Hybrid Regression-Classification Models for Algorithm Selection

Lars Kotthoff[1]

Abstract. Many state of the art Algorithm Selection systems use Machine Learning to either predict the run time or a similar performance measure of each of a set of algorithms and choose the algorithm with the best predicted performance or predict the best algorithm directly. We present a technique based on the well-established Machine Learning technique of stacking that combines the two approaches into a new hybrid approach and predicts the best algorithm based on predicted run times. We demonstrate significant performance improvements of up to a factor of six compared to the previous state of the art. Our approach is widely applicable and does not place any restrictions on the performance measure used, the way to predict it or the Machine Learning used to predict the best algorithm. We investigate different ways of deriving new Machine Learning features from the predicted performance measures and evaluate their effectiveness in increasing performance further. We use five different regression algorithms for performance prediction on five data sets from the literature and present strong empirical evidence that shows the effectiveness of our approach.

1 Introduction

The Algorithm Selection Problem [17] is to select the most suitable algorithm for solving a particular problem. Especially with the rise of algorithm portfolios [10], an increasing amount of research has been devoted to finding better ways of identifying suitable algorithms in practice. An algorithm portfolio usually consists of a selection of state of the art algorithms for solving a particular kind of problem. Portfolio solvers such as SATzilla [23] have achieved impressive performance improvements over individual state of the art solvers.

A crucial part of such portfolio solvers is the system that chooses which algorithm from the portfolio to use for solving a given problem. The relationship between an algorithm, a problem and the performance of the algorithm on the problem cannot be quantified easily such that the performance on unseen problems can be predicted reliably in all but the most trivial cases. Contemporary systems almost always use some kind of Machine Learning to build performance models of algorithms or algorithm portfolios and predict which algorithm to use on new problems.

There are a lot of different Machine Learning approaches that can be applied to solve the Algorithm Selection Problem. The two most common ones are as follows. A Machine Learning classifier can be trained to directly predict the portfolio algorithm with the best performance on an unseen problem. This is used for example in [8, 15, 16]. Alternatively, Machine Learning regression can be used to predict the performance, usually the run time, of each of the algorithms in the portfolio on the problem. The algorithm with the best predicted performance is chosen. This indirect way of determining the best portfolio algorithm is used for example in [18, 19, 23].

Approaches that predict the performance of each algorithm use the predicted performance at face value. While individual predictions are often far off the actual performance, the best algorithm may still be identified as long as the order of the algorithms with regards to their performance can be inferred correctly. While this approach works well in practice, there remains significant room for improvements.

We propose to take the predicted performance of individual portfolio algorithms not at face value, but rather treat them as inputs to another Machine Learning step that ultimately chooses the algorithm to be used to solve the problem. We treat all predictions of the first step equally and do not attempt to identify the best Machine Learning technique for the problem. This hybrid approach combines both regression and classification learners. The approach is applicable in general and does not rely on any particular kind of performance measure or Machine Learning to predict it. We empirically demonstrate significant performance improvements compared to always choosing the algorithm with the best predicted performance measure.

2 Background

A common approach to solving the Algorithm Selection Problem with algorithm portfolios is to predict the performance of each algorithm in the portfolio on the problem to tackle. During a training phase, the performance of each of the algorithms is observed on a set of problems. This information, along with features characterising the problems that were solved, is used to build a performance model for each algorithm. On new problems, the model uses the same features to make a performance prediction.

There are variations of the predicted performance measure; some approaches predict the run time [12, 19, 21], others the log of the run time [23] or a compound measure that accounts for possible timeouts within a set time limit [24]. In some publications, the performance is estimated by means of a proxy measure. Allen and Minton [1] for example predict the number of constraint checks required to solve a constraint problem and Lobjois and Lemaître [14] predict the number of search nodes to explore and the time per node. The predictions are used to make the selection which portfolio algorithm to choose.

The common trait of all these approaches is that, in order to combine the performance predictions for the individual algorithms, the min function is used. That is, the algorithm with the predicted lowest run time or the predicted best performance measure is chosen for solving the problem. The advantage of using this simple way of combining the predictions, apart from its computational simplicity, is that the inferred algorithm will be correct as long its predicted performance is the better than the other predicted performances. The actual value of the prediction does not matter, as it is only compared to

[1] University of St Andrews, larsko@cs.st-andrews.ac.uk

the other predictions. Similarly, the ordering of the algorithms in the portfolio according to their performance does not need to be totally correct as long as the best algorithm can be identified correctly.

While recent approaches have experimented with predicting the complete ordering of the algorithms [13], they report that the performance is not competitive with other approaches, attesting to the difficulty of such a prediction. However, orderings of classification algorithms according to their accuracy have been predicted in the Machine Learning community [3, 20].

A related approach is to predict schedules according to which to run the algorithms in the portfolio [7, 15]. Predicting schedules also relies on the fact that the performance ordering of the algorithms is correct. It is more robust with respect to prediction errors though, as more than one algorithm is run. Other research has tried to predict when to switch the algorithm being used to solve a problem [2, 4], which also reduces the negative impact of an initial bad choice.

The idea of using the predictions of one Machine Learning algorithm as features for another Machine Learning algorithm is known as stacked generalisation, or *stacking* [22]. The rationale of stacking is that the imperfect predictions of one model can be improved by another model that learns to correct the errors of the first model. The Machine Learning algorithms that are stacked on top of each other learn models that do not reflect the original problem, but try to identify and correct the prediction mistakes earlier models make. The technique is a member of the family of *ensemble techniques* [5] and closely related to *boosting* [6], where Machine Learning models with low performance are engineered to complement each other.

Stacking is a well-established technique in the Machine Learning community. However, to the best of our knowledge, stacking techniques have never been applied to Algorithm Selection before.

3 Methodology

We measure the performance of an Algorithm Selection system in terms of misclassification penalty. The misclassification penalty is the additional time a system needs to solve a set of problems because of wrong choices. If the algorithm with the best performance is chosen for each respective problem, the misclassification penalty is 0. We give the misclassification penalty in seconds. When a timeout occurs, we assume the timeout as the run time to calculate the misclassification penalty. This only gives a lower bound on the actual misclassification penalty, but we cannot determine the correct value without running the algorithm to completion.

We use the WEKA Machine Learning software [11] and the algorithms implemented in it. The evaluation includes the regression algorithms `GaussianProcesses`, `LibSVM` (ε and ν), `LinearRegression` and `REPTree`. These regression algorithms represent a variety of different state of the art Machine Learning approaches. The classification algorithm `SimpleLogistic` is used to combine the predictions of the regression models and choose the best portfolio algorithm. The maximum misclassification penalty for a particular problem instance is used as a weight during training to bias the learned logistic regression model towards the instances where we can gain or lose a lot [9]. The maximum misclassification penalty is incurred when the worst algorithm for a problem is chosen.

The performance of a Machine Learning algorithm is evaluated using ten fold stratified cross validation. A data set is partitioned into ten distinct subsets of roughly equal size. Nine of these sets serve as training data and the tenth as test data. This is repeated for all different combinations of training and test sets. In the end, every problem instance in the data set will have been used for both training and testing. Stratification ensures that the distribution of the class labels to predict is the same in each subset as it is in the full data set. We calculate the sum of the misclassification penalties across the different folds to estimate the penalty on the whole data set.

We tuned all Machine Learning algorithms by evaluating their performance across a number of different parameter configurations. The default values used by WEKA achieved the best performance and therefore we decided to use the defaults. It is conceivable that the performance could be improved further by evaluating a larger space of parameter configurations.

We evaluate the performance on five data sets taken from the literature. We take three sets from the training data for SATzilla 2009. This data consists of SAT instances from three categories – hand-crafted, industrial and random. They contain 1181, 1183 and 2308 instances and are denoted SAT-HAN, SAT-IND and SAT-RAN, respectively. The SATzilla authors use 91 attributes for each instance and select a SAT solver from a portfolio of 19 solvers[2]. We adjusted the timeout values reported in the training data available on the website to 3600 seconds after consultation with the SATzilla team as some of the reported timeout values are incorrect.

The fourth data set comes from the Quantified Boolean Formulae (QBF) Solver Evaluation 2010[3] and consists of 1368 QBF instances from the main, small hard, 2QBF and random tracks. It is denoted QBF. 46 attributes are calculated for each instance and we select from a portfolio of five 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 four core Intel E5430 2.66 GHz processor and 16 GB RAM. Our last data set, denoted CSP, is taken from from [8] and selects from a portfolio of two solvers for a total of 2028 constraint problem instances from 46 problem classes with 17 attributes each.

4 Motivation

The success of regression methods that predict the run time in Algorithm Selection suggests that most of the time, the predicted performance values allow to derive an ordering that puts the actual best portfolio algorithm at the corresponding position. In order to assess the potential improvement a hybrid approach could achieve, we measured the percentage of the problem instances of each data set where this is the case. We used ten fold stratified cross validation, as described above. The results are shown in Figure 1.

The results clearly show that there is significant scope for improvement, especially with portfolios that contain a relatively large number of algorithms. Choosing a portfolio algorithm at random would put the percentage of problems for which the best algorithm is not chosen at 50% for the CSP data set, 80% for QBF and 95% for the SAT data sets. Using the *min* function unsurprisingly improves significantly on that, but the fraction of problem instances for which the predicted lowest run time does not denote the best solver is still very high in some cases. Running `LibSVM` ν on the SAT-RAN data set and predicting the run time identifies the best algorithm correctly in only 12% of the cases for example.

Using the predictions as input to a classifier that learns to account for their errors has the potential to increase performance significantly. The idea of learning models that correct the errors of other models has been used with great success in the form of boosting in the Machine Learning community.

[2] `http://www.cs.ubc.ca/labs/beta/Projects/SATzilla/`
[3] `http://www.qbflib.org/index_eval.php`

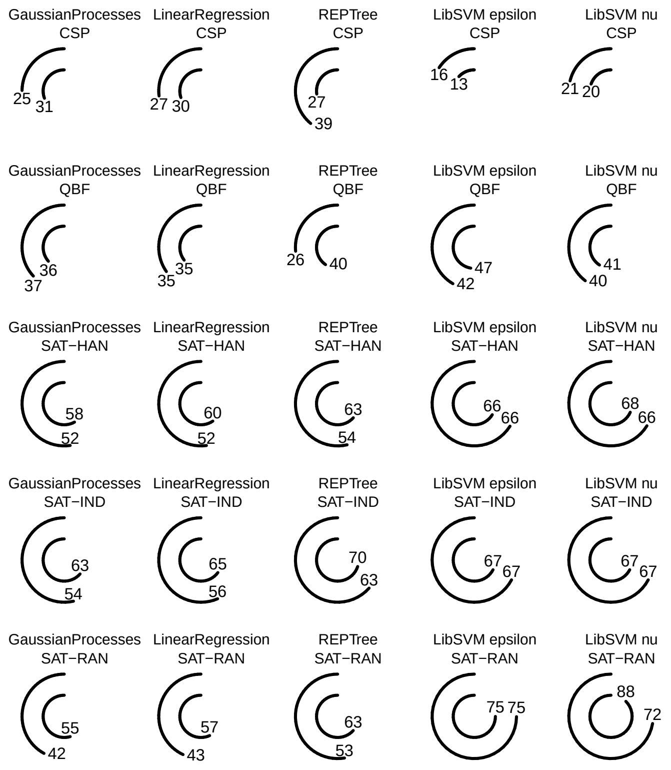

Figure 1. Percentage of problem instances on which the actual best portfolio algorithm is *not* chosen by the min function for all data sets and regression algorithms. The inner arc denotes the percentage for the predicted run time and the outer arc the percentage for the predicted log of the run time. The angle covered by the arc denotes the percentage, which is also shown as a number. A full circle means that the best portfolio algorithm is chosen for none of the problem instances, no arc means that the best algorithm was chosen in all cases.

5 Stacking classification on regression

The current state of the art is to select the algorithm with the lowest predicted run time, i.e. combining the predictions of the regression models using the min function.

$$a_{best} = \min(\{p(a) \mid a \in \mathcal{A}\})$$

where $\mathcal{A}$ is the set of algorithms in the portfolio and $p(a)$ the predicted run time of an algorithm. This relies on the predictions being accurate at least relative to each other in order for the best algorithm to be identified.

The idea behind our approach is simple. Instead of using a constant function such as min to combine the outputs of run time predictions for the algorithms in the portfolio, we use Machine Learning to induce a classifier $\mathcal{C}$ that learns to select an algorithm with the help of the predicted run times.

$$a_{best} = \mathcal{C}(\{p(a) \mid a \in \mathcal{A}\})$$

This approach allows for more flexibility in combining the individual predictions and, crucially, replaces a hand-crafted rule with a mechanism that automatically adapts the model used to make decisions to the data that it is processing. This notion is at the very core of Algorithm Selection and the source of significant performance improvements over the previous state of the art in many areas. Even if the predicted performance of the algorithms in the portfolio was the opposite of the desired outcome, i.e. the best algorithm has the worst predicted performance, our approach would be able to cope with this, whereas the min function would fail.

Our hybrid approach is agnostic to the way the predicted run times are obtained and can be combined with any of the approaches to do so from the literature. It does not require predictions of the run time either, any of the performance measures researchers have predicted to perform Algorithm Selection can be used.

We investigate the following different ways of using the predicted run times as features to induce a classifier $\mathcal{C}$. First, we use only the predicted run times themselves.

$$\mathcal{F} = \{p(a) \mid a \in \mathcal{A}\}$$

Similarly, we use the predicted logs of the run times.

$$\mathcal{F}^{log} = \{p_{log}(a) \mid a \in \mathcal{A}\}$$

where $p_{log}(a)$ is the predicted log of the run time of algorithm a on a particular problem. Second, we compute new feature sets $\mathcal{F}_{rel}$ and $\mathcal{F}_{diff}$ with features based on the differences between pairs of predicted run times or predicted logs of run times.

$$\mathcal{F}_{rel} = \{\sigma(f, g) \mid f, g \in \mathcal{F}, f \neq g\}$$

where

$$\sigma(f, g) = \begin{cases} -1 & \text{if } f < g \\ 0 & \text{if } f = g \\ 1 & \text{otherwise} \end{cases}$$

and

$$\mathcal{F}_{diff} = \{f - g \mid f, g \in \mathcal{F}, f \neq g\}$$

In the first case, we only take the *qualitative* difference between two feature values into account, i.e. whether the first value is less than, equal to or greater than the second value. In the second case, we *quantify* the difference. New feature sets $\mathcal{F}^{log}_{rel}$ and $\mathcal{F}^{log}_{diff}$ based on the predicted log of the run time are created the same way.

All new feature sets are used on their own and in combination with the original feature sets $\mathcal{F}$ and $\mathcal{F}_{log}$, respectively for the differences based on the predicted run time and the predicted log of the run time. The implementation of the computation of the new features considers each pair of predicted performance measures only once, e.g. for predicted run times f and g where $f \neq g$, we only consider $\sigma(f, g)$ and $f - g$, not $\sigma(g, f)$ or $g - f$.

In total, we consider ten different feature sets. First, the two feature sets that contain only the predictions of the run time and the log of the run time $\mathcal{F}$ and $\mathcal{F}^{log}$. Second, four computed feature sets based on the qualitative and quantitative differences between pairs of feature values for both the predictions of the run time and the log of the run time $\mathcal{F}_{rel}$, $\mathcal{F}_{diff}$, $\mathcal{F}^{log}_{rel}$ and $\mathcal{F}^{log}_{diff}$. Third, we augment the original feature sets with the appropriate new feature sets for the final four feature sets $\mathcal{F} \cup \mathcal{F}_{rel}$, $\mathcal{F} \cup \mathcal{F}_{diff}$, $\mathcal{F}^{log} \cup \mathcal{F}^{log}_{rel}$ and $\mathcal{F}^{log} \cup \mathcal{F}^{log}_{diff}$.

Using the pairwise differences between the predicted run times was motivated by the fact that the previous state of the art is to always pick the minimum predicted time. This min function is difficult to learn if only the times are available as features because Machine Learning does not usually consider the relation between feature values, but only the feature values themselves. In order to facilitate this comparison, which is necessary to learn a min function, the differences between pairs of predicted run times were added as additional features.

6 Experimental results

The results are presented in terms of the previous state of the art, the selection of the best algorithm according to the lowest predicted run time. The performance of the different feature sets we evaluated is shown in terms of that baseline. Compared to predicting the best algorithm directly with a classifier, our approach offers similar performance in most cases and significant improvements in a few cases. Our approach also compares favourably with the regression model used in SATzilla, improving on its performance by a factor of 3 in one case. Note that SATzilla's performance in practice is also dependent on the other techniques used in the system however.

Previous research is unclear as to whether predicting the run time itself or the logarithm thereof gives better overall performance. Xu et al. [23] report that they found a log transformation useful in practice, while Kotthoff et al. [13] found no significant difference. Figure 2 clearly shows that the performance of our hybrid approach is significantly better with predictions of the log of the runtime. Not only the maximum improvement is higher, but the median, i.e. expected, performance improves on the previous state of the art.

Even when applying classification in a straightforward fashion and using only the predicted log of the run times as features (feature set $\mathcal{F}^{log}$), we already beat the min function in terms of expected (median) performance. The very high improvement in one case makes the average mean improvement even more significant than that of the median. Deriving new features from the predictions of the regression algorithms improves the performance somewhat, but not significantly. The main difference is that the loss of performance compared to the min function in cases where this occurs is less severe.

Figure 2 also shows that using the quantitative difference between pairs of predicted run times as features achieves significantly better performance than using the qualitative difference. Both on their own and in conjunction with the actual predicted performance values, the performance achieved with the quantitative features beats the performance of the qualitative features.

The performance improvements achieved with the feature sets $\mathcal{F}^{log}$, $\mathcal{F}^{log}_{diff}$ and $\mathcal{F}^{log} \cup \mathcal{F}^{log}_{diff}$ are almost identical. The logistic regression classifier is able to build good models both with the predicted log of the run time and the pairwise differences between the predictions. This suggests that the differences carry as much information as the log of the run times when it comes to deciding which algorithm has the best performance on a problem. The actual value of the predicted performance measure is not relevant to the choice of the best algorithm, but the order of the algorithms according to the predicted performance has to be correct.

Figure 3 shows that the hybrid approach improves the percentage of problem instances where the best portfolio algorithm is not identified correctly in most cases (17 out of 25). This reflects the performance improvements in terms of misclassification penalty for this feature set as shown in Figure 2. The difference is shown for one of the feature sets that achieve the highest performance improvements. The graphs for the other feature sets similarly reflect the performance improvements in Figure 2 and are omitted due to space constraints.

6.1 Performance improvements

There is no feature set where our method of training and running a Machine Learning classifier to combine the predicted run times of the algorithms in the algorithm portfolio improves over using the min function to choose the best algorithm for all data sets and regression algorithms. Regression algorithm – data set pairs that do not achieve

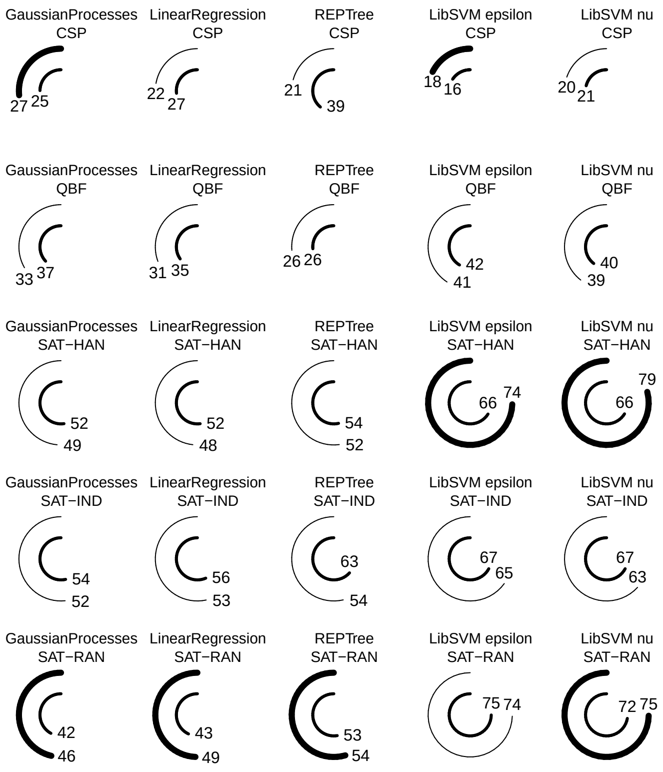

Figure 3. Percentage of problem instances on which the actual best portfolio algorithm is *not* chosen for all data sets and regression algorithms. The inner arc denotes the percentage for the min function on predictions of the log of the run time (the outer arc in Figure 1) and the outer arc that for the hybrid approach with feature set $\mathcal{F}^{log} \cup \mathcal{F}^{log}_{diff}$. If the fraction of the hybrid approach is smaller than that of the min function (the angle covered is smaller), the outer arc is drawn thinner than the inner arc, else thicker.

an improvement with one feature set do achieve it with other feature sets though. For the $\mathcal{F}^{log}_{diff}$ feature set, the performance is worse than that of the min function in only 6 out of 25 cases. For the $\mathcal{F}^{log} \cup \mathcal{F}^{log}_{diff}$ feature set, this is the case for eight regression algorithm – data set pairs. Note that these are not the same cases as those in Figure 3 that do not improve on the percentage of misclassifications though. For `REPTree` and `LinearRegression` on the SAT-IND data set, our approach is *always* better than the min function regardless of the feature set and for `LibSVM` ν on the QBF data set it is never worse.

The improvement in terms of the fraction of problem instances for which the best portfolio algorithm is not identified (cf. Figure 3) is in general a good indicator of the performance improvement in terms of misclassification penalty. There are also a few cases though when the best portfolio algorithm is identified more often but the misclassification penalty increases and vice versa, as mentioned above. However, when this occurs, the differences are small, e.g. a small improvement of the percentage of misclassifications might see a small drop in performance in terms of misclassification penalty.

There is no clear pattern that offers an explanation for the behaviour of the hybrid approach when it does not improve performance. In some cases, the behaviour occurs when the min function achieves relatively bad performance. This could indicate a *floor effect*, i.e. that any further improvements in performance are too difficult to achieve given the available information. On the other hand, the behaviour also occurs when the min function achieves very good

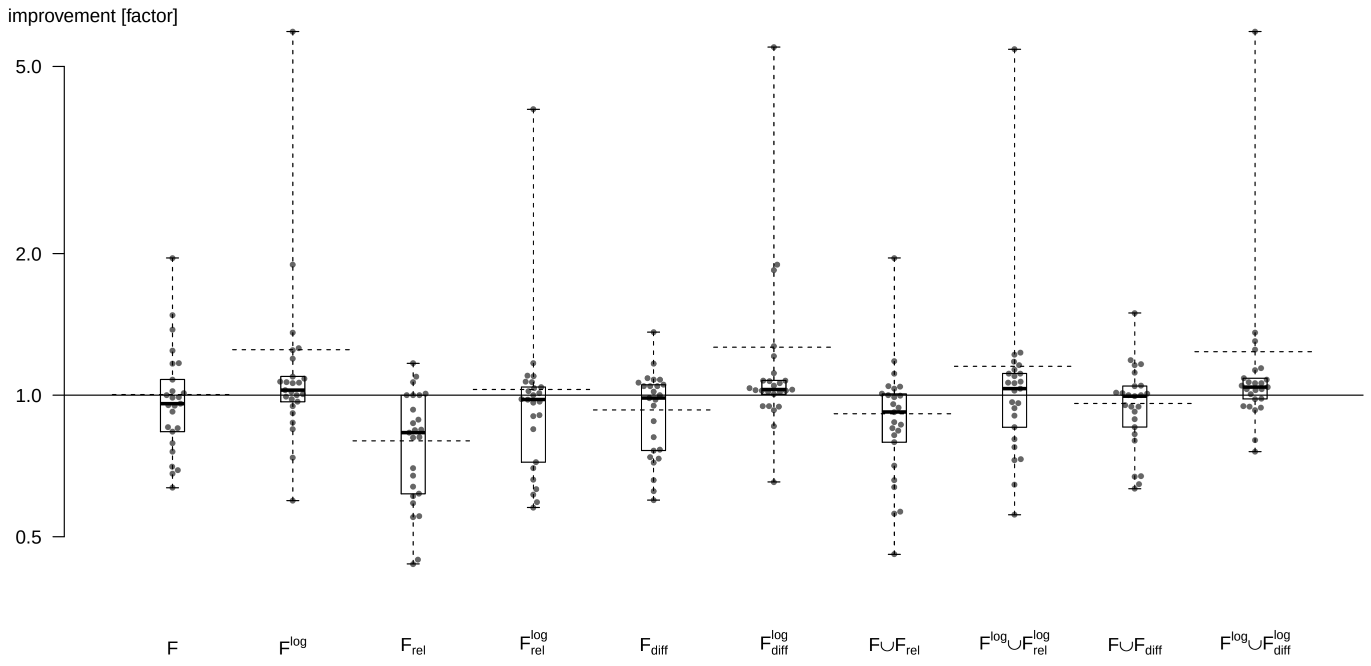

Figure 2. Performance improvements across all data sets and regression algorithms for different feature sets. Factors greater than 1 mean that the performance of the hybrid approach is better than simply using a min function to combine the predictions. The lower and upper edges of the boxes show the 25th and 75th percentile, respectively, and the thick line inside the boxes the median. The horizontal dashed lines show the mean performance improvement. The grey dots denote the performance improvement for individual Machine Learning algorithm – data set combinations.

performance already. This could be an indication of a *ceiling effect*, i.e. the original features used for the regression models enable good performance predictions that reflect the true order of the performances of the portfolio algorithms. The investigation of the causes of this behaviour and ways of mitigating it is an area of future work.

6.2 Other classifiers

In addition to the logistic regression model, we also evaluated the performance improvements of the hybrid approach with a C4.5 decision tree inducer (`J48` in WEKA) using the same feature sets and the same methodology. While the performance was not as good as that of the logistic regression classifier, we found the same general pattern as shown in Figure 2. Predicting the log of the run time instead of the run time itself and using the quantitative difference between predicted values instead of the qualitative one improves performance.

This result demonstrates that the hybrid approach is not limited to our chosen classifier. The performance improvements over using the min function will depend on that choice in practice however. If the Machine Learning algorithm is unable to build a model that reflects the relationship between the predicted performance measures and the best portfolio algorithm, it will not be able to achieve any improvements.

The concept of stacking a classifier on top of the outputs of regression models is not limited to a particular Machine Learning algorithm. In principle, any learner that is able to distinguish between multiple classes can be used. Our choice of a logistic regression classifier was motivated by the conceptual simplicity of the algorithm and its ability to learn a model akin to a min function, given the pairwise differences between predicted run times as features. Observations in the paper that introduced stacking [22] also suggest that a logistic regression approach would be suitable in this context.

6.3 Overhead

The hybrid approach incurs a certain overhead because it builds and evaluates a model on top of the run time predictions for each algorithm. The relative amount of the overhead depends on the size of the portfolio – training and running regression models for two different algorithms is much faster than doing the same for 19 algorithms. The bulk of this overhead is incurred during the offline training phase when the Machine Learning models are built though. The time taken for this phase is usually of less interest because the aim is to achieve good performance online, i.e. on new problems. In practice, the cost of running a regression model or classifier on a problem is usually dwarfed by the cost of computing the problem features. Many publications recognise this problem and take explicit steps to mitigate it, by for example running a presolver [23] or explicitly excluding features that are expensive to compute [8].

The overhead of the hybrid approach can be compared to adding one more algorithm to the portfolio and having to train an additional model for its performance. In our experiments, the time required for training the logistic regression model was approximately the same as the time required to train a performance regression model for one of the portfolio algorithms. The overhead we observed empirically corresponded to this and reflected the number of algorithms in the respective portfolios.

Over all data sets and Machine Learning algorithms in our evaluation, the mean average overhead of training the logistic regression classifier was 6% of the training time of the regression models to predict the run time. We measured the overhead for the feature set $\mathcal{F}^{log} \cup \mathcal{F}^{log}_{diff}$, but in terms of the total time required for a set of experiments, all feature sets were similar. The mean average performance improvements achieved with the $\mathcal{F}^{log} \cup \mathcal{F}^{log}_{diff}$ feature set was 24%. The highest relative overhead was incurred on the CSP data set, where the portfolio is only of size two, but this is also the data set where we achieve the highest performance improvements of up to a factor of six.

7 Conclusions and future work

In this paper, we have introduced a hybrid regression-classification approach for Algorithm Selection. The predictions of regression models for the performance of algorithms in a portfolio are used as features for a classifier that selects the best algorithm from the portfolio for a specific problem to solve. The idea was inspired by the Machine Learning technique stacking, where the predictions of a Machine Learning algorithm are used as features for another Machine Learning algorithm. We evaluated the performance of a logistic regression classifier on different data from the Algorithm Selection literature with different feature sets based on the predicted performance of a set of algorithms in a portfolio.

We have demonstrated that our hybrid approach has the potential to improve the performance of current Algorithm Selection systems significantly and realises it in most cases. We gave empirical evidence that shows that selecting an algorithm based only on the best predicted run time is wrong a large percentage of the time and showed that a classifier improves this percentage in most cases. Through careful engineering of the features derived from the predictions of run time performance and a log transformation of the run times, we achieved improvements over a straightforward stacking of Machine Learning algorithms.

For decades, researchers have used the simple approach of selecting the best algorithm directly according to the predicted run time or a similar performance measure. We have, for the first time, gone beyond the assumption that the minimum run time denotes the best algorithm and instead used Machine Learning to correct eventual errors the performance models made. This represents a significant step in applying sophisticated Machine Learning techniques to the Algorithm Selection Problem.

Our approach does not depend on a particular performance measure or way of predicting this performance measure. It is applicable to a wide variety of approaches and can be used in conjunction with virtually any of the systems that predict the behaviour of individual portfolio algorithms in the literature. It furthermore does not assume a specific classifier for selecting the best portfolio algorithm. We have demonstrated the effectiveness of the hybrid approach with the most widely used performance measures and a classifier that achieves good performance improvements, but we are confident that similar performance improvements can be achieved with other performance measures and Machine Learning algorithms.

Based on our experiments, we recommend the feature set $\mathcal{F}^{log}_{diff}$, either on its own or in conjunction with $\mathcal{F}^{log}$, to be used for an approach that builds a classifier on top of the outputs of regression models for individual portfolio algorithms to decide which algorithm to choose for a particular problem.

Our method improves over an approach that only predicts the expected performance and chooses the algorithm with the best prediction for most regression method – data set combinations that we have investigated. The best observed improvement is a factor of six over the previous state of the art. It also compares favourably with using a classifier to directly predict the best algorithm from problem features.

In the future, we would like to continue to develop the hybrid approach and in particular examine why it decreases the performance in some cases. With the identification of the factors affecting this behaviour and ways of mitigating it, the hybrid approach could be recommended unreservedly over using the simple min function.

ACKNOWLEDGEMENTS

This work was supported by an EPSRC Doctoral Prize and a SICSA studentship. We thank Ian Gent and the reviewers for feedback.

REFERENCES

[1] John A. Allen and Steven Minton, ‘Selecting the right heuristic algorithm: Runtime performance predictors’, in *Canadian Society for Computational Studies of Intelligence*, pp. 41–53, (1996).

[2] James E. Borrett, Edward P. K. Tsang, and Natasha R. Walsh, ‘Adaptive constraint satisfaction: The quickest first principle’, in *ECAI*, pp. 160–164, (1996).

[3] Pavel Brazdil and Carlos Soares, ‘A comparison of ranking methods for classification algorithm selection’, in *ECML*, pp. 63–74, (2000).

[4] Tom Carchrae and J. Christopher Beck, ‘Low-Knowledge algorithm control’, in *AAAI*, pp. 49–54, (2004).

[5] Thomas G. Dietterich, ‘Ensemble methods in machine learning’, in *International Workshop on Multiple Classifier Systems*, pp. 1–15, (2000).

[6] Yoav Freund and Robert E. Schapire, ‘A decision-theoretic generalization of on-line learning and an application to boosting’, in *EuroCOLT*, pp. 23–37, (1995).

[7] Matteo Gagliolo, Viktor Zhumatiy, and Jürgen Schmidhuber, ‘Adaptive online time allocation to search algorithms’, in *ECML*, pp. 134–143, (2004).

[8] Ian P. Gent, Chris Jefferson, Lars Kotthoff, Ian Miguel, Neil Moore, Peter Nightingale, and Karen Petrie, ‘Learning when to use lazy learning in constraint solving’, in *ECAI*, pp. 873–878, (August 2010).

[9] Ian P. Gent, Lars Kotthoff, Ian Miguel, and Peter Nightingale, ‘Machine learning for constraint solver design a case study for the alldifferent constraint’, in *TRICS*, pp. 13–25, (2010).

[10] Carla P. Gomes and Bart Selman, ‘Algorithm portfolios’, *Artif. Intell.*, **126**(1-2), 43–62, (2001).

[11] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten, ‘The WEKA data mining software: An update’, *SIGKDD Explor. Newsl.*, **11**(1), 10–18, (November 2009).

[12] Eric Horvitz, Yongshao Ruan, Carla P. Gomes, Henry A. Kautz, Bart Selman, and David M. Chickering, ‘A bayesian approach to tackling hard computational problems’, in *UAI*, pp. 235–244, (2001).

[13] Lars Kotthoff, Ian P. Gent, and Ian Miguel, ‘A preliminary evaluation of machine learning in algorithm selection for search problems’, in *SoCS*, pp. 84–91, (July 2011).

[14] Lionel Lobjois and Michel Lemaître, ‘Branch and bound algorithm selection by performance prediction’, in *AAAI*, pp. 353–358, (1998).

[15] Eoin O’Mahony, Emmanuel Hebrard, Alan Holland, Conor Nugent, and Barry O’Sullivan, ‘Using case-based reasoning in an algorithm portfolio for constraint solving’, in *Irish Conference on AI and Cognitive Science*, (2008).

[16] Luca Pulina and Armando Tacchella, ‘A multi-engine solver for quantified boolean formulas’, in *CP*, pp. 574–589, (2007).

[17] John R. Rice, ‘The algorithm selection problem’, *Advances in Computers*, **15**, 65–118, (1976).

[18] Mark Roberts and Adele E. Howe, ‘Learned models of performance for many planners’, in *ICAPS Workshop AI Planning and Learning*, (2007).

[19] Bryan Silverthorn and Risto Miikkulainen, ‘Latent class models for algorithm portfolio methods’, in *AAAI*, (2010).

[20] Carlos Soares, Pavel B. Brazdil, and Petr Kuba, ‘A Meta-Learning method to select the kernel width in support vector regression’, *Mach. Learn.*, **54**(3), 195–209, (March 2004).

[21] Sanjiva Weerawarana, Elias N. Houstis, John R. Rice, Anupam Joshi, and Catherine E. Houstis, ‘PYTHIA: a knowledge-based system to select scientific algorithms’, *ACM Trans. Math. Softw.*, **22**(4), 447–468, (1996).

[22] David H. Wolpert, ‘Stacked generalization’, *Neural Networks*, **5**, 241–259, (1992).

[23] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown, ‘SATzilla: portfolio-based algorithm selection for SAT’, *J. Artif. Intell. Res.*, **32**, 565–606, (2008).

[24] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown, ‘SATzilla2009: an automatic algorithm portfolio for SAT’, in *SAT Competition*, (2009).