Using Neural Network to Weight the Partial Rules: Application to Classification of Dopamine Antagonist Molecules

Sukree Sinthupinyo[1], Cholwich Nattee[1], Masayuki Numao[1], Takashi Okada[2], and Boonserm Kijsirikul[3]

[1] Department of Architecture for Intelligence, The Institute of Scientific and Industrial Research, Osaka University,
8-1 Mihogaoka, Ibaraki, Osaka, 567-0047, Japan
{cholwich,sukree,numao}@ai.sanken.osaka-u.ac.jp
[2] Department of Informatics, School of Science and Technology, Kwansei Gakuin University
okada@kwansei.ac.jp
[3] Department of Computer Engineering, Chulalongkorn University
boonserm.k@chula.ac.th

Abstract. In this paper, we propose an approach which can help Inductive Logic Programming (ILP) in multiclass domains and also its application to a real world domain, Classification of Dopamine Antagonist Molecules. When we classify an example by using the unordered rules constructed by standard ILP systems in multiclass domains, an example may match with the rules from different classes or may match with no rule in the rule set. Thus, using the rules alone is insufficient. We present the approach which utilises some matches in the rule to classify such examples. First, we extract the pieces of knowledge from the original rules, called partial rule. Then, we apply Neural Network to assign the importance to each partial rule. Finally, we use the weighted partial rules to classify unseen examples. Furthermore, the weights from Neural Network also show the importance of the piece of knowledge which is different from the knowledge originally represented in the form of First Order rules.

1 Introduction

In recent years, Inductive Logic Programming (ILP) has been widely used to discover knowledge from various real-world domains, especially in chemistry domains [1–4]. The knowledge obtained from ILP is explained in the form of First Order Logic which is rich in comprehensiveness and expressiveness. However, the ordinary ILP's systems are two-class classifier. They construct the rules which cover all positive examples and none of the negatives. The rules are then used to classify unseen examples. The example which is covered by some rules are classified as the positive class, otherwise classified as the negative class. Some problems arise when we need to use the rules from ILP in multiclass domains.

An example may match with the rules from different classes or may match with no rule in the rule set. Thus, ILP's rules alone are insufficient.

In this paper, we present the approach which can help ILP in such cases. Our aim is to use only the ordinary rules which are constructed as the unordered rules in multiclass domains. Our approach is based on the idea of using the partial matches from the original rule to collaboratively classify unseen examples. When an example is covered by the rules from different classes or covered by no rule in the rule set, we can make use of some partial matches in the rule to classify the example. To find the partial matches, we extract the partial rules from the original rules, and then compare them to the example. Since the partial rules are some part of the origial rule, they are more general than the original one. Then, the partial rules are used collaboratively to classify the example. Hence, the problem of using ILP's rules is transformed into how to assign the importance to each partial rule and determine which class is most suitable for the example when there are serveral partial rules cover the example.

In our previous work, we employed Winnow-based algorithm to assign the importance to each partial rule [5]. However, in this paper, we present the use of Neural Network, instead of Winnow algorithm, to determine the importance in the form of the weights. Then, we represent the knowledge hidden in the network structure and the weights of the links as *Weight Partial Rules* which not only keep the comprehensiveness and the expressiveness of First Order rule but also represent the importance of each partial rule as the attached weights.

In this work, we applied our approach to a real-world problem, classifying the activity of dopamine antagonist molecules. The dopamine antagonist molecule is a kind of molecules which can block the binding between the dopamines and dopamine receptors in signal transfer process in the brain. The excessive levels of the dopamine have been implicated in schizophrenia. Hence, for the medical treatment of schizophrenic patients, the dopamine antagonist molecules are used to reduce the signal transfer level which can limit the effect of the high density of the dopamines. The knowledge discovered from this domain may be useful for schizophrenic drug development.

The paper is organised as follows. In the next section, we present the concept of ILP and the obstacles when ILP is applied to the multiclass problems. The strategy of Weight Partial Rules is expressed in Section 3. The details of the experiments are presented in Section 4. The paper ends with the conclusion in Section 5.

2 Inductive Logic Programming (ILP)

As mentioned in the previous section, the ordinary ILP systems are two-class classifier. ILP's search strategies aim to seek for the hypothesis which covers all positive examples and none of negative examples. The constructed hypothesis is represented as the First Order rule set which is rich in comprehensiveness and expressiveness. Then, the rule set is used to classify unseen examples. The examples which match with some rule(s) are classified as the positive class, while the

examples which do not match with any rule are classified as the negative class. However, when we need to use ILP in multiclass domains, we must employ other methods to help ILP construct the rule and determine the class of examples. As described before, ILP constructs the rules which cover all positive examples and none of the negatives. While in multiclass problems, we must construct the rules for each class, so that another method is needed. Moreover, after we obtain the rules of each class, the problem of selecting the class for examples may arise. An example may match with the rules from different classes or may not match with any rule in the rule set.

There have been some works which can be employed to help ILP in such cases. Dietterich and Bakiri [6] proposed the method which employs the error correcting code to represent the class of examples and tries to predict the most suitable class for test examples. Round Robin Rule Learning proposed by Fürnkranz [7] focuses on training examples rearrangement. The training examples from each class are used to train the learner several times. A test example is tested with all trained classifiers. The most winning class is selected as the class of the test example. Eineborg and Bostr [8] proposed the method for selecting the class for the uncovered examples, Rule Stretching. The method aims to deal with an uncovered example by generalising the original rules to cover the example and select the most accurate rule as the rule which best matches with the example.

The proposed approach is different from the works described above. Our approach utilises the unordered ILP's rules which are constructed by using the common algorithm, the one-against-all method in which the k-class problem is reduced to k two-class problems. The rules of class i are constructed by using the training examples of class i as positive examples and the training examples of class j where $j = 1, ..., k$ and $j \neq i$ as negative examples. For example, our data set contains 4 classes, i.e. D1, D2, D3, and D4 (as will be described in Section 4). We use the training examples of class D1 as the positive examples and use those of classes D2, D3, D4 as the negatives for learning rules of class D1. Using this strategy, the obtained rules are unordered, so that when an unseen example matches with no rule or with multiple rules from different classes, we cannot select which rule should be applied.

In this work, we select an ILP system, Aleph [9], to construct the rules. Aleph employs the Inverse Entailment algorithm which was previously used by Progol [10] to generate the most specific clause, called *bottom clause*. Then, to seek for the best generalised clause, Aleph provides many search algorithms which users can select the most suitable one for their domain. In our experiments, we selected the randomised search method using an altered form of the GSAT algorithm [11] that was originally proposed for solving propositional satisfiability problems. The GSAT algorithm provided by Aleph is modified to suit the sequence of literals searching process in ILP fashion.

3 Weighted Partial Rules

Our approach is based on the idea that some partial matches of the rule can be used to classify unseen examples. The partial matches of the rule are represented in the form of the partial rules which are extracted from the original rule. Then, we utilise all extracted partial rules to classify the unseen examples. The class of a test example should be the class from which the important partial rules cover the example more than those from other classes. In this paper, to determine the importance of each partial rule, we train a neural network to assign the importance to the partial rule in the form of the weights of the links.

3.1 Partial Rule Extraction

A *partial rule* is *a rule whose body contains a valid sequence of the literals, from the body of the original rule, which starts with the literal consuming the input variables in the head of the rule.* Our partial rule extraction algorithm is based on the idea of the new introduced variables, similar idea as the feature extraction in BANNAR [12]. A literal will be added to the sequence if it consumes some new variables introduced by previously added literals in the sequnce. The extraction procedure starts with an empty sequence, and uses variables in the head of the rule as new variables. Then, the literal which consumes the new variables as input variables is gradually added to the sequence. The new variables introduced in the newly added literal are again used as the new variables for searching other literals to be added. The search stops when the newly added literal introduces no new variable or cannot find any literal which consumes the new variables in this newly added literal. Finally, we make all possible combinations of the two sequences which have the common variables not occurring in the head. The partial rule extraction algorithm is shown in Figure 1.

3.2 Neural Network Training

As mentioned earlier, when we use the unordered rules to classify examples in multiclass fashion. The examples may be not covered by any rule or may be multiple covered by the rules from different classes. In the previous subsection, we can see that the literals in the body of partial rules are subset of the literals in the body of the original rule. This causes the partial rules are more general than the original one. Hence, when we use the partial rules instead of the original rule, the number of the uncovered examples decreases while the number of the multiple covered examples increases. So that the problem is now transformed into how to select the class for the example which is covered by the rules from different classes. The class of the example should be the class from which the number of the covering important partial rules is more than those from other classes. We thus employ neural network algorithm to assign the weight to each partial rule.

Since we must finally extract the knowledge from the network, we decide to use the simple network structure which consists of only two layers, i.e. *input layer*

```
function ExtractPartialRule(rule)
returns a sequence list
  inputs:    rule as original rule
  variables: literals, remained_literals as sequence of literals
             output, combined, partial_rules as sequence list
             literal as literal
  output ← empty list
  literals ← the body of rule
  for each literal, in literals, which consumes the variables in the head of
  rule as input
    remained_literals ← remove literal from literals
    partial_rules ← SearchPartialRule(literal, literal, remained_literals,
                    output)
    output ← add partial_rules to output
  combined ← make all possible combination of sequence of literals in
             output, which have common variables that do not occur
             in the head of the rule
  output ← add combined to output
  remove redundant sequence of literals from output
  return output


function SearchPartialRule(input_literal, partial_rule, literals, output)
returns a sequence list
  inputs:    input_literal as literal
             partial_rule, literals as sequence of literals
             partial_rules as sequence list
  variables: literal as literal
             remained_literals, new_partial_rule as sequence of literals
             unfinish as sequence of literals
             new_partial_rules as sequence list
             found as boolean, initially false
  new_partial_rule ← add input_literal to partial_rule
  if no new argument in input_literal then
    partial_rules ← add new_partial_rule to partial_rules
    output ← add partial_rules to output
    return output
  for each literal, in literals, which consumes the new variables
  in input_literal as input
    unfinish ← add literal to new_partial_rule
    remained_literals ← remove literal from literals
    partial_rules ← SearchPartialRule(literal, unfinish,
                    remained_literals, output)
    output ← add partial_rules to output
    found ← true
  if found then
    partial_rules ← add new_partial_rule to partial_rules
    output ← add partial_rules to output
  return output
```

Fig. 1. Partial Rule Extraction Algorithm

and *output layer*. Each input node represents the truth value of the body of each partial rule and each output node represents a class. The input nodes are fully connected to the output nodes. The number of input nodes is the number of the partial rules extracted from the whole original rules from all classes. The number of output nodes is the number of the classes. All output nodes are sigmoid unit.

In training process, an input vector contains elements each of which is the truth value of each partial rule evaluated against a training example and the corresponding output vector is the vector which represents the class of the example. The input value of the input node representing the *true* partial rule is 1, while the input value of the node representing the *false* partial rule is -1. For the output vector, we assign 1 for the node that represents the class of the example and 0 for the others. The structure of our neural network is shown in Figure 2. The example of generating a training vector for the neural network is shown below.

Consider the following rules R_1, R_2, R_3, R_4, for classes D1, D2, D3, and D4, respectively.

R_1: `molecule(A,d1) :- atm(A, B, C, D, E, F), F=1.4, E=2.3,`
`atm(A, G, H, I, J, K), J=1.5.`
R_2: `molecule(A,d2) :- atm(A, B, C, D, E, F), bond(A, G, H, I, J, K),`
`E=2.8, C=n, gteq(K, 1.5).`
R_3: `molecule(A,d3) :- link(A, B, C, D), atm(A, E, F, G, H, I),`
`I=3.8, H=3.6, D=4.8.`
R_4: `molecule(A,d4) :- link(A, B, C, D), atm(A, E, F, G, H, I),`
`I=1.4, H=8.5, D=2.9.`

The following rule R_iC_j is a partial rule extracted from R_i.

P_1: R_1C_1: `molecule(A,d1) :- atm(A, B, C, D, E, F), F=1.4.`
P_2: R_1C_2: `molecule(A,d1) :- atm(A, B, C, D, E, F), E=2.3.`
P_3: R_1C_3: `molecule(A,d1) :- atm(A, G, H, I, J, K), J=1.5.`
P_4: R_1C_4: `molecule(A,d1) :- atm(A, B, C, D, E, F), F=1.4, E=2.3.`

P_5: R_2C_1: `molecule(A,d2) :- atm(A, B, C, D, E, F), E=2.8.`
P_6: R_2C_2: `molecule(A,d2) :- atm(A, B, C, D, E, F), C=n.`
P_7: R_2C_3: `molecule(A,d2) :- bond(A, G, H, I, J, K), gteq(K, 1.5).`
P_8: R_2C_4: `molecule(A,d2) :- atm(A, B, C, D, E, F), E=2.8, C=n.`

P_9: R_3C_1: `molecule(A,d3) :- link(A, B, C, D), D=4.8.`
P_{10}: R_3C_2: `molecule(A,d3) :- atm(A, E, F, G, H, I), I=3.8.`
P_{11}: R_3C_3: `molecule(A,d3) :- atm(A, E, F, G, H, I), H=3.6.`
P_{12}: R_3C_4: `molecule(A,d3) :- atm(A, E, F, G, H, I), I=3.8, H=3.6.`

P_{13}: R_4C_1: `molecule(A,d4) :- link(A, B, C, D), D=2.9.`
P_{14}: R_4C_2: `molecule(A,d4) :- atm(A, E, F, G, H, I), I=1.4.`
P_{15}: R_4C_3: `molecule(A,d4) :- atm(A, E, F, G, H, I), H=8.5.`

P_{16}: R_4C_4: `molecule(A,d4) :- atm(A, E, F, G, H, I), I=1.4, H=8.5.`

We now have 16 partial rules of 4 classes, so that our neural network contains 16 input nodes and 4 output nodes. Assume that we are constructing the input and output vector of `molecule(m06497)`, an example of class D1. We first evaluate the truth value of `molecule(m06497)` when applying it to each partial rule. Then, the obtained truth values are organised as an input vector. For example, if the truth values of all partial rules from R_1 and only the second one from R_2 are *true* while the others are $false$, the input vector of this example will be $(1, 1, 1, 1, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1)$. As this is an example of class D1, the output value of the output node representing class D1 is only activated while the others are zero. Thus, the output vector is $(1, 0, 0, 0)$.

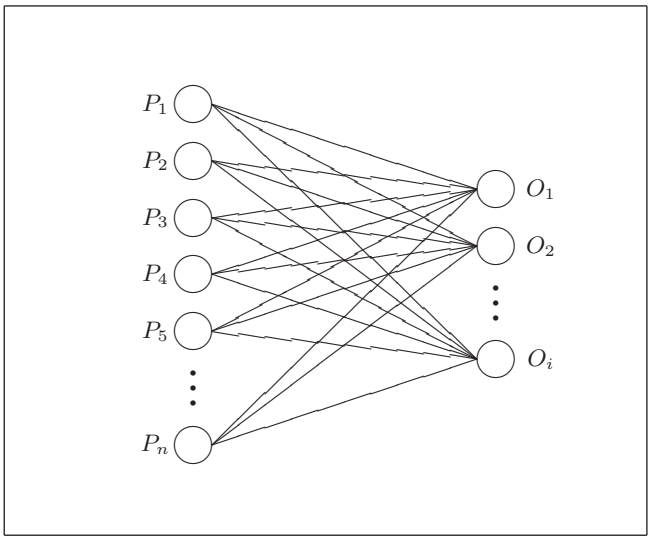

Fig. 2. The Neural Network Structure

3.3 Weighted Partial Rules

After the neural network is trained, the importance of each partial rule is hidden in the weights and the structure of the network. This makes the comprehensiveness disappear, if we directly classify a new example by using the network instead of the original rules. We thus propose a knowledge extraction technique to convert the weights and the structure of the neural network into the human-understandable form. The strategy used in our approach is described as follows.

Given a problem with n partial rules and m classes. W_i is a vector of length m, where the element $w_{i,j}$ of W_i is the weight of the link between input node i and output node j, P_i is the partial rule represented by input node i, W_0 is a vector of the bias of output nodes, and $P_0 = true$. The *Weighted Partial Rules*, WPR, is a vector of length $n+1$, where the i^{th} element of WPR is a pair (W_i, P_i).

To classify example e, we use the following strategy.

- Let V is a vector of length m, initially with $V = W_0$.
- Compare e with P_i in all elements of WPR. If e matches with P_i, $V \leftarrow V + W_i$ else, $V \leftarrow V - W_i$.
- Let v_l be the maximum in V, return class l.

In above strategy, we represent the knowledge hidden in the neural network in the form of a vector of the pairs (W_i, P_i), where W_i is a vector of the weights of the links from the input node representing the partial rule P_i to all output nodes. The class which has the maximum output value is selected as the class of the example.

In the feed forwarding step of our backpropagation neural network, the output node is composed of two consecutive units , i.e. the *summation unit* and the *activation unit*. The inputs of the summation unit are the multiplication of the input values and the weights of the links from all input nodes to that output node and the output of the unit is the summation of all inputs. The output of the summation unit is sent through the activation unit which uses the sigmoid function as the activation function. Hence, the final output of the output node is limited to the interval $(0, 1)$. However, in our approach, the information we only need is which output node gives the maximum value. Thus, we can ignore the usage of the sigmoid function because the sigmoid function, $sigmoid(x) = \frac{1}{1+e^{-x}}$, is an increasing function where $sigmoid(x_i) > sigmoid(x_j)$ when $x_i > x_j$. The output of the function is higher for the higher input value. Therefore, only for comparison purpose as in our approach, it is not necessary to use the activation unit, and only the summation unit is sufficient.

For example, let $w_{0,i}$ be the bias value of output node i and $w_{i,j}$ is the weight of the link between input node i and output node j. From the previous example, the WPR which represents the original rules and their importance is shown below:

$$
\begin{aligned}
WPR \quad = \quad (\quad & ((w_{0,1}, w_{0,2}, w_{0,3}, w_{0,4}), true), \\
& ((w_{1,1}, w_{1,2}, w_{1,3}, w_{1,4}), P_1), \\
& ((w_{2,1}, w_{2,2}, w_{2,3}, w_{2,4}), P_2), \\
& ((w_{3,1}, w_{3,2}, w_{3,3}, w_{3,4}), P_3), \\
& ((w_{4,1}, w_{4,2}, w_{4,3}, w_{4,4}), P_4), \\
& ((w_{5,1}, w_{5,2}, w_{5,3}, w_{5,4}), P_5), \\
& ((w_{6,1}, w_{6,2}, w_{6,3}, w_{6,4}), P_6), \\
& ... \\
& ((w_{16,1}, w_{16,2}, w_{16,3}, w_{16,4}), P_{16}) \quad)
\end{aligned}
$$

4 Experiments

The data set used in the experiments contained 1366 molecules of dopamine antagonist molecules of 4 classes, D1, D2, D3, and D4 [13]. The information of the molecules was originally described in the form of the position in three dimension space of atoms, types of atoms, types of bonds, and dopamine antagonist activity of molecules. However, the position in three dimension space was not useful for discriminating examples because a molecule could rotate or move to other positions in the space. Hence, we converted the positions of atoms to the relations between atoms and bonds. We instead represented the information of atoms, bonds, and distances between atoms in term of 3 predicates, `atm/6`, `bond/6`, and `link/4`, respectively. The details of these three predicates are described below:

- `atm(A,B,C,D,E,F)` represents that the atom `B` is in molecule `A`, is type `C`, forms a bond with oxygen atom if `D` is 1, otherwise it does not link to any oxygen atom, has distance `E` to the nearest oxygen atom, and has distance `F` to the nearest nitrogen atom.
- `bond(A,B,C,D,E,F)` represents that the bond `B` is in molecule `A`, has atoms `C` and `D` on each end, is type `E`, and has length `F`.
- `link(A,B,C,D)` represents that in the molecule `A`, the distance between atoms `B` and `C` is `D`.

4.1 Compared Approaches

We compare our approach with other three approaches, i.e. Winnow-Based algorithm [14], Majority Class [15, 16] method and Decision Tree Learning algorithm. The brief review of each approach is described below.

Winnow-Based Algorithm

Each partial rule can be viewed as an expert which can vote for the class of test examples. The weights of the partial rules which cover the example are summarised and the class with the highest summation is selected as the class of the example. The weights are obtained by using the training strategy described below:

Given a problem with n partial rules, m classes, and promotion factor α. P is a vector of length n, where element p_i of P is a partial rule. W_i is a vector of length m, where element $w_{i,j}$ of W_i is the weight of class j of partial rule p_i. V is a summation vector of length m, where v_i of V is the summation of the weights of class i. The weight vector W_i are updated by using the following procedure.

- Initialize all $w_{i,j} = 1$
- Until termination condition is met, Do
  - For each training example e, Do
    - Initialize all $v_i = 0$ and c as the class of e

- For all partial rules p_i which match with e, add corresponding W_i to V,

$$V = V + W_i$$

- Let v_k be the maximum element in V, predict the example e as class k
- If $c = k$, no update is required; otherwise the weight w_i corresponding to p_i which matches with e is updated by,

$$w_{i,j} = \begin{cases} \alpha w_{i,j} & \text{if } j = k, \\ \alpha^{-1} w_{i,j} & \text{if } j = c. \end{cases}$$

For more details please refer to [5].

Majority Class Method

In the Majority Class method, we selected the class which had the maximum number of examples in training set as the default class. An example which matched with only rule(s) from one class was classified as that class, while an example which could not match with any rule was classified as the default class. In case of the examples which matched with the rules from two or more classes, we selected the class of which the matched rules covered maximum number of examples.

Decision Tree Learning Algorithm

Decision Tree Learning (DTL) is a well-known propositional Machine Learning technique which employs the Information Theory to guide in searching for the best theory. DTL has been successfully applied to many attribute-value real-world domains [17–19]. The decision tree learner used in our experiments was C4.5 system [20].

We trained C4.5 using the features obtained by comparing the partial rules with an example, and these features were a set of truth values (either $true$ or $false$). The features were the same as those used as the input vector of the neural network. The reason that we selected C4.5 to apply to the same feature set is to compare the efficiency of the weights from the neural network which is employed in the proposed method with the decision tree.

4.2 Experimental Results

We ran 10-fold cross validation experiment using four approaches, the original ILP system with the Majority Class method (ILP+Majority Class), Partial Rules and C4.5 (PR+C4.5), Partial Rules and Winnow (PR+Winnow), and our approach, Weighted Partial Rules (WPR).

Table 1 shows the accuracy of each approach on the test examples. The accuracy of ILP+Majority Class approach is 79.11%. The accuracy of PR+C4.5 is

85.71%, higher than ILP+Majority with 99.5% confidence level using the standard paired t-test method. The accuracy of PR+Winnow is 88.65%, higher than ILP+Majority and PR+C4.5 with 99.5% and 99.0% respectively, The accuracy of WPR is 92.03%, higher than other approaches with 99.5% confidence level using the same comparing method.

Table 1. The accuracy of the compared approaches.

Method	Accuracy (%)
ILP+Majority Class	79.11±4.37
PR+C4.5	85.71±3.41
PR+Winnow	88.65±3.85
WPR	92.03±3.14

Furthermore, in Table 2, we show the ratio between the number of examples correctly classified and the number of examples for each portion. Exactly Covered column indicates the number of the examples covered by the rule(s) from only one class, Multiple Covered column indicates the number of the examples covered by the rules from different classes, and Uncovered column indicates the number of the examples which are not covered by any rule. For exactly covered examples, WPR correctly classified 991 of 1049 examples, whereas 965 were correctly classified by the rules from ILP. For the examples covered by multiple rules from different classes, WPR correctly classified 81 of 97 examples, more 32 examples than the Majority Class method. Finally, 185 of 220 examples were correctly classified by WPR, while only 68 examples were correctly classified by the Majority Class method. These results show that WPR much improved the accuracy in Multiple Covered and Uncovered, and slightly improved in Exactly Covered portion.

Table 2. Improvements of WPR over the original rules with Majority Class method, reported according to exactly covered examples, multiple covered examples, and uncovered examples.

Method	Exactly Covered	Multiple Covered	Uncovered
ILP+Majority Class	965/1049	49/97	68/220
WPR	991/1049	81/97	185/220

An example of some partial rules which are highly weighted is shown below.

```
molecule(A) :- atm(A, B, C, D, E, F), bond(A, G, B, H, I, J),
   bond(A, K, H, L, M, J), atm(A, L, C, D, E, S), F=3.3.
[2.88, -1.95, -0.97, -0.27]
```

```
[The original rule is
molecule(A) :- atm(A, B, C, D, E, F), F=3.3, bond(A, G, B, H, I,
   J), bond(A, K, H, L, M, J), bond(A, N, L, O, M, P), bond(A, Q,
   O, R, M, P), atm(A, L, C, D, E, S).]

molecule(A) :- atm(A, E, F, G, H, I), bond(A, N, E, O, P, M),
   atm(A, O, F, G, Q, R), H=2.4.
[-1.42, 4.12, -1.11, -2.16]
[The original rule is
molecule(A) :- link(A, B, C, D), atm(A, E, F, G, H, I), D=5.6,
   H=2.4, gteq(I, 3.8), bond(A, J, B, K, L, M), bond(A, N, E, O,
   P, M), atm(A, O, F, G, Q, R), lteq(Q, 2.9), lteq(M, 1.4),
   bond(A, S, C, T, P, U).]
```

The Weight Partial Rules in the above example show another advantage of our approach. We can see that when an example matches with these highly weighted partial rules, the example has the high probability of being classified as the class whose weight is very high. This provides us some knowledge which can be discovered from the dataset, different from the original rules which sometimes are too specific and not useful. Our approach can seek for some pieces of knowledge which are more important than the others in the original rule.

5 Conclusion

We have proposed an approach which not only improves the accuracy of ILP's rules in multiclass problems but also extracts some pieces of knowledge from the rules. The approach is based on the idea that the importance of the partial matches of the rules is different. In this work, we train the neural network to assign the weights, which represent the importance, to the partial rules. The experimental results on classifying the activity of the dopamine antagonist molecules show that our approach was successfully applied to the domain by yielding 92.03% accuracy. Moreover, the weights of the partial rules also show some important pieces of knowledge which are previously hidden in the original rules.

References

1. Finn, P.W., Muggleton, S., Page, D., Srinivasan, A.: Pharmacophore Discovery Using the Inductive Logic Programming System (PROGOL). Machine Learning **30** (1998)
2. Pompe, U., Kononenko, I., Makše, T.: An application of ILP in a musical database: Learning to compose the two-voice counterpoint. In: Proceedings of the MLnet Familiarization Workshop on Data Mining with Inductive Logic Programing. (1996) 1–11

3. Quiniou, R., Cordier, M.O., Carrault, G., Wang, F.: Application of ILP to cardiac arrhythmia characterization for chronicle recognition. Lecture Notes in Computer Science **2157** (2001) 220–227
4. Turcotte, M., Muggleton, S.H., Sternberg, M.J.E.: Application of Inductive Logic Programming to discover rules governing the three-dimensional topology of protein structure. In Page, D., ed.: Proceedings of the 8th International Conference on Inductive Logic Programming. Volume 1446., Springer-Verlag (1998) 53–64
5. Sinthupinyo, S., Nattee, C., Numao, M., Okada, T., Kijsirikul, B.: Combining partial rules and winnow algorithm: Results on classification of dopamine antagonist molecules. In: The Third Internation Workshop on Active Mining. (2004)
6. Dietterich, T.G., Bakiri, G.: Solving multiclass learning problems via error-correcting output codes. Journal of Artificial Intelligence Research **2** (1995) 263–286
7. Fürnkranz, J.: Round robin rule learning. In Brodley, C.E., Danyluk, A.P., eds.: Proceedings of the 18th International Conference on Machine Learning (ICML-01), Williamstown, MA, Morgan Kaufmann Publishers (2001) 146–153
8. Eineborg, M., Boström, H.: Classifying uncovered examples by rule stretching. Lecture Notes in Computer Science **2157** (2001)
9. Srinivasan, A.: The Aleph Manual (2001)
10. Muggleton, S.: Inverse Entailment and Progol. New Generation Computing **13** (1995) 245–286
11. Selman, B., Levesque, H.J., Mitchell, D.: A New Method for Solving Hard Satisfiability Problems. In: Proc. 10th National Conference on Artificial Intelligence, AAAI Press (1992) 440–446
12. Kijsirikul, B., Sinthupinyo, S., Chongkasemwongse, K.: Approximate match of rules using backpropagation neural networks. Machine Learning **44** (2001) 273–299
13. Nattee, C., Sinthupinyo, S., Numao, M., Okada, T.: Knowledge discovery using inductive logic programming in dopamine antagonist structure data. Technical report, I.S.I.R., Osaka University (2004) http://libra.ai.sanken.osaka-u.ac.jp/usr/cholwich/techreport.pdf.
14. Littlestone, N.: Learning quickly when irrelevant attributes abound: A new linearthreshold algorithm. Machine Learning **2** (1988) 285–318
15. Clark, P., Boswell, R.: Rule induction with CN2: Some recent improvements. In: Proc. Fifth European Working Session on Learning, Berlin, Springer (1991) 151–163
16. Laer, W.V., Raedt, L.D., Dzeroski, S.: On multi-class problems and discretization in inductive logic programming. In: International Syposium on Methodologies for Intelligent Systems. (1997) 277–286
17. Milde, H., Hotz, L., Kahl, J., Neumann, B., Wessel, S.: MAD: A real world application of qualitative model-based decision tree generation for diagnosis. In: Industrial and Engineering Applications of Artificial Intelligence and Expert Systems. (1999) 246–255
18. Milde, H., Hotz, L., Kahl, J., Wessel, S.: Qualitative model-based decision tree generation for diagnosis in real world industrial application. KI - Kunstliche Intelligenz **13** (1999) 30–35
19. Ye, N., Li, X., Emran, S.M.: Decision tree for signature recognition and state classification. In: IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop June 6-7, 2000 at West Point, New York. (2000) 194–199
20. Quinlan, J.: C4.5: Programs for machine learning. Morgan Kaufmann Publishers (1993)