Latent Features for Algorithm Selection

Yuri Malitsky and **Barry O'Sullivan**
Insight Centre for Data Analytics
Department of Computer Science, University College Cork, Ireland
(yuri.malitsky, barry.osullivan)@insight-centre.org

Abstract

The success and power of algorithm selection techniques has been empirically demonstrated on numerous occasions, most noticeably in the competition settings like those for SAT, CSP, MaxSAT, QBF, etc. Yet while there is now a plethora of competing approaches, all of them are dependent on the quality of a set of structural features they use to distinguish amongst the instances. Over the years, each domain has defined and refined its own set of features, yet at their core they are mostly a collection of everything that was considered useful in the past. As an alternative to this shotgun generation of features, this paper instead proposes a more systematic approach. Specifically, the paper shows how latent features gathered from matrix decomposition are enough for a linear model to achieve a level of performance comparable to a perfect Oracle portfolio. This information can, in turn, help guide researchers to the kinds of structural features they should be looking for, or even just identifying when such features are missing.

Introduction

Over the last decade, algorithm selection has led to some very impressive results in a wide variety of fields, like satisfiability (SAT) (Xu et al. 2012b; Kadioglu et al. 2011), constraint satisfaction (CSP) (O'Mahony et al. 2008), and machine learning (Thornton et al. 2013). The underlying principle guiding this research is that there is no single algorithm that performs optimally on every instance. That whenever a solver is developed to perform well in the general case, it can only do so by sacrificing quality on some instances. Algorithm selection therefore studies each instance's structural properties through a collection of numeric features to try to predict the algorithm that will perform best on it. The ways in which these decisions are made is constantly growing, with only a small number referred to in the following sections, but the results have been dominating international competitions of SAT, CSP, MaxSAT, and others over the last few years. For a general overview of some of the portfolio methodologies, we refer the reader to a recent literature review (Kotthoff, Gent, and Miguel 2012).

To get a better handle on this field, consider the semiannual SAT Competition domain, where in 2012 there were 65 solvers submitted by 27 research groups. If we consider just the top 29 solvers and evaluate each with a 5,000 second timeout on 4,260 instances, we would observe that the best solver can find a solution to 2,697 instances, or 63.3%. Ignoring the runner-up, which was an alternate version of the winner, the next best solver could only solve 1,861 instances (43.7%). However, if we instead introduce an Oracle solver, that would always choose the fastest solver for each instance, an additional 979 instances could be solved, 23% more than the best solver. Furthermore, if we look at the solvers chosen by this oracle in this particular case, we would observe that the best solver is never chosen. This means that while the best solver is good overall, it is never the fastest for any given instance. Closing this gap between the best single solver and the oracle is the objective of algorithm selection.

Yet, while the performance of algorithm selection techniques is continually improving, it does so at the cost of transparency of the employed models. The new version of SATzilla, the winning portfolio in 2012, trains a tree to predict the winner between every pair of solvers (Xu et al. 2012b). CSHC, the 2013 winner, takes an alternate approach of introducing a new splitting criterion for trees that makes sure that each subtree is more consistent on the preferred solver than its root (Malitsky et al. 2013). But in order to make the approach competitive, many of these trees need to be grouped into a forest. Yet other approaches create schedules of solvers to improve the chances of solving each instance (Kadioglu et al. 2011; Helmert, Röger, and Karpas 2011). Unfortunately, even though all these approaches are highly effective at solving instances, once they are trained they are nearly impossible to use to get a better understanding of the fundamental issues of a particular problem domain. In short, we can now answer *what* we should do when we see a new instance, the new question should therefore be *why* a particular solver is chosen and we should use this information to spur the development of a new wave of solvers.

The focus of this paper is therefore to present a new portfolio approach that can achieve similar performance to leading portfolios while also presenting a human interpretable model. Specifically, this is done with the help of matrix decomposition. The paper shows that instead of using the current set of features, which have been generated sporadically and manually by including anything that may be of use, it is necessary to instead turn to latent features generated by studying the matrix detailing the performance of each solver on each instance. It is argued that these latent features capture the differences between solvers by observing their actual performances. It is then shown that a portfolio trained on these features can significantly outperform the best existing approaches. Of course, because in the regular setting these features are not actually available, the paper shows how the currently existing features can be used to approximate these latent features. This has the additional benefit of revealing the absence of some crucial information if the existing features are unable to accurately predict a latent feature. This information can in turn be used to guide future research in getting a better understanding of what it is exactly that causes one solver to outperform another.

The remainder of the paper is broken down into three sections. First, we present an assortment of algorithm selection techniques and apply them to problems in three separate domains. The section therefore demonstrates both the effectiveness and the drawbacks of current techniques. The following section then presents the concept of latent features and how they are computed using singular value decomposition. This section goes into detail about the strengths and applications of the new approach. The paper concludes with a general discussion of the further avenues of research that become accessible using the described technique.

Algorithm Selection

Although the term was first introduced by Rice in 1976 (Rice 1976), it is only recently that the study of algorithm selection has begun to take off. A survey by Kotthoff et.al. touches on many of the recent works (Kotthoff, Gent, and Miguel 2012). Unfortunately, at this time many of the top approaches are either not open source or are very closely linked to a specific dataset or format. This makes an exhaustive comparison with the state of the art impractical. There are currently a number of steps being taken to unify the field, such as the COnfiguration and SElection of ALgorithms (COSEAL) project (COSEAL 2014), but this paper will instead focus on the standard machine learning approaches to algorithm selection. Yet, as was recently shown (Hutter et al. 2014), one of the most effective approaches is to rely on a random forest to predict the best performing solver. This random forest approach is one of the techniques presented here.

This section will first introduce the three problem domains that will be explored in the paper and the features used to analyze them. The section will then proceed to present the results of standard algorithm selection techniques and the results of feature filtering. All presented experiments were run on two X Intel Xeon E5430 processors (2.66 GHz).

Satisfiability

Satisfiability testing (SAT) is one of the premier domains where algorithm portfolios have been shown to excel. Ever since SATzilla (Xu et al. 2008) was first entered in the 2007 SAT competition, a portfolio approach has always placed in each subsequent competition. SAT tackles the problem of assigning a truth value to a collection of boolean variables so as to satisfy a conjunction of clauses, where each clause is a disjunction of the variables. It is a classical NP-complete problem that is widely studied due to its application to formal verification, scheduling, and planning, to name just a few.

Since 2007, the set of features used to describe the SAT instances has been steadily expanding to now number 138 in total. These features are meant to capture the structural properties of an instance, like the number of variables, number of clauses, number of variables per clause, frequency with which two variables appear in the same clause, or two clauses share a variable. These features also include a number of stochastic features gathered after running a few solvers for a short time. These features include the number of propagations, the number of steps to a local optimum, and the number of learned clauses. In the interest of space, we refer the reader to the official paper (Xu et al. 2012a) for a complete list of these features. Yet as expansive as this list is, it was generated by including everything that was thought useful, resulting in many features that are not usually very informative and are usually removed through feature filtering (Kroer and Malitsky 2011).

The dataset investigated in this paper is comprised of 1,098 industrial instances gathered from SAT Competitions dating back to 2006. We focus solely on industrial instances as these are of more general interest in practice. Each instance was evaluated with the top 28 solvers in the 2012 competition with a timeout of 5,000 seconds. These solvers are: clasp.2.1.1_jumpy, clasp2.1.1_trendy, ebminisat, glueminisat, lingeling, lrglshr, picosat, restartsat, circminisat, clasp1, cryptominisat_2011, eagleup, gnoveltyp2, march_rw, mphaseSAT, mphaseSATm, precosat, qutersat, sapperlot, sat4j.2.3.2, sattimep, sparrow, tnm, cryptominisat295, minisatPSM, sattime2011, ccasat, and glucose_21.

MaxSAT

MaxSAT is the optimization version of the SAT problem, aiming to find a variable setting that maximizes the number of satisfied clauses. With applications in bioinformatics, timetabling, scheduling, and many others, the problem domain is of particular interest for research as it can reason about both optimality and feasibility. The particular dataset used in this paper focuses on the regular and partial Weighted MaxSAT (WMS and WPMS respectively) problem. This means that the problem clauses are split and classified as either hard or soft. The objective of a WPMS solver is to satisfy all of the "hard" clauses while also maximizing the cumulative weight of the satisfied "soft" clauses. WMS has a similar objective except that all the "soft" clauses have the same weight.

The investigated dataset is comprised of 1,077 instances gathered from the 2013 MaxSAT Evaluation. Each of the instances is identified by a total of 37 features introduced as part of a recently dominating MaxSAT portfolio (Ansótegui, Malitsky, and Sellmann 2014). These are a subset of the 32 deterministic features used to classify SAT instances. The other 5 features measure the

Table 1: Performance of algorithm selection techniques on SAT, MaxSAT and CSP datasets. The algorithms are compared using the average runtime (AVG), the timeout penalized runtime (PAR10), and the number of instances not solved (NS).

	SAT			MaxSAT			CSP		
	AVG	PAR10	NS	AVG	PAR10	NS	AVG	PAR10	NS
BSS	672	3,620	69	739	6,919	412	1,156	9,851	362
Tree	630	3,114	57	107	709	40	379	2,444	86
Linear	521	2,176	38	104	661	37	321	1,978	69
SVM (radial)	609	3,487	66	542	5,028	299	284	1,533	52
K-NN	463	1,985	35	50.3	292	16	272	1,521	52
Forest-500	384	1,473	25	45.9	226	12	220	940	30
VBS	245	245	0	26.6	26.6	0	131	131	0

percentage of clauses that are soft, and the statistics of the distribution of weights: avg, min, max, stdev. Each of the instances is evaluated with 15 top solvers from 2013 for 1800 seconds. The solvers are: ahmaxsat, akmaxsat_ls, ckmax_small, ILP_2013, maxsatz2013f, MSUnCore, scip_maxsat, shinmaxsat, WMaxSatz+, WMaxSatz09, WPM1_2013, WPM2_2013, WPM2shuc_b, QMaxSAT2_m, pwbo2.33, and six parameterizations of wpm2 found in (Ansótegui, Malitsky, and Sellmann 2014).

Constraint Satisfaction

A constraint satisfaction problem (CSP) is a powerful formulation that can model a large number of practical problems such as planning, routing, and configuration. Such problem instances are represented by a set of variables, each with their own domain of possible assignments. The actual assignments are determined by adherence to a set of constraints. A typical example of such a constraint is the *all-different* constraint which symbolizes that no two variables take the same value.

In this paper we consider the 2,207 instances used to train the Proteus portfolio approach (Hurley et al. 2014). This is a subset of instances from the CSP solver competition[1] containing Graph Coloring, Random, Quasi-random, Black Hole, Quasi-group Completion, Quasi-group with Holes, Langford, Towers of Hanoi and Pigeon Hole problems. Each instance was solved with a 3,600 second timeout by the four top solvers: mistral, gecode, choco, and abscon. A total of 36 features were used for each instance using mistral (Hebrard 2008). The set includes static features like statistics about the types of constraints used, average and maximum domain size; and dynamic statistics recorded by running mistral for 2 seconds: average and standard deviation of variable weights, number of nodes, number of propagations and a few others.

Numerical Evaluation

As a baseline for each of the datasets described above we present the performance of the best single solver (BSS) as well as that of the virtual best performance of an oracle virtual best solver (VBS). We then evaluate a number of machine learning techniques for algorithm selection, each of which will try to predict the runtime of each solver, selecting the expected best. The evaluations are done using LLAMA (Kotthoff 2013), a freely available R package for fast prototyping of algorithm selection techniques. Additionally, the parameters of each of the machine learning approaches were tuned using the R caret (Kuhn 2014) library, which uses grid based search and cross validation to find the best parameters for each technique.

Table 1 presents a comparison of a small subset of algorithm portfolio techniques on the three datasets. Here, all of the algorithms are evaluated using 10-fold cross validation, presenting the average runtime (AVG), the timeout penalized average runtime (PAR10), and the number of not solved instances (NS). Note that for AVG, unsolved tasks are counted with the timeout. For PAR10, if a solver timesout on an instance, the time taken is recorded as 10 times the timeout, otherwise the regular time is recorded. The table compares the performances of training a tree, a linear model, a support vector machine with a radial basis kernel, a k-nearest neighbor, and a random forest consisting of 500 trees. For these particular datasets, the instances where no solver finished within the timeout are removed, hence the 0 unsolved instances for VBS.

Note that in the presented results, all approaches perform better than any single solver, highlighting once again the power of algorithm selection. Note also that among the chosen approaches, the tree and linear models are arguably the most easily interpretable. Yet they are not the best performing portfolios, so it is not clear whether it is safe to make judgments based on their results. After all, it is impossible to tell why they get certain instances wrong. It would be much better if the predictions of the preferred solver were more accurate.

On the other end of the spectrum, k-nearest neighbor and a random forest are much stronger predictors. Forests in particular are now commonly referred to in the literature as the best approach for algorithm selection. Yet as a price for this improved performance, the resulting models are increasingly opaque. Looking at the 500 generated trees, it is impossible to say why one solver is better than another for a particular type of instance or the exact changes among features that highlight this difference. The same is true for k-nearest neighbor. Of course it is possible to look at all the neighboring instances that impact a solver's decision, but this only highlights the potentially similar instances, not

[1]CSP solver competition instances: http://www.cril.univ-artois.fr/~lecoutre/benchmarks.html

Table 2: Performance of algorithm selection techniques after feature filtering on SAT, MaxSAT and CSP datasets. The algorithms are compared using the average runtime (AVG), the timeout penalized runtime (PAR10), and the number of instances not solved (NS).

	SAT			MaxSAT			CSP		
	AVG	PAR10	NS	AVG	PAR10	NS	AVG	PAR10	NS
BSS	672	3,620	69	739	6,919	412	1,156	9,851	362
Tree	632	3,239	60	107	709	40	379	2,444	86
Linear	520	2,219	39	104	661	37	323	2,004	70
SVM (radial)	621	3,412	64	55.7	281	15	282	1,507	51
K-NN	475	2,172	39	48.9	290	16	274	1,523	52
Forest-500	381	1,382	23	47.1	227	12	225	994	32
VBS	245	245	0	26.6	26.6	0	131	131	0

specifically what those differences are.

Feature Filtering

In practice, it is well accepted that dealing with problems with large feature vectors is often ill-advised. One of the reasons for this is that the more numerous the feature set, the more instances are needed to differentiate useful data from noise that happens to look beneficial. After all in a collection of thousands randomly generated features and only one hundred instances, there is a high probability that at least one of the features will correspond with the target value. This issue is mitigated through the use of feature filtering.

There are numerous techniques designed to isolate and remove unhelpful or even adverse features. In this paper we utilize four common filtering techniques made available by the FSelector R library (Romanski 2013): Chi-squared, information gain, gain ratio, and symmetrical uncertainty. Because all these approaches depend on a classification for each instance, we use the name of the best solver for that purpose.

Chi-squared. The Chi-squared test is a correlation-based filter and makes use of "contingency tables". One advantage of this function is that it does not need the discretization of continuous features. It is defined as:

$$\chi^2 = \sum_{ij} (M_{ij} - m_{ij})^2 / m_{ij} \qquad \text{where } m_{ij} = M_i.M_{.j}/m$$

M_{ij} is the number of times instances with target value $Y = y_j$ and feature value $X = x_i$ appear in a dataset, where y are the class and x are features. Here, m denotes the total number of instances.

Information gain. Information gain is based on information theory and is often used in decision trees and is based on the calculation of entropy of the data as a whole and for each class. For this ranking function continuous features must be discretized in advance.

Gain ratio. This function is a modified version of the *information gain* and it takes into account the *mutual information* for giving equal weight to features with many values and features with few values. It is considered to be a stable evaluation.

Symmetrical uncertainty. The symmetrical uncertainty is built on top of the mutual information and entropy measures. It is particularly noted for its low bias for multivalued features.

Each of these filtering techniques assigns a "relevance" score to each of the features, with higher values signifying greater information quality. To select the final subset of features, we order the features based on these scores and use the biggest difference drop as a place to cut the important from the remainder.

In the interest of space, Table 2 only shows the performances of the algorithm selection techniques after the most empirically effective approach: Chi-squared filtering. Applying the filtering technique on the datasets certainly has the desired effect of maintaining (or even improving) performance while reducing the feature set. In the case of SAT, we observe that only 59 of the 138 features are needed. For MaxSAT the reduction is smaller (34 of 37), while for CSP it is 31 of 36. Yet as can be clearly seen from the table, while these filtering techniques can focus us on more informative features, it does not greatly improve the performance of the transparent models, leaving the opaque random forest model the clear winner.

Latent Features

A latent variable is by definition something that is not directly observable but rather inferred from observations. This is a concept that is highly related to that of hidden variables, and is employed in a number of disciplines including economics (Rutz, Bucklin, and Sonnier 2012), medicine (Yang et al. 2012) and machine learning (Ghahramani, Griffiths, and Sollich 2006). This section introduces the idea of collecting latent variables that best describe the changes in the actual performance of solvers on instances. Thereby instead of composing a large set of structural features that might possibly correlate with the performance, this paper proposes a top down approach. Specifically, the paper proposes that matrix factorization like singular value decomposition (SVD) can be used for this purpose.

Singular Value Decomposition

The ideas behind Singular Value Decomposition herald back to the late 1800's when they were independently discovered by five mathematicians: Eugenio Beltrami, Camille Jordan, James Joseph Sylvester, Erhard Schmidt, and Hermann

Table 3: Performance of algorithm selection techniques using the latent features computed after singular value decomposition on SAT, MaxSAT and CSP datasets. The algorithms are compared using the average runtime (AVG), the timeout penalized runtime (PAR10), and the number of instances not solved (NS). We therefore observe that a linear model using the SVD features could potentially perform as well as an oracle.

		SAT			MaxSAT			CSP		
		AVG	PAR10	NS	AVG	PAR10	NS	AVG	PAR10	NS
Standard portfolio	BSS	672	3,620	69	739	6,919	412	1,156	9,851	362
	Forest-500 (orig)	381	1,382	23	47.1	227	12	225	994	32
	VBS	245	245	0	26.6	26.6	0	131	131	0
SVD based portfolio	Tree	508	1,635	26	98.0	563	31	167	287	5
	Linear	245	245	0	26.6	26.6	0	131	131	0
	SVM (radial)	286	373	2	38.8	114	5	134	134	0
	K-NN	331	589	6	34.1	109	5	135	159	1
	Forest-500	300	386	2	32.0	77.0	3	135	231	4

Weyl. In practice, the technique is now currently embraced for tasks like image compression (Prasantha 2007) and data mining (Martin and Porter 2012) by reducing massive systems to manageable problems by eliminating redundant information and retaining data critical to the system.

At its essence, Singular Value Decomposition is a method for identifying and ordering the dimensions along which data points exhibit the most variation, which is mathematically represented by the following equation:

$$M = U\Sigma V^T,$$

where M is the $m \times n$ matrix representing the original data. Here, there are m instances each described by n values. The columns of U are the orthonormal eigenvectors of MM^T, the columns of V are orthonormal eigenvectors of M^TM, and Σ is a diagonal matrix containing the square roots of eigenvalues from U or V in descending order.

Note that if $m > n$ then, being a diagonal matrix, most of the rows in Σ will be zeros. This means that after multiplication, only the first n columns of U are needed. So for all intents and purposes, for $m > n$, U is an $m \times n$ matrix, while both Σ and V^T are $n \times n$.

Because U and V are orthonormal, intuitively one can interpret the columns of these matrices as a linear vector in the problem space that captures most of the variance in the original matrix M. The values in Σ then specify how much of the variance each column captures. The lower the value in Σ, the less important a particular column is. This is where the concept of compression comes into play, when the amount of columns in U and V can be reduced while still capturing most of the variability in the original matrix.

From the perspective of data mining, the columns of the U matrix and the rows of the V matrix have an additional interpretation. Let us assume that our matrix M records the performance of n solvers over m instances. In such a case it is usually safe to assume that $m > n$. So each row of the U matrix still describes each of the original instances in M. But now each column can be interpreted as a latent topic or feature that describes that instance. Meanwhile, each column of the V^T matrix refers to each solver, while each row presents how active, or important a particular topic is for that solver.

These latent features in U give us exactly the information necessary to determine the runtime of each solver. This is because once the three matrices are multiplied out we are able to reconstruct the original performance matrix M. So if we are given a new instance i, if we are able to identify its latent features, we could multiply by the existing Σ and V^T matrices to get back the performance of each solver.

Therefore, if we had the latent features for an instance as computed after the Singular Value Decomposition, it would be possible to train a linear model to accurately predict the performance of every solver. A linear model where we can see exactly which features influence the performance of each solver. This is exactly what Table 3 presents. Like before here, we perform 10-fold cross validation. For each training set we compute matrices U, V and Σ and train each model to use the latent features in U to predict the solver performances. For the test instances, we use the runtimes, P, to compute what the values of U should be by computing $PV\Sigma^{-1}$. These latent features are then used by the trained models to predict the best solver.

Unfortunately, these latent features are only available by decomposing the original performance matrix. This is information that we only have available *after* all the solvers have been run on an instance. Information that once computed means we already know which solver should have been run.

Yet, note that the performance of the models is much better than it was using the original features, especially for the linear model. This is again a completely unfair comparison, but it is not as obvious as it first appears. What we can gather from these results is that the matrix V and Σ are still relevant even when applied to previously unseen instances. This means that the differences between solvers can in fact be differentiated by a linear model, provided it has the correct structural information about the instance. This also means that if we are able to replicate the latent features of a new instance, the supporting matrices computed by the decomposition will be able to establish the performances.

Recall also that the values in Σ are based on the eigenvalues of M. This means that the columns associated with the lower valued entries in Σ encapsulate less of the variance in the data than the higher valued entries. Figure 1 therefore shows the performance of the linear model as more of the

Table 4: Performance of an algorithm selection technique that predicts the latent features of each instance using a random forest on the SAT, MaxSAT and CSP datasets. The algorithms are compared using the average runtime (AVG), the timeout penalized runtime (PAR10), and the number of instances not solved (NS). Note that even using predicted latent features, a linear model of "SVD (predicted)" can achieve the same level of performance as the more powerful, but more opaque, random forest.

	SAT			MaxSAT			CSP		
	AVG	PAR10	NS	AVG	PAR10	NS	AVG	PAR10	NS
BSS	672	3,620	69	739	6,919	412	1,156	9,851	362
Forest-500 (orig)	381	1,382	23	47.1	227	12	225	994	32
VBS	245	245	0	26.6	26.6	0	131	131	0
SVD (predicted)	379	1277	21	49.6	274	15	219	964	31

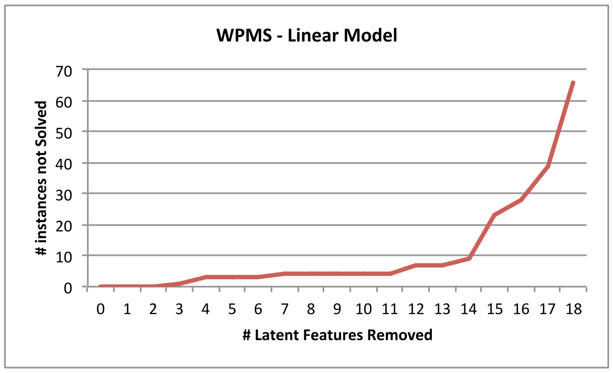

Figure 1: Number of unsolved instances remaining after using a linear model trained on the latent features after singular value decomposition. The features were removed with those with lowest eigenvalues first. The data is collected on the WPMS dataset. This means that even removing over half the latent features, a portfolio can be trained that solves all but 4 instances.

latent features are removed under the WPMS dataset. We just use the WPMS dataset for the example because the CSP dataset only has 4 solvers and the results for the SAT dataset are similar to those presented. Note that while all of the latent features are necessary to recreate the performance of the VBS, it is possible to remove over half the latent features and still be able to solve all but 4 instances.

Estimating Latent Features

Although we do not have direct access to the latent features for a previously unseen instance, we can still estimate them using the original set of features.

Note that while it is possible to use the result of $\Sigma V'$ as a means of computing the final times, training a linear model on top of the latent features is the better option. True, both approaches would be performing a linear transformation of the features, but the linear model will also be able to automatically take into account any small errors in the predictions of the latent features. Therefore, the method proposed in this section would use a variety of models to predict each latent feature using the original features. The resulting predicted features will then be used to train a set of linear models to predict the runtime of each solver. The solver with the best predicted runtime will be evaluated.

To predict each of our latent features it is of course possible to use any regression based approach available in machine learning. From running just the five approaches that we have utilized in the previous sections, unsurprisingly we observe that a random forest provides the highest quality prediction. The results are presented in Table 4. Here SVD_predicted uses a random forest to predict the values of each latent feature and then trains a secondary linear model over the latent features to predict the runtime of each solver.

From the numbers we observe that this portfolio behaves similarly to the original Random Forest approach that simply predicts the runtime of each solver. This is to be expected since the entire procedure as we described so far can be seen as simply adding a single meta layer to the model. After all, one of the nice properties of forests is that they are able to capture highly nonlinear relations between the features and target value. All we are doing here is adding several forests that are then linearly combined into a single value. But this procedure does provide one crucial piece of new information.

Whereas before there was little feedback as to which features were causing the issues, we now know that if we have a perfect prediction of the latent features we can dramatically improve the performance of the resulting portfolio. Furthermore, we know that we don't even need to focus on all of the latent features equally, since Figure 1 revealed that we can survive with less than half of them.

Therefore, using singular value decomposition we can now identify the latent features that are hard to predict, the ones resulting in the highest error. We can then subsequently use this information to claim that the reason we are unable to predict this value is because the regular features we have available are not properly capturing all of the structural nuances that are needed to distinguish instances. This observation can subsequently be used to split the instances into two groups, one where the random forest over predicts and one where it under predicts. This is information that can then help guide researchers to identify new features that do capture the needed value to differentiate the two groups. This therefore introduces a more systematic approach to generating new features.

Just from the results in Table 4 we know that our current feature vectors are not enough when compared to what is achievable in Table 3. We also see that for the well studied SAT and CSP instances, the performance is better than for MaxSAT where the feature vectors have only recently been introduced.

We can then just aim to observe the next latent feature to focus on. This can be simply done by iterating over each latent feature and artificially assigning it the "correct" value while maintaining all the other predictions. Whichever feature thus corrected results in the most noticeable gains is the one that should be focused on next. Whenever two latent features tie in the possible gains, we should also focus on matching the one with the lower index, since mathematically, that is the feature that captures more of the variance.

If we go by instance names as a descriptive marker, surprisingly following our approach results in a separation where both subsets have instances with the same names. So following the latent feature suggested for the MaxSAT dataset we observe that there is a difference between "ped2.G.recomb10-0.10-8.wcnf' and "ped2.G.recomb1-0.20-14.wcnf". For CSP, we are told that "fapp26-2300-8.xml" and "fapp26-2300-3.xml" should be different. This means that the performance of a solver on an instance goes beyond just the way that instance was generated. There are still some fundamental structural differences between instances that our current features are not able to identify. This only highlights the need for a systematic way in which to continue to expand our feature sets.

Conclusions

Algorithm selection is an extremely powerful tool that can be used to distinguish between variations among problem instances and using this information to recommend the solver most likely to yield the best performance. Over just the last few years this field of research has flourished with numerous new algorithms being introduced that consistently dominate international competitions. Yet the entire success and failure of these approaches is fundamentally tied to the quality of the features used to distinguish between instances. Despite its obvious importance, there has been remarkably little research devoted to the creation and study of these features. Instead researchers prefer to utilize a shotgun approach of including everything they imagine might be remotely useful, relying on feature filtering to remove the unhelpful values. While this approach has worked out fine by capturing most of the key structural differences, at its core it depends on luck.

This paper therefore aims to introduce a more systematic way of coming up with a feature set for a new problem domain. Specifically, it is suggested that a matrix decomposition approach, like singular value decomposition (SVD), be used to analyze the performances on a number of solvers on a representative set of instances. This procedure will therefore generate a set of latent features that best separate the data and identify when a particular solver is preferred. Through experimental results on three diverse datasets, the paper shows that if we had access to these latent features, a straightforward linear model can achieve a level of performance identical to that of the perfect oracle portfolio.

While in practice we do not have access to these features, the paper further shows that we can still predict their values using our original features. A portfolio that uses these predicted values is able to achieve the same level of performance as one of the best portfolio approaches, if not slightly better. Furthermore, any latent feature where the prediction error is high helps identify the instances that the researcher must focus on differentiating. This will then lead to a more systematic method of generating features, and will later help us understand why certain instances can be solved by certain approaches, leading to a new wave of algorithm development.

Acknowledgments. This publication has emanated from research supported in part by a research grant from Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289.

It is also funded, in part, by the FP7-funded FET-Open project called Inductive Constraint Programming (ICON), Grant Agreement 284715.

References

Ansótegui, C.; Malitsky, Y.; and Sellmann, M. 2014. MaxSAT by improved instance-specific algorithm configuration. *AAAI*.

COSEAL. 2014. Configuration and selection of algorithms project. https://code.google.com/p/coseal/.

Ghahramani, Z.; Griffiths, T. L.; and Sollich, P. 2006. Bayesian nonparametric latent feature models. *World Meeting on Bayesian Statistics*.

Hebrard, E. 2008. Mistral, a constraint satisfaction library. *Proceedings of the Third International CSP Solver Competition*.

Helmert, M.; Röger, G.; and Karpas, E. 2011. Fast Downward Stone Soup: A baseline for building planner portfolios. *ICAPS*.

Hurley, B.; Kotthoff, L.; Malitsky, Y.; and O'Sullivan, B. 2014. Proteus: A hierarchical portfolio of solvers and transformations. *CPAIOR*.

Hutter, F.; Xu, L.; Hoos, H. H.; and Leyton-Brown, K. 2014. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence* 206:79–111.

Kadioglu, S.; Malitsky, Y.; Sabharwal, A.; Samulowitz, H.; and Sellmann, M. 2011. Algorithm selection and scheduling. *CP* 454–469.

Kotthoff, L.; Gent, I.; and Miguel, I. P. 2012. An evaluation of machine learning in algorithm selection for search problems. *AI Communications* 25:257–270.

Kotthoff, L. 2013. LLAMA: leveraging learning to automatically manage algorithms. Technical Report arXiv:1306.1031. http://arxiv.org/abs/1306.1031.

Kroer, C., and Malitsky, Y. 2011. Feature filtering for instance-specific algorithm configuration. *ICTAI* 849–855.

Kuhn, M. 2014. Classification and regression training. http://cran.r-project.org/web/packages/caret/caret.pdf.

Malitsky, Y.; Sabharwal, A.; Samulowitz, H.; and Sellmann, M. 2013. Algorithm portfolios based on cost-sensitive hierarchical clustering. *IJCAI*.

Martin, C., and Porter, M. 2012. The extraordinary SVD. *Mathematical Association of America* 838–851.

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. *AICS*.

Prasantha, H. 2007. Image compression using SVD. *Conference on Computational Intelligence and Multimedia Applications* 143–145.

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

Romanski, P. 2013. Fselector. CRAN - R Library.

Rutz, O. J.; Bucklin, R. E.; and Sonnier, G. P. 2012. A latent instrumental variables approach to modeling keyword conversion in paid search advertising. *Journal of Marketing Research* 49:306–319.

Thornton, C.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2013. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. *KDD* 847–855.

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2008. SATzilla: Portfolio-based algorithm selection for SAT. *JAIR* 32:565–606.

Xu, L.; Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2012a. Features for SAT. http://www.cs.ubc.ca/labs/beta/Projects/SATzilla/Report_SAT_features.pdf.

Xu, L.; Hutter, F.; Shen, J.; Hoos, H. H.; and Leyton-Brown, K. 2012b. SATzilla2012: Improved algorithm selection based on cost-sensitive classification models. SAT Competition.

Yang, W.; Yi, D.; Xie, Y.; and Tian, F. 2012. Statistical identification of syndromes feature and structure of disease of western medicine based on general latent structure mode. *Chinese Journal of Integrative Medicine* 18:850–861.