Using LikeLTD

David Balding

November 5, 2013

```
NULL
```

1 Code usage

Computing a likehood model takes place in three-parts: (i) creation of the hypothesis, (ii) creation of the model itself from the hypothesis, (iii) maximizing the likehood function over the set of nuisance parameters.

1.1 Creating a hypothesis

A hypothesis consists of all the parameters are used to describe the model, e. g. the known profiles to include, the number of unprofiled contributors, and whether to include dropin. A proper description of the hypothesis should be independent of other information, including neither information about the maximization of the likelihood nor any information which needs to be computed.

```
>   require(likeLTD)
>   require(DEoptim)
>   # Case we are going to be looking at.
>   caseName = 'hammer'
>   datapath = file.path(system.file("extdata", package="likeLTD"),
+                        caseName)
>   args = list(
+     databaseFile = NULL,
+     cspFile    = file.path(datapath, 'hammer-CSP.csv'),
+     refFile      = file.path(datapath, 'hammer-reference.csv'),
+     nUnknowns    = 0,
+     doDropin     = TRUE,
+     ethnic       = "EA1",
+     adj          = 1.0,
+     fst          = 0.02,
+     relatedness  = c(0, 0)/4
+   )
>   # Create hypothesis for defence and prosecution.
>   defenceHyp = do.call(defence.hypothesis, args)
>   prosecuHyp = do.call(prosecution.hypothesis, args)
```

Two methods are provided to facilitate the creation of a hypothesis from a common minimal set of input parameters, `defence.hypothesis` for the defence

and `prosecution.hypothesis` for the prosecution. These two methods read the allele database, the known profiles, and the crime-scene profile from file. They also automate sensible default decisions about the input, (determining which known profiles need be subject to dropout; in the defence hypothesis the queried individual is replaced by an unprofiled contributor; in the prosecution case, relatedness is set to zero.) These default choices may be further modified by the user at this point. These methods return lists with all the input needed for model execution.

```
>   defenceModel <- create.likelihood(defenceHyp)
>   prosecuModel <- create.likelihood(prosecuHyp)
```

`create.likelihood` returns a method which takes as arguments the nuisance parameters and computes the full weight of evidence, e. g. the product of the likelihoods and penalties associated with each locus.

```
>   defenceModel(rcont=c(1, 1e-8, 1.63),
+                degradation=c(10^-2.27, 10^-2.74, 10^-2.47),
+                locusAdjustment=list(D3=0.983, vWA=1.010, D16=1.028,
+                                     D2=1.072, D8=1.020, D21=0.930,
+                                     D18=0.850, D19=0.932,
+                                     TH01=1.041, FGA=0.916),
+                dropout=c(0.5072, 1e-8),
+                dropin=1.0216,
+                power=-4.4462)
```

The function above returns a scalar which represents the weight of evidence for the given values of the nuisance parameters. One could then use `defenceModel` to perform an optimisation or to create a plot with respect to various arguments. For instance, the following leads to Fig. 1:

```
>   require(ggplot2)
>   require(scales)
>   # Function that winnows down to a single value
>   scalarWoE <- function(x) {
+      defenceModel(locusAdjustment=list(D3=0.983, vWA=1.010,
+                                        D16=1.028, D2=1.072,
+                                        D8=1.020, D21=0.930,
+                                        D18=0.850, D19=0.932,
+                                        TH01=1.041, FGA=0.916),
+                   dropout=c(0.5072, 1e-8),
+                   degradation=c(10^-2.27, 10^-2.74, 10^-2.47),
+                   rcont=c(x, 1e-8, 1),
+                   dropin=1.0216,
+                   power=-4.4462)
+   }
>   x = 0:30/30 * 3e0
>   data = data.frame(x=x, y=sapply(x, scalarWoE))
>   plots <- ggplot(data, aes(x=x, y=y))                    +
+               geom_line()                                 +
+               xlab("Relative contribution of Victim 1") +
```

```
+                ylab("Weight of Evidence")                +
+                scale_y_log10(
+                  labels=trans_format("log10", math_format(10^.x)))
>   print(plots)
```

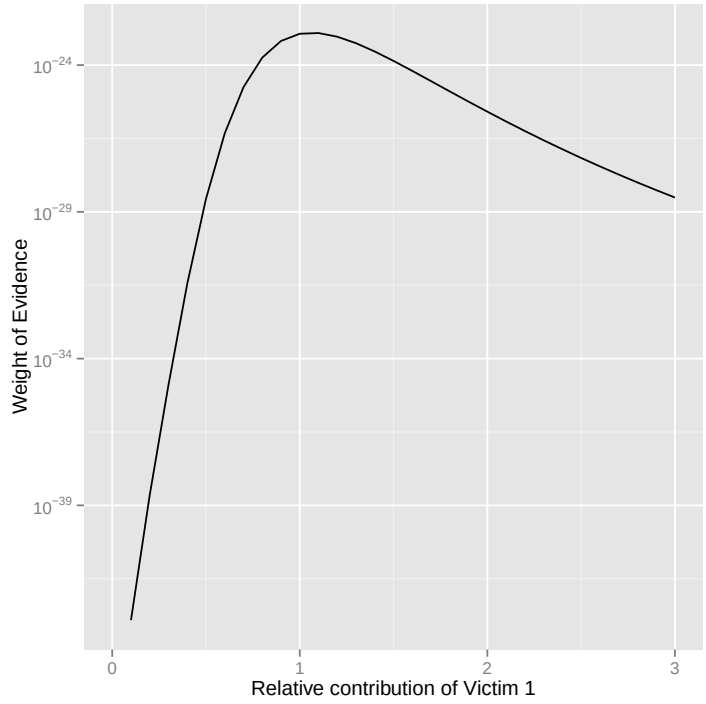

Figure 1: Logarithmic plot of the weight of evidence versus the relative contribution from "Victim 1". The likelihood is for the Hammer case, with one unprofiled contributor ("X"), and including dropin. The fixed parameters are given in Tab. 1.

1.2 Maximizing the likehood

Once we have a likelihood method, it is possible to use the `stats` package to maximize it. However, the likelihood method takes several arguments (`rcont`, `degradation`, etc), whereas `DEoptim::DEoptim` expects a methods which takes only a vector as argument. Hence, we need to transform our method into the form the optimisation method expects:

```
>   skeleton = initial.arguments(defenceHyp)
>   vector.model <- function(x) {
+     args <- relist(x, skeleton)
+     args[["degradation"]] = 10^args[["degradation"]]
+     result <- do.call(defenceModel, args)
+     log10(result)
```

```
+   }
>   # Call vector.model with vector argument.
>   arguments = skeleton
>   arguments[["degradation"]] = log10(arguments[["degradation"]])
>   vector.model( as.vector(unlist(arguments)) )
```

The new method `vectorModel` achieves three objectives: (i) it recreates the list of arguments for `defenceModel`, (ii) it transforms the degradation parameter from an exponential form, (iii) it takes the logarithm of the weight of evidence. The last two points make optimisation somewhat easier. We can now apply the optimisation methods on `vectorModel`.

```
>   require(stats)
>   # define upper and lower bounds for constrained maximization
>   nloci = ncol(defenceHyp$cspProfile)
>   upper = list(locusAdjustment = rep(1.5, nloci),
+                dropout         = c(1-1e-3, 1-1e-3),
+                degradation     = rep(-1e-3, 3),
+                rcont           = rep(100, 2),
+                dropin          = 1,
+                power      = -2 )[names(arguments)]
>   lower = list(locusAdjustment = rep(0.5, nloci),
+                dropout         = c(1e-3, 1e-3),
+                degradation     = rep(-20, 3),
+                rcont           = rep(1e-3, 2),
+                dropin          = 1e-3,
+                power      = -6 )[names(arguments)]
>   # perform maximization
>   result <- DEoptim(fn  = vector.model,
+                     upper = unlist(upper),
+                     lower = unlist(lower),
+                     control = list(strategy=3, itermax=500)
+                     )
>   opti = relist(result$optim$bestmem, skeleton)
>   cat(sprintf("Resulting Weight of Evidence: 10^%f\n",
+               -result$optim$bestval))

[1] -46.40983

Resulting Weight of Evidence: 10^Inf
```

The above calculates the maximum of the likelihood using a differential evolution (DE) algorithm to to perform evolutionary global optimization. The particular flavor of DE algorithm used here allows the user to set upper and lower bounds for parameters. Upon convergence, it returns a list with the optimum and its location. Please see the `DEoptim` package for description.

The functionality of the above code can be achieved more succinctly through a convenience method provided by LikeLTD `optimisation.params`. It returns a list of adequate arguments for `optim` given a hypothesis:

```
>   params = optimisation.params(defenceHyp, verbose=FALSE)
>   params$control$itermax=50 # Less strict convergence, for demo purposes.
```

```
>   results <- do.call(DEoptim, params)
>   arguments <- relistArguments(results$optim$bestmem, defenceHyp)
```

Running the above yields the parameters in Tab. 1. The last line transforms the linear vector of arguments returned by `DEoptim` back into a more meaningful list, much as `relist` did earlier. However, it takes care of some specialized problems with the operation, and should be preferred over `relist`.

A method for ensuring proper convergence is provided by LikeLTD `DEoptimLoop`. This calls the external function `DEoptim::DEoptim`, but crucially compares the optimised result every 50 generations with the previous optimised result, and quits once the relative difference is below the given tolerance.

1.3 Testing

LikeLTDcomes a fairly extensive suite of tests. The tests can be run as part of the installation process, or using the following commands:

```
> library(svUnit)
> library(likeLTD)
> runTest( svSuite("package:likeLTD") )
> Log()
```

Although not shown here, this snippet will print results for each tests. Each should return "OK".

```
Iteration: 1 bestvalit: 77.371073 bestmemit:    0.753767    0.938040    0.714451    0.6278
Iteration: 2 bestvalit: 74.673777 bestmemit:    1.435828    0.560219    0.834674    1.3634
Iteration: 3 bestvalit: 66.159163 bestmemit:    0.782973    0.841728    0.578927    0.5439
Iteration: 4 bestvalit: 66.159163 bestmemit:    0.782973    0.841728    0.578927    0.5439
Iteration: 5 bestvalit: 61.188081 bestmemit:    0.753767    0.938040    0.714451    0.6687
Iteration: 6 bestvalit: 59.238295 bestmemit:    0.753767    0.938040    0.714451    0.6687
Iteration: 7 bestvalit: 59.238295 bestmemit:    0.753767    0.938040    0.714451    0.6687
Iteration: 8 bestvalit: 59.238295 bestmemit:    0.753767    0.938040    0.714451    0.6687
Iteration: 9 bestvalit: 59.238295 bestmemit:    0.753767    0.938040    0.714451    0.6687
Iteration: 10 bestvalit: 59.238295 bestmemit:    0.753767    0.938040    0.714451    0.668
Iteration: 11 bestvalit: 59.238295 bestmemit:    0.753767    0.938040    0.714451    0.668
Iteration: 12 bestvalit: 59.238295 bestmemit:    0.753767    0.938040    0.714451    0.668
Iteration: 13 bestvalit: 59.238295 bestmemit:    0.753767    0.938040    0.714451    0.668
Iteration: 14 bestvalit: 59.238295 bestmemit:    0.753767    0.938040    0.714451    0.668
Iteration: 15 bestvalit: 48.722432 bestmemit:    1.103435    0.674143    0.878928    0.548
Iteration: 16 bestvalit: 47.739146 bestmemit:    1.175779    0.547791    1.137900    1.275
Iteration: 17 bestvalit: 47.739146 bestmemit:    1.175779    0.547791    1.137900    1.275
Iteration: 18 bestvalit: 46.526704 bestmemit:    1.175779    0.547791    1.137900    1.275
Iteration: 19 bestvalit: 46.526704 bestmemit:    1.175779    0.547791    1.137900    1.275
Iteration: 20 bestvalit: 46.526704 bestmemit:    1.175779    0.547791    1.137900    1.275
Iteration: 21 bestvalit: 46.526704 bestmemit:    1.175779    0.547791    1.137900    1.275
Iteration: 22 bestvalit: 43.971049 bestmemit:    1.175779    0.547791    1.137900    1.275
Iteration: 23 bestvalit: 43.723295 bestmemit:    1.175779    0.547791    1.137900    1.275
Iteration: 24 bestvalit: 43.466552 bestmemit:    1.175779    0.547791    1.137900    1.275
Iteration: 25 bestvalit: 43.466552 bestmemit:    1.175779    0.547791    1.137900    1.275
Iteration: 26 bestvalit: 39.828037 bestmemit:    1.175779    0.930315    1.137900    1.275
Iteration: 27 bestvalit: 39.828027 bestmemit:    1.175779    0.930315    1.137900    1.275
Iteration: 28 bestvalit: 39.531274 bestmemit:    1.175779    0.930315    1.137900    1.275
Iteration: 29 bestvalit: 39.531274 bestmemit:    1.175779    0.930315    1.137900    1.275
Iteration: 30 bestvalit: 39.531274 bestmemit:    1.175779    0.930315    1.137900    1.275
Iteration: 31 bestvalit: 39.531274 bestmemit:    1.175779    0.930315    1.137900    1.275
Iteration: 32 bestvalit: 39.355714 bestmemit:    1.175779    0.930315    1.137900    1.275
Iteration: 33 bestvalit: 39.355714 bestmemit:    1.175779    0.930315    1.137900    1.275
Iteration: 34 bestvalit: 39.355714 bestmemit:    1.175779    0.930315    1.137900    1.275
Iteration: 35 bestvalit: 39.331535 bestmemit:    1.175779    0.930315    0.972301    1.275
Iteration: 36 bestvalit: 39.331535 bestmemit:    1.175779    0.930315    0.972301    1.275
Iteration: 37 bestvalit: 39.331535 bestmemit:    1.175779    0.930315    0.972301    1.275
Iteration: 38 bestvalit: 39.331535 bestmemit:    1.175779    0.930315    0.972301    1.275
Iteration: 39 bestvalit: 38.847680 bestmemit:    1.175779    0.930315    0.972301    1.111
Iteration: 40 bestvalit: 38.847680 bestmemit:    1.175779    0.930315    0.972301    1.111
Iteration: 41 bestvalit: 38.847680 bestmemit:    1.175779    0.930315    0.972301    1.111
Iteration: 42 bestvalit: 38.847680 bestmemit:    1.175779    0.930315    0.972301    1.111
Iteration: 43 bestvalit: 38.847680 bestmemit:    1.175779    0.930315    0.972301    1.111
Iteration: 44 bestvalit: 38.847680 bestmemit:    1.175779    0.930315    0.972301    1.111
Iteration: 45 bestvalit: 38.847680 bestmemit:    1.175779    0.930315    0.972301    1.111
Iteration: 46 bestvalit: 38.008512 bestmemit:    0.913107    0.933845    1.070469    0.991
Iteration: 47 bestvalit: 38.008512 bestmemit:    0.913107    0.933845    1.070469    0.991
Iteration: 48 bestvalit: 38.008512 bestmemit:    0.913107    0.933845    1.070469    0.991
Iteration: 49 bestvalit: 38.008512 bestmemit:    0.913107    0.933845    1.070469    0.991
Iteration: 50 bestvalit: 38.008512 bestmemit:    0.913107    0.933845    1.070469    0.991
```

	Victim 1	Victim 2	X
rcont	1.000	5.617	0.829
degradation	$10^{7.24e-09}$	$10^{3.12e-20}$	$10^{4.32e-07}$

Locus Ajustments for each locus				
D3	vWA	D16	D2	D8
0.913	0.934	1.070	0.992	1.047
D21	D18	D19	TH01	FGA
0.837	0.873	0.954	0.971	0.909