R package [assignPOP v1.1] tutorial

Alex Chen

2016-12-27

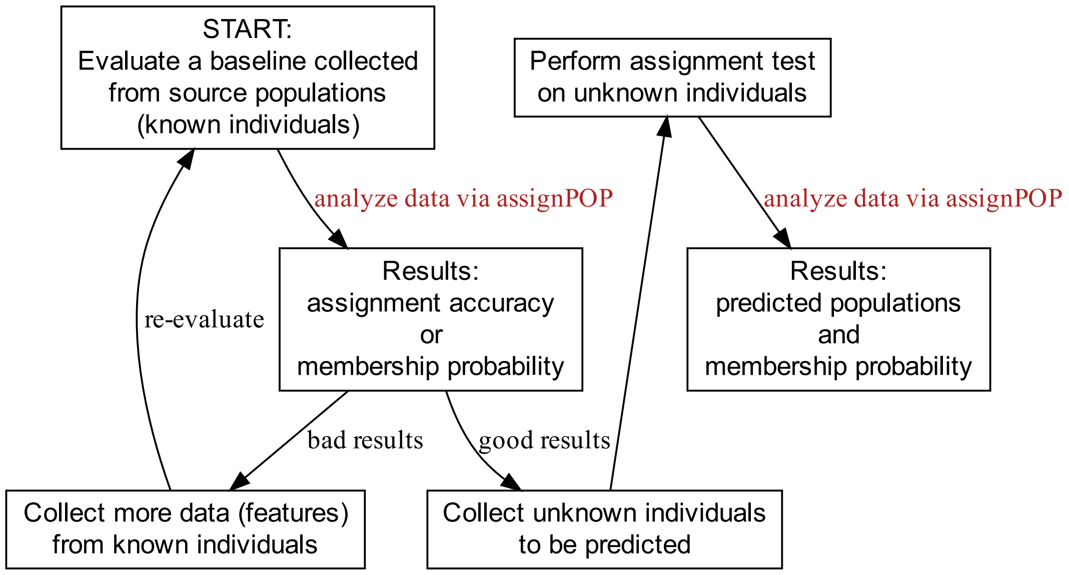
The assignPOP package helps perform population assignment using a machine learning framework. It employs supervised machine learning methods to evaluate the discriminatory power of your known data set, and is capable of analyzing **large genetic, non-genetic, or integrated (genetic plus non-genetic) data sets**. This framework is also designed for solving the upwardly biased issue that was discussed in previous molecular genetic studies (Anderson 2010[1]; Waples 2010[2]). Other features are listed below.

- Use principle component analysis (PCA) for dimensionality reduction (or data transformation)
- Use Monte-Carlo cross-validation to evaluate the variation of assignment accuracy
- Use *K*-fold cross-validation to estimate membership probability
- Allow to resample training individuals with various proportions or numbers
- Allow to resample training loci with various proportions either randomly or based on locus F_{ST} value
- Provide several machine learning classifiers, including LDA, SVM, naive Bayes, decision tree, and random forest to build tunable predictive models.
- Output results in publication-quality plots while being editable using ggplot2 library

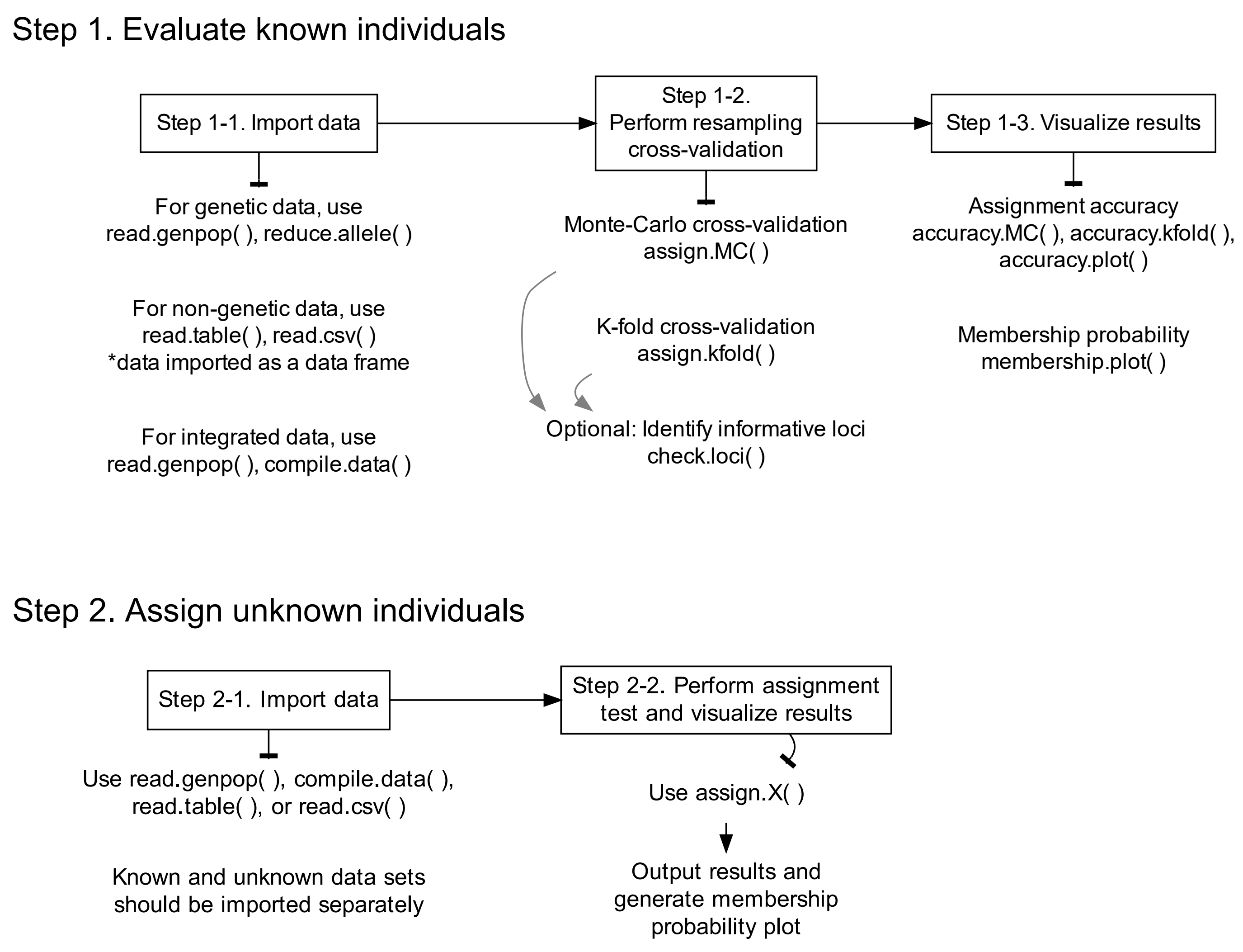
Another copy of this tutorial can be found at http://rpubs.com/alexkychen/assignPOP which includes a side menu for convenient browsing.

Conceptual framework

The ability to assign unknown individuals to source populations relies on a robust baseline collected from the source populations. Therefore, we should evaluate whether the baseline has high assignment accuracy on the known individuals themselves in advance of performing assignment on unknown individuals. The diagram below illustrates how the assignPOP package incorporates into the assignment framework.


START: Evaluate a baseline collected from source populations (known individuals)
Perform assignment test on unknown individuals
analyze data via assignPOP
analyze data via assignPOP
re-evaluate
Results: assignment accuracy or membership probability
Results: predicted populations and membership probability
bad results
good results
Collect more data (features) from known individuals
Collect unknown individuals to be predicted

Analytical workflow


Step 1. Evaluate known individuals
Step 1-1. Import data
Step 1-2. Perform resampling cross-validation
Step 1-3. Visualize results
For genetic data, use read.genpop( ), reduce.allele( )
For non-genetic data, use read.table( ), read.csv( ) *data imported as a data frame
For integrated data, use read.genpop( ), compile.data( )
Monte-Carlo cross-validation assign.MC( )
K-fold cross-validation assign.kfold( )
Optional: Identify informative loci check.loci( )
Assignment accuracy accuracy.MC( ), accuracy.kfold( ), accuracy.plot( )
Membership probability membership.plot( )
Step 2. Assign unknown individuals
Step 2-1. Import data
Step 2-2. Perform assignment test and visualize results
Use read.genpop( ), compile.data( ), read.table( ), or read.csv( )
Known and unknown data sets should be imported separately
Use assign.X( )
Output results and generate membership probability plot

Install and import assignPOP

To install the package from CRAN, type:

```r
install.packages("assignPOP")
```

Alternatively, you can install the package from the Github repository, but it requires you to install the [devtools] package first.

```r
install.packages("devtools")
library(devtools)
install_github("alexkychen/assignPOP")
```

To import the package to your R environment, type:

```
library(assignPOP)
```

Prepare and import data

Prepare and import genetic data of source populations

Currently, assignPOP accepts genetic data in GENEPOP format (Rousset 2008[3]), a commonly used format in molecular ecology studies. GENEPOP has two forms, and both forms are accepted by assignPOP. For details about GENEPOP format, please visit http://genepop.curtin.edu.au/help_input.html.

To import a GENEPOP file, use the function `read.genpop()` and provide a return object name (e.g., `genin`) as follows.

```
genin <- read.genpop( "simGenepop.txt", pop.names=c("pop_A","pop_B","pop_C") )
```

The example input file, *simGenepop.txt*, contains **three** simulated populations (or subpopulations) and each population has 30 individuals by 1,000 SNP loci. You can use the argument `pop.names` to name your populations in your GENEPOP (must follow the group order[4]) so that later you can customize assignment accuracy plots by specifying these names. If `pop.names` is not provided, pop.*NUMBER* (e.g., pop.1, pop.2) will be used for the population name.

After importing the file, you should see the following message printed in your R console. Use this information to double check your data was correctly imported.

```
## ############### assignPOP v1.1 ###############
##
## A GENEPOP format file was successfully imported!
##
## DataInfo: 90 obs. by 1000 loci (with 2000 var.)
## DataMatrix: 90 rows by 2001 columns
## Number of pop:3
## Number of inds (pop_A): 30
## Number of inds (pop_B): 30
## Number of inds (pop_C): 30
##
## Data output in a list comprising the following three elements:
## DataMatrix @ YOUR_LIST_NAME[[1]]
## IndividualNames @ YOUR_LIST_NAME[[2]]
## LocusNames @ YOUR_LIST_NAME[[3]]
```

Import genetic data of unknown individuals

When assigning unknown individuals (see Step 2 in Analytical workflow), you will first import a GENEPOP file that includes known individuals (for building predictive models), and then import another GENEPOP that includes unknown individuals. All unknown individuals in your GENEPOP should be saved in one single group, meaning that you will have only one ‘pop’ label in the file. Again, use `read.genpop()` to import the file and ignore the argument `pop.names`, but make sure you provide different object names for known and unknown data sets.

```
# Import a GENEPOP file containing unknown individuals
genin_unknown <- read.genpop( "simGenepopX.txt" )
```

Remove low variance loci (optional)

For a very large data file (e.g, tens of thousands of loci), you can choose to remove loci that have low variance[5] across the data set in advance of further analyses. The default of variance threshold is 0.95, meaning that any locus with a major allele occuring in over 95% of individuals across all populations will be removed from the data set.

To remove low variance loci, use the following function and provide a new object name (e.g., `genin_rd`) :

```
genin_rd <- reduce.allele(genin, p = 0.95)
```

After entering the code, R console should print the following message.

```
## New data matrix has created! :)
## New DataMatrix size: 90 rows by 1387 columns
## 614 columns (alleles) have been removed
## 693 loci remain
```

In our example, 307 low variance loci[6] were removed, leaving 693 loci in the new data set, which will be used in further analyses.

Concatenate genetic and non-genetic data

A novel feature of the package is that it can integrate genetic and non-genetic data for population assignment tests. After importing your GENEPOP file, you can use the function `compile.data()` to concatenate a non-genetic data set and the genetic matrix that was returned from either `read.genpop()` or `reduce.allele()`. Your non-genetic data should be saved in a .csv file (elements separated by commas) or table-like text file (elements separated by spaces) in which the first column must be sample IDs that match the IDs in your GENEPOP file, and the rest of columns are non-genetic data, whether it is numeric or categorical. If a sample ID exists in only one of the data sets, the individual will be ignored in the new integrated data.

To concatenate data, use the following function and provide a new object name (e.g., `comin`):

```
comin <- compile.data(genin_rd, "morphData.csv")
```

The above *morphData.csv* file has the same individuals as in *simGenepop.txt* and contains 4 numeric variables (morphormetric measurements) for each individual. After executing the function, it will prompt the following message and wait for your answer to verify the data type (numeric or categorical).

```
## Import a .CSV file.
## 4 additional variables detected.
## Checking variable data type...
## D1.2(integer) D2.3(integer) D3.4(integer) D1.4(integer)
## Are they correct? (enter Y/N):
```

If a variable is categorical, it will automatically convert it to new dummy variables using model.matrix() so that the categorical data can be quantified for further analyses. If your data type was incorrectly identified, simply enter **N** and then follow the prompted dialogue to change your data type. If your data type was correctly identified (entry is **Y** in this example), then it will print the following message.

```
## New data set created!!
## It has 90 observations by 1391 variables
## including 693 loci(1386 alleles) plus 4 additional variables(4 columns)
```

If you are evaluating a baseline using integrated data, the return object (e.g., `comin`) will be used in resampling cross-validation. On the other hand, if you are assigning unknown individuals, be sure to run the function `compile.data()` for known and unknown data sets, respectively, and provide different return object names. These two objects will be used in the assignment test (`assign.X()`).

Prepare and import non-genetic data

In addition to analyzing genetic and integrated data, you can perform assignment tests using only non-genetic data. This allows you to compare the results between data types. A non-genetic data set for baseline evaluation should include 1) sample IDs in the first column, 2) population label[7] in the last column, and 3) features in columns between the first and last column. The file should be saved in a .csv (comma dilimited) or table-like text file that can be read into R using `read.csv()` or `read.table()`. For example, if we are going to perform assignment test using the morphormetric data (*morphData.csv*) alone, we would need to add a population column after the last feature column. It can be done by either manually editing the file or adding a column after the file was imported (as shown below).

```
#Import morphormetric data containing sample IDs and features
morphdf <- read.csv( "morphData.csv", header=TRUE )
#Create a string vector for population label (repeat each name for 30 individuals)
pop_label <- c( rep("pop_A", 30), rep("pop_B", 30), rep("pop_C", 30) )
#Add the pop_label to the last column; 'morphdf_pop' is a data frame with population label in the
last column
morphdf_pop <- cbind(morphdf, pop_label)
```

A non-genetic data containing the population label could be used for baseline evaluation or as a baseline to predict unknown individuals. In other words, when performing an assignment test on unknown individuals using non-genetic data alone, the population column should be included in the known data set but not in the unknown data set.

Perform population assignment

To evaluate discriminatory power of a baseline, the package uses Monte-Carlo cross-validation (function `assign.MC()`) to estimate assignment accuracies over all populations and for each population, and uses *K*-fold cross-validation (function `assign.kfold()`) to estimate membership probabilities across all individuals. About Monte-Carlo and *K*-fold cross-validation

Monte-Carlo cross-validation for baseline evaluation

When performing Monte-Carlo cross-validation using genetic or integrated data, users can specify multiple proportions or multiple fixed numbers of individuals from each population to be used as training individuals (arg. `train.inds`) and multiple porportions of loci to be used as training loci (arg. `train.loci`). A set of training loci can be chosen either randomly (arg. `loci.sample="random"`) or based on locus F_{ST} estimated within training individuals (arg. `loci.sample="fst"`). Each combination of training individuals and loci can be resampled multiple times (arg. `iterations`). For example, the following code performs a total of 360 assignment tests, sampling 50%, 70% and 90% of individuals randomly from each population, using the highest 10%, 25%, 50% of F_{ST} loci as well as all loci for the training data. Each sample size combination was resampled 30 times.

```
assign.MC(genin_rd, train.inds=c(0.5, 0.7, 0.9), train.loci=c(0.1, 0.25, 0.5, 1),
          loci.sample="fst", iterations=30, dir="Result-folder/", model="svm" )
```

To use certain numbers, rather than proportions, of individuals from each population as training data, enter positive integers in the argument `train.inds`.

If you are analyzing non-genetic data alone, the arguments, `train.loci` and `loci.sample`, will be automatically ignored.

The results will be saved in a folder created via the argument `dir`. In the above example, a folder named “Result-folder” is created and results are saved in it. You must include a forward slash (/) at the end of your folder name.

In addition, you can use the argument `model` to select a classifier for prediction. In this case, the Support Vector Machine (SVM) classifier is used. Other SVM related arguments, `svm.kernel` and `svm.cost`, can be specified to fine tune the predictive model. See the e1071 package for details about SVM. Other classifiers, including LDA, naive Bayes, decision tree, and random forest, can be used to build predictive models. Type `?assign.MC` to see details.

K-fold cross-validation for baseline evaluation

While Monte-Carlo cross-validation helps evaluate variation in assignment accuracy through resampling, it does not guarantee that every individual was tested, and multiple tests on an individual also makes it difficult to estimate membership probabilities across individuals and populations. As such, we introduced *K*-fold cross-validation to help estimate membership probability because it gurantees that every individual will be tested once while none of them will be used as training and test individuals in the same test. To perform *K*-fold cross-validation, use the following function:

```
assign.kfold(genin_rd, k.fold=c(3, 4, 5), train.loci=c(0.1, 0.25, 0.5, 1),
             loci.sample="random", dir="Result-folder2/", model="lda" )
```

In the *K*-fold cross-validation, rather than specifying the proportions of individuals to be sampled, use the argument `k.fold` to specify the number of groups to divide each population into. The above example divides individuals from each population into 3, 4, or 5 groups. Within each fold, 10%, 25% and 50% of random loci (arg. `loci.sample = "random"`), as well as all loci, were sampled for training data (arg. `train.loci`). As such, the *K*-fold cross-validation performs a total of 48 assignment tests (N = (3+4+5)*4 tests).

When analyzing non-genetic data alone, the arguments, `train.loci` and `loci.sample`, will be automatically ignored.

To most efficiently compute the analyses, by default, all available cores/threads minus one (N-1) of your computer’s CPU are used for parallel analyses so that multiple assignment tests are performed simultaneously (assuming the computer has a multi-core CPU). To change the number of cores/threads used, simply specify a number in the argument `processors`.

Depending on the size of your dataset, running `assign.MC()` may take a few minutes to hours; `assign.kfold()` is usually quicker because of fewer tests. You can monitor the analysis status by counting the output files in the result folder.

Assign unknown individuals

In addition to evaluting a known data set, the package also provides a function `assign.X()` that allows you to perform an assignment test on unknown individuals using the known data that was evaluated (assuming the results are satisfactory). After importing your known and unknown data sets (they should be the same data type and have same feature names), use the arguments `x1` and `x2` to specify the known and unknown data, respectively.

```
# Perform an assignment test using decision tree as the predictive model
assign.X( x1=genin, x2=genin_unknown, dir="Result-folder3/", model="tree")

# Use `?assign.X` to see other argument settings.
```

The assignment result will be saved in a text file named *AssignmentResult.txt* under the designated folder. This file includes sample IDs of your unknown individuals, predicted populations and the probabilities. When the assignment is done, it will prompt a message asking you whether a membership probability plot should be generated. You can enter **Y** to visualize the result right away.

PCA for data transformation

The assignment functions, `assign.MC()`, `assign.kfold()` and `assign.X()`, use PCA for dimensionality reduction. It converts your independent variables to Principle Components and retain the PCs that have eigenvalues greater than 1 (the Kaiser-Guttman criterion) as new features (arg. `pca.PCs="kaiser-guttman"`). Or, you can specify an integer in the argument `pca.PCs` (e.g., `pca.PCs = 10`) to choose a specific number of PCs to be retained.

The PCA will always perform on the genetic data, whether you are analyzing genetic-only or integrated data. When analyzing integrated data, you have three options to perform PCA on the data. First, you can perform PCA across all features (set arg. `pca.method="mixed"`), meaning that each PC includes information of genetic and non-genetic data. This is default setting. Second, you can perform PCA on genetic and non-genetic data indpendently (set arg. `pca.method="independent"`), meaning that your transformed data includes genetic PCs and non-genetic PCs. Third, you can choose to use original non-genetic data as features without peforming PCA on them (set. arg. `pca.method="original"`). This option ends up using genetic PCs and original non-genetic data to build predictive models.

When analyzing non-genetic data alone, you can determine whether PCA is performed on the data. You set the argument `pca.method=TRUE` to perform PCA, or `pca.method=FALSE` to not perform PCA.

Calculate assignment accuracy

When the assignment analysis (`assign.MC()` or `assign.kfold`) is complete, assignment accuracies can be calculated using the following functions.

```
accuMC <- accuracy.MC(dir = "Result-folder/") #Use this function for Monte-Carlo cross-validation results
accuKF <- accuracy.kfold(dir = "Result-folder2/") #Use this function for K-fold cross-validation results
```

These functions read through each assignment result in the designated folder (arg. `dir`) and calculate the assignment accuracies over all populations and for each population. The results will be saved in a text file (named **Rate_of….txt**) and a returned object if provided (e.g., `accuMC` and `accuKF`). This file or object (as a data frame object) will be used to create assignment accuracy plots (described in the next section). If you did not provide a returned object or if you want to read the data from the text file later on, simply use the `read.table()` function to import the data into R.

```
accuMC <- read.table("Rate_of....txt", header=T)
```

Because Monte-Carlo cross-validation is designed for evaluating assignment accuracy but *K*-fold cross-validation is not, executing the function `accuracy.MC()` for the Monte-Carlo results should suffice. The results from *K*-fold cross-validation will be used to make membership probability plots.

Visualize results

Assignment accuracy

To visualize the results of assignment accuracy, use the following function to create a boxplot.

```
accuracy.plot(accuMC, pop = "all")
```

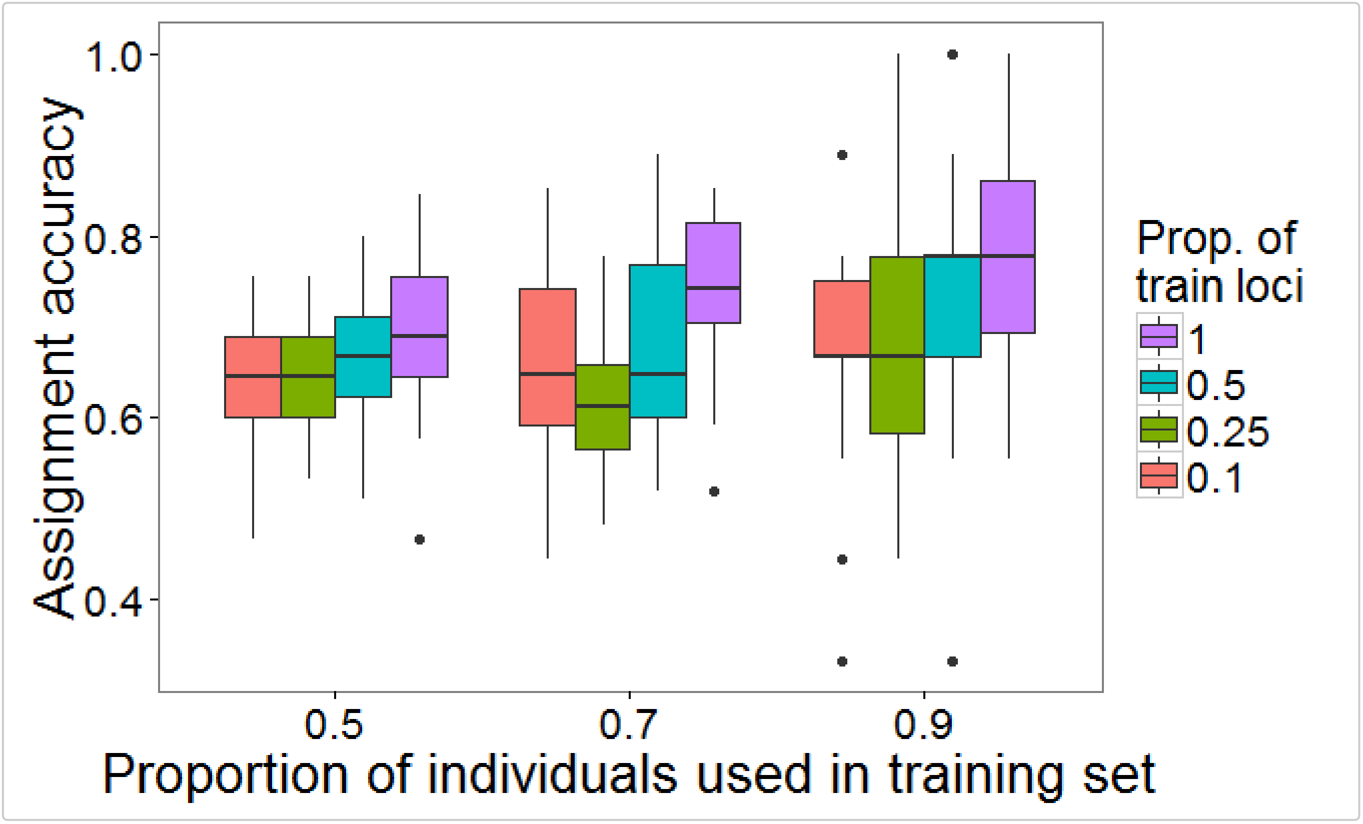


Figure 1. Assignment accuracies estimated via Monte-Carlo cross-validation, with three levels of training individuals (50%, 70% and 90% of individuals from each population, on x-axis) by four levels of training loci (top 10%, 25% and 50% highest Fst loci and all loci in color-coded boxes) by 30 resampling events.

Note that the argument `pop` is used to specify which population to be included in the plot. You can specify multiple populations in the argument (e.g., `pop=c("pop_A","pop_B")`) to make a faceted plot (population names are those you provided in the function `read.genpop()`). If the population is not specified, assignment accuracy of overall population (arg. `pop="all"`) will be plotted as the default.

The function `accuracy.plot()` is built based on the ggplot2 library, so you can import the library `library(ggplot2)` and modify the plot using ggplot2 functions/arguments. For example, set the *y*-axis (assignment accuracy) limits (`ylim()`), draw a horizontal line (`annotate()`), or add a plot title (`ggtitle()`).

```
library(ggplot2)
accuracy.plot(accuMC, pop=c("all", "pop_A", "pop_B", "pop_C")) +
  ylim(0, 1) + #Set y limit between 0 and 1
  annotate("segment",x=0.4,xend=3.6,y=0.33,yend=0.33,colour="red",size=1) + #Add a red horizontal
line at y = 0.33 (null assignment rate for 3 populations)
  ggtitle("Monte-Carlo cross-validation using genetic loci")+ #Add a plot title
  theme(plot.title = element_text(size=16)) #Edit plot title text size
```

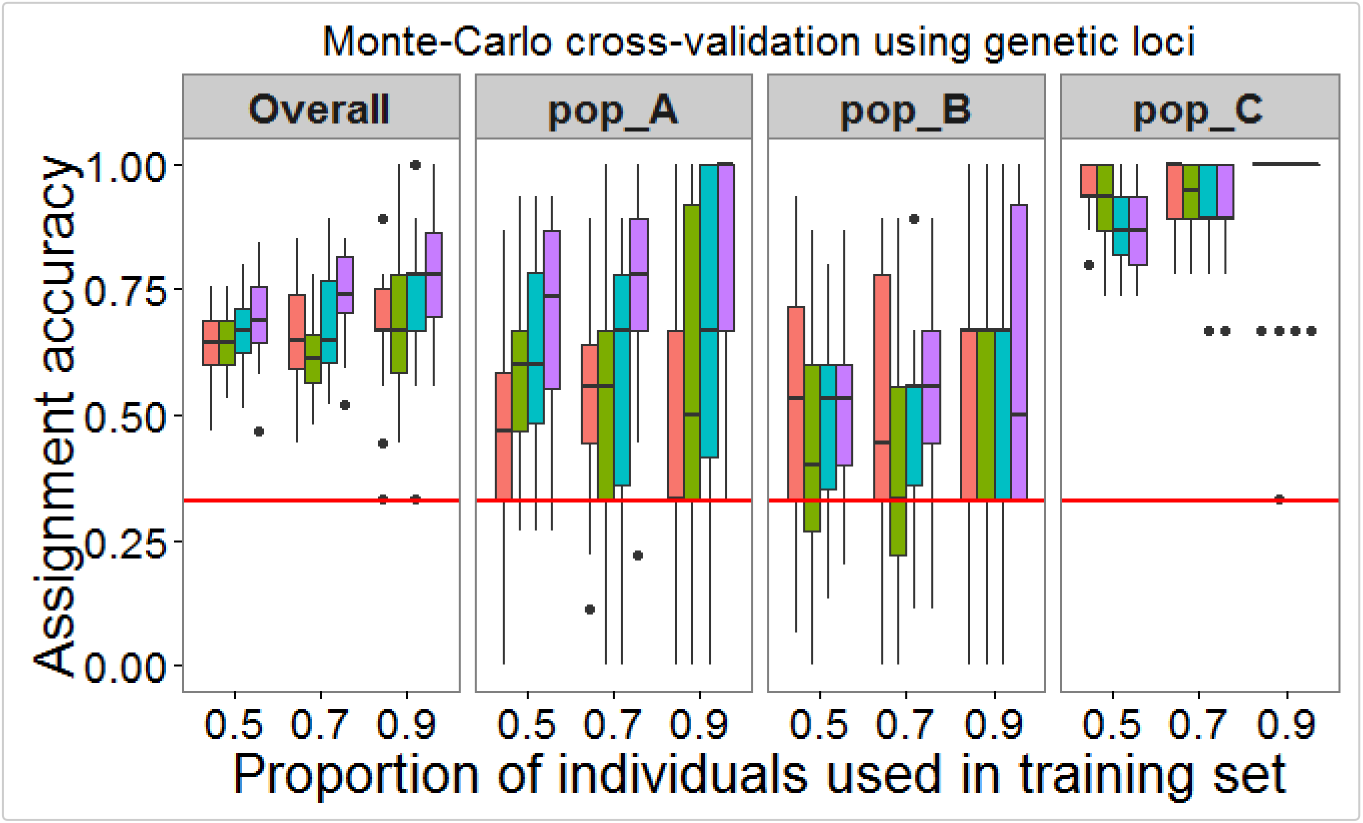


Figure 2. Assignment accuracies estimated via Monte-Carlo cross-validation, with genetic data (693 SNP loci) for three hypothetical populations of 30 individuals. Red horizontal lines indicate 0.33 null assignment rate.

The above results were estimated from the genetic-only dataset. Assignment accuracies between populations A and B are relatively low, indicating that the genetic data could not distinguish between them well. However, assignment accuracies of population C are much higher, suggesting that the genetic loci can be used to differentiate population C from the other two populations. Next, we can perform the assignment test on the genetic-morphometric dataset (using object `comin` returned from `compile.data()`) to see if using integrated data improves assignment success. (We skip the codes and show the final result below)

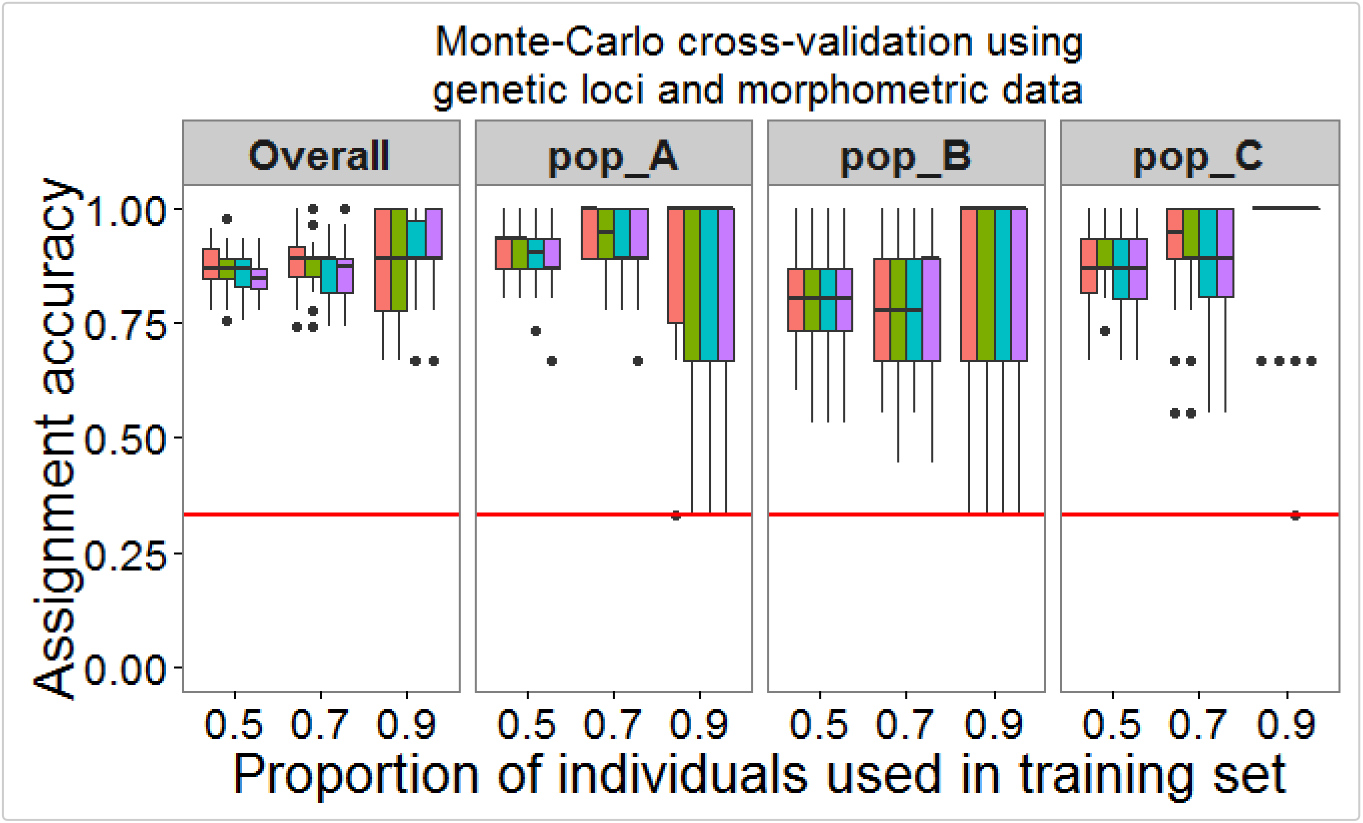


Figure 3. Assignment accuracies estimated via Monte-Carlo cross-validation, using genetic and morphometric data. Each box is the results of using a proportion of training loci plus 4 morphometric variables

When using the genetic-morphometric dataset, the assignment accuracies of populations A and B increased and that of population C remained high, resulting in increasing overall assignment accuracy. These results demonstrate the potential of using multiple data types to improve assignment success. Additionally, we can use the same analytical methods (i.e., Monte-Carlo cross-validation with the same proportions of training individuals, iterations, and classifier) to evaluate morphormetric data alone. The results are as follows.

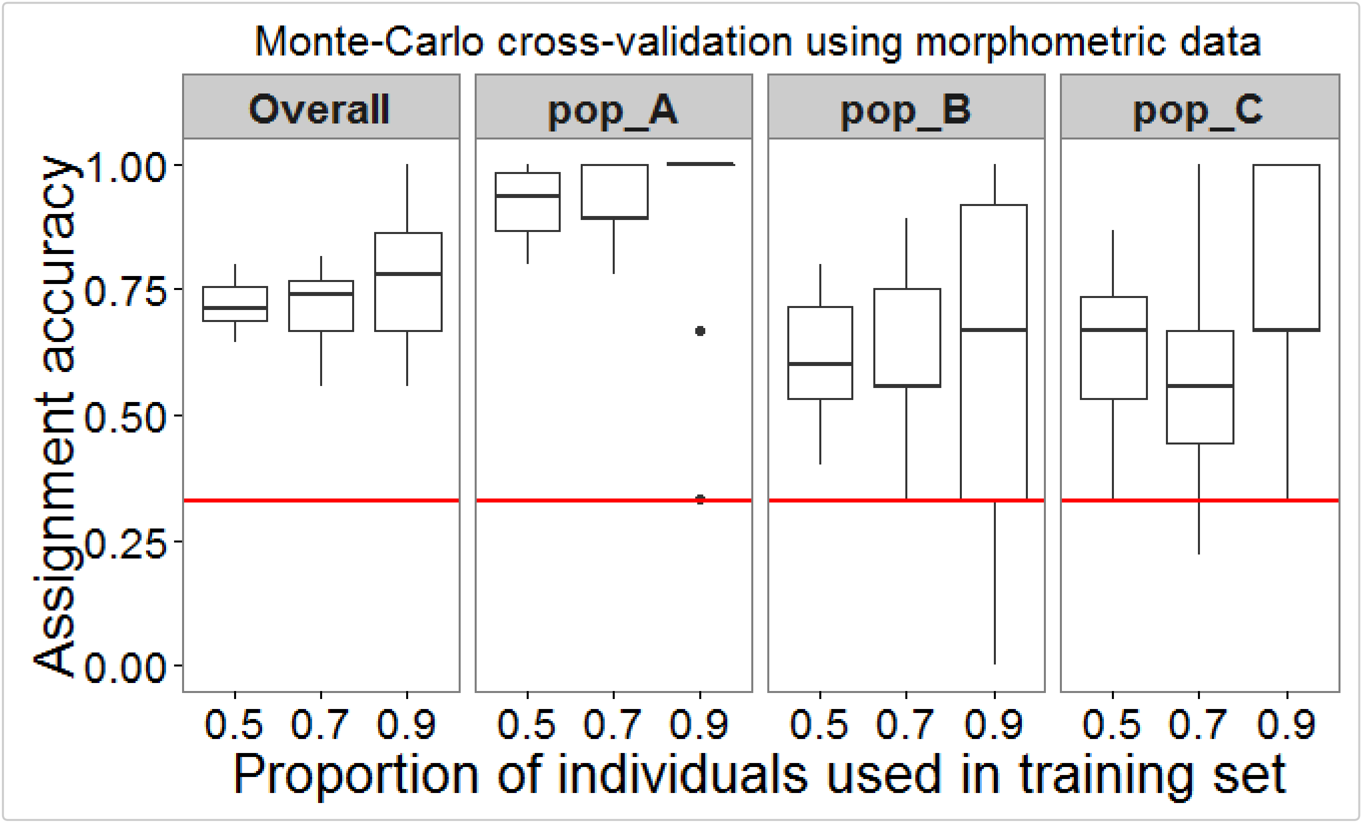


Figure 4. Assignment accuracies estimated via Monte-Carlo cross-validation, using 4 morphormetric measurements.

The above results indicate that morphormetric data helps distinguish population A from the other two. Therefore, it is exepected that using integrated data would best discriminate among the three populations (Figure 3) despite that fact that using genetics or morphormetrics alone was unable to distinguish population B from the other two (see population B in Figure 2 and 4).

Membership probability

In addition to estimating assignment accuracy, we can also use probabilities to understand how individuals are assigned to populations. To visualize membership probility, we use the results from *K*-fold cross-validation and create a stacked bar plot (also known as **STRUCTURE plot**) that is commonly used in molecular biology papers. To create the plot, use the following function and specify the folder containing your *K*-fold cross-validation results.

```
membership.plot(dir = "Result-folder2/")
```

After entering the code, it will prompt a few questions and allow you to choose which dataset and plot style to be used. The interactive conversation is shown as follows.

```
## K = 3  4  5  are found.
## Please enter one of the K numbers: (You will enter your answer here)
```

Next, it will ask you to enter which proportion of training loci you would like to make the plot for.

```
## 4 levels of training loci are found.
## Levels[train.loci]: 0.1  0.25  0.5  1
## Please enter one of the levels: (You will enter your answer here)
```

Lastly, if you didn't specify the output style (e.g., `style = 1`) in `membership.plot()`, then it will print the following text and ask you to choose an output style.

```
## Finally, select one of the output styles.
## [1] Random order (Individuals on x-axis are in random order)
## [2] Sorted by probability (Individuals are sorted by probabilities within each group)
## [3] Separated by fold (Individuals of different folds are in separate plots)
## [4] Separated and Sorted (Individuals are separated by fold and sorted by probability)
## Please enter 1, 2, 3, or 4: (You will enter your answer here)
```

Below we use the results of 3-fold and all loci (train.loci = 1) to demonstrate membership probability plots, with four different output styles.

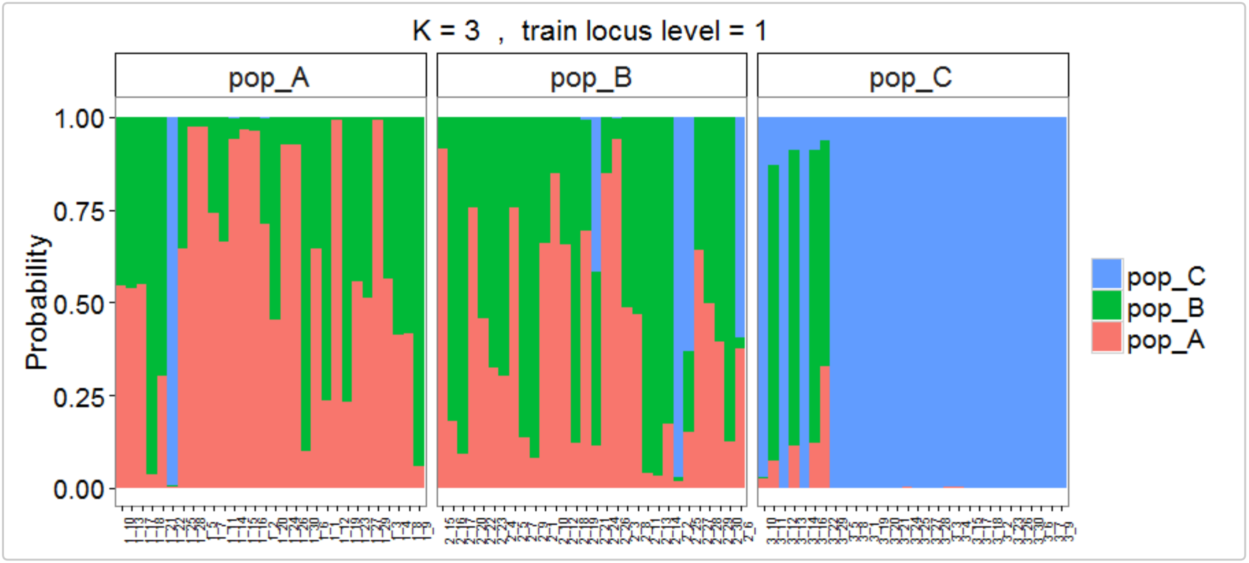


Figure 5. Membership probability of three hypothetical populations, with results estimated via 3-fold cross-validation using overall loci (693 SNPs). Output style = 1.

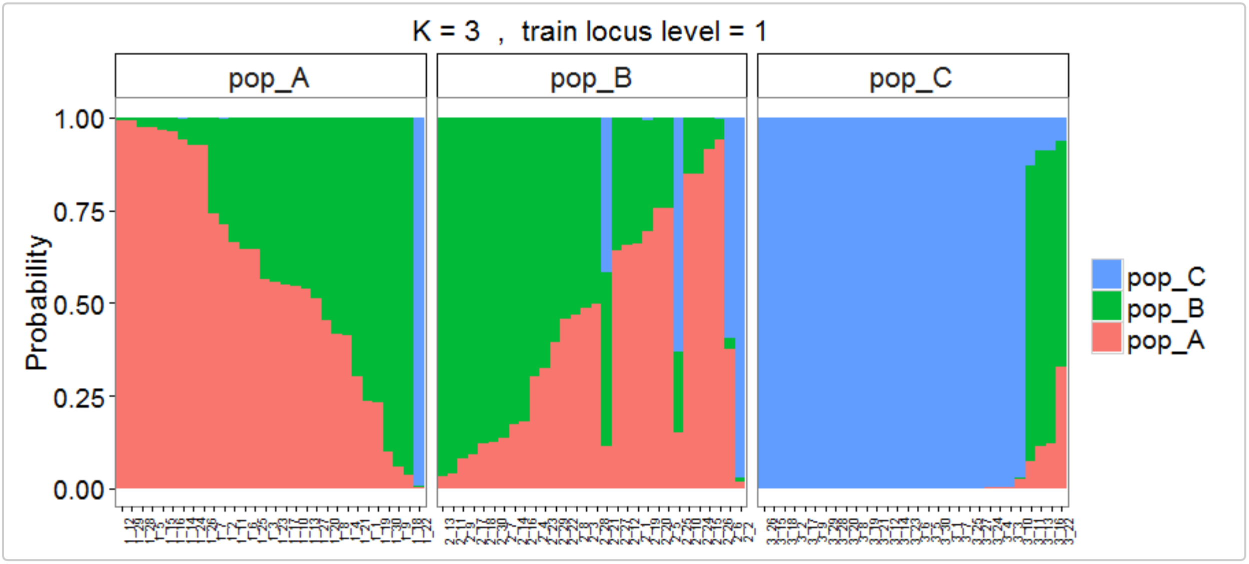


Figure 6. Output style = 2. Individuals are sorted based on the probability of assignment to their original populations.

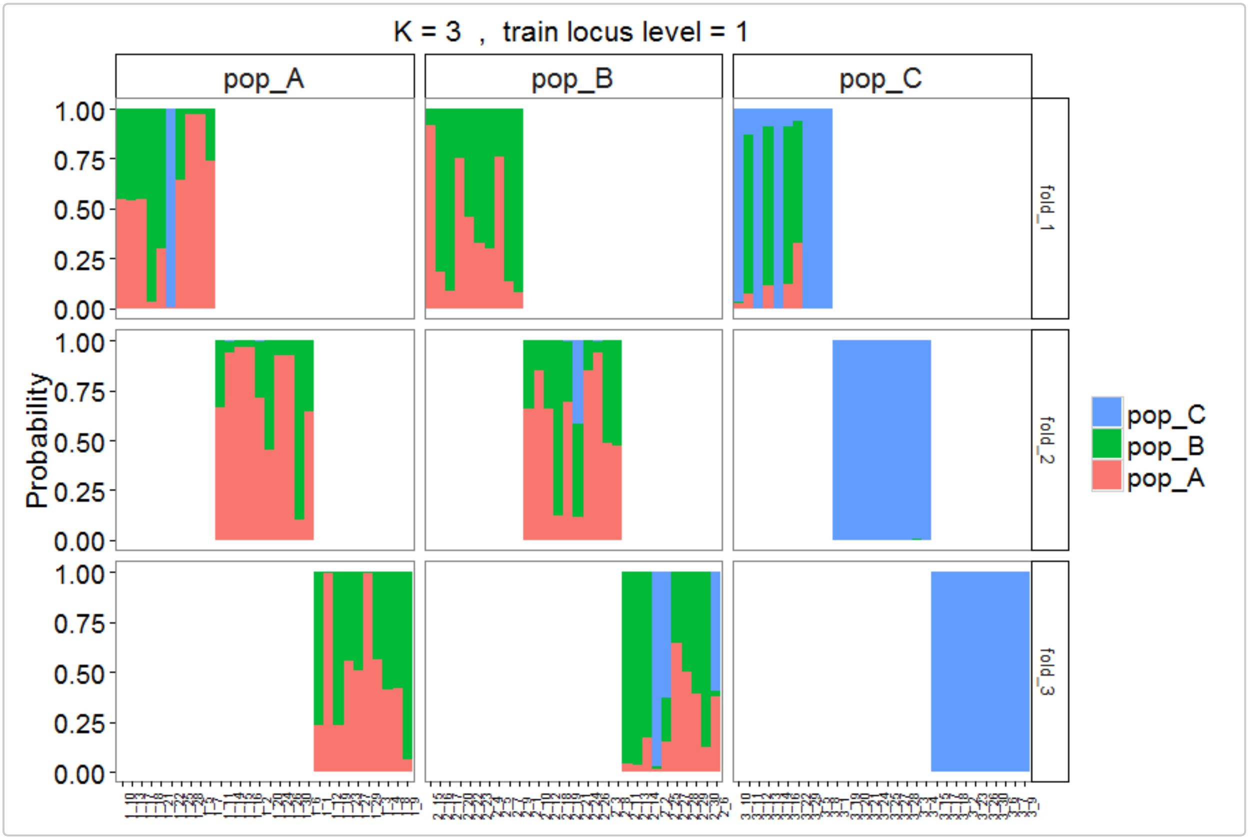


Figure 7. Output style = 3. Similar to Figure 5, but with individuals assigned to different folds in separate panels.

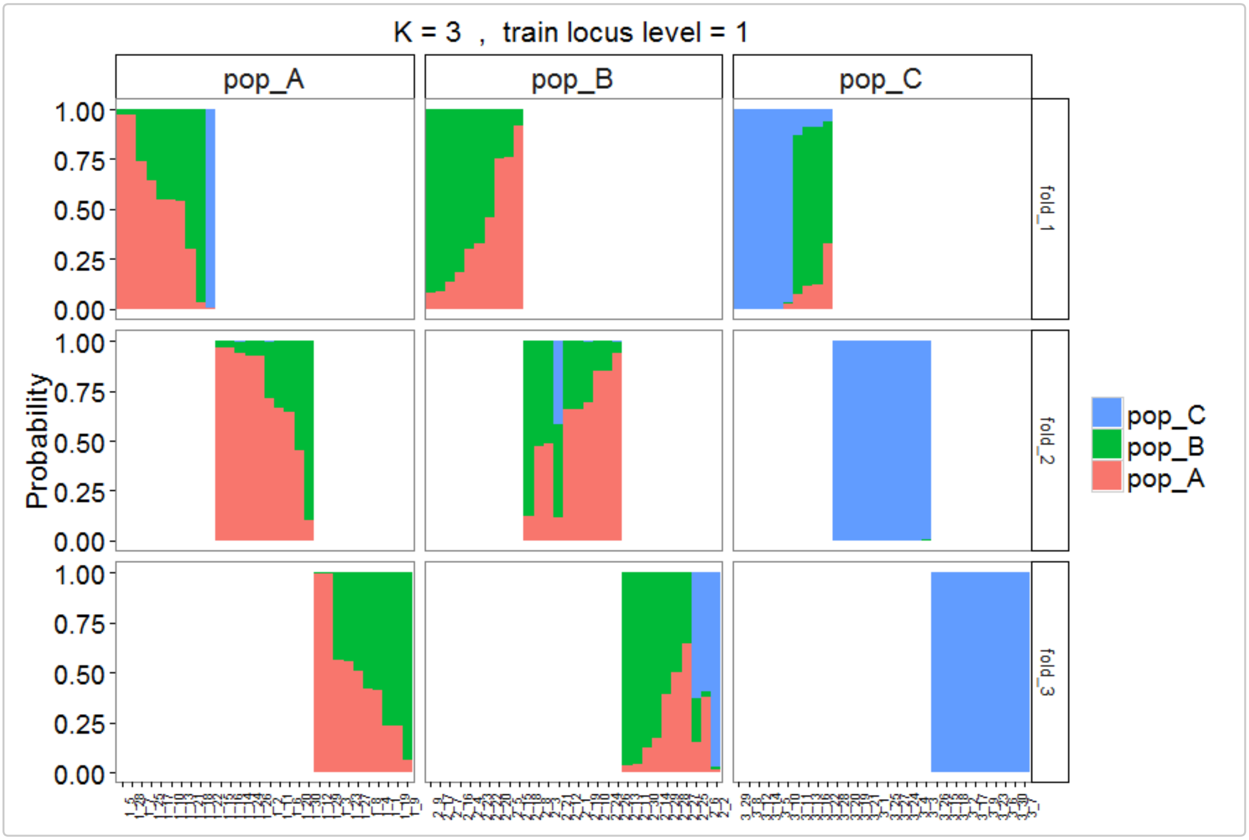


Figure 8. Output style = 4. Similar to Figure 7, but with individuals sorted based on probability of assignment to their original populations.

The above Figures 5 to 8 are the results from the genetic-only dataset. Here we also analyzed the genetic-morphometric data and created the following membership probability plot.

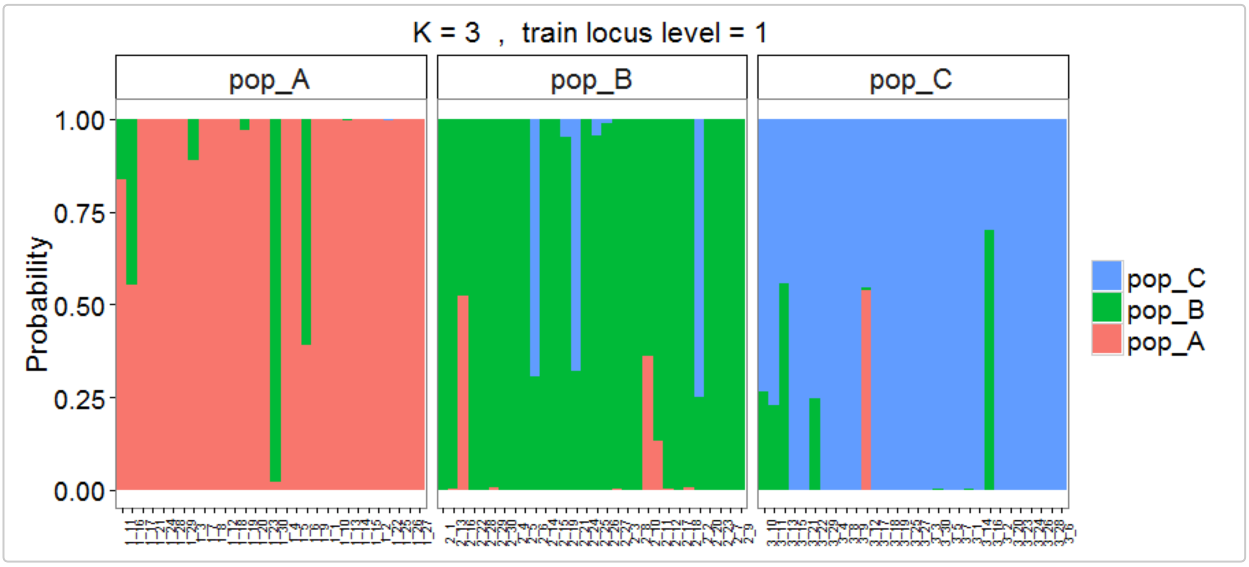


Figure 9. Membership probability of three hypothetical populations, with results estimated based on 693 loci plus 4 morphometric measurements.

Using the integrated dataset allowed many more correct assignments than using the genetic data alone (see Figure 5 or 6 vs. 9).

Identify informative loci

In some cases, using a subset of high F_{ST} loci may produce similar assignment accuracy as using all available loci. Identification of these loci could help reduce time and cost in preparing samples in the future and help identify loci that might be associated with functional genes.

The following function reads through the training locus file for each assignment test, counts the frequency of occurrence of each locus, and writes the results to a text file. This function should be run on the cross-validation results based on resampling high F_{ST} training loci (`loci.sample = "fst"`), rather than the randomly resampled loci (`loci.sample = "random"`), as the latter would be uninformative.

```
check.loci(dir = "Result-folder/", top.loci = 20)
```

By default, this function outputs top 20 loci that are most frequently used as training loci (determined by locus F_{ST} value and proportions of your training loci) across your assignment tests, but you can instead specify the number of top training loci to be included.

When the function is executed, you will be prompted to choose which specific proportion of training individuals you would like to look up.

```
## 3 levels of training individuals are found.
## Which levels would you like to check? (separate levels by a whitespace if multiple)
## Options: 0.5, 0.7, 0.9, or all
## enter here: (You will enter your answer here)
```

The F_{ST} value calculated for a locus will differ depending on which subset of individuals was sampled for the training set. We recommend running the `check.loci()` function across all levels of proportion of training individuals to evaluate if the loci are consistently found to be informative.

The output file (*High_Fst_Locus_Freq.txt*) includes a list of locus names ordered by F_{ST} value (highest F_{ST} in the top row). The number in parentheses following each locus name indicates how many tests that locus appeared in that rank. A sample of output content is shown below.

```
Loci occur in top 20 high Fst across all training data
top.1(1): Locus_171(360),
top.2(1): Locus_442(360),
top.3(1): Locus_475(360),
top.4(1): Locus_481(360),
top.5(1): Locus_696(360),
top.6(1): Locus_697(360),
top.7(1): Locus_729(360),
top.8(1): Locus_745(360),
top.9(1): Locus_812(360),
top.10(1): Locus_941(360),
top.11(1): Locus_992(360),
top.12(5): Locus_113(248), Locus_114(76), Locus_115(24), Locus_320(8), Locus_139(4),
top.13(7): Locus_245(240), Locus_181(72), Locus_147(24), Locus_115(8), Locus_560(8), Locus_137(4),
Locus_160(4),
```

The above results show that across our 360 tests, the top 11 highest F_{ST} loci are the same across all 360 tests. Four loci (`Locus_113, Locus_114, Locus_115, and Locus_320`) appear to be the 12th highest F_{ST} locus across the tests, and `Locus_113` occurs most frequently (248 out of 360 tests) in this rank.

Report package issues

Please leave your comments or questions at https://github.com/alexkychen/assignPOP/issues

---

References and footnotes

---

1. Anderson, E. C. (2010). Assessing the Power of Informative Subsets of Loci for Population Assignment: Standard Methods Are Upwardly Biased. *Molecular Ecology Resources* 10(4): 701–710. doi:10.1111/j.1755-0998.2010.02846.x.↩
2. Waples, R. S. (2010). High-Grading Bias: Subtle Problems with Assessing Power of Selected Subsets of Loci for Population Assignment. *Molecular Ecology* 19(13): 2599–2601. doi:10.1111/j.1365-294X.2010.04675.x.↩
3. Rousset, F. (2008). genepop’007: A Complete Re-Implementation of the Genepop Software for Windows and Linux. *Molecular Ecology Resources* 8(1): 103–106. doi:10.1111/j.1471-8286.2007.01931.x.↩

4. The name order in the argument `pop.names` should match the group order in your GENEPOP file. For example, the first name, `pop_A`, in `pop.names=c("pop_A","pop_B","pop_C")` is the top first group (pop) of samples in the GENEPOP.↩

5. A low variance locus is one in which a single (major) allele occurs in most individuals, with an alternate (minor) allele occuring in only a few individuals. Removal of these low variance loci will help avoid the problem of a minor allele ending up being assigned to either only training or only test sets. For example, if 9 out of 10 individuals have the same genotype (e.g., AA, A is major allele) and only 1 individual has the minor allele (e.g., Aa, a is minor allele with 0.05 frequency), the locus will not help predict the group of individuals regardless of whether the rare Aa individul is assigned to a training or test set. Therefore, removal of low variance loci does not influence results but helps accelerate post analyses.↩
6. In this case, we have 614 alleles across 307 loci, meaning that every locus is biallelic (307 loci x 2 alleles = 614 alleles)↩
7. A column includes population names for each individual. Also be sure to include a column name in table header.↩