Soft Comput (2009) 13:259–272
DOI 10.1007/s00500-008-0321-0

FOCUS

Hierarchical classification of protein function with ensembles of rules and particle swarm optimisation

Nicholas Holden · **Alex A. Freitas**

Published online: 27 May 2008

Abstract This paper focuses on hierarchical classification problems where the classes to be predicted are organized in the form of a tree. The standard top-down divide and conquer approach for hierarchical classification consists of building a hierarchy of classifiers where a classifier is built for each internal (non-leaf) node in the class tree. Each classifier discriminates only between its child classes. After the tree of classifiers is built, the system uses them to classify test examples one class level at a time, so that when the example is assigned a class at a given level, only the child classes need to be considered at the next level. This approach has the drawback that, if a test example is misclassified at a certain class level, it will be misclassified at deeper levels too. In this paper we propose hierarchical classification methods to mitigate this drawback. More precisely, we propose a method called hierarchical ensemble of hierarchical rule sets (HEHRS), where different ensembles are built at different levels in the class tree and each ensemble consists of different rule sets built from training examples at different levels of the class tree. We also use a particle swarm optimisation (PSO) algorithm to optimise the rule weights used by HEHRS to combine the predictions of different rules into a class to be assigned to a given test example. In addition, we propose a variant of a method to mitigate the aforementioned drawback of top-down classification. These three types of methods are compared against the standard top-down hierarchical classification method in six challenging bioinformatics datasets, involving the prediction of protein function. Overall HEHRS with the rule weights optimised by the PSO algorithm obtains the best predictive accuracy out of the four types of hierarchical classification method.

Keywords Particle swarm optimisation · Hierarchical classification · Protein function prediction

1 Introduction

Data mining consists of a set of concepts and techniques used to find useful knowledge in real-world data (Witten and Frank 2005; Fayyad et al. 1996). In this project the discovered knowledge is represented as classification rules. A rule consists of an antecedent (a set of attribute values) and a consequent (class):

$$\text{IF} < \text{attrib} = \text{value} > \text{AND} \ldots \text{AND} < \text{attrib} = \text{value} > \\ \text{THEN} < \text{predicted class} >$$

This kind of knowledge representation has the advantage of being intuitively comprehensible to the user. This is important, because in certain fields a major goal of data mining is to discover knowledge that is not only accurate, but also comprehensible (Witten and Frank 2005; Fayyad et al. 1996) in order to give the user insights about the data, the application domain and also provide a way in which to validate the model and predictions produced.

In this paper the classes are arranged in a tree structure where each node (class) has only one parent—with the exception of the root of the tree, which does not have any parent and does not correspond to any class. For example, if there were a hierarchical dataset about animals the classes might be structured in the following way. The top level classes might be cat and dog, then the cat class might have child classes Siamese

N. Holden (✉) · A. A. Freitas
Computing Laboratory, University of Kent,
Canterbury, CT2 7NF, UK
e-mail: nh56@kent.ac.uk

A. A. Freitas
e-mail: A.A.Freitas@kent.ac.uk

and Burmese, the dog class might have child classes called Greyhound and Datsun. Hierarchical class datasets present two main new challenges when compared to flat class datasets. Firstly, many (depending on the depth) more classes must be assigned to the examples. Secondly, the prediction of a class becomes increasingly difficult as deeper levels are considered, due to the smaller number of examples per class.

In this paper we apply classification techniques to predict protein function, a very active research topic in bioinformatics. It is important to discover the function for new proteins to advance our understanding of the workings of the cell and to create more effective medical treatments. It can be a very time consuming task to "manually" assign a single protein with a function. When using data mining techniques it is possible to predict the function of many proteins, quickly and with reasonable accuracy. Also, the patterns discovered in the process can be used to gain insight into the general nature of the relationship between protein biochemical properties and function in different sets of proteins.

The datasets examined in this paper, as with many other bioinformatics datasets, pose a significant problem for any classification technique as they have a relatively large number of attributes, in this case ranging from 126 to 708 attributes. The large number of attributes increases the search space for the classification algorithm which often leads to sub-optimal performance.

In this paper we propose novel ensemble based data mining methods tailored to the hierarchical classification problem and apply them to six protein data sets. Although some research has applied ensemble methods to protein data sets (Huang et al. 2004; Bulashevska and Eils 2006; Tan et al. 2003) and to hierarchical data sets (Eisner et al. 2005), previous research concentrates on "classical" ensemble techniques such as bagging, or ignores any hierarchy present. Some work has been conducted in the field of hierarchical multi-label protein function prediction (Clare 2003; Blockeel et al. 2006) but their approaches rely on modifying the base classification algorithm, rather than using ensemble techniques.

The remainder of this paper is organised as follows. Section 2 presents a brief introduction to top-down hierarchical classification. Section 3 describes our proposed techniques for hierarchical classification. Section 4 describes the creation of the bioinformatics data sets. Section 5 reports the results of the experiments. Section 6 discusses conclusions and future research directions.

2 Top-down hierarchical classification

This paper focuses on hierarchical classification problems where the classes to be predicted are organized in the form of a tree, hereafter referred to as a class tree. The simplest—and naïve—way to deal with hierarchical classification is to ignore the class hierarchy completely and so only predict classes at the bottommost level, indirectly predicting the classes at higher levels. For instance, in a 3-level classification problem, if the algorithm predicts for a given example the class 2.1.5, it is also predicting class 2 at the first level and class 2.1 at the second level. This approach avoids the complexity associated with a hierarchical classification algorithm at the expense of not discovering more generalised knowledge expressed by higher level rules. It discovers only lowest level, specific rules. Such specific rules tend to be less accurate than generic rules (predicting classes at higher levels of the hierarchy), because each of the low-level, specific rules is usually covering a smaller number of examples than a high-level, generic rule—since the number of examples per class at the bottommost class level tends to be smaller than the number of examples per class at higher levels of the class tree. Another problem with this naïve approach is that it does not use information associated with higher-level classes in order to guide the prediction of lower-level classes. This point is explained in more detail below.

To avoid the problems associated with the aforementioned naïve approach, it is possible to use a top-down approach based on the divide-and-conquer principle (Sun and Lim 2001). This top-down approach has the important advantage of using information associated with higher-level classes in order to guide the prediction of lower-level classes. For instance, if class 1.X.X.X (where X denotes any digit) is predicted at the first level and the tree node for that class has only the child nodes 1.1.X.X and 1.2.X.X, only these two class nodes should be considered and not the children belonging to node 2.X.X.X. In general, any model constructed in the top-down divide and conquer tree only has to discriminate between sibling classes. When it comes to the classification of an example, each classifier chooses which child classifier to send the example to, or if the classifier is at the leaf level what final class the examples should be assigned to. For instance, the root classifier, which discriminates between classes 1.X.X.X and 2.X.X.X will decide if an example should be sent to the classifier discriminating between the child classes of class 1.X.X.X or 2.X.X.X. If the example is first classified as 1.X.X.X then it will be sent to the classifier discriminating between classes 1.1.X.X and 1.2.X.X. This classifier will decide if the example should be sent to the classifier discriminating between the child classes of class 1.1.X.X or 1.2.X.X. If the example is then classified as 1.1.X.X then the next classifier (discriminating between 1.1.1.X and 1.1.2.X) will decide if it should be sent to the classifiers discriminating between the child classes of 1.1.1.X or 1.1.2.X and so on.

3 Proposed methods to improve top-down hierarchical classification

3.1 Hierarchical ensemble of hierarchical rule sets (HEHRS)

Ensemble classifiers normally try to combine the predictions from separate classifiers in order to increase predictive accuracy (Tan et al. 2006). The two main considerations when designing an ensemble of classifiers are the accuracy of the component classifiers and the diversity of the component classifiers. In single classifiers the accuracy is the most important factor. However in ensemble classifiers each classifier need not necessarily be especially accurate for the ensemble to make an accurate prediction. Skalak (1997) discusses an example of this phenomenon where a classifier that is 69% accurate is combined with classifiers that are 23% accurate and 25% accurate, and this boosts overall accuracy to 88%. The diversity of the component classifiers is very important in ensemble approaches, because component classifiers must make different errors to make the overall ensemble more accurate (Brown et al. 2005; Battiti and Colla 1994). There is often a trade-off between accuracy and diversity in classifiers, as it is often easier to make more diverse (uncorrelated) classifiers when the classification accuracies of the individual classifiers are lowered.

One of the most popular kinds of ensemble method is bagging (Breiman 1996). In bagging the training set is re-sampled several times in some way to generate separate classifiers, so that each classifier is trained with a different training set. The predictions of these classifiers are classically combined by a voting scheme which may or not be weighted. There are many methods to try and increase the performance of the combining scheme used in bagging (Derbeko et al. 2002; Dietterich 1997; Günter and Bunke 2004). Some of the most advanced and successful of these methods are genetic algorithms (Günter and Bunke 2004; Stefano et al. 2003; Peng et al. 2006; Ranawana and Palade 2005).

In essence, the proposed ensemble method for hierarchical classification can be considered as a new variation of bagging adapted to hierarchical classification. In this method, an ensemble of rules is created by varying the sets of positive and negative examples *according to the class hierarchy*. In order to classify test examples, the predictions of the rules are combined by using a weighted voting scheme. Such a method should improve the accuracy beyond the use of a non-ensemble based technique as the errors in each model can be, to some extent, mitigated by combing the predictions made by multiple models, as discussed previously. As the bioinformatics data sets used in this paper make it more difficult to induce accurate models—because of the high number of classes, attributes and the sparseness of the data at lower levels—the potential benefits from using such an error correcting technique become greater.

3.1.1 Technical details of the HEHRS method

Let us first describe the basic idea of the proposed method at a high level of abstraction. Recall that in the standard top-down hierarchical classification approach a rule set is built to distinguish between a set of sibling class nodes, using the training examples belonging to those sibling class nodes. By contrast, in the proposed HEHRS method K rule sets will be built for each set of sibling nodes in the class tree, where K is the number of class levels between the current level (inclusive) and the deepest class level (inclusive) which is a descendant from either of the current class nodes. For instance for a non-leaf node in level 2 of the class tree and a class tree with 4 levels (not counting the root node which is at the 0th level), $K = 3$ rule sets will be generated, namely one rule set for each of the class levels 2, 3 and 4. All these rule sets contain rules predicting classes at the second level of the class tree, but they are called here hierarchical rule sets because they have been produced from examples at different levels of the class tree. In addition, to continue with this example, these three rule sets also form an ensemble of rule sets, and the proposed method builds several ensembles like this, at different levels of the class tree. Therefore, the ensembles also form a hierarchy, namely a hierarchy of ensembles, where each ensemble consists of a hierarchical rule set. Hence, this approach is here called HEHRS.

Let us now describe HEHRS in more detail, starting with notation issues. In general an ensemble of rule sets created for a given set of sibling class nodes is denoted as $E_{s,}$ where S is a set of sibling class nodes. A rule set within this ensemble (one of the K rule sets) is denoted by the letter i, where i corresponds to the level at which the rule set is built. Therefore any rule set belonging to E_s at the level i, used to distinguish between a set of sibling class nodes S, is denoted by E_{si}. Note that each rule in E_{si} will predict one of the classes in S, so there will be one or more rules (i.e. a subset of the rules in E_{si}) predicting each class in S. A set of rules in E_{si} built at level i predicting a single class d in S is denoted as E_{sid}. As discussed previously E_{sid} consists of K rule sets, each containing rules produced from a different level of the class tree. For each level i and for each class c which is a descendant class of d, the rule induction algorithm will discover rules predicting class d, using as positive examples the examples having class c, and using as negative examples the examples having any class different from c at level i that is a descendant of d in the class tree. These concepts are illustrated in Fig. 1. Note that Fig. 1 refers to a hierarchy of rule sets, rather than the class hierarchy. Hence, each node (d) in Fig. 1 denotes an ensemble of rule sets for a given class (as explained next).

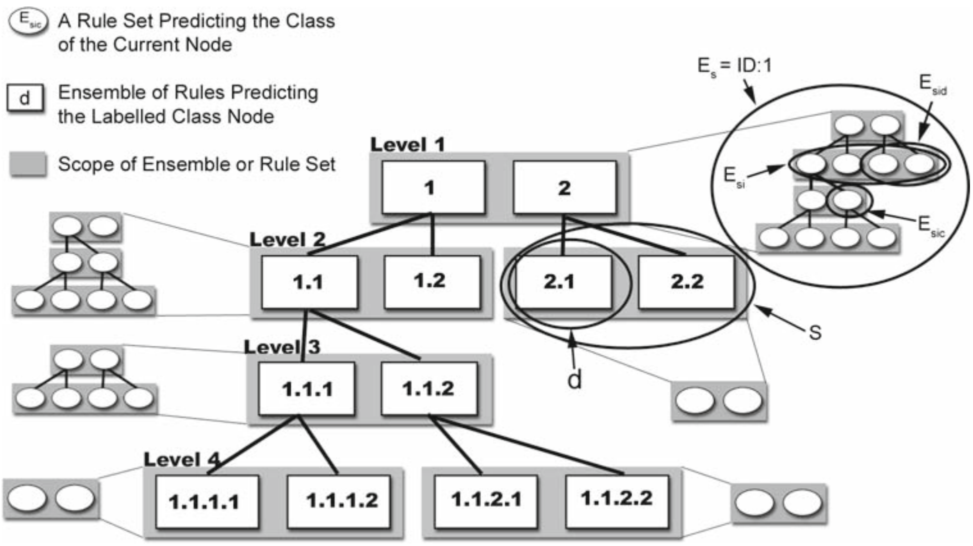
E_sic
A Rule Set Predicting the Class of the Current Node
d
Ensemble of Rules Predicting the Labelled Class Node
Scope of Ensemble or Rule Set
E_s = ID:1
E_sid
E_si
E_sic
Level 1
1
2
Level 2
1.1
1.2
2.1
2.2
S
d
Level 3
1.1.1
1.1.2
Level 4
1.1.1.1
1.1.1.2
1.1.2.1
1.1.2.2

Fig. 1 Hierarchical ensemble of hierarchical rule sets (*HEHRS*)

Table 1 Values (Classes) taken by variable c at each level i used to construct the Rule Sets in E_s ID:1, in Fig. 1

	Class (d) in set of sibling classes S	
Level (i)	$d = 1$	$d = 2$
1	1	2
2	1.1, 1.2	2.1, 2.2
3	1.1.1, 1.1.2	NA
4	1.1.1.1, 1.1.1.2, 1.1.2.1, 1.1.2.2	NA

In Fig. 1 the grey boxes represent the scope of the classification performed by a given rule set (E_{si}) or ensemble (E_s). In the case of E_s the scope of the classification involves the sibling classes S. The main tree—i.e, the large tree at the centre of Fig. 1—shows the hierarchy of ensembles in a standard top-down approach. The expanded (smaller) trees show the rule sets (E_{si}) generated by HEHRS. For each set of sibling classes (S) in the main tree, there is a hierarchy of rule sets in the corresponding smaller tree, indicated by the presence of several grey boxes in the smaller tree. The label S in Fig. 1 shows an example set of sibling classes (2.1 and 2.2) predicted by an ensemble, and the label d within S shows one of the classes (2.1) in the set S.

Table 1 shows in detail the variation in the sets of examples used at different class levels when inducing classification rules for HEHRS, with respect to Fig. 1. For example, let us consider the construction of the ensemble of rule sets E_s labelled ID:1 in the top-right part of Fig. 1. This ensemble will consist of rules predicting either class 1 or class 2, that is the set of sibling classes $S = \{1, 2\}$. So, the variable d, indicating the class to be predicted by a rule in E_s, will take on the value 1 or 2. This ensemble E_s will consist of four rule sets, each of them denoted $E_{si}, i = 1, \ldots, 4$, where the i-th rule set is constructed from examples in the i-th level of the class tree.

As can be seen in Table 1 at the first level i is set to 1. The rule induction algorithm is given the training set with examples belonging to classes (c) 1 and 2, it then returns a rule set predicting classes (d) 1 and 2 for the first rule set $E_{si}.S = \{1, 2\}, i = 1$. At the second level i is set to 2. The rule induction algorithm is given the training set with examples belonging to classes (c) 1.1, 1.2, 2.1 and 2.2 (descendants of the classes in S). It then returns a rule set with rules discriminating between these classes. The rules predicting classes 1.1 and 1.2 (E_{sid} where $i = 2$ and $d = 1$) have their consequent changed to predict class (d) 1. The rules predicting classes 2.1 and 2.2 (E_{sid} where $i = 2$ and $d = 2$) are changed to predict class (d) 2 and are added, with the other rules now predicting class 1, to the second rule set $E_{si}.$, $S = \{1, 2\}, i = 2$. At the third level i is set to 3. As there are no third level descendant classes of class 2 (in the right-hand side of Tables 1 and 2 the term "NA" means "not applicable") only rules predicting class 1 will be contained in this E_{si}. The rule induction algorithm is given a training set containing examples belonging to classes (c) 1.1.1 and 1.1.2. The rules predicting the classes 1.1.1 and 1.1.2 have their consequent class changed to predict class (d) 1. They are then added to E_{sid} where $i = 3$ and $d = 1$, which in this case is equal to E_{si} where $i = 3$. An analogous procedure (as described in Table 1) is performed at level 4 ($i = 4$) where again, because this is quite an unbalanced class tree, there are only rules predicting class (d) 1 in the rule set E_{si} where $i = 4$.

Note also that when the level i is set to 2 and the class (d) being predicted by the ensemble is 1, the rule induction algorithm produces a rule set discriminating between

Table 2 Main characteristics of the datasets used in the experiments

	GPCR/Prints	GPCR/Prosite	GPCR/Interpro	EC/Prints	EC/Prosite	EC/pfam
#Attributes	283	129	450	382	585	708
#Examples	5,422	6,261	7,461	14,038	14,048	13,995
#Classes	8/46/76/49	9/50/79/49	12/54/82/50	6/45/92/208	6/42/89/187	6/41/96/190

classes (*c*) 1.1 and 1.2 (along with 2.1 and 2.2), which at first glance seems counter-intuitive—as they are both descendants of class 1, the class (*d*) being predicted. The reason for this is to try and encourage diversity in the rules generated. As the rule induction algorithm is unaware of the hierarchical relationships between classes, the algorithm could produce the same (or very similar) rule sets for, say, the following two classification scenarios: (a) class 1 versus class 2; and (b) class 1.1 versus other non-descendant classes of class 1 (i.e. classes 2.1 and 2.2). Although it is still possible that the same or very similar rules will be generated between different levels and classes even when using the method described in this section, the probability (dependant on the make-up of the training set) of this happening is smaller, when including the examples belonging to sibling classes as negative examples when inducing rules. Recall that to make an effective ensemble it is very important that the component classifiers be diverse, even if at the expense of some accuracy (Skalak 1997; Brown et al. 2005; Battiti and Colla 1994).

3.1.2 Combining the predictions from the multiple rules in HEHRS

After the entire hierarchical ensemble of hierarchical rule sets has been induced in the training phase, all the induced rules can be used to predict the class of a new example in the test set. In this testing phase, in order to combine the predictions of the rules in the hierarchical ensemble into a single predicted class at each level of the class tree for a given test example, each rule in the ensemble is assigned a weight. That is, for each ensemble of rule sets E_s, each rule in E_s is assigned a weight.

The weight of a rule is a measure of its predictive accuracy. When the class predicted by a rule is the majority class (a class having more examples than the rest of the training set combined) its weight is computed by the product of the rule's sensitivity and specificity (Hand 1997), as shown in Eq. 1, where *TP*, *FN*, *FP* and *TN* are, respectively, the number of true positives, false negatives, false positives and true negatives associated with the rule (Witten and Frank 2005).

When a rule predicts a minority class (i.e. any class different from the majority class) the precision (Hand 1997), shown in Eq. 2, is used as the rule's weight. This approach, based on measuring rule quality either as the product of sensitivity and specificity or as precision, depending on the relative frequency of the class predicted by the rule, is an attempt to get a more "balanced" weight in extreme cases.

When there are a small number of examples in the class being predicted, when compared to the overall size of the training set, then the way in which specificity accounts for the number of false positives becomes problematic. This is because sensitivity multiplied by specificity weights the sensitivity ($TP/(TP+FN)$) and the specificity ($TN/(TN+FP)$) equally, ignoring the actual number of true positives and false positives. Therefore, in the case where the minority class is being predicted, it is possible to obtain a good rule quality even though the ratio of TP/(TP+FP)—that is the precision—is bad. Such a situation will likely produce a low accuracy as although a high sensitivity and relatively high specificity may be obtained, many examples may be misclassified as this minority class (due to the absolute number of false positives). The opposite is true for the majority class; the absolute number of false positives becomes less important for producing good accuracy as the number of true positives will likely be much higher. Sensitivity multiplied by specificity increases the importance of obtaining a low number of false positives when compared to precision. This is because it is more useful to consider the ratio $TN/(TN+FP)$, rather than ratio of the large number of true positives to the low possible number of false positives (as it is implicitly the case with precision). For a more detailed discussion see Holden and Freitas (2006).

$$Sensitivity \times Specificity = TP/(TP+FN) \times TN/(TN+FP)$$

Equation 1: Rule Weight (Majority Class)

$$Precision = TP/(TP+FP)$$

Equation 2: Rule Weight (Minority Class)

During the testing phase, a test example is classified in a top-down fashion, as follows. Let S be the set of sibling classes out of which one class must be assigned to the example. Initially, S contains the set of classes in the first class level. For each class d in S, the weight of class d is given by the summation of the weights of all the rules in the ensemble of rule sets E_s that cover the test example and predict class d. The class with the greatest weight is assigned to the test example at the first level. Next the example is pushed down to the second level, where the set S is updated to contain the child classes of the class assigned to the example in the first level—the ensemble E_s is also updated accordingly. Again, for each class d in the current S the weight of class d is computed—adding the weights of all rules in the current E_s that cover the test example and predict class d—and the class with the greatest weight is assigned to the test example at the current (second) level, and so on. This process is repeated until the test example reaches a leaf node in the class tree.

To validate a prediction made by HEHRS for a given example a simple procedure can be implemented. After the example has been fully classified to the leaf level it is possible to examine all the rules that covered it. All the rules that have consequent classes that are parents of the final leaf classification can be used to present an overview of the classification process to the user. Taking the union of the terms in the antecedents of these rules will produce a combined rule that will show why the example has been classified to that particular leaf class node. In the same fashion it would be possible to show such a combined rule for each separate ensemble used in the HEHRS tree.

3.2 Optimising HEHRS' rule weights with particle swarm optimization (PSO)

As computed by Eqs. 1 and 2, the weight of a rule in HEHRS depends only on the predictive accuracy of that individual rule, and it does not take into account the complex interactions of the rules in an ensemble. It is possible to optimise the set of rule weights by taking rule interaction into account, by defining two elements:

(a) An evaluation function that measures the quality of a candidate set of rule weight values. The evaluation function to be maximised is the normalised total number of correct predictions made at each internal (non-leaf) class node and each leaf class node.

(b) An optimization method, which searches for the optimal set of rule weight values in the space of candidate weight values. In this work we use, as an optimization method, a PSO algorithm.

Particle swarm optimization is a meta-heuristics that maintains a population of particles—each of them a candidate solution to the target problem—that iteratively move around the search space (Kennedy et al. 2001). The position of a particle in the search space represents the contents of its candidate solution, and so moving the particles correspond to generating new candidate solutions. First, each particle is initialised with randomly deviating (±1) position generated from the rule weight equations and random velocity. Each particle keeps track of the best position it has ever held, according to the evaluation function. At each iteration, each particle finds its best neighbour (in a local or global neighbourhood). The particle then moves towards a combination of its best neighbour's position and its own best ever position, with a velocity calculated as shown below. This process is repeated until a maximum number of iterations have been performed. To calculate the velocity and so the new position of a particle, Eqs. 3 and 4, are used:

$$v_{id}(t) = W^*(v_{id}(t-1)) + \varphi_1^* Rand()^*(p_{id} - x_{id}(t-1)) \\ + \varphi_2^* Rand()^*(p_{gd} - x_{id}(t-1))$$

Equation 3: A Particle's Velocity at time t

$$x_{id}(t) = x_{id}(t-1) + v_{id}(t)$$

Equation 4: A Particle's Position at time t

Where x_{id} is the particle i's position in dimension d, t is the iteration (time) index, v_{id} is particle i's velocity in dimension d, W is an inertial constant to prevent the particle gaining too much speed. φ_1 and φ_2 are user-defined personal and social learning constants, respectively. p_{gd} is the best position of the particle's neighbours in dimension d and p_{id} is the best position particle i has ever held in dimension d. In addition to W, a maximum velocity is also used to prevent the particle from flying out of the search space. *Rand* () generates a random number in $[0 \ldots 1]$.

The main motivations for using PSO is that it performs a global search in the search space (rather than the greedy search performed by local search algorithms), and has been empirically shown to be a powerful optimizer, often outperforming more traditional population-based optimizers such as evolutionary algorithms (EAs) (Kennedy and Spears 1998; Mouser and Dunn 2004). In any case, we do not claim that PSO is the "optimal" algorithm for our rule weight optimization problem. It produced very good results—as will be shown later—but it is possible that other global search optimization methods such as EAs would produce a similarly good result. The issue of comparing PSO and EAs is out of the scope of this paper, and is left for future research.

Two versions of the PSO for rule weight optimization are proposed in this work, one where *negative weight values are allowed* and another one where *they are not*. In the former case, if a rule is extremely unreliable it may be assigned a negative weight, detracting from the class predicted by that rule. An example of where a negative value may be appropriate for a rule is where that rule covers more examples of other classes than its own consequent class and so, in fact, signals that other classes are more likely. In the version where negative values are not allowed, the lowest possible rule weight is 0, where a rule will not have any influence in the classification of a test example.

The two main elements of the proposed PSO for rule weight optimization are the particle representation and the fitness function. The particle representation consists of a vector with n components, each of them denoted w_i, $i = 1, \ldots, n$, where w_i is the weight associated with i-th rule and n is the total number of rules. That is:

$$Particle = w_1, w_2, \ldots, w_n$$

The fitness function measures the quality of a particle, that is the quality of a candidate set of rule weights. In order to compute the fitness of a particle, for each example in the training set, the system extracts the rule weights from the particle and uses those weights to decide which class will be assigned to the example. This decision is made by computing, for each class, the total weight of rules that cover the example and have that class, as discussed earlier. The class chosen to be assigned to the example is the class with the largest total weight. After every training example has been completely classified (i.e. assigned a class at a leaf node in the class tree), the value of the fitness function for the current particle is the classification accuracy on the training set, this is the average accuracy across all four class levels.

In some cases it does not matter what the weights associated with certain rules are during the training phase, for instance if all examples are always correctly classified by all rules, then as long as the weights are all positive it does not matter what the weight values are. This can cause a problem, as even though all examples are correctly classified by all rules during the training phase they may not be during the testing phase. Therefore during the testing phase the exact weights may become important. To combat this situation it is detected whether any rules do not take part in any contentions (where two or more rules predict different classes for any given example) during the training phase, if they do *not* they will *not* have their weights optimised and default to the normal rule weights. Such contentions (or lack of) can be detected by assigning a flag to each rule (with a default value of *off*); the sets of rules covering each example can then be examined. If any set of rules contain rules with different consequent classes then the contention flag is set to *on* for those rules, meaning that the weight for those rules should be optimised. The rules left with a flag of *off* should not have their weight optimised.

Note that, ideally, the fitness function should be based on the classification accuracy on a hold out set, that is the original training set should be divided into a building set (used to build the rules) and a validation, hold out set, used to compute the classification accuracy to be used as the fitness of a particle. This would have the advantage of avoiding overfitting of the rule weights optimised by the PSO to the training set. However, it was not feasible to use such a hold out set in our experiments, due to the sparseness of data at lower levels of the class tree. It would be impossible to induce rules for some classes if examples from the training set were reserved for a hold out set. We consider the benefits of creating rules for all classes outweigh the problems due to possible overfitting from the lack of a hold out set. Initial tests confirmed this hypothesis as the decrease in overall predictive accuracy due to being unable to induce rules for some classes was quite severe.

3.3 Rule-based extended multiplicative method

This method is derived from a method proposed by Sun et al. (2004) to reduce the problem of blocking in hierarchical multi-label classification. The blocking problem was described by Sun et al. in the following way. Each class node in the class tree is associated with a probabilistic classifier, learned during the training phase. In the testing phase, an example with unknown class is classified in a top-down fashion, as follows. For each class node in the first level of the class tree, the example is assigned that class if the corresponding classifier predicts that class with a probability greater than a predefined threshold. An example is said to be rejected by a classifier if the probability of the example having the class predicted by the classifier is smaller than or equal to the threshold. For each of the (parent) classes assigned to the example at the first level, the example is pushed down the class tree to the child class nodes of those parent classes. Then, for each of those child classes the example is either assigned that child class or is rejected by the corresponding classifier depending on the probability of that class as computed by the classifier, etc. This top-down classification process is repeated until the example reaches the leaf nodes of the class tree. In this context, blocking occurs when an example is wrongly rejected by a classifier in an internal (non-leaf) node of the class tree, and so the example can never be shown to the classifiers that are descendants of the classifier that made the wrong rejection. As a result, the example can never be correctly classified at class levels deeper than d, where d is the level of the classifier that wrongly rejected the example.

One of the methods proposed by Sun et al. to cope with the blocking problem consists of assigning an example to a leaf class in the class tree if the multiplied probabilities of the example belonging to the internal (non-leaf) classes along the path from the root node to the leaf class node exceed a certain threshold. The authors called their approach the extended multiplicative method (EMM).

Note that in Sun et al.'s work an example can be assigned to more than one class at each hierarchical level, which is characteristic of multi-label classification problems. This is not the case in the data sets used in this paper, where a single class must be assigned for each level. In addition, EMM was proposed in the context of probabilistic classifiers, which again is not the case in our work, where the classifier consists of a set of IF-THEN classification rules.

Therefore, we adapted EMM to the context of our work, where the classification of test set examples is performed by classification rules and we must assign only one class label per hierarchical level for each example. In this context, there is no need for the threshold used by EMM, since a testing example is simply assigned the best predicted class at each hierarchical class level. In addition, note that different leaf class nodes can be at different depths in the class tree. Hence, just multiplying the probabilities along each path from the root to a leaf class node is not appropriate because, when we compare the probabilities associated with different leaf classes in order to choose the best leaf class to be assigned to the testing example, shallower leaf class nodes would have an advantage over deeper ones—given the reductive nature of multiplying positive numbers smaller than 1. Furthermore, there is no innate sense of probabilistic matching given a (non-fuzzy) rule-based classifier, so it is natural to use a measure of rule quality instead of probabilistic matching.

Given this discussion, our variant of EMM, called Rule-based EMM, finds the best "path" consisting of a series of rules discovered by HEHRS—one rule for each class level. It considers every possible path by considering not only the best rule covering the current test example at each class level, but all possible rules that cover the current test example. The best path is considered to be the path with the highest value of the geometric mean of all the rule weights along the path from the root to the class leaf node, as given by Eq. 5.

$$PathQuality = \sqrt[l]{w_1 \times w_2 \times \ldots w_l}$$

Equation 5: EMM for Rules Path Quality

Where $PathQuality$ is the score for a certain path and $w_i, i = 1, \ldots, l$, is the weight associated with the rule covering the example at class level i in that path and l is the number of rules that cover the example (i.e. the number of class levels) in that path. The formula used to compute each rule weight is given by Eqs. 1 and 2.

4 The creation of the bioinformatics data sets

The hierarchical classification methods proposed in the previous section were evaluated in six challenging real-world datasets involving the prediction of protein function. The protein functional classes to be predicted in these data sets are the functional classes of G-protein-coupled receptors (GPCRs) or Enzymes.

G-protein-coupled receptors are proteins involved in signalling. They span cell walls so that they influence the chemistry inside the cell by sensing the chemistry outside the cell. More specifically, when a ligand (a substance that binds to a protein) is received by a GPCR, it causes the attached G-proteins to activate and detach, this is a mechanical biological switch that causes the released G-Protein to affect other reactions within the cell. This kind of protein is particularly important for medical applications because it is believed that 40–50% of current drugs target GPCR activity (Fillmore 2004). Enzymes are another subset of proteins; they are catalysts which are used to speed up and make possible many of the chemical reactions that take place within the cell, without being altered themselves during the reaction. They are usually very specific and only catalyse one type of reaction within the cell. Often they can be turned on and off by another ligand. This is used to control both the speed of reaction and the course of overall reaction pathways that take place within the cell.

The protein functional classes are given unique hierarchical indexes by GPCRDB (2006, http://www.gpcr.org/7tm/, Visited on July) in the case of GPCRs and by Enzyme Nomenclature (2006, http://www.chem.qmul.ac.uk/iubmb/enzyme/, Visited on July) (Enzyme Commission Codes) in the case of enzymes. In the case of GPCRs, examples (proteins) have up to five class levels, but only 4 levels are used in the datasets created in this work, as the data in the 5th level is too sparse for training—that is in general there are too few examples of each class at the fifth level. In any case, it should be noted that predicting all the first four levels of GPCR's classes is already a challenging task. Indeed, most works on the classification of GPCRs limit the predictions to just the topmost or the two topmost class levels (families and subfamilies but not groups, etc.) Bhasin and Raghava (2004); Guo et al. (2006); Karchin et al. (2002); Papasaikas et al. (2003). All four levels of the Enzyme Commission Codes are used in the created Enzymes data sets.

The data used in our experiments was constructed from data in UniProt (2006, http://www.ebi.uniprot.org/, Visited on July) and GPCRDB (2006, http://www.gpcr.org/7tm/, Visited on July). UniProt is a well known biological database, containing sequence data and a rich annotation about a large number of different kinds of proteins. It also has cross-references for other major biological databases such as Prosite (2006, http://us.expasy.org/prosite/, Visited on July), Prints (2006, http://umber.sbs.man.ac.uk/dbbrowser/PRINTS/, Visited on January), Pfam (2006, http://www.sanger.ac.uk/Software/Pfam/, Visited on July) and Interpro (2006, http://www.ebi.ac.uk/interpro/, Visited on July) (McDowall 2005) (see below). It was extensively used in this work as a source of data for creating the data sets used in our experiments. Only the UniProtKB/Swiss-Prot was used as a data source, as it contains a higher quality, manually annotated set of proteins. Unlike Uniprot, GPCRDB is a biological database specialised on GPCR proteins.

We did experiments with four different kinds of predictor attributes, each of them representing a kind of "protein signature", or "motif", namely: FingerPrints from the Prints database, Prosite patterns, Pfam and Interpro entries. Prosite patterns are regular expressions describing short fragments of protein sequences. Such patterns are especially good at detecting highly conserved functional regions like catalytic sites in enzymes (Hulo et al. 2004), as they do not allow partial hits. They also have the advantage of being comprehensible to the user. In other words, in general, a protein either contains or does not contain a Prosite pattern, involving an "all-or-nothing" matching. However due to this rigidity there tend to be a large number of false negatives associated with each Prosite pattern (McDowall 2005). Pfam entries are different from Prosite patterns in that they employ Hidden Markov Models rather than regular expressions. Prints uses motifs in a similar way to Prosite, however it contains multiple non-overlapping motifs in a single entry. Prints therefore provides a more flexible descriptive language for a protein signature. Another difference is that Prosite patterns usually correspond to functional regions, whilst it is often the case that a Prints motif (FingerPrint) refers only to a highly conserved region with no specific function. Interpro integrates several protein motif databases into one.

We created six data sets to evaluate the proposed hierarchical classification methods, three GPCR data sets and three Enzyme data sets. For the GPCR data sets the predictor attributes were Prints, Prosite and Interpro entries and the protein's molecular weight and sequence length. For the Enzyme data sets the predictor attributes were Prosite, Interpro and Pfam entries and the protein's molecular weight and sequence length.

Any duplicate examples (proteins) in a data set are removed in a pre-processing step, that is before the hierarchical classification algorithm is run, to avoid redundancy. For both GPCR and Enzyme data sets, if there are fewer than 10 examples in any given class in the class tree that class is merged with its parent class. If the parent class is the root node, the entire small class is removed from the data set. This process ensures there is enough training and test data per class to carry out the experiments. (If a class had less than ten examples, during the tenfold cross-validation procedure there would be at least one iteration where there would be no example of that class in the test set, an undesirable situation.) Any binary attribute that has a value which occurs in only one example is removed from the corresponding data set, since these binary attributes in general do not have a good predictive power. An initial random sample of 15,000 enzymes from the UniProt database was used to generate the enzyme data sets. Less than the original 15,000 examples occur in the final data sets because of the duplicate and small class removal process.

After data pre-processing, the final datasets used in the experiments have the numbers of attributes, examples (proteins) and classes per level (expressed as level 1/level 2/level 3/level 4) indicated in Table 2.

5 Computational results

This section reports computational results evaluating the methods proposed in Sect. 3 in the created datasets described in Sect. 4. Recall that Sect. 3 proposed 3 types of hierarchical classification methods, namely:

(a) Hierarchical ensemble of hierarchical rule sets with rule weights computed by equations 1 and 2;

(b) HEHRS with rule weights optimized by PSO—two versions of the PSO were proposed, with and without a lower limit of 0 for the rule weights; these two versions are hereafter referred to as LimPSO–HEHRS and PSO–HEHRS, respectively. Both versions of PSO are a "vanilla" PSO (Kennedy et al. 2001) with standard parameter settings (Clerc and Kennedy 2002): $W = 0.73$, $\varphi_1 = \varphi_2 = 2.05$.

(c) The Extended Multiplicative Method adapted for rule-based (rather than probabilistic) classifiers—hereafter called Rule-EMM for short.

These methods are compared against a baseline method, namely the standard top-down approach for hierarchical classification. This approach consists of simply running a rule induction algorithm at each internal (non-leaf) node of the class tree, as described in Sect. 2. In the proposed and baseline methods the base rule induction algorithm used in our experiments was the well-known Ripper algorithm Witten and Frank (2005).

Throughout the entire set of experiments tenfold cross validation (Witten and Frank 2005) is used. Since PSO is a stochastic method, the PSO-HEHRS and Lim PSO-HEHRS methods are run ten times each—with different random seeds used to create the initial population in each run—for each one of the 10 folds. As the remainder of the methods are deterministic, they are run just once for each of the tenfolds.

Tables 3, 4, 5, 6, 7 and 8 show the predictive accuracy that the different methods achieved in each data set during tenfold cross validation. The numbers after the "±" symbol are standard deviations. In these tables a cell is coloured dark grey if there is a statistically significant win of the method in the corresponding column against the baseline method, according to a two-tailed Student's t test with significance level of 0.05. The t test used is WEKA's implementation of Nadeau and Bengio's corrected re-sampled t test (UniProt 2006, http://www.ebi.uniprot.org/, Visited on July). This more conservative corrected t test takes into account the ratio of training and test examples in an attempt to limit the number of significant results occurring by chance. A cell is coloured light grey if there is a statistically significant loss when compared to the baseline method. Table 9 shows the cumulative scores—calculated based on the results of the Student's t test—for each method, at each class level, for all data sets. For each data set, in Table 9 one is added to the score of each cell if its corresponding method (indicated by the column label) at the corresponding class level (indicated by the row label) significantly beats the baseline approach in that data set. One is deducted from the score in the cell for a loss against the baseline approach in the same manner. The totals in the bottom row of the table are simply the summed results—over all data sets—from each class level for each method.

Table 3 Predictive accuracy (%) with Prints attributes and GPCR classes

Class level	Rule-EMM	PSO-HEHRS	LimPSO-HEHRS	HEHRS	Baseline
1	91.0±0.65	91.5±0.8	91.3±0.83	90.6±0.41	91.2±0.74
2	65.1±1.25	82.0±1.09	81.7±1.06	77.9±0.46	80.3±1.12
3	37.5±0.84	56.1±1.43	56.1±1.2	55.1±0.95	53.5±1.5
4	44.0±3.49	83.1±3.03	83.0±2.78	82.1±2.33	78.3±2.53

Table 4 Predictive accuracy (%) with InterPro attributes and GPCR classes

Class level	Rule-EMM	PSO-HEHRS	LimPSO-HEHRS	HEHRS	Baseline
1	90.2±0.69	91.0±0.71	91.1±0.76	89.7±0.3	90.3±0.71
2	68.5±0.79	83.3±0.97	82.9±0.82	79.1±0.47	81.1±0.74
3	36.4±1.03	55.2±1.33	55.4±1.15	54.6±1.16	52.8±0.87
4	46.0±2.86	86.9±1.78	86.6±1.81	86.5±2.23	82.4±2.65

Table 5 Predictive accuracy (%) with Prosite attributes and GPCR classes

Class level	Rule-EMM	PSO-HEHRS	LimPSO-HEHRS	HEHRS	Baseline
1	87.4±0.88	87.8±0.62	87.5±1.0	86.3±1.36	87.6±0.92
2	49.8±1.18	63.5±1.77	62.9±1.91	61.5±1.79	63.9±1.43
3	18.1±0.59	32.2±1.74	32.3±1.91	29.5±1.62	29.3±1.56
4	12.8±2.39	45.5±3.18	45.5±3.93	36.5±2.46	35.4±1.84

Table 6 Predictive accuracy (%) with Prints attributes and enzyme classes

Class level	Rule-EMM	PSO-HEHRS	LimPSO-HEHRS	HEHRS	Baseline
1	48.9±2.41	97.8±0.34	97.8±0.41	96.7±0.35	97.4±0.28
2	33.5±2.61	95.0±0.47	95.2±0.67	93.3±0.29	94.6±0.46
3	32.8±1.03	94.1±0.34	94.3±0.65	90.1±0.97	93.8±0.54
4	29.7±0.91	93.4±0.69	93.7±0.79	93.3±0.75	92.8±0.87

Table 7 Predictive accuracy (%) with Pfam attributes and enzyme classes

Class level	Rule-EMM	PSO-HEHRS	LimPSO-HEHRS	HEHRS	Baseline
1	37.0±0.24	98.0±0.2	98.0±0.32	92.3±1.01	95.8±1.84
2	23.3±0.8	96.2±0.43	96.3±0.37	88.7±1.07	94.0±2.04
3	23.5±0.74	94.9±0.5	94.9±0.45	87.6±1.01	92.6±2.26
4	23.5±0.75	96.0±0.48	96.1±0.33	95.1±0.89	94.5±1.19

Table 8 Predictive accuracy (%) with Prosite attributes and enzyme classes

Class level	Rule-EMM	PSO-HEHRS	LimPSO-HEHRS	HEHRS	Baseline
1	40.7±0.4	98.7±0.3	98.7±0.24	96.6±0.48	98.5±0.24
2	28.1±0.42	97.4±0.45	97.3±0.41	94.1±0.27	97.1±0.42
3	26.2±0.44	96.2±0.39	96.0±0.34	92.4±0.45	95.9±0.19
4	23.3±0.44	95.2±0.34	95.3±0.41	95.2±0.42	95.0±0.42

Tables 10 and 11 show the un-weighted and weighted—respectively—misclassification costs associated with each experiment. The misclassification cost is computed by finding the shortest path in the class tree from the predicted class node to the actual class node. In the case of the weighted misclassification cost this path is then weighted (the values of the edges of the path are added), with edges between the root node and the first class level given a weight of 0.26, the edges between the first a second class level given a weight of 0.13, between the second and third a weight of 0.07 and between the third and fourth class levels a weight of 0.04. The reason for this weighting is to assign a higher cost to more general misclassifications. These general errors are more serious than the finer grained errors at lower levels of the class tree, as if a general error is made, no information about the true class of an example is gained.

Table 9 Overall performance according to WEKA's Student's *t* test, when compared to baseline

Overall Scores Against Baseline – Best possible score for each cell in the first 4 rows is 6 (number of data sets)				
Class level	Rule-EMM	PSO-HEHRS	LimPSO-HEHRS	HEHRS
1	-3	3	2	-4
2	-6	4	3	-6
3	-6	4	4	-1
4	-6	4	5	4
Totals	-21	15	14	-7

Table 10 The un-weighted misclassification cost, with Student's *t* tests comparing each approach against the baseline

Data Set	Rule-EMM	PSO-HEHRS	LimPSO-HEHRS	HEHRS	Baseline
GPCR Prints	28.3±0.78	18.72±0.62	18.88±0.69	20.76±0.33	19.86±0.61
GPCR Interpro	25.53±0.5	17.13±0.51	17.19±0.5	19.21±0.31	18.44±0.36
GPCR Prosite	37.62±0.66	30.8±0.74	31.21±0.79	32.83±1.13	31.43±0.94
Enzyme Prints	60.5±1.74	4.78±0.39	4.67±0.55	6.74±0.4	5.2±0.42
Enzyme Pfam	69.45±0.51	3.68±0.29	3.65±0.26	9.68±0.91	5.73±1.83
Enzyme Prosite	66.59±0.41	2.99±0.29	3.07±0.22	5.68±0.32	3.21±0.23
Accumulative *t*-test score against baseline	-6	5	3	-6	

Table 11 The weighted misclassification cost, with Student's *t* tests comparing each approach against the baseline

Data Set	Rule-EMM	PSO-HEHRS	LimPSO-HEHRS	HEHRS	Baseline
GPCR Prints	22.24±0.65	15.1±0.65	15.31±0.75	17.01±0.3	15.98±0.66
GPCR Interpro	20.62±0.53	14.3±0.63	14.35±0.65	16.38±0.23	15.51±0.53
GPCR Prosite	30.31±0.77	24.91±0.7	25.33±0.95	26.87±1.29	25.25±0.96
Enzyme Prints	57.74±2.16	3.73±0.36	3.66±0.49	5.4±0.27	4.16±0.32
Enzyme Pfam	68.23±0.43	3.04±0.25	3.0±0.27	9.33±0.96	5.21±1.89
Enzyme Prosite	64.62±0.38	2.2±0.31	2.27±0.23	4.9±0.37	2.42±0.24
Accumulative *t*-test score against baseline	-6	4	3	-6	

In the case of the un-weighted misclassification score, each edge is assigned a weight of one. The misclassification score for each example is then normalised by dividing the number of edges between the predicted and the actual class nodes in the class tree by the number of edges in the worst possible score for that example. The latter can be found by finding the weight (number of edges) from the actual class to any leaf node via the root node, and taking the largest weight as the worst possible misclassification score. The scores from every test example classification are then added and divided by the total number of test examples to give the final misclassification score. The accumulative *t* test score at the bottom of Tables 10 and 11 shows the number of times the corresponding method is significantly better (+1) or worse (−1) than the baseline approach across all the experiments. The misclassification costs shown in Tables 10 and 11 are useful as (unlike the accuracy rates shown in Tables 3, 4, 5, 6, 7, 8, 9) they take into account the hierarchical structure of the classes, and so they provide a way to quickly assess the performance of a hierarchical approach. They can also be tailored to concentrate on general or fine grained errors using weighting.

Let us first analyse the results with respect to accuracy rate (Tables 3, 4, 5, 6, 7, 8, 9). The pure HEHRS—without rule weights optimised by PSO—achieved a disappointing performance: it obtained an overall score of −7. Observing both Table 9 and the more detailed results per dataset in Tables 3, 4, 5, 6, 7 and 8, one can see that, in all the six datasets, HEHRS obtains results significantly worse than the baseline method's results in the first two (shallower) class levels. On the other hand, in all the six datasets HEHRS obtains results significantly better than the baseline method's results in the fourth (deepest) class level. The consistency of these results is interesting, considering that the six datasets contain very different numbers of attributes and examples, as well as different kinds of biological motifs as predictor attributes—as indicated in Table 1.

The poor performance of HEHRS is likely due to its bias towards deeper classes. As it predicts a class based upon the addition of rule weights, classes that are deeper will have more nodes and so more weights when compared to shallower ones. This explanation is supported by the differences seen between the GPCR and Enzyme data sets. In the GPCR data sets the number of examples in each class is quite unbalanced, with one class having a large portion of the examples, this is even more so the case at lower levels. This is an advantage for HEHRS at the lower levels (3 and 4) because it tends to try and classify more examples as the deeper class, which also happens to be one of the largest. The classes are more balanced in the enzyme data set but again the bias towards deeper classes still reaps rewards in the fourth level.

One method of dealing with this bias would be to average the rule weights rather than adding them. However, it is likely that this would cause the opposite problem in HEHRS—a bias towards shallower classes. This is because, in general, rules at deeper class levels tend to have lower qualities, due to the higher number of classes and lower number of examples per class. Hence, the averaging process would favour the classes with fewer descendants, giving fewer and higher weights. Investigating the effect of this averaging process empirically could be a topic for future research.

The Rule-EMM method achieved by far the worst results, significantly losing to the baseline method in 21 out of 24 cases. This very bad performance is most likely due to the way in which a decision list is generated by the rule induction algorithm and its interaction with the EMM approach. The Rule-EMM method is reliant on not choosing only the best matching rule (as in RIPPER), but all rules that match the test example at all (at each class level) in a rule list. This is the trade off needed when attempting to find all possible paths to class leaf nodes. The trade off does not seem to pay off with the current rule induction algorithm, RIPPER. It is possible that if the rules produced by the rule induction algorithm were unordered the misclassifications would become less of a problem, since unordered rules tend to be more modular than ordered rules. Investigating this hypothesis is an interesting topic for future research.

In general the best performing methods in terms of predictive accuracy are LimPSO–HEHRS and PSO–HEHRS, with the version of PSO *without* a lower limit on the rule weights (PSO–HEHRS) beating the PSO version with a lower limit (LimPSO–HEHRS) by only one test. Both methods obtained a good performance, with an overall score of 15 or 14, respectively—the maximum possible score is 24 (4 class levels × 6 datasets).

These conclusions derived from the analysis of accuracy rates are also reflected in general in the misclassification costs, with HEHRS and Rule-EMM getting the same overall negative score against the baseline and the two versions of the PSO getting an overall positive score against the baseline. Also, when the finer grained misclassifications are weighted more evenly (as with the un-weighted misclassification costs) the difference between LimPSO–HEHRS and PSO–HEHRS becomes more apparent, with PSO–HEHRS outperforming LimPSO–HEHRS significantly in two out of six tests.

Using the PSO to optimise rule weights has the disadvantage that a PSO run is very computationally expensive. On a machine with a P4 3.0 GHz CPU it takes about five hours to optimise the weights for the rules generated from a single 10 times tenfold cross validation run (depending on the number of rules). Also HEHRS itself requires more computational time as many more rule sets must be induced using larger training sets (when compared to the baseline approach). On the same machine the models for a single run of the baseline approach are induced within ten minutes, whereas the HEHRS models take up to an hour on the larger datasets. These models do not variate between approaches and so can be cached, increasing efficiency when comparing multiple approaches. However, note that maximising classification accuracy is usually considered more important than minimising the processing time taken by a classification algorithm. This is particularly the case in real-world scenarios like the bioinformatics problems addressed in this work, where the time taken by a run of the PSO algorithm is a very small fraction of the time that was spent in preparing our datasets for data mining purposes (about 4 months). This scenario is also often found in other data mining applications, where most of the time taken by the entire knowledge discovery process is spent preparing data.

6 Conclusions and future research

This work proposed new hierarchical classification methods that use characteristics of hierarchical class data (where the classes are arranged in a tree structure) to try to improve predictive accuracy, with respect to a standard top-down hierarchical classification method. More precisely, three main types of hierarchical classification methods were proposed, namely: (a) HEHRS (Hierarchical Ensemble of Hierarchical Rule Sets), a method based on exploiting the hierarchical nature of the data to create different training sets to be given as input to a bagging-like ensemble method; (b) two versions of a PSO method for optimising the rule weights used by HEHRS to classify test examples; and (c) Rule-EMM, the rule-based version of the Extended Multiplicative Method, which tries to reduce the problem of misclassifications at shallower class levels leading to misclassifications at deeper class levels in the standard top-down approach for hierarchical classification.

Out of these three types of methods, the pure HEHRS method and Rule-EMM produced disappoint results, in general significantly worse than the standard top-down approach. However, the development of a PSO algorithm to optimise rule weights for HEHRS was very effective, leading to a hierarchical classification system that obtained, overall, predictive accuracies significantly better than the accuracies obtained by the standard top-down approach. These results were to a large extent consistent across 6 different bioinformatics datasets involving the hierarchical classification of protein functions, a set of challenging real-world bioinformatics problems with large numbers of predictor attributes and large numbers of classes to be predicted. Indeed, in this work we predicted GPCR and Enzyme classes up to the fourth level of the class hierarchy, whilst most of the literature addresses the less challenging problem of predicting GPCR and Enzyme classes only at the first and sometimes second level of the class hierarchy (Bhasin and Raghava 2004; Guo et al. 2006; Karchin et al. 2002; Papasaikas et al. 2003).

There are several potential avenues for future research. Since optimising the rule weights used by HEHRS with the PSO method proved to be very effective, perhaps the rule weights used by Rule-EMM could be optimised in just as an effective way. In addition, as mentioned earlier, it would be interesting to investigate the performance of Rule-EMM when the base rule induction algorithm used to discover classification rules produces an unordered rule set, rather than an ordered rule list (see Sect. 5). Also, it would be interesting to investigate if the performance of HEHRS can be improved by averaging (rather than adding) rule weights, as mentioned in Sect. 5.

References

Battiti R, Colla AM (1994) Democracy in neural nets: voting schemes for accuracy. Neural Netw 7:691–707

Bhasin M, Raghava GP (2004) GPCRpred: an SVM-based method for prediction of families and subfamilies of G-protein coupled receptors. Nucleic Acids Res. 1 32(Web Server issue):383–9

Blockeel H, Schietgat L, Struyf J, Dzeroski S, Clare A (2006) Decision trees for hierarchical multilabel classification: a case study in functional genomics. In: Proceedings of PKDD 2006

Breiman L (1996) Bagging predictors. Mach Learn 24:123–140

Brown G, Wyatt J, Harris R, Yao X (2005) Diversity creation methods: a survey and categorisation. Inf Fusion 6(1):5–20

Bulashevska A, Eils R (2006) Predicting protein subcellular locations using hierarchical ensemble of Bayesian classifiers based on Markov chains. BMC Bioinformatics 7:298

Clare A (2003) Machine learning and data mining for yeast functional genomics. PhD Thesis, University of Wales Aberystwyth

Clerc M, Kennedy J (2002) The particle swarm-explosion, stability and convergence in a multidimensional complex space. IEEE Transactions on Evolutionary Computation 6(1)

Derbeko P, El-Yaniv R, Meir R (2002) Variance Optimized Bagging. In: Proceedings of 13th European conference on machine learning, pp 60–71

Dietterich TG (1997) Machine learning research: four current directions. AI Magaz 18(4):97–136

Eisner R, Poulin B, Szafron D, Lu P, Greiner R (2005) Improving protein function prediction using the hierarchical structure of the gene ontology. In: Proceedings of 2005 IEEE symposium on computational intelligence in bioinformatics and computational biology

Fayyad UM, Piatetsky-Shapiro G, Smyth P (1996) From data mining to knowledge discovery: an overview, Advances in Knowledge Discovery and Data Mining. AAAI/MIT, Menlo Park, pp 1–34

Fillmore D (2004) It's a GPCR world. Modern Drug Discovery 11(7):24–28

Guenter S, Bunke H (2004) Optimization of weights in a multiple classifier handwritten word recognition system using a genetic algorithm. ELCVIA(3) 1:25–44

Guo YZ, Li ML, Wang KL, Wen ZN, Lu MC, Liu LX, Jiang L (2006) Classifying G protein-coupled receptors and nuclear receptors on the basis of protein power spectrum from fast Fourier transform. Amino Acids 30(4):397–402

Günter S, Bunke H (2004) Evaluation of classical and novel ensemble methods for handwritten word recognition. In: Proceedings of 10th international workshop on structural and syntactic pattern recognition (SSPR), pp 583–591

Hand DJ (1997) Construction and assessment of classification rules. Wiley, New York

Holden N, Freitas AA (2006) Hierarchical classification of G-protein-coupled receptors with a PSO/ACO algorithm. In: Proceedings of IEEE swarm intelligence symposium (SIS-06). IEEE Press, New York, pp 77–84

Huang Y, Cai J, Ji L, Li Y (2004) Classifying G-protein coupled receptors with bagging classification tree. Comput Biol Chem 28(4):275–280

Hulo N, Sigrist CJA, Le Saux V, Langendijk-Genevaux PS, Bordoli L, Gattiker A, De Castro E, Bucher P, Bairoch A (2004) Recent improvements to the PROSITE database. Nucleic Acids Res 32(Database issue):D134–D137

Karchin R, Karplus K, Haussler D (2002) Classifying G-protein coupled receptors with support vector machines. Bioinformatics 18(1):147–59

Kennedy J, Spears W (1998) Matching algorithms to problems: an experimental test of the particle swarm and some genetic algorithms on the multimodal problem generator. In: IEEE international conference on evolutionary computation, May

Kennedy J, Eberhart RC, Shi Y (2001) Swarm intelligence. Morgan Kaufmann/Academic Press, San Francisco/New York

McDowall J (2005) InterPro: Exploring a Powerful Protein Diagnostic Tool, ECCB05, Tutorial, pp 14

Mouser CR, Dunn SA (2004) Comparing genetic algorithms and particle swarm optimisation for an inverse problem exercise. In: Computational techniques and applications conference (CTAC-2004), September

Papasaikas PK, Bagos PG, Litou ZI, Hamodrakas SJ (2003) A novel method for GPCR recognition and family classification from sequence alone using signatures derived from profile hidden Markov models. SAR QSAR Environ Res 14(5–6): 413–420

Ranawana R, Palade V (2005) MVGen—Ensemble learning for mcs majority voting with a genetic algorithm , internal report. Oxford University Computing Laboratory

Skalak DB (1997) Prototype selection for composite nearest neighbour classifiers. PhD Thesis, University of Massachusetts, Amherst

Stefano CD, Cioppa AD, Marcelli A (2003) Exploiting reliability for dynamic selection of classifiers by means of genetic algorithms. In: Proceedings of 7th Internatioanl conference on document analysis and recognition, vol 2, ICDAR. IEEE Computer Society, Washington, DC, pp 671

Sun A, Lim E-P (2001) Hierarchical text classification and evaluation. In: Proceedings of 2001 IEEE international conference on data mining, pp 521–528

Sun A, Lim E-P, Ng WK, Srivastava J (2004) Blocking reduction strategies in hierarchical text classification. IEEE Trans Knowl Data Eng 16(10):1305–1308

Peng T, Zuo W, He F (2006) Text Classification from Positive and Unlabeled Documents Based on GA, vecpar 06(7)

Tan A, Gilbert D, Deville Y (2003) Multi-class protein fold classification using a new ensemble machine learning approach. Genome Inf 14:206–217

Tan P-N, Steinbach M, Kumar V (2006) Introduction to data mining. Addison-Wesley, Reading

Witten IH, Frank E (2005) Data mining: practical machine learning tools and techniques, 2nd edn. Morgan Kaufmann, San Francisco