Manual for *hierarchicalBoosting* package

Marc Pybus, Pierre Luisi, Giovanni Dall'Olio, Manu Uzkudun,
Hafid Laayouni, Jaume Bertranpetit and Johannes Engelken *

April 27, 2015

This manual explains how to use the 'Hierarchical Boosting' method described in the manuscript entitled "A Machine-Learning Framework to Detect and Classify Hard Selective Sweeps in Human Populations". Check `http://hsb.upf.edu/` for more information on citation.

1 Principle

The basic idea of the framework provided in this package is described in detail in the manuscript mentioned above, please read it before using it. The manuscript describes a framework to analyze and classify positive selection in genomic regions using polymorphism data. The novelty of the approach lies in the way the selection analysis is performed. Instead of addressing the inference of positive selection as a binary outcome (selection vs. non-selection), it assumes that there are different selective scenarios for which different selection tests show different power to detect. Under this assumption, it is possible to use this disparity in sensitivity towards different selection regimes to classify hypothetical selective sweeps into further properties, such as sweep age, completeness or intensity.

Our package is based on a machine-learning classification algorithm called *boosting* (from the *mboost* package - Buehmann and Hothorn 2007). *Boosting* is a supervised algorithm that estimates linear regressions (we call it *boosting* functions) of input variables (summary statistics or selection tests) to maximize the differences between two competing scenarios (e.g. selection vs. neutrality). Our method sequentially applies different boosting functions (as used by Lin *et al.* 2011) into a hierarchical classification scheme to optimally classify genomic regions into different selection regimes. The framework implemented here relies on the results of several selection test that need to be previously computed on simulated and empirical data. We do not provide the software needed to estimate selection tests or run the simulations.

New methods to detect selection are published every year improving the capabilities of previous ones or expanding the range of selection scenarios detected (hard and soft sweeps or balancing selection). We advise to use as many different selection tests as possible in the algorithm training process. High correlation (e.g. r2 $>$ 0.8) between summary statistics or selection tests should be avoided since it will prevent coefficient convergence of the implemented boosting algorithm (as explained in the manuscript).

During the training process, the algorithm is fed with a table containing selection tests scores or summary statistics (as columns) for the different selection scenarios to be classified (as rows). The more replicates are used in the training proces, the better. Table 1 shows an example of such input table.

*Institut de Biologia Evolutiva (UPF-CSIC), Universitat Pompeu Fabra, Barcelona, Spain. Contact: marc.pybus@upf.edu / jaume.bertranpetit@upf.edu

scenario	CLR	iHS	FayWuH	TajimasD	FuLiD	XPEHH
neutral	0.2789	0.4424	-15.6810	-0.5916	0.6760	0.6449
neutral	0.7685	0.5439	2.2182	-0.2074	0.6502	-0.7827
...	...	...	...	...	...	...
ancient_complete	3.3130	1.1149	0.4764	2.23085	-3.5784	3.1230
ancient_complete	11.2205	1.6295	1.3220	-2.4338	-3.8018	4.0536
...	...	...	...	...	...	...
recent_incomplete	0.4207	1.8648	-21.6999	-1.7474	-0.44682	6.0771
recent_incomplete	0.5756	1.3418	-24.5180	-1.5016	-0.05735	8.3427

Table 1. Training table example.

2 Usage

Several steps need to be followed to properly use this package. This is a brief description of the typical selection detection and classification procedure:

1. *Decide which selection tests to use:* depending on the type and properties of the data to analyze, different selection test should be used. Some tests allow the use of missing data, others need complete genotyping data. Some test may show sensitivity to many types of selection while others only to very specific ones. You should think which type of selection you are trying to detect (or discard) and choose your selection tests accordingly.

2. *Run simulations for the training process:* simulations should be as similar as possible to the empirical dataset you want to analyze. You should use state-of-the-art simulation software allowing for complex demographic and genomic models. Simulators should recreate different selective regimes (positive, negative, balancing selection) at different timeframes and populations. We recommend using one of the following simulation packages: SFS_CODE (Hernandez 2008), SLiM (Messer 2013), msms (Ewing and Hermisson 2010) or cosi (Schaffner *et al.* 2005). Simulated data should mimic real genomic regions and match sample sizes and possible site frequency spectrum bias of your reference dataset. Specific recombination maps for the analyzed genomic regions should also be considered.

3. *Apply your selection tests to the simulated and empirical data:* selection tests usually come in different flavors: some are SNP-based, others are window-based, some contain several internal parameters that need to be tuned according to your data, others are applied as provided. Nonetheless, at the end you want to obtain a unique score per test and per region so they can be properly combined in a linear regression. We advice to use some sort of summary statistics (mean, maximum, minimum) to unify your input variables in a given genomic region (empirical or simulated).

4. *Define the classification tree in which data will be sequentially classified:* you need to envision which classification structure will provide the best results. You can also try several configurations and pick the best performing one. One good strategy is to first distinguish neutral from selection scenarios, and in the following steps, classify the selection scenarios according to common features, such as age, sweep completeness or intensity. The power of your selection tests towards specific selective scenario should guide you on the structure of the classification tree. Remember that each node is a specifically trained *boosting* algorithm that maximizes different between two competing scenarios (Figure 1).

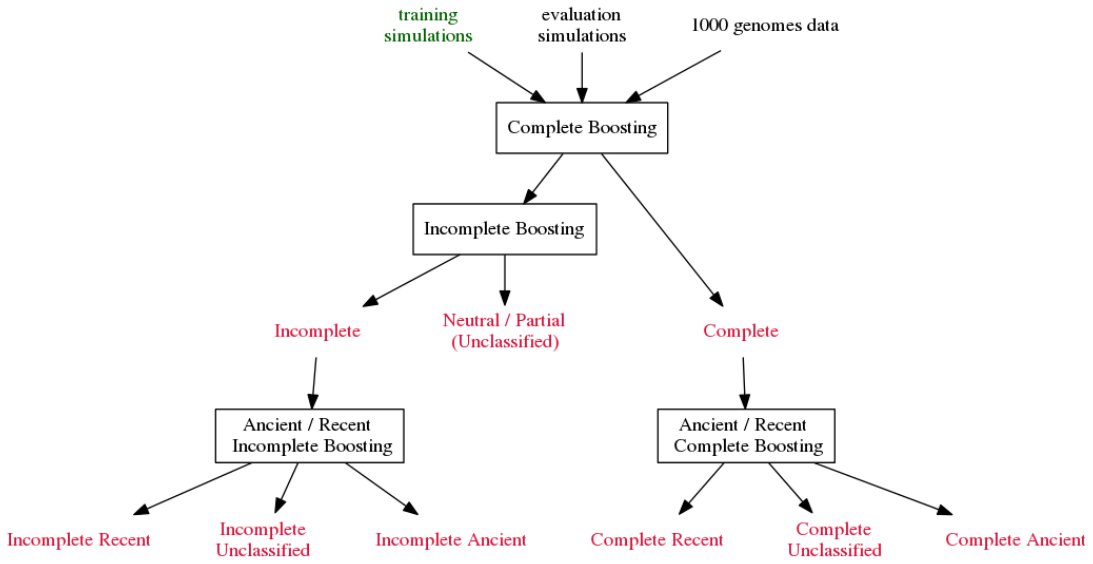

Figure 1. *hierarchicalBoosting* classification tree structure (from the manuscript).

5. *Generate validation plots and interpret them:* we have implemented a function that generates three validation plots for each trained *boosting* function. First validation plot is coefficient convergence plot (Figure 2). Here you must check that coefficients for your input variables reach convergence within the *boosting* iterative process. Lack of convergence may appear due to high correlation between input variables or low number of *boosting* iterations. Correct this when convergence is not reached with the default parameters. A second validation plot is standardized coefficient plot (Figure 3). This plot shows the standardized coefficients for each *boosting* function, which can give you an insight about the importance of each selection test to uncover a specific selection scenario. The amplitude of the boxplot reflects the robustness of the estimated *boosting* function (generated by the boostrapping process). To reduce variance in the coefficient estimation you can increase the number of replicates for each scenario. And finally, the third validation plot informs about the distribution of your *boosting* scores in scenario A and scenario B, along with the significance threshold used (Figure 4). In this plot one can check how good is the estimated *boosting* function in distiguishing between the two competing scenarios, as well as which significance thresholds are more appropiate for a given *boosting* function.

6. *Apply the trained hierarchicalBoosting tree to your empirical data:* once you have your *boosting* functions trained, validated and embedded in the classification tree (we call it a *hierarchicalBoosting* object), you can use it to scan your empirical data to detect and classify selection signals.

3 Example

A full example is shown below. In this example the training data comes from the original manuscript (although it is not the exact same dataset). The data consists in 5 different selection scenarios generated using the *cosi* package (Schaffner *et al.* 2005). For each scenario 100

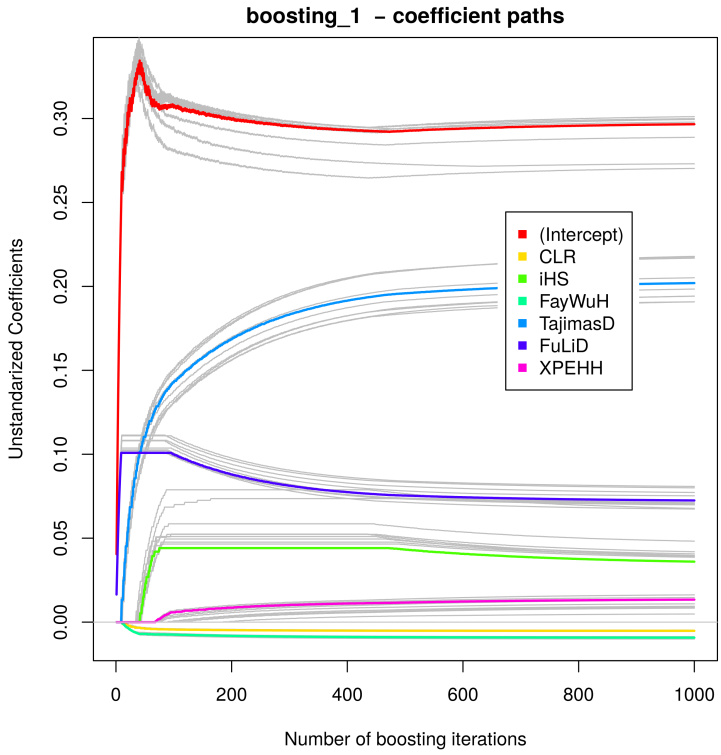

Figure 4. Coefficient convergence plot for the *boosting* function #1 from the example.

replicates are used. The simulated data mimics some population genetics features shown by the CEU population from The 1000 Genomes Project - Phase I dataset (The 1000 Genomes Project Consortium 2012). The selection scenarios simulated are: neutral, recent incomplete selection, recent complete selection, ancient incomplete selection and ancient complete selection. *Complete* and *incomplete* scenarios refer to the frequency of the allele under selection at the time of sampling: *incomplete* refers to selective sweeps with a final allele frequency of 0.8 (s = 0.0147), and *complete* specify selective sweeps that already reached fixation (s = 0.0257). *Recent* and *ancient* scenarios indicate when those selective sweeps happened, being *recent* a sweep ending at the moment of sampling, and *ancient* a sweep that ended 30,000 years ago (generation time = 25 years).

The selection tests used here are the following: CLR (Nielsen *et al.* 2005), iHS (Voight *et al.* 2006), Fay & Wu's H (Fay and Wu 2000), Tajima's D (Tajima 1989), Fu & Li's D (Fu and Li 1993), and XPEHH (Sabeti *et al.* 2002). Those tests were run under test-specific internal parameters (window size, MAF filtering,...) and then summarized in a common window size (using minimum or maximum, or mean) as described in detail in the manuscript. A window size of 25 Kbp was used to unify the selection tests.

The obtained data (selection tests scores for the simulated selection scenarios) was merged into a table that can be found in "exdata" directory in the package. This table is used as training dataset for the *train.hierarchicalBoosting* function.

A similar table containing selection tests scores for a real genomic region around the gene SLC45A2 (known to be under selection in CEU population) can be found in the same folder. We

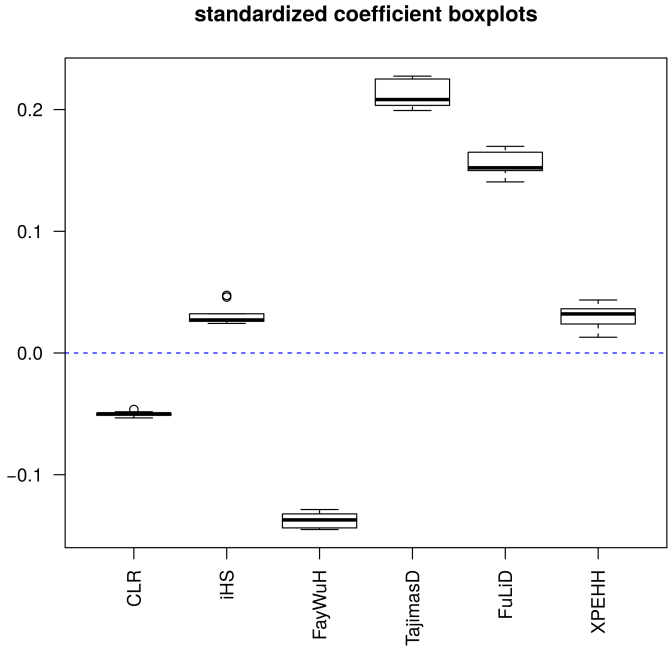

Figure 4. Standardized coefficient plot for the *boosting* function #1 from the example.

use this dataset as an example of empirical genomic region to be analyzed with the *apply.hierarchicalBoosting* function once the training step is finished. This dataset was generated by applying the same selection tests and the same summarizing approach used to create the training dataset.

	input_data	scenario_A	scenario_B	threshold_A	threshold_B
boosting_1	original	recent_complete, ancient_complete	neutral,recent_incomplete, ancient_incomplete	FALSE	0.01
boosting_2	boosting_1:scenario_B	neutral	recent_incomplete, ancient_incomplete	0.99	FALSE
boosting_3	boosting_1:scenario_A	recent_complete	ancient_complete	0.95	0.05
boosting_4	boosting_2:scenario_B	recent_incomplete	ancient_incomplete	0.95	0.05

Table 2. Configuration table for *hierarchicalBoosting*.

A configuration table containing the parameters to apply the *hierarchicalBoosting* algorithm must be created (Table 2). It describes the relation between the inputs and outputs of each *boosting* function (Figure 1), the competing scenarios used, and the significance thresholds needed to classify the different selection scenarios. Thus, to run a given *boosting* function one must define the following parameters:

1. *input_data*: the first *boosting* function must use as input dataset the original training table. This way, the following *boosting* functions can use the outputs from boosting #1 (refered as scenario A and scenario B) to further classify the data. When two thresholds are given to a *boosting* function a third classification outcome can be produced (scenario C). Figure 2 shows the score distribution of a typical *boosting* function in two competing scenarios with two significance thresholds used.

2. *scenario_A* and *scenario_B*: here one must describe the competing scenarios to be used at each *boosting* function. Scenarios names should be present in the training dataset and be comma-separated (as shown in Table 2).

3. *significance thresholds for each scenario*: values for the significance thresholds (between 0 and 1) for scenario A and scenario B. This allows to classify empirical data into the two or three possible outcomes. When only one threshold is used (to discard neutrality, for example), the other one must be set as FALSE.

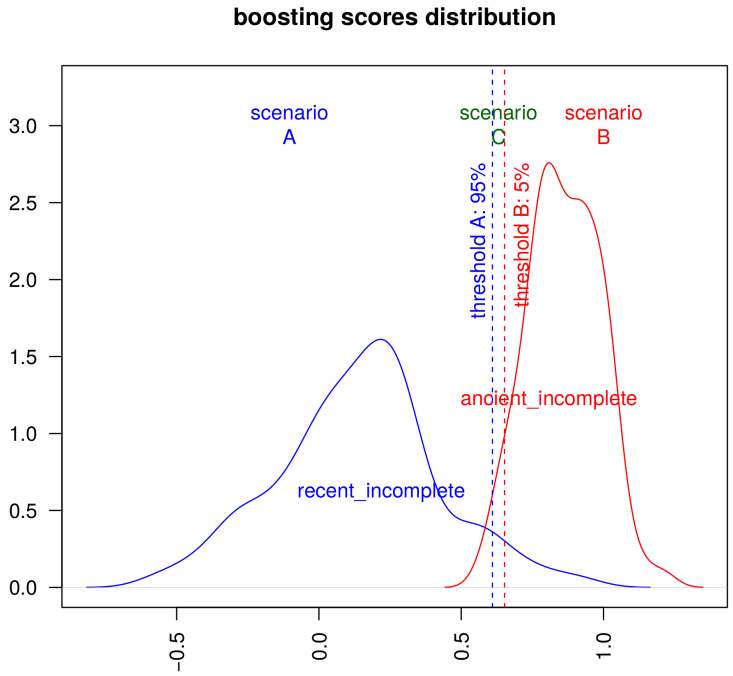

Figure 4. Distribution plot for the *boosting* function #4 (with two thresholds) from the example.

Finally, three more parameters should be set. First, the number of boosting iterations for the *mboost* algorithm, which is set at 1000 as default. And second, the number of bootstrapping iterations (*bootstrap_iterations*) performed by the method at each *boosting* function estimation plus the sample size used (*bootstrap_sampling* - as the fraction of the total number of replicates per scenario) at each bootstrap iteration.

Thus, if one wants to follow the steps used in the manuscript to classify empirical data into the 5 possible outcomes (scenarios), the following commands should be used:

```
> library("hierarchicalBoosting")
```

Load input dataset:

```
> input_path <- system.file("exdata/training_data.txt",
+         package="hierarchicalBoosting")
> input <- read.table(input_path, header=T, stringsAsFactors=F)
```

Define individual *boosting* parameters:

```
> boosting_1 <- c()
> boosting_1["input_data"] <- "original"
> boosting_1["scenario_A"] <- "recent_complete,ancient_complete"
> boosting_1["scenario_B"] <- "neutral,recent_incomplete,ancient_incomplete"
> boosting_1["threshold_scenario_A"] <- FALSE
> boosting_1["threshold_scenario_B"] <- 0.01

> boosting_2 <- c()
> boosting_2["input_data"] <- "boosting_1:scenario_B"
> boosting_2["scenario_A"] <- "neutral"
> boosting_2["scenario_B"] <- "recent_incomplete,ancient_incomplete"
> boosting_2["threshold_scenario_A"] <- 0.99
> boosting_2["threshold_scenario_B"] <- FALSE

> boosting_3 <- c()
> boosting_3["input_data"] <- "boosting_1:scenario_A"
> boosting_3["scenario_A"] <- "recent_complete"
> boosting_3["scenario_B"] <- "ancient_complete"
> boosting_3["threshold_scenario_A"] <- 0.95
> boosting_3["threshold_scenario_B"] <- 0.05

> boosting_4 <- c()
> boosting_4["input_data"] <- "boosting_2:scenario_B"
> boosting_4["scenario_A"] <- "recent_incomplete"
> boosting_4["scenario_B"] <- "ancient_incomplete"
> boosting_4["threshold_scenario_A"] <- 0.95
> boosting_4["threshold_scenario_B"] <- 0.05
```

Create configuration table:

```
> config_table <- data.frame(rbind(boosting_1, boosting_2, boosting_3,
+         boosting_4), stringsAsFactors=F)
```

Define bootstrapping parameters:

```
> bootstrap_iterations <- 20
> bootstrap_sampling <- 0.9
```

Check compatibility:

```
> check.HierarchicalBoosting(config_table, input)

checking scenarios names...ok
checking significance thresholds...ok
checking classification tree structure...ok
```

Create *hierarchicalBoosting* object:

```
> hierarchicalBoosting <- train.HierarchicalBoosting(input, config_table,
+         bootstrap_iterations, bootstrap_sampling)

Training Boosting #1

recent_complete/ancient_complete vs neutral/recent_incomplete/ancient_incomplete
input data: original dataset
number of replicates    => scenario A: 199         scenario B: 300
significance thresholds => scenario A: FALSE         scenario B: 0.01
boostrapping.....................done

Training Boosting #2

neutral vs recent_incomplete/ancient_incomplete
input data: scenario_B from boosting_1
number of replicates    => scenario A: 100         scenario B: 200
significance thresholds => scenario A: 0.99         scenario B: FALSE
boostrapping.....................done

Training Boosting #3

recent_complete vs ancient_complete
input data: scenario_A from boosting_1
number of replicates    => scenario A: 99         scenario B: 100
significance thresholds => scenario A: 0.95         scenario B: 0.05
boostrapping.....................done

Training Boosting #4

recent_incomplete vs ancient_incomplete
input data: scenario_B from boosting_2
number of replicates    => scenario A: 100         scenario B: 100
significance thresholds => scenario A: 0.95         scenario B: 0.05
boostrapping.....................done
```

Create validation plots:

```
> plot.HierarchicalBoosting(hierarchicalBoosting, config_table, input)

Generating validation plots...
   - boosting_1.plots.pdf
   - boosting_2.plots.pdf
   - boosting_3.plots.pdf
   - boosting_4.plots.pdf
```

Load empirical dataset (selection tests run at SLC45A2 genomic region in CEU):

```
> empirical_path <- system.file("exdata/SLC45A2_CEU_sweep.txt",
+          package="hierarchicalBoosting")
> empirical <- read.table(empirical_path, header=T, stringsAsFactors=F)
```

Check compatibility:

```
> check.HierarchicalBoosting(config_table, input,
+         hierarchicalBoosting=hierarchicalBoosting)

checking scenarios names...ok
checking significance thresholds...ok
checking classification tree structure...ok
checking compatibility: input data and hierarchicalBoosting object...ok
```

Apply *hierarchicalBoosting* object to empirical dataset:

```
> hierarchicalBoosting_results <- apply.HierarchicalBoosting(empirical,
+         config_table, hierarchicalBoosting)

Applying hierarchicalBoosting...
 1 2 3 4
Classifying data...
```

Summarize hierarchicalBoosting results:

```
> classification <- summarize.HierarchicalBoosting(hierarchicalBoosting_results)

Summarizing classification...

> print(classification)

   chromosome    start      end     classification
1           5 33450000 33475000            neutral
2           5 33475000 33500000            neutral
3           5 33500000 33525000            neutral
4           5 33525000 33550000            neutral
5           5 33550000 33575000            neutral
6           5 33575000 33600000            neutral
7           5 33600000 33625000            neutral
8           5 33625000 33650000            neutral
9           5 33650000 33675000            neutral
10          5 33675000 33700000            neutral
11          5 33700000 33725000            neutral
12          5 33725000 33750000            neutral
13          5 33750000 33775000            neutral
14          5 33775000 33800000            neutral
15          5 33800000 33825000            neutral
16          5 33825000 33850000            neutral
17          5 33850000 33875000            neutral
18          5 33875000 33900000            neutral
19          5 33900000 33925000 ancient_incomplete
20          5 33925000 33950000 ancient_incomplete
21          5 33950000 33975000 ancient_incomplete
22          5 33975000 34000000            neutral
23          5 34000000 34025000            neutral
24          5 34025000 34050000            neutral
```

```
25          5 34050000 34075000           neutral
26          5 34075000 34100000           neutral
27          5 34100000 34125000           neutral
28          5 34125000 34150000           neutral
29          5 34150000 34175000           neutral
30          5 34175000 34200000              <NA>
31          5 34200000 34225000              <NA>
32          5 34225000 34250000              <NA>
33          5 34250000 34275000           neutral
34          5 34275000 34300000           neutral
35          5 34300000 34325000           neutral
36          5 34325000 34350000           neutral
37          5 34350000 34375000           neutral
38          5 34375000 34400000           neutral
```

4 References

K. Lin, H. Li, C. Schloetterer and A. Futschik (2011) Distinguishing positive selection from neutral evolution: boosting the performance of summary statistics. *Genetics* 187: 229-244.

P. Buehmann and T. Hothorn (2007) Boosting algorithms: regularization, prediction, and model fitting. *Stat. Sci.* 22: 477-505.

PW. Messer (2013) SLiM: Simulating Evolution with Selection and Linkage. *Genetics*. 194:1037

RD. Hernandez (2008) A flexible forward simulator for populations subject to selection and demography. *Bioinformatics*, 24:2786-2787

G. Ewing and J. Hermisson (2010) MSMS: A coalescent simulation program including recombination, demographic structure, and selection at a single locus. *Bioinformatics* 26:2064-2065.

SF. Schaffner, C. Foo, S. Gabriel, D. Reich, MJ. Daly, D. Altshuler (2005) Calibrating a coalescent simulation of human genome sequence variation. *Genome Res.* 15(11): 1576–1583.

The 1000 Genomes Project Consortium (2012) An integrated map of genetic variation from 1,092 human genomes. *Nature* 491: 56–65.

R. Nielsen, S. Williamson, Y. Kim, MJ. Hubisz, AG. Clark et al. (2005) Genomic scans for selective sweeps using SNP data. *Genome Res.* 15: 1566–1575

BF. Voight, S. Kudaravallis, X. Wen, JK. Pritchard (2006) A map of recent positive selection in the human genome. *PLoS Biol.* 4: e72.

J.C. Fay and C.I. Wu (2000) Hitchhiking under positive Darwinian selection. *Genetics* 155:1405-1413.

F. Tajima(1989) Statistical method for testing the nuetral mutation hypothesis by DNA polymorphism. *Genetics* 123:585-595.

YX. Fu and WH. Li (1993) Statistical tests of neutrality of mutations. *Genetics* 133: 693–709.

P.C. Sabeti, D.E. Reich, J.M. Higgins, H.Z.P. Levine, D.J. Richter et al.(2002) Detecting recent positive selection in the human genome from haplotype structure. *Nature* 419:832-837.