A Preliminary Evaluation of Machine Learning in Algorithm Selection for Search Problems

Lars Kotthoff, Ian P. Gent, Ian Miguel
University of St Andrews
{*larsko,ipg,ianm*}@*cs.st-andrews.ac.uk*

Abstract

Machine learning is an established method of selecting algorithms to solve hard search problems. Despite this, to date no systematic comparison and evaluation of the different techniques has been performed and the performance of existing systems has not been critically compared to other approaches. We compare machine learning techniques for algorithm selection on real-world data sets of hard search problems. In addition to well-established approaches, for the first time we also apply statistical relational learning to this problem. We demonstrate that most machine learning techniques and existing systems perform less well than one might expect. To guide practitioners, we close by giving clear recommendations as to which machine learning techniques are likely to perform well based on our experiments.

Introduction

The technique of portfolio creation and algorithm selection has received a lot of attention in areas of artificial intelligence that deal with solving computationally hard problems recently (Xu et al. 2008; O'Mahony et al. 2008). The current state of the art is such that often there are many algorithms and systems for solving the same kind of problem; each with its own performance on a particular problem.

It has long been recognised that there is no single algorithm or system that will deliver the best performance in all cases (Wolpert and Macready 1997). For this reason, recent research has focussed on creating *algorithm portfolios*, which contain a selection of state of the art algorithms. To solve a particular problem with this portfolio, a preprocessing step is run where the suitability of each algorithm in the portfolio for the problem at hand is assessed. This step often involves some kind of machine learning, as the actual performance of each algorithm on the given, unseen problem is unknown.

The algorithm selection problem was first described many decades ago in (Rice 1976) and numerous systems that employ machine learning techniques have been developed (Xu et al. 2008; O'Mahony et al. 2008; Pulina and Tacchella 2009; Weerawarana et al. 1996). While there has been some small-scale work to compare the performance of different machine learning algorithms (e.g. (Pulina and Tacchella 2009)), there has been no comparison of the machine learning methodologies available for algorithm selection and large-scale evaluation of their performance to date.

The systems that do algorithm selection usually justify their choice of a machine learning methodology (or a combination of several) with their performance compared to individual algorithms and do not critically assess the real performance – could we do as good or even better by using just a single algorithm instead of having to deal with portfolios and complex machine learning?

This paper presents a comprehensive comparison of machine learning paradigms for tackling algorithm selection and evaluating their performance on data sets used in real-world systems. We furthermore compare our results to existing systems and to a simple "winner-takes-all" approach where the best overall algorithm is always selected, which performs surprisingly well in practice. Based on the results of these extensive experiments and additional statistical simulations, we give recommendations as to which machine learning techniques should be considered when performing algorithm selection. We identify support vector machines as a particularly promising set of techniques.

Background

We are addressing an instance of the algorithm selection problem (Rice 1976), which, given variable performance among a set of algorithms, is to choose the best candidate for a particular problem instance. Machine learning is an established method of addressing this problem (Lobjois and Lemâitre 1998; Fink 1998). Given the performance of each algorithm on a set of training problems, we try to predict the performance on unseen problems.

An algorithm portfolio (Gomes and Selman 2001; Leyton-Brown et al. 2003) consists of a set of algorithms. A subset is selected and applied sequentially or in parallel to a problem instance, according to some schedule. The schedule may involve switching between algorithms while the problem is being solved (e.g. (Lagoudakis and Littman 2000; Streeter and Smith 2008)). We consider the problem of choosing the best algorithm from the portfolio (i.e. a subset of size 1) and using it to solve the particular problem instance to completion because the widest range of machine learning techniques are applicable in this context. Some of the techniques are also valid in other contexts – performance predictions can easily be used to devise a schedule with time allocations for each algorithm in the portfolio, which can then be applied sequentially or in parallel. Therefore some of our results are also applicable to other approaches.

There have been many systems that use algorithm portfolios in some form developed over the years and an exhaustive list is beyond the scope of this paper. For a comprehensive survey, see for example (Smith-Miles 2009). One of the earliest systems was Prodigy (Carbonell et al. 1991), a planning system that uses various machine learning methodologies to select from search strategies. PYTHIA (Weerawarana et al. 1996) is more general and selects from among scientific algorithms. (Borrett, Tsang, and Walsh 1996) employed a sequential portfolio of constraint solvers. More recently, (Guerri and Milano 2004) use a decision-tree based technique to select among a portfolio of constraint- and integer-programming based solution methods for the bid evaluation problem. In the area of hard combinatorial search problems, a successful approach in satisfiability (SAT) is SATzilla (Xu et al. 2008). In constraint programming, CP-Hydra uses a similar approach (O'Mahony et al. 2008). The AQME (Pulina and Tacchella 2009) system does algorithm selection for finding satisfying assignments for quantified Boolean formulae.

A closely related field is concerned with so-called hyperheuristics. In the context of algorithm selection, a hyperheuristic would for example choose a heuristic to choose a machine learning technique to select algorithms from a portfolio for a specific scenario. An introduction can be found in (Burke et al. 2003).

Algorithm selection methodologies

An established approach to solve the algorithm selection problem is to use machine learning. In an ideal world, we would know enough about the algorithms in the portfolio to formulate rules to select a particular one based on certain characteristics of a problem to solve. In practice, this is not possible except in trivial cases. For complex algorithms and systems, like the ones mentioned above, we do not understand the factors that affect the performance of a specific algorithm on a specific problem enough to make the decisions the algorithm selection problem requires with confidence.

Several machine learning methodologies are applicable here. We present the most prevalent ones below. In addition to these, we use a simple majority predictor that always predicts the portfolio algorithm that has the best performance most often on the set of training instances ("winner-takes-all" approach) for comparison purposes. This provides an evaluation of the real performance improvement over manually picking the best algorithm from the portfolio. For this purpose, we use the WEKA (Hall et al. 2009) `ZeroR` classifier implementation.

Case-based reasoning

Case-based reasoning informs decisions for unseen problems with knowledge about past problems. An introduction to the field can be found in e.g. (Riesbeck and Schank 1989). The idea behind case-based reasoning is that instead of trying to construct a theory of what characteristics affect the performance, examples of past performance are used to infer performance on new problems.

The main part of a case-based reasoning system is the case base. We use the WEKA `IBk` nearest-neighbour classifier with 1, 3, 5 and 10 nearest neighbours considered as our case-based reasoning algorithms. The case base consists of the problem instances we have encountered in the past and the best portfolio algorithm for each of them – the set of training instances and labels. Each case is a point in n-dimensional space, where n is the number of attributes each problem has. The nearest neighbours are determined by calculating the Euclidean distance. While this is a very weak form of case-based reasoning, it is consistent with the observation above that we simply do not have more information about the problems and portfolio algorithms that we could encode in the reasoner.

We use the AQME system (Pulina and Tacchella 2009) as a representative of this methodology.

Classification

Intuitively, algorithm selection is a simple classification problem – label each problem instance with the portfolio algorithm that should be used to solve it. We can solve this classification problem by learning a classifier that discriminates between the algorithms in the portfolio based on the characteristics of the problem. A set of labelled training examples is given to the learner and the learned classifier is then evaluated on a set of test instances.

We use the WEKA `LADTree`, `J48graft`, `JRip`, `BayesNet`, `ConjunctiveRule`, `J48`, `BFTree`, `HyperPipes`, `PART`, `RandomTree`, `OneR`, `REPTree`, `RandomForest`, `DecisionTable`, `LibSVM` (with radial basis and sigmoid function kernels), `FT`, `MultilayerPerceptron` and `AdaBoostM1` classifiers. Our selection is large and inclusive and contains classifiers that learn all major types of classification models. In addition to the WEKA classifiers, we used a custom one that assumes that the distribution of the class labels for the test set is the same as for the training set and samples from this distribution without taking features into account.

We considered (Gent et al. 2010) to evaluate the performance of this methodology.

Regression

Instead of considering all portfolio algorithms together and selecting the one with the best performance, we can also try to predict the performance of each algorithm on a given problem independently and then select the best one based on the predicted performance measures. The downside is that instead of running the machine learning once per problem, we need to run it for each algorithm in the portfolio for a single problem.

Regression is usually performed on the runtime of an algorithm on a problem. (Xu et al. 2008) predict the logarithm of the runtime because they "have found this log transformation of runtime to be very important due to the large variation in runtimes for hard combinatorial problems."

We use the WEKA `LinearRegression`, `REPTree`, `LibSVM` (ε and ν), `SMOreg` and `GaussianProcesses` learners to predict both the runtime and the logarithm of the runtime. Again we have tried to be inclusive and add as many different regression learners as possible regardless of our expectations as to their suitability or performance.

(Xu et al. 2008) is the system we use to evaluate the performance of regression.

Statistical relational learning

Statistical relational learning is a relatively new discipline of machine learning that attempts to predict complex structures instead of simple labels (classification) or values (regression) while also addressing uncertainty. An introduction can be found in (Getoor and Taskar 2007). For algorithm selection, we try to predict the performance ranking of portfolio algorithms on a particular problem.

We use the support vector machine SVM^{rank} instantiation[1] of SVM^{struct} (Joachims 2006). It was designed to predict ranking scores. Instances are labelled and grouped according to some criteria. The labels are then ranked within each group. We can use the system unmodified for our purposes and predict the ranking score for each algorithm on each problem. We left the parameters at their default values and used a value of 0.1 for the convergence parameter ε except in cases where the model learner did not converge within an hour. In these cases, we set $\varepsilon = 0.5$.

To the best of our knowledge, statistical relational learning has never before been applied to algorithm selection.

Evaluation data sets

We evaluate and compare the performance of the approaches mentioned above on five real-world data sets of hard algorithm selection problems. We take three sets from the training data for SATzilla 2009. This data consists of SAT instances from three categories – handmade, industrial and random. They contain 1181, 1183 and 2308 instances, respectively. The authors use 91 attributes for each instance and select a SAT solver from a portfolio of 19 solvers[2]. We compare the performance of each of our methodologies to the SATzilla system which was trained using this data and won several medals in SAT competitions.

The fourth data set comes from the QBF Solver Evaluation 2010[3] and consists of 1368 QBF instances from the main, small hard, 2QBF and random tracks. 46 attributes are calculated for each instance and we select from a portfolio of 5 QBF solvers. Each solver was run on each instance for at most 3600 CPU seconds. If the solver ran out of memory or was unable to solve an instance, we assumed the timeout value for the runtime. The experiments were run on a machine with a dual 4 core Intel E5430 2.66 GHz processor and 16 GB RAM. We compare the performance to that of the AQME system, which was trained on parts of this data.

Our last data set is taken from (Gent et al. 2010) and selects from a portfolio of two solvers for a total of 2028 constraint problem instances with 17 attributes each from 46 problem classes. We compare our performance to the classifier described in the paper.

We chose the data sets because they represent algorithm selection problems from three areas where the technique of algorithm portfolios has attracted a lot of attention recently. For all sets, reference systems exist that we can compare against. Furthermore, the number of algorithms in the respective portfolios for the data sets is different.

It should be noted that the systems we are comparing against are given an unfair advantage. They have been trained on at least parts of the data that we are using for the evaluation. Their performance was assessed on the full data set as a black box system. The machine learning algorithms we use however are given disjoint sets of training and test instances.

Methodology

The focus of our evaluation is the performance of the machine learning algorithms. Additional factors which would impact the performance of an algorithm selection system in practice are not taken into account. These factors include the time to calculate problem features and additional considerations for selecting algorithms, such as memory requirements.

We measured the performance of the learned models in terms of misclassification penalty. The misclassification penalty is the additional CPU time we need to solve a problem instance if not choosing the best algorithm from the portfolio, i.e. the difference between the CPU time the selected algorithm required and the CPU time the fastest algorithm would have required. If the selected algorithm was not able to solve the problem, we assumed the timeout value minus the fastest CPU time to be the misclassification penalty. This only gives a weak lower bound, but we cannot determine the correct value without running the algorithm to completion.

For the classification learners, we attached the misclassification penalty as a weight to the respective problem instance during the training phase. The intuition is that instances with a large performance difference between the algorithms in the portfolio are more important to classify correctly than the ones with almost no difference.

The handling of missing attribute values was left up to the specific machine learning system. We estimated the performance of the learned models using ten-fold stratified cross-validation (Kohavi 1995). We summed the misclassification penalties across the individual folds to provide an estimate of the misclassification penalty on the whole data set assuming that it had never been seen before.

For each data set, we used two sets of features – the full set and the subset of the most predictive features. We used WEKA's `CfsSubsetEval` attribute selector with the `BestFirst` search method, again with default parameters, to determine the most predictive features for the different machine learning methodologies. We treated SVM^{rank} as a black box algorithm and therefore did not determine the most predictive features for it.

We performed a full factorial set of experiments where we ran each machine learning algorithm of each methodology on each data set. We also evaluated the performance with thinned out training data. We randomly deleted 25, 50 and

[1] http://www.cs.cornell.edu/People/tj/svm˙light/svm˙rank.html

[2] http://www.cs.ubc.ca/labs/beta/Projects/SATzilla/

[3] http://www.qbflib.org/index˙eval.php

75% of the problem-portfolio algorithm pairs in the training set. We thus simulated partial training data where not all algorithms in the algorithm portfolio had been run on all problem instances.

To evaluate the performance of the existing algorithm selection systems, we ran them on the full, unpartitioned data set. The misclassification penalty was calculated in the same way as for the machine learning algorithms.

Machine learning algorithm parameters

We used the default parameters for all machine learning algorithms unless noted otherwise. It is likely that the performance could be improved by tuning the parameters, but there are a number of reasons not to do this in this study.

First, finding the optimal parameters for all the considered algorithms and data sets would require a lot of resources. Every machine learning algorithm would need to be tuned for every data set; there are 180 such tuning pairs in our case. We could of course limit the tuning to a smaller number of machine learning algorithms, but then the question of how to choose those algorithms arises. The ones commonly used in the literature are not necessarily the best ones. We therefore need to identify the machine learning techniques with the most promising performance as a starting point for tuning.

Second, our intention in this paper is to compare the same machine learning algorithms across the different data sets to assess their performance. Parameter tuning would result in a large number of algorithm-parameter pairs and it is very unlikely that we would be able to draw consistent conclusions across all data sets. For new algorithm selection problems, we would not be able to recommend a specific set of machine learning techniques. Our purpose is not to find the best machine learning algorithm for a single data set, but one that has consistently good performance across all data sets.

Finally, the parameters of machine learning techniques are not always tuned in existing algorithm selection systems. Researchers who use algorithm portfolios are usually not machine learning experts. If the performance is already good enough (i.e. better than existing systems), there is little incentive to invest a huge amount of effort into tuning the parameters with no guarantee of a proportional return in performance improvement.

Experimental results

We first present and analyse the results for each machine learning methodology and then take a closer look at the individual machine learning algorithms and their performance.

The misclassification penalty in terms of the majority predictor for all methodologies and data sets is shown in Figure 1. At first glance, no methodology seems to be inherently superior. This result is not surprising, as it is predicted by the "No free lunch" theorem (Wolpert and Macready 1997). We were however surprised by the good performance of the majority predictor, which in particular delivers excellent performance on the industrial SAT data set. The SVM^{rank} relational approach is similar to the majority predictor when it delivers good performance. We were also surprised by the performance of the SATzilla system, which performs significantly worse than the majority predictor on two of the three data sets it has been trained on. This only applies to the machine learning aspect however; in practice additional techniques such as using a presolver improves its performance.

Most of the literature completely fails to compare to the majority predictor, thus creating a misleading impression of the true performance. As our results demonstrate, always choosing the best algorithm from a portfolio without any analysis or machine learning can *significantly outperform* more sophisticated approaches.

A statistical significance test (non-parametric Wilcoxon signed rank test) of the difference between performing regression on the runtime and log of the runtime showed that the difference was not significant. We estimated the performance with different data by choosing 1000 bootstrap samples from the set of data sets and comparing the performance of each machine learning algorithm for both types of regression. Regression on the log of the runtime has a higher chance of better performance – with a probability of $\approx$64% it will be at least as good as regression on the runtime which only has a chance of $\approx$52% of being at least as good as regression on the log. We therefore only consider regression on the log of the runtime, but still show normal regression for comparison.

Figure 2 shows the results for the set of the most predictive features. At first glance, the results look very similar to the ones with the full set of features. We performed the same statistical significance test as above on the difference between using all the features and selecting the most important ones. Again the difference was not statistically significant. A bootstrapping estimate as described above indicated that the probability of the full feature set delivering results at least as good as the set of the most important features is $\approx$67%, whereas the probability of the smaller set of features being as good or better is only $\approx$44%. Therefore, we only consider the full set of features in the remainder of this paper because it does not require the additional feature selection step.

The effects of thinning out the training set were different across the data sets and are shown in Figure 3. On the SAT data sets, the performance varied seemingly at random; sometimes increasing with thinned out training data for one machine learning methodology while decreasing for another one on the same data set. On the QBF data, the performance decreased across all methodologies as the training data was thinned out while it increased on the CSP data set. Statistical relational learning was almost unaffected in most cases.

The size of the algorithm portfolio did not have a measurable effect on the performance of the different machine learning methodologies. Our intuition was that as the size of the portfolio increases, classification would perform less well because the learned model would be more complex. In practice however it turned out that the number of portfolio algorithms selected at all is small in all cases.

There is no clear conclusion to be drawn from these results as the effect differs across data sets and methodologies. They however suggest that, as we are dealing with inherently noisy data, deleting a proportion of the training data may reduce the noise and improve the performance of the machine learning algorithms. At the very least, not running all algorithms on all problems because of resource constraints is unlikely to

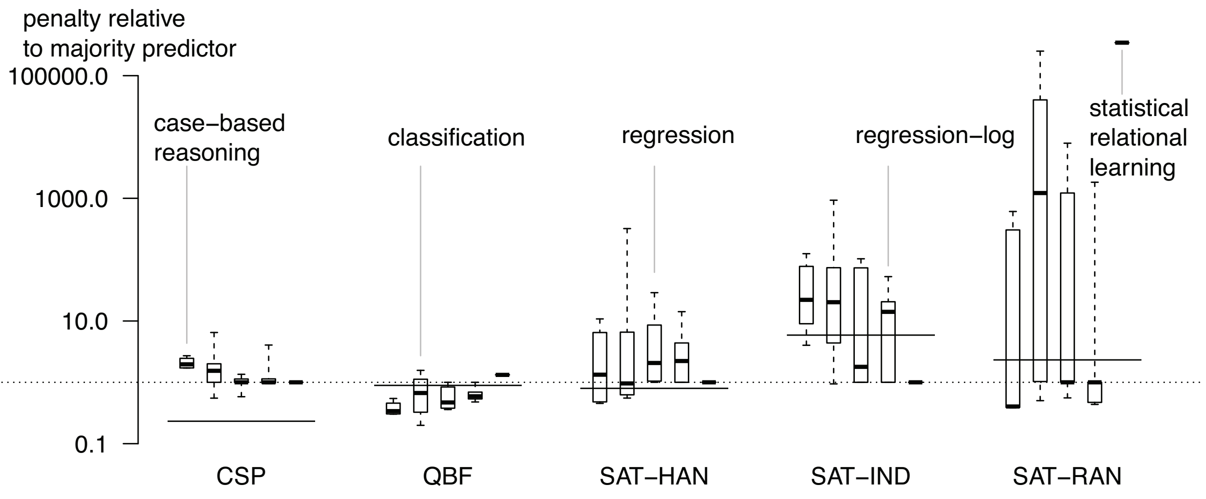

Figure 1: Experimental results with full feature sets and training data across all methodologies and data sets. The plots show the 0th (bottom line), 25th (lower edge of box), 50th (thick line inside box), 75th (upper edge of box) and 100th (top line) percentile of the performance of the machine learning algorithms for a particular methodology (4 for case-based reasoning, 19 for classification, 6 for regression and 1 for statistical relational learning). The boxes for each data set are, from left to right, case-based reasoning, classification, regression, regression on the log and statistical relational learning. The performance is shown as a factor of the simple majority predictor which is shown as a dotted line. Numbers less than 1 indicate that the performance is better than that of the majority predictor. The solid lines for each data set show the performance of the systems we compare against ((Gent et al. 2010) for the CSP data set, (Pulina and Tacchella 2009) for the QBF data set and (Xu et al. 2008) for the SAT data sets).

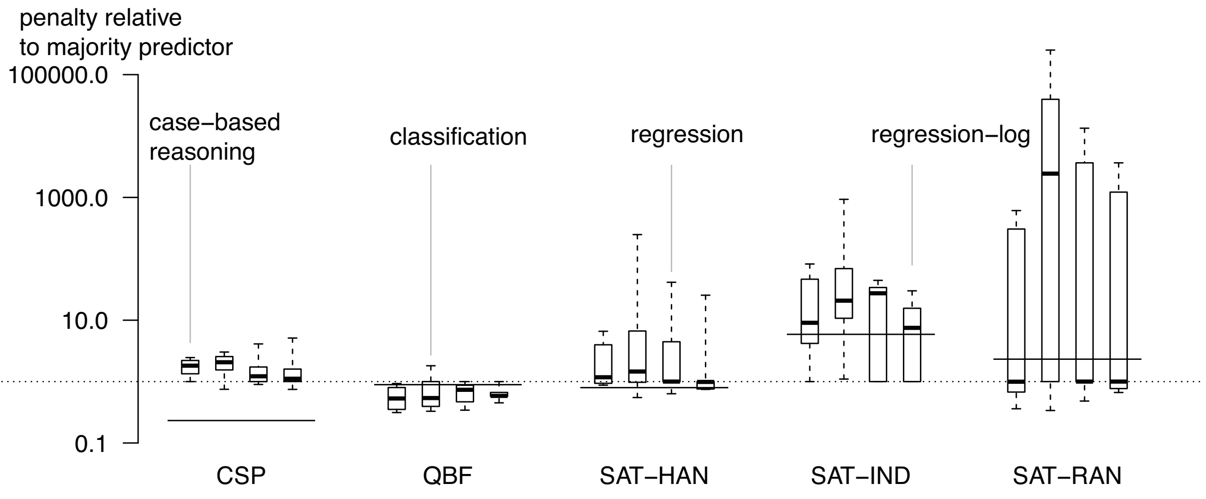

Figure 2: Experimental results with reduced feature sets across all methodologies and data sets. For each data set, the most predictive features were selected and used for the machine learning.

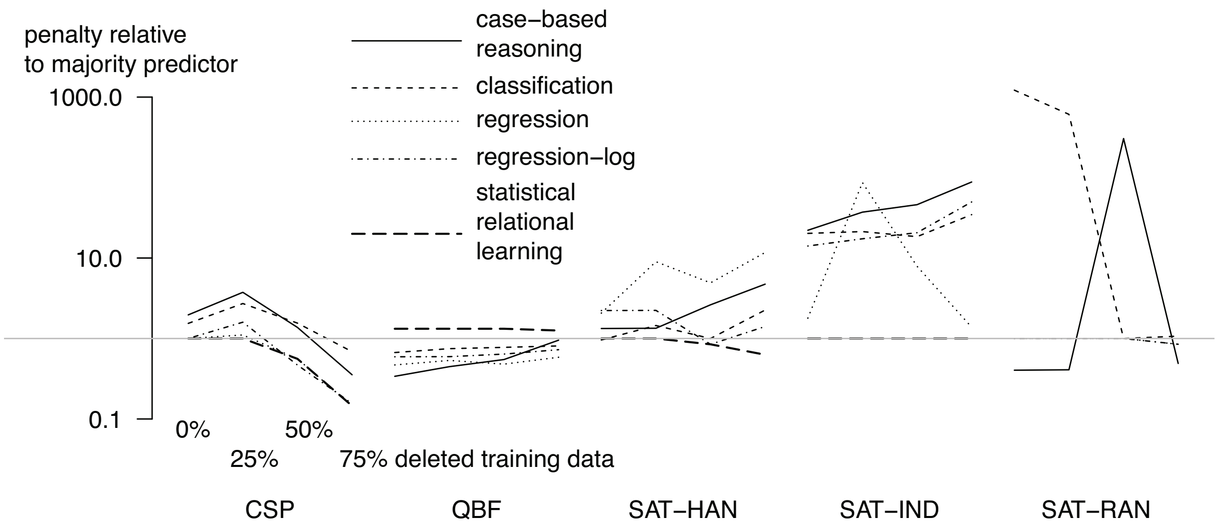

Figure 3: Experimental results with full feature sets and thinned out training data across all methodologies and data sets. The lines show the median penalty (thick line inside the box in the previous plots) for 0%, 25%, 50% and 75% of the training data deleted. The performance is shown as a factor of the simple majority predictor which is shown as a grey line. Numbers less than 1 indicate that the performance is better than that of the majority predictor.

have a large negative impact on performance as long as most algorithms are run on most problems.

As it is not obvious from the results which methodology is the best, we again used bootstrapping to estimate the probability of being the best performer for each one. We sampled, with replacement, from the set of data sets and for each methodology from the set of machine learning algorithms used and calculated the ranking of the median performances across the different methodologies. Repeated 1000 times, this gives us the likelihood of an average algorithm of each methodology being ranked 1st, 2nd and 3rd. We chose to compare the median performance because there was no machine learning algorithm with a clearly better performance than all of the others and algorithms with a good performance on one data set would perform much worse on different data. We used the same bootstrapping method to estimate the likelihood that an average machine learning algorithm of a certain methodology would perform better than the simple majority predictor. The probabilities are summarised in Table 1.

Based on the bootstrapping estimates, it looks like *case-based reasoning* is the methodology most likely to give the best performance. The methodology most likely to deliver good performance in terms of being at least as good as the majority predictor however is *regression* on the log of the runtime. Statistical relational learning has the most consistent performance and gets better as the training data is thinned out, but never even matches the performance of the majority predictor on the full set of training data. This means that most of the machine learning methodologies are outperformed by the majority predictor.

We observe that the majority classifier is very likely to have performance equivalent to or even better than sophisticated machine learning approaches. Its advantages over all the other approaches are its simplicity and that no problem features need to be computed, a task that can further impact the overall performance negatively.

Determining the best machine learning

The results are not clear – no methodology has a high probability of giving performance at least as good as the majority predictor. But should we focus on choosing a methodology first, or is choosing an algorithm independent of the methodology likely to have a larger impact on performance? We calculated the coefficients of variation of the misclassification penalty for each data set for the algorithms within a methodology and for the median performance across methodologies. If the coefficient of variation within a machine learning methodology was small, there would be little variability in performance once the decision which methodology to use was fixed. That is, after deciding to use e.g. regression, it would make little difference what specific machine learning algorithm for regression to choose. Figure 4 compares the results. Choosing a machine learning algorithm independently of methodology is likely to have a far greater impact than choosing the methodology and then the algorithm.

What is the machine learning algorithm with the overall best performance? There is no clear overall winner; a different algorithm is the best one on each data set. We therefore concentrate on finding an algorithm with *good* and reliable, not necessarily the best performance. It is unlikely that one of the best algorithms here will be the best one on new data, but an algorithm with good performance is more probable to exhibit good performance on unseen data. We performed a bootstrap estimate of whether the performance of each individual algorithm would be at least as good as that of the majority predictor. The results are summarised in Table 2.

The results are surprisingly clear – for *untuned* machine learning algorithms, doing regression on the log of the run-

methodology	rank with full training data 1	2	3	at least as good as majority predictor	rank 1 with deleted training data 25%	50%	75%
case-based reasoning	**41%**	27%	27%	43%	36%	27%	13%
classification	17%	21%	26%	39%	10%	13%	4%
regression-log	13%	19%	19%	**50%**	2%	**39%**	18%
statistical relational learning	30%	**33%**	**28%**	0%	**53%**	20%	**66%**

Table 1: Probabilities for each methodology ranking at a specific place with regard to the median performance of its algorithms and probability that this performance will be better than that of the majority predictor. We also show the probabilities that the median performance of the algorithms of a methodology will be the best for thinned out training data. All probabilities are rounded to the nearest percent. The highest probabilities for each rank are in **bold**.

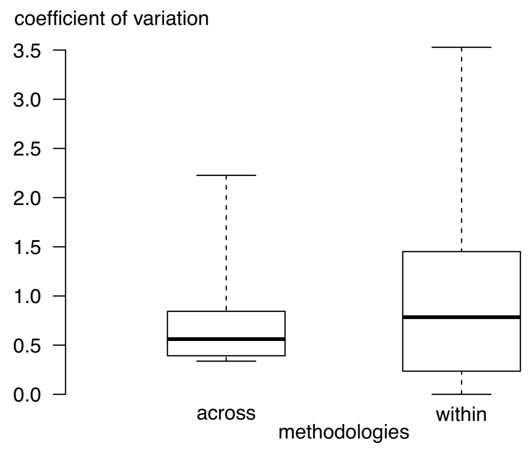

Figure 4: Comparison of the coefficients of variation of misclassification penalty across and within machine learning methodologies across the different data sets.

machine learning methodology	machine learning algorithm	at least as good as (better than) majority predictor
regression-log	LibSVM ε	100% (40%)
regression-log	LibSVM ν	100% (40%)
classification	LibSVM sigmoid function	80% (0%)
case-based reasoning	IBk 10 neighbour	60% (60%)
classification	AdaBoostM1	60% (60%)
classification	LibSVM radial basis function	60% (60%)
classification	PART	60% (60%)
classification	RandomForest	60% (60%)
regression-log	SMOreg	60% (60%)

Table 2: Probability that a particular machine learning algorithm will perform at least as good as (better than) the majority predictor on the full training data. We only show ones with a probability higher than 50%. All probabilities are rounded to the nearest percent.

time with a `LibSVM` learner will always be at least as good as the majority predictor and has a substantial chance of being better than it. Classification with a sigmoid function `LibSVM` however only emulates the majority predictor when it delivers good performance. The other algorithms have a lower chance of being at least as good as the majority predictor, but are more likely to be better – in particular they are more likely than not to be better – and are therefore reasonable choices as well. The `AdaBoostM1` and `RandomForest` algorithms were also the best ones on an individual data set.

The high probabilities of `LibSVM` ε and ν performing regression on the runtime being at least as good as the majority predictor are consistent with the observation that regression on the log of the runtime is the paradigm most likely to achieve such performance in Table 1.

Our intuition is that these particular machine learning algorithms perform well because they are relatively tolerant to noise. In the case of `AdaBoostM1`, which *is* susceptible to noise, the boosting of the initial weak learner seems to make up for this. This is probably because almost all the algorithms we have tried are weak in the sense that they are not (much) better than the majority predictor and therefore an algorithm that assumes weak learners has an advantage. The good performance of the `RandomForest` ensemble method is consistent with observations in (Kotthoff, Miguel, and Nightingale 2010).

Conclusions

In this paper, we investigated the performance of several different machine learning methodologies and algorithms for algorithm selection on several real-world data sets. We compared the performance not only among these methodologies and algorithms, but also to existing algorithm selection systems. To the best of our knowledge, we presented the first comparison of machine learning methodologies applicable to algorithm selection. In particular, statistical relational learning has never been applied to algorithm selection before.

We used the performance of the simple majority predictor as a baseline and evaluated the performance of everything else in terms of it. This is a less favourable evaluation than usually found in the literature, but gives a better picture of the real performance improvement of algorithm portfolio techniques over just using a single algorithm. This method of evaluation clearly demonstrates that the machine learning performance of existing algorithm selection systems is not as good as commonly perceived. In particular, much more sophisticated (and computationally expensive) approaches have inferior performance in some cases.

The perhaps surprising result of our evaluation was that most machine learning techniques are unlikely to even match the performance of the majority predictor, much less exceed it. We believe that this finding could provide an incentive for machine learning research to improve the current state of the art in this area. This applies in particular to statistical relational learning.

We provide strong evidence that when performing algorithm selection, choosing the right machine learning algorithm is much more important than deciding on the right methodology and then choosing an algorithm. We furthermore demonstrate that methodologies and algorithms that have the best performance on one data set do not necessarily have good performance on all data sets.

Another non-intuitive result of our investigation is that deleting parts of the training data can help improve the overall performance, presumably by reducing some of the noise inherent in the empirical runtime data.

The probabilities in Table 1 suggest that the majority predictor is a feasible alternative to more sophisticated approaches, likely to provide similar performance despite the theoretical effectiveness of algorithm portfolios and lots of research into algorithm selection. Comparing new approaches to the majority predictor would provide a better picture of the real performance improvement over using a single algorithm, something not commonly found in the literature.

Based on a statistical simulation with bootstrapping, we give recommendations as to which algorithms are likely to have good performance. We identify *support vector machines* in the various forms used in this paper as a particularly promising type of machine learning algorithms. They are very likely to achieve performance at least as good as the simple majority predictor.

Acknowledgments

We thank the reviewers for their feedback and Kristian Kersting for pointing out SVM^{struct} to us. This research was supported by EPSRC grant EP/H004092/1. Lars Kotthoff is supported by a SICSA studentship.

References

Borrett, J. E.; Tsang, E. P. K.; and Walsh, N. R. 1996. Adaptive constraint satisfaction: The quickest first principle. In *ECAI*, 160–164.

Burke, E.; Hart, E.; Kendall, G.; Newall, J.; Ross, P.; and Schulenburg, S. 2003. Hyper heuristics: an emerging direction in modern search technology. In *Handbook of Meta-Heuristics*, volume 57. Kluwer.

Carbonell, J.; Etzioni, O.; Gil, Y.; Joseph, R.; Knoblock, C.; Minton, S.; and Veloso, M. 1991. PRODIGY: an integrated architecture for planning and learning. *SIGART Bull.* 2:51–55.

Fink, E. 1998. How to solve it automatically: Selection among Problem-Solving methods. In *ICAPS*, 128–136.

Gent, I.; Jefferson, C.; Kotthoff, L.; Miguel, I.; Moore, N.; Nightingale, P.; and Petrie, K. 2010. Learning when to use lazy learning in constraint solving. In *ECAI*, 873–878.

Getoor, L., and Taskar, B. 2007. *Introduction to Statistical Relational Learning*. MIT Press.

Gomes, C. P., and Selman, B. 2001. Algorithm portfolios. *Artif. Intell.* 126(1-2):43–62.

Guerri, A., and Milano, M. 2004. Learning techniques for automatic algorithm portfolio selection. In *ECAI*, 475–479.

Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; and Witten, I. 2009. The WEKA data mining software: An update. *SIGKDD Explorations* 11(1).

Joachims, T. 2006. Training linear SVMs in linear time. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 217–226.

Kohavi, R. 1995. A study of Cross-Validation and bootstrap for accuracy estimation and model selection. In *IJCAI*, 1137–1143.

Kotthoff, L.; Miguel, I.; and Nightingale, P. 2010. Ensemble classification for constraint solver configuration. In *CP*, 321–329.

Lagoudakis, M. G., and Littman, M. L. 2000. Algorithm selection using reinforcement learning. In *ICML*, 511–518.

Leyton-Brown, K.; Nudelman, E.; Andrew, G.; McFadden, J.; and Shoham, Y. 2003. A portfolio approach to algorithm selection. In *IJCAI*, 1542.

Lobjois, L., and Lemâitre, M. 1998. Branch and bound algorithm selection by performance prediction. In *AAAI*, 353–358.

O'Mahony, E.; Hebrard, E.; Holland, A.; Nugent, C.; and O'Sullivan, B. 2008. Using case-based reasoning in an algorithm portfolio for constraint solving. In *Irish Conference on Artificial Intelligence and Cognitive Science*.

Pulina, L., and Tacchella, A. 2009. A self-adaptive multi-engine solver for quantified boolean formulas. *Constraints* 14(1):80–116.

Rice, J. R. 1976. The algorithm selection problem. *Advances in Computers* 15:65–118.

Riesbeck, C. K., and Schank, R. C. 1989. *Inside Case-Based Reasoning*. L. Erlbaum Associates Inc.

Smith-Miles, K. A. 2009. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Comput. Surv.* 41:6:16:25.

Streeter, M. J., and Smith, S. F. 2008. New techniques for algorithm portfolio design. In *UAI*, 519–527.

Weerawarana, S.; Houstis, E. N.; Rice, J. R.; Joshi, A.; and Houstis, C. E. 1996. PYTHIA: a knowledge-based system to select scientific algorithms. *ACM Trans. Math. Softw.* 22(4):447–468.

Wolpert, D. H., and Macready, W. G. 1997. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* 1(1):6782.

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2008. SATzilla: portfolio-based algorithm selection for SAT. *J. Artif. Intell. Res. (JAIR)* 32:565–606.