Transfer Learning with Abstract Relational Features

Yelena Bayeva

Thesis advisor: Maarten van Someren

Master of Science in Artificial Intelligence
Universiteit van Amsterdam
August, 2011
Contents

1 Introduction ................................................. 1
  1.1 Research Questions and Scope ...................... 5
  1.2 Contribution ........................................ 6
  1.3 Overview ........................................... 6

2 Background and Related Work ................................. 7
  2.1 Inductive Learning .................................. 7
  2.2 Transfer Learning .................................. 7
  2.3 Relational Knowledge Transfer ...................... 9
  2.4 Domain Adaptation .................................. 11
  2.5 Relational Feature Extraction ....................... 12
  2.6 Methods for Relational Feature Extraction .......... 13
  2.7 Subgroup Discovery ................................ 14
    2.7.1 Subgroup Discovery Definition .................. 14
    2.7.2 Relational Subgroup Discovery Algorithms ....... 15
    2.7.3 Association Subgroup Discovery Algorithms .... 16

3 Approach .................................................. 18
  3.1 Problem Formalism .................................... 18
  3.2 Data Representation .................................. 20
    3.2.1 First-Order Logic Definitions .................... 20
    3.2.2 Domain Representation ............................ 20
  3.3 RFE Algorithm ....................................... 23
  3.4 Putting it Together .................................. 25

4 Experiments and Results .................................. 26
  4.1 Games Domain ....................................... 26
    4.1.1 Domain Description ............................... 26
    4.1.2 Experimental Setup ............................... 29
    4.1.3 Results ......................................... 30
  4.2 Human Behavior Recognition Domain ................. 33
    4.2.1 Domain Description ............................... 33
    4.2.2 Experimental Setup ............................... 36
    4.2.3 Results ......................................... 39

5 Discussion ............................................... 44

6 Conclusions and Future Work ............................... 47
Abstract

Traditional inductive learning methods assume that training and test data come from the same distribution, share feature spaces and class labels. The transfer learning framework removes all or some of these assumptions and allows to extract and reuse information from different domains to improve the accuracy on the target task (e.g. classification). Different degrees of transfer are possible, from shallow, where test instances come from the same domain but follow different distributions, to deep, where test instances are from different domains entirely. We propose an approach that allows for an intermediate level of transfer. It combines an abstract relational data representation and an efficient association subgroup discovery algorithm for extraction of abstract relational features and provides a way of incorporating domain knowledge. The method has been applied to games and human behavior recognition domains with target tasks of game state and human behavior recognition respectively. The results show that in general our method helps improve the classification accuracy in both types of domains.
Chapter 1

Introduction

The field of machine learning is concerned with the question of how to construct computer programs that automatically improve with experience [43]. Until recently the main focus of machine learning has been on *inductive learning* - a way of generalizing from training instances to test instances from the same distribution. A typical inductive learning task can be defined as class prediction of a previously unseen instance represented by a set of features. To accomplish this task, a learner is trained on a set of labeled (supervised), unlabeled (unsupervised), or partially labeled (semi-supervised) instances, such that the highest accuracy is achieved on a test set. In this setting, training and test data are typically drawn from the same domain, the same feature space, and the same distribution. However, approximating isolated functions from examples of input-output pairs is just a subroutine for learning and, according to Mitchell, it is more or less a solved problem at this point [44].

The next steps are now taken to uncover more general properties of domains and come up with cross domain solutions. The idea of extracting general information and reusing it whenever possible is at the core of a recently emerged learning framework called *transfer learning* or *knowledge transfer*. Transfer learning addresses the problem of how to leverage knowledge acquired in a source domain to improve the accuracy and speed of learning in a related target domain [42]. The transfer learning setting includes a source domain and its corresponding learning task and a target domain and its learning task. The knowledge is extracted from the source domain and transferred to the target domain in hopes that it improves the learning of the target predictive function or makes the learning task easier in some way. For example, if very little training data is available in the domain of interest, and the training data is plentiful in another related domain, knowledge transfer, if done successfully, can leverage information from the source domain to greatly improve the performance of learning in the target domain.

A typical transfer learning task consists of two steps. First, some specific knowledge (e.g. priors, learning parameters, features) is extracted from the source domain. Second, this knowledge is used to carry out a learning task in the target domain. For example, if our learning task in the target domain is classification, and the source and the target domains share abstract features, these features can be extracted from the source domain and used to enrich the target domain to help improve classification accuracy.

Transfer learning methods attempted thus far have been categorized by Pan and Yang into four groups, depending on the type of knowledge being transferred: *instance transfer,*
parameter transfer, feature representation transfer, and relational knowledge transfer [46]. The instance transfer approach re-weights labeled data in the source domain for use in the target domain. The parameter transfer approach assumes that source and target tasks share some parameters or prior distributions of the hyper-parameters of the models. This shared knowledge can be discovered and transferred across tasks. Feature representation transfer finds abstract features that reduce the difference between domains and the classification and regression errors of the models. The goal is to learn a new feature representation for the target domain, such that the knowledge used for transfer across domains is encoded into this representation and yields an improvement in the performance on the target task. Finally, relational knowledge transfer assumes that some relationships present in the source and target domains are similar, and can be transferred.

In addition, Davis and Domingos [19] distinguish shallow and deep types of transfer. Shallow transfer assumes that the instances are from the same domain - have shared features and class values, but have different distributions. Deep transfer, on the other hand, assumes that instances are from entirely different domains - have different features and class values. Much prior work has been done in the direction of shallow transfer [51], [16], [11], [56]. Deep transfer has not yet been explored to a great extent. Recently a state-of-the-art deep transfer algorithm DTM (Deep Transfer via Markov Logic) has been introduced in the context of relational domains and used to successfully transfer general knowledge across different domains, including molecular biology, social network and Web domains [19].

None of these works, however, attempt to transfer abstract relational knowledge - knowledge that explicitly captures relationships between the original features within each individual instance. These relationships become abstract when they are mapped to a feature space that is shared across domains. Thus, the assumption is made that the original features within each data instance are related in some way (are non-i.i.d), while instances within the domain are i.i.d. This is different from the relational knowledge transfer approaches [42],[41],[19] that assume the data to be in a relational, non-i.i.d form. For example, a relational data set would include one big graph of interdependent objects and their properties, e.g. a social network graph, while we assume that there are many small graphs, each representing an independent instance. Such relational knowledge can be captured by a combination of an abstract data representation as with feature representation transfer, and explicit relations as with relational knowledge transfer.

In this thesis we propose an approach that extracts and transfers abstract relational features from source to target domains with the aim to improve the accuracy of the classification task in the target domain. The prospect is that these new features are more general and capture more information than what is available in the target domain, and thus, can help boost the results of the target task. Abstract relational features are features that constitute each instance and represent relations between the original features mapped to a feature space shared between source and target domains.

To understand this concept, let’s consider a specific example with two related bioinformatics domains, where each domain corresponds to a data set of instances describing a particular kind of cancer, e.g. breast, prostate, lung, etc. We assume that each domain is represented by non-overlapping sets of genes with similar properties. The data in each domain contains positive instances with cancer (P) and negative instances (N) without cancer. This example is a modification and simplification of the real gene expression data sets described in [59]. The instances are represented by a set of differentially expressed genes $G_C(c)$ for every
class $c \in C$. Gene expression is the process of transcribing a gene’s DNA sequence into the RNA that serves as a template for protein production. Examples of positive instances represented as a set of gene ids for domains D1 and D2 can look something like this:

D1: P 6714, 1874
D2: P 67, 74, 155

In addition to gene ids, gene ontologies (GOs) contain background knowledge about genes, like description of cellular components, molecular functions, biological processes and gene interactions. This knowledge can be used to introduce additional useful information about the domain and can help better discriminate cancerous instances from non-cancerous ones. To introduce this knowledge, we represent the data in first-order logic form by adding gene predicates, and then we add new predicates describing gene properties and interactions using the information from GO. The examples of previously introduced positive instances will now look something like this:

D1: P Gene(6714), Gene(5194), Function(6714,‘zinc ion binding’), Component(5194,’integral to membrane’),..., Interaction(6714,5194)
D2: P Gene(67), Gene(74), Gene(155), Function(67,’iron ion binding’), Component(74,’intrinsic to membrane’),...,Interaction(74,155), Interaction(67)

As we can see, with this representation there is no explicit overlap between the two domains. However, if we replace gene ids with generic variables and abstract away feature values by replacing them with more abstract concepts that describe them, then it is possible to extract rules that are shared across domains. For example, both ‘zinc ion binding’ and ‘iron ion binding’ can be represented by more general ‘metal ion binding’, and both ‘integral to membrane’ and ‘intrinsic to membrane’ are associated with membrane and can be replaced by ‘membrane’.

The instances with new abstract values will then look like this:

D1: P Gene(g1), Gene(g2), Function(g1,’metal ion binding’), Component(g2,’membrane’),...,Interaction(g1,g2)
D2: P Gene(g1), Gene(g2), Gene(g3), Function(g1,’metal ion binding’), Component(g2,’membrane’),...,Interaction(g1,g2), Interaction(g1,g3)

The following rule is an example of a new abstract relational feature extracted from combined domains D1 and D2:

Function(g1, ‘metal ion binding’) AND Component(g2,’membrane’) AND Interaction(g1,g2) => P

The rule captures relationships between the original features, it is general and informative of a class, and it is shared across domains. Thus, it can be transferred to a new related domain D3.

In the example above, instances originally have a vector representation, that is they consist of a set of feature values. Individual features within each instance are related with respect to some underlying structure inherent to the domain, and, based on that structure, they can form groups informative of a class. These are the new relational features, represented here by first-order logic rules. Furthermore, if the values of the original features are different across
domains, but on some abstract level they can be described by the same general concept, it can be possible to construct relational features from abstractions of the original features. Such abstractions or models can be either learned from the data or introduced as a background knowledge. This is what we mean by mapping of the original feature values to the shared feature space.

In order to introduce abstract relational features several problems have to be solved. First, we have to find a data representation that captures relations between the original features, for example, graphs, logical rules, formulae, an so on. Second, given relations constructed using a representation of choice, useful relations have to be identified. We define usefulness in terms of a combination of feature informativeness of class, generality, and transferability.

To solve these problems we use a combination of an abstract relational data representation, as described in the example above, and an efficient pattern extraction algorithm. First, the data is represented in terms of first-order logic predicates with values instantiated using a feature space shared across all domains. The first-order predicates can be introduced with the help of the background knowledge or derived from the data automatically, e.g. by turning feature names into predicates and feature values into predicate values. Second, the original feature/predicate values are replaced with abstract feature values, which are also constructed either automatically, or with the help of the background knowledge, or a combination of the two. Third, in order to extract useful features informative of a class we use an association subgroup discovery algorithm. It extracts statistically significant and interesting rules, which in the context of our data representation are conjunctions of first-order predicates. The new features are then used to help improve the performance of the classifier in the target domain.

This approach allows for an intermediate level of transfer that lies between shallow and deep types of transfer. It fits into a bigger vision of transfer learning, where the right level of transfer is automatically determined for source and target domains based on the amount of overlap. Deep transfer is applied when there is very little overlap, intermediate, when the abstract feature values and the underlying structure of the domains are shared, and shallow, when only distributions differ between domains.

We apply our transfer learning method to two types of domains - games and human behavior recognition. In the first case, the target learning task is to recognize board configurations containing a type of a trap called a double threat. Instances represent different board configurations accompanied with positive and negative class labels. Several kinds of games are used, each representing a new domain - tic tac toe, chess, and 9 men's morris. In the second case, the task is to recognize human activities based on sensor readings. The activities are recorded by sensors located in the different parts of the house. Here, each house represents a new domain, instances include sensor information, and class labels are different activities. Games domains are relatively noise-free and have binary classification as a target task. Human behavior recognition domains has a lot of noise associated with sensor readings. The target task here is a multi-class classification.

The results indicate that our abstract relational feature transfer approach, when applied to the games domain, significantly improves classification accuracy in the target domain in the presence of very little target training data compared to the no transfer case. In the human behavior recognition domain the improvement is less dramatic, but often statistically significant. The differences in the impact of transferred features between these two types of domains can be explained by the differences in their properties: little vs. a lot of noise, binary vs. multi-class classification.
classification, dense vs. sparse feature spaces.

1.1 Research Questions and Scope

The research goal of this thesis is to determine whether or not it is possible to extract and effectively transfer abstract relational features - features that capture relational structure shared between domains at an abstract level. This means that the dependencies between the original domain features might not be explicitly expressed in the data and need to be discovered and/or introduced with the help of the background knowledge. To reach our goal of extracting abstract relational features, we address the following research questions:

*How can we represent the data to achieve maximum overlap between domains?*

There are many ways to represent and extract the underlying structure of the domains. For example, via latent variables [62], feature spaces [47], parameters [36], [9], [58], priors [51], [28], objective functions [39], similarity functions or kernels [17], manifold regularizers [24], or in the form of explicit features [8], [52],[4], [19]. In this work, we investigate the transfer of features in the form of explicit structural patterns or rules. Among the proposed approaches targeted at solving this problem are the transfer of a linear combination of basis vectors [52], models of pivot features [8], word clusters shared between domains [15], logic rules (first or second-order) [42],[41],[19], and others.

Because we would like to extract explicit features representing relations between the original features, we adopt a relational data representation approach. With this approach, domain features become objects characterized by properties and relations to other objects, all represented by logic predicates. The property values are further abstracted away, e.g. via statistical models constructed either fully automatically or with the help of the background knowledge. The abstract representation increases the overlap between domains. This can help with extraction of more general features that can be transferred to related domains.

*What methods can be used to extract abstract relational features efficiently?*

Many pattern mining algorithms have been developed to date. For example, association-based algorithms extract frequently co-occurring sets of items [25], [1], graph-based algorithms find subgraph structures [63], [32], logic-based algorithms use ILP (Inductive Logic Programming) systems [12], and standard classification rule learning approaches adopted for subgroup discovery [33], [35], [12] find statistically significant and interesting patterns. Since our goal is to extract features that can be used for a classification target task, we focus our attention on subgroup discovery [26] methods that extract statistically significant features that are also informative of a class.

Because we would like to extract patterns for transfer from as many domains and as much data as possible, the algorithm has to be efficient and scale well. For this reason, we propose an *association subgroup discovery* algorithm called RFE (Relational Feature Extraction). It combines an efficient frequent pattern mining algorithm called FP-growth [25] and a *weighted relative accuracy* subgroup quality function [33], [60], which measures interestingness or unusualness of patterns.
How can we transfer new features to achieve improvement in the target learning task?

Typically, the extracted knowledge is transferred as a bias in a form of parameters [36], [9], [58], regularizers [24], priors [51] and meta priors [28], Markov logic networks [42],[41],[19], and so on. In all these cases this transferred bias is used to guide a search in the target domain. If the underlying structure of the domains is shared, we can attempt to transfer relations that capture this structure and transfer them explicitly as features. In this work, we assume that the underlying structure of the domains (in a form of first-order predicates) is shared and values are abstracted away through an intermediate representation such that they become shared as well. Under these assumptions, transferring explicit features is possible. To further improve the benefits of transfer, or to lift some of the assumptions stated above, it is possible to extend the proposed method, such that the abstract relational features are used as a bias in the target domain.

1.2 Contribution

The contribution this research makes is three fold. First, we propose a new category of transfer learning - abstract relational transfer learning that allows for an intermediate level of transfer - less general compared to deep transfer, and less specific compared to shallow transfer. With our approach we extract and transfer features capturing relationships between the original features in the domains. These features are extracted from the training data of source and target domains mapped to a common feature space. Unlike deep transfer that transfers second-order formulas, we transfer first-order predicates instantiated to abstract values. Unlike shallow transfer, we don’t assume that the original feature spaces of the domains are shared, only the abstract ones are. Second, we propose a method that combines an abstract relational data representation and an efficient association subgroup discovery algorithm for extraction of new relational features. In addition, the method provides a way of incorporating domain knowledge into these features. Third, we show how the method can be successfully applied to enhance game state detection and human behavior recognition learning tasks.

1.3 Overview

This thesis is organized as follows. In chapter 2 we give background information about inductive and transfer learning, as well as related research work in the areas of relational transfer learning, domain adaptation, relational feature extraction, and subgroup discovery. In chapter 3 we describe our feature based approach to transfer learning. In chapter 4 we present evaluation and results of the experiments we conducted on two domains to verify the validity of our approach. In chapter 5 we further discuss the approach and results. Finally, in chapter 6 we draw conclusions as put forth some ideas for future work.
Chapter 2

Background and Related Work

Our goal is to find a method for extraction and effective transfer of features that capture the structure shared between domains at an abstract level. To achieve this, we propose an approach that combines an abstract relational data representation with an efficient association subgroup discovery algorithm. Thus, we combine elements from different lines of work - from relational knowledge transfer to subgroup discovery algorithms.

In this section we accomplish three goals: first, we give background information related to our research in sections 2.1, 2.2, 2.7, second, we summarize methods that are similar to or useful to our approach in sections 2.3, 2.4, and third, we describe work that contributes to our proposed method in sections 2.5, 2.6.

2.1 Inductive Learning

Inductive learning can be defined as learning by example, where a system attempts to discover a general rule from a set of observed instances. Two types of inductive learning methods can be distinguished: the ones that operate on examples represented by attribute-value pairs - traditional classification algorithms like Decision Trees [50], Support Vector Machines [10], Naive Bayes Classifier [37], and so on, and the ones that handle relational information, like FOIL (First Order Inductive Learner) [49] and ILP systems.

Traditional formulation of inductive learning assumes that training data and test data come from the same distribution, share feature spaces and class labels. In real applications however, training and test data often come from different distributions. Moreover, to solve learning problems in domains with limited data, it is often desirable to use information from other related domains. What is needed then is the ability to discover knowledge that is shared between many different domains and use it to help improve performance of the learning task. This gap is filled by the transfer learning framework.

2.2 Transfer Learning

In many real world applications labeled data are scarce, especially for learning tasks in new domains. Obtaining labeled data in a new domain is often expensive and time consuming, while many other domains can have plenty of labeled data. Traditional machine learning methods
can not do learning across different domains, and thus, can not take advantage of the data from related domains to help with a learning task in the new domain.

Transfer learning addresses the problem of how to leverage knowledge acquired in a source domain to improve the accuracy and speed of learning in a related target domain [42]. Many recent works have defined and developed transfer learning methods that outperform traditional learners [13], [16], [51], [52], [42], [19], [46]. In all these works the underlying structure of one or more domains is learned and transferred to another related domain.

The amount of knowledge shared between domains can vary. Many domains have little overlap in their superficial descriptions, but still share the underlying structural patterns. In such cases, a deep transfer approach [19] is needed to capture structural regularities of domains, despite the differences in feature and class spaces. In other cases, the domains share feature and class spaces, but come from different distributions. Generalizing to different distributions over the same features is addressed by shallow transfer, sometimes it is referred to as domain adaptation [17], [16], [8], [48], etc. Thus, shallow transfer, that assumes a lot of overlap between domains, and deep transfer, that assumes very little overlap, are the two extremes of transfer learning. Furthermore, various degrees of partial overlap between domains are possible. Accordingly, different methods have been proposed to deal with these variations.

For example, a cross-domain learning approach has been proposed by Daumé III & Marcu [16] to deal with data from different domains that share the same categories or classes and have a partial overlap in features (e.g. words in a text domain). This allows for a representation of the domains in terms of a mixture of domain specific (includes features particular to a domain) and domain-general (includes features shared between domains) distributions. The domain-general distribution is then learned for transfer.

Other examples of solutions to partial overlap are self-taught learning [52] and cross-category learning [51] that address cases where abstract features are shared and class spaces can be different. In the first case, the unlabeled data from source domains is used to model the input space and extract abstract features (e.g. basis vectors) for transfer. In the second case, labeled source data is drawn from different classes than the target data and a Gaussian prior with a covariance matrix capturing dependencies between parameters are transferred.

Of particular interest are methods that make their goal an extraction of abstract features or feature correspondences specific to a certain type of a domain. Such transfer allows to capture the right amount of information - the underlying structure and the specifics of the domain represented by the abstract features. Abstract features can be expressed in different ways, e.g. as basis vectors [52] in domains where instances are represented as their linear combinations, or as logical formulas in relational domains [19]. Finding correspondences between features in different domains can be achieved through modeling their correlations with pivot features [8], mapping them to shared low dimensional feature spaces [47], finding mappings between domain predicates in relational data [42], and so on.

A bigger goal within the transfer learning framework can be automatic detection of the appropriate level of overlap between domains, so that information can be transferred at the right level of abstraction. Such a method would address all three main transfer learning research issues summarized in [46]: 1) what to transfer - which part of knowledge can be transferred across domains or tasks, 2) how to transfer - which method should be used for transfer, and 3)
when to transfer - in which situations transfer should be done. Furthermore, it will be a step forward in the direction of continuous learning or life-long learning which has a goal of creating systems that learn online and are able to self-reflect and make corrections to the learning process, much like human learning.

One way to achieve this is by extracting features at different levels of abstraction - from most general, like with deep transfer methods, to most specific, like with shallow transfer methods - and choosing the features that yield the best improvement in the performance on the target task. If data instances are represented in a logic form, with predicates as features, features can be extracted from one or more domains at different levels of abstraction - from non-grounded second-order logic formulas to grounded first-order logic rules. A non-grounded second-order logic predicate has both its name and its value represented by variables not instantiated or grounded in the data, e.g. $F(t)$, where $F$ and $t$ are both variables. A non-grounded first-order logic predicate has only the predicate name instantiated, while its grounded counterpart has both variables instantiated. For example $F(t)$ can become $Actor(t)$ in case of a non-grounded first-order predicate, or $Actor(Bob)$ in case of a grounded first-order predicate in the movies domain. If only the underlying (second-order) structure is shared, then second-order features can be transferred. Otherwise, if predicates are shared but the variable values are different, then first-order features can be tried. Finally, if values are shared, then grounded first-order predicates can be used for transfer. Furthermore, if values don’t overlap between domains, everything else being equal, one can attempt to abstract away the values such that they become shared. For example, Bob and Alice are different values, but they correspond to the same abstract concept of a person. This accomplishes an intermediate level of transfer.

Our goal is to develop an intermediate level transfer learning approach that can capture both structural regularities and abstract concepts shared between domains and transfer them in the form of abstract features to the target domain, where they can be used “as is” or as a bias to achieve improvement in the classification learning task. To do this, we represent the relationships in a logic form (e.g. in terms of first-order logic predicates), abstract away feature values whenever possible either automatically or with the help of the background knowledge, extract frequently occurring and interesting patterns or rules from the data available for training in source and target domains, and use them to enhance the classification learning task in the target domain.

2.3 Relational Knowledge Transfer

The closest related work to the extraction of abstract logic-based features is in the area of relational knowledge transfer [42], [41], [19]. Relational knowledge transfer methods deal with transfer learning problems in relational domains, where the data drawn from each domain are non-i.i.d. as traditionally assumed, but are represented by multiple relations, such as networked data and social network data [46]. In this context, the goal is to transfer relations among data from source to target domains. Statistical relational learning techniques have been proposed to solve this problem. Most successful approaches use Markov Logic Networks [54] - a powerful formalism, that combines the compact expressiveness of first-order logic with flexibility of probability, for statistical relational learning [46].

An MLN consists of a set of first or second-order clauses, each of which has an attached weight. The weight of a clause determines the likelihood of the situation in which the clause is
satisfied over the one in which it is not satisfied [41]. A more general case of transfer, where data are represented in terms of second-order logic predicates, is handled by DTM (Deep Transfer via Markov logic) algorithm [19]. Less general first-order transfer is achieved by TAMAR and SR2LR systems [42],[41].

DTM (Deep Transfer via Markov logic) algorithm [19] offers a flexible way to extract very general information across domains that have different types of objects and variables by combining second-order logic and Markov Logic Network (MLN) [54]. It discovers structural regularities in the source domain in the form of Markov logic formulas with predicate variables, and instantiates these formulas with predicates from the target domain [19]. It captures structural patterns shared between seemingly unrelated domains and uses them as bias for search in a new domain. The discovered formulas include broadly useful properties of predicates, like symmetry and transitivity, and relations among predicates, such as various forms of homophily. However, if very little training data is available in the target domain, then relying on such search might not yield high quality features, and thus might not bear much improvement in the target learning task. In this case, if more data is available in other related domains, we would like to extract and re-use as much information as possible from the source domains.

Mihalkova et al. use first-order MLN to solve the problem of knowledge transfer across domains that use different representations - contain different sets of relations. For example, knowledge about an academic domain is similar to that about movie industry. It is assumed that the underlying second-order structure is similar, but the names of the predicates differ - students are similar to actors, advisers are similar to movie directors, and the relationships between students and advisers are similar to those of actors and movie directors. The proposed algorithm TAMAR goes about solving this problem by finding a mapping between the relations in the source and target domains. First, it constructs a mapping from source domain to the target domain based on the weighted pseudo loglikelihood measure (WPLL) [42]. Then, an ILP algorithm FORTE [53] is used to to revise the first-order theories. The revised MLN is then used as a relational model for inference in the target domain. In addition, they propose the SR2LR algorithm. It improves upon TAMAR by allowing to map source knowledge when minimal target domain data is available in a single-entity-centered setting - a setting where the learner is provided with information concerning only a single entity [41].

TAMAR and SR2LR methods provide a mapping between different predicate names that serve the same purpose in different domains. However, these methods do not take into account possible differences in variable values. The values of predicate variables can themselves be specific to different domains, but represent similar concepts. In this case, it might be possible to come up with an abstraction, a mapping from more specific to more general or abstract values shared across domains.

All of the above works act on relational, non-i.i.d data from which a relational model is constructed and used for inference in the target domain. Instead, we assume that the data instances are i.i.d., but the features are related and take on a logic form. In this case, each instance is represented by a set of predicates. This representation can be used with different data sets that can assume relational form, including most vector form data. Relational features or rules can then be extracted from the data and used as new features to re-represent it. In this way, the data takes on a vector form with a fixed number of features and can be used with any learner. For our purposes, we adopt first-order logic representation of domain instances.
In addition, all discussed MLN based approaches make no reference to the class labels - the underlying structure is transferred regardless of the class. Our goal, however, is to improve the accuracy of the classification task in the target domain. Thus, we aim is to capture structural patterns that are informative of a class. For this reason, we use a subgroup discovery approach to extract patterns that are both statistically significant and informative of a class. Because of the relational representation of the data, these patterns are in a form of logic rules, e.g. conjunctions of predicates. The features are learned from labeled data available in source and target domains and are used to carry out a learning task in the target domain.

2.4 Domain Adaptation

The problem of domain adaptation or shallow transfer arises when the feature spaces are shared between domains, but the data comes from different distributions. In such cases, steps must be taken to adapt a model trained on the source domain for use in the target domain [48]. The goal then becomes to minimize the discrepancies between domain distributions.

Many solutions have been proposed for NLP domains, for example, representation of data in terms of a mixture of general, in-domain and out-domain distributions [16], heuristic kernel that augments features with general, source-specific, and target-specific versions [17], and others. Many more recent methods focus on learning a common feature representation that is meaningful across domains [8],[47], [48]. The prospect is that a common representation can help reduce the difference in distributions between domains and at the same time preserve important properties (e.g. geometric, statistical, etc) of the domains, especially of the target domain.

Blitzer et.al.[8] have developed a structural correspondence learning (SCL) algorithm, to induce correspondences among features from different domains. They do this by modeling correlations of regular features with pivot features - features which behave in the same way for learning tasks in all domains. Pan et al. [47] proposed a dimensionality reduction method called Maximum Mean Discrepancy Embedding (MMDE) for domain adaptation that learns shared latent space underlying the domains such that the distance between distributions is reduced while the data variance is preserved. As a follow up to this work, a more efficient transfer component analysis (TSA) [48] algorithm has been developed. This algorithm tries to learn a set of common transfer components underlying both domains such that the difference in data distributions of the different domains, when projected onto this subspace, can be dramatically reduced and data properties can be preserved. The idea is that if domains are related in some way, their underlying structure can be captured by some common components or latent variables. Some of these components are relevant, while others are not. The goal is then to discover components that are shared between domains and preserve the structure or task-relevant information of the original data.

We take into account two main points set forth by this line of research. First, if the features serve the same function in different domains, they can be abstracted away to a common feature space, such that they become shared. Second, even when feature spaces are shared across domains the discrepancies in distributions of different domains might need to be adjusted to achieve improvement in performance of the target learning task. In other words, when new features are extracted from source domains and used for transfer, we have to make sure that irrelevant information introduced by these features doesn’t overshadow information relevant to the target task, e.g. information specific to the target domain.
2.5 Relational Feature Extraction

Features representing instances are often related in some way. These relationships can be implicit, e.g. determined by latent variables, or explicit, e.g. constructed with the help of the background knowledge. Relational induction methods like ILP systems expect the data to assume strictly relational form, where instances are non-i.i.d. This allows for induction based on facts and relations explicitly present in the data. We investigate ways in which features can be extracted from data with i.i.d. instances and non-i.i.d. features. We focus on extraction of features for a classification learning task.

Construction of relational features - features that capture relationships between the original features of the domain - can offer many advantages. First, they can convey additional information not only about each individual instance, but also about the domain in general. Thus, they can capture knowledge that is not otherwise available to a simple classifier and can improve its accuracy.

Second, in addition to improving the accuracy of a learner, relational features can represent the data more compactly (in terms of general patterns or rules), and allow for a more efficient classification with any learner. This is especially true if all or many of the original features can be replaced with a few more general relational ones, leading to a significant reduction in the number of features.

Third, some data sets have examples containing a variable numbers of features. In this case, the application of standard classifiers is difficult because most of them expect data representation in terms of a fixed number of features. Representing the data in terms of a fixed number of relational features can solve this problem.

Third, construction of relational features can help improve classification for domains where the features are very sparse. For example, in text domain many words occur infrequently, yet they can be similar (e.g. synonymous) to other more frequently occurring words. Similarly, in human behavior recognition domains, activities and their values vary greatly depending on time, person, location, but can maintain a similar trend. Relational features combined with abstract value representations can help capture general structural patterns despite variations in values.

Finally, because such features can capture general knowledge, they can be transferred between domains. In this case, a transfer learning task can be defined where relational features are extracted from one or more domains and used to aid the learning task of the target domain. The hope is that performance of a learner in the target domain is improved due to these features. In some cases, it is possible that relational features can completely replace the original features while still maintaining superior performance of a learner.

In order to introduce relational features, several problems have to be solved. First, for any given task and domain we have to find a data representation that captures relations between the original features, for example, graphs, logic rules, formulae, an so on. Second, given relations constructed using a representation of choice, useful relations have to be identified. Usefulness can be determined by feature informativeness of class, generality, and transferability. First, features have to be informative of class, otherwise their inclusion will provide little improvement in performance of a classifier. Second, the more general the relations are, the more examples belonging to a given class they can cover. Thus, more general patterns have to be preferred to
less general ones. Finally, if transfer is desired, then the features extracted from source domains have to be applicable to the target domain.

The first requirement - extracting features informative of class - can be satisfied with the help of a variety of methods that we summarize in section 2.6. We find subgroup discovery algorithms, discussed in section 2.7, to be the most appropriate for the task. Generality of features can be achieved through a combination of an appropriate feature extraction algorithm (section 3.3) and abstract data representation (section 3.2). To obtain features that can be transferred across domains, training data from source domains and target domains can be combined for feature extraction.

### 2.6 Methods for Relational Feature Extraction

To date many techniques have been developed in areas of structural knowledge discovery, information extraction, relation extraction, and relational rule learning, that can be adopted to relational feature extraction.

Structural knowledge discovery includes methods like Subdue [27] - a graph-based pattern learning system that finds relational patterns in the data represented as a graph. This system can mine a graph and extract frequent subgraphs that, when removed, reduce the size of the original graph. The goal is to compress the graph as much as possible. Subdue is used for frequent subgraph discovery [32], and provides a capability to extract frequent subgraphs that are particular to certain target variable values and not to others. In the context of our feature extraction task, each data instance can be represented as a graph, where nodes are features, and edges are relationships between them. The graphs can be mined by Subdue, such that frequently occurring subgraphs representing structural regularities in the domain can be extracted and used as relational features.

While graph-based methods like Subdue enable basic structural analysis of the data and capture the underlying structural relationships, e.g. frequently occurring subgraphs represented by nodes and edges, such an analysis might be insufficient for capturing additional, non-structural information like different properties of features at different levels of abstraction and background knowledge.

The formalism of relational logic can fill this gap and help capture relevant background knowledge and convey it with the help of logic formulas or rules. ILP targets the appropriate selection and formulation of background knowledge to be used by the selected ILP learner. Here, a lot of the effort is put into the construction of features - the data representation is transformed via constructive induction such that new attributes can be added using the background knowledge and irrelevant attributes can be removed. Constructive induction is used in propositional learning, and predicate invention is its first-order alternative. Features constructed this way are then used for relational rule learning. Learning sets of relational rules that capture properties of class values is the goal of classification rule learning algorithms.

Some relational logic algorithms like RSD [33] and CN2 [35] adopt an intermediate approach and use predicate invention to construct first-order features that are then used for induction of rules. These algorithms adapt classification rule learner algorithms to subgroup discovery, such that rule interestingness factor - unusual statistical distribution with respect
to the class label - is considered. Unfortunately, algorithms based on relational logic remain inefficient and difficult to apply to large multi-dimensional data sets.

Subgroup discovery itself falls under a recently proposed paradigm called *Supervised Descriptive Rule Induction* (SDRI) [45], the main objective of which is to extract descriptive knowledge from the data about a property of interest [26]. *Contrast set mining* [6] and *Emerging Pattern Mining* [21] also fall under SDRI. These algorithms, however, only look for relationships in the data with respect to the possible values of the target variable (based on measures of coverage and accuracy). Subgroup discovery, in addition to this, is also concerned with interestingness of discovered patterns with respect to class values based on novelty and unusualness measures. For this reason, we only consider the latter as it is better suited to our goals.

As an alternative to logic-based algorithms, several extensions to association algorithms have been proposed to solve the subgroup discovery problem. These algorithms are summarized in [26]: APRIORI algorithm [1] extensions - APRIORI-SD, SD4TS, FP-Growth algorithm [25] extensions - SD-MAP, DpSubgroup, and Merge-SD. These algorithms have the advantage of being more efficient compared to their rule learning counterparts. They include additional quality measures (e.g. unusualness, prediction quality, etc) to select rules that are not only frequent, but also descriptive of subgroups present within each class.

Much work has been done in areas of *information extraction* and *relation extraction*. Some of the prior work on relation extraction has been surveyed in [29], from hand-written patterns, learning generalized extraction patterns expressed as characters, words or syntactic categories of words, clustering approaches based on co-occurrence [18] and a variety of kernel-based methods [20], [11], [3] including local alignment (LA) kernel proposed in [29]. Aside from kernel methods, the above techniques tend to be specific to text domain and don’t generalize well beyond it. The basic goal fulfilled by *kernel methods* is similar to that of the *relation feature extraction* - identifying examples most similar to examples belonging to a certain class. This task is done implicitly by the kernel function that uses some similarity measure to compare new examples to all existing training data. On the other hand, with relation features, similarity information is embedded in the abstract relational features that can be used with any basic classifier. The advantage of doing this is efficiency - we no longer need to compare new examples to all existing training data.

### 2.7 Subgroup Discovery

Subgroup discovery satisfies the requirements of relational feature extraction as it discovers patterns (or subgroups) that are interesting and informative of a class and “combines statistical significance with an understandable representation of patterns as a logical formula” [57]. In this section, we go over subgroup discovery algorithms and identify the most suitable algorithm for the task of relation feature extraction.

#### 2.7.1 Subgroup Discovery Definition

Subgroup discovery attempts to search for and extract relations between different variables of a set with respect to a target variable [26]. It targets the extractions of top k patterns or rules with interesting characteristics, e.g. unusual statistical distribution - the one that deviates significantly with respect to the target variable.
The extracted patterns representing relations (also called subgroups) are described in the form of individual rules (R) that can be defined as in [23],[34]: \( R : \text{Cond} \rightarrow \text{Target value} \), where Cond is usually a conjunction of features (attribute-value pairs) representing a subgroup description with respect to the Target value - a target variable or a class [26]. It is therefore a supervised approach.

Subgroup discovery fits somewhere halfway between predictive and descriptive induction. The former includes classical rule learning algorithms designed to construct prediction and classification rules [14], [40]. The latter is concerned with extraction of interesting knowledge from the data and includes techniques like association rules mining [2], summarization [64], or subgroup discovery [31], [61].

Thus, subgroup discovery can help extract features informative of a class. It does it by finding rules representing interesting (according to some measure) subgroups belonging to different values of a target variable. Next, we look at two variants of subgroup discovery algorithms: relational and association.

### 2.7.2 Relational Subgroup Discovery Algorithms

RSD [60], [33] is a state of the art relational subgroup discovery algorithm that appropriately adapts rule learning and first-order feature construction. It is applied to individual centered domains - domains represented by individuals, e.g. molecules in molecular biology.

Given a set of instances in relational form and a mode-language that specifies user-defined constraints (e.g. rule length, number of predicates of a certain type in each instance, and so on), RSD induces rules or relational features in the form of first-order logic atom conjunctions, e.g. worker(X), manager(Y), manages(Y,X), where upper cases denote variables, and a comma between two logical literals denotes a conjunction [60].

The algorithm that constructs first-order features or rules can be split into three relatively independent problems [33]. First problem is identifying all first-order literal conjunctions that form a feature and at the same time comply to user-defined constraints. Second problem is using the data to instantiate feature variables. Third problem is identifying and eliminating irrelevant features and representing data using the generated feature set.

To discover first-order feature representation of subgroups, RSD employs the following main ingredients: exhaustive first-order feature construction, elimination of irrelevant features, implementation of a relational rule learner, use of the weighted covering algorithm, and incorporation of example weights into the weighted relative accuracy heuristic [33]. First-order features are composed of one or more structural predicates that introduce new variables, representing parts of individuals, and of utility predicates that consist of structural predicates and don’t introduce new variables.

Rule learning with RSD typically consists of two main procedures: the search procedure and the control procedure that repeatedly executes the search. The search procedure performs beam search using weighted relative accuracy as a subgroup quality function, which measures unusualness and can be meaningfully applied both in the descriptive and predictive induction frameworks. This function was further modified to enable handling of example weights such that different instances can be considered at each iteration of the covering algorithm - at each
iteration the weight of already covered examples is decreased (set to $\frac{1}{\text{iteration}+1}$).

$$WRAcc(H \rightarrow B) = \frac{n'(B)}{N'} \left( \frac{n'(H.B)}{n'(B)} - \frac{n'(H)}{N'} \right).$$  \hspace{1cm} (2.1)

Here, $H \rightarrow B$ represents a rule - with head $H$ and body $B$, $N'$ is the sum of weights of all covered examples, and $n'(H.B)$ is the sum of weights of all correctly covered examples [33], [60]. A simplified version of this measure shows that it trades of generality of a rule $p(B)$ and its relative accuracy ($p(H|B) - p(H)$), which yields high quality rules:

$$WRAcc(H \rightarrow B) = p(B) \ast (p(H|B) - p(H)).$$

Propositionalized representation of rules induced by RSD is ideal for incorporating domain knowledge into the classifier as well as constructing general rules. This makes RSD a good candidate for relational feature extraction.

However, some limitations of RSD prevent its direct application to this task. First, RSD inherits scalability issues of the rule learning algorithms which makes it difficult to apply to large, high-dimensional data sets. Second, recursive clauses are not supported, which makes it inapplicable to domains that have recursive structure like sequences or trees.

### 2.7.3 Association Subgroup Discovery Algorithms

Association subgroup discovery algorithms offer better scalability compared to their relational counterparts. Many of the frequent itemset mining (FIM) algorithms, like APRIORI [1] and FP-Growth [25] have been extended to fit the subgroup discovery task. Similar to the traditional association mining algorithms, these algorithms, given a set of instances represented by unordered sets of features, efficiently extract frequently co-occurring sets of features represented as rules, based on minimum confidence and support scores set by the user. The support is the proportion of instances in the data set that contain a rule. Confidence is the probability of a rule. Similar to subgroup discovery algorithms, they include quality functions that keep rules characteristic of subgroups present in the data and filter out the rest.

FP-Growth is more efficient compared to APRIORI as it avoids multiple scans of the database for testing each frequent pattern. Instead, it applies a recursive divide-and-conquer technique [5]. Further improvements to FP-Growth have been achieved through parallelization of the algorithm [38] and other heuristics implemented in Apache Mahout scalable machine learning library\(^1\).

SD-Map algorithm described in [5] extends FP-Growth to do subgroup discovery. SD-Map exploits FP-Growth’s frequent pattern tree data structure (FP-tree) that stores count information about the frequent patterns. It uses it to store additional counts needed for computation of the pattern quality function (e.g. Piatetsky-Shapiro, unusualness, the binomial test, etc). It also has the capability of handling missing values.

An equivalent subgroup discovery extension of APRIORI algorithm is APRIORI-SD [30]. It makes the APRIORI-C algorithm appropriate for subgroup discovery by implementing an example weighting scheme in rule post-processing and applying weighted relative accuracy heuristic to extract the best rules. The algorithm iterates over the ruleset induced by APRIORI-C, removing the best rules from it until none are left or all examples are covered. The best rules

\(^1\)https://cwiki.apache.org/MAHOUT/parallel-frequent-pattern-mining.html
are selected according to the weighted relative accuracy measure (WRAcc). For each of the best rules, examples covered by it are re-weighted according to the formula \( w(e_i, i) = \frac{1}{i+1} \) (\( e_i \) is an \( i^{th} \) example) and removed if they are covered more than \( k \)-times.

Despite the scalability of associative algorithms, the induced rules often disregard more complex (other than association) relationships present in the data and don’t generalize beyond a single domain. For this reason, out-of-the-box application of such algorithms might not be sufficient for abstract relational feature extraction.

To construct an algorithm suitable for relational feature extraction we combine useful features of the relational and association subgroup discovery algorithms. The first one offers generality of induced rules achieved through data representation, while the second offers a scalable solution to feature extraction. For this reason, we adopt a propositional data representation similar to the one used with RSD, and use an efficient association subgroup discovery algorithm that extracts statistically significant and interesting rules. We create an algorithm that is a hybrid of APRIORI-SD and FP-Growth algorithms and further extend it to filter out less accurate and less general variants of the extracted rules. The resulting algorithm inherits the efficiency of FP-Growth and the weighted relative accuracy measure of APRIORI-SD, and with the help of an extra filtering step it eliminates redundant information. Thus, our algorithm is more efficient and more selective than the other two.
Chapter 3

Approach

We propose a method for transfer learning that enables transfer of abstract relational features between related domains. We assume that the underlying structure, represented in terms of logic predicates, is shared across domains. It is possible that the actual names of the predicates are not shared and need to be mapped between domains, but this problem has already been solved in [42], [41].

Based on the observation that at an abstract level there is more overlap between domains, we want to model feature values across domains to find a shared representation. Thus, we represent the data in terms of first-order predicates instantiated with abstract feature values. In this way, features become shared between domains. We then apply a fast pattern mining algorithm called RFE (Relational Feature Extraction) to obtain rules - conjunctions of first-order predicates that serve as new features for a classification task in the target domain.

Depending on the amount of overlap between domains, the new features may or may not subsume the original features of the target domain. The former is typically the case in noise free domains with fewer discrepancies in data distributions. In this case, features constructed with the help of source domains can be transferred “as is”. The latter case can happen when there is less overlap between domains, the domains are noisy, and/or the data distributions are quite different. In this case, it is necessary to adopt features to the target domain or use them to augment the original feature space.

In the following sections we will describe the details of the approach - problem formalism 3.1, data representation 3.2, a feature extraction algorithm 3.3, and how it all fits together 3.4.

3.1 Problem Formalism

A supervised learning task $T$ can be defined as the prediction of a class label $y$ given a previously unseen instance $x$ in a vector form - represented by a set of $l$ features $\{f_1, f_2, \ldots, f_l\}$ from a feature space $F$. It can be represented in terms of two components, a label space $Y$, and a predictive function $f(.)$ learned on a set of $m$ labeled instances $(x_i, y_i)$, $i = 1 \ldots, m$, such that the highest accuracy is achieved on a test set $(x_j, y_j)$, $j = m + 1, \ldots, n$. Here, all instances come from a set $X = \{x_1, \ldots, x_n\}$, where $x_i$ is a feature vector with a corresponding label $y_i \in Y$. A learning task carried out in a particular domain $D$ is represented by two components: a feature space $F$ and a marginal probability distribution $P(X)$ and can be written as $D = \{F, P(X)\}$. 
A transfer learning task includes a source domain $\mathcal{D}_S$ and a learning task $\mathcal{T}_S$, a target domain $\mathcal{D}_T$ and its learning task $\mathcal{T}_T$ and aims to improve the learning of the target predictive function $f_T(.)$ in $\mathcal{D}_T$ using the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$, where $\mathcal{D}_S \neq \mathcal{D}_T$, or $\mathcal{T}_S \neq \mathcal{T}_T$.

In this thesis, we focus on the problem of extracting and transferring abstract relational features from the training data available in source and target domains, such that the data can be represented in terms of features coming from a new feature space $\mathcal{R}$ that is shared across all domains. We assume that labels in all domains come from the same label set $\mathcal{Y}$, but feature spaces $\mathcal{F}$ and distributions $P(X)$ are different. However, we do assume that source and target domains are related - share the underlying relational structure. Intuitively, if the structure is shared between domains, the new abstract features can help extract general knowledge about all domains and thus improve classification accuracy in the target domain where little training data is available.

We distinguish between the original features, coming from the feature space $\mathcal{F}$, and relational features coming from a feature space $\mathcal{R}$ represented by first-order rules - conjunctions of first-order predicates $\mathcal{P}$ informative of class. The rules are extracted by representing the data in a first-order logic form, and then applying an efficient rule extraction algorithm to this representation. Two types of predicates are distinguished: the structural predicates represent objects and properties, and utility predicates that link objects and properties predicates. The feature space $\mathcal{R}$ is thus a richer version of the original feature space $\mathcal{F}$ as it contains extra information about individual features and relates them via predicates. The rules, in turn, relate predicates to each other and to class values, and thus capture the domain structure. The information about what predicates should be used to represent instances is domain specific and introduced manually as the background knowledge.

The rules extracted from the data represented by first-order predicates grounded in the original feature space $\mathcal{F}$ are called non-abstract relational features. Alternatively, the predicates can be grounded in $\mathcal{F}'$ - an abstract representation of $\mathcal{F}$ shared across domains. In this case, we extract abstract relational features. The values from $\mathcal{F}$ are abstracted away with respect to the first-order predicates and class values. The abstractions $\mathcal{F}'$ are learned from the training data of source and target domains. The new features $\mathcal{R}$ grounded in $\mathcal{F}$ (non-abstract) or $\mathcal{F}'$ (abstract) are then transferred and used for the target classification task. If the first-order predicates $\mathcal{P}$ wrapping the original features from $\mathcal{F}$ are well chosen (or learned from the data), and if the original features are mapped to an abstract feature space $\mathcal{F}'$, then the extracted rules $\mathcal{R}$ can be transferred across domains of the same family. A family of domains is characterized by a shared underlying relational structure and abstract feature space $\mathcal{F}'$.

To extract abstract relational features, we first represent the data in the first-order logic form if it is not in this form already. Then, we combine the training data from source and target domains and map all feature values to a common feature space. Next, we apply an association subgroup discovery algorithm to extract abstract relational features. Once the abstract relational features have been extracted, they are transferred to a target domain $\mathcal{D}_T$ to carry out a supervised target learning task $\mathcal{T}_T$. Depending on the amount of overlap between domains, each instance $x_i$ can be either represented solely in terms of $k$ new features $\{f'_1, f'_2, \ldots, f'_k\} \in \mathcal{F}'$, or it can be augmented with these features $\{f_1, \ldots, f_l, f'_1, \ldots, f'_k\}$, or the new features can be used as a bias to discover yet another set of features in $\mathcal{D}_T$. In this work, we only use features “as is”, without conducting additional search in the target domain.
However, in cases when there is not enough training data in the target domain, or if there is a lot of irrelevant, noisy data in source domains, or if there is little overlap between source and target domains, we extract features separately from target and source domains and combine them to train the classifier for the target learning task. In other words, we augment features extracted from the target domain with features extracted from source domains. This is done in cases when we don’t want information about the target domain to be lost in the rules extracted from source domains. Note that features that we extract from the target domain also have an abstract relational representation.

### 3.2 Data Representation

#### 3.2.1 First-Order Logic Definitions

First-order logic is a powerful formal language that employs predicates and quantification in addition to dealing with simple declarative propositions, like propositional logic. Predicates are boolean valued functions which truth values depend on the values of the variables. Quantification is achieved through existential (\( \exists x \text{Person}(x) \)), there exists an x that is a student) and universal quantifiers (\( \forall x \text{Person}(x) \)), all x are persons).

Logic symbols can be divided into two groups: logical symbols and non-logical symbols. The first kind includes symbols like quantifiers \( \exists \) and \( \forall \), logical connectives - conjunction \( \lor \), disjunction \( \land \), implication \( \rightarrow \), negation \( \neg \), etc, punctuation symbols, and an infinite set of variables. The non-logical symbols include predicates (relations), functions and constants. Constants describe the objects in a domain and can have types. We denote them by a single upper case letter. Variables act as placeholders to allow for quantification. We describe them with lower case strings. Predicates represent relations in the domain and are represented by strings starting with an upper case letter. Functions of tuples of objects are denoted by lower case symbols (e.g. \( f(x) \)). A term is a constant, a variable, or a function applied to terms. Ground terms contain no variables. An atom is a predicate applied to terms. A positive literal is an atom, and a negative literal is a negated atom.

A formula consists of literals connected by logical connectives (e.g. \( \lor \) and \( \land \)). A rule is an implication \( X \rightarrow Y \), where Y is a target class, and X is or a logic formula, in our case, it is a conjunction of features. A relational feature is represented by a formula obtained from a rule after its class value is dropped. We extract relational rules, but when features are transferred or used for classification, the class values are dropped. A relational feature becomes abstract when its predicate variables are grounded in a feature space (in this case features are predicate variables) shared across all domains, and when existential quantifiers are introduced to generalize over the basic relations.

Our first-order logic representation of domains has no functions and includes only conjunctions as predicate connectives in the formulas. However, it is possible to extend it to add negation and implication.

#### 3.2.2 Domain Representation

We represent each domain instance in terms of simple relations or first-order logic predicates that include structural predicates describing objects, their parts and properties, and utility predicates that link structural predicates. All the predicates are grounded - contain constants instead
of variables. This is different from most relational algorithms (e.g. RSD) where instantiation is done as a separate step. Instances can be of variable length - contain different number of predicates. Thus, each instance will take on a form:

\[
\text{ObjectPredicate1}(\text{variable1}), \text{ObjectPredicate2}(\text{variable2}), \ldots, \text{PropertyPredicate1}(\text{variable1}, \text{constant1}), \text{PropertyPredicate2}(\text{variable1}, \text{constant2}), \ldots, \text{UtilityPredicate}(\text{variable1}, \text{variable2}), \ldots
\]

To prepare the data for rule extraction the following transformations have to be applied:

1. Convert the data to a first-order logic form.
2. Abstract away constant values associated with different types of predicates and classes.
3. Add existential and universal quantifiers to allow for compact representation of repeated predicates.

Figure 3.1 illustrates a general idea behind the new data representation - a conjunction of predicates with abstract values.

![Figure 3.1: Abstract data representation](image)

The first step is to represent the data in a first-order logic form. The data can exist in a vector form (each instance is represented by a set of independent features), or it can be in a relational form and already consist of a set of predicates. In the latter case we don’t have to do anything further. Otherwise, a mapping between the original raw data representation and its logic form has to be provided. In this case, the structural predicates (representing objects and properties) can be obtained from the original features that describe the instances or with the help of the background knowledge (manually), e.g. predicates describing dimension of the vector. Utility predicates (connecting objects and properties) are typically not present in the original representation, and are thus introduced with the help of the background knowledge. We provide an API that constructs first-order instances, but requires users to specify predicate names and values. For this purpose, we distinguish three types of predicates: \textit{object predicates}, \textit{property predicates}, \textit{connect predicates}. The first two are structural types of predicates, while the last one is a utility type.

- \textit{object predicates} - describe an object that has a name and a set of properties, e.g. a \texttt{Car(car1)}.
- \textit{property predicates} - describe object properties, e.g. \texttt{Length(car1,long)} describes the length of the \texttt{car1} instance of the \texttt{Car} object.
• **connect predicates** - relate objects to each other in some way, e.g. \(\text{Adjacent(car1,car2)}\) describes that two Car objects are adjacent to each other. In addition to objects, object properties can also be connected. For example, it could be relevant that a long car is adjacent to a short car. Then, \(\text{Adjacent(car1.Length(long),car2.Length(short))}\) relation can be added, so that it is given a chance of being selected as part of the rule extraction process.

Thus, in order to convert the original data to a relational form, some assumptions have to be made about the domain, e.g. the types of predicates that should be added. The best approach is to add all possible relations that describe the domain, such that irrelevant ones can be filtered out at the later stages, e.g. during abstract relational feature extraction, see section 3.3. It is possible to have a general, domain independent, method for converting the data from its original to relational form. An example of a data conversion tool capable of connecting to data sources of an extensive variety of types including relational (i.e Progol) is Sumatra TT\(^1\).

The second step maps feature values or constants associated with different predicates to a space shared between source and target domains. This can be achieved in many different ways depending on the original data representation. One way is fully automatic and accomplished through gathering statistical information about values associated with particular predicates and using it to abstract away specific values, e.g. replace them with value ranges or percentiles. Alternatively, any model of the feature space can be constructed and used for mapping of the original values to abstract ones.

The last step - adding relations equivalent to first-order quantification - is useful for domains in which instances can contain repeated predicates (e.g. several objects with some or all property values shared). In this case, stating that all objects in an instance have the same value for a given property, or that there exists at least one property with a certain value can add additional piece of information potentially useful for prediction.

Let’s consider an example - a vector representation of the trains data set\(^2\), where each instance represents a train object with its parts and properties in terms of the following features:

**general properties of the train:**
- **class** values (east, west).
- **Num-of-cars** (integer in [3-5]), **Num-of-different-loads** (integer in [1-4]),

**attributes describing each car:**
- **num-wheels** (integer in [2-3]), **length** (short or long), **shape** (closedrect, diblopnrect, ellipse, etc),
- **num-loads** (integer in [0-3]), **load-shape** (circlelod, hexagonlod, rectanglod, trianglod), 10 Boolean attributes describing whether 2 types of loads are on adjacent cars of the train.

A train instance in a vector form will look like this:

\[
east, 5, 4, 2, \text{long}, \text{openrect}, 3, \text{rectanglod}, 2, \text{short}, \ldots, 1, 0, \ldots
\]

\(^1\)http://krizik.felk.cvut.cz/sumatra/
\(^2\)http://archive.ics.uci.edu/ml/datasets/Trains
The same instance in first-order logic form, after step 1, will look like this:

\[
\text{east, Num-of-cars(5), Num-of-different-loads(4), Car(car1),}
\]
\[
\text{Num-wheels(car1,2), Length(car1,long), Shape(car1,openrect),}
\]
\[
\text{Num-loads(car1,3), Load-shape(car1,rectanglod),}
\]
\[
\text{Car(car2), Num-wheels(car2,2), Adjacent(car1, car2),...}
\]

As part of step 2, the values are replaced by abstract values shared between domains. One way this can be achieved is by replacing the original values with value ranges learned from the data and associated with a particular type of a predicate. For example, if for class \textit{east} the values of \textit{Num-of-cars} predicate type range from 4 to 6, then a predicate \textit{Num-of-cars(5)} will be replaced with \textit{Num-of-cars(4-6)}. Many different generalizations are possible and encouraged. For instance, numeric values can be compared to the minimum and maximum values in the distribution, e.g. \textit{Num-of-cars(5)} will be replaced with \textit{Num-of-cars(gt(4))}, where \textit{gt} stands for “greater than”. The idea is that predicates that are relevant (e.g. frequent and interesting) to describing each class can be chosen by the feature extraction algorithm. In this way, the right level of the abstraction is determined for transfer. Thus, this step provides an opportunity to introduce additional background knowledge not explicitly present in the data.

In step 3 predicates representing quantifiers \textit{exists} and \textit{for-all} are introduced. These predicates indicate that either all relations of a kind have the same values (e.g. \textit{For-all(car, Length(car, long))}), or that there exists at least one value of a kind (e.g. \textit{exists(car, Length(car, long))}). This notation is equivalent to \(\forall \text{car} \in C, \text{Length(car, long)}\) or \(\exists \text{car} \in C, \text{Length(car, long)}\), where \(C\) is a set of all cars.

We have implemented a system that provides an interface for converting data in raw format to a first-order form. The system also has the capability to add quantification predicates and basic modeling of predicate values, e.g. representation in terms of value ranges extracted from the data.

### 3.3 RFE Algorithm

Next, given the data in relational form, we need an efficient algorithm that can extract general rules informative of class values and representing different subgroups within each class.

We consider two association subgroup discovery algorithms - SD-Map [5] based on FP-Growth [25] algorithm and APRIORI-SD [30] based on APRIORI [1] that we have described in section 2.7.3. Because we don’t have to worry about missing variables, given our data representation (instances can have variable length, so no need for inclusion of missing values), and because we would like to adopt the weighted relative accuracy measure, we use APRIORI-SD algorithm’s rule post-processing scheme.

The overall algorithm, which we will refer to as RFE (Relation Feature Extraction) is described below (see Table 1). Note that most of the algorithm follows almost exactly the APRIORI-SD algorithm proposed in [30]. Here, however we use FP-Growth, which is more efficient than APRIORI, to extract statistically significant patterns. We also chose to extract and filter rules separately for each class to get better results for data sets with unbalanced class distributions.
Furthermore, we add an additional filtering step (see ruleFilter procedure in Algorithm 1) that can help eliminate less general rules. This heuristic simply eliminates rules covering examples which are also covered by more accurate (in terms of weighted relative accuracy) rules. The coverage and accuracy requirements help eliminate less general rules.

Algorithm 1 The Pseudo-Code for RFE Algorithm

```
algorithm RFE (Examples, Classes, minSup, maxNoPatterns, k)
  FilteredResultset = {}
  for Class in Classes do
    Resultset = {}
    Ruleset = FP-Growth(Examples, Class, minSup, maxNoPatterns) set all example weights of Examples to 1
    repeat
      BestRule = rule with the highest weighted accuracy value in Ruleset (computed using Equation 1)
      Resultset = Resultset ∪ BestRule
      Ruleset = Ruleset \ BestRule, decrease the weights of examples covered by BestRule (using example weighting scheme), remove from Examples the examples covered more than k-times
    until Examples = {} or Ruleset = {} 
    FilteredResultset = FilteredResultset ∪ ruleFilter(Examples, Resultset)
  end for
  return FilteredResultset

procedure ruleFilter(Examples, Ruleset)
  for each Rule in Ruleset do
    Construct a set of examples in Examples that it covers.
  end for
  Compare every rule Rule1 in Ruleset to every rule Rule2 in Ruleset.
  if examples covered by Rule1 include all the examples covered by Rule2
    and accuracy of Rule1 > accuracy of Rule2
    delete Rule2
  end if
```

24
Abstract Relational Feature Extraction

1. Convert data to first-order form
2. Combine training data from source and target domains
3. Map the data to a shared feature space
4. Use RFE to extract abstract relational features
5. Represent the data in the target domain in terms of new features
6. Train and evaluate the classifier in the target domain

Table 3.1: Pseudocode listing steps for abstract relational feature extraction

3.4 Putting it Together

The overall transfer task from source domains to a target domain is accomplished using steps specified in Table 3.1. The process is also described in Figure 3.2, where we list software modules that accomplish each of the steps. First, the data is converted to an abstract first-order logic form as described in 3.2.2. This step is represented by the Relational Data Converter module in Figure 3.2. Then, all the training data from source and target domains is combined to a single data set, mapped to an abstract representation as described in 3.2.2, and passed to an RFE algorithm that extracts the best rules for each class according to statistical significance and weighted average accuracy measures as described in 2.7.2. Once the abstract features have been extracted by the RFE algorithm, the features are then transferred to the target domain. The test and training data sets in the target domain are re-represented in terms of new features via Vector Data Converter module prior to training and evaluation with a classifier of choice.

![Figure 3.2: Overview of the Transfer Task](image-url)
Chapter 4

Experiments and Results

In order to validate the advantage of our approach compared to no transfer setup, we perform experiments in two types of domains with different properties: games domain with binary classification as a target task, and a human behavior recognition domain with multi-class classification as a target task. The first type of a domain is relatively noise free and simple in structure, while the second involves a time series of sensor firings with a lot of noise.

By using different domain types, we illustrate how our method performs under different conditions and also show different ways abstract features can be constructed with the help of the domain knowledge. In both applications, abstract features are inferred and added to the data. In the case of the games domain, we map the original feature values to a common feature space by automatically extracting value ranges from the data associated with different class and predicate values. In the case of the human behavior recognition domain, we construct models of feature values for different sensor types. The models are built automatically from data, with the background knowledge introduced in the form of additional features like number of sensor firings and firing durations.

In the following sections, we first describe the domains and the experimental setup in detail, then we present and discuss the results. We start with a games domain 4.1 followed by a human behavior recognition domain 4.2.

4.1 Games Domain

4.1.1 Domain Description

In the context of a games domain, our goal is to detect board configurations (or game states) in two player games that contain move sequences in which one player is unable to respond to all threats, and thus, the other player realizes an advantage. Such states typically serve as the main building blocks of game tactics. In chess, the examples are forks, skewers, pins, overloading, batteries, etc. Here, we focus on detecting a double threat - a game state where two pieces of the current player are attacked and only one can be saved. However, other kinds of states could also be considered, for example, a more abstract concept of a trap. A trap is a board state in which one of the players can make a move that looks good, but that is actually quite bad, as it can cause a greater loss in value in the long run.
Different games have very different rules and board configurations, and a double threat is represented very differently in these games. For example, Figure 4.1 shows double threat states in chess, tic tac toe, and 9 men’s morris games. A simple case of a double threat in chess is called a fork, when one of the sides is attacking two of opponent’s valuable pieces, and at most one of them can be defended. A tic tac toe equivalent of a double threat is when one of the players threatening two of the empty squares, while the opponent can defend only one. In nine men’s morris the goal is to form mills (three pieces in a row), that enable removal of one of the opponent pieces. Figure 4.1 shows a board configuration that allows the white player to move a piece back an forth and form a mill, each time removing the opponent’s piece - this is a form of a double threat.

A game state can be described at several levels of abstraction - visual (pictorial representation of a board), positional (exact or relative positions of player’s pieces on the board represented in textual form, e.g. PGN format), and game tree based, where a game tree is a directed graph with nodes represented by positions in the game and edges represented by possible moves which carry a certain value.

We use a game tree - the most general level of abstraction - to represent each game. In terms of the game tree, a trap can be defined as an inevitable reduction in value for one of the players after two or more consecutive moves (two or more levels of the game tree). A double threat is a type of a trap, such that there’s a reduction in value, and there are at least two possible winning moves that can be carried out by the opponent given the current board configuration. One can go on traversing the game tree recursively, until the game is over, to detect different kinds of traps, however, for our purposes, two-level game trees are sufficient. A game tree is accompanied by a turn that determines which of the two players can make the next move. Figure 4.2 shows positive and negative examples of game tree representations.

Using a definition of a double threat and two-level game tree representations of game states we construct positive and negative examples for chess, tic tac toe, and 9 men’s morris games, considering the following properties:

- There are two class values - positive (P) indicative of a double threat present in a given board configuration and negative (N) - indicative of no such threat.
- First level moves are represented by Amove predicates, second level moves by Bmove (A for first and B for second).
• Number of wins possible at each level prior to any action: Num-of-awins(count) and Num-of-bwins(count). This is a background knowledge information that can help distinguish double threats from other types of traps.

• There are two types of properties associated with each move: move value Val and move position Pos. The first one can be either 0 or 1 for each player, while the second is specific to the game and the board and represents a change in the board configuration.

• Note, that we only use structural predicates here representing objects - Amove and Bmove, their properties - Pos and Val, and instance properties Num-of-awins(count) and Num-of-bwins(count). There is no need for utility predicates connecting Amove and Bmove objects and their properties, as it is implicit that the former is always followed by the latter.

A simple first-order logic representation of this data looks like this:

\[ N, \text{Num-of-awins}(0), \text{Num-of-bwins}(1), \text{Amove}(x), \text{Pos}(x, p6-4), \text{Val}(x, 0), \ldots, \text{Bmove}(y), \text{Pos}(y, p2-1), \text{Val}(y, 1), \ldots \]

This example is negative, since the opponent wins, but the number of initial winning moves is only one (Num-of-bwins(1)), when a minimum of two is required for a double threat.

The same example with abstract values (e.g. value ranges) and existential quantifiers added as described in 3.2.2 looks like this:

\[ N, \text{Num-of-awins}(0), \text{Num-of-awins}(0-4), \text{Num-of-bwins}(0-4), \text{Num-of-bwins}(1), \text{For-all}(\text{amove}, \text{Val}(\text{amove}, 0)), \text{For-all}(\text{bmove}, \text{Val}(\text{bmove}, 1)), \text{Exists}(\text{amove}, \text{Pos}(\text{amove}, p0-4)), \text{Exists}(\text{amove}, \text{Pos}(\text{amove}, p0-4), \text{Val}(\text{amove}, 0)), \ldots, \text{Exists}(\text{bmove}, \text{Pos}(\text{bmove}, p7-2)), \text{Exists}(\text{bmove}, \text{Pos}(\text{bmove}, p7-2), \text{Val}(\text{bmove}, 1)), \ldots \]

Note that \( \text{For-all}(\text{amove}, \text{Val}(\text{amove}, 0)) \) is equivalent to \( \forall \text{amove} \in \{x|\text{Amove}(x)\}, \text{Val}(\text{amove}, 0) \). Similarly, \( \text{Exists}(\text{amove}, \text{Val}(\text{amove}, 0)) \) can be written as \( \exists \text{amove} \in \{x|\text{Amove}(x)\}, \text{Val}(\text{amove}, 0) \).

Statistical properties of the data sets are listed in table 4.1. We can see that both chess and tic tac toe are quite large, while 9 men’s morris has only 6 examples. We can also see that
the examples of the chess data set are highly unbalanced - there's a lot more negative examples than positive. The other two data sets have an equal example distribution. The RFE algorithm should handle both cases well as it extracts rules for each class value separately.

4.1.2 Experimental Setup

To carry out a transfer task in games domains, we use two data sets as source domains used for feature extraction and one data set as a target domain used for a classification learning task. The features are extracted by the RFE algorithm from source domains. Then, test and training data in the target domain are represented in terms of these new abstract features and are used to train and evaluate the classifier. We use a simple vector-based representation of the data, where 1 indicates that the example is covered by a given rule or feature, and 0 otherwise. It is also possible to represent each instance as a set of features (or feature ids) that it covers. We use SVM\(^1\) in all experiments, but any other classifier can be used as well. This version of SVM classifier predicts accuracy values for multi-class classification using one-against-one approach.

\(\text{Tic tac toe}\) and \(9 \text{ men’s morris}\) data sets are computer generated, while the chess data set has been extracted from real games in PGN (Portable Game Notation) format. Conversion of data to an abstract first-order logic form has been accomplished using the API of the DataConverter software we have developed.

RFE Algorithm requires a minimum support value to be specified. Support in the context of RFE is the number of examples containing a rule. We set support value dynamically, by requiring each rule to be statistically significant if at least 50% of data instances contain it - support threshold of 0.5. In some cases, e.g. for a combined data set of chess and tic tac toe games, we increase the threshold to 0.7 or 70% to make sure more general rules are extracted from such large data sets.

In our experiments, we first analyze the extracted rules to note the role played by the filtering step of the RFE algorithm 3.3, the abstract feature values, and quantifiers. Then, we conduct the two types of experiments to establish the value brought in by abstract relational features and evaluate our transfer learning approach.

In the first experiment, we compare classification accuracy on chess and tic tac toe data sets represented in terms of abstract and non-abstract relational features. The non-abstract relational features differ from abstract ones in that they are instantiated to non-abstract values from the original feature space of the domain and don’t make use of quantification. In other words, they are the original features of the domain put into relational first-order form. In both cases the features are extracted from the target training data only, no source domains

\(^1\)http://www.csie.ntu.edu.tw/~cjlin/libsvm/
are involved. The idea here is to determine the role that abstract features play in improving the accuracy within the same domain. We expect abstract features to improve classification accuracy over non-abstract features. We split the data into test and training sets by choosing 5 data points for testing and using the rest for training. We select test points at random 10 times (conduct a 10-fold cross-validation) and average the results.

In the second experiment, we compare classification accuracy on 9 men’s morris data set expressed in terms of abstract, non-abstract, and transferred features. The abstract features are extracted from chess and tic tac toe domains. We do the same for chess and tic tac toe games, where for chess we use rules transferred from tic tac toe and 9 men’s morris domains, and for tic tac toe the rules are transferred from chess and 9 men’s morris domains. The goal here is to determine how well our transfer approach does in the presence of very little training data in the target domain. We split the data into test and training sets as follows. For 9 men’s morris we have only 6 instances for training and testing. For this reason, we leave 2 instances for testing (one of each class), and use the remaining 4 for training. We rotate the examples chosen for testing and average the results. For chess and tic tac toe, that have a lot more instances, we use 10-fold cross validation and select 2 points for testing and 4 for training at random. For all three domains, we expect the transferred features to give us the highest accuracy and the abstract features to have accuracy higher than that of the non-abstract features.

4.1.3 Results

We have applied RFE to chess and tic tac toe games. The extracted rules or abstract relational features are listed in Tables 2 and 3. Let’s first examine a few rules from table 2. For example, rule

N: [For-all(bmove, Val(bmove,1)), Num-of-awins(0-4), Num-of-bwins(0-4)]

can be written in a first-order logic form as follows: \(\forall bmove \in \{x\} Bmove(x)) Val(bmove, 1) \wedge Num-of-awins(0-4) \wedge Num-of-bwins(0-4) \Rightarrow N\). This rule states that if all \(bmove\) have value of 1 and the number of awins and the number of bwins range anywhere from 0 to 4, then the instance belongs to a negative class. Another rule,

P: [For-all(amove, Val(amove,0)), For-all(bmove, Val(bmove,1)), Num-of-bwins(2-4)]

or \(\forall amove \in \{x\} Amove(x), Val(amove, 0) \wedge \forall bmove, Val(bmove, 1) \wedge Num-of-bwins(2-4) \Rightarrow P\) states that if all \(amove\) values are 0 and all \(bmove\) values are 1, and the number of bwins ranges from 2 to 4, then the instance belongs to the positive class. Both rules convey information about the presence or absence of a double threat in a game state.
Table 2 lists rules obtained with the filtering step in the RFE algorithm enabled. The rules are extracted for tic tac toe, chess, and combined chess and tic tac toe data sets. We can see that rules generated for all games are the same, except for slight variations in values observed with tic tac toe. This can be explained by the fact that RFE was able to extract very general rules shared across domains.

Table 2 Filtered Rules

Chess and Combined Chess and TTT data sets:
N: [Num-of-awins(0-4), Num-of-bwins(0-4)]
N: [For-all(bmove, Val(bmove, 1)), Num-of-awins(0-4), Num-of-bwins(0-4)]
P: [For-all(amove, Val(amove, 0)), For-all(bmove, Val(bmove, 1)), Num-of-awins(0), Num-of-bwins(2-4)]

Tic Tac Toe data set:
N: [Num-of-awins(0-1), Num-of-bwins(0-4)]
N: [For-all(bmove, Val(bmove, 1)), Num-of-awins(0-1), Num-of-bwins(0-4)]
P: [For-all(amove, Val(amove, 0)), For-all(bmove, Val(bmove, 1)), Num-of-awins(0), Num-of-bwins(2-3)]

Table 3 lists rules generated when the filtering step is skipped in RFE algorithm - ruleFilter procedure is not invoked. The rules that are in bold are the rules that are kept when the filtering step is added, and the rest are the rules that are filtered out. We can see that filtered out rules contain information that is too specific or redundant and is already covered by selected rules. For example, rule 1 contains more general information compared to rules 4 and 5, while rule 2 expresses the same information as rules 6 and 7, but more precisely and compactly (it’s enough to know that all bmoves are winning). Similarly, rule 3 is a more precise version of rules 8 and 9.

Table 3 Unfiltered Rules

Combined Chess and TTT data sets:
1) N: [Num-of-awins(0-4), Num-of-bwins(0-4)]
2) N: [For-all(bmove, Val(bmove, 1)), Num-of-awins(0-4), Num-of-bwins(0-4)]
3) P: [For-all(amove, Val(amove, 0)), For-all(bmove, Val(bmove, 1)), Num-of-awins(0), Num-of-bwins(2-4)]
4) N: [Num-of-awins(0-4), Num-of-bwins(0-4), Num-of-bwins(1)]
5) N: [Exists(amove, Val(amove, 1)), Num-of-awins(0-4), Num-of-bwins(0-4)]
6) N: [Exists(amove, Val(amove, 0)), Exists(amove, Val(amove, 1)), Num-of-awins(0-4)]
7) N: [Exists(amove, Val(amove, 0)), Exists(amove, Val(amove, 1)), Num-of-awins(0-4), Num-of-bwins(0-4)]
8) P: [For-all(amove, Val(amove, 0)), For-all(bmove, Val(bmove, 1)), Num-of-awins(0), Num-of-bwins(2), Num-of-bwins(2-4)]
9) P: [Num-of-awins(0), Num-of-bwins(2), Num-of-bwins(2-4)]

Table 4 shows the rules constructed from the data in non-abstract first-order form. We see here a few rules that include Num-of-wins relations that are introduced as background knowledge to help differentiate a double threat from a more generic trap. This information is already quite general and informs of the possible wins by either of the two sides prior to any action. The rest of the rules, however, are descriptions of specific (perhaps frequent) moves, which are not very transferable to other domains (e.g. there is no overlap in the move position values between domains).
It is clear that the extracted rules listed in Table 2 are highly general and can be easily transferred between domains. To show this, we first compare classifier performance on data represented in terms of non-abstract features to the classifier performance on data represented in terms of abstract features. The results are listed in Table 4.2.

We can see from Table 4.2 that the original features give quite high accuracy of 97% and 90% for chess and tic tac toe data sets respectively. This can be explained by the fact that the number of training data is quite large and allows to learn the needed patterns in the data. Moreover, when abstract features are used the accuracy reaches 100%.

We then investigate whether or not the general rules can be transferred to other domains. To do this, we treat each game as a target domain in turn, while using the other two games as source domains. For example, when 9 men’s morris game serves as a target domain, chess and tic tac toe are used as source domains from which new abstract relational features are extracted. Similarly, the other two combinations are: chess as target, tic tac toe and 9 men’s morris as source, and tic tac toe as target, chess and 9 men’s morris as source. The results in terms of classification accuracy for all three cases are listed in Table 4.3.

The 9 men’s morris data is not very plentiful and when basic features are used only 16.7% accuracy is achieved. Lifting the data to an abstract level changes the accuracy to 33.3%. However, the same data set represented in terms of abstract features extracted from the two other games achieves 100% accuracy. Similarly, the best performance for chess and tic tac toe games is achieved with transferred features - 95% and 100% respectively.

We expect our method to generalize to other games and tasks, e.g. detection of other kinds of traps. In this work, we include three games to evaluate our approach, but any number of different two-player games that have a game state equivalent of a double threat can be included. If a new game is added, e.g. Connect 4, then the game plays can be represented in terms of a game tree in first-order logic form with abstract variable values, just like the games we’ve discussed so far. Depending on the goal, the data from the new domain can then be used as either source or target data.
4.2 Human Behavior Recognition Domain

In addition to evaluating our approach on a relatively simple and noise free domains like games, we would also like to see how well it works on a less structured, noisy, and more realistic domains. For this reason, we apply our method to a human behavior recognition task.

The goal of human behavior recognition is automatic identification of activities in a home setting for applications like health care and intelligent systems. Transfer learning for human behavior recognition has been extensively studied by Kasteren et. al. [58]. They propose a method for applying transfer learning to time series using a hidden Markov model (HMM). The model is generative and allows for incorporation of unlabeled data during training.

In addition, they use meta-features to describe sensors. Each sensor can have a unique name and location, but with the help of the meta-features it can be mapped to a common group of sensors, e.g. Kitchen, Bathroom, etc. Meta features are thus used to smooth out the differences in sensor networks due to differences in house layouts. We use the same meta features in our experiments, but we also propose another way of dealing with this issue - through modeling of sensors.

To account for the differences in behavior of the inhabitants of different houses, a separate model is used for each house, each having its own set of model parameters. What is transferred is a prior distribution learned from source houses and used as a bias during construction of the target house model. Unlike this method, we propose transfer of explicit abstract features. Unlike [58], our model is not generative and does not use any of the unlabeled data. However, it can be extended to take on a generative character, where rules and their constituents can have confidence scores and can be used to bias a search carried out in the target domain, e.g. similar to DTM approach discussed in 2.3.

4.2.1 Domain Description

The data used for activity recognition is from a human behavior recognition domain, where the goal is to recognize activities of daily living (ADL), like cooking, sleeping, eating, etc, from
Table 4.4: Information about the datasets recorded in three different homes using a wireless sensor network. This table is taken from [58].

<table>
<thead>
<tr>
<th>House</th>
<th>House A</th>
<th>House B</th>
<th>House C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>26</td>
<td>28</td>
<td>57</td>
</tr>
<tr>
<td>Gender</td>
<td>Male</td>
<td>Male</td>
<td>Male</td>
</tr>
<tr>
<td>Setting</td>
<td>Apartment</td>
<td>Apartment</td>
<td>House</td>
</tr>
<tr>
<td>Rooms</td>
<td>3</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Duration</td>
<td>25 days</td>
<td>13 days</td>
<td>18 days</td>
</tr>
<tr>
<td>Sensors</td>
<td>14</td>
<td>23</td>
<td>21</td>
</tr>
<tr>
<td>Recorded by</td>
<td>Authors</td>
<td>Authors</td>
<td>Authors</td>
</tr>
</tbody>
</table>

wireless sensor network data [58]. It consists of three real world datasets, each corresponding to a different house with different inhabitants\(^2\). Information about each house and its inhabitants is given in Table 4.4. An example of house layouts (with sensors marked with red dots) is given in Figure 4.3.

Each data set includes dates and times annotated with activity labels and dates and times of sensor firings annotated with sensor types. There are a total of 24 annotated days for house A, 20 for house B, and 16 for house C. A summary of the data sets is given in Table 4.5. We can see that some activities are frequent but take a short time, e.g. brush teeth, while others are less frequent, but take longer times, e.g. leave house. We can also see differences in activity frequencies and durations between houses, which point out variations in habits of different house inhabitants.

In the context of transfer learning, the target task is classification of activities in a specific house. The target house has little or no training data available, while one or more of source houses have larger labelled datasets. The same list of ADLs is used for each house, while the sensor network for each house is different [58].

![Figure 4.3: Floorplan of houses A (left) and B (right), the red boxes represent wireless sensor nodes. This image is taken from [58]](https://sites.google.com/site/tim0306/datasets)
<table>
<thead>
<tr>
<th>Activity</th>
<th>House A</th>
<th>House B</th>
<th>House C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leave house</td>
<td>33</td>
<td>16</td>
<td>47</td>
</tr>
<tr>
<td>Toileting</td>
<td>114</td>
<td>28</td>
<td>89</td>
</tr>
<tr>
<td>Take shower</td>
<td>23</td>
<td>8</td>
<td>14</td>
</tr>
<tr>
<td>Brush teeth</td>
<td>16</td>
<td>12</td>
<td>26</td>
</tr>
<tr>
<td>Go to bed</td>
<td>24</td>
<td>11</td>
<td>19</td>
</tr>
<tr>
<td>Prepare Breakfast</td>
<td>20</td>
<td>7</td>
<td>18</td>
</tr>
<tr>
<td>Prepare Dinner</td>
<td>9</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>Get drink</td>
<td>20</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>Other</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4.5: The activities annotated in the different houses. The ‘Num.’ column shows the number of times the activity occurs in the dataset. The ‘Time’ column shows the percentage of time the activity takes up in the dataset. All non-annotated time slices were collected in a single ‘Other’ activity. This table is taken from [58].

Similar to the games domain, we represent the data in first-order logic form with the help of structural predicates representing Sensor objects and their properties: types, start and end times, number of firings and durations. In the same way we add quantifiers and abstract away predicate values, in this case using sensor models.

For example, an abstract relational representation of an activity instance containing information about a single sensor will look like this:

get-drink, Exists(sensor, Start-time(sensor, 849-1280), End-time(sensor, 978-1296), Fire-duration(sensor, 1-1), No-firings(sensor, 1-1), Sensor-type(sensor, KitchStor)),

Exists(sensor, Start-time(sensor, 849-1280)), Exists(sensor, End-time(sensor, 978-1296)),

Exists(sensor, Fire-duration(sensor, 1-1)), Exists(sensor, No-firings(sensor, 1-1)),

Exists(sensor, Sensor-type(sensor, KitchStor))

Note that Exists(sensor, Sensor-type(sensor, KitchStor)) can be written as \(\exists sensor \in \{x|Sensor(x)\}, Sensor-type(sensor, KitchStor)\). We have also added utility predicates connecting consecutive sensors and their properties, e.g. followedBy(sensor1, sensor2), followedBy(Sensor-type(sensor1, Kitchen), Sensor-type(sensor2, KitchStor)), etc. However, we have found that due to sparseness of data, these predicates didn’t contribute much to the performance and most of them didn’t make it into the extracted abstract relational features. For this reason, we don’t include them in our experiments.

The data sets presented here are noisy and sparse. Next, we provide additional details about peculiarities of the data such that they are considered in the experimental setup and analysis of the results.

First, annotated activities and actual firings don’t overlap exactly - sometimes firings happen later than activity start time, e.g. sleep sensor fires after the go to bed activity has started. At the same time, many activities coincide with spurious firings, some of which occur consistently for all instances. For example, bathroom door sensor always fires at the beginning of go to bed activity. Others happen randomly, e.g. bathroom sensor often fires to signify non-related activities like leave house or go to bed.
Second, the data is very sparse - activities often happen at slightly different or very different times, and there are simply not enough training instances to cover all possibilities. Also, some days do not include all of the activities.

Third, some activities are very hard to distinguish from others. For example, *use toilet* activity can happen any time during the day, and it often has the same sensors firing as for activities like *brush teeth* and *take shower*. Similarly, *get drink*, *prepare breakfast*, and *prepare dinner* activities often share sensors *kitchen* and *kitchen store*.

The transfer task is further complicated by two things. First, different houses exhibit different behavioral patterns of inhabitants, e.g. younger people tend to go to sleep much later than older, people with different work schedules leave their house at different times, and so on. Second, sensor readings vary substantially due to differences in house layouts, sensor locations, physical properties of sensors, and other factors. For example, different sensors within different houses can be associated with the same or different activities, while house layouts and sensor locations can influence which sensors will fire together when a certain activity happens. In addition, sensors firings themselves can exhibit different patterns, from short repeated firings, to firings lasting throughout activity duration and beyond. These patterns can be useful as they can help eliminate noisy information, e.g. overly long or overly short durations, and also give insights into the type of activity. For example, *leave house* is usually associated with two firings of the *front door* sensor - one when person leaves, and one when the person comes back. At the same time, *sleep* sensor often fires briefly throughout the night as it is probably triggered by human motion. To capture such patterns, we introduce additional features like number of firings and firing durations.

4.2.2 Experimental Setup

We apply our feature based approach with abstract data representation and RFE algorithm to the behavior recognition task. We extract new features from training data from source and target domains and transfer them to the target domain. We treat each of the three houses in turn as a target domain, while the rest are used as source domains. The data in the target domain is represented in terms of the new features and used with SVM classifier for training and evaluation.

To obtain annotated instances, we first discretize sensor data in time slices of length $\Delta t = 60$ seconds. We then align annotated activity timeslices with those of sensor firings. As part of this process we also merge consecutive sensor firings and record the number of such firings. For example, 300 - 320 and 320 - 330 is changed to 300 - 330 to get more accurate start times, end times, and durations.

Similarly to the games domain, the data is converted to a first-order logic form with the help of the *DataConverter* API. We extract the following initial features from the raw data: start and end times of sensor firings, duration for each firing, number of firings, and sensor types. In order to abstract away these feature values we learn a model of each sensor type. This way we can capture sensor properties that are shared across domains, for example, *kitchen* sensor usually fires in the mornings or evenings and has several short consecutive firings, while *outside* sensor associated with *leave house* activity tends to have longer durations and with two firings - one when the person leaves the house and one when he comes back. The model can be further updated when new sensors are introduced. Using this model of sensor features and the
RFE algorithm we extract new abstract features from the training data of the source and target domains. We then transfer the models of sensors and the extracted rules. The models are used in the target domain to give an abstract representation to the target data, and the features are used to give a new vector representation to this data, which is then used to train the classifier.

More precise behavioral patterns are generally extracted from the large training set of combined domains. However, such rules might not capture the specifics of the target domain. In addition, noise information can be introduced from the source domains, and useful information about the target domain might be left out. This is because there might not be enough target training data to be considered statistically significant in the presence of abundant data of source domains. To cope with this issue, we extract rules from source and target domains separately, with different support thresholds, and combine them into a single rule set. We use 0.5 support threshold for the target domain, and 0.3 for the source domains. Like with games, the threshold specifies the percentage of the data instances that contains a feature. The lower source domain threshold allows us to extract more general, but not overly general rules, while 0.5 threshold for target domain extracts rules that are specific enough to capture properties of the target domain, but not too specific that it overfits.

Because of the noise and sparseness of the data, not all test examples end up being covered by full rules. For this reason, we introduce a partial coverage as part of the data conversion process. It works as follows: if an example contains a full rule, we set the value for the feature represented by this rule to 1, otherwise, we set it to a fraction \( \frac{\text{covered predicates}}{\text{rule length}} \). Ablation tests have shown that this eliminates randomness when predicting a class for instances not covered by the rules.

Similar to [58], we split the data into training and test sets using a ‘leave one day out’ approach, such that one full day of sensor readings is used for testing and the remaining days are used for training. Each day is used as a test day once and the results are averaged. However, the results from run to run have very high variance that can be explained by the fact that not all the days contain all possible activities. For this reason, in addition to simply rotating the test days in a randomly selected data sample and averaging the results, we also add several iterations of selecting different data sets. We have found that 100 iterations gives us more stable results.

Experiments conducted in [58], use the F-measure to evaluated performance of the model, which is calculated from the precision and recall scores, see Equations 4.1. Since the target task is a multi-class classification, the notions of true positives (TP), false negatives (FN), and false positives (FP) for each class are obtained with the help of the NxN confusion matrix, where N is the number of classes, see Table 4.6.

\[
\text{Precision} = \frac{1}{N} \sum_{i=1}^{N} \frac{TP_i}{TP_i + FP_i} \quad (4.1a)
\]

\[
\text{Recall} = \frac{1}{N} \sum_{i=1}^{N} \frac{TP_i}{TP_i + FN_i} \quad (4.1b)
\]

\[
\text{F-Measure} = \frac{2 \times \text{precision \times recall}}{\text{precision + recall}} \quad (4.1c)
\]
Table 4.6: Confusion Matrix showing the true positives (TP), false negatives (FN) and false positives (FP) for each class. The $\epsilon_{ij}$ terms show the error between true class i and inferred class j. FN is the sum of the error terms in the same row, FP the sum of the error terms in the same column. This table is taken from [58].

However, we have found that this metric is not appropriate for our method and the given domain. The problem with the F-score measure as presented above is that in some cases it can be undefined due to division by zero in precision and recall computations. Precision is undefined if there are no positive predictions (TP + FP = 0), while recall is undefined when there are no positives in the test set (TP + FN = 0) [22]. These issues can arise with small or class-imbalanced test sets, which is true in our case. Because the data is sparse, we often get very unbalanced selection of classes in our test sets with some of the classes underrepresented or missing altogether. For example, we can have 30 instances of one class and only 2 of another class. If both instances of the second class are misclassified, then division by zero will happen when computing the precision, since both true positives and false positive counts are zero for that class. If the class is missing, true positives, false positives, and false negatives are all zero and, thus, division by zero will happen when computing precision and recall scores.

To deal with this issue, we use average accuracy measure to evaluate our model. A simple accuracy measure is defined by equation 4.2, while the average accuracy is defined by equation 4.3. In order to compute average accuracy, we first sum up all the correct predictions (true positives) and all sample sizes separately, and divide the two sums at the end. We do it this way to take into account differences in class distributions, such that misclassifying 1 of 2 datapoints for a single class is not treated as significantly as misclassifying 50 out of 100. In addition to accuracies, we also compute standard deviation and perform statistical significance tests (t-test) for transfer and no transfer cases.

\[
\text{Accuracy} = \frac{TP}{\text{DataSize}} \quad (4.2a)
\]

\[
\text{Avg. Accuracy} = \frac{\sum_{i=1}^{N} TP}{\sum_{i=1}^{N} \text{SampleSize}} \quad (4.2b)
\]
4.2.3 Results

We have applied RFE to human behavior recognition domains. Some examples of abstract relational features extracted for house A are listed in Table 5. We can see that the rules specify sensor properties with generic value ranges obtained from sensor models learned from the data. The rules are in fact descriptive of the activities, for example, `prepare-breakfast` starts and ends in the morning or early afternoon, while `prepare-dinner` activity is done in the evening. `Brush-teeth` activity has wide ranges for start and end times, as it is typically done in the morning and at night. The last rule in Table 5 is an example of a rule extracted from noisy readings. Here, `go-to-bed` activity is associated with the BathroomDoor sensor.

<table>
<thead>
<tr>
<th>Table 5 Rules Extracted for House A</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>brush-teeth</code>: Exists(sensor, Start-time(sensor, 410-1410), End-time(sensor, 347-1374), Fire-duration(sensor, 2-3), No-firings(sensor, 1-1), Sensor-type(sensor, BathroomDoor))</td>
</tr>
<tr>
<td><code>brush-teeth</code>: Exists(sensor, Sensor-type(sensor, Bathroom)), Exists(sensor, Sensor-type(sensor, BathroomDoor)), For-all(sensor, No-firings(sensor, 1-1))</td>
</tr>
<tr>
<td><code>use-toilet</code>: Exists(sensor, Sensor-type(sensor, Toilet)), For-all(sensor, No-firings(sensor, 1-1))</td>
</tr>
<tr>
<td><code>take-shower</code>: Exists(sensor, Exists(sensor, Fire-duration(sensor, 1-2)), Exists(sensor, Sensor-type(sensor, Bathroom)), Exists(sensor, Sensor-type(sensor, BathroomDoor)))</td>
</tr>
<tr>
<td><code>prep-breakfast</code>: Exists(sensor, Start-time(sensor, 571-984), End-time(sensor, 570-616), Fire-duration(sensor, 1-59), No-firings(sensor, 1-1), Sensor-type(sensor, KitchStor), Exists(sensor, No-firings(sensor, 1-1)), Exists(sensor, Sensor-type(sensor, KitchHeat)))</td>
</tr>
<tr>
<td><code>prep-dinner</code>: Exists(sensor, Start-time(sensor, 849-1280), End-time(sensor, 978-1296), Exists(sensor, Sensor-type(sensor, KitchStor)))</td>
</tr>
<tr>
<td><code>leave-house</code>: Exists(sensor, End-time(sensor, 978-1296), Exists(sensor, Sensor-type(sensor, KitchStor)))</td>
</tr>
<tr>
<td><code>go-to-bed</code>: Exists(sensor, Start-time(sensor, 354-904), End-time(sensor, 386-539), Fire-duration(sensor, 1-473), No-firings(sensor, 2-3), Sensor-type(sensor, SleepDoor))</td>
</tr>
<tr>
<td><code>go-to-bed</code>: Exists(sensor, No-firings(sensor, 1-1)), Exists(sensor, Sensor-type(sensor, BathroomDoor))</td>
</tr>
</tbody>
</table>

The results in terms of average accuracies for all three houses A, B, and C with and without transfer are shown in Table 4.7 and plotted on Figure 4.4. In addition to average accuracies, we include standard deviation and statistical significance (t-test) scores to give an idea of variations between samples and significance of the results.

Despite all the challenges introduced by the data sets, the transfer approach has managed to improve upon the non-transfer setup. Specifically, based on the t-test measures, some results are statistically significant, e.g. houses A 1, B 1, B 5, B 10, C 1 and C 5, some are not quite significant, e.g. house A 10 and A 20, and some are not significant, e.g. houses A 2 and A 5, C 2 and C 10, where each number after the house name represents the number of days in the training set. The reason we see only slight or no improvement in some cases is because the source domains don’t introduce more useful information than already present in the target training data.
Figure 4.4: Results for behavior recognition learning task in houses A, B, and C in terms of average accuracies with and without transfer. The x-axis are in log-scale and shows the number of labeled days used for training. The y-axis shows average accuracy of the classifier.
### Table 4.7: Classifier performance for houses A, B, and C in terms of average accuracies with and without transfer. The Size column specifies the number of days used for training, the second and third columns indicate average accuracies with and without transfer respectively, while the final column gives t-test scores to gauge statistical significance.

<table>
<thead>
<tr>
<th>House</th>
<th>#Days</th>
<th>Accuracy No Tr.</th>
<th>Accuracy Tr.</th>
<th>T-Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>59.61% ± 17.16</td>
<td>52.11% ± 21.10</td>
<td>0.0001</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>64.29% ± 15.12</td>
<td>64.17% ± 17.24</td>
<td>0.8917</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>67.40% ± 13.92</td>
<td>68.51% ± 15.04</td>
<td>0.1025</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>70.13% ± 16.35</td>
<td>71.49% ± 14.07</td>
<td>0.0516</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>70.97% ± 18.35</td>
<td>74.19% ± 13.78</td>
<td>0.0816</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>43.80% ± 14.87</td>
<td>51.92% ± 20.56</td>
<td>0.0007</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>48.85% ± 19.12</td>
<td>51.87% ± 17.42</td>
<td>0.0107</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>59.53% ± 17.92</td>
<td>64.76% ± 15.99</td>
<td>0.0001</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>65.82% ± 18.47</td>
<td>69.52% ± 19.99</td>
<td>0.0108</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>31.10% ± 17.27</td>
<td>35.31% ± 17.37</td>
<td>0.0001</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>42.30% ± 17.70</td>
<td>42.71% ± 16.28</td>
<td>0.6900</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>47.99% ± 17.68</td>
<td>50.70% ± 16.25</td>
<td>0.0006</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>51.72% ± 20.01</td>
<td>52.75% ± 18.87</td>
<td>0.2684</td>
</tr>
</tbody>
</table>

The cases with fewer days used for training are good candidates for taking advantage of transferred rules. At the same time, they are also susceptible to noise as there is not enough target training data to counter it. We can see that for house A the transfer case does worse. This could be explained by the fact that there’s not enough target training data to cancel out the negative transfer from B and C domains. We don’t see such negative transfer for houses B and C. Perhaps domain A contains more general information reflected in its rules than houses B and C. After all, house A has the most number of days compared to B and C and has many more annotated activities within each day.
<table>
<thead>
<tr>
<th>House</th>
<th>#Days</th>
<th>F-score No Tr.</th>
<th>F-score Tr.</th>
<th>T-Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>61.07 ± 18.73</td>
<td>50.91 ± 21.14</td>
<td>0.0001</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>64.14 ± 18.97</td>
<td>59.43 ± 18.79</td>
<td>0.0001</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>63.54 ± 14.86</td>
<td>64.45 ± 16.26</td>
<td>0.2129</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>66.77 ± 18.08</td>
<td>69.09 ± 15.12</td>
<td>0.0025</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>68.84 ± 16.90</td>
<td>71.22 ± 14.50</td>
<td>0.1843</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>52.05 ± 14.02</td>
<td>55.51 ± 28.35</td>
<td>0.2374</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>54.38 ± 22.25</td>
<td>58.07 ± 18.65</td>
<td>0.0055</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>61.13 ± 18.25</td>
<td>67.46 ± 16.19</td>
<td>0.0001</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>66.52 ± 19.41</td>
<td>72.77 ± 19.53</td>
<td>0.0001</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>28.73 ± 17.80</td>
<td>30.93 ± 20.61</td>
<td>0.0092</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>44.24 ± 20.19</td>
<td>44.84 ± 17.74</td>
<td>0.5162</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>46.18 ± 19.78</td>
<td>50.81 ± 17.27</td>
<td>0.0001</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>51.45 ± 21.96</td>
<td>55.68 ± 18.48</td>
<td>0.0079</td>
</tr>
</tbody>
</table>

Table 4.8: Classifier performance for houses A, B, and C in terms of F-scores with and without transfer. The Size column specifies the number of days used for training, the second and third columns indicate F-scores with and without transfer respectively, while the final column gives t-test scores to gauge statistical significance.

In addition to average accuracy, we also provide results in terms of F-score in order to allow for a rough comparison to HMM performance in [58]. We have dealt with undefined precision and recall scores by omitting them - if precision or recall is undefined for some class, we skip that class from average precision or recall. Note that unlike [58] we do not use any of the unlabeled data for our experiments. We provide our results in Table 4.8 and Figure 4.5. The F-score measures we report, considering the differences in experimental setups, do roughly as well as or better than the results reported in [58]. For example, for houses A and B the highest F-scores are 71 and 73, whereas results reported in [58] never go over 70 and 60 respectively.

Similar to games domains, the extracted rules can be transferred between any number of domains of the same family. In this case, between any number of different houses with different sensor names and readings. If a new house is added to source domains, then sensor models can be augmented with this new information, and new features can be extracted for transfer. If the names of the sensors don’t belong to any of the known categories, they can be either assigned to one of the existing categories or sensor models (the one they are most similar to), or they can be used to create a new category. This is where the advantage of modeling a shared feature space becomes apparent.
Figure 4.5: Results for behavior recognition learning task in houses A, B, and C in terms of F-scores, with and without transfer. The x-axis are in log-scale and shows the number of labeled days used for training. The y-axis shows F-scores of the classifier.
The experimental results indicate that our transfer learning approach works for both types of domains we have applied it to - games and human behavior recognition. In the case of the games domain we don’t observe negative transfer and the improvement in results is significant compared to no transfer approach. When applied to a human behavior recognition domain, our approach does better in general compared to no transfer approach. In this case, however, we observe some negative transfer, when little training data is available in the target domain. Negative transfer happens when the source domains reduce the performance of learning in the target domain [46]. This concept is not new, but it is not well explored as of to date. Rosenstein et al. [55] empirically showed that if two tasks are too dissimilar, then brute-force transfer may hurt the performance of the target task. Some work has been done to analyse relatedness among tasks [7], which provides insights into ways the negative transfer can be detected and avoided automatically.

In case of the human behavior recognition domain, negative transfer can be explained by the presence of noise in source domains, little overlap between source and target domains, and not enough training data in the target domain to overpower the noise or irrelevant information introduced by source domains. Negative transfer for this domain is not a surprise and has been reported in [58] that uses a generative model and a parameter transfer approach.

Given that our approach yields positive results for the two types of domains, the following questions can be asked: Why does it work? How is it different from other methods proposed for the same task? How can it be improved? First, we discuss elements of our method that we think make it successful and different from other methods. Next, we discuss what our method lacks compared to other methods. Finally, we specify how our method can be improved.

The main elements of our approach are the data representation and the feature extraction algorithm. The data instances are represented by conjunctions of first-order predicates with variables instantiated in the feature space shared across domains. The logic-based data representation allows for a richer representation of the domain. Additional information can be embedded with the help of quantifiers, utility predicates linking objects and their properties, background knowledge introduced with more predicates, and so on. The abstract features space allows for a maximal overlap between domains, which is vital for a successful transfer.
Table 5.1: Properties of transfer learning methods

<table>
<thead>
<tr>
<th>Approach</th>
<th>Model Type</th>
<th>What is Transferred</th>
<th>Supervised</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMM</td>
<td>yes</td>
<td>model parameters</td>
<td>semi-supervised</td>
</tr>
<tr>
<td>DTM</td>
<td>no</td>
<td>MLN of second-order cliques</td>
<td>unsupervised</td>
</tr>
<tr>
<td>TAMAR</td>
<td>no</td>
<td>MLN of first-order cliques</td>
<td>unsupervised</td>
</tr>
<tr>
<td>Our Approach</td>
<td>no</td>
<td>grounded first-order rules</td>
<td>supervised</td>
</tr>
</tbody>
</table>

The goal of the data representation described above is to add as much information about the domain as possible. The goal of the feature extraction algorithm is to filter out all the irrelevant information and keep more general patterns that are shared across domains and are useful for the classification learning task. Thus, the success of our approach, as we see it, is in the generality and informativeness of rules that are transferred across domains, achieved through representation and efficient pattern mining.

This approach, however, lacks some of the advantages put forth by other methods. For example, unlike the HMM model based approach proposed in [58], our approach does not take advantage of the unlabeled data and requires at least some labeled data in the target domain. In this regard, it is more similar to MLN based methods like DTM [19] and TAMAR [42] that require some training data in the target domain to carry out a search using transferred features as a bias.

MLN approaches, unlike our method, transfer first or second-order logic cliques - sets of predicates wrapped in Markov logic networks. These MLN are learned for a source domain and used to aid in the learning of an MLN for a target domain. The relations between predicates are not specified in this case, and are determined as part of the search carried out in the target domain - some of the predicates are dropped, some are negated, new predicates can be added, and so on. Predicate values are not transferred, but are grounded in the data of the target domain. This makes these methods more general and more flexible.

Our method does not carry out additional search in the target domain. Instead, we capitalize on generality of the transferred rules. We thus transfer more information compared to other methods - specific relations between predicates (e.g. conjunctions) as well as the the values of predicate variables (abstract values obtained from domain models). Unlike MLN approaches, our features are also informative of class values. Transferred and revised MLN can be used as a relational model for inference or reasoning in the target domain, but it is not directly suitable for classification. We list properties of all these methods in Table 5.1.

Despite some of the disadvantages listed above, our method has three main advantages - flexibility of incorporating domain knowledge, increasing the overlap between domains through mapping of feature values to a shared feature space, and efficiently extracting useful general rules for transfer. An argument can be made that the mapping from the original feature space to an abstract one is domain specific and manual. However, an out of the box approach of finding such mappings can be provided. We have implemented two ways this can be done - through value ranges and feature models extracted automatically from the data. Either approach can be used with any of the domains, although, the first one is more appropriate for simpler, not very sparse or noisy domains. The second method is more flexible - it allows one to include additional background knowledge and update the model over time by adding new information as it becomes available. For example, if a new sensor is added to a house in a human behavior
recognition domain, its readings can be included in the transfer framework in two ways - either by augmenting an existing model or by learning a new model.

Thus, our method is appropriate for transferring knowledge in the form of explicit rules between families of related domains - domains that share the underlying structure and abstract concepts. The approach itself is general, but it constructs features that are specific to different types of domains. It automatically extracts abstract features from first-order data representation by learning and adding abstractions like quantifiers and models of feature values, and by efficiently extracting useful and informative rules. The user involvement can happen in two cases: 1) specifying first-order predicates representing the domain, and 2) adding background knowledge to help construct better feature models. The first case can sometimes be avoided with the help of tools that do automatic conversion of the data to a relational form. The second case is optional and can be used when a better model is desired and the background knowledge can help.

We have previously mentioned that a bigger goal of transfer learning can be to automatically determine at what level of abstraction features should be transferred between domains, perhaps based on the amount of overlap between them. If there is little overlap between domains, then deep transfer can be used, if only distributions differ, then shallow transfer is more appropriate, in other cases, mid-level transfer could be possible. Mid-level transfer is where our approach fits in.
Chapter 6

Conclusions and Future Work

We have developed a transfer learning approach that combines an abstract relational data representation and an efficient association subgroup discovery algorithm for extraction of abstract relational features. The approach provides a way of incorporating domain knowledge into the extracted features. The new features are transferred from source domains to a target domain, where they are used to aid a classification learning task. Our method has been shown to improve upon no transfer setup in two types of domains - games and human behavior recognition. We have compared it to other transfer learning methods, like HMM, DTM, and TAMAR and pointed out it’s advantages and disadvantages.

The approach can be improved and made more flexible in several ways. For example, similar to MLN approaches, the features can be used as a bias and further altered and fitted to the target domain. This can help avoid negative transfer. Richer formulas can also be introduced, e.g. negations and disjunctions. If domains are sparse and it is difficult to obtain a good model of the shared feature space, then it could be possible to do fuzzy matching of the rules - values don’t have to match exactly. For this, a similarity function should be defined, which can give fuzzy matching of the predicate values. It should be possible to make use of the unlabeled data, perhaps by using it to construct better models of the feature spaces, e.g. adjust value ranges. Another important item for future work is to apply this approach to another realistic domain with the goal of establishing a role of utility predicates that link objects and properties. In human behavior recognition domains, the data is too sparse to make a good use of information introduced by utility predicates, and games domains are too simple to benefit from it.
Bibliography


