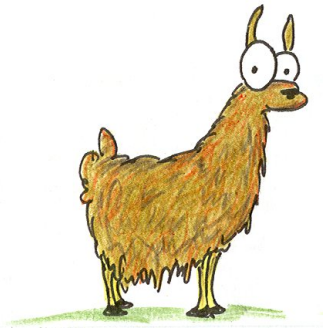
LLAMA: Leveraging Learning to Automatically Manage Algorithms

Lars Kotthoff

5th June 2013



Abstract

Algorithm portfolio approaches have achieved remarkable improvements over single solvers. However, the implementation of such systems is often highly specialized and specific to the problem domain. This makes it difficult for researchers to explore different techniques for their specific problems. We present LLAMA, a modular and extensible toolkit that facilitates the exploration of a range of different portfolio techniques on any problem domain. We describe the current capabilities and limitations of the toolkit and illustrate its usage on a set of example SAT problems.

1 Introduction

An algorithm portfolio [1,3] is a collection of state of the art solvers that are all capable of solving the same kind of problem. To use an algorithm portfolio for solving problems, a selection mechanism is required to determine the algorithm to use in the particular case. The idea behind algorithm portfolios is similar to the concept of portfolios in economics – instead of relying on a single entity to always deliver the best performance, the risk is spread over multiple entities.

Many contemporary solvers for artificial intelligence problems have complementing strengths and weaknesses. On a set of problems where one solver exhibits bad performance, another will excel while the picture may be reversed

on a different set of problems. Algorithm portfolios exploit this by relating the structure of the problem to solve to the performance of an individual solver or a set of solvers. The concept is closely related to the Algorithm Selection Problem [11], which is concerned with identifying the most suitable algorithm for solving a problem.

SAT is one of the first areas of artificial intelligence that algorithm portfolios and algorithm selection techniques have been applied to, and with great success. The most prominent system is probably SATzilla [14], which has dominated SAT solver competitions when it was introduced. More recent systems include ISAC [5], Hydra [13] and 3S [4].

Portfolio systems have achieved tremendous performance improvements over the previous state of the art in SAT, i.e. single solvers. They have dominated the field in terms of performance to such an extent that recent SAT solver competitions have introduced a separate track for portfolio solvers. The success of algorithm selection systems has not been limited to SAT though. A recent survey of the field [7] lists a number of additional application domains and provides an overview of the different approaches.

The main drawback of existing approaches is that they are highly tailored and customised for the particular problem domain or even set of problems. Even though the high-level approach can usually be applied to other problems, in practice this is almost always very difficult or even impossible. This makes it very difficult to compare different approaches and prototype new ideas especially for researchers who are not algorithm portfolio experts.

This is exactly what LLAMA addresses. Instead of providing yet another highly-specialised approach that has been tuned and customised to yield high performance on a specific data set, LLAMA is a framework that provides the building blocks for automatic portfolio selectors. It supports the most common approaches to portfolio selection and offers the possibility to combine them into more sophisticated approaches. It furthermore provides an implementation of the infrastructure that is required to build, evaluate and apply algorithm portfolios in practice. To the best of our knowledge, no similar toolkit exists.

2 Anatomy of LLAMA

The main focus of LLAMA is to provide the user with a framework for the implementation and evaluation of different algorithm selection approaches. It is *not* meant to provide turn-key algorithm portfolio systems that can be used in competitions or similar settings. While the functionality it provides can certainly be used to facilitate the creation of such systems, a lot of the technical details for practical algorithm selection systems are highly domain-specific. The main audience LLAMA targets are researchers that wish investigate and explore the performance characteristics of algorithm selection systems in general.

As such, many of the issues that are highly relevant to algorithm selection systems in competition settings do not have direct support in LLAMA. Such issues include the fall back to a default solver if feature computation takes too

long, the prediction of the time required to compute features or the running of a presolver to take care of the easy problems. LLAMA can certainly be used to address these issues by providing it with suitable input data, but there is no explicit support for it.

LLAMA is implemented as an R package and can be found at `http://cran.r-project.org/web/packages/llama/`, the development repository is at `https://bitbucket.org/lkotthoff/llama`. There are many advantages to this approach; one of the main ones is that all the functionality available in R can be used to build performance model. This is not limited to the functionality that is implemented in R – there are interfaces to many other packages, such as the well-known Weka machine learning toolkit [2].

The large number of machine learning approaches and algorithms available in R makes it possible to use LLAMA to quickly evaluate a range of different techniques for algorithm selection on given data, such as presented in [9]. Being able to do so is crucial for achieving good performance in practice.

The implementation of LLAMA provides a set of functions that facilitate the processing of performance data, the building of algorithm selection models and their evaluation. The functions are designed to assist the user rather than providing a black box that is easy to use, but hard to adjust.

3 LLAMA for domestic use: an example

The best way to explain LLAMA is to show a concrete example of how to use it. In the remainder of this section, we will illustrate how to use LLAMA to build and evaluate algorithm portfolio models. The examples given are based on the documentation of LLAMA, which is available through R's help system.

To start using LLAMA, the package needs to be loaded. We will also load example data that is included with the package. This data describes 2433 SAT instances and the performance of 19 SAT solvers on these instances.

```
> require(llama)
> data(satsolvers)
```

LLAMA provides the function `input` to process input and create the data structure it requires for further processing from a set of files that describe the features of each problem and the performance of each solver on each problem. This data structure contains some additional meta-information, such as the label of the best solver for each problem. Note that LLAMA does not limit the notion of solver performance to be time; the quality of a solution or the progress towards finding a solution could be used in the same manner. A data structure produced in this way is available in the variable `satsolvers` here.

3.1 Harnessing the LLAMA

To enable scientifically rigorous evaluation of an approach, LLAMA provides functions to split a data set into training and test sets. This is one of the

tedious and error-prone steps that researchers have to deal with in practice and that LLAMA aims to make less painful. We call the following command to partition the entire data into 10 stratified folds for cross-validation [6].

```
> folds = cvFolds(satsolvers)
```

Training an algorithm portfolio model can be done in a single line. In this example, we train a classification model that uses a C4.5 decision tree [10] to, based on the features of a problem, predict the fastest solver. The information on the fastest solvers is contained in the meta-data that LLAMA computes automatically on reading its input. The implementation for the C4.5 decision tree is taken from Weka (through the RWeka package).

```
> require(RWeka)
> res = classify(J48, folds)
```

The call to `classify` trains and tests models on each fold of the cross-validation data. After that, a model is trained on the *entire* data set. Both the predictions computed during cross-validation and the final model are returned. While the predictions allow to assess the expected performance of the approach, the returned model can be used as a building block for a portfolio system to obtain predictions on new data.

LLAMA provides several functions to evaluate the performance of an approach based on the predictions made. The code below computes the PAR10 score and the number of solved instances.

```
> parscore = sum(unlist(parscores(folds, res$predictions)))
> solved = sum(unlist(successes(folds, res$predictions)))
```

The above is already a complete example. This demonstrates the power and one of the main advantages of LLAMA – model building and evaluation can be done very concisely in just a few lines of R code.

3.2 Adjusting the harness straps

Almost all the functions that LLAMA provides take optional parameters for customisation. For example, the function to create cross-validation folds will create 10 stratified folds by default, but this behaviour can be changed easily to create e.g. 5 non-stratified folds.

```
> folds = cvFolds(satsolvers, nfolds=5, stratify=F)
```

Note that argument names are optional in R; the above could be rewritten in a more concise manner as `folds = cvFolds(satsolvers, 5, F)`. Similarly, the computation of PAR10 scores could be changed to e.g. PAR5 instead.

```
> parscore = sum(unlist(parscores(folds, res$predictions,
  factor=5)))
```

The main customisation point for building models is with the machine learning algorithm, `J48` in the example above. The parameters that will affect the learned model will be passed directly to `J48` and there is no need for LLAMA to provide support for this. In addition, the way parameters are passed to different machine learning algorithm implementations in R differ and it is infeasible to support all of them through a unified interface.

3.3 Exchanging the harness for a saddle

Building a classifier to predict the best solver for a problem is only one of the approaches to providing a selector for algorithm portfolios. A different approach is used for example in SATzilla [14]. For each solver in the portfolio, a regression model is induced to predict the performance of the solver on a particular problem. Given these predictions, the solver with the best predicted performance is chosen.

Doing this in LLAMA is a simple as calling a different function instead of the one used to build classification models. In this example, we use a linear regression performance model instead of the more sophisticated model in SATzilla, again from Weka. The input data is the same as for the classification example above.

```
> res = regression(LinearRegression, folds)
```

The results of applying the regression approach can be used and evaluated in the same way as above; we omit the repeated code.

Another common approach to algorithm selection that is used for example in ISAC [5] is to cluster the training problems and assign the best solver to each cluster based on the solver performances on the cluster problems. Again the only change is to call a different function.

Here we use the XMeans clustering algorithm from Weka. For clustering approaches, it is advisable to normalise the feature values across the training problems. This is achieved by passing an additional parameter to the model building function. Note that `normalize` is part of LLAMA and transforms the feature values to achieve a specific distribution. Details are given in the documentation.

```
> res = cluster(XMeans, folds, pre=normalize)
```

Again there is no difference in how the results of this call can be used. In particular, the model that can be used to make predictions on unseen data does not only encapsulate the clustering, but also how the input data should be normalised.

LLAMA provides a lot of additional functionality than we cannot cover in this document. There is however extensive[1] documentation available with examples for all the functionality. All the examples are self-contained and can

[1] For suitable definitions of "extensive".

be run directly. In addition, there is a set of unit tests that demonstrates not only how to use specific functions, but also how they are expected to behave.

To conclude this section, we provide an example for a more sophisticated approach, namely a hybrid regression-classification model [8]. The performance of the algorithms in the portfolio is predicted independently using linear regression models. However, instead of choosing the algorithm with the best predicted performance, a C4.5 decision tree is induced to make the prediction of best solver based on the predicted performances.

```
> res = regression(LinearRegression, folds, combine=J48,
  stack=TRUE)
```

4 The further domestication of LLAMA

The functionality currently implemented in LLAMA facilitates the exploration of the performance of many different approaches to algorithm selection. While it is extensive, there are a number of extensions of the current implementation that we are planning to provide in the future. An example of this is the possibility to provide a richer prediction output from the approaches that support it. At the moment, the only output of a model is the best algorithm to run on a particular problem. If a ranked list of algorithms was provided instead, these algorithms could be ran in parallel or according to a schedule. The main advantage of this approach is that if a bad prediction were made, the impact on overall performance is limited because a list of algorithms has a higher probability of containing the best algorithm even if predictions are bad.

Another obvious point for extension is to provide a larger number of implementations for different approaches to building performance models. In particular, there is currently no support for the approach that SATzilla uses in its most recent incarnation – for each pair of algorithms in the portfolio, train a model that predicts which one is faster and collate these predictions [15].

Another line of research into algorithm selection for SAT has focused on building hierarchical models which first predict whether a problem is satisfiable or not and have different models for solver selection for each outcome [12]. While there is currently no explicit support for approaches like this, they can be implemented through partitioning of the data that is given to LLAMA.

It is our intention for LLAMA to develop into a platform that not only facilitates the exploration and comparison of different existing approaches, but also the rapid prototyping of new approaches. The functions it provides take care of the infrastructure required to train and evaluate machine learning models in a scientifically rigorous way. Researchers wishing to improve algorithm portfolio selection do not need to concern themselves with the infrastructure, but can concentrate on the actual research.

LLAMA is still at an early stage of development and bugs may occur. In the end, the responsibility of interpreting and validating the results lies with the user. However, we have used LLAMA for a number of applications, found

and fixed a few bugs and are reasonably confident that it will useful to other people.

5 Acknowledgements

This work is supported by EU FP7 grant 284715. The drawing of a Llama is courtesy of `http://www.bluebison.net/`.

References

[1] Carla P. Gomes and Bart Selman. Algorithm portfolios. *Artificial Intelligence*, 126(1-2):43–62, 2001.

[2] 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.

[3] Bernardo A. Huberman, Rajan M. Lukose, and Tad Hogg. An economics approach to hard computational problems. *Science*, 275(5296):51–54, 1997.

[4] Serdar Kadioglu, Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Algorithm selection and scheduling. In *17th International Conference on Principles and Practice of Constraint Programming*, pages 454–469, 2011.

[5] Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. ISAC instance-specific algorithm configuration. In *Proceeding of the 2010 conference on ECAI 2010: 19th European Conference on Artificial Intelligence*, pages 751–756, Amsterdam, The Netherlands, The Netherlands, 2010. IOS Press.

[6] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1137–1143. Morgan Kaufmann, 1995.

[7] Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. Technical report, University College Cork, 2012.

[8] Lars Kotthoff. Hybrid regression-classification models for algorithm selection. In *20th European Conference on Artificial Intelligence*, pages 480–485, August 2012.

[9] Lars Kotthoff, Ian P. Gent, and Ian Miguel. An evaluation of machine learning in algorithm selection for search problems. *AI Communications*, 25(3):257–270, 2012.

[10] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1 edition, January 1993.

[11] John R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

[12] Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. Hierarchical hardness models for SAT. In *CP*, pages 696–711, 2007.

[13] Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. Hydra: Automatically configuring algorithms for portfolio-based selection. In *Twenty-Fourth Conference of the Association for the Advancement of Artificial Intelligence*, pages 210–216, 2010.

[14] 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.

[15] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Hydra-MIP: automated algorithm configuration and selection for mixed integer programming. In *RCRA Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion*, pages 16–30, 2011.