Supporting Information
How to choose sets of ancestry informative markers: A supervised feature selection approach

Peter Pfaffelhuber, Franziska Grundner-Culemann, Veronika Lipphardt, Franz Baumdicker

August 21, 2019

Abstract

We describe some details for the analysis carried out in [6]. In particular, we give details to the R-scripts and simulations used in this study.

1 Prerequisites

For data analysis as well as for our simulation studies, we rely on scripts in the statistical language R [7].

1. The simulation studies are performed using msprime as implemented by [3] (and most easily installed via `pip3 install msprime`), which is a fast coalescent simulator. In particular, structured populations (with varying population sizes etc) can be simulated. We are using the python-interface of msprime. See `https://msprime.readthedocs.io/en/stable/` for more information.

2. For multicore-computing, we require the R-package `parallel`.

3. Since, both data from the 1000 genomes project [1], which is analysed here, and the coalescent simulations, come in vcf-format (which stands for *variant-call-format*), we require the R-package `vcfR`; see [4].

4. For some steps in the analysis, we use `vcftools` [1] and `bcftools` [5].

In addition, data from the 1000 genomes project (phase 3) was downloaded, as well as information on the sampling locations. For this, we used
`wget ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502/ALL.*`
`wget ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502/integrated_call_samples_v3.20130502.ALL.panel`. The latter file was renamed `1000G_SampleListWithLocations.txt`, and the first row (the header) was removed.

2 File structure

We are using the following filestructure:

- `code`: Contains the functions `tools.r` as described in Section 5, as well as commands for using the coalescent simulations, as described in Section 4;
- `data/1000G`: Contains the data of the 1000 genomes project (see the `wget`-commands from above);
- `data/sim`: Contains the simulations; subfolders are `ooa` for the Out-Of-Africa model of human evolution, and `si` for the symmetric island model.
- `data/AIMsets`: Contains the AIMsets and further analysis we find using our method;
- `doc`: Contains the documentation.

We note that several files `pipeline*.r` exist with all commands for obtaining interesting results, as described below. In particular, the master directory contains a file `pipeline_example.r`, which serves as a starting point to check the installation and for a first impression. We briefly describe it next.

3 A small example from `pipeline_example.r`

Using the example datafile `data/sim/ooa/ooa_chromosome_1_example.vcf.gz`, together with the information contained in `ooa_SampleListWithLocations.txt` on sampling locations, all steps as described in Section 6 are carried out. The main output is in the following files:

- `AIMs`: A list of AIMs is produced here. Note that the identifiers of SNPs in the simulated dataset which is used here, are of the form `chr_pos`.
- `AIMs.vcf.gz`: Here, the data is stored for all AIMs in `AIMs` and all 2400 individuals from the dataset (800 diploids from the simulated version of Africa, Europe and Asia each).

In addition, `nss.tab` stores the numbers of segregating sites in all input files. Since there is only one input file in this example, just a single number is stored. `ooa_SampleListWithLocations.txt` is generated from the datafile and contains the names of individuals and their sampling locations, `training_ooa_SampleListWithLocations.txt` and `test_ooa_SampleListWithLocations.txt` are the subsets of training and test data. `predictions.csv` store the posterior probabilities for each individual to come from any of the continents. `classifications.tab` stores for each individual which deme has the maximal posteriori probability. `sampleAIMs.rds` is an R-file, which stores an intermediate step. Some more files are generated within the folder `tmp`.

4 Coalescent simulations

Once `msprime` is installed, the python interface can be used. In `code/simulate4biogeo.py`, we collected the commands needed to run `msprime`. A file `pipeline_simulated_ooa_data.r` used for analyzing the simulations is described in Section 7.

We ran two sets of simulation: (i) a symmetric island model; (ii) the model for human population history from [2], briefly called the out-of-Africa model (ooa) in the sequel. Both cases come with the parameters:

- `num_chromosomes`: This is the number of chromosomes which are simulated. Here, chromosomes are treated as (stochastically) independent copies of the same coalescent process.
- `random_seed`: In order to have reproducible results, a seed can (but does not have to) be set.
- `ploidy`: Since the output of `msprime` usually only treats haploid samples, they have to be combined in order to form diploids. So, one might want to set `ploidy = 2` for humans.

In order to have the same pipleine for simulated and real data, we use the function `write_vcfgz` in order to obtain the simulations and real data in the same format.

4.1 Symmetric island model

Consider a population structure given by 5 continents or islands with constant bidirectional and equal migration rate m between all pairs of continents. For each migration rate $m = 1, 2, \ldots, 8$ (which stands for the expected number of haploid individuals migrating between any (fixed) pair of continents. We performed 10 independent simulations, where each dataset consists of 800 individuals from each of the five continents. Simulations are generated with the command:

`python3 code/simulate4biogeo.py si` *m* `random_seed num_chromosomes`

4.2 Out of africa model

The human demographic history as inferred by Gutenkunst et al. [2] was used to mimic the human population structure. Adopting the model for human genetic history from [2] (see their Table 1 and Figure 2), we have populations 1, 2 and 3, modeling Africa/YRI, Europe/CEU and Asia/CHB, respectively. We simulated 10 independent datasets under this model with a constant mutation rate of 0.125 and a recombination rate of 1.25. For each simulation run 1600 (haploid) individuals for each of the three continents (Africa, Europe and Asia) were generated. Simulations are generated with the command:

```
python3 code/simulate4biogeo.py ooa seed 20
```

5 R-functions

Here is a short description of used variables, all of which are used in the functions described below:

- `filenameInds`: A tab-delimited file, read by `read.table`, which is required to have the following columns 1st: identifier for an individual; 3rd: population of the individual; the file must not have a header and then the same number of lines as `filenameData`; There are two more variables, `filenameIndsTraining` and `filenameIndsTest`, which have the same structure, and contain training and test sets;
- `inds`: A matrix with two columns; the first column contains the individual identifiers, the second contains the sampling location;
- `samplesInDeme`: A vector of integers containing the number of individuals for all demes;
- `nss`: A vector of integers containing the number of segregating sites for all files;
- `snplist`: A file containing SNP-identifiers. Usually used for extracting these SNPs using `vcftools`;
- `sample`: A matrix of `nss` columns (the SNPs) and `sum(samplesInDeme)` rows (the individuals). Entries are either 0/1 (if `diploid = FALSE`) or 0/1/2 (if `diploid = TRUE`);
- `samLoc`: A vector of length `nrows(inds)`, indicating which sample comes from which deme;
- `demes = unique(samLoc)` are all sampling locations;
- `freqs` (Usually obtained through `getfreqs(sample, samLoc, diploid)`): A matrix with `length(demes)` rows and `nss` columns, containing the frequencies of the derived allele in the sample;
- `error.type`: The type of error by which the classification method is evaluated; either `misclassification` or `logloss`;
- `filenameData`: The name of the file of the data, which must be a `...vcf.gz` file. Note that `filenameData` must not contain this suffix. This file must only contain bi-allelic markers and no missing data;
- `filenameDataVCFGZ`: Same as `filenameData`, but including the suffix `.vcf.gz`;
- `method`: The method used for classification; either `naiveBayes` (most often) or `informativeness`; other methods, such as logistic regression, are not implemented;

Here is a list of functions contained in `tools.r`:

- `appTestError(sample.test, freqs.training, samplesInDeme.training, samLoc.test, method = "naiveBayes", error.type="misclassification", diploid = FALSE)`
  For `error.type`, which is either `misclassification` of `logloss`, we give the error when classifying `sample.test`, and `freqs.training` is used to obtain the predictions;

- `cut.VCF(filenamesDataVCFGZ, stepsize, removeUnnamed = TRUE)`
  For handling of large files and limited memory, it might be beneficial to cut it in smaller pieces, containing only a subset of SNPs. Here, `stepsize` determines the number of SNPs in the smaller files. By default, SNPs without an identifier are excluded. The resulting files are `tmp/chr1_segment1.recode.vcf.gz` etc.;

- `extract.and.merge(filenamesDataVCFGZ, snplist, outfile)`
  Here, `filenamesDataVCFGZ` is a vector of filenames and `snplist` is a file containing a list of SNPs (e.g. possible AIMs). The result is a file `outfile` of `vcf.gz`-format, which stores exactly the sample information for SNPs in `snplist`;

- `find.AIMs.fromRDS(filenamesRDS, sampleAIMs = NULL, inds, numAIM, method = "naiveBayes", error.type="misclassification", diploid = FALSE, outfile = "AIMs")`
  `filenamesRDS` is a vector of rds-files (i.e. R-files containing a single R-object) containing the samples, i.e. all files in `filenamesRDS` contain the same individuals, but different SNPs. `numAIM` is the number of AIMs which should be found. It is possible to start already with a few AIMs, stored in `sampleAIMs`. During runtime, when a new AIM is found, it is appended to the file `AIMs`, and `sampleAIMs.rds` stores the data at the AIM sets;

- `find.nextAIM.fromRDS(filenameRDS, sampleAIMs, inds, method, error.type, diploid)`
  This function is called repeatedly from `find.AIMs.fromRDS` in order to find a single new AIM. This is called stepwise forwards search of AIMs;

- `find.SNPs.inVCFGZ(filenamesDataVCFGZ, snplist)`
  Given the SNPs in the file `snplist`, determine which of them are present in the set of files filenamesDataVCFGZ;

- `getData(filenameDataVCFGZ, skip=0, windowSize=-1, inds, snps = TRUE, diploid = FALSE)`
  Using the function `read.vcfR` from the package `vcfR`, the data is read. Using `skip` and `windowSize`, only a subset of SNPs from `filenameDataVCFGZ` can be obtained; the value is a list containing `sample` and `samLoc`;

- `getfreqs(sample, samLoc, diploid = FALSE)`
  From `sample` and `samLoc`, we know which frequencies are observed in which demes. The result is given here, a matrix with `length(demes)` rows and `nss` columns;

- `getfreqs.from.vcf(filenameData, filenameInds, snplist=FALSE)`
  For large files, `getfreqs` from above might have memory issues. This funtion uses `vcftools` with its `freqs2`-option in order to write `.frq`-files for frequencies in all demes. These are subsequenty extracted and reported in the file `paste(filenameData, "_freqs")`;

- `getIdentifiers(filenameDataVCFGZ)` Returns the SNP-identifiers used in file `filenameDataVCFGZ`;

- `getNss(filenameDataVCFGZ)` Returns the number of segregating sites in the file `filenameDataVCFGZ`;

- `getTestandtrainingset(filenameInds, filenameIndsTraining, filenameIndsTest, size)`
  Using a random number generator, the individuals are split into a training and a test set. `filenameIndsTraining` and `filenameIndsTest` are the filenames where the individuals for these two sets are stored; `size` is a vector containing the sizes of the testset in all populations;

- `informativeness.global(freqs)`
  A vector of length `nss`. This function computes the informativeness for all SNPs in `freqs`. This function needs`xlogx2(x)` and `xlogx(x)`;

- `logloss.error(prediction, true.value)`
  The logarithmic loss;

- `log.prediction.naiveBayes(sample.test, freqs.training, samplesInDeme.training, diploid)` and `prediction.naiveBayes(sample.test, freqs.training, samplesInDeme.training, diploid)`
  Based on `freqs.training`, probabilities are computed which give the chances that some individual from `sample.test` comes from one of the demes. Since this is based on the naive Bayes approach, we also need number how many samples were taken from each deme (`samplesInDeme.training`); We use here `prediction.naiveBayes = exp(log.prediction.naiveBayes)` for numerical stability;

- `misclassification.error(prediction, true.value)`
  The misclassification error;

- `setcores(cores=1)`:
  Determines the number of cores used for classification (in `step6` below); this requires the `R`-package `parallel`;

6 Analyzing data from the 1000 genomes project

We now describe the file `pipeline_1000G_AIMs_noAMR.r` in detail. All other files called `pipeline*.r` work similarly. After importing necessary packages, we set the number of cores

using `setcores` to a maximum number on our machine. These will be used in parallel to find AIMs in `step6` below. Then, some parameters need to be set:
The variable `testindsperdeme = 100` indicates that 100 samples per deme/population/continent are taken. The rest of the data is used as training set. Since we ignore all samples in population `AMR`, we only have `demes=4`. The `informativenessBound = 0.9` is used as a presection step for the AIMs. This number means that only SNPs which have an informativeness in the top 10% of al observed informativenesses qualify to be chosen as AIM. Then, `stepsize = 10000` is used when chopping the whole data to smaller files. We found this number to work well with 64GB of RAM on a machine with 32 cores. Then, `numAIM = 30` will eventually produce a set of 30 AIMs. Finally, we use the standard setting, as also described in the main text: `method = "naiveBayes"`, `error.type="logloss"` (which is used to find the AIMs, although we finally report the misclassification error) and `diploid = TRUE`. The user might want to set as seed using something like `set.seed(31)` in order to have reproducable results. (This is only used for creating the training and test set of individuals. The filenames `filenameInds`, `filenameIndsTraining`, `filenameIndsTest` and the vector `filenameData` is created for future reference.

Then, the following steps are carried out (which can be in- or excluded depending on where the user wants to make progress):

- `prepare`: Based on the file `data/1000G/1000G_SampleListWithLocations.txt`, we exclude `AMR`s and write the resulting file in the current directory. Test and training set are also written using `getTestandtrainingset`.
- `step0`: Using `vcftools`, we reduce dataset to bi-allelic SNPs, and remove indels. The resulting files are called `tmp/chr1_biallelic.recode.vcf.gz` etc.
- `step1`: Using the function `getfreqs.from.vcf`, we obtain frequencies for all SNPs in all populations for the training set.
- `step2`: Filter the most informative SNPs (i.e. SNPs with informativeness above the `informativenessBound` quantile) and write a new `vcf.gz`-File, denoted `tmp/chr1_informative.recode.vcf.gz` etc. In these files, we have all SNPs which are eligible to become AIMs.
- `step3`: For better handling, we cut the files in smaller pieces with `stepsize` SNPs each using the function `cut.VCF`. The resulting files are `tmp/chr1_segment1.recode.vcf.gz` etc.
- `step4`: Since they are needed in the remaining steps, we read off the number of informative sites for all chromosomes are write them into the file `nss`.
- `step5`: We use all files from `step3`, read in the data in the format we need, and store the result in an R-file (i.e. ending with `.rds`). The advantage is that these files are very quick to read in R.
- `step6`: The main (i.e. most time-consuming) step in the analysis is where we find the AIMs. Each of the rds-files is read, the best SNP is found in each file (i.e. the SNP which,

if included as AIM, has the smallest logloss-error), and we finally take the SNP with the smallest such error. We look stepwise for `numAIM` AIMs. However, each time a new AIM is found, it is added to the file `AIMs`, and an rds-file containing all AIMs ist stored in `sampleAIMs.rds`.

- `step7`: Finally, we have all AIMs and create (using `extract.and.merge`) a `vcf.gz`-file for all SNPs from `step6`. The resulting file is called AIMs.vcf.gz.
- `step8`: From `AIMs.vcf.gz`, we obtain for all individuals the posteriori probabilities to come from one of the four continents. The results are stored in `predictions.csv`. In addition, `classifications.tab` stores the continents with the maximal posteriori probability.

7 Analyzing simulated data from the Out-Of-Africa model

Since simulated data also comes in the `vcf.gz`-format, we can use the same pipeline as in Section 6. The main difference is that the data is already bi-allelic, but we have to simulate the data before we can begin:

- `simulate`: A multicore call is used for simulating the model as described in Section 4.2. The data is stored in `data/sim/ooa/tmp/`.
- `prepare`: Here, we known that the samples come in the order of the coalescent simulations. This means that in the data file, there are 800 diploids from AFR, EUR and ASI. The identifiers of the individuals are stored in `ooa_SampleListWithLocations.txt`. Test and training-set are obtained as in the analysis of the data from the 1000 gebomes.
- `step1` – `step 8` are carried out as in Section 6.

References

[1] Petr Danecek, Adam Auton, Goncalo Abecasis, Cornelis A. Albers, Eric Banks, Mark A. DePristo, Robert E. Handsaker, Gerton Lunter, Gabor T. Marth, Stephen T. Sherry, Gilean McVean, Richard Durbin, and 1000 Genomes Project Analysis Group. The variant call format and vcftools. *Bioinformatics (Oxford, England)*, 27(15):2156–2158, 2011.

[2] Gutenkunst, R. N., R. D. Hernandez, S. H. Williamson, and C. D. Bustamante (2009). Inferring the joint demographic history of multiple populations from multidimensional snp frequency data. *PLoS genetics 5*(10), e1000695.

[3] Jerome Kelleher, Alison M. Etheridge, and Gilean McVean. Efficient coalescent simulation and genealogical analysis for large sample sizes. *PLoS computational biology*, 12(5):e1004842, 2016.

[4] B. J. Knaus and N. J. Grünwald. Vcfr: an R package to manipulate and visualize vcf format data. *Mol. Ecol. Resour.*, 17(1):44–53, 2017.

[5] Heng Li. A statistical framework for SNP calling, mutation discovery, association mapping and population genetical parameter estimation from sequencing data. *Bioinformatics (Oxford, England)*, 27(21):2987–2993, 2011.

[6] P. Pfaffelhuber, F. Grundner-Culemann, V. Lipphardt, and F. Baumdicker. How to choose sets of ancestry informative markers: A supervised feature selection approach, 2019.

[7] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.