GreedyExperimentalDesign: Finding Experimental Designs using Greedy Search with Random Restarts

Adam Kapelner et al.
Queens College, City University of New York
Department of Mathematics

Abstract

This package greedily finds experimental designs with good balance based on a balance metric of your choice.

Keywords: experimental design, greedy search, optimization, R, Java.

1. Introduction

Assume a randomized controlled two-arm experiment with n subjects and treatment (T) and control (C) denoted by the n-binary vector $\mathbf{1}_T$ where entries of 1 in location i indicates subject i was administered T and entries of 0 indicates C. Define the number of treatments $n_T := \sum_{i=1}^n \mathbf{1}_{T,i}$ and the number of controls $n_C := n - n_T$. For each subject, p covariates $\boldsymbol{X} := [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p]$ are measured. Define $\bar{\boldsymbol{X}}_T$ as the p-vector of sample averages for each of the covariates in subjects where $\mathbf{1}_{T,i} = 1$ (the treatments) and $\bar{\boldsymbol{X}}_C$ as the p-vector of sample averages for each of the covariates in subjects where $\mathbf{1}_T = 0$ (the controls). The investigator will eventually measure one response for each subject collected in the n-vector $\boldsymbol{y}$, but this is not our current interest. We assume that each of the p covariates is standardized.

There are many functions of $\mathbf{1}_T$ and $\boldsymbol{X}$ that will yield higher efficiency when testing null hypotheses about effects of the treatment. Below are a couple:[1]

1. (ABS) $\sum_{j=1}^p \left|\bar{\boldsymbol{X}}_{T,j} - \bar{\boldsymbol{X}}_{C,j}\right|$ which is a measure of balance between the covariate distributions. Covariate distribution permitting, zero is the optimal value.

2. (MAHAL) $\frac{n_T n_C}{n} \left(\bar{\boldsymbol{X}}_T - \bar{\boldsymbol{X}}_C\right)^\top \boldsymbol{S}_{\boldsymbol{X}}^{-1} \left(\bar{\boldsymbol{X}}_T - \bar{\boldsymbol{X}}_C\right)$ is a Mahalanobis-like distance metric. Covariate distribution permitting, zero is the optimal value.

We will fix $n_T = n_C$ and then minimize one of the two objective functions above. Other balcne objective functions can be programmed in by extending the `ObjectiveFunction` class in Java.

[1] There are also metrics which measure the similarity between the two joint densities f_T and f_C which we may want to explore later.

2. Greedy Switches Algorithm

We begin with an n-subject dataset $\boldsymbol{X}$. Each subject can be assigned to treatment or control but $n_T = n_C$. Thus, the space of possible $\mathbf{1}_T$'s has $\binom{n}{n/2}$ elements. We outline the algorithm now below:

Algorithm 1 Greedy search for design vector local maximum

```
1: Let $\mathbf{1}_T^*$ be a random draw from the space of $\binom{n}{n/2}$ balanced vectors.
2: Create a list of the indices of size $n/2$ corresponding to where $\mathbf{1}_T^* = 1$ (call it $I_T$). Create
   a list of the indices of size $n/2$ corresponding to where $\mathbf{1}_T = 0$ (call it $I_C$).
3: For every pair in $I_T \times I_C$ (i.e. $\frac{n}{2} \times \frac{n}{2} = \frac{n^2}{4}$ total pairs), switch the 0 and 1 within $\mathbf{1}_T^*$ and
   record the resulting value of the objective function. Find the switch which yielded the
   minimum value of the objective function. Make that switch inside $\mathbf{1}_T$.
4: Repeat the previous step until the minimum value of the objective function does not
   improve.
5: Repeat the entire procedure (steps 1–4) $d$ times where $d$ is constrained only by your
   computing resources and time.
```

3. The package

We first load the package which relies on **rJava**. Thus, the first line should give parameters for the Java Virtual Machine initialization. In our example, 1GB of memory is probably enough:

```
> options(java.parameters = "-Xmx1000m")
> library(GreedyExperimentalDesign)
```

To construct a **GreedyExperimentalDesign** object, use the function `initGreedyExperimentalDesignObject`. This function takes your data as parameter `X` which can be a matrix or dataframe. You then specify the objective function which is either ABS or MAHAL.

The next thing to specify is `max_designs` which is the d value in Algorithm **??** line 5 which controls how many searches are done in the treatment vector space. As the speed of the algorithm depends on n, p and the objective function, it is hard to gauge how long it will take to finish all searches. Luckily, it doesn't matter, since the searching is done in the background and you can stop it whenever you wish. Thus, we recommend making this parameter very large. As we will see in the examples, there is benefit to doing an exhaustive search on the $\mathbf{1}_T$ space (well, as exhaustive as you can).

The last parameter is the number of cores. This should be all your cores if you are using a server. On a workstation which you are using, all your cores less one if you still want your workstation to be usable. The algorithm is perfectly parallelized; thus doubling the cores will double your rate of vectors found.

To begin the search use the `startGreedySearch` function.

```
> ged = initGreedyExperimentalDesignObject(X,
        max_designs = 1000, num_cores = 3, objective = "abs_sum_diff")
> startGreedySearch(ged)
```

For this example, $\boldsymbol{X}$ had 200 subjects with 40 measurements each and each measurement was generated as independent realizations of a standard normal.

Once the search has begun, it runs in the background on the number of cores specified. At any time you can check the progress via printing the object:

```
> ged
The search has found 144 vectors thus far (14%).
```

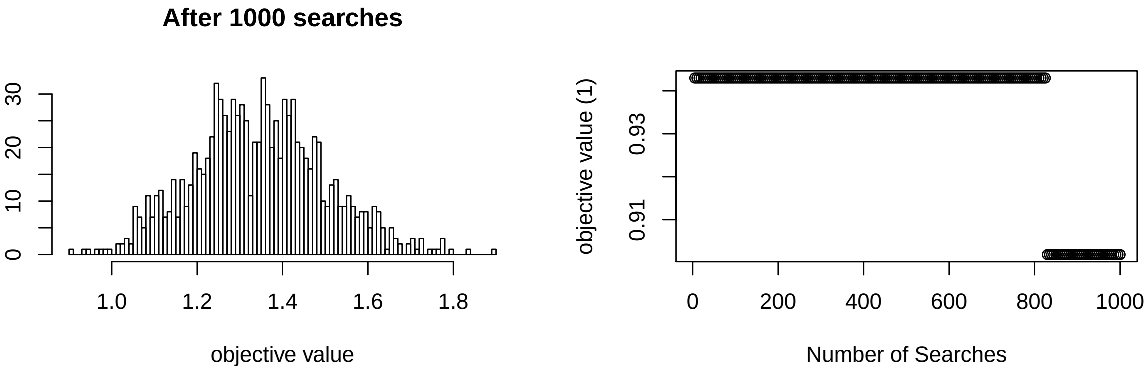
When it is done, it will display:

```
> ged
The search has completed. 1000 vectors have been found.
```

You can plot the histogram of the objective values for each of the d vectors found as well as the minimum objective value as a function of the number of searches via:

```
> plot(ged)
```

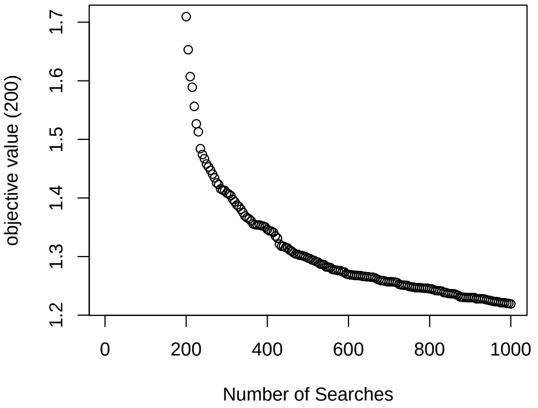
which produces



An important application of this package is the ability to use these treatment vectors in experimentation. To run a hypothesis test, a permutation test can be used. Thus, you will need many vectors, let's say c vectors. You can see the cth order statistic as a function of the search in realtime. For instance, if you want 200 vectors,

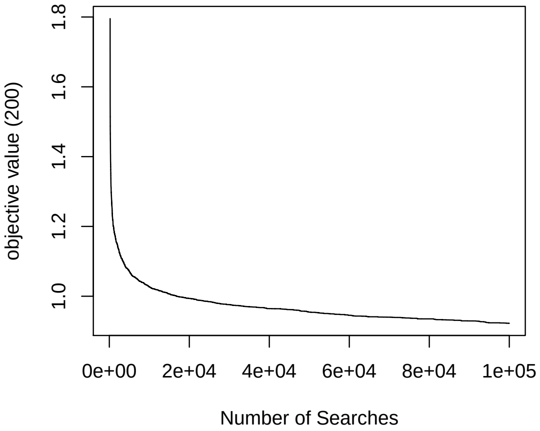
```
> plot_obj_val_order_statistic(ged, order_stat = 200)
```

which produces



as we can see, there is benefit to running more than $d = 1000$ here as we can still find a set of 200 treatment vectors with better balance.

In fact, even at 100000 searches, there is still room for improvement.



If it any time you wish to stop the search, you can use

```
> stopGreedySearch(ged)
```

Results (including vectors) can be retrieved via the `resultsGreedySearch` method. This method has a parameter `max_vectors` which will return this numbed of vectors with the smallest objective values. We do not recommend returning all d vectors as this is an expensive operation. The result object is a list. We demonstrate how to pull out the best design

```
> res = resultsGreedySearch(ged, max_vectors = 100)
> best_design = res$indicTs[1, ]
```

Replication

The stable version of **GreedyExperimentalDesign** will be soon on CRAN and the development version is located at `https://github.com/kapelner/GreedyExperimentalDesign`. You can install the package by cloning the repository, then running `ant` to compile the Java code then running `R CMD INSTALL GreedyExperimentalDesign`. The package code is licensed under GPL3 and LGPL. Results, tables, and figures found in this paper can be replicated via the scripts located in the `git` repository in the folder `GreedyExperimentalDesign/vignettes`.

Acknowledgements

We thank Simon Urbanek for his very generous help with **rJava**.

Affiliation:

Adam Kapelner
Department of Mathematics
Queens College, City University of New York
64-19 Kissena Blvd Room 325
Flushing, NY, 11367
E-mail: `kapelner@qc.cuny.edu`
URL: `http://kapelner.com`