Random forests for estimation of parameter importance and interaction

Leslie Pérez Cáceres

Random forest can be used to model configuration performance defining the training data as parameter values as predictors and a performance measure as dependent variable. Such models can be applied to predict performance and also to analyze configuration fitness landscape. These models can be a useful tool during the configuration process and also after it for analyzing the performance data obtained in the process.

Irace generates considerable amounts of performance data and thus, in this report we explore the use of Random forest as a tool to analyze the data produced by the configurator. There are two possible main applications for these models:

- Support during the configuration process
- Post-execution analysis

Parameter importance could be estimated using the structure of the trees generated in random forests. One can assume that, if certain parameters are used as split in many trees, it means that their values are "important" for predicting algorithm performance.

Some importance measures in Random Forest may indicate how much a variable contributes to have a good classification (or regression). These measures can be based on the mean squared error or the entropy. There are other important measures based on the structure of the tree, which may indicate somehow a "different type of importance". For example, if a parameter split is commonly located close to the root of the trees it might indicate higher importance as they allow to divide the data to predict performance. This can be evaluated with an indicator called *"mean min depth"* which indicates the average of the min depth of a variable across the trees in the forest. The min depth of a variable `v` is the depth of the max sub-tree of the variable. The max sub-tree is the sub-tree whose root uses `v` to split the data and it is closest to the root of the full tree.

The mean min depth indicator can be extended to study interactions between parameters, for this, the min depth idea can be applied to calculate the min depth of a variable in the max sub-tree of another variable. The smaller the min depth, the more likely is that there is a *relevant* relationship between variables. Performing this in one tree is not enough as a small min depth can be just due to the stochasticity of the tree building process. Nevertheless, if this measure is aggregated across several trees, tendencies may indicate real interactions between parameters.

The randonForestExplainer (https://cran.r-project.org/web/packages/randomForestExplainer/vignettes/randomForestExplainer.html) provides different importance measures for the variables:

- mean_min_depth: mean depth of the subtree closest to the tree root, where a variable is root of the subtree
- no_of_nodes: number of nodes in which the variable was used to split
- mse_increase: increment of prediction mean squared error (regression)
- accuracy_decrease: reduction in classification accuracy (classification)
- no_of_trees: number of trees in which the variable was used
- times_a_root: number of times a variable was selected as root variable

We can have an idea of the importance of the parameters using these measures, but since the configuration data is inherently incomplete and dependent, we need to adjust the data to detect importance.

1. Configuration data pre-processing and model training

First we create a data set based on configuration data obtained from irace. We apply some transformations to the configurations data in order to deal with conditionality of parameters:

- we remove parameters that have only one value (NA or value)
- impute missing parameter values: – special value `"__miss__"` for categorical parameters – value out of domain for numerical (`upper_bound * 2`)
- add instances as a categorical variable
- add a reference predictor variable that is numerical randomly distributed variable (dummy): current implementation add this variable as integer with range `[1,10]`

TODO: evaluate discretization of numerical parameters for modeling

Once we have filtered the data, we train the forest and calculate importance of the predictors (parameters). All parameters showing less importance than the reference variable are removed, in this way we aim to filter unimportant parameters.

TODO: how to define those that have similar performance to dummy (it could be a bit higher). Maybe use the p-value? or a combined importance measure approach?

If a conditional parameter (`param1`) is detected as important, we need to detect if the effect over the performance really comes from the parameter values or the enabling/disabled nature of it:

- filter all configurations where the conditional parameter is not active (`param1=NA`)
- remove parameters that have only one value (NA or value) after the filter
- resample dummy variable to avoid any issue with the values
- re-train the model and check if the parameter is still detected as important
- if the parameter is not detected as important, then the parameter is removed from the important parameter list

TODO: evaluate if in case a parameter is not detected as important the effect can be added to the importance of the root activator

TODO: evaluate an inverse process; first filter detect importance in independent parameters and then analyse dependent ones

We then select the top N most important parameters and perform analysis of interactions (currently 5):

- add reference variable to important parameters to perform this analysis (dummy)
- re-train the model including only important parameters
- remove any interaction less important than the interactions detected with the dummy parameter
- aggregate important measures of bidirectional interactions (`param1 -> param2` and `param2 -> param1`)

1.1 What to predict?

When building the data set one can define the dependent variable as the performance directly or use the ranking of the configurations. Using performance is the most straightforward strategy, but it has the downside of having numerical differences that might make more difficult to model the performance as the model must discriminate if the scale of these differences are due to the parameter values, the instances or the stochasticity. Large numerical differences can be reduced by normalizing the performance, but this does not removes all the issues associated to compare the performance across instances.

Using the ranking of the configurations allows to remove the numerical scale differences and thus, allow the model to focus on the relative performance. Nevertheless, this strategy has one practical problem when using irace data, the configuration data is not complete and thus, the rankings are not comparable. An option would be to to impute the missing performance, this can be applied also for the performance-based forest, but it would not be advised as it is not necessary. The performance of a missing experiment could be imputed by:

- Building a model to predict it

- Assume missing performance to be representative of the irace elimination procedure
- Assume missing performance as the overall worst performance reported in an instance

The second option is possible due to the nature of the irace data as missing evaluations are due to the elimination of configurations that are poor performing. In this work we apply this option, we impute the rank of missing configurations as assigning them a surrogate perforamce and then calculating the ranking.

```
performance = max_performance + na_rank
```

where `max_performance` is the worst performance in the instance and `na_rank` is the ranking of the performance regarding the number of not executed instances in the race by the configuration whose performance is missing. The more executions a configurations performed, the smaller the ranking and thus, the best overall ranking.

TODO: this imputation strategy is not completely conceptually correct as some configurations are better than others even if they were discarded, but since they are eliminated they all have missing data experiments. A better strategy would be impute based on the iteration in which the configurations were created, assuming that configurations in later iterations should have in average better performance.

In this case, further filtering training data might be useful:

- removing instances with few configurations executed in them
- removing configurations with too few instances executed

These filters are particularly useful due the technical limitations of random forests. Categorical variables in the forest implementation used cannot have more than 53 variable levels. Then, if a categorical parameter has a very large domain or the set of training instances is composed of more than 53 instances we can apply such filters.

Finally with the imputed ranks in place it is possible to compute the rankings on all the data set and train the forest as normal.

An alternative is to predict the ranking directly is to predict the **ranking quartile** to which the configuration belongs. In this way, the modelling task would be more a classification task, in which the model aims to classify to which quartile a configuration belongs.

1.2 About instances

The use of instances as predictors in the ranked data could be used to estimate if the scenario is heterogeneous. If the instance is detected as an important variable, it means that it is useful to predict rankings, thus, means that different configurations could be better for certain instances. Note that this is not necessarily the case with a forest trained to predict performance as the importance of the instance variable might represent the numerical difference of the solution quality. Furthermore, analyzing the splits could give an idea of the instance sets. Interactions might indicate parameters the can are the responsible of generating better results for one instance or another.