

# Clonal Selection Algorithm for Evolving Agregated Linear Classifiers

Michał Bereta<sup>1</sup> and Tadeusz Burczyński<sup>1,2</sup>

<sup>1</sup> Institute of Computer Modeling, Cracow University of Technology, Cracow, Poland

<sup>2</sup> Department for Strength of Materials and Computational Mechanics, Silesian  
University of Technology, Gliwice, Poland

## Abstract

Artificial immune systems (AIS) have become popular among researchers and have been applied to variety of tasks. Developing supervised learning algorithms based on metaphors from the immune system is still an area in which there is much to explore. In this paper a novel supervised immune algorithm based on clonal selection framework is proposed. It evolves a population of linear classifiers used to construct a set of classification rules. Aggregating strategies such as bagging and boosting are presented to work well with the proposed algorithm as the base classifier.

**Keywords:** artificial immune systems, clonal selection, linear classifiers, bagging, boosting

## 1 Introduction

This work presents a novel immune algorithm, which is an example of applying the general framework of clonal selection with suppression based on usefulness to create new computational techniques. Clonal selection is one of the most important paradigms of artificial immune systems (AIS) (Dasgupta, 1998). There are several characteristics of this algorithm that contribute to its novelty and originality. It assumes hyperplanes as the form of lymphocytes representation. It evolves a population of hyperplanes, which solve a multiclass classification task. It is a supervised learning algorithm, as it makes use of class information in the form of samples' class labels during training. In its basic form it is formulated for two-classes case, then it is extended to multiclass case. The final classification is based on voting among rules, which use weighted attributes and are created by means of the evolved linear classifiers. It is formulated as a clonal selection algorithm. All steps are defined with respect to the assumed representation of lymphocytes as linear classifiers. Linear classifiers (artificial lymphocytes) are learnt to solve the problem not only by competition, but first of all by cooperation, which is due to the proper formulation of the suppression step. The idea of suppression based on usefulness is utilised. It was proposed in (Bereta and Burczyński, 2007) and further developed in (Bereta and Burczyński, 2006). Proper definition of the

*usefulness* of a given lymphocyte for the whole population is crucial for the evolution of the population with desired properties. This approach is different to the commonly used suppression based on similarity, in which if there are too similar lymphocytes, the worst of them are removed. As it was the case in the cited works, also in this paper it is presented that suppression based on *usefulness* of the lymphocytes allows to develop populations of different, interesting properties.

The rest of the work is structured as follows. First, the idea of a linear classifier is briefly reminded. Next, the proposed algorithm is described in detail. The generalization ability of the proposed method is discussed and possible ways of improvement are proposed, such as committee voting, bagging and boosting. Finally, very promising results on several benchmark databases are presented.

## 2 Linear Classifiers

Let us assume that the training data consist of  $N$ -dimensional samples and the class labels are given in the form of  $T = \{(\mathbf{x}_i, y_i), i = 1, \dots, M\}$ , where  $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{iN})$  is the vector of real valued features (attributes),  $M$  is the number of samples in the training set, and  $y_i \in \{1, -1\}$  is the class label, into which  $x_i$  belongs. Linear classifiers are natural discriminators in the case when the samples are linearly separable, i.e. there exists a hyperplane  $\mathbf{w} = \{w_0, w_1, \dots, w_N\}$  such that  $\mathbf{w} \cdot \mathbf{x}_i + w_0 > 0$  and  $\mathbf{w} \cdot \mathbf{x}_i + w_0 < 0$  for all  $\mathbf{x}_i$  from class 1 and  $-1$ , respectively. The element  $w_0$  of  $\mathbf{w}$  is often referred to as *bias*. As it is always possible to extend the training input vector  $\mathbf{x}_i$  to  $\mathbf{x}'_i = (1, \mathbf{x}_i) = (1, x_1, x_2, \dots, x_N)$ , the following notation can be assumed, without losing the generality of the discussion:  $\mathbf{w} \cdot \mathbf{x}_i > 0$  for all  $\mathbf{x}_i$  from class 1, and  $\mathbf{w} \cdot \mathbf{x}_i < 0$  for all  $\mathbf{x}_i$  from class  $-1$ . In the case of data which is not linearly separable, one single plane which separates all training samples cannot be found. The most common approach to deal with nonlinear problems is to use of a nonlinear transformation of the original problem to the new feature space, most likely with more dimensions, in which the problem is linearly separable. The transformation can be achieved for example by means of appropriate kernel function (SVM) or multiple layers in neural networks (e.g., MLP, RBF). The problem with this approach is the choice of the appropriate transformation, it means, the proper kernel function, the proper number of hidden neurons, etc. There exists no easy solution for these problems.

The approach proposed is different. The goal is to find a set (a population) of linear classifiers, among which none can solve the problem separately, but only in cooperation with other classifiers. Fig. 1 shows the idea. The data presented in Fig. 1 is not linearly separable, i.e., there exists no single line which separates these two classes. However, without any nonlinear transformation, it is possible to separate the two classes with two lines,  $\mathbf{w}_1$  and  $\mathbf{w}_2$ . The final classification has to take into account on which side of *each* line a given sample is. It can be viewed as performing two **linear** transformations, and constructing a set of rules of the following type: **IF**  $\mathbf{w}_1 \cdot \mathbf{x}_i > 0$  **AND**  $\mathbf{w}_2 \cdot \mathbf{x}_i < 0$  **THEN**  $y_i = -1$ . The output of the algorithm proposed is a population of linear classifiers, which can serve as a base for creation of such classification rules. The proposed algorithm is well suited within the general framework of clonal selection with the suppression

based on properly defined *usefulness*. As in all previous algorithms based on this framework (Bereta and Burczyński, 2007; Bereta and Burczyński, 2006), an important characteristic of the proposed algorithm is the ability to determine the population size dynamically during evolutionary process, i.e., for simple problems a small populations are expected. In classification problems, in the case of linearly separable data, a population consisting of only one lymphocyte (one hyperplane) is expected.

Clonal selection paradigm has been relatively seldom used to create supervised learning algorithms for multiclass classification problems. The most known, successful and influential ones are (Carter, 2000; Watkins, 2005; Watkins *et al.*, 2004). The proposed technique proves that the creation of such algorithms is not only possible, but also can bring very satisfying results. Even more rarely linear classifiers have been used in artificial immune systems as artificial lymphocytes. The work presented in (Ando and Iba, 2003) seems to be an exception. It is, however, completely different than that presented here.

### 3 Immune Hyperplane Algorithm

In order to solve wide range of classification problems, also nonlinear, the proposed algorithm, Immune Hyperplane Algorithm (IHA), evolves a population of linear classifiers, which solve the problem through cooperation, i.e., none of them is globally optimal. They are locally optimal, and cooperate globally. It means that suppression should remove classifiers, which do not contribute to the global solution. This general definition of usefulness is, however, not so obvious to formulate in detail. In the case of linear classifiers, none of them has class label appointed to it, as none of them solely can decide about the final classification. The problem is, that the influence of a single linear classifier spread through the whole space, i.e., it divides the whole space into two subspaces. Thus, as each classifier has to be locally tuned, checking the number of training samples which are correctly classified by this classifier does not work. As an example, consider the situation presented in Fig. 1. The line 1 is definitely better than the line 2, in the sense that it makes less misclassifications globally. On the other hand, both lines are equally important from the point of view of cooperative solution of the problem. This example explains why suppression which takes into account the total number of misclassifications of a given hyperplane, is not a good choice. Favoring hyperplanes which make few errors produces globally optimal planes, which try to solve the problem independently, which is not the goal. In this place it is worth mentioning, that during evolution of the population of hyperplanes, no classification rules (as presented earlier) are created. The rules are created after the evolutionary process, based on already existing linear classifiers.

Fortunately, there is a straightforward way to define the usefulness of a given hyperplane lymphocyte. To achieve that, the algorithm does not work on original data, but rather on samples matched in pairs. Assuming two-class classification case, in each pair one sample is from the 1 class and the other is from  $-1$  class. The original set of training samples  $T = \{(\mathbf{x}_i, y_i), i = 1..M\}$  is used to create a new training set in the following form:

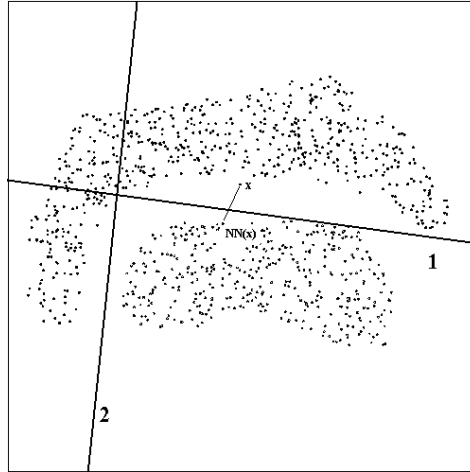


FIGURE 1: Line 1 is better than line 2 (it makes less misclassifications), however, both lines have to cooperate to separate the data. Example  $\mathbf{x}$  is misclassified by line 2, however the pair  $(\mathbf{x}, NN(\mathbf{x}))$  is not misclassified and does not decrease the usefulness of line 2.

$$T_{pairs} = \{(\mathbf{x}_i, y_i), (\mathbf{x}_j, y_j) : i = 1..M, j \in \{1..M\}, NN(\mathbf{x}_i) = \mathbf{x}_j, y_i \neq y_j\} \quad (1)$$

where  $NN(x_i)$  is the nearest neighbor of  $x_i$ . It means that for each training sample, its nearest neighbor from the other class is found and the pair constructed from these samples is added to  $T_{pairs}$ . This form of representation of the training data has very useful aftermath. It is very easy to observe, that by counting the number of correctly classified pairs, it is possible to assess the usefulness of a given linear classifier in a way which leads the population to the desired configuration. A given pair  $\{(\mathbf{x}_i, y_i), (\mathbf{x}_j, y_j)\} \in T_{pairs}$  is correctly classified by a given linear classifier  $\mathbf{w}$  if  $signum(\mathbf{w} \cdot \mathbf{x}_i) = y_i$  and  $signum(\mathbf{w} \cdot \mathbf{x}_j) = y_j$ , where  $signum(u)$  returns 1 if  $u > 0$  and  $-1$  if  $u \leq 0$ . Counting the number of correctly classified pairs for each plane does not have some of the disadvantages of counting the total number of misclassifications. For example, line 1 in Fig. 1 intersects several segments appointed by pairs in  $T_{pairs}$ . Training sample  $\mathbf{x}$  is misclassified by line 2, however, the segment appointed by  $\mathbf{x}$  and its  $NN$  is not intersected by line 2, thus, it is considered neither as error nor as correct classification of this pair, and it does not decrease the usefulness of line 2.

To summarize, a given lymphocyte (hyperplane) is considered as useful for the whole population, if there exists at least one pair from  $T_{pairs}$ , which is correctly classified only by this lymphocyte. If there exists no such pair, the linear classifier is considered as useless. It means that if the subtask of the overall classification task performed by the given lymphocyte can be distributed among other individuals in the population, the given classifier can be removed without any loss in the performance of the whole system. On the other hand, a given classifier can be

considered as useful even if it is a very poor global classifier, i.e., there are more misclassifications than correct classifications.

### 3.1 Two classes case

In this section, a basic formulation of the proposed algorithm is given. The training examples  $\mathbf{x}_i$  from set  $T$  are considered to be vectors of real valued features (attributes). The basic formulation of the IHA is for the two-classes case. The proposed algorithm goes as follows:

#### IMMUNE HYPERPLANES ALGORITHM

*T*: A set of training samples from two classes, 1 and  $-1$ .

*Pop*: Population of lymphocytes, i.e., linear classifiers.

*LC<sub>i</sub>*: *i*-th linear classifier in *Pop*.

*startPopSize*: The starting size of *Pop*.

*mutRange*: The parameter of mutation.

*clonesNum*: The number of clones of each *LC<sub>i</sub>*.

*newNum*: The number of newly generated linear classifiers in each iteration.

#### START

1. Create  $T_{pairs}$  set.
2. Generate the starting population *Pop* of size *startPopSize*.
3. For each *LC<sub>i</sub>* in *Pop*,  $i = 1, \dots, |Pop|$ , check which pairs from  $T_{pairs}$  are correctly classified by *LC<sub>i</sub>*.
4. Sort *Pop* in the descending order according to the number of correctly classified pairs from  $T_{pairs}$ .
5. Create *clonesNum* clones for each *LC<sub>i</sub>* in *Pop*,  $i = 1, \dots, |Pop|$ .
6. Mutate each clone created in step 5 and check correctly classified pairs from  $T_{pairs}$  by each mutated clone.
7. Add *newNum* newly generated linear classifiers into *Pop* and check correctly classified pairs from  $T_{pairs}$  by each newly generated classifier.
8. Sort *Pop* in the descending order according to the number of correctly classified pairs from  $T_{pairs}$ .
9. Perform suppression. Starting from the worst (i.e., the one with the smallest number of correctly classified pairs from  $T_{pairs}$ ) *LC<sub>i</sub>* in sorted *Pop*, for each *LC<sub>i</sub>* temporarily remove it from *Pop*. If there exist no loss in the total number of correctly classified pairs by all classifiers in *Pop*, remove *LC<sub>i</sub>* definitely.
10. Repeat from step 5 until termination condition is satisfied.

#### END

After creation of  $T_{pairs}$  set, a random initial population is generated. To alleviate the learning process, the generation process assures that the generated linear classifier separates at least two learning samples, one from each class. This is achieved by randomly selecting two training samples,  $\mathbf{x}_{-1}$  and  $\mathbf{x}_1$ , from class  $-1$  and  $1$ , respectively. Next, the new weight vector  $\mathbf{w}_i$  of the *i*-th generated linear classifier, *LC<sub>i</sub>*, is constructed as a hyper plane perpendicular to the direction of vector  $\mathbf{x}_1 - \mathbf{x}_{-1}$ , and intersecting the middle point between  $\mathbf{x}_{-1}$  and

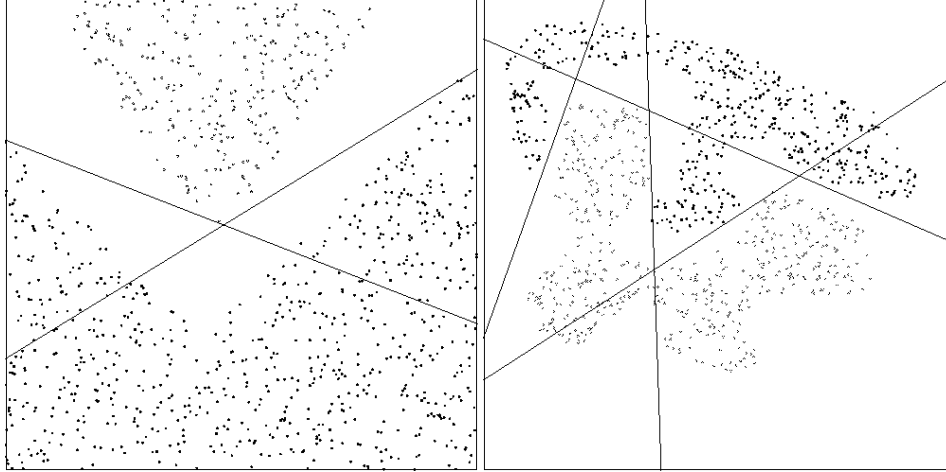


FIGURE 2: Example results of Immune Hyperplane Algorithm applied to 2D artificial data (classification problems with two classes). The proposed method is able to find the proper population size for the given problem.

$\mathbf{x}_1$ . From that  $\mathbf{w}_i = \mathbf{x}_1 - \mathbf{x}_{-1}$  and bias of  $\mathbf{w}_i$ ,  $w_{i0}$  is determined as  $w_{i0} = -\sum_{j=1}^N w_{ij} * (x_{-1j} + x_{1j})/2$ . The new lymphocytes generated in step 7 are constructed in the same way. In all experiments presented in this paper, it is assumed that every  $LC_i$  produces the same number of clones, regardless of the number of correctly classified pairs from  $T_{pairs}$ , which could be, in fact, interpreted as stimulation level and used to promote better classifiers by allowing them to produce a bigger number of clones. It is not used here, however, as the algorithm works well without this distinction. The mutation is performed in a very simple manner. Parameter  $mutRange$  describes the maximum allowed decrease or increase of the value of each  $w_{ij}$  of a given  $\mathbf{w}_i$ , i.e.  $w_{ij} = w_{ij} + mutRange * (2 * random(0, 1) - 1)$ , where  $random(0, 1)$  is a random value uniformly sampled from the range  $[0, 1]$ ,  $j = 0, \dots, N$  (bias is also mutated).

The results of applying the proposed algorithm to artificial 2D data sets are presented in Fig. 2. Each example is a two–classes classification problem. The algorithm is able to find the appropriate number of lines, that separates the classes cooperatively.

### 3.2 Dealing with noise

The formulation of the algorithm from section 3.1 is very optimistic. The naive assumption was that even in the case of nonlinear boundaries between classes, the samples from different classes are not mixed, i.e., the classes distributions do not overlap and there is no noise or incorrect labels in the training set. If the distributions do overlap or some incorrect labels exist in the training set, the construction of the set  $T_{pairs}$  becomes erroneous, in the sense, that there are pairs from  $T_{pairs}$  entirely included inside one of the classes. That leads to populations

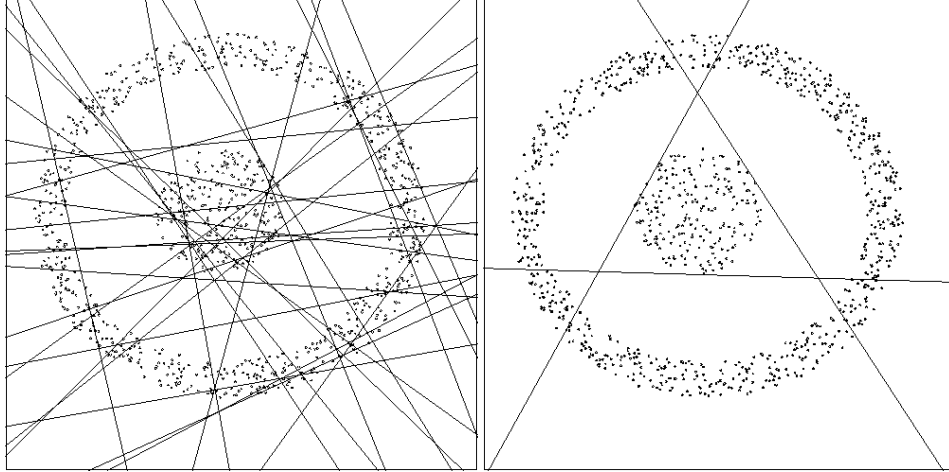


FIGURE 3: The population evolved on data with added 5% of noise (left). The population evolved on noisy data with modified method of  $T_{pairs}$  construction (right).

in which many classifiers are adapted only to these noisy pairs. This undesirable effect is illustrated in Fig. 3, in which two classes with artificially added noise are presented, together with a population of evolved linear classifiers. Note, that there are many unnecessary lines, which is due to the existence of misleading pairs.

Fortunately, there is a simple modification that can be introduced into the algorithm that allows to limit the undesirable effect of the misleading pairs. Such pairs can be in many cases discovered during the first step of the algorithm, in which  $T_{pairs}$  set is constructed. This procedure is modified as follows. For each  $\mathbf{x}_i$  its nearest neighbor among samples from the other class is found, as previously. Additionally, the nearest neighbor from both classes, i.e., from the whole training set  $T$ , is found, too. If both nearest neighbors are in fact the same training sample,  $\mathbf{x}_i$  does not create **any** pair which could be used during training and no new element is added to  $T_{pairs}$ . Although this simple procedure does not eliminate all undesirable pairs, it definitely makes IHA very resistant to noisy samples. Fig. 3 presents an example result of applying modified IHA to noisy data. Note, that the modification has no effect if there is no noise present in the data.

### 3.3 Classification rules

As it was already mentioned, a final classification of an unknown sample by means of a set of hyperplanes requires checking on which side of every hyperplane a given sample is. Assuming that there is  $K$  hyperplanes in the final population in IHA, a Immune Hyperplane Classifier (IHC) can be constructed, which is represented as a set of rules of a type

$$\text{IF } \mathbf{w}_1 \cdot \mathbf{x}_i > 0 \text{ AND } \mathbf{w}_2 \cdot \mathbf{x}_i < 0 \text{ AND } \dots \mathbf{w}_K \cdot \mathbf{x}_i > 0 \text{ THEN } y_i = -1 \quad (2)$$

Having the population of hyperplanes already evolved, it is relatively easy to create a set of such rules. The procedure is straightforward. Each training sample  $\mathbf{x}_i \in T$ ,  $i = 1, \dots, M$  is checked against each linear classifier  $LC_j$ ,  $j = 1, \dots, K$  whether it satisfies a condition  $\mathbf{x}_i \cdot \mathbf{w}_j > 0$ . Thus, a premise of a  $l$ -th rule is in the form of a vector of boolean values  $\mathbf{Premise}_l = (p_{l1}, p_{l2}, \dots, p_{lK})$ , where  $p_{lj} = true$  if  $\mathbf{x}_i \cdot \mathbf{w}_j > 0$  and  $p_{lj} = false$  otherwise,  $j = 1, \dots, K$  and the conclusion of the  $l$ -th rule,  $Concl_l$  is the class label, i.e., either 1 or  $-1$ . Additionally, each rule has an additional field,  $Perc_l$ , in which a percentage of correctly classified training samples by this rule is stored. After the training sample  $\mathbf{x}_i$  is checked against all  $LC_j$ , a vector  $tempPremise$  of binary values is formed. If it is equal to a premise of already existing,  $l$ -th, rule and  $y_i = Concl_l$ , i.e.,  $\mathbf{x}_i$  is correctly classified by the  $l$ -th rule, then  $Perc_l$  is increased properly. If the class labels do not match or if there is no rule with a premise equal to  $tempPremise$ , a new rule is constructed with the premise  $tempPremise$  and class label  $y_i$ . It means, that there can exist rules with the same premises but different conclusions (different class labels). This is not a problem, as each rule has its  $Perc$  value which allows to decide which rule should make the decision about the final classification, in the case of a test sample which satisfies both rules. Fig. 4 presents classes boundaries found by means of the set of rules constructed as described.

It can be observed even in these 2D examples, that during testing of unknown sample  $\mathbf{x}_{test}$  with the set of rules, it is possible that  $tempPremise$  constructed by checking  $\mathbf{x}_{test}$  against each  $LC_j$ , does not match to a premise of any known rule. Obviously, it is possible to use the idea of partial matching, popular in many artificial immune algorithms. If  $tempPremise$  of  $\mathbf{x}_{test}$  is interpreted as an antigen and  $Premise_l$  as antibody receptor, it is possible to check how many single conditions in  $Premise_l$  and  $tempPremise$  match. The final classification procedure is defined as follows:

1. For a given  $\mathbf{x}_{test}$  construct  $tempPremise$  by checking  $\mathbf{x}_{test}$  against all  $LC_j$ ,  $j = 1, \dots, K$ .
2. Find a rule which premise exactly matches to  $tempPremise$ . If there are at least two such rules, choose that with the highest value of  $Perc$ . Return  $Concl$  of the chosen rule. If there is no exactly matching rule, go to step 3.
3. Find a rule that best matches  $tempPremise$ , i.e., the number of single conditions that match in the premises, is the biggest. If there are at least two such rules that match  $tempPremise$  equally well, chose that with the highest value of  $Perc$ . Return  $Concl$  of the chosen rule.

This approach allows also for signaling that a given test sample  $\mathbf{x}_{test}$  is classified, but with a very low confidence level, i.e., it is classified by a rule with a very weak match to  $tempPremise$  generated by  $\mathbf{x}_{test}$ . a threshold can be set to the matching level, in order to signal the impossibility of classification rather than a classification with a very low confidence. Fig. 4 presents the classes boundaries found by the aforementioned approach with partial matching.

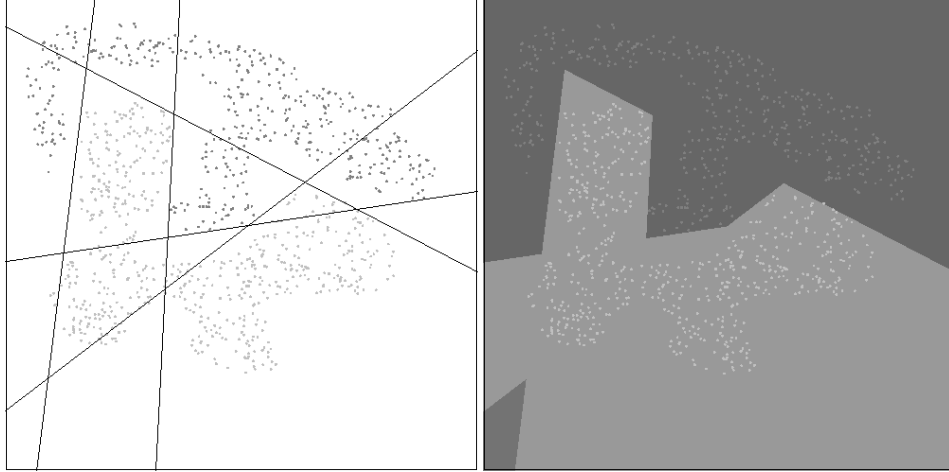


FIGURE 4: The results of applying classification rules with partial matching constructed by means of evolved populations of hyperplanes (lines in 2D).

### 3.4 Multiclass case

The aforementioned formulation of Immune Hyperplane Algorithm deals with classification problems with only two classes. It can be, however, easily extended to the multiclass case. Let us consider a classification problem in which a training set  $T$  consists of samples belonging to  $C$  different classes, i.e.,  $T = \{(\mathbf{x}_i, y_i), i = 1, \dots, M\}$ , where  $y_i \in G$ . The set  $G = \{1, 2, \dots, C\}$  is a set of the possible class labels. The idea of solving classification problem with  $C$  possible classes by means of the proposed IHA is to create one classifier for each class treated as class 1 and all samples from the rest of the classes as representing class  $-1$ . It will result in a set of  $C$  two-class IHC. During the classification of an unknown sample  $x_{test}$ , each  $j$ -th basic IHC,  $j = 1, \dots, C$ , tries to classify  $x_{test}$  as 1 or as  $-1$ , as it was described in the previous section. First, only the exact matches are considered. The answer 1 of  $j$ -th classifier means that  $j$ -th IHA classifier classifies  $x_{test}$  into  $j$ -th class. The answer  $-1$  means that the  $j$ -th IHA classifier classifies  $x_{test}$  as *not belonging to the  $j$ -th class*. If there are more basic classifiers that give answer 1, the final decision is made based on the *Perc* values, i.e., the one with the highest *Perc* wins. In the case of no exact matches, partial matches are considered. Note, however, that each basic IHC can have rules with premises of different lengths. This comes from the fact, that each of the original classes can be separated from all others by a different number of hyperplanes and  $x_{test}$  has to be checked, in general, against different number of hyperplanes. For example, if one of the  $C$  classes can be linearly separable from all others, its rules will have premises of length 1 (assuming that IHA correctly finds a single separating hyperplane), while in the case of nonlinear class boundaries, more than one hyperplane is needed to separate classes, thus, the premise part of the rules have length greater than 1. Because of that, if there are no exact matching and partial matching has to be used, the matching level of

different IHC cannot be compared directly, but rather a percentage of the exact match should be used. For example, if one of the IHCs has premise of length 4 and in one of its rules there are 2 conditions that match  $x_{test}$ , and in the case of other IHC with premise of length 9 there are 3 matches, it is better to choose the first case as there is 50% of the ideal match, and only 33% in the second case.

### 3.5 Generalization Issues

In IHA, if there is no rule that correctly matches a new sample to be classified, a rule with the highest, but incomplete matching, is selected. This can be viewed as a limited generalization ability, this however does not solve the problem. Even if the modified procedure of constructing  $T_{pairs}$  set helps with dealing with noise in IHA, it is clear that the algorithm is constructed to maximize the overall number of correctly classified pairs. No evolutionary pressure is put upon the future generalization ability on test data.

Classification trees are **unstable** classifiers, i.e. slightly different training sets lead to different classifiers. It is similar to the proposed IHA. Differences in training sets result in different  $T_{pairs}$  to which IHA adapts. There exists yet another source of instability. The evolution is stochastic and after each run, different populations, and thus different sets of rules, are achieved. In general, although after each run one can separate the training data even ideally, the potential future classification error on test data is expected to have great variability, i.e., it is unstable. The instability of classifiers is not necessarily their drawback. It turned out that such classifiers are especially suited for constructing families of classifiers, through techniques such as bagging or boosting. The easiest way to limit the variability of a single classifier from IHA is to combine them in family **F** and use simple majority voting to make the final decision. If there are two classes with the same number of votes, a sum of all *Perc* values of IHC classifiers voting for a given class are used, and the class with the highest value wins.

The techniques bagging and boosting come from the attempts to answer the question whether weak classifiers (so called weak learners) can be used to form a better classifier through the aggregation of the weak ones. Weak classifier is a classifier that gives not much better results than random classification. However, assuming that there are  $C$  **statistically independent** weak classifiers the expected performance can be improved. Having enough classifiers one can count on correct classification by means of plurality voting. The problem is that having one training set  $T$  it is impossible to produce a set of independent classifiers. However, creating multiple training sets by randomly sampling the original training set and using them to train weak classifiers can bring significant improvements, although the classifiers are not independent. **Bagging** states for *bootstrap aggregating*, a method for generating multiple versions of classifiers and aggregating them into an aggregated predictor, proposed by Breiman (Breiman, 1996). Bagging can improve accuracy if there is a big variability in the base classifier, i.e., if perturbing the training set can cause significant changes in the predictor constructed. For that reason, commonly used as base classifiers in bagging, are classification trees, due to their instability, which is their advantage in this case. Each replicate training data set is formed by sampling the original training set at

random (with uniform probability), but with *replicates*. The bagging procedure is independent of the base classifier. That allows to use the proposed IHC as the base classifier. This seems to be justified, as IHC are also unstable. The expected result would be decreased variability of IHC, even more than due to the procedure of constructing a simple family of IHC. This was empirically verified and the results are presented in section 4. Boosting is also a general method for improving any type of classifier. The most striking similarity to bagging is that the base classifier is constructed several times by means of a different training set generated from the original training set. Unlike in bagging, in boosting the probabilities of each training sample to be selected to the consequent training sets is not constant. The probability of being selected is increased after each call to base classifier training, for samples that were hard to learn for the previous base classifiers. One of the first and most popular boosting algorithms is **AdaBoost** (Schapire, 2001), which was used in the experiments presented in this work. Random Forests (RF) were proposed by Breiman (Breiman, 2001) as aggregated classifiers especially designed for the classification trees as the base (weak) classifiers. They are able to obtain great generalization results. For that reason RF are used in the next section for comparison.

## 4 Results and Conclusions

In this section preliminary results obtained by the proposed immune algorithm are presented and compared with the results obtained by Random Forest classifier. The proposed IHC was used as a base classifier in different strategies for constructing aggregated classifiers. These were: simple committee with majority voting, bagging, boosting (all with 50 base classifiers), bagging and boosting of committees of IHC (both with 50 committees, each consisting of 5 IHC). Several benchmark databases from UCI Repository were used for testing. The classifiers were tested by means of 5-fold crossvalidation. The tests were repeated 20 times and result were averaged (with an exception of Diabetes database, which was average over 10 runs). Standard deviations were also calculated. Table 1 summarises the results.

The results are very promising. Not only the proposed immune algorithm is able to perform comparable with the well established Random Forest (RF) algorithm, but it also outperforms RF on some of the databases. Even if the Random Forest algorithm turns out to perform better, the differences are not significant. Different aggregating strategies seem to be better suited for different databases. From the results obtained it is obvious that all aggregating strategies tested work with the proposed immune algorithms very well. This is observed in significantly better results of all aggregating classifiers when compared to a single IHC, and, secondly, in the decrease of variation of the single IHC due to the aggregation (which was expected). The important results prove that it is possible to construct supervised learning algorithms based on immune metaphors (clonal selection in this case). Being competitive even to Random Forests classifiers, they can become an important alternative in many real world classification problems.

TABLE 1: Results of 5-fold cross-validation tests of the proposed aggregated classifiers. The classifiers are: single IHC (1), committee of IHCs (2), bagging with IHC (3), boosting with IHC (4), bagging with committees of IHCs (5), boosting with committees of IHCs (6), Random Forest (7). The results are presented in the form of *MeanTestError/Std.Dev.OfTestError*. (Ionos. and Diab. stands for Ionosphere and Diabetes, respectively.)

DB	1	2	3	4	5	6	7
Iris	6.1/1.5	4.8/0.8	<b>4.3/0.7</b>	4.4/0.8	<b>4.3/1.0</b>	<b>4.3/1.0</b>	4.9/0.9
Sonar	27.0/2.3	18.2/1.8	19.8/1.9	16.8/2.1	19.2/1.6	<b>15.9/2.0</b>	18.3/2.2
Ionos.	17.1/1.5	11.7/1.0	11.5/0.8	8.0/0.9	11.9/0.7	7.3/0.9	<b>6.9/0.4</b>
Glass	46.1/2.8	33.0/2.0	32.5/1.4	30.5/1.8	32.3/1.5	29.8/1.9	<b>22.4/1.8</b>
Diab.	28.1/1.7	24.2/0.9	<b>23.7/1.0</b>	26.1/0.8	23.9/0.5	25.8/1.3	23.8/0.7
Bupa	37.9/2.8	30.0/1.8	29.1/1.8	32.2/2.3	28.7/1.5	32.4/2.1	<b>28.2/1.7</b>
Vowel	62.8/4.3	36.5/1.7	38.2/1.6	33.2/1.9	38.8/1.1	<b>31.1/1.7</b>	46.3/1.7

## References

- S. ANDO and H. IBA (2003), Artificial Immune System for Classification of Gene Expression Data, in E. Cantú-Paz ET AL., editor, *GECCO 2003*, volume 2724 of *Lecture Notes in Computer Science*, pp. 1926–1937, Springer, ISBN 3-540-40603-4.
- M. BERETA and T. BURCZYŃSKI (2006), Immune K-means: A novel immune algorithm for data clustering and multiple-class discrimination., in *Evolutionary Computation and Global Optimization 2006. 2006. Prace Naukowe, Elektronika, Warsaw Univ. of Technology Publishing House*, pp. 49–60.
- M. BERETA and T. BURCZYŃSKI (2007), Comparing binary and real-valued coding in hybrid immune algorithm for feature selection and classification of ECG signals, *Eng. Appl. Artif. Intell.*, 20(5):571–585, ISSN 0952-1976.
- L. BREIMAN (1996), Bagging predictors, *Mach. Learn.*, 24(2):123–140, ISSN 0885-6125.
- L. BREIMAN (2001), Random Forests, *Mach. Learn.*, 45(1):5–32, ISSN 0885-6125.
- J.H. CARTER (2000), The Immune System as a Model for Pattern Recognition and Classification, *Journal of the American Medical Informatics Association*, 7/1:28–41.
- D. DASGUPTA (1998), *Artificial Immune Systems and Their Applications*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, ISBN 3540643907.
- R. SCHAPIRE (2001), The boosting approach to machine learning: An overview, URL <http://citeseer.ist.psu.edu/schapi02boosting.html>.
- A. WATKINS (2005), *Exploiting Immunological Metaphors in the Development of Serial, Parallel, and Distributed Learning Algorithms*, Ph.D. thesis, University of Kent.
- A. WATKINS, J. TIMMIS, and L. BOGGESS (2004), Artificial Immune Recognition System (AIRS): An Immune-Inspired Supervised Machine Learning Algorithm, *Genetic Programming and Evolvable Machines*, 5(3):291–317, URL [citeseer.ist.psu.edu/watkins03artificial.html](http://citeseer.ist.psu.edu/watkins03artificial.html).