Fundamenta Informaticae XX (2011) 1–27

IOS Press

Relational Feature Mining with Hierarchical Multitask kFOIL*

Elisa Cilia

Département d'Informatique†

Université Libre de Bruxelles, Belgium

ecilia@ulb.ac.be

Niels Landwehr

Department of Computer Science

University of Potsdam, Germany

landwehr@cs.uni-potsdam.de

Andrea Passerini

Dipartimento di Ingegneria e Scienza dell'Informazione

University of Trento, Italy

passerini@disi.unitn.it

Abstract. We introduce hierarchical kFOIL as a simple extension of the multitask kFOIL learning algorithm. The algorithm first learns a core logic representation common to all tasks, and then refines it by specialization on a per-task basis. The approach can be easily generalized to a deeper hierarchy of tasks. A task clustering algorithm is also proposed in order to automatically generate the task hierarchy. The approach is validated on problems of drug-resistance mutation prediction and protein structural classification. Experimental results show the advantage of the hierarchical version over both single and multi task alternatives and its potential usefulness in providing explanatory features for the domain. Task clustering allows to further improve performance when a deeper hierarchy is considered.

*A preliminary version of this work was presented at the Bio-Logical workshop of AI*IA 2009.

†This work was done while she was at Dipartimento di Ingegneria e Scienza dell'Informazione, University of Trento, Italy

1. Introduction

Multitask learning [6] deals with the problem of exploiting information on related tasks in order to improve predictive performances. Existing approaches rely on some type of parameter sharing among tasks, like learning a common hidden layer representation in multitask feed-forward neural networks [6], employing common priors for task dependent parameters in hierarchical Bayesian models [2], or both [1]. Most existing approaches assume that task models can be represented as parameter vectors over which to define prior distributions. In a full relational setting, either domain knowledge allows to explicitly encode relationships between tasks, or one needs to resort to multitask relational structure learning, and specifying priors over task-dependent structures is quite challenging. A notable example in this direction is the recent work by Deshpande et al [13] for multitask learning of probabilistic planning rules from a common set of rule prototypes. Taking a discriminative viewpoint, kFOIL [24] greedily learns a relational kernel representation with high discriminant power for a certain task. The algorithm was recently [25] extended to deal with multitask problems by learning a shared relational representation which is discriminative for multiple tasks simultaneously. However, learning a single common representation prevents the model from discovering task-specific features, a problem affecting multitask neural networks as well. Moreover learning a single common representation can be highly suboptimal if tasks relatedness is not high [27]. We propose a simple extension where the common representation is viewed as an initial structure which is further specialized on a per-task basis. This simple approach can be generalized to a deeper hierarchical process of refinements whenever a hierarchical clustering of the tasks is available or can be learned from data.

We applied our hierarchical kFOIL algorithm to a dataset of HIV resistance mutations [35]. The dataset reports wild type and mutations of reverse transcriptase, a viral protein which is essential for the success of the viral propagation. A mutation can confer the mutant resistance to one or more drugs, for instance by modifying the inhibitor target site on the protein. Due to the high mutation rate of viruses, mutants typically have multiple mutations, ranging from 6 to 90 on this dataset. A possible problem in this setting is predicting which drugs a certain mutant is resistant to. This can be naturally addressed as a multitask learning problem, where each drug is a single task. However, the relationship between tasks is not necessarily strong as different drugs can target different sites in the protein. Indeed, plain multitask learning will sometimes result in a performance worsening with respect to single task in this setting, especially when drug classes are considered as tasks [35]. On the other hand, our hierarchical refinement approach succeeds in combining the advantages of the two methods, being always at least as good as either alternative. Task clustering allows to further improve performance when a finer grain of tasks is obtained by considering individuals drugs as tasks instead of drug classes. We also applied the hierarchical multitask approach to a protein structure classification problem, within the SCOP [18] hierarchy. In a setting in which quite distantly related tasks are combined, our hierarchical approach is always significantly better than both single task and multitask alternatives. Nonetheless, our main concern here is not the predictive performance itself, but rather the ability of the method to provide insights into the reasons for a certain resistance or a certain fold. From this viewpoint, multitask and hierarchical approaches have an additional advantage: the models learned can be inspected in order to relate general and task-specific features.

The rest of the paper is organized as follows: Section 2 briefly describes the original kFOIL formulation and its multitask version; Section 3 introduces our extension for hierarchical multitask learning and Section 4 describes the task clustering approach; related work is discussed in Section 5; Section 6 reports

our experimental evaluation and discusses the models obtained. Conclusions are drawn in Section 7.

2. kFOIL: Learning Relational Kernels

This section introduces concepts and terminology from relational learning, and briefly reviews the original kFOIL algorithm [24] and its multitask extension [25].

2.1. Relational Learning and Hypothesis Search

We consider relational classification problems, where training examples and background knowledge, as well as the features that are induced during learning, are represented in first-order logic. More specifically, definite clauses, which form the basis for the programming language Prolog, are used as the representation language. A definite clause is an expression of the form $h \leftarrow b_1, ..., b_n$, where h and the b_i are atoms. Atoms are expressions of the form $p(t_1, ..., t_n)$ where p/n is a predicate symbol of arity n and the t_i are terms. Terms are constants (denoted by lower case), variables (denoted by upper case), or structured terms. Structured terms are expressions of the form $f(t_1, ..., t_k)$, where f/k is a functor symbol of arity k and $t_1, ..., t_k$ are terms. The atom h is also called the head of the clause, and $b_1, ..., b_n$ its body. Intuitively, a clause represents that the head h will hold whenever the body $b_1, ..., b_n$ holds.

As an example, consider the atom $mut(A, h, C, y)$ indicating a mutation that results in the replacement of the amino acid "Histidine" by the amino acid "Tyrosine". The constants "h" and "y" represent "Histidine" and "Tyrosine", respectively. A and C are variables that are matched against a particular example; A indicates an example identifier and C the position at which the mutation occurs. Furthermore, consider the clause

$$resistant(A, nrti) \leftarrow mut(A, h, C, y), position(C, 208)$$

encoding that a mutation resulting in a change from "Histidine" to "Tyrosine" and occurring at position 208 entails resistance to the drug $nrti$. Such a clause can be matched against an example by grounding, and if the matching operation is successful the clause is said to *cover* the example.

Relational classification problems have traditionally been studied in the field of *inductive logic programming* (ILP). The goal of learning in ILP is to identify a hypothesis that covers all positive and no negative examples. Hypothesis typically take the form of a set of first order clauses, which can be interpreted as a set of relational features or rules that characterize the target concept. An example is classified as positive by the hypothesis if it is covered by one of its clauses.

There are different approaches to searching for an (approximately) optimal set of clauses within a predefined hypothesis space $\mathcal{H}$ called the *language bias*. A central idea in most ILP systems is to structure the search space $\mathcal{H}$ according to generality. A hypothesis $G \in \mathcal{H}$ is called more general than another hypothesis $S \in \mathcal{H}$, denoted $G \preceq S$, if all examples covered by S are also covered by G. The generality relation induces a lattice on the hypothesis space $\mathcal{H}$, and thus provides a way to systematically search $\mathcal{H}$. A popular approach is to search the lattice top-down, that is, from general to specific hypotheses, using a *refinement operator*. A refinement operator ρ takes a clause c and returns all specializations $c' \in \rho(c)$ of the clause that fall within the language bias. In the simplest case, these specializations (or *refinements*) are obtained by simply appending a literal to the clause c. For example, the clause $c' = \leftarrow mut(A, h, C, y), position(C, 208)$ is a refinement of the clause $c = \leftarrow mut(A, h, C, y)$. Note

that c will match any example matched by c'; thus, a hypothesis in which the clause c is replaced by the clause c' becomes more specific.

Top-down search based on refinement operators is the main principle underlying many ILP algorithms. For instance, incremental rule learners such as the well-known FOIL algorithm greedily search for a set of clauses that covers all positive examples by performing a hill-climbing search in the hypothesis space using a refinement operator [31].

2.2. The kFOIL Algorithm

kFOIL [24] is a statistical relational learning approach that combines techniques from inductive logic programming—specifically, the FOIL algorithm [31]—with kernel methods. ILP systems such as FOIL learn a first-order logical hypothesis for the target concept. This approach has several advantages, including that a large and flexible space of hypotheses is considered, and that the final model is readily interpreted by human experts. On the negative side, ILP methods do not always incorporate statistical robustness principles needed to handle noise, and searching in a large, discrete space of hypotheses can be challenging. In contrast, kernel-based learning methods are well-suited to deal with noisy training data, and learning reduces to a much simpler convex optimization problem for which globally optimal solutions can be found. It is possible to apply kernel methods to relational data by manually defining a kernel function for relational instances and using this function in standard kernel-based learners. However, this approach lacks the flexibility of an ILP-style hypothesis search, as relevant relational features have to be encoded *a priori* in the kernel function, rather than being discovered automatically from data. It also limits the amount of domain insight obtainable from the learned model, as no new relational features are induced during learning.

kFOIL integrates the clause search of ILP approaches with kernel methods, by learning a set of interpretable first-order clauses from data that define a relational kernel function. Thus, ILP methods are used as a form of structure learning to induce a suitable kernel function from data. This approach has the appealing potential of combining the advantages of both approaches, namely the flexibility and interpretability of ILP-style clause search and the robustness of kernel-based learners.

The simplest way to introduce a relational kernel function $k(x_1, x_2)$ based on a set $H = \{c_1, ..., c_n\}$ of first-order clauses is to propositionalize the examples x_1 and x_2 using H and then employ existing kernels on the resulting vectors, which we will refer to as the *feature space representation* of the examples. We will thus map each example x onto a vector $\phi(x)$ over $\{0, 1\}^n$ with $\phi(x)_i = 1$ if the i-th clause $c_i \in H$ covers the example x, and $\phi(x)_i = 0$ otherwise. Figure 1 shows an example of the propositionalizing function $\phi(x)$ and feature space representation of examples $\mathrm{m261}$ and $\mathrm{m1636}$ in the HIV resistance domain. Each mutant is mapped to a vector with one entry for each clause, evaluating to one if the clause fires on the example (i.e. the clause and the background knowledge logically entail it) and zero otherwise. A linear kernel in this representation amounts to counting the number of rules firing on both examples. For details on the logic representation employed see Section 6.1.2.

kFOIL integrates ILP and kernel-based learning by solving an integrated optimization problem given by

$$\max_{H \in \mathcal{H}} \max_{f \in \mathcal{F}_H} S(f, \mathcal{D}, \mathcal{B}). \tag{1}$$

Here, $\mathcal{H}$ denotes the logical hypothesis space under consideration, i.e. the set of all possible sets of clauses. $\mathcal{H}$ is defined using language bias declarations similar to those employed in FOIL and other ILP

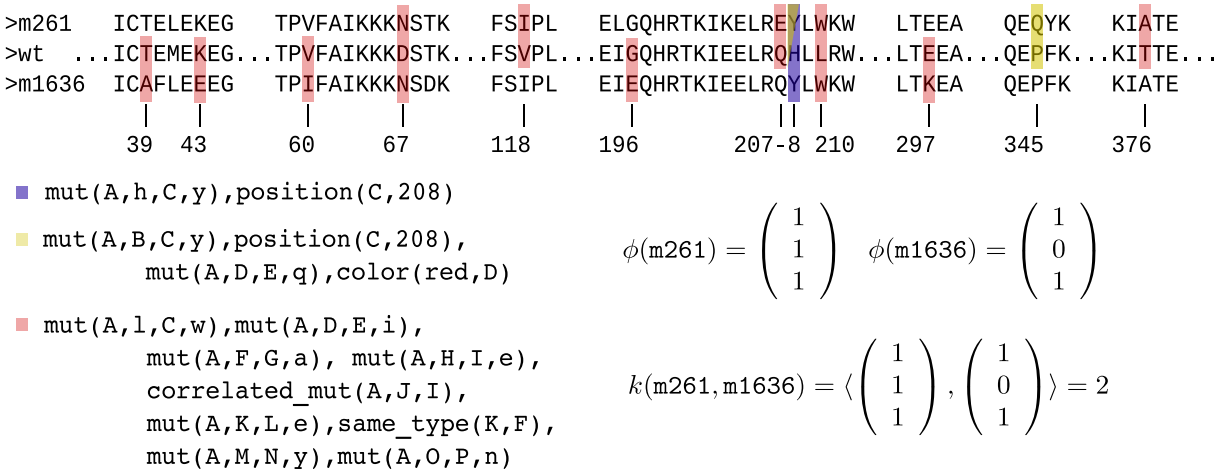


Figure 1. Example of a relational kernel function for the HIV resistance mutation database. An alignment between the wild type and two mutants is reported with colors highlighting positions which jointly satisfy the corresponding clause. The resulting feature space representation and mutants similarity for a linear kernel are reported in the lower right part.

algorithms. $\mathcal{F}_H$ denotes the model space of a kernel machine working on the feature space representation given by H. S denotes a scoring function that measures the predictive performance of f on the training data $\mathcal{D}$, and $\mathcal{B}$ the available logical background knowledge.

Note that Problem (1) consist of jointly optimizing the logical hypothesis H defining the feature space representation, and the function $f(x; H, \mathcal{B})$ implemented by a kernel machine working on this representation. In the following, we will refer to the outer optimization problem as *hypothesis learning* and the inner optimization problem as *function learning*. As discussed above, hypothesis learning implies searching in a discrete space of candidates, which is a complex task. Thus, heuristic strategies will be employed. In contrast, function learning takes place in a continuous space, for which principled search techniques are available. It is thus unclear whether scoring functions employed for function learning are also suitable for hypothesis learning. In fact, statistical relational learning systems often employ different scoring functions for learning the logical model structure and the statistical part of the model. Problem (1) should therefore be generalized to the following formulation:

$$\max_{H \in \mathcal{H}} S_O \left(\operatorname*{argmax}_{f \in \mathcal{F}_H} S_I(f, \mathcal{D}, \mathcal{B}), \mathcal{D}, \mathcal{B} \right) \tag{2}$$

where S_O and S_I are the scoring functions used for hypothesis and function learning respectively (see [25] for a more detailed discussion).

To solve the outer optimization problem (that is, learn the hypothesis H), kFOIL follows the well-known FOIL algorithm [31] for learning a set of clauses from data. The search procedure is sketched in Algorithm 1. In an outer loop, kFOIL repeatedly searches for clauses that score well with respect to the data set and the current hypothesis, and adds them to the current hypothesis. The initial hypothesis H_0 is the empty set in the standard algorithm, but it will be a partial model when the procedure will be used in the hierarchical version of the algorithm described in Section 3. In the inner loop, kFOIL greedily searches for a clause that scores well. To this aim, it employs a general-to-specific hill-climbing

Algorithm 1 kFOIL algorithm.

procedure KFOIL($H_0, \mathcal{D}, \mathcal{B}$)
  Initialize $H \leftarrow H_0$
  repeat
    Initialize $c := p(X_1, \cdots, X_n) \leftarrow$
    repeat
      $c := \text{argmax}_{c' \in \rho(c)} S(H \cup \{c'\}, \mathcal{D}, \mathcal{B})$
    until stopping criterion
    $H := H \cup \{c\}$
  until stopping criterion
  return H
end procedure

search strategy. Let p/n denote the predicate that is being learned. Then the most general clause, which succeeds on all examples, is "$p(X_1, ..., X_n) \leftarrow$". This clause serves as a starting point for the hill-climbing search. The set of all refinements of a clause c within the language bias is produced by a *refinement operator* $\rho(c)$. Clauses are greedily refined until a stopping criterion is met, and the highest-scoring clause encountered during the search is added to the hypothesis H. The joint optimization of the hypothesis H and kernel machine f reflected in Equation (2) and Algorithm 1 means that kFOIL falls into the framework of *dynamic propositionalization* [23]. This is in contrast to *static propositionalization* approaches that decouple propositionalization of the data and propositional learning.

Several scoring functions $S(H \cup \{c'\}, \mathcal{D})$ for candidate hypotheses have been proposed. One class of scoring functions uses the performance of a kernel machine trained on the feature space representation of the data $\phi(D)$ obtained from $H \cup \{c'\}$, by setting

$$S(H, \mathcal{D}, \mathcal{B}) = S_O \left(\underset{f \in \mathcal{F}_H}{\text{argmax}} \, S_I(f, \mathcal{D}, \mathcal{B}), \mathcal{D}, \mathcal{B} \right). \tag{3}$$

These measures require that the statistical learner is trained for each candidate clause, and its performance on the training set is reported. An efficient alternative consist of using kernel target alignment (KTA) [9], defined as:

$$S(H, \mathcal{D}, \mathcal{B}) = \frac{\langle K, yy^T \rangle_F}{\sqrt{\langle K, K \rangle_F \langle yy^T, yy^T \rangle_F}} \tag{4}$$

where K is the kernel matrix resulting from H, $y \in \{-1, 1\}^m$ is the target vector for m examples, y^T is the transpose of y, and the Frobenius product is defined as $\langle M, N \rangle_F = \sum_{ij} M_{ij} N_{ij}$. Intuitively, the alignment measures how the kernel adheres to a "perfect" kernel, scoring one and minus one for examples belonging to the same and different classes respectively, and is thus an indication of the performance a kernel machine can reach using it.

As a stopping criterion, the original FOIL algorithm stops when it fails to find a clause that covers additional positive examples. As an equally simple stopping criterion, learning in kFOIL is stopped when there is no improvement in score between two successive iterations.

2.3. Multitask kFOIL

The original kFOIL algorithm has been recently extended to a multitask setting [25]. Multitask learning in kFOIL is based on sharing the learned feature representation (or, equivalently, the relational kernel function) across tasks. This can be achieved by learning a single joint set of clauses for all tasks under consideration, such that all task-specific kernel machines trained on this representation achieve good performance in their respective prediction tasks. In a multitask setting, Algorithm 1 is thus adapted by replacing the single-task scoring function $S(H, \mathcal{D}, \mathcal{B})$ by an appropriate multitask scoring function, which is obtained as a combination of single-task scoring functions on the individual tasks. Assume that $S(H, \mathcal{D}, \mathcal{B})$ is an (outer) scoring function as given by Equation (3) or Equation (4), and that $\mathcal{D}_1, ..., \mathcal{D}_M$ are the available training data for M tasks $T_1, ..., T_M$. A simple but effective multitask scoring function is obtained by averaging single-task scores. That is, we replace Equation (3) with

$$S(H, \mathcal{D}, \mathcal{B}) = \frac{1}{M} \sum_{t=1}^{M} S_O \left(\operatorname*{argmax}_{f \in \mathcal{F}_H} S_I(f, \mathcal{D}_t, \mathcal{B}), \mathcal{D}_t, \mathcal{B} \right), \tag{5}$$

or equivalently, replace Equation (4) with

$$S(H, \mathcal{D}, \mathcal{B}) = \frac{1}{M} \sum_{t=1}^{M} \frac{\langle K, y_t y_t^T \rangle_F}{\sqrt{\langle K, K \rangle_F \langle y_t y_t^T, y_t y_t^T \rangle}} \tag{6}$$

where y_t is the label vector of task $t \in 1, ..., M$.

Experimental results presented in [25] show several advantages of multitask learning in the proposed setting. First, generalization performance is improved in some cases, confirming the advantages of multitask learning observed in the literature [6]. Second, in our setting learning a shared clause set for multiple tasks leads to a more compact representation of the learned concept, as the multitask clause set is significantly smaller than the union of the task-specific clause sets. In terms of the learned similarity/kernel function, this yields a generic definition of similarity that is shared between tasks and should be easier to interpret than a number of task-specific similarity functions. Finally, multitask learning also results in significant computational savings.

3. Hierarchical kFOIL

Learning a representation which is common to multiple related tasks is a way to reduce the risk of overfitting the data [2]. However, it prevents the algorithm to learn specific task-dependent features, which can be harmful for some of the other tasks. Furthermore, it assumes a high relatedness among tasks, and performances can be badly affected when relatedness is not that high. In this work we propose a simple extension to the multitask kFOIL learning algorithm dealing with this problem. A hierarchical approach to multitask learning is taken: the algorithm first learns a representation which is common to all tasks by using the multitask kFOIL; then such initial representation is refined separately for each task, leading to task-dependent final representations obtained as extensions of a common core. Algorithm 2 shows the resulting hierarchical kFOIL learning system, where $\{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$ are the datasets for the k different tasks. For simplicity we assumed a common background knowledge $\mathcal{B}$, but it is straightforward to replace it with task-specific background knowledge, as well as a task-specific language bias defining the clause refinement operator ρ.

Algorithm 2 Hierarchical kFOIL algorithm.

procedure HIERARCHICALKFOIL($\{\mathcal{D}_1, \ldots, \mathcal{D}_k\}, \mathcal{B}$)
  Initialize $H_0 \leftarrow$ KFOIL($\emptyset, \{\mathcal{D}_1, \ldots, \mathcal{D}_k\}, \mathcal{B}$) ▷ compute initial representation
  for all $\mathcal{D}_i \in \{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$ **do**
    $H_i \leftarrow$ KFOILREFINE($H_0, \mathcal{D}_i, \mathcal{B}$) ▷ compute task-dependent refinements
  end for
  return $\{H_1, \ldots, H_k\}$
end procedure

Algorithm 3 kFOIL refinement algorithm.

procedure KFOILREFINE($H_0, \mathcal{D}, \mathcal{B}$)
  Initialize $H \leftarrow H_0$
  for all $c \in H_0$ **do** ▷ refinement of existing clauses
    $H \leftarrow H \setminus \{c\}$
    repeat
      $c \leftarrow \mathrm{argmax}_{c' \in \rho(c)} S(H \cup \{c'\}, \mathcal{D}, \mathcal{B})$
    until stopping criterion
    if score improvement **then**
      $H \leftarrow H \cup \{c\}$
    end if
  end for
  $H \leftarrow$ KFOIL($H, \mathcal{D}, \mathcal{B}$) ▷ search for novel clauses
  return H
end procedure

The refinement stage is described in Algorithm 3 and consists of two steps. First, each clause from the initial representation is further refined guided by the task-specific score. If the (possibly) refined clause fails to improve the score, it is not added to the task-specific model. Then, the model from the previous step is enlarged by creating novel clauses using the plain kFOIL procedure. In principle, when refining general clauses, specializing the initial representation is not the only available option. We also implemented a search operator that considers both specializations and generalizations of the current clause, by greedily adding or removing a single literal. As this did not change results significantly in our experimental settings, in this paper we only report results obtained with the specialization operator.

A detailed analysis of computational complexity for both single task and multitask kFOIL is reported in [25], showing the increase in efficiency of the latter especially when KTA scoring is employed. The additional complexity of the refinement stage depends on the number of single-task clauses added: the more related the tasks are, the more the multitask clauses will already explain them and the refinement size will be limited.

The hierarchical kFOIL algorithm can be easily generalized to multiple levels of refinements whenever tasks can be naturally aggregated into a hierarchical structure. When the existence of a hierarchy can be guessed but its structure is unknown, the approach can be combined with a task-clustering algorithm as suggested by Thrun and O'Sullivan [38].

4. Task Clustering

The rationale of multitask learning is that predictive performance on a certain task should improve if information from related tasks can be exploited. However, performance worsening can also be experienced when trying to transfer information between unrelated tasks. A *selective* transfer [38] approach would be advisable. In such approach the transfer occurs only among tasks related one to each other. This requirement can be combined with the potential advantages of a multi-level representation by pursuing a hierarchical clustering approach. We basically adapt the task-clustering algorithm of Thrun and O'Sullivan [38] to our setting in which multitask learning is seen as a way to improve performance among training tasks rather than on novel test ones, and the clustering structure is used to learn interpretable hierarchical models and discover hierarchical rules relating tasks and groups of tasks.

Most clustering approaches rely on a measure of similarity (or distance) among instances. Intuitively, two tasks are similar if they have the same output over the same (or similar) examples. Whenever few (or no) examples are shared between tasks, we cannot just rely on them in order to compute a reliable measure of similarity. However, we can simply fill missing labels using predicted ones. Assume $\mathcal{D}$ is the overall set of training instances, and $\mathcal{D}_i$ its subset having labels for task i. We compute missing task-dependent labels for $\mathcal{D} \setminus \mathcal{D}_i$ as those predicted by a model for task i trained on $\mathcal{D}_i$. Repeating the procedure for all tasks, we compute pairwise task similarity as:

$$sim(i,j) = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \delta(\hat{f}_i(x), \hat{f}_j(x)) \tag{7}$$

where $\hat{f}_k(x)$ is the true task-specific label y_k for x if available, or the predicted one $f_k(x)$ otherwise. Learning a model for each task requires providing a representation for instances. We rely on the initial common representation H_0 (see Algorithm 2) and use it to derive task-dependent models, pairwise similarities and resulting hierarchical clustering. We employed an agglomerative hierarchical clustering approach with average pairwise similarity between elements for cluster similarity. Once the clustering structure is available, a multi-level refinement can be applied. Note however that the amount of transfer and the granularity of the refinement structure can be adaptively adjusted to the problem at hand. At each node in the clustering structure, two choices are possible, depending on the similarity between the node children: 1) learn a node-dependent model, and refine it for each child; this would be the default strategy to achieve a fully hierarchical model; 2) learn a node-dependent model, and use it within each child; this option would reduce the granularity of the hierarchical structure by merging together nodes containing similar tasks. The choice between these two options can be made separately for each choice point, by specifying a similarity threshold θ_{max}, or jointly by constraining the size of the representational structure (e.g. three levels from the root to the task-specific models). Algorithm 4 reports the pseudo-code of the hierarchical kFOIL procedure with task clustering. The initial representation H_0 obtained by running the plain multitask kFOIL on all tasks is used to infer a hierarchical clustering rooted at C_0. The representation H_0 is then refined on a per-node basis for each child of C_0, using the subset of $\mathcal{D}$ containing examples for tasks in C_i. $\text{SIM}(C_0)$ returns the similarity between the children of C_0, needed to decide on the amount of transfer. Algorithm 5 describes the node-dependent refinement for a generic node C. If the similarity value s between the node and its siblings is below θ_{max}, or the node contains a single task, a node-refined representation is generated calling the kFOIL refinement algorithm, implementing case 1 of node choices previously described. Otherwise, no node-wise refinement is performed (case 2) and the

Algorithm 4 Hierarchical clustering kFOIL algorithm.

procedure HIERARCHICALCLUSTKFOIL($\mathcal{D}, \mathcal{B}$)
  Initialize $H_0 \leftarrow$ KFOIL($\emptyset, \mathcal{D}, \mathcal{B}$) ▷ compute initial representation
  $C_0 \leftarrow$ HIERARCHICALCLUSTERING($H_0, \mathcal{D}, \mathcal{B}$) ▷ compute clustering
  for all C_i children of C_0 **do** ▷ compute node-dependent refinements
    $\mathcal{D}_i \leftarrow$ subset of $\mathcal{D}$ involving tasks in C_i
    CLUSTERNODEKFOIL(H_0, C_i, SIM(C_0), $\mathcal{D}_i, \mathcal{B}$)
  end for
end procedure

Algorithm 5 kFOIL algorithm for cluster node.

procedure CLUSTERNODEKFOIL($H_0, C, s, \mathcal{D}, \mathcal{B}$)
  if $s \leq \theta_{max} \vee$ ISLEAF(C) **then**
    $H \leftarrow$ KFOILREFINE($H_0, \mathcal{D}, \mathcal{B}$)
  else
    $H \leftarrow H_0$
  end if
  for all C_i children of C **do**
    $\mathcal{D}_i \leftarrow$ subset of $\mathcal{D}$ involving tasks in C_i
    CLUSTERNODEKFOIL(H, C_i, SIM(C), $\mathcal{D}_i, \mathcal{B}$)
  end for
end procedure

parent model is directly passed to the children.

5. Related Work

Multitask learning [6] is an active research area and various techniques have been proposed in the literature. The underlying idea is that of introducing a form of parameter sharing among tasks. In feed-forward neural networks [6], this is achieved by learning a common hidden layer for the tasks, while in kernel machines a matrix encoding the tasks relationship is included in the regularization term [14]. In a fully Bayesian framework, hierarchical Bayesian models [2] introduce a common prior distribution for the task-specific models, favouring parameter sharing among related tasks. These models allow for a great flexibility in modeling different levels of tasks relatedness. Bakker and Heskes [1] for instance, generalize standard multitask feed-forward neural networks with prior distributions over the output layer weights. In the task gating approach, they allow for both task clustering, by using a mixture model for the prior distribution, and task-specific mixing proportions for the clustering. However, the number of clusters needs to be pre-specified according to the available domain knowledge, or determined by model selection strategies. Non-parametric Bayesian approaches have gained increasing popularity in recent years as they allow to overcome this problem, by sampling distributions over parameters rather than parameters directly. Dirichlet processes [15] have been employed for multitask learning [43] in order to automatically select the appropriate number of clusters for task-dependent parameters. Approaches to learn an entire latent hierarchy of tasks have been also proposed [29, 19].

These approaches, however, assume that task models can be eventually represented as parameter vectors. The setting considered in this paper is rather different, as we are learning hybrid statistical-logical models made up of first-order clauses. It would be quite challenging to devise prior distributions for such discrete structures. Our approach exploits multitask information for learning a general-to-specific hierarchy of logical hypotheses, represented as relational kernel functions, to be coupled with appropriate task-specific weights. To the best of the authors' knowledge, multitask kFOIL was the first attempt to address multitask kernel learning in a statistical relational learning context. Indeed, multitask structure learning itself has received little attention in the statistical relational learning setting. A notable exception is the work by Deshpande et al. [12] on learning multitask probabilistic relational rule sets. Their approach assumes that the prior information shared among tasks is a set of rule *prototypes*. Task-specific rules are derived from these prototypes by a generative probabilistic process involving prototype selection and modification steps or rule generation from scratch. From the point of view of learning hierarchies of concepts in relational domains, the works on infinite relational models [42, 20] are also worth mentioning. Here latent indicator variables on entities or concepts are introduced on top of the relational structure, allowing to cluster them using Dirichlet process priors. The approach has also been extended [36] to discover a hierarchical structure of concepts. While these models do not perform structure learning in terms of logical hypotheses, their ability to infer arbitrary clustering structures is an appealing feature to be integrated in our approach.

In the ILP community, multitask learning has been tackled in the form of learning several related concepts simultaneously. Different approaches have been pursued. Related to our approach is the work by [32], where the assessment of candidate clauses on the primary task is augmented with the performance of similar rules on a secondary task. Furthermore, a scenario resembling multitask learning has been studied in [10], where (sub)structures of concepts already learned are used as building blocks when learning a new concept. A further related scenario is that of *repeat learning* and *multiple predicate learning* [21, 11], where an ILP learner has to discover a series of related concepts drawn from some (initially unknown) distribution. Moreover, *predictive clustering trees* have been used in an ILP setting. These trees can be used in a multitask setting, where predictions for several tasks are made at every leaf [4, 5].

6. Experimental Evaluation

We evaluated our hierarchical approach on datasets from two domains, namely HIV resistance mutations and SCOP protein structure classification hierarchy. In all experiments the KTA (see Equation 4) was used as scoring function for guiding the search of the kFOIL algorithm, as it is more efficient even if less effective [25] in general than measures based on a trained kernel machine. A simple linear kernel was employed in order to maximize the understandability of the learned models. Note that we are not interested in pushing the performance of the learning algorithm by fine tuning its parameters but rather comparing the respective advantages of the different approaches and evaluate their explanatory power. The Hierarchical kFOIL program is freely available online [1]. The package also includes all the prediction results and learned models of the experiments reported in the following sections.

[1] `http://www.disi.unitn.it/~passerini/software/HkFOIL.tgz`

6.1. Predicting Drug-Resistance of Mutants

Viruses are characterized by a very high mutation rate, which allows them to quickly develop drug-resistant strains. A single- or multiple-point mutation can confer the mutant resistance to one or more drugs, for instance by modifying the inhibitor target site on the protein. Predicting the drug-resistance of mutants can be of valuable help in designing more effective drugs, especially if interpretable models can be provided. This can be addressed as a multitask learning problem, where each drug is a single task. However, the relationship between tasks is not necessarily strong as different drugs can target different sites in the protein. A hierarchical approach seems a natural candidate in this setting. We focused on HIV, both for the impact of the virus and the availability of annotated databases of mutants.

6.1.1. The HIV Resistance Mutations Datasets

We experimented on a dataset of mutations from the Los Alamos National Laboratories (LANL) HIV resistance database[2]. The dataset was derived in [35] and is composed of 2,339 mutants of the HIV reverse transcriptase (RT). RT is a DNA polymerase enzyme that transcribes RNA into DNA, allowing it to be integrated into the genome of the host cell and replicated along with it, and is thus crucial for virus propagation. Richter et al. [35] formulated the learning problem as a mining task and applied a relational association rule miner to derive rules relating different mutations and their resistance properties. We take a slightly different approach here and provide supervision at the mutant rather than mutation level. A mutant is considered resistant to a drug if it contains at least one observed resistance mutation to that drug. We derived two different versions of the dataset, considering resistance to drug classes or specific drugs respectively.

drug class resistance dataset We selected the three classes of drugs: (a) NonNucleoside RT Inhibitors (NNRTI); (b) NonCompetitive RT inhibitors (NCRTI); (c) Pyrophosphate Analogue RT Inhibitors (PARTI). In the dataset 1081 mutants are labelled as resistant to NNRTI, 75 to NCRTI and 53 to PARTI. We ignored the Nucleoside RT Inhibitors (NRTI) since all the mutants in this dataset had at least one mutation conferring resistance to that class of drugs.

drug resistance dataset We used all the four classes of drugs from the original dataset, thus including the NRTI class. We specialized the labeling on a single-drug basis, extracting this information directly from the LANL HIV resistance database when available. Table 1 reports the drugs belonging to the four inhibitor classes and the number of mutants resistant to each of them. In this way we deepen the hierarchy at the single drug level with the aim of testing the usefulness of our hierarchical clustering approach.

6.1.2. Background Knowledge

We built a relational knowledge base for the domain at hand. Table 2 summarizes the predicates we included as a background knowledge. We represented the amino acids of the wild type with their positions in the primary sequence (`aa/2`) and the specific mutations characterizing them (`mut/4`). Target predicates were encoded as resistance of the mutant to a certain drug (`res_against/2`).

[2] `ttp://www.iv.lanl.gov/content/sequence/RESDB/`

Drug class	Drug	# mutants
NRTI	Zidovudine (azt)	2211
	Lamivudine (3tc)	1356
	Abacavir (abc)	422
	Zalcitabine (ddc)	336
	Didanosine (ddi)	280
NNRTI	Efavirenz (efv)	883
	Nevirapine (nvp)	833
	Delavirdine (dlv)	796
	ADAMII	114
	Trovirdine	113
NCRTI	MSK-076	75
PARTI	foscarnet	53

Table 1. Hierarchical task structure for the HIV resistance mutation datasets with associated number of instances.

Background Knowledge Predicates	
`aa(Pos,AA)`	indicates a residue in the wild type sequence
`mut(Mutant,AA,Pos,AA1)`	indicates a mutation: mutant identifier, position and amino acids involved, before and after the substitution
`res_against(Mutant,Drug)`	indicates whether a mutant is resistant to a certain drug
`color(Color,AA)`	indicates the type of a natural amino acid
`same_type(R1,R2)`	indicates whether two residues are of the same type
`same_type_mut(Mutant,Pos)`	indicates a mutation to an amino acid from the same type
`different_type_mut(Mutant,Pos)`	indicates a mutation changing the type of residue
`correlated_mut(Mutant,Pos1,Pos2)`	indicates whether two mutations are correlated (see the text for the details)

Table 2. Summary of the HIV drug resistance background knowledge facts and rules.

Color Class	Amino Acids	Description
red	AVFPMILW	small and/or hydrophobic and/or aromatic
blue	DE	acidic
magenta	RK	basic
green	STYHCNGQ	hydroxyl and/or polar and/or basic

Table 3. Amino acid types encoded in color classes.

Additional background knowledge was included in order to highlight characteristics of residues and relationships between mutations:

`color/2` indicates the type of the natural amino acids according to the coloring proposed in [37] and also reported in Table 3. For example the magenta class includes basic amino acids as lysine and arginine while the blue class includes acidic amino acids as aspartic and glutamic acids.

`same_type/2` indicates whether two residues belong to the same type, i.e. a change from one residue to the other conserves the type of the amino acid.

`same_type_mut/2` indicates that a residue substitution at a certain position does not modify the amino acid type with respect to the wild type. For example mutation d123e conserves the amino acid type while mutation d123a does not (i.e. `different_type_mut/2` holds for it).

`correlated_mut/3` states that two mutations are potentially correlated. We considered two mutations in different positions along the primary sequence as correlated, if they compensate reciprocally for the substitutions of the amino acids. This predicate captures simple cases like the two mutations d123a and a321d, and more complex correlations in which the changes involve not exactly the same residue but residues of the same type, like d123a and a321e or d123a and v321e. By simply looking at the characteristics of pairs of mutated positions we collect potential instances of compensatory mutations. These counterbalance other eventually destabilizing replacements in the protein structure, thus playing a key role in the emergence of drug resistance [16].

6.1.3. Hierarchical multitask experiments

We evaluated our hierarchical kFOIL algorithm on the drug class resistance dataset, with three tasks corresponding to the three drug classes (see Section 6.1.1). We compared the results with the two alternatives of: 1) considering three separate single task learning problems; 2) considering a single common multitask learning problem with no per-task refinement. We performed 3-fold cross-validation procedures stratified at the single-task level to ensure a good balancing between positive and negative examples for each learning task. The area under the ROC curve (AUC) was employed as measure of performance in all experiments.

We experimented with two variants of the language bias guiding the learner, by varying the constraints on the use of the `mut/4` predicate. In the first variant (V1) the learner can extend a clause by using the predicate `mut/4` with the position variable already instantiated, and thus scoring it according to the mutants that have a mutation in that position. In the second variant (V2), in the predicate

	V1			V2		
Drug class	single task	multitask	hierarchical	single task	multitask	hierarchical
NNRTI	0.95○	0.77	0.96○	0.68	0.66	0.71○●
NCRTI	0.95	0.99	0.99●	0.88	0.86	0.88
PARTI	0.81	0.95●	0.98●	0.65	0.84●	0.90●

Table 4. Summary of the hierarchical multitask experiments (kta scoring). Statistical tests for the significance of the differences in AUC were computed for the methods within each language bias variant using the two-tailed Hanley-McNeil test [17] (p=0.05). A bullet (●) indicates that the method is significantly better than the single task approach, while a circle (○) indicates a significant improvement over the multitask one. All other differences are not statistically significant.

`mut/4` the position variable is not instantiated while the variable corresponding to the mutated form of the residue is instantiated instead. In the first case the search space of the learner will contain predicates like `mut(Mutant,a,123,Rnew)` with a specific position instantiated, while in the second case it will contain predicates like `mut(Mutant,Rold,Position,k)` where k indicates a change resulting in a lysine. The rationale for considering the two variants is that the former will tend to learn more specific clauses involving relationships between point-wise mutations, as for the association rules discovered in [35]. Conversely, the latter variant will be biased to learn possibly suboptimal but more general and hopefully more interesting mutation rules, trying to discover higher level patterns relating different mutations.

Table 4 reports the results of the two variants V1 and V2. Different behaviours can be detected for different drug classes. Overall, multitask learning achieves comparable results with respect to a standard single task approach in both variants, being twice significantly better and once significantly worse than the alternative. The result suggests that for this dataset we are not always able to take a real advantage from the multitask learning approach, possibly because classes of drugs can be quite unrelated by targeting different binding sites. By adding a refinement stage on a per-task basis, we succeed in improving the results with respect to both single task and multitask approaches. Hierarchical kFOIL is never significantly worse than any of the two alternatives, while being significantly better than at least one of them in five out of six cases.

6.1.4. Discussion and Rule Interpretation

Concerning the language bias variants, we can observe from Table 4 that as expected the instantiation of a specific position (V1) gives better results compared to searching more general rules (V2). The former models can exploit the fact that mutants resistant to the same drug often share mutations in the same positions. These could be located in the protein binding site or its vicinity, but drug resistance could also be conferred more indirectly by other conformational modifications.

Figure 2 shows an example of hierarchy of learned clauses in the V1 variant setting. The root clause `mut(A,g,196,B),color(blue,B)` was learned in the multitask models of all folds. The clause states that a mutation in position 196 changing a glycine into an acidic residue (aspartic or glutamic acid) can be important for a mutant to develop resistance to the three kinds of drugs analysed. This could provide hints for understanding how the binding works and is affected by the surrounding residues. Moreover it underlines the potential of the approach also in other contexts: for example to gain insights on the wild type protein function and on its active site starting from its mutants usually obtained by random

mutagenesis. We are currently pursuing such research direction on an amidase. When inspecting the models resulting from the single task refinements, we found that for the NNRTI task the above-mentioned clause is not extended. Additional mutations are instead included in the extended clauses for the NCRTI and the PARTI tasks. This could suggest that in those cases the drug-specific resistance results from the combination with one or two other mutations.

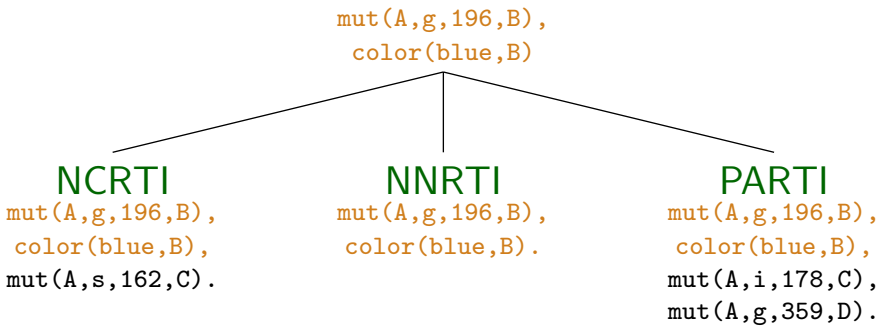


Figure 2. Example hierarchy of learned clauses.

Some mutations, like the mutation of the asparagine in position 348, appear in multitask model and in all the single task models (in all the folds) with the addition of at most one other mutation. This seems to suggest that such mutation is important for the mutant resistance to the three drug classes. Interestingly the refined model further enriches the corresponding clause by associating the mutation with up to three other mutations.

The learned rules generated by kFOIL for NNRTIs, NCRTIs and PARTIs can be compared with those reported by the rule miner used in [35] which are not explicitly linked to the resistance to NRTIs. These rules associate mutations in position 41 and 215, sometimes also with other mutations in position 277 and 293. kFOIL identified the association between the mutations at positions 215 and 41. This association was previously observed to be related to resistance to NRTIs [26]. kFOIL identified the same association for the resistance to the quite different other classes of inhibitors. In particular the two mutated positions were found among the rules for the resistance to PARTIs where additional mutated position are highlighted: 67 and 376.

The learned models in the variant V2 contain clauses like

```
mut(A,B,C,w),mut(A,D,E,i),mut(A,F,G,l),mut(A,H,I,d),mut(A,J,K,a)
```

or

```
mut(A,B,C,g),position(C,135),mut(A,D,E,m),
                    correlated_mut(A,E,F),position(F,138)
```

which combine a quite large set of mutations, with the latter clause including an explicitly correlated pair of mutations. As a further example the refined model on the PARTI task suggests, in all folds, the clause:

```
mut(A,B,C,w),different_type_mut(A,C)
```

which highlights a mutation into a tryptophan that completely changes the type of amino acid in the mutated position. Note that the need for multiple mutations in order to induce a change in the phenotype

has recently found confirmation [40] in experimental studies on molecular phenotypes. A suggested interpretation [40] states that "neutral mutations prepare the ground for later evolutionary adaptation". While this is far from being a confirmation for the specific patterns found by our algorithm, the obvious limitation of learning techniques focusing on single point mutations alone is an additional stimulus for this research direction.

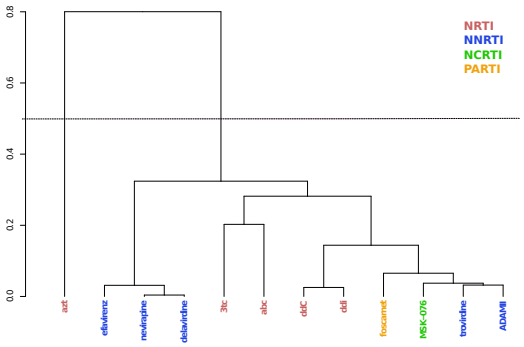
6.1.5. Hierarchical clustering experiments

We evaluated our task clustering approach for hierarchical multitask learning on the drug resistance dataset (see Section 6.1.1) which provides a deeper structure of tasks as information on both drug classes and specific drugs are available. We compared the results with the three alternatives of: 1) considering three separate single task learning problems; 2) considering a single common multitask learning problem with no per-task refinement; 3) learning a per-task refinement of an initial common multitask model as in the previous section. In order to feed the clustering algorithm with an accurate estimate of tasks similarities, we focused on the variant of the language bias providing the better results (V1). The validation procedure is the same as for the hierarchical multitask experiments.

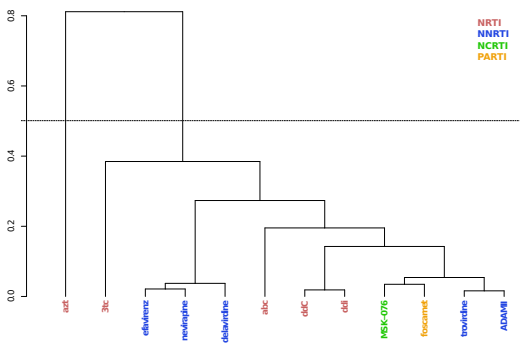
Figure 3 shows the dendrograms of the clusterings obtained in each one of the folds of the cross-validation. Drug classes are highlighted with different colors: red for drugs belonging to the NRTIs, blue for those belonging to NNRTIs, orange for PARTIs and green for NCRTIs. Note that the hierarchical clustering obtained is rather stable across folds. Interestingly, Zidovudine (azt) is consistently separated from the rest of the drugs and merged only at the root. A possible explanation for this behaviour is that the large number of examples available for the drug (see Table 1) makes the kernel machine employed in kFOIL more focused on the task and less predictive for different drugs. Indeed, multitask learning already achieves an AUC of 0.95 for azt (see Table 5), which is rather high considered that the resulting binary classification problem is the most balanced.

A dotted line in Figure 3 indicates the value of the similarity threshold employed ($\theta_{max} = 0.5$): a single intermediate level of refinement is obtained in all folds between the shared multitask model at the root and the per-task refinements at the leaves.

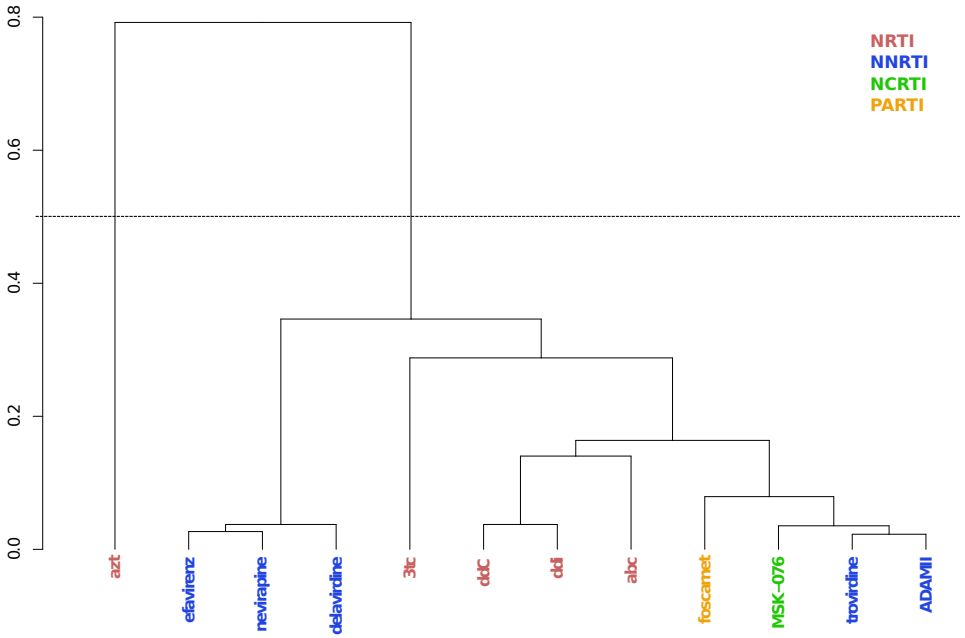
Table 5 reports AUC results for the different learning strategies for the V1 variant of the language bias. The problem of using a plain multitask approach is even more severe when considering single drugs from all classes, as the single task alternative is significantly better in six out of twelve cases and never significantly worse. Again, a per-task refinement of the multitask model allows to recover the single-task performance. However, only in one case (azt) the refinement is significantly better than the single-task alternative. This seems to indicate a poor overall contribution of task transfer, possibly because of the difference between target sites for the four inhibitor classes. Indeed, such inhibitors rely on quite different mechanisms. NNRTI and NCRTI inhibit the reverse transcriptase by binding to the enzyme active site, therefore directly interfering with the enzyme function. NRTI is instead incorporated into the newly synthesized viral DNA for preventing its elongation. Finally the PARTI targets the pyrophosphate binding site and it is employed, as part of a salvage therapy, on patients in which the HIV infection shows resistance to the other classes of antiretroviral drugs. A further refinement level guided by the clustering procedure allows to increase to four the number of cases in which a hierarchical approach is significantly better than the single-task alternative. Furthermore, in one of these cases the approach is significantly better than the shallow hierarchical multitask model too. Note that in one case (foscarnet) the hierarchical clustering is actually significantly worse than the shallow approach. However, in this case a plain single



(a) fold 0



(b) fold 1



(c) fold 2

Figure 3. Hierarchical clustering dendrograms. Specific drugs belonging to the same class are colored respectively in red for NRTI, blue for NNRTI, orange for PARTI, green for NCRTI.

task model is significantly better than both hierarchical versions[3]. This drug has very few examples (53),

[3]Note that the multitask approach for foscarnet is not significantly better than the hierarchical ones even if its AUC value is the

Drug Class	Drug	multitask	single task	hierarchical multitask	hierarchical clustering
NRTI	azt	0.95	0.93	0.97●○	0.97●○
	3tc	0.76	0.92○	0.91○	0.94●○◇
	abc	0.92	0.95○	0.96○	0.95○
	ddC	0.94	0.92	0.94	0.94●
	ddi	0.95	0.94	0.94	0.95
NNRTI	efavirenz	0.93	0.94○	0.95○	0.96●○
	nevirapine	0.92	0.95○	0.96○	0.96○
	delavirdine	0.93	0.96○	0.97○	0.97○
	ADAMII	0.87	0.90	0.91	0.90
	trovirdine	0.91	0.96○	0.96○	0.97○
NCRTI	MSK-076	0.99	0.98	0.97	0.99
PARTI	foscarnet	0.88	0.88◇⋆	0.86⋆	0.84

Table 5. Summary of the hierarchical clustering experiments (kta scoring) with the V1 version of the language bias. AUC values are reported, together to the results of the statistical tests for the significance of AUC differences computed using the two-tailed Hanley-McNeil test [17] (p=0.05). A symbol after an AUC value indicates that the corresponding method is significantly better than: single task (●), multitask (○), hierarchical multitask (◇), hierarchical clustering (⋆). All other differences are not statistically significant.

indicating a possible sub-optimality of the hierarchical versions when insufficient data are available.

6.1.6. Discussion and Rule Interpretation

By mining the learned rules some interesting examples of hierarchical concept learning can be found: the clause `mut(A,w,88,B)` learned in the multitask setting is specialized at the first refinement stage into the clause `mut(A,w,88,B),mut(A,C,D,E),color(blue,C),color(green,E)`, in which an additional rather high-level mutation description is included. In the second refinement stage, the per-drug one, the clause is further specialized depending on the specific drug under consideration. In the case of Lamivudine (3tc), for instance, the final clause is:

`mut(A,w,88,B),mut(A,C,D,E),color(blue,C),color(green,E),mut(A,t,400,F)`

that is, a specific position is selected in combination with the previously characterized mutations.

Another interesting example is given by the clause `mut(A,t,215,y),mut(A,g,196,B)`, which is initially refined into `mut(A,t,215,y),mut(A,g,196,B),mut(A,t,357,C)`. When refining on a single task basis for the NRTI drug Abacavir (abc), for instance, the previously observed [41] the association of mutations in positions 215 and 41 is recovered:

`mut(A,t,215,y),mut(A,g,196,B),mut(A,t,357,C),mut(A,m,41,D),mut(A,i,293,E).`

The association of mutations in position 41, 215 and 293 was also highlighted by the relational rule miner used in [35]. Here we are able to be a bit more specific and relate the association to resistance to a particular drug (abc).

same as the single-task one. This is due to the fact that in the latter case predictions are highly related to the hierarchical ones, and the Hanley-McNeil test [17] considers such relatedness in computing the critical ratio.

All-α	N	All-β	N	α/β	N	$\alpha+\beta$	N
DNA 3-helical	30	Ig beta-sandwich	45	β/α (TIM)-barrel	55	Ferredoxin-like	26
EF hand-like	14	Tryp ser proteases	21	Rossmann-fold	21	Zincin-like	13
Globin-like	13	OB-fold	20	P-loop	14	SH2-like	13
4-Helical cytokines	10	SH3-like barrel	16	Periplasmic II	13	beta-Grasp	12
Lambda repressor	10	Lipocalins	14	α/β-Hydrolases	12	Interleukin	9

Table 6. Number of examples (N) in each of the folds in the SCOP dataset. Folds are grouped by fold classes: All-α, All-β, α/β and $\alpha+\beta$.

Among the kFOIL generated rules many surveillance mutations [3] indicated for the resistance to NRTI and NNRTI can be found. Those mutations often appear with other mutations that potentially participate in conferring the resistance to a class of inhibitors or to a specific inhibitor. For instance, the mutations in position 184 that were shown to be involved in 3tc resistance appear jointly with a mutation in position 69. In abc resistance the same 184 mutated position appears with a mutation in position 10 in the generated model.

6.2. Protein structure classification

After translation from mRNA, the linear chain of amino acids (*primary structure*) composing a protein folds in the three-dimensional space assuming a specific native conformation (*tertiary structure*). Large regularities can be observed in these conformations, from the local arrangements into *secondary structure* elements, α-helices and β-strands, to their aggregation into domains, protein subunits characterized by a semi-independent evolution and function with respect to the rest of the protein structure.

A number of hierarchies of protein three-dimensional structures have been created, based on evolutionary and/or structural considerations. A protein structure classification task in this setting consists of automatically assigning a protein structure to the correct class, relying on information like the arrangement of its secondary structure elements.

SCOP [18] is a manually curated hierarchy based on both structural and evolutionary relationships between protein domains. Turcotte et al. [39] extracted a dataset made of the five most populated folds of each of the four main classes (see Table 6). They learned a set of rules characterizing each of the folds with respect to the other folds of the same class, in a binary classification setting. The problem was later generalized [8] to full multiclass classification at the fold-class level. We consider this last setting in our experiments.

6.2.1. Background Knowledge

We used the background knowledge encoded in [39] for representing the three-dimensional structure information of the protein domains. Table 7 reports the background knowledge predicates divided into three classes: global knowledge, which encodes global characteristics of the protein domain, namely, the number of residues and the number and type of secondary structure elements; local knowledge, which encodes local information of a single secondary structure element (SSE); relational knowledge, introducing relationships between secondary structure elements and their properties.

Background Knowledge Predicates	
global background knowledge	
`len_interval(Min,Domain,Max)`	indicates that the number of amino acids composing a Domain is between Min and Max
`nb_alpha_interval(Min,Domain,Max)`	indicates that the number of α-helices composing a Domain is between Min and Max
`nb_beta_interval(Min,Domain,Max)`	indicates that the number of β-strands composing a Domain is between Min and Max
local background knowledge	
`unit_len(SSE,Value)`	indicates the length of an SSE as `very_low`, `lo`, `hi`, `very_high`
`unit_aveh(SSE,Value)`	indicates the average hydrophobicity of an SSE as `very_low`, `lo`, `hi`, `very_high`
`unit_hmom(SSE,Value)`	indicates the hydrophobic moment of an SSE as `very_low`, `lo`, `hi`, `very_high`
`has_pro(SSE)`	indicates whether an SSE contains a proline
relational background knowledge	
`adjacent(Domain,SSE1,SSE2,N,TypeSSE1,TypeSSE2)`	indicates that the N-th (position along the chain) secondary structure element SSE1 of type TypeSSE1 is followed by SSE2 of type TypeSSE2. The type can be `h` for α-helices or `e` for β-strands. α-helices and β-strands are numbered separately
`coil(SSE1,SSE2,Len)`	indicates that there are Len residues between two SSEs

Table 7. Summary of the SCOP background knowledge.

Fold class	single task	multi task	hierarchical
All-α	0.66$\circ$	0.45	0.69$\circ\bullet$
All-β	0.78$\circ$	0.65	0.87$\circ\bullet$
α/β	0.55$\circ$	0.50	0.59$\circ\bullet$
$\alpha+\beta$	0.59$\circ$	0.46	0.67$\circ\bullet$

Table 8. Summary of the hierarchical multitask experiments for the SCOP dataset. Multiclass accuracies at the fold-class level are reported, averaged over 5 folds. The results of a two-tailed paired t-test for the significance of accuracy differences are also shown (p=0.05). A symbol after an accuracy value indicates that the corresponding method is significantly better than single task ($\bullet$) or multitask ($\circ$) respectively.

6.2.2. Hierarchical multitask experiments

Single task problems here consist of discriminating a fold against the other folds in the same fold class. A common multitask learning problem can be devised by jointly addressing all 20 single tasks. We compared the two alternatives with our hierarchical approach. We adhered to the experimental setting in [8], running a 5-fold cross validation procedure and reporting multiclass classification accuracies at the fold-class level.

Experimental results are summarized in Table 8. Single task learning is always significantly better than the multitask approach. This is rather expected in this setting, as different fold classes have quite different structural characteristics. This is a clear example where plain multitask learning is badly harmful because of the limited relationship between tasks. On the other hand, our hierarchical approach is always significantly better than both multitask and single task alternatives. That is, it succeeds in collecting the useful information coming from loosely related tasks, to be effectively refined on a per-task basis. Indeed, only a fraction of the multitask clauses are actually retained, or further specialized, in the refined models. Note that a deeper hierarchical structure could be conceived by learning a common representation at the fold-class level, to be further refined. This is a special case in which the multitask problem is actually a multiclass one. While single tasks are definitely related in this case, there is no increase in the training set size as all examples appear in all tasks. We experienced a performance degradation when including this additional level of the hierarchy. Our results are roughly comparable to those reported in [8]. We achieve better results on the two most populated fold classes, and worse on the other two. However, a sound comparison cannot be conducted because of the different learning setting. The set of rules employed in [8] were learned on the entire dataset, and only their weights were learned and evaluated with a cross-validation procedure.

6.2.3. Discussion and Rule Interpretation

Figure 4 highlights the different roles of global, relational and local background knowledge predicates (on the x axis) in our learned models. On the y axis we report the relative frequency of occurrence of each predicate among the ten best clauses learned in the different settings (multitask, single task and hierarchical).

As expected the multitask learning produces models mainly including global background knowledge predicates, especially those characterizing the number and type of secondary structure elements. Indeed these are the main features characterizing the different fold classes. In single task learning models and re-

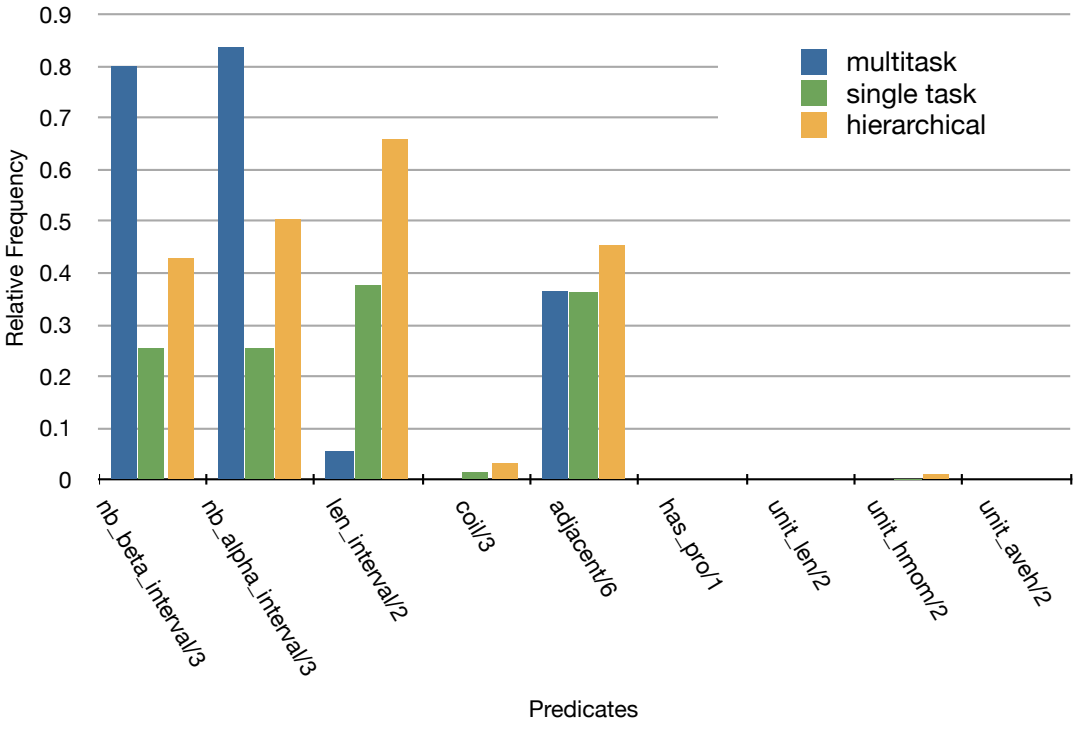


Figure 4. Relative frequency of the background knowledge predicates in the learned models.

fined models relational background knowledge predicates gain more relevance. The results also confirm the observation in [7] that local information, related to the hydrophobicity (unit_aveh/2 and unit_hmom/2) and the presence of a proline in the SSE (has_pro/1), has a quite marginal role.

To give a more detailed idea of the learned clauses interpretation, we report an example of hierarchical rule. At the root level, the multitask model includes the following clause:

```
nb_beta_interval(0,A,3),nb_alpha_interval(1,A,18).
```

The clause is refined at the single-task level both by adding global information on domain lengths and relational one for SSE pairs. Examples of the former type of refinement include the 4-Helical cytokines fold from the All-α class:

```
nb_beta_interval(0,A,3),nb_alpha_interval(1,A,18),len_interval(104,A,145).
```

and the Ferredoxin-like for the $\alpha + \beta$ one:

```
nb_beta_interval(0,A,3),nb_alpha_interval(1,A,18),len_interval(58,A,72).
```

Relational background knowledge is used with regard to β-strands in the Interleukin fold from the $\alpha + \beta$ class:

```
nb_beta_interval(0,A,3),nb_alpha_interval(1,A,18),adjacent(A,B,C,2,e,e).
```

and to α-helices in the EF-hand like fold (All-α).

```
nb_beta_interval(0,A,3),nb_alpha_interval(1,A,18),
                        nb_alpha_interval(3,A,5),adjacent(A,B,C,1,h,h).
```

Note that in this last clause the number of α-helices is also restricted to a range from three to five. The domain should then contain two consecutive helices at the beginning of the polypeptide chain. We report in Figure 5 an example of EF Hand-like protein structure (PDB code 1CNP) that respects the above rule. The two consecutive α-helices are highlighted in red.

7. Conclusion

We developed hierarchical kFOIL as a simple extension of the multitask kFOIL learning algorithm. The algorithm addresses the limitations of learning a single relational structure for multiple tasks, by taking a hierarchical approach and successively refining a common model on a per-task basis. The algorithm can be generalized to deeper levels of task structure, which can itself be learned during the process by a hierarchical task clustering approach. We applied the algorithm to problems of drug-resistance mutant prediction and protein structure classification, showing its advantage over both single and multi task alternatives. We stress here that a major advantage of the adopted strategy is the ability to provide explanations for the learned models which are themselves hierarchical: a subset of relational features relevant to all tasks can be identified together with more specific task-dependent ones.

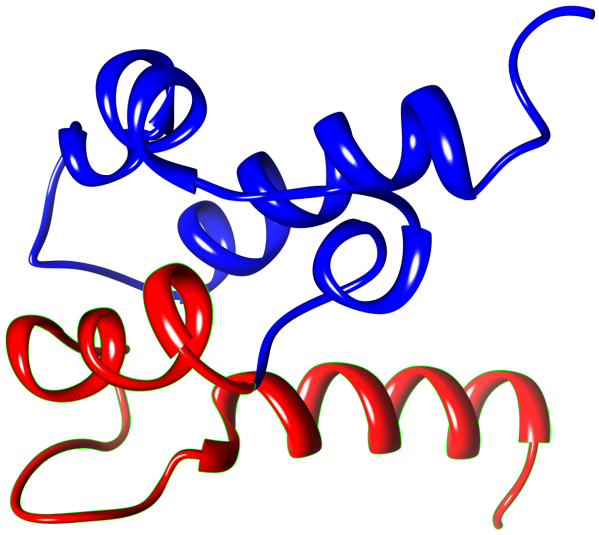


Figure 5. Example of EF Hand-like protein structure (PDB code 1CNP). The first two consecutive α-helices are highlighted in red.

Acknowledgment

We would like to thank Lothar Richter, Regina Augustin and Stefan Kramer for sharing the processed version of the HIV resistance mutation database. In this paper Figure 5 was produced using the UCSF Chimera package [30] from the Resource for Biocomputing, Visualization, and Informatics at the University of California, San Francisco (supported by NIH P41 RR-01081).

References

[1] Bakker, B., Heskes, T.: Task clustering and gating for bayesian multitask learning, *Journal of Machine Learning Research*, **4**, 2003, 83–99.

[2] Baxter, J.: A Bayesian/information theoretic model of learning to learn via multiple task sampling, *Machine Learning*, 1997, 7–39.

[3] Bennett, D. E., Camacho, R. J., Otelea, D., Kuritzkes, D. R., Fleury, H., Kiuchi, M., Heneine, W., Kantor, R., Jordan, M. R., Schapiro, J. M., Vandamme, A.-M., Sandstrom, P., Boucher, C. a. B., van de Vijver, D., Rhee, S.-Y., Liu, T. F., Pillay, D., Shafer, R. W.: Drug resistance mutations for surveillance of transmitted HIV-1 drug-resistance: 2009 update., *PloS one*, **4**(3), 2009, e4724.

[4] Blockeel, H., De Raedt, L., Ramon, J.: Top-down Induction of Clustering Trees, *Proceeding of the 15th International Conference on Machine Learning*, Madison, Wisconsin, USA, 1998.

[5] Blockeel, H., Dzeroski, S., Kompare, B., Kramer, S., Pfahringer, B., Laer, W.: Experiments In Predicting Biodegradability., *Applied Artificial Intelligence*, **18**(2), 2004, 157–181.

[6] Caruana, R.: Multitask Learning, *Machine Learning*, **28**(1), 1997, 41–75.

[7] Chen, J., Kelley, L., Muggleton, S., Sternberg, M.: Multi-class prediction using stochastic logic programs, *Inductive Logic Programming*, 2007, 109–124.

[8] Chen, J., Kelley, L., Muggleton, S., Sternberg, M.: Protein fold discovery using stochastic logic programs, in: *Probabilistic inductive logic programming* (L. De Raedt, P. Frasconi, K. Kersting, S. Muggleton, Eds.), Springer-Verlag, Berlin, Heidelberg, 2008, 244–262.

[9] Cristianini, N., Shawe-Taylor, J., Elisseef, A., Kandola, J.: On kernel-target alignment, *Proceedings of NIPS 14*, 2001.

[10] Datta, P., Kibler, D. F.: Concept Sharing: A Means to Improve Multi-Concept Learning, *Proceedings of the 10th International Conference on Machine Learning, Amherst, MA, USA*, 1993.

[11] De Raedt, L., Lavrac, N., Dzeroski, S.: Multiple Predicate Learning, *Proceedings of the 13th International Joint Conference on Artificial Intelligence, Chambery, France*, 1993.

[12] Deshpande, A., Milch, B., Zettlemoyer, L., Kaelbling, L.: Learning Probabilistic Relational Dynamics for Multiple Tasks., *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI-07)*, 2007.

[13] Deshpande, A., Milch, B., Zettlemoyer, L. S., Kaelbling, L. P.: Learning Probabilistic Relational Dynamics for Multiple Tasks, *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI)*, 2007.

[14] Evgeniou, T., Micchelli, C. A., Pontil, M.: Learning Multiple Tasks with Kernel Methods, *Journal of Machine Learning Research*, **6**, 2005, 615–637.

[15] Ferguson, T. S.: A Bayesian Analysis of Some Nonparametric Problems, *The Annals of Statistics*, **1**(2), 1973, 209–230.

[16] Handel, A., Regoes, R. R., Antia, R.: The Role of Compensatory Mutations in the Emergence of Drug Resistance, *PLoS Computational Biology*, **2**(10), 10 2006, e137.

[17] Hanley, J., McNeil, B.: A method of comparing the areas under receiver operating characteristic curves derived from the same cases, *Radiology*, **148**(3), 1983, 839–843.

[18] Hubbard, T., Murzin, A., Brenner, S., Chothia, C.: SCOP: a structural classification of proteins database, *Nucleic Acids Research*, **25**(1), January 1997, 236–9.

[19] III, H. D.: Bayesian Multitask Learning with Latent Hierarchies, *Proceedings of the 25th Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-09)*, AUAI Press, Corvallis, Oregon, 2009.

[20] Kemp, C., Tenenbaum, J. B., Griffiths, T. L., Yamada, T., Ueda, N.: Learning systems of concepts with an infinite relational model, *AAAI'06: Proceedings of the 21st national conference on Artificial intelligence*, AAAI Press, 2006.

[21] Khan, K., Muggleton, S., Parson, R.: Repeat Learning Using Predicate Invention, *Proceedings of Inductive Logic Programming, 8th International Workshop, Madison, Wisconsin, USA*, 1446, Springer, 1998.

[22] Kramer, S., De Raedt, L.: Feature Construction with Version Spaces for Biochemical Applications, *Proceedings of the 18th International Conference on Machine Learning*, Morgan Kaufmann Publishers Inc., 2001.

[23] Landwehr, N., Kersting, K., De Raedt, L.: nFOIL: Integrating Naive Bayes and FOIL., *Proceedings of the 20th National Conference on Artificial Intelligence, Pittsburgh, Pennsylvania, USA*, 2005.

[24] Landwehr, N., Passerini, A., De Raedt, L., Frasconi, P.: kFOIL: learning simple relational kernels, *Proceedings of AAAI'06*, 2006.

[25] Landwehr, N., Passerini, A., De Raedt, L., Frasconi, P.: Fast Learning of Relational Kernels, *Machine Learning*, **79**(3), 2010, 305–342.

[26] Lengauer, T., Sing, T.: Bioinformatics-assisted anti-HIV therapy, *Nature Reviews Microbiology*, **4**(10), 2006, 790–797.

[27] Madrid-Sanchez, J., Parrado-Hernandez, E., Figueiras-Vidal, A.: Selective Multitask Learning by Coupling Common and Private Representations, *Proceedings of NIPS 08 Workshop on Learning from Multiple Sources*, 2008.

[28] Muggleton, S., Raedt, L. U. C. D. E.: Inductive Logic Programming : Theory and Methods, *University Computing*, 1994, 629–682.

[29] Neal, R.: Density modeling and clustering using dirichlet diffusion trees, *Bayesian Statistics 7*, 2003.

[30] Pettersen, E. F., Goddard, T. D., Huang, C. C., Couch, G. S., Greenblatt, D. M., Meng, E. C., Ferrin, T. E.: UCSF Chimera–a visualization system for exploratory research and analysis., *Journal of Computational Chemistry*, **25**(13), October 2004, 1605–1612, ISSN 0192-8651.

[31] Quinlan, J.: Learning Logical Definitions from Relations, *Machine Learning*, **5**, 1990, 239–266.

[32] Reid, M. D.: Improving Rule Evaluation Using Multitask Learning, *Proceedings of Inductive Logic Programming, 14th International Conference, Porto, Portugal*, 3194, Springer, 2004.

[33] Rhee, S., Taylor, J., Wadhera, G., Ben-Hur, A.: Genotypic predictors of Human Immunodeficiency Virus type 1 drug resistance, *Proceedings of the National Academy of Sciences*, Jan 2006.

[34] Rhee, S.-Y., Gonzales, M. J., Kantor, R., Betts, B. J., Ravela, J., Shafer, R. W.: Human Immunodeficiency Virus reverse transcriptase and protease sequence database, *Nucleic Acids Research*, **31**(1), Jan 2003, 298–303.

[35] Richter, L., Augustin, R., Kramer, S.: Finding Relational Associations in HIV Resistance Mutation Data, *Proceeding of 19th International Conference on Inductive Logic Programming (ILP09)*, Jun 2009.

[36] Roy, D. M., Kemp, C., Mansinghka, V. K., Tenenbaum, J. B.: Learning annotated hierarchies from relational data, in: *Advances in Neural Information Processing Systems 19* (B. Schölkopf, J. Platt, T. Hoffman, Eds.), MIT Press, Cambridge, MA, 2007, 1185–1192.

[37] Taylor, W. R.: The classification of amino acid conservation., *J Theor Biol*, **119**(2), March 1986, 205–218.

[38] Thrun, S., O'Sullivan, J.: Discovering Structure in Multiple Learning Tasks: The TC Algorithm, *Proceedings of the International Conference on Machine Learning '96*, 1996.

[39] Turcotte, M., Muggleton, S. H., Sternberg, M. J.: Automated discovery of structural signatures of protein fold and function, *Journal of Molecular Biology*, **306**(3), 2001, 591 – 605, ISSN 0022-2836.

[40] Wagner, A.: Neutralism and selectionism: a network-based reconciliation., *Nature reviews. Genetics*, **9**(12), December 2008, 965–974.

[41] Walter, H., Schmidt, B., Werwein, M., Schwingel, E., Korn, K.: Prediction of abacavir resistance from genotypic data: impact of zidovudine and lamivudine resistance in vitro and in vivo, *Antimicrobial agents and chemotherapy*, **46**(1), 2002, 89.

[42] Xu, Z., Tresp, V., Yu, K., Kriegel, H.: Infinite Hidden Relational Models, *Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence (UAI 2006)*, Cambridge, MA, USA, July 2006.

[43] Xue, Y., Liao, X., Carin, L., Krishnapuram, B.: Multi-Task Learning for Classification with Dirichlet Process Priors, *Journal of Machine Learning Research*, **8**, 2007, 35–63.