# Ensemble classification for constraint solver configuration

Lars Kotthoff, Ian Miguel, and Peter Nightingale
{larsko,ianm,pn}@cs.st-andrews.ac.uk

University of St Andrews

**Abstract.** The automatic tuning of the parameters of algorithms and automatic selection of algorithms has received a lot of attention recently. One possible approach is the use of machine learning techniques to learn classifiers which, given the characteristics of a particular problem, make a decision as to which algorithm or what parameters to use. Little research has been done into which machine learning algorithms are suitable and the impact of picking the "right" over the "wrong" technique. This paper investigates the differences in performance of several techniques on different data sets. It furthermore provides evidence that by using a meta-technique which combines several machine learning algorithms, we can avoid the problem of having to pick the "best" one and still achieve good performance.

## 1 Introduction

The automatic selection of algorithms or parameters of algorithms is a problem that has been recognised for decades [19], but systematic investigation has only started relatively recently. Nowadays, systems that incorporate automatic selection of algorithms provide major performance improvements [17, 22, 12].

Most of these approaches use machine learning to uncover how the attributes of a particular problem affect the performance of a set of algorithms or a set of parameters for one algorithm. The designers of such systems face several difficult challenges. Which attributes are needed to capture the important effects on performance? Is the set of training instances representative and are the results likely to be applicable in general? Is the learned classifier overfitted?

The success or failure of such systems does not only depend on these factors, but also on the selection of an appropriate machine learning algorithm. This problem has received little attention so far – the choice is usually justified by the success of the system. But has the most appropriate technique been chosen and how likely is it that the chosen technique is also the best one in general?

We investigate this very question by applying many different machine learning techniques to two different algorithm selection problems. We propose an ensemble classification approach, which uses many different machine learning algorithms. We show that its performance is as good as and sometimes better than the performance of the best individual technique and at the same time more predictable and stable.

## 2 Background

The underlying problem is the algorithm selection problem, which was first described in [19]. Given a choice of algorithms and parameter settings, we want to choose the algorithm-parameter combination that delivers the best performance for a specific problem. Machine learning is an established approach for solving this problem, used for example in the PYTHIA [20] and MULTI-TAC [16] systems.

Two successful approaches in SAT are SATzilla [22] and SATenstein [12]. In CP, CP-Hydra uses a similar approach [17]. MULTI-TAC configured aspects of a solver based on instances. ACE [7] and Bain *et al* [2] learn search heuristics from instances. Genetic algorithms have been shown to be effective as well [1].

In machine learning, the combination of several classifiers is a well-established technique. In so-called ensemble learning [6], there are many different methods for creating different classifiers and combining their predictions, such as bootstrap aggregating, boosting and stacking [21].

The difference between different parameter settings for the same algorithms can have a more profound effect than choosing a different algorithm [14]. Selecting the most suitable machine learning algorithm and parameters for a set of instances is an area of active research in machine learning itself [4].

## 3 Algorithm selection data sets

We investigate the performance of different machine learning algorithms on two algorithm selection problems. First, we decide whether to use g-nogood learning with lazy explanations [9] or not. Second, we consider the nine different versions of the alldifferent constraint detailed in [10]. These problems represent different important areas in constraint solver design. Lazy learning affects the search procedure, while the alldifferent constraint affects strength and efficiency of propagation. For lazy learning in particular, selecting different implementations instead of using a default one can provide a speedup of orders of magnitude.

We selected benchmark instances from Lecoutre's XCSP repository[1] and from our own stock of problems, which includes many instances from previous CSP solver competitions. For lazy learning, we used 2028 problem instances from 46 different problem classes. Within a time limit of 5000 seconds, both the standard and the learning solvers were able to solve 1773 instances. For the alldifferent constraint, we used 1313 different instances from 16 different problem classes. We imposed a time limit of 3600 seconds; 1221 instances were solved by at least one of the candidate solvers within this limit. We took the median of three runs as the run time.

All experiments were run with binaries compiled with g++ version 4.4.3 and Boost version 1.40.0 on machines with 8 core Intel E5430 2.66GHz, 8GB RAM running CentOS with Linux kernel 2.6.18-164.6.1.el5 64Bit. Our reference solver is Minion [8] version 0.9. The binaries and instances required to reproduce the results are available from the authors on request.

[1] http://tinyurl.com/y6hpphs

## 4 Instance attributes and their measurement

We measured 38 attributes of the problem instances. They describe a wide range of features such as constraint and variable statistics and a number of attributes based on the primal graph. The primal graph $g = \langle V, E \rangle$ has a vertex for every CSP variable, and two vertices are connected by an edge iff the two variables are in the scope of a constraint together.

**Edge density** The number of edges in $g$ divided by the number of pairs of distinct vertices.

**Clustering coefficient** For a vertex $v$, the set of neighbours of $v$ is $n(v)$. The edge density among the vertices $n(v)$ is calculated. The clustering coefficient is the mean average of this local edge density for all $v$.

**Normalised degree** The normalised degree of a vertex is its degree divided by $|V|$. The minimum, maximum, mean and median normalised degree are used.

**Normalised standard deviation of degree** The standard deviation of vertex degree is normalised by dividing by $|V|$.

**Width of ordering** The width of a vertex $v$ in an ordered graph, given by the variable ordering, is its number of *parents* (i.e. neighbours that precede $v$ in the ordering). The width of the ordering is the maximum width over all vertices [5] and normalised by the number of vertices.

**Width of graph** The width of a graph is the minimum width over all possible orderings, normalised by the number of vertices.

**Variable domains** The quartiles and mean over the domains of all variables.

**Constraint arity** The quartiles and the mean of the arity of all constraints (the number of variables constrained by it), normalised by the number of constraints.

**Multiple shared variables** The proportion of pairs of constraints that share more than one variable.

**Normalised mean constraints per variable** For each variable, we count the number of constraints on the variable. The mean average is taken, and this is normalised by dividing by the number of constraints.

**Normalised SAC literals** The number of literals pruned by singleton consistency preprocessing, as a proportion of all literals.

**Ratio of auxiliary variables to other variables** The ratio of auxiliary variables to other variables.

**Tightness** The tightness of a constraint is the proportion of disallowed tuples. The tightness is estimated by sampling 1000 random tuples from the variable domains and testing if the tuple satisfies the constraint. The tightness quartiles and the mean over all constraints are used.

**Proportion of symmetric variables** In many CSPs, the variables form equivalence classes where the number and type of constraints a variable is in are the same. The first stage of the algorithm used by Nauty [15] detects this property. Given a partition of $n$ variables generated by this algorithm, we transform this into a number between 0 and 1 by taking the proportion of all pairs of variables which are in the same part of the partition.

**Alldifferent statistics** The size of the union of all variables in an alldifferent constraint divided by the number of variables $|V|$. We used the quartiles and the mean over all alldifferent constraints.

We intended to cover a wide range of possible factors that affect the performance of the different algorithms with these attributes. We normalised attributes that would be specific to problem instances of a particular size. This is based on the intuition that similar instances of different sizes are likely to behave similarly with respect to the investigated algorithms. Computing the attributes took about 15 seconds per instance on average.

Not all attributes were applicable for both of the algorithm selection problems. For lazy learning, the alldifferent statistics did not apply. For alldifferent, we did not use the number of literals that SAC removes because it is different for different versions of the constraint.

## 5 Learning classifiers

We annotated each benchmark instance with the algorithm variant that performed best on it based on the run times of the candidate algorithms for the specific algorithm selection problem. For the decision between the different implementations of the alldifferent constraint, we additionally considered the number of search nodes per second. If the problem instance was solved by no candidate within the timeout, we assigned the annotation "don't know". This data was given to the machine learning algorithms to learn a classifier that, given an instances, predicts which one of the algorithms will have the best performance on it.

We used the WEKA [11] machine learning software through the R interface to learn classifiers. We used almost all of the WEKA algorithms that were applicable to our problems – decision rules, decision trees, Bayesian classifiers, nearest neighbour and neural networks. Our selection is broad and includes almost all major machine learning methodologies. The specific classifiers we used are `BayesNet`, `BFTree`, `ConjunctiveRule`, `DecisionTable`, `FT`, `HyperPipes`, `IBk`, `J48`, `J48graft`, `JRip`, `LADTree`, `MultilayerPerceptron`, `NBTree`, `OneR`, `PART`, `RandomForest`, `RandomTree` and `REPTree`, all of which are described in [21].

We decided to measure the performance of the learned classifiers not in terms of the usual machine learning performance measures, but in terms of misclassification penalty [22]. The misclassification penalty is the additional CPU time we need to solve a problem instance if not choosing the optimal algorithm. This is based on the intuition that we do not particularly care about classifying as many instances correctly as possible; we rather care that the instances that are important to us are classified correctly. The higher the potential gain for an instance, the more important it is to us. If the selected algorithm was not able to solve the problem, we assumed the timeout value minus the CPU time the fastest algorithm took to be the misclassification penalty. This only gives a weak lower bound, but the correct value cannot be estimated easily.

We furthermore decided to assign the maximum misclassification penalty (or the maximum possible gain) as a cost to each instance to bias machine learning towards the instances we care about most. Each instance was attached a cost of $max(1, 1 + \log_2(\texttt{penalty}))$. Note that we used the absolute and not the relative

cost value – if the difference in absolute time is only 0.1 seconds, it does not matter if the relative difference is orders of magnitude.

To combine the different classifiers, we take the predictions of each classifier for an individual problem and choose the one that occurs most often; breaking ties by alphanumeric ordering. A thorough investigation in [3] showed that voting performs better in general than other techniques.

For each data set of the different algorithm selection problems, we generated partitions as follows. First, we removed the instances of a randomly selected problem class. Then we removed about 33% of the remaining instances at random. This data was used for training and the removed instances for testing. For both data sets, we generated 10 different partitions of approximately equal size this way.

The most important issue we are addressing is the generality of the learned classifier – given its performance on the data set we are using for testing, will it perform equally well on unknown data? There are two different cases. The unknown data could be new instances from a problem class which the classifier has seen before or the data could consist of unknown instances from unknown problem classes. We address both scenarios by removing individual problem classes and random instances from the original data set. Using this method, we test for overfitted classifiers at the same time.

We ran each machine learning algorithm on each training partition and evaluated its performance in terms of misclassification penalty through stratified 10-fold cross-validation [13]. The median of the 10 folds denotes our overall performance estimate. We then evaluated the performance of each of these classifiers on the respective test partition.

## 6 Results

Figure 1 shows the performance on the different partitions. It is obvious that the performance on one set of data, even when using cross-validation, is not a good predictor of the performance on another set of data, as shown by the length of the arrows. In only one of twenty cases, the classifier which performs best on the training partition is also the best one on the test partition. In two cases, the best classifier actually becomes the worst on new data. It also becomes clear that this effect is attenuated by using the ensemble classifier – in almost all cases, the performance differences on different sets of data are less pronounced, as denoted by the lengths of the arrows starting at the crosses (ensemble classifier) versus the ones starting at the circles (single classifier). The ensemble classifier is more robust in that its performance is predictable more reliably. The algorithm with the best average performance over all the data was `BFTree`; the difference to the ensemble classifier was about 1%.

The figure furthermore shows that for several different partitions, our ensemble classifier performs better than the "best" individual algorithm on a single partition most of the time and is often close to or even better than the individual classifier that would be the best on unseen data. We achieve significant improve-

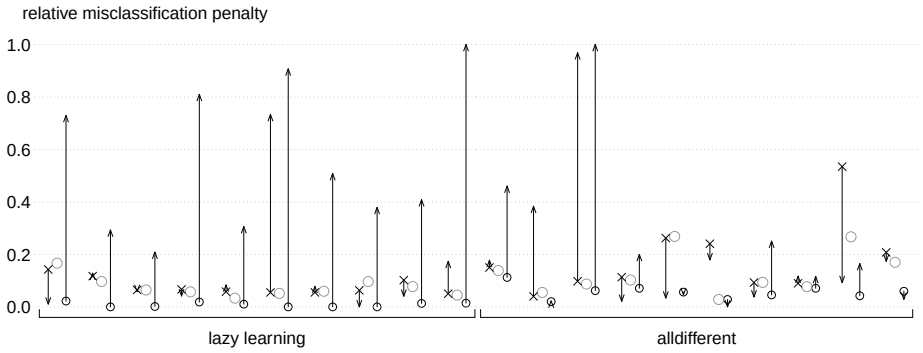


**Fig. 1.** Classifier performance on the different partitions. The misclassification penalties were normalised by the best classifier across all partitions (i.e. the misclassification penalty of the best classifier is always 1) and then scaled between best and worst classifier to make the different data sets comparable (i.e. the best classifier is now 0 and the worst 1). In absolute terms, the difference between best and worst is up to several orders of magnitude. The black circles show the performance of the classifier which performed best during cross-validation. The larger, grey circles show the performance of the classifier which was the best one on the test partition. The cross denotes the performance of the ensemble classifier during cross-validation. The end of the arrows denotes the performance on the test partition for best individual and ensemble classifiers. The length of the arrow denotes the uncertainty of the prediction of classifier performance from cross-validation.

ments without time-consuming inspection and evaluation of individual machine learning algorithms to select the most suitable one and tune its parameters.

Our results do not depend on a large number of machine learning algorithms. Figure 2 shows the results with just three algorithms from different machine learning methodologies – `BayesNet`, `MultilayerPerceptron` (a neural network algorithm) and `J48` (an implementation of the well-known C4.5 algorithm [18]). The improvements over using a single classifier are comparable to the ones shown in Figure 1. In some cases we even achieve an improvement over the best individual classifier through the combination of the predictions of several classifiers. Note that these three individual algorithms were not selected because of particularly good overall performance – none of them was the best on average and they often performed worse than the ensemble classifier.

In terms of solve time, the ensemble classification approach improves over always making a default decision. The improvement is substantial for lazy learning and marginal for alldifferent.

## 7 Conclusions

We have presented a thorough and in-depth investigation into the variability of the performance of different machine learning algorithms and techniques on two real-world algorithm selection problems. We based our investigation on experimental results for a large number of diverse problems and a large number of

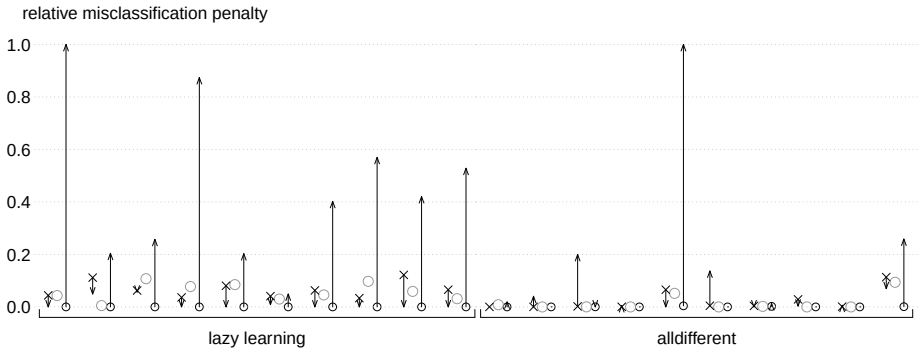


**Fig. 2.** Classifier performance for three different classifiers. Note that in five cases the performance of the ensemble classifier was better than that of an individual best one.

different machine learning techniques. Although much research has been done in the field of algorithm selection and algorithm tuning, no similar evaluation of the methodology has been undertaken, to the best of our knowledge.

Our results conclusively show that the performance of a machine learning algorithm is so variable that predictions as to its generality and performance on new data cannot be made without investing significant effort into substantiating these claims. Furthermore, an algorithm which may have a low performance and therefore appear unsuitable on test data has the potential for performing much better on unknown data.

The technique we are proposing for the configuration of constraint solvers, ensemble classification by combining the predictions of several classifiers by majority vote, improves on this. Our experiments provide strong evidence that its performance on several data sets will in general be better than the performance of an individual best classifier on one data set. Indeed it will be close to the performance achieved by the classifier in the ensemble which is the best for a given data set. We furthermore observed cases in which the ensemble classifier was better than a single classifier even on a single data set.

While combining several classifiers adds significant overhead in the offline phase when more classifiers need to be learned, the overhead in the online phase when new instances are classified was negligible in our experiments. The time required to compute the instance attributes was much higher than the time for running additional classifiers and combining their predictions in all cases. In particular, running an individual classifier on a single problem instance only takes a few milliseconds on average.

The main advantage of ensemble classification is that individual machine learning algorithms can be combined without intrinsic knowledge about each one of them. The level of machine learning expertise required is reduced significantly without affecting the results significantly. Note that this does not mitigate the need for domain knowledge to select relevant features for example. Ensemble classification enables practitioners without a lot of machine learning knowledge to apply machine learning to their problems.

**Acknowledgements**

We thank Chris Jefferson for the description of one of the problem attributes used in the analysis, Jesse Hoey for useful discussions about machine learning, and anonymous reviewers for their feedback. Lars Kotthoff is supported by a SICSA studentship. This work was supported by EPSRC grant EP/H004092/1.

## References

1. Ansótegui, C., Sellmann, M., Tierney, K.: A gender-based genetic algorithm for the automatic configuration of algorithms. In: CP. pp. 142–157 (2009)
2. Bain, S., Thornton, J., Sattar, A.: Evolving Variable-Ordering heuristics for constrained optimisation. In: CP. p. 732736 (2005)
3. Bauer, E., Kohavi, R.: An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. Mach. Learn. 36(1-2), 105–139 (1999)
4. Chen, F., Jin, R.: Active algorithm selection. In: AAAI. pp. 534–539 (2007)
5. Dechter, R.: Constraint Processing. Elsevier Science (2003)
6. Dietterich, T.G.: Ensemble methods in machine learning. In: First International Workshop on Multiple Classifier Systems. pp. 1–15 (2000)
7. Epstein, S., Freuder, E., Wallace, R., Morozov, A., Samuels, B.: The adaptive constraint engine. In: CP. pp. 525–542 (2002)
8. Gent, I., Jefferson, C., Miguel, I.: Minion: A fast scalable constraint solver. In: ECAI. pp. 98–102 (2006)
9. Gent, I., Miguel, I., Moore, N.: Lazy explanations for constraint propagators. In: PADL (2010)
10. Gent, I., Miguel, I., Nightingale, P.: Generalised arc consistency for the alldifferent constraint: An empirical survey. Artif. Intell. 172(18), 1973–2000 (2008)
11. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.: The WEKA data mining software: An update. SIGKDD Explorations 11(1) (2009)
12. KhudaBukhsh, A., Xu, L., Hoos, H., Leyton-Brown, K.: SATenstein: Automatically building local search SAT solvers from components. In: IJCAI. pp. 517–524 (2009)
13. Kohavi, R.: A study of Cross-Validation and bootstrap for accuracy estimation and model selection. In: IJCAI. pp. 1137–1143 (1995)
14. Lavesson, N., Davidsson, P.: Quantifying the impact of learning algorithm parameter tuning. In: AAAI. pp. 395–400 (2006)
15. McKay, B.: Practical graph isomorphism. In: Numerical mathematics and computing. pp. 45–87 (1981)
16. Minton, S.: Automatically configuring constraint satisfaction programs: A case study. Constraints 1(1/2), 7–43 (1996)
17. O'Mahony, E., Hebrard, E., Holland, A., Nugent, C., O'Sullivan, B.: Using case-based reasoning in an algorithm portfolio for constraint solving. In: Irish Conf. on AI (2008)
18. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann (1993)
19. Rice, J.: The algorithm selection problem. Adv. in Computers 15, 65–118 (1976)
20. Weerawarana, S., Houstis, E.N., Rice, J.R., Joshi, A., Houstis, C.E.: PYTHIA: a knowledge-based system to select scientific algorithms. ACM Trans. Math. Softw. 22(4), 447–468 (1996)
21. Witten, I., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann (2005)
22. Xu, L., Hutter, F., Hoos, H., Leyton-Brown, K.: SATzilla: Portfolio-based algorithm selection for SAT. JAIR 32, 565–606 (2008)