Importance transdapters

Miron B. Kursa

November 19, 2024

1 Overview

Boruta is essentially a significance test for the importance of features generated by some importance source. This source is usually the normalised permutational importance from a Random Forest [1] model, although other options are possible. The interface between Boruta and importance source is called *importance adapter*, which is an R function fed into `getImp` argument of the `Boruta` function. They should expect three arguments, `x`, for predictors table with shadows, `y`, for the decision, and `...`, for arbitrary additional arguments relayed from the Boruta call. For instance, `Boruta(Species .,iris,num.threads=1)` is going to pass the `num.threads` argument to the adapter function, which in turn will append it to its call to `ranger`. The adapter output is expected to be a numerical vector of the same length as `ncol(x)`, with i-th element being the importance of the i-th column of `x`; importance scale is arbitrary, but it must be a finite real number and rise with feature importance. Adapters shipped with the package are, by convention, given names starting with `getImp`, with the default called `getImpRfZ`.

Transdapters are functions which wrap the adapter function with additional behaviours — they may intercept and modify the transient data set produced for the adapter, as well as modify the returned importance scores. They are implemented as functions expecting at least one argument, the adapter function, and immediately returning a function wrapping the given adapter.

One can write an example no-op transdapter in the following way:

```
> library(Boruta)
> noopTransdapter<-function(adapter=getImpRfZ){
+  adapter
+ }
```

... and use it this way:

```
> set.seed(17)
> Boruta(Species~.,iris,getImp=noopTransdapter())

Boruta performed 9 iterations in 0.2164261 secs.
 4 attributes confirmed important: Petal.Length, Petal.Width,
```

```
Sepal.Length, Sepal.Width;
 No attributes deemed unimportant.
```

As expected, this transdpater has not altered the Boruta result.

The Boruta package ships with several experimental transdapters. Let us now briefly discuss each of them.

2 Impute transdapter

Many potential importance sources, in particular the default ranger [2] implementation of Random Forest, can't handle missing values (`NA`s) in its input. One can avoid this problem by performing imputation, that is a process of filling missing values with substitutes generated in some way, depending on the analysis aims and assumptions. The problem here is that such a process may spuriously increase the importance of imputed features, bringing bias to the Boruta output, in particular when the imputation process uses the decision feature for its inference.

The impute transdapter alleviates this problem by performing a very simple, stochastic imputation on input, yet repeated every time importance is requested. This way, imputation biases have a chance to cancel out, moreover, the effect of imputation is also reflected on shadows, allowing Boruta to better quantify its impact.

Let us test this on the `srx` data:

```
> set.seed(17)
> data(srx)
> srx_na<-srx
> # Randomly punch 25 holes in the SRX data
> holes<-25
> holes<-cbind(
+  sample(nrow(srx),holes,replace=TRUE),
+  sample(ncol(srx),holes,replace=TRUE)
+ )
> srx_na[holes]<-NA
> # Use impute transdapter to mitigate them with internal imputation
> Boruta(Y~.,srx_na,getImp=imputeTransdapter(getImpRfZ))

Boruta performed 26 iterations in 0.311367 secs.
 5 attributes confirmed important: A, AnB, AoB, B, nA;
 3 attributes confirmed unimportant: N1, N2, N3;
```

The precise method of imputation is to fill gaps with a random sample of other values in a feature, taken with replacement. All-missing features are replaced with a vector of zeros, which is non-important by design and should be given an importance of zero by a typical importance source.

One should note that this method is best suited for data sets with only a minor fraction of missing values, preferably randomly distributed. Also it rather underestimates the importance of incomplete features.

3 Decohere transdapter

The main gain from using Boruta is that it can take into account multivariate interactions between groups features and the decision. Decoherence transdapter can be used to validate whether multivariate inference is required to identify a certain feature as relevant, or just its marginal interaction with the decision suffices.

The decoherence transformation is defined for categorical decisions only, and is performed by randomly mixing the order of values in each feature, yet only within decision classes. This way, feature-decision joint distribution is preserved, but feature-feature interactions are asymptotically knocked out.

Let us try this on the `srx` data:

```
> set.seed(17)
> Boruta(Y~.,srx,getImp=decohereTransdapter())

Boruta performed 14 iterations in 0.2149677 secs.
 2 attributes confirmed important: AnB, AoB;
 6 attributes confirmed unimportant: A, B, N1, N2, N3 and 1 more;
```

We can see that most originally confirmed features are no longer relevant after decoherence transform is applied, save for `AnB` and `AoB` of a substantial marginal importance.

One should note that this approach is a heuristic and should be treated with reservation. For instance, it may reduce the estimated shadow importance and henceforth spuriously elevate originally rejected features to a confirmed status.

4 Conditional transdapter

The conditional transdapter applies the wrapped importance source to each stratum of a given stratification vector, and averages importance scores with weights proportional to strata sizes. This allows one to control for a certain variable or inject additional information.

Note that each strata should be large enough to be a proper sample for the information source used; stratification into too many very small groups may lead to spurious results. Moreover, be aware that certain importance sources may be incompatible with the way this transdapter collects importance from strata; additional normalisation might be necessary.

References

[1] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[2] Marvin N. Wright and Andreas Ziegler. ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R. *Journal of Statistical Software*, 77(1), 2015.