Credit Card Fraud Detection with Artificial Immune System

Manoel Fernando Alonso Gadi[1,2], Xidi Wang[3] and Alair Pereira do Lago[1]

[1] Departamento de Ciência de Computação
Instituto de Matemática e Estatística
Universidade de São Paulo
05508-090, São Paulo, SP, Brazil
+55 11 3091-6135
[2] Grupo Santander, Abbey National plc, Milton Keynes, United Kingdom
[3] Citibank, São Paulo, Brazil
manoel.gadi@abbey.com,xidi.wang@citi.com,alair@ime.usp.br

Abstract. We apply Artificial Immune Systems(AIS) [4] for credit card fraud detection and we compare it to other methods such as Neural Nets(NN) [8] and Bayesian Nets(BN) [2], Naive Bayes(NB) and Decision Trees(DT) [13]. Exhaustive search and Genetic Algorithm(GA) [7] are used to select optimized parameters sets, which minimizes the fraud cost for a credit card database provided by a Brazilian card issuer. The specifics of the fraud database are taken into account, such as skewness of data and different costs associated with false positives and negatives. Tests are done with holdout sample sets, and all executions are run using Weka [18], a publicly available software. Our results are consistent with the early result of Maes in [12] which concludes that BN is better than NN, and this occurred in all our evaluated tests. Although NN is widely used in the market today, the evaluated implementation of NN is among the worse methods for our database. In spite of a poor behavior if used with the default parameters set, AIS has the best performance when parameters optimized by GA are used.

1 Introduction

In recent years many *bio-inspired* algorithms are sprouting for solving the classification problems as one can see for instance in [3]. In 1998, Neal et al. [9] developed an artificial immune system (AIS), JISYS, applied it for mortgage fraud detection, and reported some first results, still based on simulated data. In 2002, the journal Nature [10] published an article on AIS where it indicated that AIS had many kinds of applications, including the detection of fraudulent financial transactions. Even though this article previewed a possible commercial application for 2003 by a British company, we are not aware of any subsequent publication on AIS in financial fraud detection which reported good experimental results. The current paper reports our studies and application of AIS on credit card fraud detection. Moreover, in contrast to the poor performance of

AIS with the default parameters, we report here an optimized and robust set of parameters under which AIS led to the best results, even when compared to the best results from all other analyzed methods.

The lack of publicly available database has been a limiting factor for the publications on financial fraud detection [14], particularly credit card transactions. In fact, only few publications on this field bring a real contribution based on experiments. For instance, the method AdaCost [16, 6] was developed from Adaboost [15] for credit card fraud detection, and resulted in the metaheurists Cost Sensitive [5], which can be applied for many applications where there are different costs for false positive and false negative. Comparative studies between Neural Networks (NN) and Bayesian Networks (BN) in credit card fraud detection were reported [12], which favored the result of BN.

In this paper, we present our studies of AIS compared to other techniques such as BN and NN as well. In addition, we have also included comparative studies with two other methods: Decision Trees (DT) and Naive Bayes (NB). Moreover, we take into account the skewed nature of the dataset, the different costs for false positive and false negative in order to evaluate a classifier performance, as well as the need of a parametric adjustment in order to obtain the best results for every compared method.

Background: Fraud prevention is interesting for financial institutions. The advent of new technologies as telephone, automated teller machines (ATMs) and credit card systems have amplified the amount of fraud loss for many banks. Analyzing whether each transaction is legitimate or not is very expensive. Confirming whether a transaction was done by a client or a fraudster by phoning all card holders is cost prohibitive if we check them in all transactions. Fraud prevention by automatic fraud detections is where the well-known classification methods can be applied, where pattern recognition systems play a very important role. One can learn from past (fraud happened in the past) and classify new instances (transactions). In credit card business today, perhaps the most commonly used technique is Neural Networks, for example in Fair Isaac's Falcon software as claimed in its website (http://www.fairisaac.com/fic/en/product-service/product-index/falcon-fraud-manager/). In general, the NN implementation is inside a complex work-flow system which is integrated with the bank database. When a new transaction comes in, the work-flow calculates all the input variables and outputs a fraud score. Then this score is used to decide which transaction is going to be checked manually and to order its priority.

Skewed data and other discussions: Fraud detection model is among the most complicated models used for the credit card industry. Skewness of the data, search space dimensionality, different cost of false positive and false negative, durability of the model and short time-to-answer are among the problems one has to face in developing a fraud detection model. In this article we focus our attention on skewness of the data by comparing five methods[4] .

[4] The problem of taking into account the different cost between false positive and false negative during the training phase needs a special investigation which is what

Fraud Tagging: We have obtained our database from a large Brazilian bank, with registers within time window between Jul/14/2004 through Sep/12/2004. Each register represents a credit card authorization, with only approved transactions excluding the denied transactions. One applies the following rule for classifying an authorization: a transaction is considered *fraudulent* if, in the next 2 months after the date of the transaction, which is called performance period, either the client queried the transaction, or the bank distrusts it as a legitimate transaction and confirms it does not belong to the client; otherwise the transaction is tagged as *legitimate*. When an authorization is tagged as fraudulent[5], the Bank has almost 100% of certainty about this claim, but when the transaction is tagged legitimate, it cannot be affirmed this is in fact legitimate, but it can only be sure that the transaction was still not identified as fraudulent in the performance window. However, according to the Bank, at least 80% of the occurred frauds are identified as fraudulent in 2-month period.

Sampling: The sampling of transactions is done in two steps: first, one randomly samples card numbers to be analyzed in this period, irrespective to whether the card had or not a fraud transaction in the historical period; second, there is a weighted sampling of the class where 10% of legitimate transactions are selected and 100% fraudulent transactions are selected.

In the end, the database that we have received from the bank contains 41647 registers, from which 3.74% are fraudlent.

Categorization: We preprocess the database in three steps:

1. We apply statistical analysis in order to remove variables that are considered unimportant for the modeling (ex: card number). From 33 variables in the beginning we had 17 independent variables and 1 dependent variable (flag_fraud) after this phase;
2. We bind the variables. All variables but Merchant Category Code (MCC)[6] are categorized in at most 10 groups, one digit only. See Table 1.
3. We generate 9 splits (also known as samples) from the databases. Each split contains a pair of databases: 70% of transactions for development (training set), and 30% of transaction for validation (testing set, holdout sample). Table 2 shows that these splits have about the same number of frauds and legitimates transactions.

All 9 splits are subsequently converted to Weka [18] format (.arff), on which our studies are executed. The software Weka-3-4-11 is used for all of our studies and

we intend to conclude before December this year. The durability and short time-to-answer problem we intend to start to analyze next year.

[5] According to the scope of the annotated dataset provided by the Bank, we dealed with the fraud modalities *Lost/Stolen*, *Skimming*, *Mail Order*, *Account Take Over* and *Telephone Order*; and we did not manage other types like *Never Received Issuance*, *Manual Counterfeit* and *Fraud Application*.

[6] MCC got 33 categories so it could fit the number of groups of Transaction Category Code (TCC).

name	mcc	mcc_previous	zip_code	zip_code_previous	value_trans
# of categ.	33	33	10	10	10
name	value_trans_previous	pos_entry_mode	credit_limit	brand	variant
# of categ.	10	10	10	6	6
name	score	type_person	type_of_trans	# of statements	speed
# of categ.	10	2	2	4	8
name	diff_score	credit_line	flag_fraud		
# of categ.	6	9	2		

Table 1. Number of categories for each variable. *Previous* represents the value of the last transaction made for the same client.

base	1	2	3	4	5	6	7	8	9
development frauds	1,084	1,092	1,088	1,075	1,081	1,116	1,099	1,106	1,100
development legitimates	27,904	28,012	28,061	28,145	28,045	27,973	28,113	27,884	28,188
validation frauds	475	467	471	484	478	443	460	453	459
validation legitimates	12,184	12,076	12,027	11,943	12,043	12,115	11,975	12,204	11,960

Table 2. Number of frauds and legitimates in each split

the implementations used for DT, BN, NB and NN are built in Weka. The only plugged in implementation was the AIS, the AIRS2 version 1.6 (March 2006) implemented by Jason Brownlee [1], originally designed by Watkins et al. [17].

Performance measures: In order to evaluate the classifiers, we have considered the use of KS, ROC Curve, Lift Curve, Precision (Hit Rate) and Recall accuracy (Detection Rate). From conversations with fraud prevention specialists and the first results using ROC curve and Hit Rate, we found out that we would obtain more appliable results if we used a cost function in which we adopted an average cost of $ 1 for every verification, and an average loss of $ 100 for every undetected fraud. This cost function combines Hit Rate and Detection Rate in one unique measure, and evaluates the function in only one point, the applicable cut-off. This was considered to be more similar to the used practice of a fraud score than a ROC curve that compares multiple references simultaneously. If we denote *tp*, *fp* and *fn* as the number of true positives (true frauds), false positive and false negatives, the final cost is given by:

$$\$cost = \$100 \times fn + \$1 \times (fp + tp).$$

Since the received database had only 10% of legitimate and 100% of fraudulent transactions, we had to adjust the cost function to:

$$\$cost = \$100 \times fn + \$10 \times fp + \$1 \times tp.$$

Once we prepared the data, we chose the methods to compare with the optimization criteria.

2 Parameter space

In this small section we just introduce a very short description of the input parameters for the five chosen methods. A better description of these parameters can be found in the Appendix, and details about the methodologies and their parameters can be found in Weka documentations [19, 18] as well. The methods and their respective parameters are:

- NB has no parameter;
- DT has 2 parameters (C, M);
- BN has 3 parameters (D, Q, E) and 3 sub parameter (P, S, A);
- NN has 7 parameters (L, M, N, V, S, E, H);
- AIS has 9 parameters (S, F, C, H, R, V, A, E, K).

The methods NB and DT have a small parameter space. The parameter space of BN is also quite small, especially if we notice that there are few choices for many of them.

3 Optimization of parameters

The parameter spaces of the methods Decision Tree, Bayesian Network and Naive Bayes are small enough in such a way that an exhaustive exploration of all possible parameter is possible. However, this is not the case for Neural Networks and Artificial Immune Systems. In order to find an optimized parameter set for these methods, we performed a parameters set optimization based on a Genetic Algorithm (GA).

As showed in Figure 1, we start with an initial pool of 50 random executions, followed by 20 Genetic Algorithm (GA) generations. Each GA generation combines two randomly selected candidates among the best 15 from previous generation. This combination performs: cross over, mutation, random change or no action for each parameter independently. As the generation goes by, the chance of no action increases. In the end, we perform a local search around the optimized founded by GA optimization. Notice that the final solution cannot be claimed to be optimal, and it is usually not optimal, but only suboptimal.

4 Robustness of the parameters

Given a classification method M, after the parameter optimization, all optimized parameters may be independent of the split. In this case we say that this parameter set is *robust* and we name it $ROBUST(M)$.

When this does not happen, the optimization process is not as strong since the obtained optimized parameter set loses generalization power. In this case we decided to sacrifice prediction in order to gain robustness in the parameter set. In order to rewrite the optimization function that should be used in a GA algorithm, we have used a visualization procedure with computed costs for many equally spaced parameter sets in the parameter space. After defined a good optimization function, we proceeded not with another GA optimization because our time constraints, but we reused our initial runs used in the visualization, with the following kind of *multiresolution optimization* [9]:

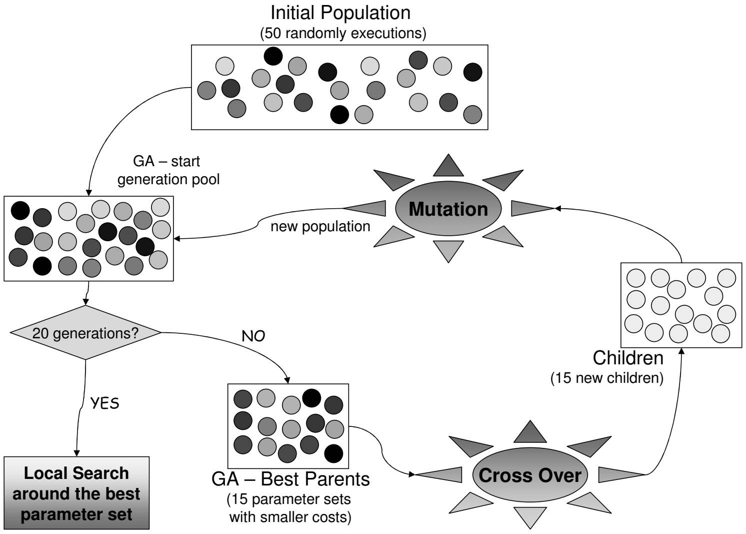

Fig. 1. Genetic Algorithm for parameters optimization

1. we identify those parameters that have not changed, and we freeze these values for these respective parameters;
2. for any other parameter we screen the 20 best parameter sets for each split and identify reasonable range;
3. for all non-robust parameters, we choose an integer step s so the the searching space does not explode;
4. we evaluate the costs for all possible combinations according to the searching space defined above, and find the parameter set P that brings the minimum average cost among all the different used splits;
5. we zoom the screen to the neighborhood of P, refine steps s, and repeat the process from then on, until no refinement is possible.

In this case, after this process, we also call this parameters set *robust* and we name it $ROBUST(M)$. We should notice that we could also have used a GA optimization instead of a multiresolution optimization like the one performed by our multiresolution optimization.

In order to run the multiresolution optimization, we elected 6 splits (2,3,4,5,6 and 7) as the *robustization split group*, and 3 others (8,9 and 1) as the *evaluation split group* for posterior evaluation and comparison of all methods.

5 Results

We compare the following five classification methods: Naive Bayes (NB), Neural Network (NN), Bayesian Network (BN), Artificial Immune System (AIS) and Decision Tree(DT). For any method M, we have applied three different strategies: $DEFAULT(M)$, $OPTIMIZED(M)$ and $ROBUST(M)$, in which $DEFAULT$ means to use default parameters provided by Weka; $OPTIMIZED$ refers to an optimized set of parameters obtained as described in Section 3, and $ROBUST$ is an optimized robust set of parameters.

One can see in Figure 2 and Table 3 the final costs of the classification

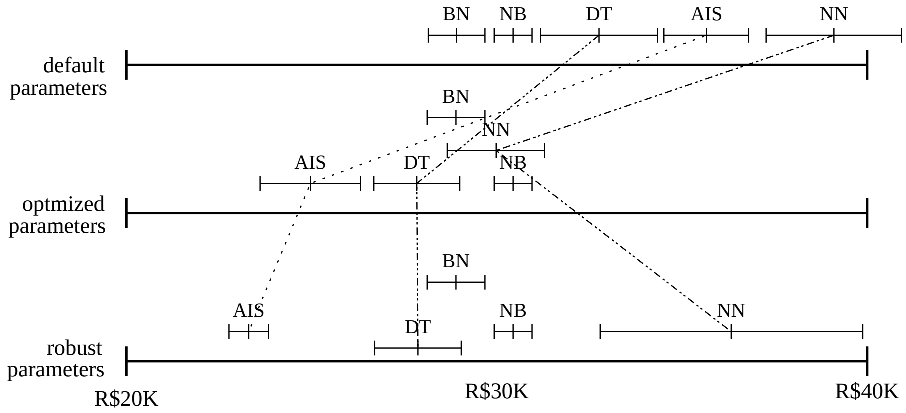

Fig. 2. Summary results for the methods in all strategies. Average and standard deviation (statistics based on the 3 evaluation splits) are represented by small error-bars, for the 5 methods, for the 3 strategies. The figure is divided in three stacked horizontal lines with their methods statistics (the error-bars) in order to separate strategies: default parameters, optimized parameters and robust parameters, in order of evolution. All 3 large horizontal lines represent the cost functions, ranging from R$ 20 thousand in the left end to R$ 40 thousand in the right end. In order to better display the error-bars, some of them were vertically shifted. AIS led to the smallest cost with robust parameters, followed by DT, and NN led to the largest cost.

Strategy	DT	AIS	BN	NN	NB
DEFAULT	32.76 (4.83%)	35.66 (3.21%)	**28.91** (2.65%)	39.10 (4.68%)	30.44 (1.68%)
OPTIMIZED	27.84 (4.16%)	**24.97** (5.43%)	28.90 (2.69%)	29.98 (4.38%)	30.44 (1.68%)
ROBUST	27.87 (4.21%)	**23.30** (2.29%)	28.90 (2.69%)	36.33 (9.75%)	30.44 (1.68%)

Table 3. Summary results for the methods in all strategies. Average and standard deviation for the 3 evaluation splits.

methods obtained for all strategies. We show here only the average costs with their standard deviations for the 3 splits used for evaluation of the robust parameter sets. The cost is represented in thousand of Reais (Brazilian Currency), the smaller, the better. The standard deviations (num%) are considered in the same way as errors. From these results one can notice that:

- The Bayesian methods BN and NB are such that their results are independent from the used strategies. This is expected for NB, since there are no parameters. For BN, the default parameters performed almost in the same way as the optimized strategies, independently from the splits. The maximum number of node parents influences the final topology and probability tables but not enough to impact the final costs;
- For strategy $DEFAULT$ we used the default parameters. BN was the best method. AIS and NN got relatively poor results compared to the others. Particularly, NN improved only 15.4%[7] in relation to a strategy which considers all transactions as legitimate;
- For what concerns the strategy $OPTIMIZED$ with optimized parameters, we verified that almost all the methods led to reduced costs in comparison to the case with default parameters. The method that reduced its cost the most, with 29.98%[8] of cost reduction, was AIS and it became the best method for this strategy. The second best method was DT, that reached a 15.01% of cost reduction. NN reduced its cost by 23.33%[9] ;
- When we analyzed the strategy $ROBUST$, we saw two important facts: first, there was an abrupt cost increase for $ROBUST(NN)$ in relation to $OPTIMIZED(NN)$, that shows the over-fitting tendency of method NN with optimized parameters. There was a cost reduction for $ROBUST(AIS)$ in relation to $OPTIMIZED(AIS)$. We suppose that this happened due to the fact that AIS has more parameters and also the largest parametric search space. In this way, when the parametric space is reduced, after the freezing of some parameters during the parameters robustization process, it can be observed a more efficient optimization. This phenomenon is many times mentioned as “Curse of Dimensionality”.

Robust set of parameters: The table 4 shows the set of optimized robust parameters for each method.

At first glance, we can observe that for DT we have a tree with minimum pruning according to parameter M. For NN, we see that the parameters L and M achieved very interesting values with a big L (Learning Rate) and very small M (Momentum). This fact allows us to trace a parallel with DT, saying that,

[7] 15.4% = \$39.1 thousands/\$46.2 thousands, where \$46.2 thousands corresponds to the average cost of the validation part of the splits 8, 9 and 1 when one simply decides letting frauds happen unwatched.

[8] 29.98% = 1 - \$ 24.97 thousands / \$ 35.66 thousands = $1 - OPTIMIZED(AIS)/DEFAULT(AIS)$.

[9] 23.33% = 1 - \$ 29.98 thousands / \$ 39.10 thousands = $1 - OPTIMIZED(NN)/DEFAULT(NN)$.

Method	Average Cost on validation	Robust parameters in command line display
DT	$ 27,870.66	-C 0.49 -M 1
NB	$ 30,439.33	n/a
BN	$ 28,901.66	-D -Q weka.classifiers.bayes.net.search.local.K2 – -P 17 -S BAYES -E weka.classifiers.bayes.net.estimate.SimpleEstimator – -A 0.5
NN	$ 36,332.33	-L 0.40 -M 0.12 -H 20 -E 0 -V 0 -N 500 -S 0
AIS	$ 23,303.00	-C 30 -R 177 -E 5 -K 1 -F 0 -H 10 -V 1 -A -1 -S 1

Table 4. Summary of optimized robust parameters. Parameters N,S for NN and A,S for AIS were not iterated. Parameters E,V for NN and K,F,H,V for AIS were frozen for the multiresolution optimization. Parameters L,M,H for NN and C,R,E for AIS needed a multiresolution optimization. Parameter H=20 in NN is the number of attributes + number of classes + 1, parameter P=17 for BN is the number of attributes.

as well as DT, NN takes a step to less pruning and more over-fitting. BN was already optimal with default parameters. Finally, for AIS, we obtained a very good set of parameters from GA execution, which made the multiresolution optimization phase quite easy in order to obtain a good optimized and robust set of parameters. One of the most surprising results was K equals to 1, which means that no voting is necessary: the first rule that matches decides the class.

Final comparison of all methods: Since the standard deviation seen in Figure 2 suggests us that DT, BN and NB could have the same costs, we performed four statistics t-student tests with 100 new random splits in the same proportion. These splits were specially created for these tests. We tested if $ROBUST(AIS) - ROBUST(DT) = 0$, $ROBUST(DT) - ROBUST(BN) = 0$, $ROBUST(BN) - ROBUST(NB) = 0$ and $ROBUST(NB) - ROBUST(NN) = 0$. Not surprisingly, with 99.9% of certainty, all $H0$ were rejected, which means that none of them is equal. In the end, the average of costs for strategy robust is what defines the rank of methods. From the Figure 2, we can notice that AIS produced the best classifiers, followed by DT, BN, NB, and NN, in this order.

6 Future Work

We intend to analyze in details the optimized parameters in the coming future, and try to reach better relations between the value of each parameter and its relation to the skewness of the data, at same time that we enquire why AIRS2 implementation of AIS outperforms the implementations of other methods. We are also extending the analysis in such a way to evaluate the influence of a metaheuristics like Cost Sensitive Classifier [5], which takes into account the different costs of false positive and false negative in the training phase. Using this metaheuristics, in our preliminary and unfinished results, we are observing that one may obtain better classifiers for all methods, up to Naive Bayes. We also consider the inclusion of Support Vector Machines (SVM) in the pool of compared methods. And given we are using AIS, a suitable comparison method would be k nearest neighbour.

We intend to apply the models for unseen out-of-date datasets to compare stability and life expectancies. Since, as we know, the fraudulent behavior is very dynamic, often a model loses its prediction power in a short time. Besides knowing which method generates the most accurate model, it is important to know which one generates the model that remains predictive for a longer time.

7 Conclusions

In this paper, we present a comparative study of five classification methods (Decision Tree, Neural Network, Bayesian Network, Naive Bayes and Artificial Immune System). The used definition of an objective function to be optimized that takes into account different costs for false positives and false negatives is important. In all our executions, except for NB (no parameter needed) and BN, we concluded that the best results had not been reached with default set of parameters as given in Weka. Particularly for AIS and NN, the results gotten using default parameters are very poor if compared with those gotten after a parametric adjustment using GA. Our tests results show that BN is better than NN, the most used method in real application today, which reproduces the results from Maes [11, 12]. In addition, we obtained that AIS and DT also surpass NN. Perhaps because DT is a classic classification method, it has been forgotten in recent works. However, it still reveals itself as one of the best methods, with sufficient competitive results. On our tests AIS had a surprisingly large increase of performance from default parameters to GA optimized parameters, and this performance was kept in the obtaining of an optimized robust parameter set.

To sum up, AIS produced the best classifiers, followed by DT, BN, NB, and NN, respectively.

8 Acknowledgments

This work was partially supported by Projects PRONEX-FAPESP # 03/09925-5 and CNPq # 306217/2004-0. We would like to acknowledge the banks Santander and Citybank for supporting the development of this reasearch by their employees. We would also like to thank Roberto Cesar Marcondes, Nina Hirata, Casimir Kulikowski, João Eduardo Ferreira, Yoshiharu Kohayakawa for their comments and support.

References

1. Jason Brownlee. Artificial immune recognition system (airs) - a review and analysis. Technical report, Victoria, Australia: Centre for Intelligent Systems and Complex Processes (CISCP), Faculty of Information and Communication Technologies (ICT), Swinburne University of Technology, Jan 2005.

2. Eugene Charniak. Bayesians networks without tears. *AI Magazine*, 1991.
3. Dipankar DasGupta. *Artficial Immune Systems and Their Applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1998.
4. Leandro Nunes de Castro and J. Timmis. *Artificial Immune Systems: A Novel Paradigm to Pattern Recognition*. University of Paisley, 2002.
5. Charles Elkan. The foundations of cost-sensitive learning. In *IJCAI*, pages 973–978, 2001.
6. Wei Fan, Salvatore J. Stolfo, Junxin Zhang, and Philip K. Chan. AdaCost: misclassification cost-sensitive boosting. In *Proc. 16th International Conf. on Machine Learning*, pages 97–105. Morgan Kaufmann, San Francisco, CA, 1999.
7. John Holland. *Adaptation in Natural and Artificial Systems*. 1st MIT press, 1975.
8. J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Science*, volume 79, pages 2554–2558, 1982.
9. Jinwoo Kim and Bernard P. Zeigler. A framework for multiresolution optimization in a parallel/distributed environment: simulation of hierarchical gas. *J. Parallel Distrib. Comput.*, 32(1):90–102, 1996.
10. Erica Klarreich. Inspired by immunity. *Nature*, (415):468–470, January 2002. doi:10.1038/415468a.
11. S. Maes, K. Tuyls, and B. Vanschoenwinkel. Machine learning techniques for fraud detection, 2000.
12. Sam Maes, Karl Tuyls, Bram Vanschoenwinkel, and Bernard Manderick. Credit card fraud detection using bayesian and neural networks. In *Proceedings of NF2002, 16-19 January*, Havana, Cuba, 2002.
13. Sreeram K. Murthy, Enno Ohlebusch, and Stefan Kurtz. Automatic construction of decision trees from data: A multi-disciplinary survey. In *Data Mining and Knowledge Discovery*, volume 2, pages 345 – 389, USA, 1998. Kluwer Academic Publishers.
14. Clifton Phua, Vincent Lee, Kate Smith, and Ross Gayler. A comprehensive survey of data mining-based fraud detection research. In *Artificial Intelligence Review (submitted for publication)*, 2005.
15. Robert E. Schapire and Yoram Singer. Improved boosting using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.
16. S. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. Chan. Credit card fraud detection using meta-learning: Issues and initial results, 1997. Working notes of AAAI Workshop on AI Approaches to Fraud Detection and Risk Management.
17. Andrew Watkins, Jon Timmis, and Lois Boggess. Artificial immune recognition system (AIRS): An immune-inspired supervised machine learning algorithm. *Genetic Programming and Evolvable Machines*, 5(3):291–317, September 2004.
18. Ian H. Witten and Eibe Franku. *Data Mining: Practical Machine Learning Tools and Techniques (Second Edition)*. Elsevier, 2005.
19. Ian H. Witten and Eibe Franku. Software documentation: Weka, 2008. file `/Weka-3-4-11/doc/weka/classifiers/functions/MultilayerPerceptron.html` from Weka package.

Appendix

For next paragraph, let us define VR = [X1;X2; step = S] as been the allowed variation range from X1 to X2 and S, the precision step for this specific parameter S.

Naive Bayes: **NB** does not have any parameter.

Decision Tree: **DT** has two parameters C and M:

- **C:** the confidence threshold for pruning. (Default: 0.25). VR = [0.01;1.00; step = 0.01].
- **M:** the minimum number of instances per leaf. (Default: 2). VR = [1;100; step = 1].

Bayesian Network: **BN** has three parameters (D, Q, E):

- **D:** defines whether a structure called ADTree will or not be used;
- **Q:** defines which search for topology algorithm will be used. The available ones are: GeneticSearch, HillClimber, K2, LocalScoreSearchAlgorithm, RepeatedHillClimber, SimulatedAnnealing, TabuSearch e TAN. Every search algorithm has two parameters:
  - **P:** defines the number of parent's allowed in the topology.
  - **S:** defines the type of score to be used to build the conditional table, they are: BAYES, BDeu, MDL, ENTROPY e AIC;
- **E:** defines the estimator algorithm to calculate the conditional tables. In Weka they are: BayesNetEstimator, BMAEstimator, MultiNomialBMAEstimator and SimpleEstimator (this estimator has one parameter (A), called alpha, and it ranges between 0% e 100%, and it represents a start value for the conditional probability.).

Neural Network: **NN** has seven parameters (L, M, N, V, S, E, H):

- **L:** the learning rate. (default 0.3). The closer to zero, the smaller the impact of the incoming information to be learnt. VR = [0.01;1.00; step = 0.01].
- **M:** the momentum (default 0.2). Its inclusion (values greater than zero) has for objective to increase the speed of the training of a neural net and to reduce the instability. VR = [0.00;1.00; step = 0.01].
- **N:** the number of epochs to train through. (default 500). our tests indicates that using N greater than 500 does not increase the performance significantly, and fixing it to its default 500. VR = [500;500; step = 0].
- **V:** the percentage size of the validation set from the training to use. (default 0 (no validation set is used, instead number of epochs is used). It ranges between 0% and 99,99%, when this parameter is greater that zero intend to reduce over-fitting. VR = [0.00;0.99; step = 0.01].
- **S:** the seed for the random number generator. We used default value. VR = [0;0; step = 0].
- **E:** the threshold for the number of consecutive errors allowed during validation testing. (default 20). Number between 1 and 100. This parameter participates with N to form the stop condition of the algorithm. VR = [1;100; step = 1].
- **H:** string of numbers of nodes to be used on each layer. Each number represents its own layer and the number of nodes on that layer. There are also some wildcards: 'a', 'i', 'o', 't'. These are 'a' = (number of attributes + number of classes) / 2, 'i' = number of attributes, 'o' = number of classes, and 't' = number of attributes + number of classes. VR = [1;20; step = 1].

Artificial Immune System: **AIS** has 9 parameters (S, F, C, H, R, V, A, E, K):

- **S:** the seed for the random number generator. (default 0). We adopted the fixed value 1. VR = [1;1; step = 0].
- **F:** the minimum number percentage affinity threshold (see [17] page 6). VR = [0.00;0.5; step = 0.01].
- **C:** the Clonal Rate is an integer that ranges between 0 ant 100. VR = [1;100; step = 1].
- **H:** the Hyper-mutation rate. Ranges between 0 and 100 and determines the percentage of clones (from last parameter) that will suffer mutation. VR = [0;10; step = 1].
- **R:** the total resources is the maximum number of B-Cell (or ARB) allowed in the system. VR = [0;200; step = 1].
- **V:** the Stimulation threshold is a number between 0 and 1 used as criteria to keep or drop a given B-Cell. VR = [0.00;1.00; step = 0.01].
- **A:** the number of affinity threshold instances. Because of lack of documentation in [1] we used the default (-1) value. VR = [-1;-1; step = 0].
- **E:** the memory pool size. Define the number of random initialization instances. By simplicity we varied it between 0 and 10. VR = [0;10; step = 1].
- **K:** the number of nearest neighbors representing B-Cells to be matched and consulted in a voting election of which class the current transaction belongs to. K equals to 1 means no voting. VR = [0;10; step = 1].