gained through experience and/or association, and that can be used for example categorization.

The dissertation addresses learning in domains for which expert DK is close to complete knowledge, i.e., the expert is capable of reliably categorizing the concept examples. It aims at utilizing explicit expert knowledge in classification learning.

There are two main approaches to combining expert DK and ML: (1) extracting explicit expert DK as a pre-learning step and incorporating it in the learning process of supervised learning algorithms, and (2) interactive data mining – eliciting expert DK in an interactive process composed of automatic pattern extraction and human control.

3.3.1 Incorporating Expert Domain Knowledge in the Learning Process of Inductive Machine Learning Algorithms

Yu (2007) presents a framework for incorporating explicit expert DK in the learning process of inductive ML algorithms. He adopts the view of learning as a search for optimal hypothesis in a hypothesis space (Section 3.1) and defines how expert DK may guide the search in order to enhance learning. DK may enhance learning from three aspects: (1) consistency, (2) generalization, and (3) convergence.

Consistency with domain knowledge

A classifier is consistent with an available set of training examples if it correctly classifies each training example. Agnostic learners aim at minimizing the training error, i.e., the inconsistency between the classifier and the training examples. If expert DK is represented by a set of relations, the inconsistency between a classifier and expert DK may be specified in similar terms:

$$error_{knowledge} = \frac{number\ of\ DK\ relations\ not\ captured\ by\ the\ classifier}{number\ of\ DK\ relations}$$

In argument-based ML (Možina, 2009) discussed in Subsection 2.1.2, for example, a classifier is consistent with expert DK if it captures at least one positive argument for all of the argumented examples and none of their negative arguments.

For the purpose of incorporating expert DK in classification learning, agnostic learners should minimize an objective function of the form:

$$w_t * error_{train} + w_k * error_{knowledge} \tag{3.6}$$

where $error_{train}$ represents the training error, $error_{knowledge}$ represents the expert DK error, and w_t and w_k are coefficients balancing the effect of the both components in the learning process.

Expert DK representing a set of relations concerning the learned concept brings additional information to the learning process. In many cases, this set of relations may be transformed to a set of additional training examples. Approaches to virtual-example creation from expert DK were presented in Subsection 2.1.1. Because an increase in the number of training examples decreases the true-error bound of the learned classifier as indicated by Equation (3.3), adding consistency with expert DK as an additional term in the objective function of agnostic learners contributes to improvement in the learned-classifier quality.

Generalization with domain knowledge

The learning algorithm searches for an optimal classifier in a hypothesis space. The hypothesis space needs to be large enough in order to encompass the target function; however increase in the size of the hypothesis space increases the true-error bound also (Equation (3.3)). Expert DK may influence generalization in two ways: (1) parts of the hypothesis

space inconsistent with the expert DK may be removed; reducing the size of the hypothesis space tightens the true-error bound of the learned classifier, and (2) specifying an initial classifier which will be refined by the learning algorithm in the process of searching; setting a meaningful initial classifier reduces the chance of the search to end in a "meaningless" local optimum.

Convergence with domain knowledge

Yu (2007) addresses three aspects of convergence: feasibility, efficiency and accuracy. Feasibility of the hypothesis space asserts if the hypothesis space encompasses an acceptable approximation of the target function. If analysis of the hypothesis space by expert DK indicates that it does not reach minimal requirements, learning will fail no matter how much effort is put to convergence. Efficiency is directly connected to the size of the hypothesis space and the followed searched path. Using expert DK to remove parts of the hypothesis space and/or to alter the search objective contributes to reducing the searched hypothesis space and/or the length of the search path, thus increasing the efficiency of the learning process. Expert DK may also provide a tradeoff between computational cost and accuracy. In this case, expert DK may be used for setting a stopping criterion when an acceptable classifier is reached.

3.3.2 Interactive Data Mining

Zhao (2009) addresses interactive data-mining system design. She points out two general problems of automatic data mining: (1) these approaches overemphasize the automation and efficiency of the system, neglecting user's subjective understanding, interpretation and evaluation, and (2) they lack explanations and interpretations of the extracted knowledge. Although computers are capable of manipulating large volumes of data as well as performing computation-intensive activities, cognitive functions such as evaluation of patterns' quality with respect to the learning domain is a user's task. Zhao (2009) proposes integration of users' DK into the knowledge discovery process in the following way:

- interactive data preparation – user's interaction with a visualization tool, allowing the user to examine the data distribution and attribute relationships;
- interactive data selection and reduction – the user selects the examples to be used in the induction phase, possibly restricting the set of attributes;
- interactive data preprocessing and transformation – the user specifies needed attribute transformations and discretization;
- interactive pattern discovery – pattern extraction by ML algorithms under human control. It is an iterative process for smart examination of large hypothesis spaces, such as the approach proposed by Vidulin (2012);
- interactive pattern evaluation – user's judgment of the quality and usefulness of the extracted patterns;
- interactive pattern representation – user's interaction with a visualization tool, allowing the user to examine the patterns extracted in the pattern discovery phase.

The knowledge discovery process is a loop which is iterated until satisfactory results are obtained.

The research in interactive data mining is primarily focused on providing support to the human expert in the process of constructing conclusions about a domain of interest,

not on creating good predictors. However, as the expert improves his/her understanding of a domain of interest, he/she may adjust the learning problem, add credible and remove obsolete patterns from the learned classifier, contributing to improvement of the quality of the learned classifier.

3.4 Inductive Machine Learning with Expert Domain Knowledge

The dissertation formally specifies learning with both training examples and expert DK akin to the formalization used by (Možina, 2009). Given:

- E, a set of training examples where each $e \in E$ is a labeled example of the form $<attributes_e, class_e>$,
- B, an inductive bias,
- K, expert DK

find a classifier H such that:

$$\forall e, H \wedge attributes_e \vdash class_e$$

$$K \wedge B \wedge E \vdash H \tag{3.7}$$

The symbol $\vdash$ represents logical derivation.

4 Motivating Domains

The development of CDKML was motivated by the following domains: behavioral cloning, posture recognition and fall detection. This chapter provides a description of each of them together with their requirements.

4.1 Behavioral Cloning[1]

Computer simulations of real-world processes and systems are widely used for the purpose of analysis, performance optimization and training. Examples include simulations for traffic analysis (Rossetti et al., 2002), evaluating evacuation scenarios (Sagun et al., 2011) and military training (Bohemia Interactive Australia, 2013). For credible results, such simulations need a realistic model of human behavior.

Human behavior can be modeled by means of behavioral cloning (Bratko and Urbančič, 1997). Behavioral cloning aims at learning human behavior patterns from task demonstrations by means of ML. This concept is also termed "learning from demonstration" (Argall et al., 2009) and "imitation learning" (Thurau et al., 2004). It has been successfully applied in a range of applications, such as development of Robosoccer software agents (Aler et al., 2009), helicopter controllers (Coates et al., 2010) and realistic game characters (Schadd et al., 2007).

Behavioral cloning is a challenging task. Human actions are influenced by context, by knowledge or experience of dependencies between actions, and by expectations of how the situation is going to develop (Hollnagel, 1993). Actions are purposeful. They are taken for the purpose of achieving a concrete goal, responding at the same time to critical events in the environment. In the Contextual Control Model, Hollnagel (1993) describes human action selection by four control modes: strategic (actions directed towards higher-level goal-achievement based on long-term planning), tactical (known procedures or rules for reaction to situations), opportunistic (actions triggered by salient features of the current context) and scrambled (random). Cognitive psychology research proposes that human behavior models should capture the underlying features of each control mode, situations when each control mode is dominant as well as the conditions under which control-mode transitions occur (Hollnagel, 2000).

In artificial intelligence research, human behavior is represented using the agent paradigm. In general, an agent is an autonomous entity that observes the environment through sensors and acts upon it using actuators (Russell and Norvig, 2010). Lettmann et al. (2011) present a basic, formal model of agents as a universal description of their properties, unifying existing work on the topic (Ferber, 1999; Wooldridge, 2009; Russell and Norvig, 2010). Agents act in an environment abstracted as a state transition system. Based on sensor input, they determine environment's state using a vision function that considers sensor noise. The central concept of the model is the agent's mental state. The mental state encompasses all concepts relevant to the agent's decision making: the agent's internal state, its sensed environment

[1] This section is based on the publication Mirchevska et al. (2012) and Mirchevska et al. (2013a).

state, cognition function (defines the agent's internal state based on its previous internal state and the sensed environment state), policy function (defines the action to be executed according to the agent's internal state) and internal state transition function (defines the agent's successive internal state based on its current internal state and executed action). Our view of agents is based on the model of Lettmann et al. (2011), with two extensions. First, the agent role concept is added to the model representing the agent's responsibilities in the multi-agent system. In dynamic environments, agents change roles to fulfill their goal the most effectively given the current environment state (Bežek, 2006). The policy function depends on the agent's role. Second, we extend the action definition by associating the triple (preconditions, parameters, effects) to each action (Bežek, 2006). To execute an action, its preconditions must be met. The way the actions are performed depends on their parameters. Effects define the environmental state when the action is terminated. This provides flexibility for action definition.

Definition 4.1: *An agent A* is a tuple $(S, S_A, I_A, R_A, M_A, A_A, v_A, adapt_A)$ where:

- S is a countable set of environmental states.
- S_A is a countable set of internal representations of the environment's states.
- I_A is a countable set of A's internal states.
- R_A is a countable set of A's roles.
- M_A is a countable set of A's mental states. The mental state of the agent, i.e., its "mind", contains all information relevant to the agent's decision making.
- A_A is a countable set of A's possible low-level actions, where each $a \in A_A$ is defined as $a = a$*(preconditions, parameters, effects)* containing at least one special action representing no action $a_0 = a_0$("always", $\emptyset$, "no change").
- $v_A : S \longrightarrow \Pi(S_A)$ is a probabilistic vision function that maps the current environmental state to a probability distribution over all possible internal representations of the environmental states.
- $adapt_A : M_A \longrightarrow M_A$ is an adaptation mechanism that translates the current mental state into another mental state.

Definition 4.2: *A single mental state* $m_A \in M_A$ is defined as a tuple $m_A = (s_A, i_A, r_A, \varrho_A, \pi_A, o_A, \tau_A)$ where:

- $s_A \in S_A$ is the internal representation of the environment's state of agent A.
- $i_A \in I_A$ is the current internal state of agent A.
- $r_A \in R_A$ is the current role of agent A.
- $\rho_A : S_A \times I_A \longrightarrow I_A$ is a cognition function that calculates the successive internal state of the agent based on the internal representation of environmental state s_A and the current internal state i_A.
- $\pi_A : I_A \times R_A \longrightarrow \Pi(A_A)$ is the agent's probabilistic policy function. It defines the probability of executing a low-level action $a \in A_A$ if the agent is in the internal state $i_A \in I_A$ and has role $r_A \in R_A$.

- o_A is an action selector mechanism (e.g., Roulette wheel selector) that selects an action for the agent based on the probability distribution over the possible actions $\Pi(A_A)$.
- $\tau_A : I_A \times A_A \longrightarrow I_A$ is a state transition function. It defines the successive internal state $i'_A \in I_A$ if the agent performs action $a \in A_A$ in the internal state $i_A \in I_A$.

The formal agent definition defines the policy function π_A as an interface to the agent behavior model, while the concrete application determines its implementation.

We addressed behavioral cloning (i.e., retrieving an agent's policy function π_A) in a serious game as a subtask of a larger system aiming at evaluating a person's rules of conducts. First, examples of the person's decisions were obtained by letting him/her interact with the serious game. Second, a behavior clone was created using the captured decision examples. Finally, the behavior clone was added to the serious game, and a set of situation developments were recorded for the purpose of identifying advantages and drawbacks in the person's rules of conduct.

We aimed at capturing a single-level policy function π_A having the form of a rule-based classifier. Rules of the following form were captured in the classifier:

IF *internal state* THEN *action* WITH certainty C

Internal state encompasses a set of features describing a person's internal state i_A causing the execution of the low-level action *action*. The internal state i_A captures the person's position, emotional state, interactions with other people and action history. The role r_A of the person is not part of the rule condition because we cloned the behavior of people having only one role. The certainty level captures the likelihood with which a person executes a concrete low-level action *action* when being in an internal state i_A. It captures uncertainty arising from the missing higher-level reasoning functions (strategic and tactical) which directly influence the choice of the low-level action. In addition to this, the certainty level enables dealing with uncertainties caused by incompleteness in the representation of the person's internal-state features, primarily uncertainties about the emotional and cognitive aspects of the person's internal state.

We divided the policy-extraction problem to two subtasks: (1) extraction of characteristic rule patterns, and (2) determination of optimal pattern-parameter values. Because we did not have any other information concerning the person's policy function beside the decision examples, we approached the first subtask by extracting patterns which repeat in ML classifiers. The intuition is that the more frequently a pattern appears, the more characteristic it is. Having the patterns extracted, a solution to the problem posed by the second subtask was needed: *how can optimal pattern-parameter values be obtained from the available decision examples?*

4.2 Posture Recognition

Posture and activity recognition received researchers' attention in ambient intelligence (Weber et al., 2010), a vision of a technology that will be invisibly embedded in people's natural surroundings to support them in the everyday activities providing improved safety and life quality. It is an inevitable subtask in many applications devoted to healthcare, well-being and sports (Avci et al., 2010). Since posture and activity recognition directly influences the performance of the application as a whole, they need to be reliable.

We addressed posture recognition during the development of the Confidence system (Confidence, 2012), a ubiquitous system for real-time monitoring of the elderly for the purpose of health-problem detection and prevention. The system contains three modules

devoted to health-problem detection: (1) short-term, focused on fall detection, (2) mid-term, focused on detecting mid-term behavior changes such as limping and slow moving, and (3) long-term, focused on detecting long-term behavior changes such as inactivity. Each of these modules uses the posture history of the monitored person in its reasoning. We aimed at distinguishing the following postures: standing, sitting, lying, falling, moving downwards, moving upwards and on all fours.

In Chapter 1, a decision tree induced from a posture dataset was presented indicating patterns which seam questionable with respect to DK. The decision tree is to a certain extent overfitted to the training examples gathered in the laboratory circumstances. The addition of new posture examples to the training dataset might improve the classifier quality; however, due to the wide variety of body configurations, it is difficult to record all possible situations and to obtain representative training dataset for posture classification. Since humans are good at imagining body postures, they may revise the posture patterns extracted by ML and additionally specify relevant patterns from DK. The combination of both ML and DK may provide a more reliable classifier.

We aimed at generating a rule-based classifier for posture recognition under expert supervision. Similarly to the behavioral-cloning domain, the classifier-generation problem was divided into two subtasks: (1) extraction of characteristic patterns, and (2) determination of optimal pattern-parameter values. The first subtask was performed by an expert. The expert examined posture patterns extracted by ML and decided which patterns need to be included in the posture-recognition classifier, possibly modifying them or adding patterns from DK only. This approach to pattern extraction should prevent insertion of patterns which are not characteristic for the learned concept in the classifier. However, *is an expert capable of selecting a comprehensive set of patterns of the learned concept?* Having the patterns extracted, a solution to the second subtask was also needed: *how can optimal pattern-parameter values be obtained from the available decision examples?*

4.3 Fall Detection[1]

Automatic fall detection is gaining in importance in the developed countries due to the rapid population aging. 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, 2012). 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 technical solutions can help the elderly live independently for longer and counteract reduced capabilities caused by age. The Confidence system (Confidence, 2012) is one such solution. Fall detection is one of its main tasks.

Robustness, capability of performing without failure under a wide range of conditions, is a must in the fall detection domain. Not only there is a wide range of fall types, but also falls are highly person dependent. Because falls may be caused by health problems, and may lead to injuries and even death, they have to be detected reliably. However, high recall should not be achieved at the cost of erroneous classification of non-fall events as falls. Such errors disturb users reducing the system acceptance rate. In Confidence, fall detection is addressed by a ML based and DK based approach. Each approach provides its own viewpoint on falls and it is the combination of the two that contributes to fall-detection robustness (Luštrek et al., 2011). The proposed method in the dissertation supported the development of the DK-based fall-detection classifier.

[1]This section is based on the publication Mirchevska et al. (2013b).

Three main challenges concerned this issue. First, a representative dataset for falls is difficult to obtain because of the variety of fall types, variations depending on the person, as well as ethical issues and injury dangers that prevent collecting large amounts of data from healthy people simulating falls or, even worse, the elderly. Second, generating a classifier that suits each person in each possible circumstance from the start is difficult. Confidence detects falls as situations in which a person is lying/sitting on the ground for a prolonged period of time. However, it is difficult to set a period of time to suit each person. For example, one person 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 system-related characteristics, such as 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 posture misclassifications directly influence the output of the fall-detection classifier.

Similarly to the posture-recognition domain, we aimed at developing a rule-based classifier for fall detection under expert supervision. The previously stated research questions are related to this domain, also: (1) *is an expert capable of selecting a comprehensive set of patterns of the learned concept*, and (2) *how can optimal pattern-parameter values be obtained from the available decision examples.* Additionally, fall detection posed one more requirement: online classifier adaptation. In the Confidence system user-specific data is collected online by means of user feedback. User feedback is obtained occasionally, and contains information about false negatives (i.e., the system did not detect a fall when there was one) and false positives (i.e., the system detected a fall when there was none). *How can we leverage user feedback for online classifier fine-tuning to user needs?*

5 CDKML – A Method for Combining Domain Knowledge and Machine Learning for Classifier Generation and Online Adaptation[1]

This chapter presents the main contribution of the dissertation – the CDKML method. It is a method for classifier generation from a limited amount of training examples (representing a subset of the possible real-life cases). The basic idea is to incorporate DK into the learning process, thus making up for information not captured in the training examples. CDKML (Figure 5.1) consists of three phases: (1) initialization, (2) refinement, and (3) online adaptation.

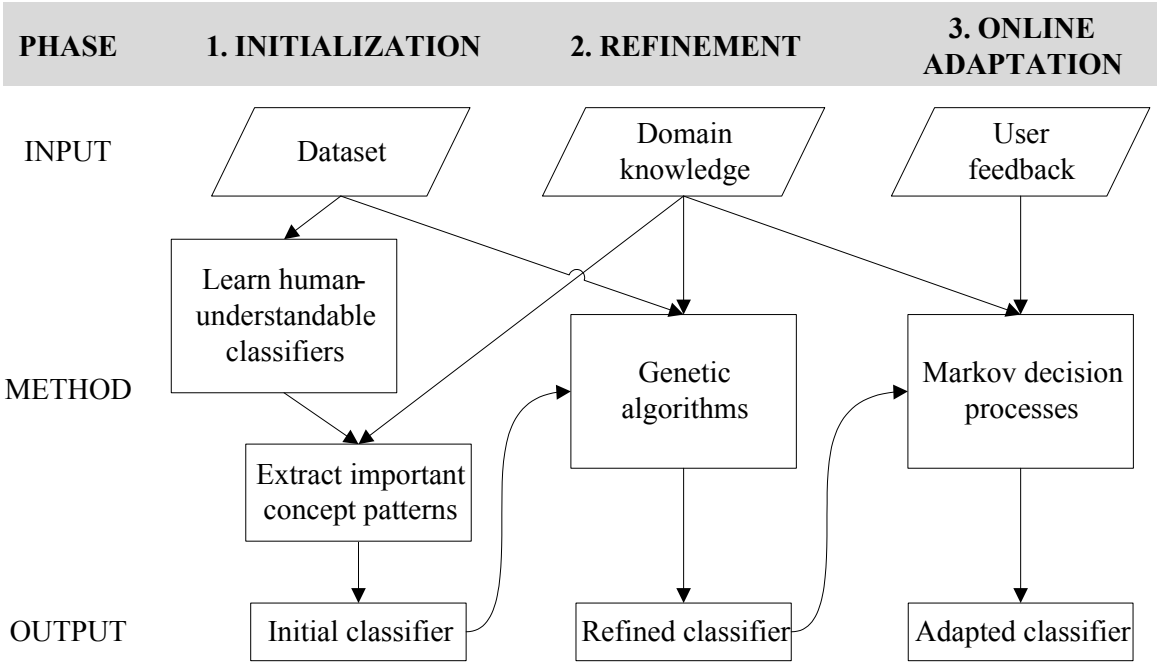

Figure 5.1: Schema of the proposed method for combining DK and ML for classifier generation and online adaptation (CDKML).

The aim of the initialization phase is to extract a comprehensive set of concept patterns supported both by the available training examples and by DK. Input to the initialization phase are human-understandable ML classifiers and patterns present in DK. An expert creates an initial classifier by selecting patterns present in the ML classifiers, by adding modifications of those patterns as well as by adding relevant patterns present in DK.

Having the classifier's patterns, the aim of the refinement phase is to determine the most suitable general-purpose pattern-parameter values. Each pattern implicitly represents a class-boundary segment whose layout (e.g., position, length) is specified by the pattern's parameter values. The optimal layout of the class-boundary segment greatly depends on its

[1]This chapter is based on the publication Mirchevska et al. (2013b).

interconnection with the segments represented by the rest of the classifier's patterns. This interconnection is not captured in the initialization phase, where the pattern-parameter values are obtained separately either from a ML classifier or are estimated using DK. The problem of determining the most suitable general-purpose pattern-parameter values is defined as an accuracy maximization search through the parameter-value space. Input to the refinement phase are the initial classifier, the training examples on which classifier's accuracy is measured and DK which poses constraints on the search space. An optimization algorithm is used for finding the optimal parameter values. These values are inserted in the initial classifier to create the refined classifier.

The aim of the online adaptation phase is to find the most suitable pattern-parameter values for a particular system deployment (e.g., for a particular user). The adaptation is defined as a Markov decision process which leverages user feedback (considered as a reward signal). User 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) and false positives (i.e., the system detected the class of interest when there was none). DK specifies how user feedback is to be translated to state rewards. Adaptation is performed online after each received user feedback outputting an adapted classifier.

The CDKML method is based on the following assumptions:

1. Concepts can be comprehensively described by a relatively low number of rules.
2. A domain expert with the help of interactive data mining is able to specify a classifier that encompasses a comprehensive set of concept patterns.
3. The most suitable general-purpose pattern-parameter values may be obtained by maximizing the classifier's accuracy on the available training examples. Such parameter tuning is not prone to overfitting, because the classifier contains only patterns characteristic for the learned concept.
4. Occasional user feedback containing information about false positives (non-members of the concept class classified as concept members by the classifier) and false negatives (members of the concept class classified as non-members of the concept class by the classifier) may be used for online classifier adaptation.

We start this chapter by presenting the format of the classifiers created using CDKML. Then we provide a detailed presentation of the three CDKML phases: initialization, refinement and online adaptation. The presentation is accompanied with examples from the application of CDKML in the fall-detection domain.

5.1 The Classifier

Due to the requirements posed by the motivating domains, CDKML was applied to classifiers in the form presented in Figure 5.2. The classifiers consist of a set of rules of the form

IF *conditions* THEN *class* (*conf_rule*)

where the rule's confidence *conf_rule* is an indicator of the rule's certainty. Each rule is checked when an input example is presented to the classifier. The rules whose conditions hold for the given input example vote for their class value. The votes are weighted by the amount of the rule's confidence *conf_rule*. A conflict resolver collects the votes and determines the final class value. We use a maximum confidence conflict resolution strategy, i.e., the class of the rule whose vote is the highest (the most certain rule) is outputted.

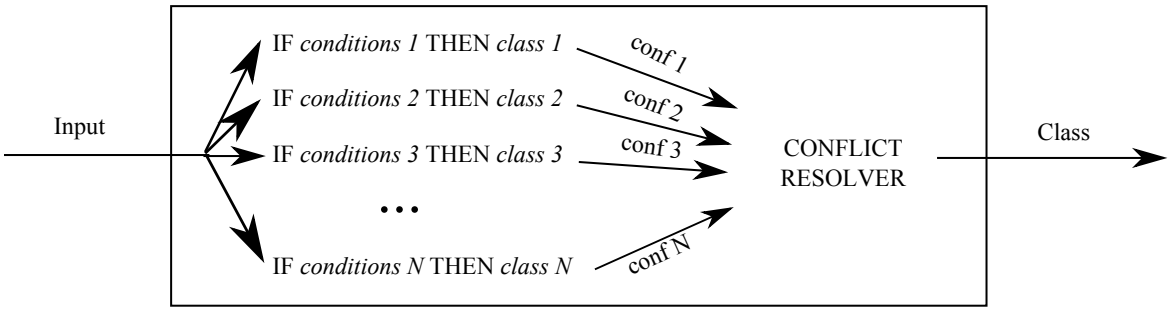

Figure 5.2: CDKML's classifier format.

Depending on the domain, other conflict resolution strategies may also be used (e.g., a predefined class priority list or a roulette wheel selector).

The rule's confidence measure should satisfy the following requirements:

- having all other criteria equal, a rule with a higher precision should have a higher confidence level. The precision is a measure of a rule's purity. Let Ex_{rule} represent a set of examples for which the rule's conditions *conditions* hold. Let $Ex_{correct}$ represent a subset of Ex_{rule} containing only examples whose class value equals the rule's class *class*. The precision is calculated as follows:

$$precision_{rule} = \frac{|Ex_{correct}|}{|Ex_{rule}|} \tag{5.1}$$

  where $|Set|$ represents the number of examples in the set *Set*.

- having all other criteria equal, a rule with a higher recall should have a higher confidence level. The recall is a measure of a rule's sensitivity. Let Ex_{class} represent a set of examples whose class value equals the rule's class *class*. Let $Ex_{correct}$ represent a subset of Ex_{class} containing only examples for which the rule's conditions *conditions* hold. The recall is calculated as follows:

$$recall_{rule} = \frac{|Ex_{correct}|}{|Ex_{class}|} \tag{5.2}$$

We selected the F1-score, a weighted average of precision and recall, for measuring the rule's confidence:

$$confidence_{rule} = \frac{2 * precision_{rule} * recall_{rule}}{precision_{rule} + recall_{rule}} \tag{5.3}$$

The confidence level is a value in the interval [0, 1] where higher values indicate a higher confidence level.

Figure 5.3 shows the confidence level of an example rule present in the fall-detection domain: "IF a person is lying on the ground for P_{lying} % of T_{lying} seconds THEN Fall". The rule's confidence was computed on a training dataset encompassing 40 fall and 40 non-fall events. As expected, the rule's confidence increases with the increase of both the T_{lying} and P_{lying} as long periods of lying on the ground are associated with a high probability of a fall.

This classifier form was chosen because it can be constructed manually or with the help of classification learning and modified by an optimization algorithm or Markov decision processes. We would like to note that CDKML is not bound to this specific classifier form; however, it requires a human-understandable form.

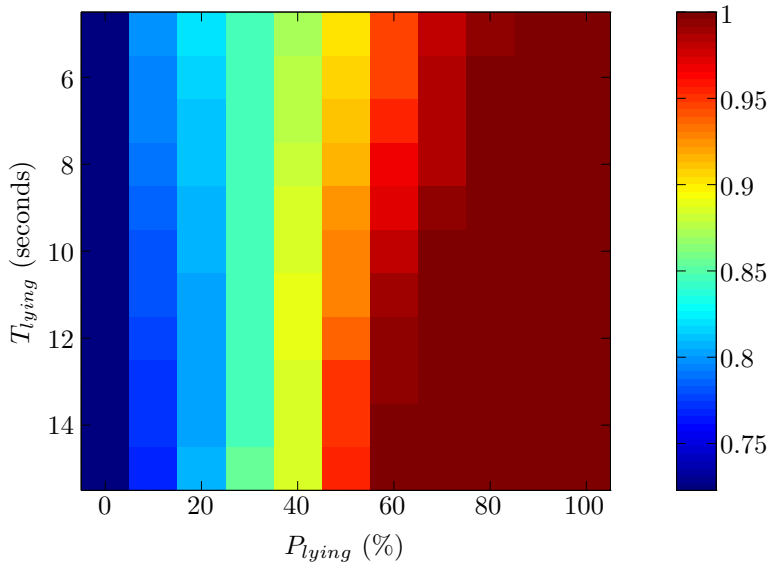

Figure 5.3: Visualization of the confidence level of the fall-detection rule "IF a person is lying on the ground for P_{lying} % of T_{lying} seconds THEN Fall".

5.2 Initialization

The aim of the initialization phase is to extract a set of concept patterns from ML classifiers and DK under expert supervision. ML is a source of novel, while DK of known concept patterns, both of which are needed for creating a comprehensive and reliable classifier. However, both sources may contain certain deficiencies. On the one hand, ML classifiers may contain patterns which, although representative of the available training examples, are not characteristic of the learned concept (Chapter 1). DK may spot such obsolete patterns. On the other hand, certain DK patterns may become questionable as learning progresses and novel knowledge is extracted. ML patterns may influence DK adjustment. Therefore, an expert specifies the set of patterns which constitute the initial classifier. The patterns are selected from the ML classifiers, DK or represent modifications of such patterns.

Algorithm 5.1: CDKML phase 1 – initialization.

INITIALIZATION(training examples Ex)

1 CL_{init} = empty set; //the initial classifier
2 $ALGORITHMS$ = {decision-tree and rule induction algorithms};
3 FOR EACH alg IN $ALGORITHMS$
4 create a set of ML classifiers on Ex by varying example attributes and alg parameter values;
5 explore rule patterns in the induced ML classifiers;
6 add ML patterns verified by DK to CL_{init};
7 add ML patterns adjusted by DK to CL_{init};
END FOR EACH
8 add DK patterns to CL_{init};
return CL_{init};

Algorithm 5.1 outlines the initialization phase:

Step 1: The initial classifier CL_{init} is initialized to an empty set.

Steps 2–7: The expert examines human-understandable classifiers induced from the available training data, adding ML patterns verified by DK or ML patterns adjusted by DK to the initial classifier CL_{init}. These steps provide an additional insight in the domain and may contribute to DK modification.

Step 8: The initial classifier CL_{init} is supplemented with patterns present in DK.

Algorithm 5.2: Decision-tree hypothesis-space examination.

DECISION_TREE_PATTERN_EXAMINATION(training examples Ex)

```
1   Out of Ex, create K training subsets Ex_i, i ∈ [1, K];
2   FOR EACH Ex_i DO
3     Induce a decision tree cls from Ex_i;
4     Explore the patterns in cls;
5     UNTIL significant drop in accuracy DO
6       Remove the root-node attribute and/or an attribute in the root descendents;
7       Induce a decision tree cls with the reduced attribute set from Ex_i;
8       Explore the patterns in cls;
      END UNTIL
    END FOR EACH
```

The fundamental part of the initialization phase is the examination of human-understandable classifiers. Interactive data mining methods, which provide smart hypothesis-space examination focused on the most promising parts, may be used for this purpose. The Human-Machine Data Mining method (Vidulin and Gams, 2011) and the multi-criteria decision analysis approach to evaluating decision trees (Osei-Bryson, 2004) are just two examples. The initial classifiers in the dissertation were mostly created by examining decision-tree classifiers as outlined in Algorithm 5.2:

Steps 1–2: Training subsets Ex_i, $i \in [1, K]$ are created from the available training examples Ex. If the available data is recorded for K people, for example, K training subsets can be created in each of which the data of one person is left out. Patterns characteristic for the learned concept should appear in the decision-tree classifiers induced from most of the training subsets Ex_i.

Steps 3–4: Explore patterns in the decision tree induced from a training subset Ex_i.

Steps 4–8: Explore patterns in decision trees induced from Ex_i using only a subset of the available example attributes. As presented in Section 3.1, the inductive bias of decision-tree induction algorithms, such as J48, prefers shorter trees over longer ones and trees that place attributes with high information gain on the training examples close to the root over those that do not. The decision-tree induction algorithms perform general-to-specific hill-climbing search through the space of possible classifiers outputting a single, best decision tree with respect to the inductive bias. However, other classifiers, even if they are somewhat weaker with respect to the inductive bias, may be interesting from the expert's perspective. To find them, we induce several decision trees with different attribute subsets. We consider removing the root-node attribute and/or attributes in the root descendents, with the aim of finding relevant hidden classifiers, until the classification accuracy of the resultant tree significantly drops.

The rule patterns extracted in the fall-detection domain are presented as an example. They are derived from the fact 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. The following rule patterns were included in the initial classifier:

1. IF falling activity within $T1_{fall}$ seconds AND the person is lying/sitting on the ground $P1_{activity}$ % of $T1_{activity}$ seconds AND the person is not moving $P1_{moving}$ % of $T1_{moving}$ seconds THEN fall;

2. IF falling activity within $T2_{fall}$ seconds AND the person is lying/sitting on the ground area afterward $P2_{activity}$ % of $T2_{activity}$ seconds THEN fall;

3. IF a person is lying/sitting on the ground for $P3_{activity}$ % of $T3_{activity}$ seconds AND the person is not moving $P3_{moving}$ % of $T3_{moving}$ seconds THEN fall;

4. IF a person is lying/sitting on the ground for $P4_{activity}$ % of $T4_{activity}$ seconds THEN fall.

The focus of the initialization phase is to obtain a comprehensive set of patterns under expert supervision. The expert also provides initial pattern-parameter values either from a ML classifier or from DK. Nevertheless, determining the most suitable pattern-parameter values is addressed separately in the CDKML's refinement phase.

5.3 Refinement

The aim of the refinement phase is to determine the most suitable general-purpose pattern-parameter values. Because the initial pattern-parameter values are obtained separately, they do not capture the interconnections between the patterns. In addition, estimating the pattern-parameter values using DK may be an issue. System-related features (e.g., the ability of the fall-detection system to correctly detect the lying/sitting posture) influence these values and need to be considered when determining them.

The refinement phase determines the pattern-parameter values using the training examples. It relies on the assumption that pattern-parameter values which contribute to accurate training-example classification, would provide reliable prediction over other unobserved examples. Overfitting is hopefully avoided because the patterns are specified by a domain expert. The refinement is defined as an accuracy maximization search through the parameter-value space, where accuracy is measured on the training examples. The search is performed using an optimization algorithm.

DK poses constraints on the pattern-parameter values (the search space) in this phase. 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 person to be immovable and to lie/sit on the ground to detect a fall, whereas the fourth rule type requires only a person 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. This relation between the required periods of lying/sitting on the ground in the rules should be added as a DK parameter-value constraint. 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 should also be added as a DK parameter-value constraint.

In the dissertation we use genetic algorithms, a stochastic optimization method, for finding the optimal pattern-parameter values with respect to training accuracy. Below we outline how they are used in the refinement phase.

In order to apply genetic algorithms, a fitness function needs to be specified. The fitness function is an individual's (solution's) quality measure. Algorithm 5.3 outlines the fitness function used in the dissertation. Input to the fitness function are an individual I whose fitness is to be determined, a classifier CL_{in} encompassing a set of patterns whose parameter values are provided in the individual I, the DK parameter-value constraints *Constraints* and training examples *Ex*. The values in I are assigned to the CL_{in}'s parameters thus creating the classifier CL. If CL violates the DK constraints *Constraints* zero, i.e., minimal fitness, is returned. Otherwise, the function outputs the accuracy of CL on the training examples *Ex*. The fitness value falls within the interval [0, 1], where higher values indicate higher fitness.

Algorithm 5.3: A classifier's quality estimator – CDKML's fitness function.

FITNESS_FUNCTION(an individual I, a classifier CL_{in}, DK parameter-value constraints *Constraints*, training examples *Ex*)

1 $CL = CL_{in}$;
2 assign the values in I to the CL's parameters;
3 IF CL violates *Constraints*
4 **return** 0;
ELSE
5 **return** ACCURACY(CL, *Ex*);
END IF

The refinement phase is outlined in Algorithm 5.4. Input to the refinement phase are the initial classifier CL_{init}, a set of DK parameter-value constraints *Constraints* and training examples *Ex*. Additionally, the genetic-algorithm parameter-values are given: a population size *POPULATION_SIZE*, a crossover rate *CROSSOVER_RATE*, a mutation rate *MUTATION_RATE*, and a target accuracy *TARGET_ACC* and a maximum number of iterations *MAX_ITERATIONS* as stopping criteria. It is performed as follows:
Steps 1–5: An individual I_{base} representing CL_{init}'s parameter values is created. We use the Pittsburgh approach, i.e., each individual in the population represents one possible solution. The individual is a vector containing the parameter values of all patterns in the classifier. For example, if the classifier contains 8 patterns with 4 parameters each, the individual is 32 elements long. The elements may be discrete or continuous. Constraints on their values are specified in *Constraints*.
Steps 6–11: An initial population P is created. I_{base} is added to the initial population. It is a base for creating the rest of individuals I_i. I_i's gene-values $I_i \rightarrow$ Gene(g) are selected randomly from the interval $[(1 - frac) * I_{base} \rightarrow \text{Gene(g)}, (1 + frac) * I_{base} \rightarrow \text{Gene(g)}]$, where *frac* is selected by the expert.
Steps 12–17: The population is evolved using genetic operators with the genetic parameter-values given as input. Individual's fitness is computed as presented in Algorithm 5.3. Elitism is used, i.e., the best individual I_{best} is always transferred to the new population.
Steps 18–20: The refined classifier CL_{ref} is initialized to CL_{init}. Its parameters are assigned to the values in I_{best}.

One of the main difficulties faced in applying genetic algorithms is the determination of the appropriate algorithm parameter-values, such as the population size, the crossover rate, the mutation rate and the stopping criteria (in our case, a target accuracy and a maximum number of iterations). The value of these parameters influences the size of the explored search space and the search efficiency, determining whether an optimal or near-optimal so-

Algorithm 5.4: CDKML phase 2 – refinement

REFINEMENT(initial classifier CL_{init}, DK parameter-value constraints *Constraints*, training examples *Ex*, genetic-algorithm parameter-values *POPULATION_SIZE*, *CROSSOVER_RATE*, *MUTATION_RATE*, *TARGET_ACC*, *MAX_ITERATIONS*)

1 //Create an individual I_{base} representing the CL_{init} parameter values;
2 I_{base} = empty vector;
3 FOR EACH *pattern* IN CL_{init}
DO
4 put *pattern*'s parameter values to a single vector Vec_r;
5 append Vec_r to I_{base};
END FOR
6 //Create an initial population P
7 P = empty set;
8 add I_{base} to P;
9 FOR $i = 2$ to *POPULATION_SIZE*
DO
10 create an individual I_i by random changes of I_{base};
11 add I_i to P;
END FOR
12 $iter = 0$;
13 $I_{best} = P \rightarrow$ GetFittestIndividual();
14 WHILE (($I_{best} \rightarrow$ Get_Fitness() $<$ *TARGET_ACC*) AND ($iter <$ *MAX_ITERATIONS*))
15 $iter = iter + 1$;
16 $P \rightarrow$ Evolve(*CROSSOVER_RATE*, *MUTATION_RATE*, *ELITISM* = true);
17 $I_{best} = P \rightarrow$ GetFittestIndividual();
END WHILE
18 $CL_{ref} = CL_{init}$;
19 assign the values in I_{best} to the CL_{ref}'s parameters;
20 **return** CL_{ref};

lution will be reached, as well as whether the solution will be found efficiently (Eiben and Smith, 2003). According to Smit and Eiben (2009), there are no algorithms for parameter tuning that are widely accepted in the field of evolutionary algorithms, a subclass of which are the genetic algorithms. We use experimental comparison on a limited scale for determining the population size, crossover rate and mutation rate. In the posture-recognition and behavioral-cloning experiments, for example, we check the following parameter-value combinations: population size of 100 and 150 individuals, crossover rate of 50 %, 70 % and 90 %, and mutation rate of 0 %, 10 %, 20 % and 30 %; this set may be adjusted depending on the domain. The stopping criteria are specified by the expert because these parameter values depend on what a satisfactory solution is. The approach to tuning the population size, crossover rate and mutation rate is outlined in Algorithm 5.5:

Steps 1–4: The quality of each parameter-value combination *pop_size*, *crossover* and *mutation* is measured on validation examples Ex_{val}. In the behavioral-cloning domain, for example, 9 game recordings were available during training. For the purpose of parameter

Algorithm 5.5: Tuning the genetic-algorithm parameter values in CDKML's refinement phase.

GA_PARAMETER_TUNING(an initial classifier CL_{init}, DK parameter-value constraints *Constraints*, training examples Ex_{train}, validation examples Ex_{val}, a target accuracy *TARGET_ACC*, a maximum number of iterations *MAX_ITERATIONS*, minimum population size *PS_MIN*, maximum population size *PS_MAX*, population-size step *PS_STEP*, minimum crossover *CO_MIN*, maximum crossover *CO_MAX*, crossover step *CO_STEP*, minimum mutation *MU_MIN*, maximum mutation *MU_MAX*, mutation step *MU_STEP*)

```
1   IF Ex_val not specified
    THEN
2     Ex = Ex_train → GetRandom((1/4) * SIZE(Ex_train));
3     Ex_val = Ex;
4     Ex_train = Ex_train \ Ex_val;
    END IF
5   best_crossover = unknown;
6   best_mutation = unknown;
7   best_population_size = unknown;
8   max_acc = 0;

9   FOR (pop_size = PS_MIN; pop_size ≤ PS_MAX; pop_size += PS_STEP)
    DO
10    FOR (crossover = CO_MIN; crossover ≤ CO_MAX; crossover += CO_STEP)
      DO
11      FOR (mutation = MU_MIN; mutation ≤ MU_MAX; mutation += MU_STEP)
        DO
12        acc_run = 0;
13        FOR (run = 1; run ≤ 5; run += 1)
          DO
14          CL_tmp = REFINEMENT(CL_init, Constraints, Ex_train,
                                pop_size, crossover, mutation,
                                TARGET_ACC, MAX_ITERATIONS);
15          acc = ACCURACY(CL_tmp, Ex_val);
16          acc_run += (1/5) * acc;
          END FOR
17        IF (acc_run > max_acc)
          THEN
18          best_population_size = pop_size;
19          best_crossover = crossover;
20          best_mutation = mutation;
21          max_acc = acc_run;
          END IF
        END FOR
      END FOR
    END FOR
22  return {best_population_size, best_crossover, best_mutation};
```

---

tuning, 7 of them were assigned to Ex_{train} and 2 to Ex_{val}. If validation examples are not given as input, 25 % of the training examples is set aside for validation.
Steps 5–8: Parameter-value initialization. The aim of this parameter tuning procedure is to find which parameter-value combination *pop_size*, *crossover* and *mutation* contributes to the highest classifier accuracy. The maximum accuracy variable *max_acc* keeps track of the highest obtained accuracy on the validation examples. It is initialized to zero. The population size *pop_size*, the crossover rate *crossover* and the mutation rate *mutation* are unknown.
Steps 9–21: Refinement is performed five times for each parameter-value combination *pop_size*, *crossover* and *mutation* using the training examples Ex_{train}. Five classifiers CL_{tmp} are created. The average classifier accuracy *acc_run* is computed using the validation examples. The maximum accuracy and the parameter values for which it was obtained are kept.
Step 22: The function outputs the population size, crossover rate and mutation rate.

The genetic algorithm outputs the final general-purpose classifier.

5.4 Online Adaptation

The aim of the online adaptation phase is to find the most suitable pattern-parameter values for a particular system deployment. People, for example, may have specific needs and preferences. In the fall-detection domain, one person 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. System adaptation to such user characteristics is needed for maximum performance.

Deployment-specific information is obtained through user feedback which is given occasionally, and contains information about false negatives (i.e., the system did not detect the class of interest when there was one) and false positives (i.e., the system detected the class of interest when there was none). User feedback reflects an underlying reward function, in our case a parameter-value desirability indicator. The mapping from user feedback to parameter-value rewards is specified by DK. Learning from rewards is mainly used in sequential decision-making domains, where the reward function is often considered as the most parsimonious description of a task (Ng and Russell, 2000). The online adaptation phase resembles sequential decision-making tasks as parameter-value adaptation is performed in a step-by-step manner until satisfactory, deployment-specific parameter values are reached.

The learning task is formulated using Markov decision processes (Russell and Norvig, 2010). A Markov decision process (MDP) is a 4-tuple (S, A, P, R), where S represents a finite set of states, A represents a finite set of actions, $P = P(s, a, s')$ is a transition probability matrix specifying the probability that an action a in state s would lead to state s' and $R = R(s)$ is a reward matrix representing state desirability. The MDP's solution specifies the most reward-bringing action for each state $s \in S$.

MDP application to parameter-value adaptation is presented in Algorithm 5.6 and Algorithm 5.7. The adaptation process is illustrated using a rule from the fall-detection domain: "IF a person is lying on the ground for P_{lying} % in T_{lying} THEN fall".

The first online-adaptation step is the initialization of the patterns' MDPs. A pattern's MDP initialization, which is performed for each classifier's pattern, is outlined in Algorithm 5.6:
Step 1: A Markov decision process $MDP_{pattern}$ is created.
Step 2: The number of pattern parameters *num_parameters* is retrieved. This variable is needed for the initialization of the MDP's state space S and action space A.

Algorithm 5.6: CDKML phase 3 – initialization of a pattern's Markov decision process.

MDP_INITIALIZATION(a pattern $pattern \in CL_{ref}$)

create a Markov decision process $MDP_{pattern}$;
$num\_parameters = pattern \rightarrow$ GetNumOfParameters();
FOR EACH *parameter* IN $pattern \rightarrow$ GetContinuousParameters()
  Discretize *parameter*;
END FOR EACH
$S = \prod_{p=1}^{num\_parameters} (pattern \rightarrow \text{GetValues(p)})$;
$MDP_{pattern} \rightarrow$SetStateSpace($S$);
$actions\_per\_parameter = \{INCREASE, NO\_CHANGE, DECREASE\}$;
$A = \prod_{p=1}^{num\_parameters} actions\_per\_parameter$;
$MDP_{pattern} \rightarrow$SetActionSpace($A$);
$P = S \times A \times S$;
FOR EACH s IN S
  FOR EACH a IN A
    correct_next_state = empty vector;
    FOR($p = 0$; $p <$ *num_parameters*; p++)
      IF ($a \rightarrow$ Get(p) == *INCREASE*)
      THEN
        correct_next_state $\rightarrow$ Append($s \rightarrow$ Get(p) + $S \rightarrow$ GetStep(p));
      ELSE IF ($a \rightarrow$ Get(p) == *NO_CHANGE*)
      THEN
        correct_next_state $\rightarrow$ Append($s \rightarrow$ Get(p));
      ELSE //$a \rightarrow$ Get(p) == *DECREASE*
        correct_next_state $\rightarrow$ Append($s \rightarrow$ Get(p) – $S \rightarrow$ GetStep(p));
      END IF
    END FOR
    $P(s, a,$ *correct_next_state*$) = 1$;
    $P(s, a, s') = 0$, $s' \neq$ *correct_next_state*;
  END FOR EACH
END FOR EACH
$MDP_{pattern} \rightarrow$SetTransitionProbabilityMatrix(P);
$R(s) = 0, \forall\ s \in S$;
$MDP_{pattern} \rightarrow$SetRewardMatrix($R$);
$MDP_{rule} \rightarrow$SetCurrentState ($pattern \rightarrow$GetParameterValues());
return $MDP_{pattern}$;

Steps 3–6: Initialization of the $MDP_{pattern}$'s state space S. The state space S contains *num_parameters* dimensions, each of which represents a pattern parameter. The fall-detection rule's state space is two-dimensional, with one dimension representing the set of possible percentage values P_{lying} and the other representing the possible time interval values T_{lying}. Visualization of this state space is presented Figure 5.4. Each state $s \in S$ is a vector containing *num_parameters* elements representing one possible parameter-value assignment. Because MDPs are defined for finite state spaces S, the continuous parameters need to be discretized.
Steps 7–9: Initialization of the $MDP_{pattern}$'s action space A. We consider three possible actions per pattern parameter: increase by one unit, do not change or decrease by one unit. An action $a \in A$ is a vector containing *num_parameters* elements, each of which specifies an action per pattern parameter.
Steps 10–23: Initialization of the $MDP_{pattern}$'s transition probability matrix P. We consider deterministic parameter-value changes. Therefore, the values in the transition probability matrix $P = P(s, a, s')$ equal 1 if *s'* represents the pattern-parameter values that are obtained by applying the changes specified in a to the parameter values in s, otherwise they equal 0.
Steps 24–25: The $MDP_{pattern}$'s reward matrix R is initialized to zero for all states (Figure 5.4a). The elements of the reward matrix R reflect the obtained user feedback and may change.
Step 26: The $MDP_{pattern}$'s current state is set to the refined classifier's parameter values.
Step 27: The function outputs $MDP_{pattern}$.

Having the patterns' MDPs initialized, parameter-value adaptation may be performed. Here, we define two notions used in the parameter-value adaptation procedure: a state-example distance and MDP state dominance. The state-example distance $dist_S(s, Ex)$ represents the minimum number of steps needed for a rule having the parameter values of s to be brought to cover the example Ex[1]. An MDP state s dominates an MDP state *s'* if a pattern having the parameter values of *s'* covers a subset of the examples covered by a pattern having the parameter values of s.

Algorithm 5.7 outlines the parameter-value adaptation procedure. Input to the algorithm are a set of patterns' MDPs CL_{MDP}, an obtained user feedback *UF*, an example which triggered the user feedback *Ex*, a penalty amount for false positives *PaFp* and for false negatives *PaFn*. Parameter-value adaptation is performed as follows:
Steps 1–6: Adaptation procedure in case of a false positive. The MDPs of the patterns that caused a false positive are added to the set MDP_{fp}. For each $MDP_{pattern} \in MDP_{fp}$ three steps are performed: (1) the reward of $MDP_{pattern}$'s current state and all states that dominate it is reduced by the penalty amount for false positives *PaFp*, (2) a set of new state candidates C_{states} is created encompassing the neighboring states of $MDP_{pattern}$'s current state which have the highest reward, and (3) the $MDP_{pattern}$'s current state is set to the state $s \in C_{states}$ with maximum distance from the triggering example *Ex*.
Steps 7–10: Adaptation procedure in case of a false negative. The MDP $MDP_{fn} \in CL_{MDP}$ whose current state is at minimum distance from the triggering example *Ex* is selected. The reward of MDP_{fn}'s current state and all states that it dominates is reduced by the penalty amount for false negative *PaFn*. A set of new state candidates C_{states} is created encompassing the neighboring states of MDP_{fn}'s current state which have the highest reward. The MDP_{fn}'s current state is set to the state $s \in C_{states}$ with minimum distance from the triggering example *Ex*.

[1] A pattern covers an example *Ex* if its condition part is true for the example *Ex*(Clark and Niblett, 1989)

Algorithm 5.7: CDKML phase 3 – classifier adaptation upon user feedback.

ADAPTATION(classifier's patterns' MDPs CL_{MDP}, user feedback *UF*, a triggering example *Ex*, penalty amount for false positives *PaFp*, penalty amount for false negatives *PaFn*)

1 IF ($UF == FALSE\_POSITIVE$)
THEN
2 MDP_{fp} = a subset of CL_{MDP} containing the MDPs of the patterns that caused a false positive;
3 FOR EACH $MDP_{pattern} \in MDP_{fp}$
DO
4 in $MDP_{pattern}$ reduce the reward of the current state and all states that dominate it by *PaFp*;
5 C_{states} = set of the neighboring states of $MDP_{pattern} \rightarrow$ currentState() with the highest reward;
6 $MDP_{pattern} \rightarrow \text{SetCurrentState}(\underset{s \in C_{states}}{\text{argmax}}\, dist_S(s, Ex))$;
END FOR EACH
ELSE // $UF ==$ false negative
7 $MDP_{fn} = \underset{MDP_{pattern} \in CL_{MDP}}{\text{argmin}}\, dist_S(MDP_{pattern} \rightarrow \text{currentState}(), Ex)$;
8 in MDP_{fn} reduce the reward of the current state and all states that it dominates by *PaFn*;
9 C_{states} = set of neighboring states of $MDP_{fn} \rightarrow$ currentState() with the highest reward;
10 $MDP_{fn} \rightarrow \text{SetCurrentState}(\underset{s \in C_{states}}{\text{argmin}}\, dist_S(s, Ex))$;
END IF
11 *adapted_parameters* = empty vector;
12 FOR EACH $MDP_{pattern}$ in CL_{MDP}
DO
13 *adapted_parameters* $\rightarrow$ Add($MDP_{pattern} \rightarrow$ currentState());
END FOR EACH
14 **return** *adapted_parameters*;

Steps 11–14: The adapted parameter values are returned. These are the parameter values represented by the CL_{MDP}'s current states.

Figure 5.4 illustrates the parameter-value adaptation process on the example fall-detection rule. The initial $MDP_{pattern}$ is presented in Figure 5.4a. $MDP_{pattern}$'s reward matrix is initialized to zero for all states and its current state is set to the refined classifier's rule values (P_{lying} %, T_{lying}) = (70 %, 9 s). The $MDP_{pattern}$'s current state is highlighted with a black rectangle. We assume that, after a certain period of time, a false positive feedback is obtained due to a classification error of the example rule. In this concrete rule, a false positive feedback reduces the reward of the current state and of all states that dominate it (states with less or equally 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 rule's parameter values must be made stricter (Figure 5.4b). After updating the state rewards, the set of neighboring states of $MDP_{pattern} \rightarrow$ currentState() with the highest reward is determined and the $MDP_{pattern}$'s current state is set to the state with the

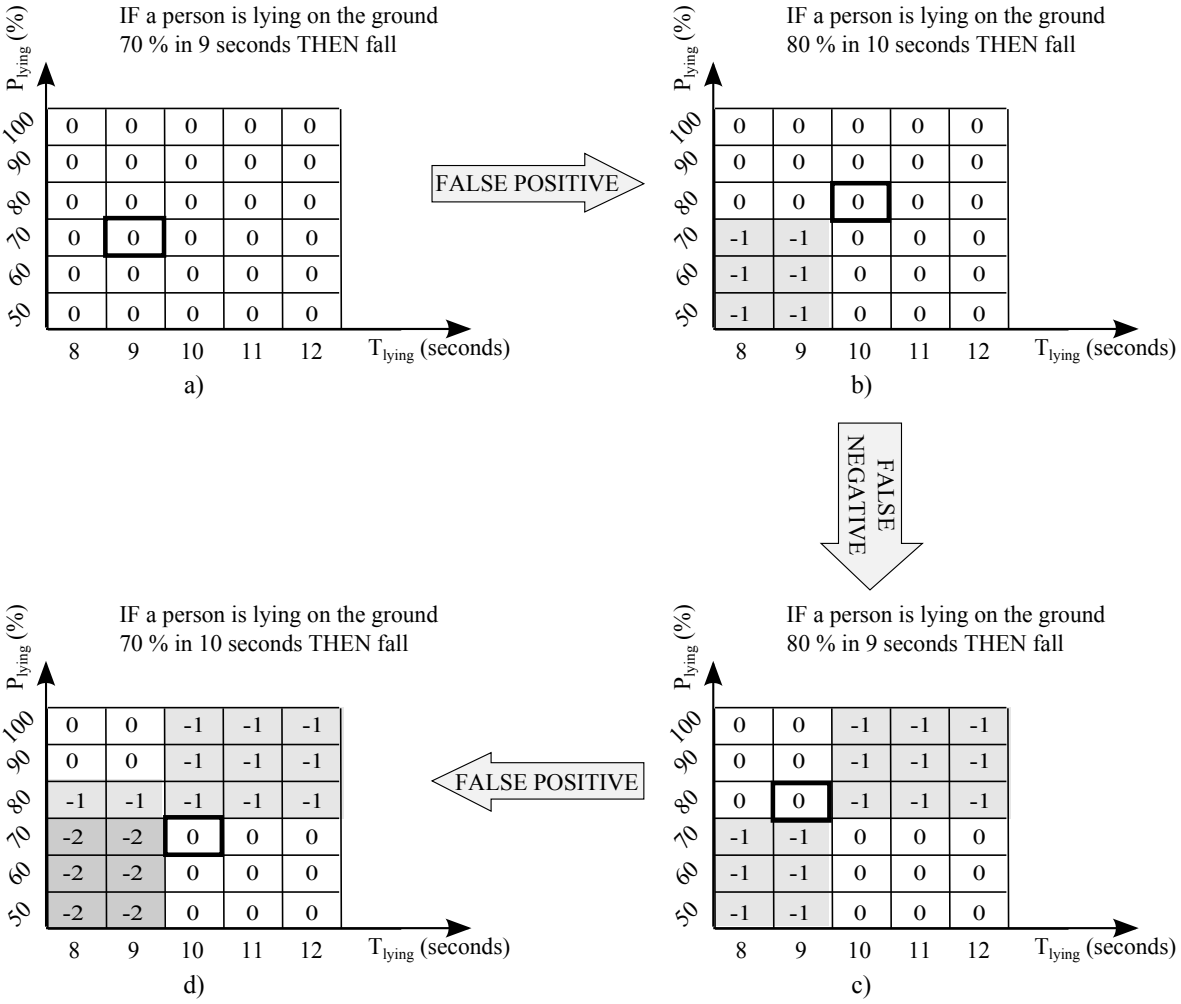

Figure 5.4: Visualization of CDKML's online adaptation process: a pattern's Markov decision process after a) initialization, b) a false positive user feedback, c) a false positive and a false negative user feedback, d) two false positive and one false negative user feedback.

maximum distance from the example that triggered the false positive. In the example rule, the distance from a state $s \in S$ to an example Ex that triggered user feedback is calculated as follows:

$$dist_S(s, Ex) = \min_{t \in T} dist_S(s, Ex, t)$$

where $T = S \rightarrow \text{GetValues}(dim_{time})$ is the set of values for T_{lying}. We calculate $dist_S(s, Ex, t)$ as follows:

$$dist_S(s, Ex, t) = \\ = \max\left(\left\lceil \frac{|s \rightarrow \text{GetValue}(dim_{time}) - t|}{S \rightarrow \text{GetStep}(dim_{time})} \right\rceil, \left\lceil \frac{|s \rightarrow \text{GetValue}(dim_{perc}) - perc|}{S \rightarrow \text{GetStep}(dim_{perc})} \right\rceil\right)$$

where $perc = Ex \rightarrow$GetLyingPerc(t). The new $MDP_{pattern}$'s current state (Figure 5.4b) 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 and that the example-rule's current state is at minimum distance from the triggering example. A false negative feedback reduces the reward of the current state and of all states that it dominates (states

with stricter or equal 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 rule's parameter values are too strict and need to be relaxed (Figure 5.4c). Again, the set of neighboring states of $MDP_{pattern}$'s current state with the highest reward is determined and the $MDP_{pattern}$'s current state is set to the state with the minimum parameter-value distance from the example that caused the false negative. Figure 5.4c presents a case where the feedback result reduced the strictness of the time parameter, while the percentage parameter remained unchanged. The initial state was avoided because of the negative reward received during the first false positive. A possible outcome after an additional false positive feedback is presented in Figure 5.4d. Pattern-parameters values are adapted in this way after each obtained user feedback.

6 Evaluation

This chapter focuses on evaluating the performance of the CDKML method. We would like to note that statistical comparison between CDKML and other methods is extremely difficult. CDKML is not run automatically. It involves a domain expert in the classifier generation process requiring a considerable amount of effort and time from him/her. The evaluation, therefore, considers experiments directed towards answering the following questions:

- How important is expert input in the CDKML method? Can we circumvent expert input?
- Is an expert capable of selecting a comprehensive set of concept patterns in the CDKML method?
- Does CDKML's approach to combining DK and ML contribute to improved classifier's performance?

The experiments were conducted using a custom implementation of CDKML in Java which uses two open-source software packages: (1) Weka (Hall et al., 2009) for ML classifier generation in the initialization phase, and (2) the Java Genetic Algorithms and Genetic Programming Package (Meffert et al., 2011) for pattern-parameter tuning in the refinement phase using genetic algorithms.

6.1 Behavioral Cloning[1]

CDKML points out the importance of expert input for classifier generation in the case of scarce data. The expert provides the set of patterns in CDKML's initialization phase and poses constraints on the search space in CDKML's refinement phase. However, can we circumvent expert input? This section presents a case study in which the set of patterns is extracted automatically according to their frequency of appearance in ML classifiers and without search-space constraints.

The case study was performed in the behavioral-cloning domain. We start by describing the serious game used for capturing behavior-examples as well as for evaluating behavior-clones' quality (Subsection 6.1.1). Then, we present the created datasets (Subsection 6.1.2). Finally, we discuss the results (Subsection 6.1.3).

6.1.1 The Serious Game

This study is based on a serious game which simulates the interaction between participants of two asymmetric, opposing groups: civilians and soldiers. The interaction takes place at a camp entrance where the civilians are gathering in order to apply for a job. The soldiers are guarding the camp. A screenshot of the serious-game environment is shown in Figure 6.1.

[1] This section supplements the publication Mirchevska et al. (2012).

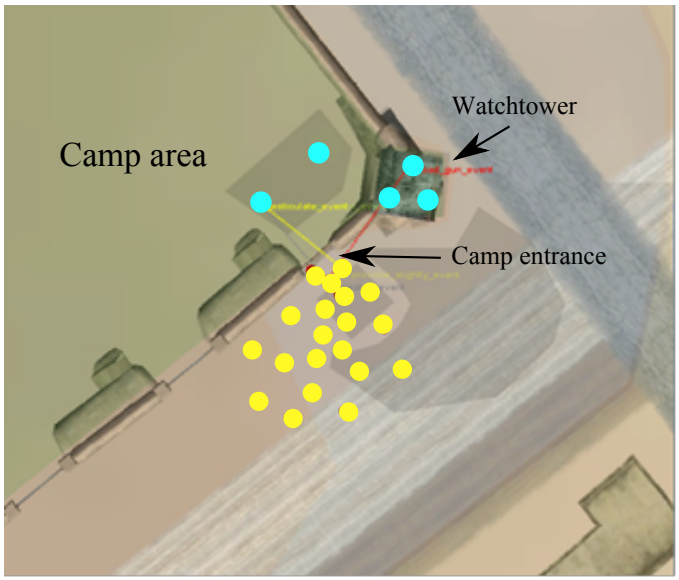

Figure 6.1: The serious-game environment.

The blue dots represent the soldiers, while the yellow dots outside the camp are the civilians. Civilians' behavior is predefined and specified in PECS (Physis, Emotion, Cognition, Social Status) reference models (Schmidt, 2000), while soldiers' behavior is controlled by a person.

6.1.2 Data

Behavior examples were obtained through the serious game by letting a person control the soldier agents, while the civilians behaved as specified in the PECS reference models. An interface was created for this purpose. It reported the serious-game state each time an action was expected from a soldier agent after which the person selected an action to be executed. A behavior example was created for each executed soldier action. Behavior data was recorded in ten serious-game runs which on average lasted 2772 steps. Table 6.1 presents the number of examples per soldier action in each serious-game run.

The serious-game course was also logged using twelve indicators (measures of effectiveness – MoEs). Examples of recorded MoEs are the number of injured, the level of civilians' anger and civilian leader's readiness for aggression. The MoEs were recorded in each game step.

6.1.3 Evaluation of CDKML in the Absence of Domain Knowledge

This evaluation presents an attempt to circumvent expert input in CDKML's initialization and refinement phases (Figure 6.2).

The Classifier

This subsection presents the created rule-based behavior clone. As presented in Section 4.1, the goal was to extract a single-level policy function consisting of a set of rules. The set of rules was selected using the CDKML's initialization phase as follows. Ten behavior subsets were created, each of which contained the examples of nine out of the ten serious-game logs. On each behavior subset, a set of decision trees were induced as proposed in CDKML's initialization. If a rule pattern was present in the decision trees of at least five out of the ten behavior subsets, it was included in the initial classifier. The rule's parameter values

Table 6.1: Number of action examples.

	run 1	run 2	run 3	run 4	run 5	run 6	run 7	run 8	run 9	run 10	avg.
communicate calming event	3	0	1	0	2	25	3	13	44	27	11.8
communicate warning event	1	3	1	4	6	3	6	20	4	11	5.9
gesticulate	0	0	0	3	2	0	6	3	0	0	1.4
show weapon	2	2	2	2	2	1	0	2	1	0	1.4
load gun	2	2	2	2	2	1	4	15	1	0	3.1
perform warning shot	5	2	3	6	24	0	64	23	0	24	15.1
perform effective shot	3	3	3	3	3	0	0	0	0	2	1.7

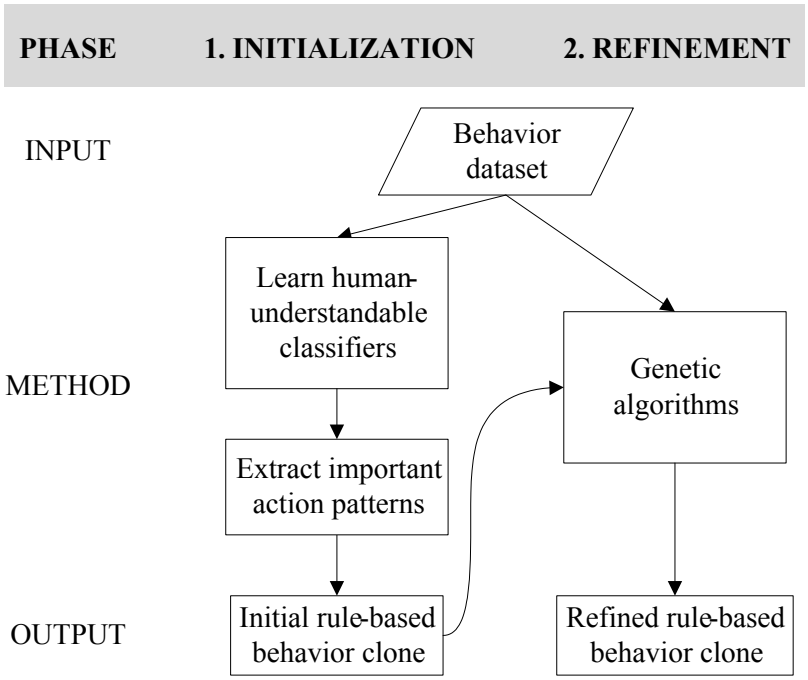

Figure 6.2: CDKML's application to the behavioral-cloning domain – creating a rule-based behavior clone using CDKML's initialization and refinement phases.

were tuned using the available training data in the CDKML's refinement phase without constraints on the search space.

The behavior clone encompasses the soldiers' internal states which cause the execution of seven actions: communication of a calming event, communication of a warning event, gesticulation, show of weapon, gun loading, performing a warning shot and performing an effective shot. A soldier's internal state captures four attribute sets: soldier's location (e.g., at the entrance, on the watchtower), soldier's interaction with the civilian leaders (e.g., is a civilian leader near the soldier, the civilian leaders' anger value, their leading motive,

performed provocations by the civilian leaders), the average civilians' anger and previously performed soldier actions (e.g., previous gesticulation, previous communication of a calming event and similar).

The behavior clone contained fifteen rules. Below we give examples of rules present in the rule engine:

1. IF $\text{communicated\_calm\_event}_{\text{soldier}}$ AND NOT($\text{performed\_warning\_shot}_{\text{soldier}}$) AND $\text{avg\_anger}_{\text{civilians}} > 58$ THEN communicate_calm_event (confidence: 0.80);

2. IF $\text{communicated\_calm\_event}_{\text{soldier}}$ AND NOT($\text{performed\_warning\_shot}_{\text{soldier}}$) AND $\text{avg\_anger}_{\text{civilians}} > 5$ AND $\text{anger}_{\text{civilian\_leader}} > 18$ AND $\text{anger}_{\text{civilian\_leader}} < 69$ THEN communicate_warning_event (confidence:0.53);

3. IF $\text{communicated\_calm\_event}_{\text{soldier}}$ AND $\text{gesticulated}_{\text{soldier}}$ AND $\text{avg\_anger}_{\text{civilians}} < 90$ THEN gesticulate_event (confidence: 0.24);

4. IF $\text{communicated\_calm\_event}_{\text{soldier}}$ AND $\text{avg\_anger}_{\text{civilians}} > 10$ AND $\text{avg\_anger}_{\text{civilians}} < 60$ AND $\text{anger}_{\text{civilian\_leader}} > 47$ THEN perform_warning_shot (confidence: 0.82).

The number of parameters in the behavior clone (i.e., the chromosome length in CDKML's refinement phase) equaled 46.

Results

This section compares the performance of CDKML's behavior clone to the performance of ML behavior clones induced in Weka (Hall et al., 2009) with SMO, RandomForest, NaiveBayes, JRip and J48. We used the default Weka's algorithm parameter-values, and the same instance attributes (soldiers' internal-state features) as applied in creating CDKML's behavior clone.

The behavior clones were evaluated based on two measures: accuracy and game-course difference. The accuracy (ACC) equals the proportion of correctly predicted examples from all examples:

$$ACC = \frac{number\ of\ correctly\ predicted\ examples}{number\ of\ examples} \tag{6.1}$$

The game-course difference (DIFF) shows to what degree the behavior clones are capable of reproducing the same serious-game outcome as the modeled entity. It therefore gives a global judgment of the behavior clones' performance in the analyzed environment. The game-course difference was computed using dynamic time warping (Müller, 2007). Dynamic time warping is used for estimating the difference between two time series. It determines the ideal warp, i.e., optimal alignment, between two time series by comparing the distance between each possible pair of points of the two time series. In order to compare the MoE values of two serious-game runs, a distance metric between two multi-dimensional time points is needed. We calculate this distance as follows:

$$dist(moe_1, moe_2) = \sum_{d=1}^{D} (moe_1(d) - moe_2(d)) \tag{6.2}$$

where moe_1 and moe_2 are points on two D-dimensional MoE time series, and $moe_i(d)$ represents the value of the d-th dimension of the point moe_i. The sum of the distances of

Table 6.2: Behavior-clone comparison with respect to accuracy.

	CDKML	SMO	Random-Forest	Naive-Bayes	JRip	J48
Predictive accuracy	0.51	0.57	0.51	0.50	0.53	0.47

the pairs of points on the ideal warp represents the difference between the two time series, i.e., the game-course difference:

$$DIFF = \sum_{d \in ideal\_wrap} dist_{moes}(d) \tag{6.3}$$

It is a value in the interval [0, 100] where higher values indicate higher time-series difference.

Separate-training-and-test-set evaluation scenario was used for estimating behavior-clones' performance. Training was performed using data in nine out of the ten serious-game logs (CDKML's refinement used seven out of the nine logs for training and two for validation). The tenth serious-game log (not seen during training) was used for estimating the behavior clones' accuracy. For the purpose of measuring the difference between the serious-game course produced by the person and a behavior clone, we incorporated the behavior clone in the serious game and recorded its course ten times for 1500 steps. The game-course difference between the person and a behavior clone was calculated as the average of the ten person-clone MoE differences, each of which compared one of the ten clone's serious-game courses with the serious-game course in the tenth person's serious-game log (not seen during training). This test scenario was performed ten times for each learning approach, each time leaving out from training a different serious-game log. The performance of each learning approach is represented as the average behavior-clone accuracy and game-course difference in the ten test runs.

Table 6.2 compares the performance of CDKML and the ML algorithms with respect to the behavior clones' accuracy. CDKML's accuracy is slightly higher than J48's and NaiveBayes's accuracy, the same as RandomForest's accuracy, and lower than JRip's and SMO's accuracy. Despite the difference in accuracy, superiority of one rule-based behavior clone (consisting of rules or behavior clones which can be converted to a ruleset) over the others was not evident. Although having higher average accuracy, CDKML was better than J48 on 4 out of the 10 test datasets. It had higher accuracy than RandomForest on 3 test datasets and the same accuracy on one dataset. Finally, although having lower average accuracy, CDKML outperformed JRip on 6 test datasets. The SMO behavior clone, however, showed better performance than CDKML with respect to accuracy. It had higher accuracy in 8 of the 10 test datasets, while the accuracy was the same on one dataset.

Table 6.3 compares the performance of CDKML and the ML algorithms with respect to game-course difference. Each cell in the table represents the average difference between the MoE values of the game type in the row and the column of the cell. The person-CDKML MoE difference is on average 7, the same as the person-SMO MoE difference, while the person-J48, the person-JRip and the person-RandomForest average MoE difference is slightly lower. Despite having different person-clone MoE differences, superiority of one rule-based behavior clone over the others was not evident. The person-CDKML MoE difference is in the interval [4, 14], the person-J48 difference is in the interval [1, 13], the person-JRip difference is in the interval [1, 14], the person-RandomForest difference in the interval [1, 17].

Table 6.3: Behavior-clone comparison with respect to game-course difference.

	A person	CDKML	SMO	Random-Forest	Naive-Bayes	JRip	J48
A person	3	7	7	4	2	5	5
CDKML		3	2	7	7	7	5
SMO			2	6	7	6	5
RandomForest				6	4	6	6
NaiveBayes					1	5	5
JRip						6	6
J48							5

The NaiveBayes behavior clone, however, showed better performance than CDKML with respect to the game-course difference. The person-NaiveBayes MoE difference was in the interval [1, 4].

The presented approach to circumventing expert input did not show improvement in comparison to standard ML. Although the analysis in this case study is limited, in the absence of DK it would be difficult to outperform standard ML by separating the classifier generation process into two subtasks, extraction of characteristic patterns and pattern-parameter value optimization, as proposed by CDKML's initialization and refinement phase. The analyzes that follow, however, show that if expert DK is available, this approach achieves higher accuracy than standard ML when the training dataset captures a limited amount of concept examples.

6.2 Posture Recognition

The analysis in the posture-recognition domain concerns two questions: (1) is an expert capable of extracting a comprehensive set of concept patterns in the CDKML method, and (2) does the combination of expert DK and ML as proposed by CDKML contribute to improved classifier's performance.

We start this section by describing the reasoning flow in the Confidence system using which we generated posture examples (Subsection 6.2.1). Then, we present the created datasets (Subsection 6.2.2). Finally, we discuss the evaluation results. Subsection 6.2.3 evaluates a posture-recognition classifier constructed by a domain expert, while Subsection 6.2.4 evaluates a classifier generated using CDKML's initialization and refinement phases.

6.2.1 The Confidence System

Confidence is a ubiquitous system for real-time health problem detection. It's target group are the elderly to whom the system should give the necessary confidence to continue living in their home, obtaining medical care only when needed. Figure 6.3 presents a simplified version of the reasoning flow in the Confidence system. Detailed system descriptions can be found in literature (Kaluža et al., 2010; Mirchevska et al., 2010; Luštrek et al., 2011; Kaluža et al., 2013).

In the Confidence system, the user is equipped with wearable tags from a real-time location system (RTLS). The RTLS system measures the x, y and z coordinates of the user's body parts to which the tags are attached. The raw RTLS data is first preprocessed to estimate missing measurements and reduce noise. The preprocessed RTLS data is then submitted as input to the attribute computation module. This module computes characteristics of a person's body, including tag velocity and amount of movement, and relations

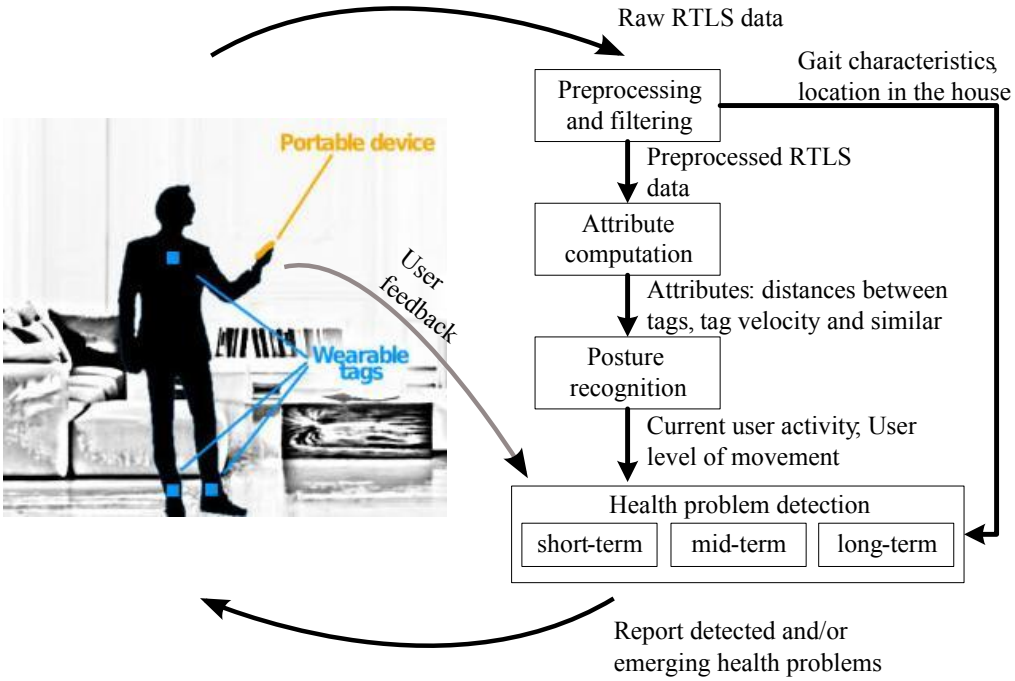

Figure 6.3: The Confidence system, a ubiquitous system for real-time health problem detection.

between body parts, including the distance between tags. The posture recognition module uses these characteristics to classify the person's posture 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 system contains three health problem detection modules, short-term, mid-term and long-term, each of which takes into consideration the output of the posture recognition module. The Confidence system communicates with the user through a portable device using which it reports detected and emerging health problems to the user. The user uses the portable device to provide feedback – report an erroneous detection of a health problem or call for help in the case of an emergency not detected by the system. In case of an emergency, the Confidence system contacts a caregiver for help.

6.2.2 Data

The experiments were performed using human posture examples recorded in two phases. The first phase, containing 135 sequences of behavior of three people, includes examples of standing/walking, lying down, sitting down and falling. The second phase, which contains 775 sequences of behavior of five people (three of which are the people present in the first-phase recordings), includes the basic behaviors recorded in the first phase, examples of several kinds of falls, and, based on discussions with physicians, examples of walking and lying of people with different health problems, such as Parkinson's disease and hemiplegia. Table 6.4 presents the number of posture examples per recording phase.

The recordings were made with the use of the Smart infrared motion capture system (eMotion, 2009), because at the time of this experiment the Confidence's RTLS hardware was under development. In the recordings, the locations of twelve tags were measured, one on each shoulder, hip, knee, ankle, elbow and wrist. The location of a virtual tag on the

Table 6.4: Number of posture examples.

Posture	First phase	Second phase
Standing	1544	39070
Sitting	733	5368
Lying	1773	5337
Falling	689	2229
Moving downwards	1696	5044
Moving upwards	0	421
On all fours	0	2183

chest was computed as the middle point between the shoulders due to difficulties in attaching a tag there and tracking it during forward falls. The coordinates of the tags were sampled at a frequency of 60 Hz. This data was processed in order to bring it in a form analogous to the anticipated Confidence's RTLS hardware – the Ubisense system (Ubisense, 2012). Two transformations were applied. First, the sampling frequency was reduced to 10 Hz. Then, Gaussian noise with standard deviation of 4.36 cm horizontally and 5.44 cm vertically was added to the data. The values of the standard deviation of the noise in the Ubisense system were obtained experimentally.

6.2.3 Evaluation of a Classifier Constructed by a Domain Expert Using Interactive Data Mining[1]

This subsection examines if an expert is capable of selecting a comprehensive set of concept patterns using DK and interactive data mining, thus creating a representative classifier. A rule-based posture-recognition classifier was generated using CDKML's initialization phase as depicted in Figure 6.4.

The Classifier

This subsection presents the posture-recognition classifier constructed by a domain expert. It is a rule-based classifier whose reasoning is based on the position of the person's chest and the ankles. More precisely, only the z coordinates of the chest and the ankles are considered. The x and y coordinates are not relevant, because they refer to the place in the room where the person is. Additionally, the chest-ankle distance in z direction and its projection on the xy plane are used. These distances are the most important for distinguishing between lying, sitting and standing. Finally, the velocity of the chest is considered. Being one of the topmost body parts, the velocity of the chest is the highest during falls, moving downwards and upwards, making it suitable for distinguishing them.

The classifier (Figure 6.5) contains three rule types: (1) strict posture rules, (2) weak posture rules, and (3) a default rule.

The strict posture rules contain precise definitions of the body configuration in each of the postures of interest. The expert examined a set of decision trees as proposed in CDKML's initialization phase. A set of rules was extracted from the decision trees, a part of which were modified by DK. Twelve strict posture rules were specified, a subset of which is provided below:

1. IF Distance_Z_AnkleLeftToChest > 1.1 m AND
   Distance_Z_AnkleRightToChest > 1.1 m AND

[1]This subsection is based on the publication Mirčevska et al. (2009).

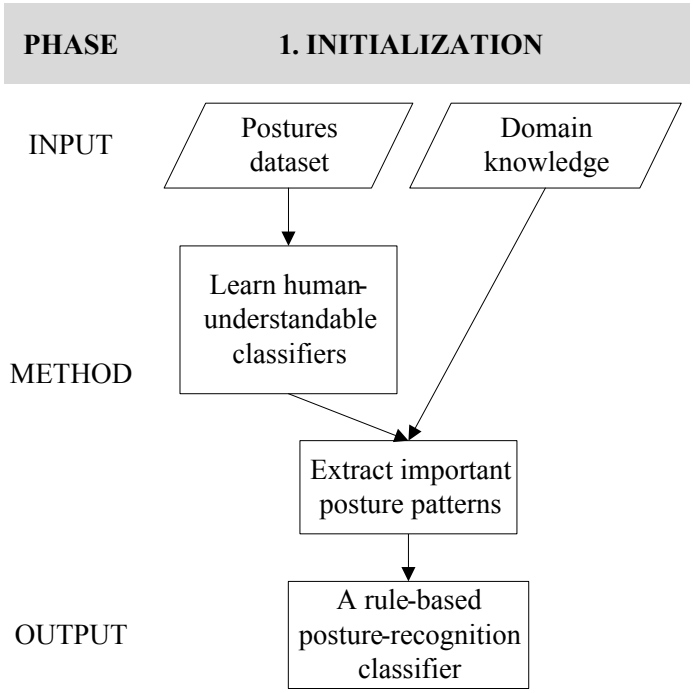

Figure 6.4: CDKML's application to the posture-recognition domain – creating a rule-based classifier using CDKML's initialization phase.

Distance_XY_AnkleLeftToChest $<$ 1 m AND
Distance_XY_AnkleRightToChest $<$ 1 m AND
Velocity_Z_Chest $>$ -0.7 m/s AND Velocity_Z_Chest $<$ 0.7 m/s THEN Standing;

2. IF Velocity_Z_Chest $<$ 0.2 m/s AND Velocity_Z_Chest $>$ -0.2 m/s AND
Distance_Z_AnkleLeftToChest $<$ 0.2 m AND
Distance_Z_AnkleRightToChest $<$ 0.2 m THEN Lying;

3. IF Velocity_Z_Chest $<$ -1.5 m/s THEN Falling;

4. IF Z_AnkleRight $<$ 0.13 m AND Z_AnkleLeft $<$ 0.13 m AND
Velocity_Z_Chest $<$ -0.5 m/s AND Velocity_Z_Chest $>$ -1.3 m/s AND
Distance_XY_AnkleLeftToChest $<$ 0.8 m AND
Distance_XY_AnkleRightToChest $<$ 0.8 m THEN Going down;

5. IF Z_Chest $>$ Z_AnkleRight AND Z_Chest $>$ Z_AnkleLeft AND
Distance_XY_AnkleLeftToChest $>$ 0.15 m AND
Distance_XY_AnkleRightToChest $>$ 0.15 m AND
Distance_XY_AnkleLeftToChest $<$ 0.7 m AND
Distance_XY_AnkleRightToChest $<$ 0.7 m AND
Distance_Z_AnkleLeftToChest $>$ 0.7 m AND
Distance_Z_AnkleRightToChest $>$ 0.7 m AND
Distance_Z_AnkleLeftToChest $<$ 1 m AND
Distance_Z_AnkleRightToChest $<$ 1 m AND
Velocity_Z_Chest $<$ 0.2 m/s AND Velocity_Z_Chest $>$ -0.2 m/s AND
Velocity_total_Chest $<$ 0.7 m/s THEN Sitting;

6. IF Velocity_Z_Chest $>$ 0.2 m/s AND Velocity_total_Chest $>$ 0.7 m/s AND
Distance_Z_AnkleLeftToChest $>$ 0.7 m AND

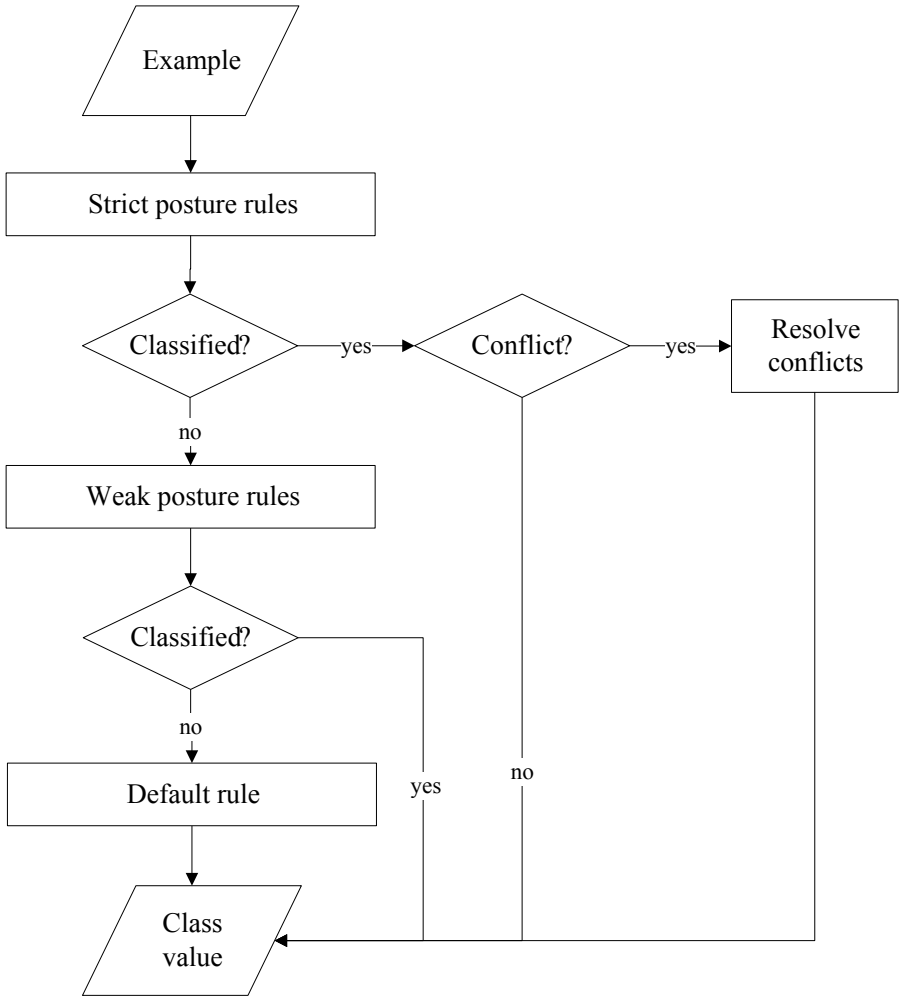

Figure 6.5: Architecture of the posture-recognition classifier constructed by a domain expert.

Distance_Z_AnkleRightToChest $>$ 0.7 m AND
Distance_Z_AnkleLeftToChest $<$ 1.1 m AND
Distance_Z_AnkleRightToChest $<$ 1.1 m THEN Standing up.

Each example is first processed by the strict posture rules. If it is covered by strict posture rules describing equal posture class, that class in assigned to it. Conflicts when a particular instance is covered by rules describing more than one posture class are resolved as presented in Table 6.5. Conflicts appear between rules for adjacent classes (e.g., standing and going down). Since the rules for standing, sitting, lying and falling were constructed in a way that only pure postures are captured, they are chosen when there is a conflict with a rule for moving downwards/upwards.

The weak posture rules specify the most probable class according to the person's chest-ankle distance and chest velocity. The weak posture rules were created using DK. Five weak posture rules were specified:

1. IF Distance_Z_AnkleLeftToChest $>$ 1.2 m AND
   Distance_Z_AnkleRightToChest $>$ 1.2 m THEN Standing;

2. IF Distance_Z_AnkleLeftToChest $<$ 0.2 m AND
   Distance_Z_AnkleRightToChest $<$ 0.2 m THEN Lying;

3. IF Velocity_Z_Chest $<$ -0.2 m/s THEN Going down;

Table 6.5: Resolution of conflicts among the rules in the posture-recognition classifier constructed by a domain expert.

Conflict	Result
Standing and moving downwards/upwards	Standing
Sitting and moving downwards/upwards	Sitting
Lying and moving downwards/upwards	Lying
Falling and moving downwards/upwards	Falling

4. IF Velocity_Z_Chest > 0.2 m/s THEN Standing up;

5. IF Distance_XY_AnkleLeftToChest < 0.7 m AND
   Distance_XY_AnkleRightToChest < 0.7 m THEN Sitting.

Each example which is not covered by any of the strict posture rules is processed by the weak posture rules, which are processed in the given order. The example obtains the class of the first weak rule which covers it, if such rule exists.

Finally, the default rule is used to assign a class to an example that is not covered by both the strict and the weak posture rules. Since the current posture of a person is highly correlated with the posture he/she had in the previous time interval, the default rule assigns the class of the previous time interval to the example in the current time interval.

Results

This subsection compares the performance of the expert's posture-recognition classifier to the performance of ML posture-recognition classifiers induced in Weka (Hall et al., 2009) with SMO, RandomForest, NaiveBayes, JRip and J48. The ML classifiers were created with the default Weka's parameter-values, except for J48, which was applied with the minimal number of instances per leaf set to 2 % of the training-dataset size. They used the same example attributes as the expert's posture-recognition classifier: the z coordinates of the chest and the ankles, the absolute chest-ankle distance, the chest-ankle distance in z direction and its projection on the xy plane, the absolute velocity, and the velocities in z direction of the chest and the ankles.

We used accuracy (Equation (6.1)) to measure the posture-recognition classifiers' quality. The ML classifiers were evaluated with 10-fold cross validation on the data from both phases together and with three separate-training-and-test-set scenarios. In the first and second separate-training-and-test-set scenario, the classifiers were induced from data in one recording phase and their quality was tested on the other recording phase. In the third scenario, the classifiers were induced from the first and second phase recordings of two people, and their quality was tested on the first and second phase recordings of the third person. The quality of the expert's classifier is presented using its accuracy on the test dataset in each separate-training-and-test-set scenario.

Examination of the ML classifiers' accuracy in the different evaluation scenarios suggests a certain degree of overfitting. The accuracy of these classifiers is the highest when evaluated with 10-fold cross validation (Table 6.6). The highest accuracy of 0.96 was achieved by the RandomForest classifier. The random selection of training and test dataset in 10-fold cross validation permits data about the behavior of a concrete person in a concrete phase to be present in both the training and test dataset, resulting in high classifiers' accuracy in this evaluation scenario. The classification accuracy decreases in the evaluation scenario in which the classifiers are induced from data about two people and tested on data of the third person

Table 6.6: Accuracy of ML posture-recognition classifiers estimated using 10-fold cross validation.

	SMO	Random-Forest	Naive-Bayes	JRip	J48
10-fold cross validation	0.90	0.96	0.81	0.93	0.88

Table 6.7: Posture-recognition classifier comparison with respect to accuracy estimated with separate-training-and-test-set evaluation.

Training dataset	Test dataset	SMO	Random-Forest	Naive-Bayes	JRip	J48	Test dataset	CDKML
First phase	Second phase	0.85	0.74	0.68	0.70	0.80	Second phase	0.91
Second phase	First phase	0.78	0.80	0.68	0.77	0.71	First phase	0.82
Two people	Third person	0.87	0.86	0.76	0.85	0.87	Third person	0.89

(Table 6.7). The highest accuracy of 0.87 was achieved by the SMO and the J48 classifiers. In this case, the training dataset does not contain data about the behavior of the person on which the classifier is tested. However, since all people were instructed to behave in the same way in both recording phases, and they were able to observe and copy each other, the classifiers induced in this evaluation scenario are likely overfitted to this particular behavior of the people. The most significant drop in accuracy happens in the evaluation scenarios in which the classifiers are induced from one recording phase and tested on the other (Table 6.7). The SMO classifier had the highest accuracy (0.85) when the first recording phase was used for training and the second for testing. The RandomForest classifier had the highest accuracy (0.80) when the second recording phase was used for training and the first for testing. In this case, the training and test datasets contain different behavior, and there are people for which recordings were only made in the second phase. The fall in classification accuracy in this scenario supports the observation that the ML classifiers get overfitted to the people and the behavior present in the training dataset.

The performance achieved by the expert's classifier suggests that a comprehensive set of rules of the learned concept may be created using DK and interactive examination of decision-tree classifiers. In the scenarios in which one recording phase is used for training and the other for testing, the accuracy of the J48 classifier is more than 0.10 lower than the accuracy of the expert's classifier. We would like to note that part of the rules in the expert's classifier were obtained by interactive examination of decision trees induced using J48. The accuracy of the other three ML classifiers is also smaller in these evaluation scenarios. The difference is the highest for the RandomForest, JRip and NaiveBayes classifiers whose accuracy is more than 0.15 lower than the expert's classifier when the training was done on the first recording phase and testing on the second. There was no significant difference in accuracy in the scenario in which the ML classifiers were trained on the recordings of two people and tested on the third person. Nevertheless, higher classification accuracy of

the expert's classifier in this case still suggests that incorporation of DK improves classifier generality.

We would like to note that the presented evaluation does not prove superiority of the expert's classifier over the classifiers induced solely by ML. We rather want to show that experts may extract a comprehensive set of patterns using DK and interactive data mining. We see the expert's classifier and the ML classifiers as two, distinct view points of the learned concept whose combination brings the highest benefit.

6.2.4 Comparison of CDKML's Performance to the Performance of Machine Learning

This subsection aims at examining if the incorporation of DK in classification learning as proposed by CDKML's initialization and refinement phase contributes to improved classifier performance. A rule-based posture-recognition classifier was generated using the CDKML's initialization and refinement as depicted in Figure 6.6.

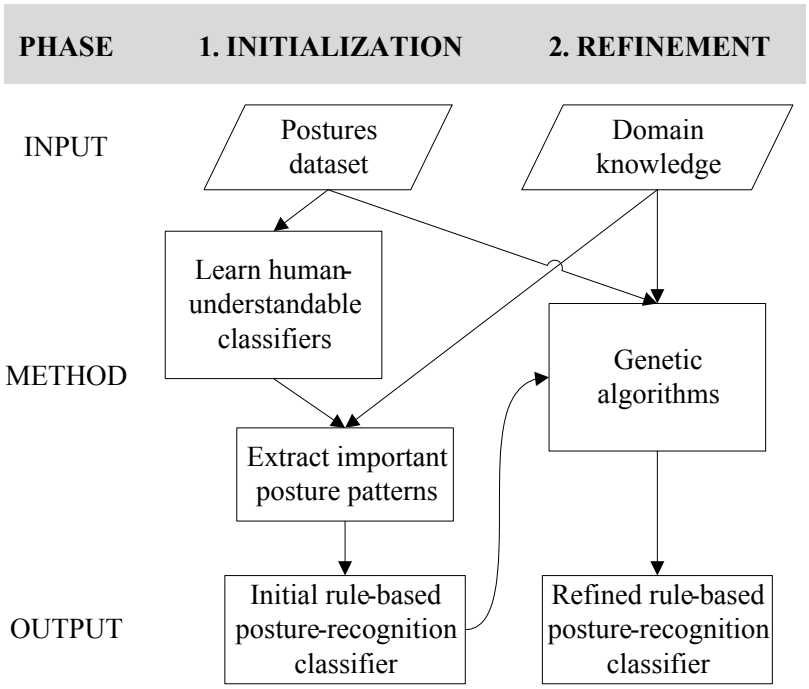

Figure 6.6: CDKML's application to the posture-recognition domain – creating a rule-based classifier using CDKML's initialization and refinement phases.

The Classifier

This evaluation is performed using the proposed CDKML's classifier form (Figure 5.2). The classifier encompassed 17 rules – the union of the strict and weak posture rules of the expert's classifier presented in Subsection 6.2.3. The number of classifier's parameters (i.e., the chromosome length in CDKML's refinement phase) equaled 47. Rule conflicts were resolved using the maximum confidence strategy. If an example was not covered by any of the rules in the classifier, its class value was designated as unknown.

The rule parameters in the initial posture-recognition classifier had the values which were specified by the expert. CDKML's refinement was then applied for the purpose of determining the most suitable general-purpose rule-parameter values based on training posture examples.

Results

This subsection compares the performance of CDKML's posture-recognition classifier to the performance of ML posture-recognition classifiers induced in Weka (Hall et al., 2009) with SMO, RandomForest, NaiveBayes, JRip and J48. The ML classifiers were created with the default Weka's parameter-values, except for J48, which was applied with the minimal number of instances per leaf set to 2 % of the training-dataset size. They used the same example attributes as CDKML's posture-recognition classifier: the z coordinates of the chest and the ankles, the absolute chest-ankle distance, the chest-ankle distance in z direction and its projection on the xy plane, the absolute velocity, and the velocities in z direction of the chest and the ankles.

This evaluation uses the separate-training-and-test-set scenario, where posture examples from the first phase were used for classifier training while classifier evaluation was done on the posture examples from the second phase. A set of training dataset sizes ranging from 50 to 6435 with a step 1000 was used. For each training dataset size, five training subsets were randomly drawn from the first-phase posture examples. A classifier was induced from each of the five training subsets and its average error (Equation (3.5)) was computed on the second-phase posture examples.

Figure 6.7 plots the error rate per training-dataset size of the CDKML's refined and the ML classifiers. We would like to note that the J48 error in Figure 6.7 is the same as the separate-training-and-test-set error presented in Figure 3.1.

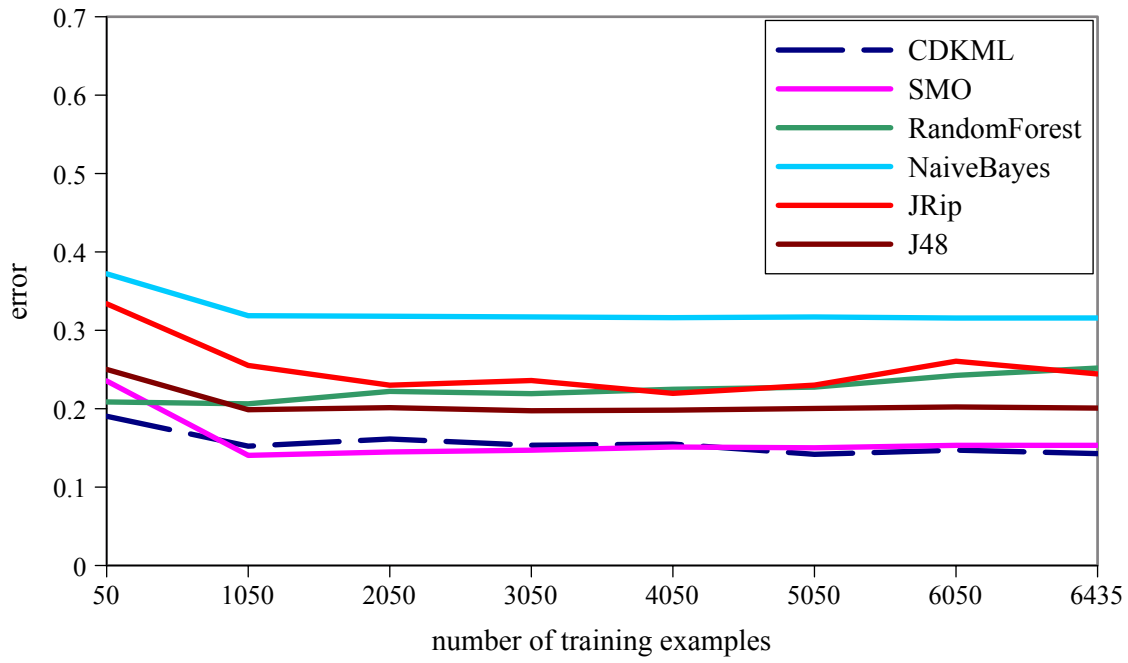

Figure 6.7: Classifiers' performance estimation. Comparison of the separate-training-and-test-set error of the CDKML's and ML's posture-recognition classifiers.

CDKML's separate-training-and-test-set error was approximately 0.19 for training dataset size of 50 examples. Its error was relatively constant at 0.15 for training dataset sizes above 1050 examples. CDKML's error was lower than the error of the rule-based ML classifiers (classifiers consisting of rules or classifiers which can be converted to a ruleset): J48, JRip and RandomForest. CDKML's error was approximately 0.05 lower than the J48's error for all training-dataset sizes. We would like to note that part of the rules in

the CDKML's classifier were obtained by interactive examination of decision trees induced using J48. CDKML's error was more than 0.07 lower than the JRip's error for all training dataset sizes and more than 0.05 lower than the RandomForest's error for training-dataset sizes having more than 1050 examples. CDKML also outperformed the NaiveBayes classifier whose error was the highest among all classifiers. CDKML's performance was comparable to the performance of SMO. SMO's error was approximately 0.24 for training dataset size of 50 examples, while being relatively constant at 0.15 for training dataset sizes above 1050 examples. Although not outperforming the SMO classifier in terms of the error rate, CDKML's transparency is an advantage compared to SMO (a black-box classifier).

Similarly to the J48's separate-training-and-test-set error discussed in Section 3.2, the classifiers' error in Figure 6.7 is relatively constant for training dataset sizes above 1050 examples. This confirms the observation that the test set contains patterns not present in the dataset used for training (also noted in the description of the two recording phases). One exception is the RandomForest's error which increases with the increase of the training-dataset size. This observation indicates that the RandomForest classifier overfits to the training examples.

6.3 Fall Detection[1]

Fall detection was addressed as a subtask in the broader system for health problem detection Confidence (Subsection 6.2.1). The fall-detection module in Confidence obtains data concerning a person's posture history and movement levels as input from the posture recognition module. It detects falls using the four rule types shown in Section 5.2, 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 because posture recognition is not perfect. The posture 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, the posture on all fours which 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 RTLS system measurements and may cause false positives. However, the amount of sitting on the ground recognized when a person is sitting on a low chair should be lower than the amount of this activity recognized when a 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.

The analysis in the fall-detection domain concerns the question: does the combination of expert DK and ML as proposed by CDKML contribute to improved classifier's performance. It is separated in two subsections, the first of which (Subsection 6.3.1) evaluates classifier performance after each of the three CDKML phases, while the second (Subsection 6.3.2) presents a more detailed evaluation of CDKML's approach to online classifier adaptation.

[1]This section is based on the publications Mirchevska et al. (2010) and Mirchevska et al. (2013b).

6.3.1 Comparison of CDKML's Performance to the Performance of Machine Learning

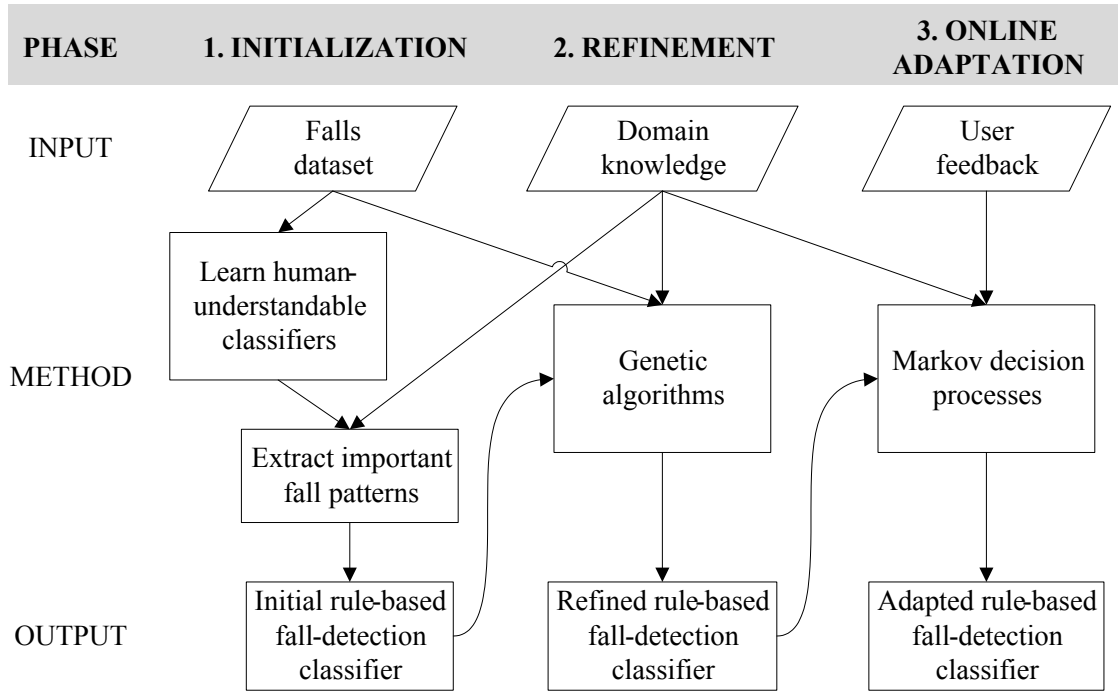

Figure 6.8: CDKML's application to the fall-detection domain – creating a rule-based classifier using CDKML's initialization, refinement and online adaptation phases.

This subsection aims at examining if the combination of expert DK and ML as proposed by CDKML contributes to improved classifier's performance. A rule-based fall-detection classifier was generated using CDKML as presented in Figure 6.8. This evaluation was performed using the fall-detection classifier presented in Section 5.2.

Data

We designed a test scenario to investigate the generality and robustness of the developed rule-based classifiers using the CDKML's initialization and refinement phases, as well as their adaptation capabilities (the CDKML's online adaptation phase). The scenario (Table 6.8) contains two types of events: straightforward and complex events.

Straightforward (SF) 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 person 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

Table 6.8: Evaluation of CDKML's approach to combining domain knowledge and machine learning – fall-detection 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 behavior, 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

a low chair are present in the scenario. They differ in the position of the person's body on the chair: the person sits straight or leans forward, backward, to the left, or to the right. In complex fall events (5 and 6), the person slowly descends to the ground, trying to hold onto nearby furniture. However, after the falling activity, the person lands lying/sitting on the ground.

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 using the real-time localization system (RTLS) Ubisense (Ubisense, 2012). Each recording lasted around 20 minutes. The recordings were made by 5 healthy volunteers (3 male and 2 female), 5 times by each. Figure 6.9 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 (CDKML's third phase) primarily occurred on them.

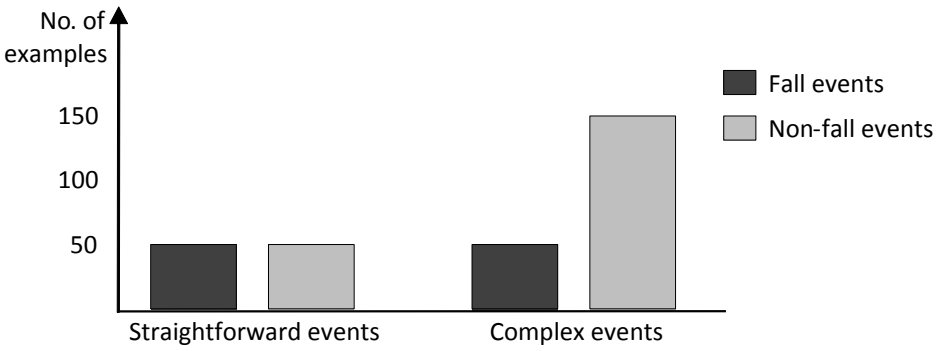

Figure 6.9: Number of fall and non-fall examples.

Results

This subsection compares the performance of CDKML's fall-detection classifier to the performance of ML fall-detection classifiers induced in Weka (Hall et al., 2009) with SMO, RandomForest, NaiveBayes, JRip and J48. The default algorithm parameter-values were used.

We evaluated the CDKML's initialization and refinement 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 perform laboratory testing). Fixed genetic-algorithm parameter values were used: population size of 40 individuals, crossover rate of 35 % and mutation rate of 8 %. The chromosome length (i.e., the number of classifier's parameters) equaled 28. The evaluation was performed using the leave-one-person-out scenario, 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 online classifier adaptation using Markov decision processes as proposed in the third CDKML phase as follows. The refined classifier was adapted to a concrete person using examples of both straightforward and complex concrete person 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 used in the CDKML's refinement phase. Four of the five concrete person 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 in case of a classification error and the fall-detection classifier was adapted, as necessary, before the next event. The false-positive penalty amount ($PaFp$) and the false-negative penalty amount ($PaFn$) were set to -1. The final adapted classifier evaluation was done on the recording, which was not used in the adaptation phase.

The ML approaches were evaluated using leave-one-person-out evaluation. The classifiers were induced from straightforward-event examples of four people and tested on both the straightforward and complex events of the fifth person. This was repeated five times, using a different person for testing each time. The instance attributes were the time since detecting the last person's falling activity, the amount of each type of posture in time intervals from 5 to 15 seconds and the amount of movement in this interval range. The attributes are equivalent to the parameters of the rules in the rule-based fall-detection classifier.

Accuracy was used for evaluating fall-detection classifier performance. Classifier's accuracy on a subset of events ACC_{events} is computed as:

$$ACC_{events} = \frac{\textit{correctly detected events of type } E \in \textit{events}}{\textit{all events of type } E \in \textit{events}} \tag{6.4}$$

Table 6.9 presents the performance of the induced fall-detection classifiers on the straightforward events only, on the 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} and overall accuracy ACC_{all}. Table 6.10 presents the accuracy of the induced classifiers on each event in the test scenario separately ACC_e. The accuracy was computed for each person separately, and the values in Tables 6.9 and 6.10 represent the averages.

Table 6.9: Fall-detection classifier comparison with respect to classifiers' accuracy on the fall examples ACC_f, classifiers' accuracy on the non-fall examples ACC_{nf} and overall accuracy ACC_{all}.

CLASSIFIER			ML					CDKML		
			J48	JRip	SMO	Random-Forest	Naive-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
Test 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
	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
	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

Table 6.9 shows that the best overall accuracy among the ML classifiers was obtained by SMO with an ACC_{all} of 0.53. 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 CDKML's initial classifier was 0.10 higher than SMO's. It slightly decreased on the ACC_f, from 1.00 to 0.98, but increased greatly on the ACC_{nf} from 0.30 to 0.46. The refinement of the initial classifier based on straightforward-event examples contributed to a 0.03 increase in accuracy. The ACC_{nf} increased to 0.53 at the cost of a slight decrease in ACC_f, which was 0.93. CDKML's adapted classifier outperformed the refined classifier in accuracy by 0.15; 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 person on which the tests were made. The adapted classifier had the highest ACC_{nf} (0.85) whereas its ACC_f had 0.71.

Table 6.10 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 DK to CDKML's initial classifier significantly improved ACC_e on the normal behavior 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. CDKML's adapted classifier correctly recognized almost all falls after which a person lay on the ground, but it had difficulties with falls after which a person 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 are the person's postures similar, but they can both last a long time, during which the person is

Table 6.10: Fall-detection classifier comparison with respect to classifiers' accuracy on each test-scenario event ACC_e separately.

CLASSIFIER/ ACC_e		STRAIGHTFORWARD TESTS				COMPLEX TESTS			
		FALLS		NON-FALLS		FALLS		NON-FALLS	
		Tripping (1)	Falling landing sitting (2)	Normal behavior (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
	Naive-Bayes	1.00	1.00	0.12	0.44	1.00	1.00	0.20	0.01
CDKML's initial classifier		1.00	0.96	0.96	0.68	0.96	1.00	0.96	0.22
CDKML's refined classifier		0.96	0.86	0.96	0.92	0.96	0.96	1.00	0.27
CDKML's adapted classifier		0.96	0.46	1.00	0.98	0.84	0.60	1.00	0.77

immovable. Some examples of sitting on a low chair are in fact indistinguishable from falls because of the noise in the measurements of the sensors used. Adapting the fall-detection classifier establishes a trade-off between these events. As sitting on a low chair is far more frequent then falls after which a person sits on the ground in a normal sitting position, misclassifications of this event are more costly. CDKML's adapted classifier is thus inclined to reduce the number of misclassifications during sitting on a low chair at the cost of not detecting certain falls after which a person lands sitting on the ground. In any case, person immovability after falls for additional or prolonged times should enable the detection of these false negatives; however, this is not within the scope of this evaluation that deals with fall detection within a reasonably short time.

6.3.2 Evaluation of CDKML's Online Classifier Adaptation

This subsection compares the CDKML's online adaptation phase to online ML classifier adaptation.

Data

Table 6.11 presents the set of events used for evaluating the CDKML's and the ML's online adaptation. Four fall events were used: (1) a person falls quickly and then lies on the ground moving for 15 s, (2) a person falls quickly and then lies immovable for 15 s, (3) a person falls slowly and then lies moving for 15 s, and (4) a person falls slowly and than lies immovable for 15 s. The cases (1) and (2) represent tripping; (2) results in an injury that prevents

Table 6.11: Evaluation of CDKML's online adaptation phase – fall-detection test scenario.

FALL EVENTS		NON-FALL EVENTS	
1	The person falls quickly and then lies on the ground moving for 15 s	5	The person is on all fours on the ground for 10 s
2	The person falls quickly and then lies immovable for 15 s	6	The person is on all fours for 5 s, then lies on the ground for 5 s
3	The person falls slowly and then lies moving for 15 s	7	The person lies on the ground for 10 s
4	The person falls slowly and then lies immovable for 15 s		

movement. The cases (3) and (4) represent falling due to dizziness or fainting. If these fall events were not detected by a fall-detection classifier, the person provided a false negative feedback after lying on the ground for 7 seconds. Three non-fall events were also recorded: (5) a person is on all fours on the ground for 10 s, (6) a person is on all fours for 5 s, then lies on the ground for 5 s, and (7) a person lies on the ground for 10 s. The person is moving in all three cases. These events may represent a person searching for things under the table or bed. They differ from the fall events by the length the person stays on the ground and in some cases the amount of movement. If a fall was detected during these events, the person provided a false positive feedback.

The events were recorded using the RTLS system Ubisense (Ubisense, 2012). For the purpose of training, five separate recordings of each fall and non-fall events were made. Additionally, for the purpose of testing, five recordings which encompassed all fall and non-fall events were made. The events in the test recordings were interspersed with short periods of walking. The recordings were made for one healthy volunteer.

CDKML

CDKML's fall-detection classifier had the form described in Section 5.2. Online adaptation was performed as described in Section 5.4.

ML

This evaluation uses the ML fall-detection module of the Confidence system. Falls are detected using two ML classifiers: (1) a decision tree created using J48 in Weka (Hall et al., 2009), and (2) a support-vector-machines classifier created using SMO in Weka (Hall et al., 2009). An example is classified as a fall if both classifiers classify it as such; otherwise it is considered as a non-fall. The classifiers' reasoning is based on the percentage of all observed person's postures and movement in periods of 5 s, 10 s and 15 s. These intervals are suitable for our experiments because we considered a reasonable period of lying at an inappropriate place after which a fall should be detected somewhere between 5 and 15 s. In real life a longer period might make more sense, in which case the intervals used in attributes should be lengthened.

Online ML classifier adaptation is performed by re-inducing both the J48 and SMO classifier each time a new fall or non-fall example is obtained. User feedback provides new training examples. In the case of a false negative, the example at the feedback point and all examples that follow it with longer amounts of person's lying on the ground are added as fall examples to the training dataset. In the case of a false positive, all the examples that

were incorrectly classified as a fall are added as non-fall examples to the training dataset. In order to escape classifier bias in the case of unbalanced dataset, after each addition, the weight of the dataset examples is updated in order to bring the ratio of fall to non-fall to neutral examples (includes standing, sitting and lying on the bed) to 40 to 30 to 30.

Results

We performed the adaptation test runs as follows. We started with classifiers that are not able to recognize any fall event in the test sequences. One training event was provided to the learning approaches (CDKML and ML) in each test step. If a provided fall training event was not detected as a fall event within 7 seconds, a false negative user feedback was triggered. If a provided non-fall training event was detected as a fall, a false positive user feedback was triggered. In the case of a false negative or a false positive user feedback, we adapted the CDKML and the ML classifiers. At the end of each test step, we measured two characteristics of the adapted classifiers on the five test sequences: accuracy (Equation (6.4)) and time-to-fall (the length of the interval from an event's start till a fall is detected by a fall-detection classifier).

The performance of CDKML's and ML's online adaptation procedures was evaluated with respect to accuracy. In order to test how classifiers' accuracy changes after each adaptation step (each new training event), three test runs were executed. In the first, the approaches were presented all fall events first, followed by all non-fall events. In the remaining two test runs, the fall and non-fall events were given randomly to CDKML's and ML's adaptation approaches. By adapting the classifiers by various training-event orders, we gain information not only how their fall-detection accuracy improves, but also how much it depends on a particular order of training scenarios.

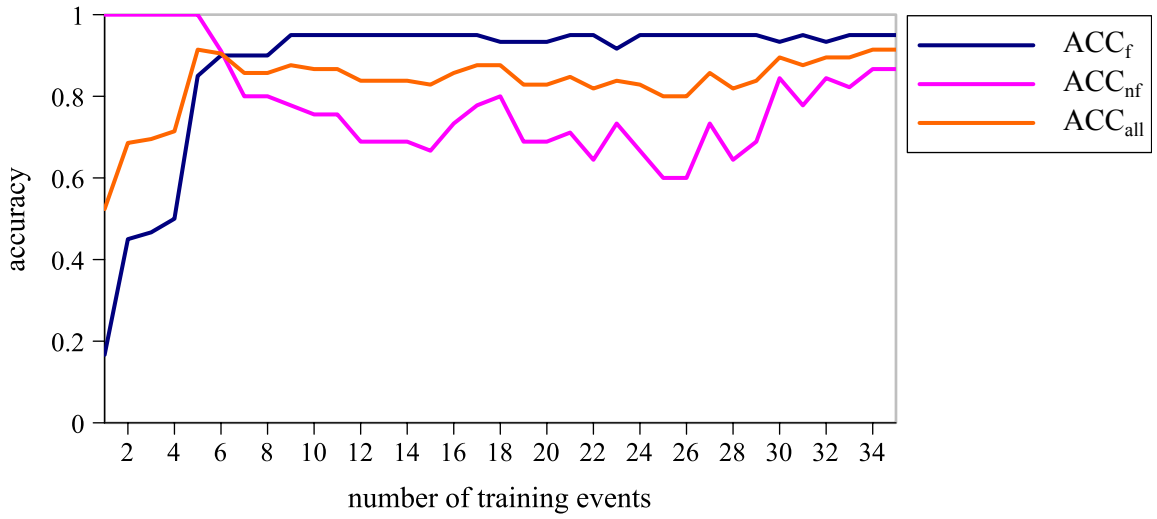

Figure 6.10: Evaluation of CDKML's online adaptation phase – fall-detection classifier's accuracy per adaptation step.

CDKML's fall-detection accuracy per adaptation step is presented in Figure 6.10, while Figure 6.11 plots ML's fall-detection accuracy per adaptation step. ACC_f represents classifier's accuracy on the fall events, ACC_{nf} is classifier's accuracy on the non-fall events, while ACC_{all} the overall classifier's accuracy. From this graph we can see that both CDKML and ML online adaptation approaches contribute to reliable fall-detection classifiers whose overall accuracy reaches 0.90.

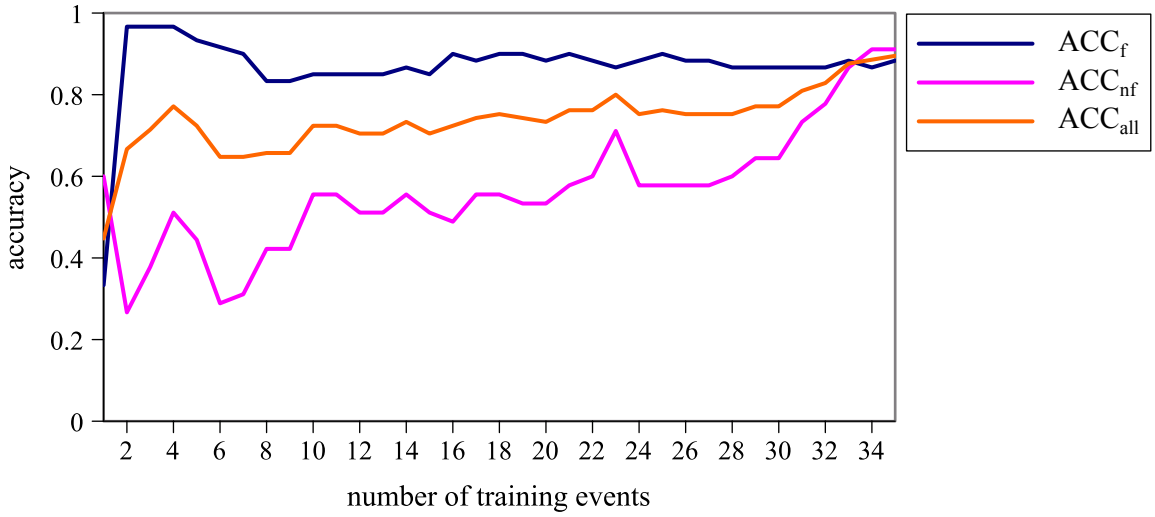

Figure 6.11: Evaluation of online ML classifier adaptation – fall-detection classifier's accuracy per adaptation step.

However, do both classifiers create the same representation of falls? As presented in Section 5.2, CDKML detects falls according to the period a person is lying or sitting on the ground. Therefore, we expected that the false negative feedback triggered in the fall events after 7 s of lying on the ground would cause CDKML's time-to-fall to be reduced to 7 s or less. However, if a fall is detected within 7 s of lying on the ground, some of the non-fall events may incorrectly be classified as falls (e.g., event 7 in Table 6.11 in which the person lied on the ground 10 s). Because of this, we expected that the non-fall events would cause CDKML's time-to-fall to increase to 10 s or more. In this experiment, CDKML's fall-detection classifier could achieve the highest accuracy only if its time-to-fall was in the interval (10 s, 15 s). How does the ML's classifier separate fall from non-fall events? Does its time-to-fall resemble CDKML's time-to-fall? We would like to note that the purpose of this experiment was not to make the classifiers' time-to-fall as low as possible, although this is strongly desired for practical applications. With rather wanted to examine if CDKML's online adaptation follows our expectations. We additionally wanted to test if the ML classifiers create the same representation of falls as CDKML.

We tested how the classifiers' time-to-fall changes by presenting all fall events to the classifiers first, followed by all non-fall events. Figure 6.12 presents how the classifiers' time-to-fall changes after each adaptation step. F_{CDKML} presents CDKML's average time-to-fall for all correctly detected fall events in the test sequences, whereas the NF_{CDKML} presents its average time-to-fall for the non-fall events incorrectly classified as falls. ML's average time-to-fall on the fall and non-fall events are presented with the lines F_{ML} and NF_{ML}, respectively. CDKML's time-to-fall follows our expectations. It decreased to around 4 s after CDKML was presented with all fall events, then increased to 8 s after the classifier received all non-fall events. We would like to note that due to errors in the classification of the lying posture, the posture-recognition module could not detect the entire 10 s period of lying in event 7 of Table 6.11 (the non-fall event which contained the longest period of lying on the ground). ML's time-to-fall, on the other hand, does not follow our expectations. It stayed fairly constant around 8 s when the fall events were presented to the classifiers, then started to fall when the non-fall events were presented to it. Time-to-fall fell to 4 s in the end. This means that the ML classifier learned to separate fall from non-fall events not

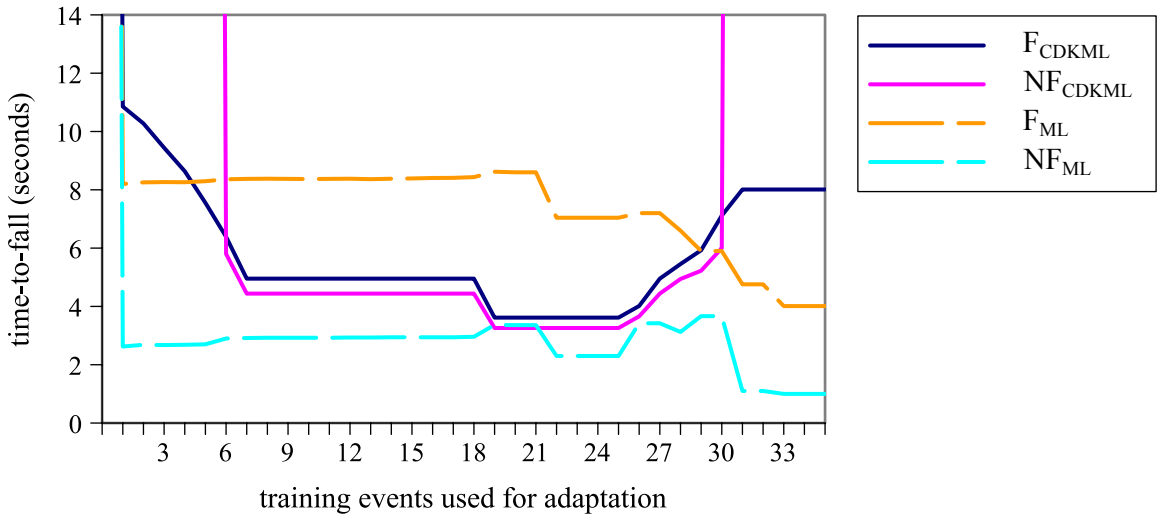

Figure 6.12: Evaluation of classifier's online adaptation – classifiers' time-to-fall on the fall events (F_{CDKML} and F_{ML}) and on the non-fall events (NF_{CDKML} and NF_{ML}).

according to the length of person lying on the ground, but according to an other feature. CDKML's and ML's fall-detection classifiers represent two separate viewpoints of falls.

6.4 Discussion

This section analyzes the results of the presented experiments with respect to the three questions stated at the beginning of Chapter 6.

How important is expert input in the CDKML method?

CDKML showed the best performance in the fall-detection domain where it considerably outperformed all five ML algorithms, the posture-recognition domain followed, while it did not show improvement in comparison to standard ML in the behavioral-cloning domain. We attribute the improvement in performance primarily to the contribution of the expert in CDKML's initialization phase, where the expert extracted the classifier patterns using DK and interactive data mining. The improvement was the most evident in the fall-detection domain where DK provided clear instructions: "If a person is lying or sitting on the ground for a long period of time then a fall happened". Formulating the patterns for the posture-recognition classifier was, however, not simple. In this case, interactive data mining played an important role, helping the expert to incorporate DK into the classifier. In the behavioral-cloning domain, we did not include DK.

Is an expert capable of selecting a comprehensive set of concept patterns in the CDKML method?

It is well documented that experts have problems formulating their knowledge on their own, a problem often referred to as the Feigenbaums bottleneck (Feigenbaum, 1981). One approach to overcoming this problem is through the use of ML tools (Michie and Bratko, 1986). In the CDKML method, interactive data mining eases knowledge acquisition from experts. Interactive data mining provides smart hypothesis-space examination focusing expert's attention on the most promising patterns.

Extracting the classifier's patterns using interactive data mining may, however, be more time consuming than classical ML. In practical terms these demands were not too severe. A

few days were needed for formulating the patterns in the posture-recognition domain, while the patterns in the fall-detection domain were defined within a few hours.

Does CDKML's approach to combining DK and ML contribute to improved classifier's performance?

CDKML's refined classifier achieved higher accuracy than the ML classifiers in the posture-recognition and the fall-detection domains. The evaluation of CDKML's online adaptation phase in the fall-detection domain shows that the proposed approach is capable of adjusting the refined classifier to correctly recognize events not present in the training dataset, making trade-offs between contradictory examples based on the cost of each misclassification. CDKML's adapted classifier achieved higher accuracy than CDKML's refined classifier.

Classifier generation using CDKML may, however, last longer than classifier generation using ML. The most time-consuming domain was the posture-recognition domain. Both CDKML's initialization and refinement phases lasted a few days in this domain. Nevertheless, time efficiency is not critical for these two phases because they are performed offline, before the classifier is deployed in a system. CDKML's online adaptation approach, on the other hand, requires a few minutes to complete, enabling real-time classifier adaptation.

7 Conclusions

The dissertation addresses the problem of classifier generation from a training dataset that captures a limited subset of the real-life cases of the learned concept. Despite the exponential growth of digital data, there are still domains for which only a limited number of examples is available. We assume there are at least two reasons for this. First, sufficient general-purpose data may be costly or otherwise difficult to obtain. A typical example is studies in the medical domain. Obtaining data for falls, for example, is costly because of ethical issues and injury danger. Second, general-purpose data may be inappropriate if the deployment needs to be adjusted to the characteristics of a particular person. Such deployments typically require online data collection and classifier adaptation.

We present a novel method for classifier generation from a training dataset that does not adequately represent all real life cases of the learned concept. In such cases it is important to take into consideration all available DK in the learning process. While ML may discover patterns in interest domains that are too subtle to be detected by humans, DK may contain information on a domain not present in the available domain dataset. The proposed method, named CDKML, considers a novel approach to combining DK and ML.

CDKML is founded on the hypothesis that a combination of interactive data mining to extract a comprehensive set of characteristic concept patterns and optimization algorithms to determine the optimal pattern-parameter values (general-purpose and deployment-specific) is needed for creation of reliable classifiers in domains for which a limited amount of concept examples is available. It encompasses three phases: initialization, refinement and online adaptation.

The initialization phase is devoted to extraction of characteristic concept patterns. We showed that an expert is capable of selecting a comprehensive set of concept patterns using DK and interactive data mining in the posture-recognition domain.

The refinement phase is devoted to finding the most suitable general-purpose pattern-parameter values. An optimization algorithm is used in order to find the parameter values which maximize the classifier's accuracy on the available training examples. In the posture-recognition and fall-detection domains, we showed that the refined classifiers have higher accuracy then the rule-based classifiers (consisting of rules or classifiers which can be converted to a ruleset) induced using ML: decision trees, a set of rules and random-forest classifiers. In the fall-detection domain, the refined-classifier's accuracy was also higher than the support-vector-machines's accuracy (overall, the support vector machines had the highest accuracy among the ML classifiers).

The online adaptation phase is devoted to finding the most suitable deployment-specific pattern-parameter values. Markov decision processes are used for fine-tuning the parameter values to user's needs and preferences obtained through user feedback. In the fall-detection domain we showed that the proposed approach is capable of adjusting the classifier to correctly recognize events not present in the training dataset, making trade-offs between contradictory examples based on the cost of each misclassification.

CDKML was applied to classifiers in the form of a set of rules. We would like to note that CDKML is not bound to this specific classifier form; however, it requires a human-

understandable form. We plan to examine CDKML's performance using other classifier forms (e.g., decision trees) as future work.

As future work, we also plan to examine two CDKML improvements. First, exploitation of DK captured in ontologies needs to be considered. The Web offers huge amounts of unstructured, textual data. Approaches to extracting domain patterns and ontology development from that kind of data are emerging (Dalvi et al., 2012). It would be interesting to research possibilities for automating CDKML's initialization by utilizing DK available on the Web. Second, the online classifier adaptation relies of adjustment of the parameter values of the refined classifier as user feedback is obtained. However, as more real-life cases of the learned concept become available, the better the capability of ML to induce a reliable concept classifier. Therefore, simultaneous adaptation of the refined classifier may be accompanied by reinducing the ML classifiers. A combination of the two classifiers in which the ML classifier's influence on the final classification increases as more data becomes available seems reasonable.

8 Acknowledgments

The dissertation would not have been possible without the generous help and support of my colleagues and family.

First of all, I would like to thank my supervisor Prof. Dr. Matjaž Gams and co-supervisor Dr. Mitja Luštrek, who have provided guidance, support, understanding, and professional and personal assistance of the most valuable kind.

I am thankful to my colleagues from the Department of Intelligent Systems at the Jožef Stefan Institute for all extensive discussions and insightful comments. With regards to the posture-recognition and the fall-detection domain, studied within the FP7 project Confidence, I would especially like to thank the Ambient Intelligence group members, in particular Dr. Boštjan Kaluža, Erik Dovgan, Rok Piltaver, Božidara Cvetković, Domen Zupančič and Bogdan Pogorelc. With regards to the behavioral-cloning domain, studied within the EUSAS project, I would especially like to thank the Agent group members, in particular Aleš Tavčar, Erik Dovgan and Damjan Kužnar.

I am thankful to my colleagues from Result d.o.o. for a very pleasant cooperation throughout my doctoral studies, in particular to Franc Škedelj and Igor Korelič.

I would like to thank Dr. Vedrana Vidulin for all the help regarding the study-related and living-related formalities.

I would like to thank my family for all the support throughout my studies.

Last but not least, I am grateful to the Department of Intelligent Systems at the Jožef Stefan Institute, Result d.o.o. and the Slovenian Technology Agency for providing me a funding, which made the dissertation possible. The research leading to the dissertation was partially financed by the European Union, European Social Fund.

9 References

Aler, R.; Valls, J. M.; Camacho, D.; Lopez, A. Programming robosoccer agents by modeling human behavior. *Expert Systems with Applications* **36**, 1850–1859 (2009).

Argall, B. D.; Chernova, S.; Veloso, M.; Browning, B. A survey of robot learning from demonstration. *Robotics and Autonomous Systems* **57**, 469–483 (2009).

Asian, O.; Yildiz, O. T.; Alpaydin, E. Calculating the VC-dimension of decision trees. In: *Proceedings of the* 24^{th} *International Symposium on Computer and Information Sciences*. 841–851 (IEEE, 2009).

Avci, A.; Bosch, S.; Marin-Perianu, M.; Marin-Perianu, R.; Havinga, P. Activity recognition using inertial sensing for healthcare, wellbeing and sports applications: A survey. In: *Proceedings of the* 23^{th} *International Conference on Architecture of Computing Systems*. 167–176 (VDE Verlag, Berlin, Germany, 2010).

Bežek, A. *Avtomatsko modeliranje večagentnih sistemov*. Ph.D. thesis (University of Ljubljana, Faculty of Computer and Information Science, Slovenia, 2006).

Blum, A.; Mitchell, T. Combining labeled and unlabeled data with co-training. In: *Proceedings of the* 11^{th} *Annual Conference on Computational Learning Theory*. 92–100 (ACM, New York, NY, USA, 1998).

Bohemia Interactive Australia. Virtual battlespace 2. http://www.vbs2.com (accessed: February 2013).

Bratko, I.; Urbančič, T. Transfer of control skill by machine learning. *Engineering Applications of Artificial Intelligence* **10**, 63–71 (1997).

Breiman, L. Random forests. *Machine Learning* **45**, 5–32 (2001).

Burns, B. D.; Danyluk, A. P. Feature selection vs. theory reformulation: A study of genetic refinement of knowledge-based neural networks. *Machine Learning* **38**, 89–107 (2000).

Caragea, D.; Cook, D.; Wickham, H.; Honavar, V. Visual methods for examining SVM classifiers. In: *Visual Data Mining*. 136–153 (Springer-Verlag, Berlin Heidelberg, Germany, 2008).

Chan, P. K.; Fan, W.; Prodromidis, A. L.; Stolfo, S. J. Distributed data mining in credit card fraud detection. *IEEE Intelligent Systems* **14**, 67–74 (1999).

Chang, C.-C.; Lin, C.-J. LibSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology* **2**, 27:1–27:27 (2011).

Clark, P.; Niblett, T. The CN2 induction algorithm. *Machine Learning* **3**, 261–283 (1989).

Coates, A.; Abbeel, P.; Ng, A. Y. Autonomous helicopter flight using reinforcement learning. In: *Encyclopedia of Machine Learning*. 53–61 (Springer-Verlag New York, Inc., New York, NY, USA, 2010).

Cohen, W. W. Fast effective rule induction. In: *Proceedings of the* 12^{th} *International Conference on Machine Learning*. 115–123 (Morgan Kaufmann, San Francisco, CA, USA 1995).

European Commission. Eurostat. http://ec.europa.eu/eurostat (accessed: November 2012).

Confidence project. http://www.confidence-eu.org/ (accessed: November 2012).

Dalvi, B.; Cohen, W. W.; Callan, J. Collectively representing semi-structured data from the web. In: *Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-Scale Knowledge Extraction*. 7–12 (Association for Computational Linguistics, Stroudsburg, PA, USA, 2012).

Dasgupta, S. Two faces of active learning. *Theoretical Computer Science* **412**, 1767–1781 (2011).

Decoste, D.; Schölkopf, B. Training invariant support vector machines. *Machine Learning* **46**, 161–190 (2002).

Eiben, A. E.; Smith, J. E. *Introduction to Evolutionary Computing* (Springer-Verlag, Berlin Heidelberg, Germany, 2003).

eMotion. Smart motion capture system. http://www. emotion3d.com/smart/smart.html (accessed: April 2009).

Fayyad, U. M.; Irani, K. B. Multi-interval discretization of continuous-valued attributes for classification learning. In: Bajcsy, R. (ed.) *Proceedings of the International Joint Conference on Uncertainty in AI*. 1022–1027 (Morgan Kaufmann, San Francisco, CA, USA, 1993).

Feigenbaum, E. A. Expert systems in the 1980s. In: Bond, A. (ed.) *Infotech State of the Art Report on Machine Intelligence*. 27–52 (Pergamon Infotch Ltd, Maidenhead, England, 1981).

Feldman, R. *Understanding Psychology* (McGraw-Hill Higher Education, Columbus, OH, USA, 2005).

Ferber, J. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence* (Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999).

Graepel, T.; Herbrich, R. Invariant pattern recognition by semi-definite programming machines. In: Thrun, S.; Saul, L.; Schölkopf, B. (eds.) *Advances in Neural Information Processing Systems 16*. 33–40 (MIT Press, Cambridge, MA, USA, 2004).

Guestrin, C. Lecture notes, Carnegie Mellon University (ML course No: 10701/15781). http://www.cs.cmu.edu/ guestrin/Class/15781/slides/learningtheory-bigpicture.pdf (accessed: February 2013).

Haasdonk, B.; Vossen, A.; Burkhardt, H. Invariance in kernel methods by Haar-integration kernels. In: *Proceedings of the* 14^{th} *Scandinavian Conference on Image Analysis*. 841–851 (Springer-Verlag, Berlin Heidelberg, Germany, 2005).

Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; Witten, I. H. The Weka data mining software: An update. *SIGKDD Explorations Newsletter* **11**, 10–18 (2009).

Heit, E. Background knowledge and models of categorization. In: Hahn, U.; Ramscar, M. (eds.) *Similarity and Categorization.* 155–178 (Oxford University Press, New York, NY, USA, 2000).

Hollnagel, E. *Human Reliability Analysis: Context and Control* (Academic Press, Waltham, MA, USA, 1993).

Hollnagel, E. Modelling the orderliness of human action. In: Amalberti, R.; Sarter, N. (eds.) *Cognitive Engineering in the Aviation Domain.* 65–98 (Lawrence Erlbaum Associates, NJ, USA, 2000).

Holmes, J. H.; Lanzi, P. L.; Stolzmann, W.; Wilson, S. W. Learning classifier systems: New models, successful applications. *Information Processing Letters* **82**, 23–30 (2002).

Hu, B.-G.; Qu, H.-B.; Wang, Y.; Yang, S.-H. A generalized-constraint neural network model: Associating partially known relationships for nonlinear regressions. *Information Sciences* **179**, 1929–1943 (2009).

Jäkel, F.; Schölkopf, B.; Wichmann, F. A. A tutorial on kernel methods for categorization. http://www.is.tuebingen.mpg.de/fileadmin/user_upload/files/publications/Jakel_etal_2007Preprint_4784[0].pdf (accessed: December 2007).

Jin, F.; Sun, S. A multitask learning approach to face recognition based on neural networks. In: *Proceedings of the* 9^{th} *International Conference on Intelligent Data Engineering and Automated Learning.* 24–31 (Springer-Verlag, Berlin Heidelberg, Germany, 2008).

John, G. H.; Langley, P. Estimating continuous distributions in bayesian classifiers. In: *Proceedings of the* 11^{th} *Conference on Uncertainty in Artificial Intelligence.* 338–345 (Morgan Kaufmann, San Francisco, CA, USA, 1995).

Kaluža, B.; Mirchevska, V.; Dovgan, E.; Luštrek, M.; Gams, M. An agent-based approach to care in independent living. In: *Proceedings of the* 1^{st} *International Joint Conference on Ambient Intelligence.* 177–186 (Springer-Verlag, Berlin Heidelberg, Germany, 2010).

Kaluža, B.; Cvetković, B.; Dovgan, E.; Gjoreski, H.; Mirchevska, V.; Gams, M.; Luštrek, M. A multi-agent care system to support independent living. *International Journal of Artificial Intelligence Tools*, in press (2013).

Kambar, S. *Generating Synthetic Data by Morphing Transformation for Handwritten Numeral Recognition (With v-SVM).* Master's thesis (Concordia University, Computer Science Department, Montreal, Canada, 2005).

Keerthi, S. S.; Shevade, S. K.; Bhattacharyya, C.; Murthy, K. R. K. Improvements to Platt's SMO algorithm for SVM classifier design. *Neural Computation* **13**, 637–649 (2001).

Kondor, R.; Jebara, T. A kernel between sets of vectors. In: *Proceedings of the* 20^{th} *International Conference on Machine Learning.* 361–368 (AAAI, Menlo Park, CA, USA, 2003).

Lauer, F.; Bloch, G. Incorporating prior knowledge in support vector machines for classification: A review. *Neurocomputing* **71**, 1578–1594 (2008).

Lavrač, N.; Džeroski, S. *Inductive Logic Programming: Techniques and Applications* (Routledge, New York, NY, USA, 1993).

Lettmann, T.; Baumann, M.; Eberling, M.; Kemmerich, T. Modeling agents and agent systems. In: *Transactions on Computational Collective Intelligence*. 157–181 (Springer-Verlag, Berlin Heidelberg, Germany, 2011).

Liu, Y.; Salvendy, G. Interactive visual decision tree classification. In: *Proceedings of the* 12^{th} *International Conference on Human-Computer Interaction: Interaction Platforms and Techniques*. 92–105 (Springer-Verlag, Berlin Heidelberg, Germany, 2007).

Loosli, G.; Canu, S.; Vishwanathan, S. V. N.; Smola, A. J. Invariances in classification: An efficient SVM implementation. In: *Proceedings of the* 11^{th} *International Symposium on Applied Stochastic Models and Data Analysis*. 543–551 (ENST Bretagne, France, 2005).

Luštrek, M.; Gjoreski, H.; Kozina, S.; Cvetković, B.; Mirchevska, V.; Gams, M. Detecting falls with location sensors and accelerometers. In: *Proceedings of the* 23^{rd} *Innovative Applications of Artificial Intelligence Conference*. 1662–1667 (AAAI, Menlo Park, CA, USA, 2011).

Meffert, K et al. JGAP – Java Genetic Algorithms and Genetic Programming Package. http://jgap.sf.net. (accessed: June 2011).

Michie, D.; Bratko, I. *Expert Systems: Automating Knowledge Acquisition* (Addison-Wesley, Boston, MA, USA, 1986).

Mirchevska, V.; Bežek, A.; Luštrek, M.; Gams, M. Discovering strategic behaviour of multi-agent systems in adversary settings. *Computing and Informatics*, in press (2013a).

Mirchevska, V.; Kaluža, B.; Luštrek, M.; Gams, M. Real-time alarm model adaptation based on user feedback. In: *Workshop on Ubiquitous Data Mining in conjunction with the* 19^{th} *European Conference on Artificial Intelligence*. 39–43 (Lisbon, 2010).

Mirchevska, V.; Luštrek, M.; Gams, M. Combining domain knowledge and machine learning for robust fall detection. *Expert Systems*, preprint published online (2013b).

Mirčevska, V.; Luštrek, M.; Vélez, I.; Vega, N. G.; Gams, M. Classifying posture based on location of radio tags. In: Čech, P.; Bureš, V.; Nerudová, L. (eds.) *Ambient Intelligence and Smart Environments: Ambient Intelligence Perspectives II*. 85–92 (IOS Press, Amsterdam, The Netherlands, 2009).

Mirchevska, V.; Tavčar, A.; Gams, M. Bahavioral cloning of asymmetric conflicts in urban environment using supervised learning. In: Bohanec, M.; Gams, M.; Mladenić, D.; Grobelnik, M.; Heričko, M.; Kordeš, U.; Smrdu, M.; Markič, O.; Pirtošek, Z.; Lenarčič, J.; Žlajpah, L.; Gams, A.; Rajkovič, V.; Urbančič, T.; Bernik, M. (eds.) *Proceedings of the* 15^{th} *International Multiconference Information Society*. 134–137 (Jožef Stefan Institute, Ljubljana, Slovenia, 2012).

Mitchell, T. M. *Machine Learning* (McGraw-Hill, Inc., New York, NY, USA, 1997).

Mooney, R. J.; Roy, L. Content-based book recommending using learning for text categorization. In: *Proceedings of the* 5^{th} *ACM Conference on Digital Libraries*. 195–204 (ACM, New York, NY, USA, 2000).

Možina, M. *Argument-Based Machine Learning*. Ph.D. thesis (University of Ljubljana, Faculty of Computer and Information Science, Slovenia, 2009).

Muggleton, S. Inverse entailment and Progol. *New Generation Computing: Special issue on Inductive Logic Programming* **13**, 245–286 (1995).

Müller, M. Dynamic time warping. In: *Information Retrieval for Music and Motion*. 69–84 (Springer-Verlag, Berlin Heidelberg, Germany, 2007).

Ng, A. Y.; Kim, H. J.; Jordan, M. I.; Sastry, S. Inverted autonomous helicopter flight via reinforcement learning. In: *International Symposium on Experimental Robotics*. 1–10 (MIT Press, Cambridge, MA, USA, 2004).

Ng, A. Y.; Russell, S. J. Algorithms for inverse reinforcement learning. In: *Proceedings of the* 17^{th} *International Conference on Machine Learning*. 663–670 (Morgan Kaufmann, San Francisco, CA, USA, 2000).

Niyogi, P.; Girosi, F.; Poggio, T. Incorporating prior information in machine learning by creating virtual examples. *Proceedings of the IEEE* **86**, 2196–2209 (1998).

Osei-Bryson, K.-M. Evaluation of decision trees: A multi-criteria approach. *Computers and Operations Research* **31**, 1933–1945 (2004).

Pan, S. L.; Scarbrough, H. Knowledge management in practice: An exploratory case study. *Technology Analysis Strategic Management* **11**, 359–374 (1999).

Pazzani, M.; Brunk, C. Finding accurate frontiers: A knowledge-intensive approach to relational learning. In: *Proceedings of the* 11^{th} *National Conference on Artificial Intelligence*. 328–334 (Morgan Kaufmann, San Francisco, CA, USA, 1993).

Pichuka, C.; Bapi, R. S.; Bhagvati, C.; Pujari, A. K.; Deekshatulu, B. L. A tighter error bound for decision tree learning using PAC learnability. In: *Proceedings of the* 20^{th} *International Joint Conference on Artificial Intelligence*. 1011–1016 (Morgan Kaufmann, San Francisco, CA, USA, 2007).

Poulet, F.; Do, T.-N. Interactive decision tree construction for interval and taxonomical data. In: *Visual Data Mining*. 123–135 (Springer-Verlag, Berlin Heidelberg, Germany, 2008).

Pozdnoukhov, A.; Bengio, S. Tangent vector kernels for invariant image classification with SVMs. In: *Proceedings of the* 17^{th} *International Conference on Pattern Recognition*. 486–489 (IEEE Computer Society, Washington, DC, USA, 2004).

Pyle, D. *Business Modeling and Data Mining* (Morgan Kaufmann, San Francisco, CA, USA, 2003).

Quinlan, J. R. Learning logical definitions from relations. *Machine Learning* **5**, 239–266 (1990).

Quinlan, J. R. *C4.5: Programs for machine learning* (Morgan Kaufmann, San Francisco, CA, USA, 1993).

Rossetti, R. J. F.; Bordini, R. H.; Bazzan, A. L. C.; Bampi, S.; Liu, R.; Van Vliet, D. Using BDI agents to improve driver modelling in a commuter scenario. *Transportation Research Part C: Emerging Technologies* **10**, 47–72 (2002).

Russell, S.; Norvig, S. *Artificial Intelligence: A Modern Approach* (Prentice Hall, NJ, USA, 2010).

Sabzekar, M.; Sadoghi Yazdi, H.; Naghibzadeh, M. Relaxed constraints support vector machines for noisy data. *Neural Computing and Applications* **20**, 671–685 (2011).

Sagun, A.; Bouchlaghem, D.; Anumba, C. J. Computer simulations vs. building guidance to enhance evacuation performance of buildings during emergency events. *Simulation Modelling Practice and Theory* **19**, 1007–1019 (2011).

Schadd, F.; Bakkes, S.; Spronck, P. Opponent modeling in real-time strategy games. In: Roccetti, M. (ed.) *AI and Simulation in Games*. 61–70 (EUROSIS, Ostend, Belgium, 2007).

Schmidt, B. *The Modelling of Human Behaviour: The PECS Reference Models* (SCS-Europe BVBA, Erlangen, Germany, 2000).

Schölkopf, B.; Burges, C.; Vapnik, V. Incorporating invariances in support vector learning machines. In: *Proceedings of the 1996 International Conference on Artificial Neural Networks*. 47–52 (Springer-Verlag, Berlin Heidelberg, Germany, 1996).

Shivaswamy, P. K.; Jebara, T. Permutation invariant SVMs. In: *Proceedings of the* 23^{rd} *International Conference on Machine Learning*. 817–824 (ACM, New York, NY, USA, 2006).

Simard, P.; LeCun, Y.; Denker, J. S. Efficient pattern recognition using a new transformation distance. In: *Advances in Neural Information Processing Systems 5*. 50–58 (Morgan Kaufmann, San Francisco, CA, USA, 1993).

Simoff, S. J.; Böhlen, M. H.; Mazeika, A. Visual Data Mining: An Introduction and Overview. In: *Visual data mining*. 1–12 (Springer-Verlag, Berlin Heidelberg, Germany, 2008).

Smit, S. K.; Eiben, A. E. Comparing parameter tuning methods for evolutionary algorithms. In: *Proceedings of the* 11^{th} *Congress on Evolutionary Computation*. 399–406 (IEEE Press, Piscataway, NJ, USA, 2009).

Srinivasan, A. A learning engine for proposing hypotheses (Aleph). http://www.cs.ox.ac.uk/activities/machlearn/Aleph/aleph.html (accessed: April 2013).

Stone, P. *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer* (MIT Press, Cambridge, MA, USA, 2000).

Stumpf, S.; Rajaram, V.; Li, L.; Wong, W.-K.; Burnett, M.; Dietterich, T.; Sullivan, E.; Herlocker, J. Interacting meaningfully with machine learning systems: Three experiments. *International Journal of Human-Computer Studies* **67**, 639–662 (2009).

Sun, S.; Hardoon, D. R. Active learning with extremely sparse labeled examples. *Neurocomputing* **73**, 2980–2988 (2010).

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

Thurau, C.; Sagere, G.; Bauckhage, C. Imitation learning at all levels of game AI. In: *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education*. 402–408 (University of Wolverhampton, School of Computing and Information Technology, UK, 2004).

Towell, G. G.; Shavlik, J. W. Knowledge-based artificial neural networks. *Artificial Intelligence* **70**, 119–165 (1994).

Ubisense. http://www.ubisense.net/ (accessed: November 2012).

Vapnik, V.; Chapelle, O. Bounds on error expectation for SVM. In: Smola, A.; Bartlett, P.; Schölkopf, B.; Schuurmans, D. (eds.) *Advances in Large Margin Classifiers.* 261–280 (MIT Press, Cambridge, MA, USA, 2000).

Vapnik, V. N. *The Nature of Statistical Learning Theory* (Springer-Verlag New York, Inc., New York, NY, USA, 1995).

Vidulin, V. *Searching for Credible Relations in Machine Learning, Slovenia.* Ph.D. thesis (Jožef Stefan International Postgraduate School, Slovenia, 2012).

Vidulin, V.; Gams, M. Impact of high-level knowledge on economic welfare through interactive data mining. *Applied Artificial Intelligence* **25**, 267–291 (2011).

Walker, T.; O'Reilly, C.; Kunapuli, G.; Natarajan, S.; Maclin, R.; Page, D.; Shavlik, J. Automating the ILP setup task: Converting user advice about specific examples into general background knowledge. In: *Proceedings of the* 20^{th} *International Conference on Inductive Logic Programming.* 253–268 (Springer-Verlag, Berlin Heidelberg, Germany, 2011).

Wang, L.; Gao, Y.; Chan, K.-L.; Xue, P.; Yau, W.-Y. Retrieval with knowledge-driven kernel design: An approach to improving SVM-based CBIR with relevance feedback. In: *Proceedings of the* 10^{th} *IEEE International Conference on Computer Vision.* 1355–1362 Vol. 2 (Institute of Electrical and Electronics Engineers Inc., USA, 2005).

Weber, W.; Rabaey, J.; Aarts, E. H. L. *Ambient Intelligence* (Springer-Verlag, Berlin Heidelberg, Germany, 2010).

Wisniewski, E. J.; Medin, D. L. On the interaction of theory and data in concept learning. *Cognitive Science* **18**, 221–281 (1994).

Wooldridge, M. J. *An Introduction to Multiagent Systems* (John Wiley & Sons, Chichester, West Sussex, UK, 2009).

Yu, T. *Incorporating Prior Domain Knowledge into Inductive Machine Learning: Its Implementation in Contemporary Capital Markets.* Ph.D. thesis (University of Technology, Faculty of Information Technology, Sydney, Australia, 2007).

Zhang, Q.; Sun, S. Multiple-view multiple-learner active learning. *Pattern Recognition* **43**, 3113–3119 (2010).

Zhao, Y. On interactive data mining. In: *Encyclopedia of Data Warehousing and Mining.* 1085–1090 (IGI Global, Hershey, PA, USA, 2009).

Zheng, V. W.; Hu, D. H.; Yang, Q. Cross-domain activity recognition. In: *Proceedings of the* 11^{th} *International Conference on Ubiquitous Computing.* 61–70 (ACM, New York, NY, USA, 2009).

List of Figures

List of Tables

List of Algorithms

Appendices

Appendix A: Bibliography

Publications related to the dissertation

Journal papers (SCI)

- Mirchevska, V.; Bežek, A.; Luštrek, M.; Gams, M. Discovering strategic behaviour of multi-agent systems in adversary settings. *Computing and Informatics*, in press (2013).

- Mirchevska, V.; Luštrek, M.; Gams, M. Combining domain knowledge and machine learning for robust fall detection. *Expert Systems*, preprint published online (2013).

- Kaluža, B.; Cvetković, B.; Dovgan, E.; Gjoreski, H.; Mirchevska, V.; Gams, M.; Luštrek, M. A multi-agent care system to support independent living. *International Journal of Artificial Intelligence Tools*, in press (2013).

Conference papers

- Mirchevska, V.; Tavčar, A.; Gams, M. Bahavioral cloning of asymmetric conflicts in urban environment using supervised learning. In: Bohanec, M.; Gams, M.; Mladenić, D.; Grobelnik, M.; Heričko, M.; Kordeš, U.; Smrdu, M.; Markič, O.; Pirtošek, Z.; Lenarčič, J.; Žlajpah, L.; Gams, A.; Rajkovič, V.; Urbančič, T.; Bernik, M. (eds.) *Proceedings of the* 15^{th} *International Multiconference Information Society.* 134–137 (Jožef Stefan Institute, Ljubljana, Slovenia, 2012).

- Luštrek, M.; Gjoreski, H.; Kozina, S.; Cvetković, B.; Mirchevska, V.; Gams, M. Detecting falls with location sensors and accelerometers. In: *Proceedings of the* 23^{rd} *Innovative Applications of Artificial Intelligence Conference.* 1662–1667 (AAAI, Menlo Park, CA, USA, 2011).

- Mirchevska, V.; Luštrek, M.; Gams, M. Towards robust fall detection. In: Bohanec, M.; Gams, M.; Mladenić, D.; Grobelnik, M.; Heričko, M.; Kordeš, U.; Markič, O.; Lenarčič, J.; Žlajpah, L.; Gams, A.; Fomichov, V.; Fomichova, O. S.; Brodnik, A.; Sosič, R.; Rajkovič, V.; Urbančič, T.; Bernik, M. (eds.) *Proceedings of the* 14^{th} *International Multiconference Information Society.* 75–78 (Jožef Stefan Institute, Ljubljana, Slovenia, 2011).

- Kaluža, B.; Mirchevska, V.; Dovgan, E.; Luštrek, M.; Gams, M. An agent-based approach to care in independent living. In: *Proceedings of the* 1^{st} *International Joint Conference on Ambient Intelligence.* 177–186 (Springer-Verlag, Berlin Heidelberg, Germany, 2010).

- Mirchevska, V.; Kaluža, B.; Luštrek, M.; Gams, M. Real-time alarm model adaptation based on user feedback. In: *Workshop on Ubiquitous Data Mining in conjunction with the* 19^{th} *European Conference on Artificial Intelligence.* 39–43 (Lisbon, 2010).

- Mirchevska, V. Alarm detection in the Confidence system. In: Vélez, I.; Gams, M. (eds.) *Odprta delavnica projekta Confidence: Proceedings of the* 13^{th} *International Multiconference Information Society.* 26–29 (Jožef Stefan Institute, Ljubljana, Slovenia, 2010).

- Mirchevska, V.; Kaluža, B. Learning through interaction. In: Kaluža, B.; Eleršič, K.; Pogorelc, B.; Šetina, B.; Vahčič, M. (eds.) *Proceedings of the* 2^{nd} *Jožef Stefan International Postgraduate School Students Conference.* 30–31 (Jožef Stefan International Postgraduate School, Ljubljana, Slovenia, 2010).

- Mirčevska, V.; Luštrek, M.; Vélez, I.; Vega, N. G.; Gams, M. Classifying posture based on location of radio tags. In: Čech, P.; Bureš, V.; Nerudová, L. (eds.) *Ambient Intelligence and Smart Environments: Ambient Intelligence Perspectives II.* 85–92 (IOS Press, Amsterdam, The Netherlands, 2009).

- Mirčevska, V.; Gams, M. Towards robust engine for classifying human posture. In: Bohanec, M.; Gams, M.; Rajkovič, V.; Urbančič, T.; Bernik, M.; Mladenić, D.; Grobelnik, M.; Heričko, M.; Kordeš, U.; Markič, O.; Lenarčič, J.; Žlajpah, L.; Gams, A.; Fomichova, O. S.; Fomichov, V.; Brodnik, A. (eds.) *Proceedings of the* 12^{th} *International Multiconference Information Society.* 112–115 (Jožef Stefan Institute, Ljubljana, Slovenia, 2009).

- Mirčevska, V.; Luštrek, M.; Gams, M. Combining machine learning and expert knowledge for classifying human posture. In: Zajc, B.; Trost, A. (eds.) *Zbornik 18. mednarodne elektrotehniške in računalniške konference.* 183–186 (Slovenska sekcija IEEE, Ljubljana, Slovenija, 2009a).

- Mirčevska, V.; Kaluža, B. Towards intelligent home caregiver. In: Šetina, B.; Junkar, I.; Kaluža, B.; EleršIč, K. (eds.) *Proceedings of the* 1^{st} *Jožef Stefan International Postgraduate School Student's Conference.* 32–33 (Jožef Stefan International Postgraduate School, Ljubljana, Slovenia, 2010b).

Appendix B: Biography

Violeta Mirchevska was born in Skopje, Macedonia, on February 1, 1984. She received a university degree in 2007 from the Faculty of Electrical Engineering and Information Technologies, Ss. Cyril and Methodius University, Skopje, Macedonia, by defending the thesis "Searching through multimedial data using the MPEG7 color descriptors". During the undergraduate studies, she was awarded several times for outstanding achievements.

In 2008, Violeta enrolled in the "New Media and E-Science" doctoral-degree study program at the Jožef Stefan International Postgraduate School, Ljubljana, Slovenia. She was awarded a scholarship from the Department of Intelligent Systems at the Jožef Stefan Institute where she started her research work under the supervision of Prof. Dr. Matjaž Gams. In 2009, cooperation with the company Result d.o.o. was established, when the Slovenian Technology Agency approved funding for a joint research project under the public call "Young researchers from industry – Generation 2009". Since June 2013, Violeta is a research assistant at the Department of Intelligent Systems at the Jožef Stefan Institute.

Violeta's research focuses on behavior modeling that leverages both existing domain knowledge and machine learning. Three application domains are addressed in her work: (i) modeling users for the purpose of detecting unusual behavior – learning everyday behavior of an elderly user in order to detect deviations related to health problems, (ii) understanding and studying the behavior of agents in a multi-agent system – analyzing interactions of opposing groups of agents, and (iii) adaptation of software applications to user needs – adaptation of the reporting level of business intelligence applications to better suit the user information needs. Violeta's research achievements were published in scientific journals and conference proceedings.