Determining the Structure of Decision Directed Acyclic Graphs for Multiclass Classification Problems

Thaise M. Quiterio
Instituto de Ciência e Tecnologia (ICT)
Universidade Federal de São Paulo (UNIFESP)
São José dos Campos-SP, Brazil
Email: thaise.quiterio@unifesp.br

Ana C. Lorena
Machine Learning Research Group
ICT - UNIFESP
São José dos Campos-SP, Brazil
Email: aclorena@unifesp.br

***Abstract*—An usual strategy to solve multiclass classification problems in Machine Learning is to decompose them into multiple binary sub-problems. The final multiclass prediction is obtained by a proper combination of the outputs of the binary classifiers induced in their solution. Decision directed acyclic graphs (DDAG) can be used to organize and to aggregate the outputs of the pairwise classifiers from the one-versus-one (OVO) decomposition. Nonetheless, there are various possible DDAG structures for problems with many classes. In this paper evolutionary algorithms are employed to heuristically find the positions of the OVO binary classifiers in a DDAG. The objective is to place easier sub-problems at higher levels of the DDAG hierarchical structure, in order to minimize the occurrence of cumulative errors. For estimating the complexity of the binary sub-problems, we employ two indexes which measure the separability of the classes. The proposed approach presented sound results in a set of experiments on benchmark datasets, although random DDAGs also performed quite well.**

1. Introduction

Many practical problems involve distinguishing data into one out of multiple classes. These multiclass problems can be simplified by their division into simpler and independent sets of binary sub-problems [1]. The outputs of the classifiers induced in their solution are them combined to obtain the final multiclass predictions. These sub-problems are usually less complex and easier to solve than their original multiclass counterpart [2]. There are also Machine Learning (ML) techniques originally conceived for binary classification, like Support Vector Machines (SVM) [3], that benefit from decomposition.

One popular decomposition strategy is one-versus-one (OVO) [4]. Herewith, each class from the original problem is confronted to another class, in pairwise combinations, producing one binary sub-problem for each pair of classes. A decision directed acyclic graph (DDAG) [5] can be used to combine the outputs from the OVO classifiers. The DDAG is a hierarchical structure in which each internal node corresponds to a binary predictor for a pair of classes, while the leaf nodes correspond to the individual classes. For predicting the class of a new example, the DDAG structure is traversed from the root node into a leaf, which gives the final prediction. In order to provide more robustness to the DDAG overall results, simpler sub-problems should be evaluated first [6], [7]. By doing such, cumulative errors can be minimized.

For evaluating the complexity of the OVO pairwise sub-problems, we employ two indexes from [8] that were proposed to estimate the difficulty/simplicity of a binary classification problem. They can be computed from training data at low computational costs [9]. The first one is the Fraction of Borderline Points (N1), which tries to estimate the size of the classification border required for separating the classes. The second measure is the Ratio of Intra/Extra Class Nearest Neighbor Distances (N2), which reports the ratio of the distances between close elements from the same class and from opposite classes. The values of these measures are used to define the ordering of the OVO classifiers in DDAGs. Thereby, simpler sub-problems according to N1 and N2 will be placed at upper levels of the DDAG hierarchy.

As noted in [10], determining the structure of a DDAG is analogous to solve a traveling salesman problem (TSP) instance. The objective comes down to finding a proper ordering of the k classes of the problem. In our case the ordering should take the values of the complexity measures into account. The number of possible DDAG structures rapidly increases as the number of classes increases, making an exhaustive search impractical. As in [10], in this paper we employ a genetic algorithm (GA) [11] to find proper DDAG structures. Here they are evolved according to the values of the complexity measures N1 or N2, while in [10] the accuracy of the DDAG structures using SVMs was estimated in validation datasets. The new approach has some advantages: there is no need to separate part of the training data for validation; the structure found is independent of a specific classification technique; and the cost for estimating the values of the complexity measures is lower than that needed for solving the multiclass classification problem multiple times. Moreover, two other optimization techniques are also explored in this paper: an estimation of distribution algorithm (EDA) and a hybrid version of GA and EDA.

For evaluating our proposals, experiments are performed using benchmark datasets containing from 10 to 26 classes. Firstly we verified that all optimization techniques were indeed able to organize the binary classifiers such that simpler binary sub-problems were placed at the top of the hierarchies. GA solutions were them chosen to be evaluated next. Herewith, classifiers were induced using the DDAG structures and their accuracy in the multiclass problem solution was verified. Multilayer Perceptron Neural Networks (MLP) [12] were used in the classifiers induction. The DDAG structures determined by the GA presented predictive performance close to those of random DDAG structures. This is an indicative that there is not a large variation in the results of distinct DDAGs, as observed in [6], [13] for SVMs too.

This paper is structured as follows. Section 2 describes the DDAGs. Section 3 presents the data complexity measures. Section 4 presents how EAs can be applied to determine the structure of DDAGs. Section 5 presents the methodology adopted in the evaluation of the DDAG structures. Section 6 presents and discusses the results achieved. Section 7 concludes this paper.

2. Decomposition of Multiclass Problems

There are many strategies for decomposing a multiclass problem into multiple binary sub-problems, for example, one-versus-all (OVA) [14] and the error correcting output Codes (ECOC) [15]. Recent studies have pointed several advantages to use one-versus-one (OVO) [4] decomposition [16]. This strategy produces $\frac{k(k-1)}{2}$ binary sub-problems, one for each pair of classes (i, j), where $i < j$. Binary classifiers are trained for solving each of the binary sub-problems generated. New examples should be submitted to part of or all classifiers whose outputs are joined to obtain the final multiclass prediction. The most common approach is to output a majority voting of the predictions.

When majority voting is used with OVO all classifiers must always be consulted and ties can happen if more than one class receives the same number of maximum votes. These problems can be relieved by using a decision directed acyclic graph (DDAG) [17]. This is a hierarchical structure where each internal node corresponds to a binary pairwise predictor, as shown by the three DDAGs in Figure 1. Starting from the root node, based on the prediction of a classifier, a class is excluded from further examination. Afterwards, either a new classifier is consulted or a final prediction is obtained. Therefore, when classifying a new example, only $k-1$ binary predictors are evaluated. This is a clear advantage of DDAG for combining OVO classifiers. Nonetheless, the DDAG predictive results are dependent on how the binary classifiers are positioned within the hierarchy. Consider, for instance, the three-class classification problem shown in Figure 1 [18]. The dashed area contains points for which different predictions are obtained depending on the DDAG structure, as shown by the three possible DDAG structures for this problem.

The number of possible DDAGs for a problem with k classes is $\frac{k!}{2}$. Some work have then investigated heuristics to determine the structure of DDAGs suited for each problem [6], [13], [19], [20]. In general, the idea is that easier binary sub-problems should be placed closer to the DDAG root, minimizing error propagation throughout the hierarchy.

Works [6] and [7] suggest that nodes from upper levels of the DDAG should contain classifiers with higher generalization ability. SVMs are employed as base classifiers and the generalization ability of the binary predictors is evaluated by leave-one-out error bounds. They obtained results close to those of the average of several random DDAG structures. In [19] the DDAG structure is modified into that of a tree which uses more classification models and various criteria are employed to estimate the difficulty in separating the classes. All of them are specific to SVM classifiers. In [13] a Genetic Algorithm (GA) is employed to find the permutation of classes in the DDAG, based on a validation error rate achieved in multiclass classification. More recently, [20] used cross-validation error estimates to determine how to place the pairwise classifiers in a DDAG. They also propose other mechanisms to join the predictions of the pairwise classifiers.

In this paper we adopt a general approach and less dependent on a specific classifier. This was accomplished by using two complexity measures for classification problems to estimate the difficulty of each binary sub-problem. The ordering of the pairs of classes pairs is performed by an evolutionary algorithm according to the calculated values, so that simpler sub-problems can be evaluated first.

3. Data Complexity Measures

There are several studies devoted to estimate the complexity of a classification problem by extracting indexes from the available training datasets [21]. In this paper we employ two indexes from [8], which is a seminal work from the area. The chosen measures are among the most successful in characterizing the complexity of binary classification problems in [8], [21]. They will allow estimating the complexity of the pairwise sub-problems in OVO according to different perspectives.

The first measure is the fraction of borderline points, denoted as N1. Within it, first a minimum spanning tree (MST) is built using the training data T [8]. The MST will connect closer points according to their Euclidean distance. Afterwards, N1 counts the number of points of opposite classes that are connected in the MST, and divides this value by n, the total number of examples in T. This gives an indicative of the size of the border necessary for separating the classes in T. N1 lies in the $[0, 1]$ interval and higher values are obtained when many examples from opposite classes are close to each other. Therefore, larger N1 values are associated with more complex problems.

The second measure is N2, the ratio of the intra/extra nearest neighbor distance. For each example $\mathbf{x}_i$, the ratio between the distance from $\mathbf{x}_i$ to the closest element from its class and the distance from $\mathbf{x}_i$ to the closest element from

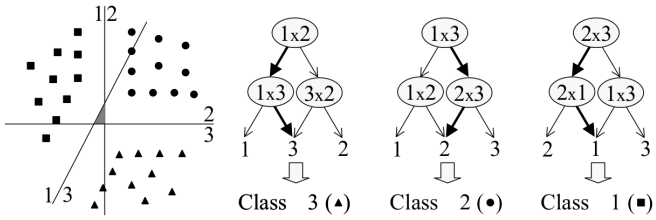

Figure 1. Classification in shaded area differs according to the DDAG structure adopted

the other class is taken. These ratios are summed up for all training examples in T. N2 minimum value is 0 and the upper bound is dependent on the problem. Higher values are obtained when, for several elements, the distance between examples from the same class exceeds the distance between examples from different classes. In this case, the problem can be considered more complex.

4. Determining the DDAG Structure

A DDAG can be represented by an array structure (in fact, a same DDAG can be represented by two arrays, since by symmetry $(i, j) = (j, i)$). These arrangements correspond to the ordering of the classes in the bottom level of the DDAG. For the DDAGs in Figure 1, the lists are (1,3,2), (1,2,3) and (2,1,3), from left to right.

Evolutionary Algorithms (EAs) [22] have been extensively explored for solving such kind of permutation problem. EAs are optimization algorithms inspired by principles of natural evolution. Ordering the classes can be considered a particular instance of the traveling salesman problem (TSP), where the objective is to visit k cities once at lower traveling cost [23]. The first EA used in the generation of DDAGs was a Genetic algorithm (GA) [13]. GAs are heuristic methods based on natural selection and genetics. They operate a population of solutions, which are continuously evolved through the use of genetic operators. The application of these operators for several generations tends to make the population converge to a good solution, although not necessarily the optimal [11].

In [13] GAs were used for defining the ordering of the k classes in a DDAG that optimized its predictive performance in the problem solution. Each individual represents a possible DDAG as a vector containing a specific sequence for the k classes, without repetition. The individuals fitness was estimated by the DDAGs performance in validation datasets, using SVMs as base classifiers. Classical genetic operators from the TSP literature were employed [24]: stochastic tournament selection, partially mapped crossover (PMX), cycle crossover (CX), order-based crossover (OX) and insertion mutation. The crossover operators were chosen probabilistically at each round of the GA.

Here a GA is also used, with tournament selection, PMX crossover and inversion mutation, both from the TSP literature [24]. But the fitness function takes into account the values of the complexity measures N1 or N2. Since the objective is to place easier problems at upper levels of the DDAG, the fitness computation will take a weighted sum of the N1 or N2 values for the pairwise combinations of classes at each level of the DDAG. The weights are proportional to the level occupied by each pair of classes in the DDAG. The first level receives the weight three, which is increased at steps of two for each new level. They are further normalized such that all weights are summed to one. Herewith, elements at bottom levels will contribute most to the fitness value, forcing the heaviest pairs according to N1 (N2) to be placed as leaves. Taking, for instance, a problem with three classes where the N1 values are: 0.3 for the pair 1 vs 2; 0.5 for the pair 1 vs 3; and 0.8 for the pair 2 vs 3. The fitness values of the DDAGs shown in Figure 1, from left to right, would be: 0.57 $((3 * 0.3 + 5 * 0.5 + 5 * 0.8)/(3 + 5 + 5))$, 0.54 $((3 * 0.5 + 5 * 0.3 + 5 * 0.8)/(3 + 5 + 5))$ and 0.49 $((3*0.8+5*0.3+5*0.5)/(3+5+5))$, respectively. Therefore, the first tree would be considered the fittest, which is indeed the best arrangement in this case, where the easier pair 1 vs 2 is placed at the root of the DDAG.

Another evolutionary algorithm used in this work is the estimation of distribution algorithm (EDA) [25]. This method is characterized by using probability models from the distribution of variables in a population. While GAs combine two solutions through crossover, EDAs build a probabilistic model based on existing solutions to generate new individuals. At each round the EDA selects a proportion of the best individuals and identify how many times each class is positioned in their first position, in their second position, and so on. This generates a probability order and a new solution is created based on the highest probability values. This new solution replaces the individual of lower fitness in the population. This process is iterated for a number of cycles. The fitness function and individuals' representation are the same as described for GAs.

We also evaluated a hybrid combination of the GA and EDA algorithms: a random number is generated at each iteration. If this number is greater than a certain threshold, the GA is used. Otherwise, the EDA is employed.

5. Methodology

Two evaluations are performed in this paper. Firstly, the different types of EAs are used to determine the structure of DDAGs according to the complexity measure values. The EA achieving best performance measured by our fitness measure previously described is evaluated next regarding the multiclass predictive performance of the generated DDAGs.

5.1. Datasets

Ten data sets were selected from the KEEL [26] and UCI Machine Learning public data repositories [27]. All are divided according to the 10-fold stratified cross-validation strategy and normalized. Some characteristics of these datasets, namely their number of predictive features, classes and examples and majority error rate (ME) are shown in Table 1.

TABLE 1. DATASETS USED IN THE EXPERIMENTS.

Name	#Attributes	#Examples	Classes	ME
yeast	8	1484	10	0.69
letter	16	20000	26	0.96
movement_libras	90	360	15	0.93
led7digit	7	500	10	0.87
optdigits	64	5620	10	0.90
penbased	16	10992	10	0.90
texture	40	5500	11	0.91
vowel	13	990	11	0.91
kr-vs-k	6	28056	18	0.84
isolet	617	6238	26	0.96

All datasets are decomposed according to the OVO strategy with the KEEL tool[1] (*Knowledge Extraction based on Evolutionary Learning*) [26], and the N1 and N2 values are computed on the obtained subsets using the DCOL-v1.1 library[2] (*Data Complexity Library in C++*) [9].

5.2. Fitness Evaluation

All EAs tested have the goal to maximize a fitness function which takes into account the values of the complexity measures for each pair of classes. We calculated theoretical best and worst fitness values that can be achieved for each dataset. The best case occurs when the pair of classes with smaller measure value is placed at the root of the DDAG, followed by the pair with the second lightest value as a child of this node, and the third as the next child and so on. These values are positioned as in a width-breath search, disregarding the classes that were already positioned in the DDAG. This is an optimistic and probably unrealistic estimate, since some pairing of classes in the DDAG are determined by their parent nodes. Therefore, there is a dependency which is disregarded in the computation of the best fitness values reported. However, these values provide a baseline for the best achievable performance. The worst case is given by the reverse reasoning, starting with the pair of highest complexity in the root of the DDAG. Again the dependency of the classes imposed by the hierarchical structure is disregarded. The EAs have to obtain solutions with fitness values at least better than this worst case value.

For all EAs, the initial population is generated randomly. As the results are stochastic, the algorithms were run 20 times for each dataset partition. The size of the tournament in the GA was pre-set at three, the mutation rate used was 0.1, the cross-over rate ranged as 0.6, 0.7 and 0.8. The population size was varied, where the lowest value tested was 400 and the highest was 2000, considering all tests. The number of generations or cycles was set from 10 to 25 for all strategies, considering variations of 5 between tests. In EDA, the rate of best individuals for pooling a new individual was 0.2. In the hybrid algorithm, the rate that determines the switch between the use of the GA or EDA ranged between 0.5 and 0.8, considering variations of 0.1 between tests. Those parameter combinations giving the best performance per dataset, using the N1-based fitness measure, were chosen. The same parameters were adopted for N2-based fitness EAs.

The average fitness values obtained by all EAs were compared. Random DDAG structures were also generated for a baseline comparison (20 per dataset). This analysis allowed to choose one of the EAs to have their solutions further evaluated, as described next.

5.3. Predictive Performance Evaluation

The solutions obtained by the best EA were used to generate multiclass classifiers for the datasets and their predictive performance was computed. Multilayer Perceptron (MLP) neural networks trained with the iRprop [28] algorithm were used in the classifiers induction. Although they can be employed directly in the multiclass problem solution, a previous decomposition of the problem can possibly simplify its solution. The FANN library[3] (Fast Artificial Neural Library) for C/C ++ language was used in the implementation of the MLP neural networks.

MLPs require a proper calibration of some parameters which directly influence their predictive results. A single hidden layer was used and the number of neurons varied from 5 to 25, with variations of 5 neurons for each configuration. The learning rate ranged as 0.1, 0.2 and 0.3. The threshold error error rate for stopping the MLP training ranged from 0.01 to 0.00001 and the maximum number of training iterations varied from 50 to 500. The configurations achieving best predictive results per dataset were chosen.

We also generated MLP classifiers using the random DDAG structures. The objective is to compare the predictive results achieved by our optimized structures against those obtained by random structures. For all classifiers generated, the average error rate in cross-validation is computed. The results of the structures obtained by the GA using N1 and N2 measures in the fitness function are compared to those from the random structures.

6. Experimental Results

This section presents the results achieved in the experiments performed in this paper. First we evaluate if the optimization methods are able to find DDAGs combinations that maximize the fitness measure based on the complexity of the binary sub-problems. Tables 2 and 3 presents the

1. http://www.keel.es
2. http://dcol.sourceforge.net/
3. http://leenissen.dk/fann/wp/

results achieved by all EAs and the random structures using N1 and N2 in the fitness computations, respectively. The best results among the EAs and the random structures in each dataset are highlighted in boldface, while the worst results are highlighted in italics. The tables also present the theoretical best and worst fitness values for each measure.

All search strategies, including the random one, performed better than the worst case theoretical baseline. Indeed, the results are always within the minimum (best) and maximum (worst) achievable values. For some datasets, as *opt*, *pen* and *tex*, most of the strategies are able to achieve the "Best" performance. All EAs showed better results than those reported for the random search and showed to be able to optimize the fitness function designed. EAs were also more stable than the random search, as the standard deviation of their results was always lower (it was null in most of the cases). Overall, the results of all EAs were very close for both N1 and N2 measures. Since the GA is a well-known algorithm and presented consistently the best results among the EAs, its solutions were chosen to be further evaluated.

The average and standard deviation of the error rates achieved by the DDAGs generated randomly and by the GA using MLPs as base classifiers are presented in Table 4. GA-N1 refers to the GA using the N1 measure and GA-N2 represents the GA using the N2 complexity measure. The same reasoning applies to the Random columns. Best predictive results per dataset are highlighted in boldface and worst results are highlighted in italics.

It is possible to observe that all DDAGs presented close predictive results, despite of the: complexity measure employed (N1 or N2) or search strategy (GA or random). The standard deviation of the results were always low. It is interesting to notice that a random sampling of the DDAG structures already present very good results concerning their predictive performance in the multiclass problems solution. This was also observed for SVMs classifiers in [13]. As SVMs, MLPs usually show a high generalization ability, which may partially justify this behavior. However, it should be noticed that for some specific datasets (*mov*, *tex*, *krv* and *iso*) the predictive results are low for all strategies. These datasets are characterized by a high class imbalance, which may have impaired the classifiers performance. But this can also be result of a poor calibration of the MLP parameter values.

7. Conclusion

In this work, a search approach was employed to determine the best DDAG structure for solving a given multiclass classification problem. Two complexity measures are used for guiding the placement of the pairs of classes in the DDAG hierarchy. The objective is to place easier binary sub-problems at upper levels of the DDAGs, while the most difficult pairings should be left to bottom levels. Three EAs using a fitness measure that considers the values of these measures were implemented: a GA, an EDA and an hybrid combination of the previous meta-heuristics. The GA showed satisfactory results in this search and the DDAG structures found by this EA were then evaluated regarding their predictive ability in the multiclass problems solution. MLPs were employed as base classifiers. In this case the random sampling performed well and the DDAGs found had predictive performance equal to that of the GA solutions. This can be attributed to the fact that the DDAG results do not vary too much for distinct structures. Nonetheless, more tests are needed to corroborate these results, mainly with other classification techniques and for problems with more classes.one relevant investigation would be to verify if multiple random DDAGs perform equally well for problems with more classes and when other classification techniques are employed as base classifiers.

As future work we shall investigate the use of other data complexity measures for optimizing the DDAG structure. Other ML techniques, specially SVMs, which are originally conceived for binary classification problems, can also be employed in the classifiers generation. And we consider performing experiments for datasets with more classes, where we expect that the predictive results for distinct DDAG strutures will vary more. It would be equally interesting to adapt the DDAG structure according to each example being classified, which can possibly improve the predictive results achieved.

Acknowledgments

To the research agencies FAPESP (2015/17291-3 and 2012/22608-8) and CNPq for the financial support.

References

[1] A. C. Lorena, A. C. P. L. F. Carvalho, and J. M. P. Gama, "A review on the combination of binary classifiers in multiclass problems," *Artificial Intelligence Review*, vol. 30, pp. 19–37, 2008.

[2] M. Galar, A. Fernández, E. Barrenechea, H. Bustince, and F. Herrera, "An overview of ensemble methods for binary classifiers in multi-class problems: Experimental study on one-vs-one and one-vs-all schemes," *Pattern Recog.*, vol. 44, no. 8, pp. 1761–1776, 2011.

[3] N. Cristianini and E. Ricci, "Support vector machines," in *Encyclopedia of Algorithms*. Springer, 2008, pp. 928–932.

[4] S. Knerr, L. Personnaz, and G. Dreyfus, "Handwritten digit recognition by neural networks with single-layer training," *Neural Networks, IEEE Transactions on*, vol. 3, no. 6, pp. 962–968, 1992.

[5] J. C. Platt, N. Cristianini, and J. Shawe-Taylor, "Large margin dags for multiclass classification." in *nips*, vol. 12, 1999, pp. 547–553.

[6] F. Takahashi and S. Abe, "Optimizing directed acyclic graph support vector machines," in *Proc. Art. Neural Netw. in Pattern Recog.*, 2003, pp. 166–170.

[7] J. Feng, Y. Yang, and J. Fan, "Fuzzy multi-class SVM classifier based on optimal directed acyclic graph using in similar handwritten chinese characters recognition," in *Proc. Int. Symp. on Neural Netw.*, ser. LNCS, vol. 3496. Springer-Verlag, 2005, pp. 875–880.

[8] T. K. Ho and M. Basu, "Complexity measures of supervised classification problems," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 3, pp. 289–300, 2002.

[9] A. Orriols-Puig, N. Macia, and T. K. Ho, "Documentation for the data complexity library in c++," *Universitat Ramon Llull, La Salle*, vol. 196, 2010.

TABLE 2. FITNESS RESULTS USING THE N1 MEASURE.

	Hybrid		EDA		GA		Random		Best		Worst	
Name	**avg**	**std**	**avg**	**std**	**avg**	**std**	**avg**	**std**	**avg**	**std**	**avg**	**std**
yea	**1.06**	0.00	1.03	0.00	**1.06**	0.00	*0.91*	0.04	1.13	0.03	0.65	0.01
let	**0.21**	0.00	0.20	0.00	**0.21**	0.00	*0.19*	0.00	0.24	0.00	0.13	0.00
mov	**1.31**	0.00	1.27	0.00	**1.31**	0.00	*1.19*	0.03	1.39	0.03	0.95	0.02
led	**1.35**	0.00	1.32	0.00	**1.35**	0.00	*1.20*	0.05	1.44	0.04	0.91	0.03
opt	**0.04**	0.00	**0.04**	0.00	**0.04**	0.00	**0.04**	0.00	0.04	0.00	0.02	0.00
pen	**0.03**	0.00	**0.03**	0.00	**0.03**	0.00	**0.03**	0.00	0.03	0.00	0.02	0.00
tex	**0.04**	0.00	**0.04**	0.00	**0.04**	0.00	*0.03*	0.00	0.04	0.00	0.02	0.00
vow	**0.69**	0.00	0.67	0.00	**0.69**	0.00	*0.63*	0.02	0.72	0.01	0.52	0.00
krv	**1.06**	0.00	1.00	0.00	**1.06**	0.00	*0.91*	0.03	1.19	0.00	0.56	0.00
iso	0.45	0.00	0.42	0.00	**0.46**	0.00	*0.37*	0.17	0.49	0.00	0.18	0.00

TABLE 3. FITNESS RESULTS USING THE N2 MEASURE.

	Hybrid		EDA		GA		Random		Best		Worst	
Name	**avg**	**std**	**avg**	**std**	**avg**	**std**	**avg**	**std**	**avg**	**std**	**avg**	**std**
yea	**2.82**	0.00	2.78	0.00	**2.82**	0.00	*2.60*	0.06	3.00	0.04	2.21	0.03
let	**4.81**	0.00	4.74	0.00	**4.81**	0.00	*4.67*	0.02	5.00	0.01	4.31	0.01
mov	**3.30**	0.00	3.25	0.00	**3.30**	0.00	*3.16*	0.03	3.39	0.06	2.89	0.05
led	**1.39**	0.00	1.36	0.00	**1.39**	0.00	*1.26*	0.04	1.45	0.10	1.02	0.05
opt	**2.79**	0.00	2.78	0.00	**2.79**	0.00	*2.71*	0.02	2.84	0.00	2.58	0.00
pen	**1.25**	0.00	1.24	0.00	**1.25**	0.00	*1.19*	0.02	1.29	0.00	1.09	0.00
tex	**1.61**	0.00	1.57	0.00	**1.61**	0.00	*1.47*	0.03	1.70	0.01	1.22	0.01
vow	**1.42**	0.00	1.40	0.00	**1.42**	0.00	*1.34*	0.02	1.47	0.02	1.20	0.01
krv	**3.99**	0.00	3.88	0.01	**3.99**	0.00	*3.71*	0.06	4.25	0.01	3.09	0.01
iso	10.17	0.01	10.07	0.00	**10.18**	0.01	*9.96*	0.03	10.52	0.01	9.37	0.01

TABLE 4. ERROR RATES OF DDAGS GENERATED.

	GA-N1		Random-N1		GA-N2		Random-N2	
Name	**avg**	**std**	**avg**	**std**	**avg**	**std**	**avg**	**std**
yea	**0.41**	0.00	**0.41**	0.01	**0.41**	0.00	**0.41**	0.01
let	**0.20**	0.00	**0.20**	0.01	**0.20**	0.00	**0.20**	0.01
mov	**0.97**	0.01	*0.98*	0.02	*0.98*	0.01	**0.97**	0.02
led	*0.30*	0.00	*0.30*	0.01	**0.29**	0.00	*0.30*	0.01
opt	**0.03**	0.00	**0.03**	0.00	**0.03**	0.00	**0.03**	0.00
pen	*0.02*	0.00	*0.02*	0.00	**0.01**	0.00	*0.02*	0.00
tex	**0.91**	0.00	**0.91**	0.01	**0.91**	0.00	**0.91**	0.01
vow	**0.08**	0.00	**0.08**	0.01	**0.08**	0.00	**0.08**	0.01
krv	**0.83**	0.00	*0.84*	0.01	**0.83**	0.00	*0.84*	0.01
iso	**0.98**	0.01	**0.98**	0.01	**0.98**	0.00	**0.98**	0.01

[10] A. C. Lorena and A. C. P. L. F. Carvalho, "An hybrid ga/svm approach for multiclass classification with directed acyclic graphs," in *Advances in Artificial Intelligence–SBIA 2004*. Springer, 2004, pp. 366–375.

[11] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Machine learning*, vol. 3, no. 2, pp. 95–99, 1989.

[12] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1994.

[13] A. C. Lorena and A. C. P. L. F. Carvalho, "Design of directed acyclic graph multiclass structures," *Neural Network World*, vol. 17, no. 6, p. 657, 2007.

[14] P. Clark and R. Boswell, *Machine Learning — EWSL-91: European Working Session on Learning Porto, Portugal, March 6–8, 1991 Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, ch. Rule induction with CN2: Some recent improvements, pp. 151–163.

[15] T. G. Dietterich and G. Bariki, "Solving multiclass learning problems via error-correcting output codes," *Journal of Artificial Intelligence Research*, vol. 2, pp. 263–286, 1995.

[16] Z. Zhang, B. Krawczyk, S. Garca, A. Rosales-Prez, and F. Herrera, "Empowering one-vs-one decomposition with ensemble learning for multi-class imbalanced data," *Knowledge-Based Systems*, 2016.

[17] J. C. Platt, N. Cristiani, and J. Shawe-Taylor, "Large margin DAGs for multiclass classification," in *Advances in Neural Information Processing Systems*, 2000, vol. 12, no. 5, pp. 547–553.

[18] B. Kijsirikul and N. Ussivakul, "Multiclass support vector machines using adaptive directed acyclic graph," in *Proc. Int. Joint Conf. on Neural Netw.*, 2002, pp. 980–985.

[19] E. Montañés, J. Barranquero, J. DíEz, and J. J. Del Coz, "Enhancing directed binary trees for multi-class classification," *Information Sciences*, vol. 223, pp. 42–55, 2013.

[20] P. Songsiri, T. Phetkaew, and B. Kijsirikul, "Enhancement of multi-class support vector machine construction from binary learners using generalization performance," *Neurocomputing*, vol. 151, 2015.

[21] M. Basu and T. K. Ho, *Data complexity in pattern recognition*. Springer, 2006.

[22] L. J. Fogel, "Autonomous automata," *Industrial Research*, vol. 4, no. 2, pp. 14–19, 1962.

[23] M. R. Noraini and J. Geraghty, "Genetic algorithm performance with different selection strategies in solving tsp," 2011.

[24] Z. Michaelewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd ed. Springer Verlag, 1996.

[25] P. Larranaga and J. A. Lozano, *Estimation of distribution algorithms: A new tool for evolutionary computation*. Springer Science & Business Media, 2002, vol. 2.

[26] J. Alcalá, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, and F. Herrera, "Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework." *Journal of Multiple-Valued Logic & Soft Computing*, vol. 17, 2011.

[27] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[28] C. Igel and M. Hüsken, "Improving the rprop learning algorithm," in *Proc. sec. Int. ICSC symp. neural computation*, 2000, pp. 115–121.