R package [assignPOP v1.0] tutorial

Alex Chen

### Introduction

The assignPOP package employs [supervised machine learning](https://en.wikipedia.org/wiki/Supervised_learning) methods to help perform population assignment using either genetic or integrated (genetic plus non-genetic) data. It uses principle component analysis (PCA) for dimensionality reduction so it can efficiently handle large genomic data (e.g., thousands of loci). It performs resampling (Monte-Carlo and *K*-fold) cross-validation coupled with various machine learning classifiers, including LDA, SVM, naive Bayes, decision tree, and random forest, to allow users to fine tune predictive models for estimating assignment accuracy and membership probability. The package provides functions to output results in publication-quality plots while being editable using ggplot2 library. In short, the package helps evaluate whether a dataset has reliable discriminatory power to perform population assignment and identify informative loci while solving the upwardly biased issue that was discussed in previous studies (Anderson 2010; Waples 2010).

[1. Install and import assignPOP](#install)  
[2. Prepare and import genetic data](#prepare)  
[3. Remove low variance loci](#remove)  
[4. Concatenate genetic and non-genetic data](#Compile)  
[5. Perform population assignment](#assignment)  
[6. Calculate assignment accuracy](#accuracy)  
[7. Make boxplots for assignment accuracy](#AAplot)  
[8. Make stacked bar plots for membership probability](#membership)  
[9. Identify informative loci](#infoloci)

### 1. Install and import assignPOP

To install the package from CRAN, type:

install.packages("assignPOP")

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

install.packages("devtools")  
library(devtools)  
install\_github("alexkychen/assignPOP")

To import the package to your R environment, type:

library(assignPOP)

### 2. Prepare and import genetic data

Currently, assignPOP accepts genetic data in GENEPOP format (Rousset 2008), 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()as follows.

infile <- read.genpop( "simGenepop.txt", pop.names=c("pop\_A","pop\_B","pop\_C") )

The example input file, [*simGenepop.txt*](https://raw.githubusercontent.com/alexkychen/assignPOP/master/inst/extdata/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 so that you can refer to them by these names in your output.

After importing the file, you should see the following message printed in your R console.

## ############### assignPOP v1.0 ###############  
##   
## 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]]

### 3. 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 across the dataset in advance of further analyses. 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. 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 dataset.

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

infile\_rd <- reduce.allele(infile, 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  
## 724 loci remain

In our example, 276 low variance loci were removed, leaving 724 loci in the new dataset, which will be used in further analyses.

### 4. Concatenate genetic and non-genetic data (optional)

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 dataset 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 datasets, the individual will be ignored in the new integrated dataset.

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

infile\_con <- compile.data(infile\_rd, "morphData.csv")

The above [*morphData.csv*](https://raw.githubusercontent.com/alexkychen/assignPOP/master/inst/extdata/morphData.csv) file has the same individuals as in [*simGenepop.txt*](https://raw.githubusercontent.com/alexkychen/assignPOP/master/inst/extdata/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](https://en.wikipedia.org/wiki/Dummy_variable_(statistics)) using [model.matrix()](https://stat.ethz.ch/R-manual/R-devel/library/stats/html/model.matrix.html) 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 724 loci(1386 alleles) plus 4 additional variables(4 columns)

### 5. Perform population assignment (using [Monte-Carlo and *K*-fold cross-validation](http://stats.stackexchange.com/questions/51416/k-fold-vs-monte-carlo-cross-validation))

We use Monte-Carlo cross-validation (function assign.MC()) to estimate assignment accuracies over all populations and for each population, and use *K*-fold cross-validation (function assign.kfold()) to estimate membership probabilities across all individuals.

When performing Monte-Carlo cross-validation, 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 or based on locus *FST* estimated within training individuals (arg. loci.sample). 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 *FST* loci as well as all loci for the training data. Each sample size combination was resampled 30 times.

assign.MC(infile\_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.

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; otherwise, the output results will be saved under your working directory.

In addition, you can use the argument model to select a classifier for prediction. In this case, the [Support Vector Machine (SVM)](https://en.wikipedia.org/wiki/Support_vector_machine) classifier is used. Other SVM related arguments, svm.kernel and svm.cost, can be specified to fine tune the predictive model. See the [e1071](https://cran.r-project.org/web/packages/e1071/index.html) package for details about SVM. Other classifiers, including [LDA](https://en.wikipedia.org/wiki/Linear_discriminant_analysis), [naive Bayes](https://en.wikipedia.org/wiki/Naive_Bayes_classifier), [decision tree](https://en.wikipedia.org/wiki/Decision_tree), and [random forest](https://en.wikipedia.org/wiki/Random_forest), can be used to build predictive models. Type ?assign.MC to see details.

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. To perform *K*-fold cross-validation, use the following function:

assign.kfold(infile\_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).

Both functions use PCA for dimensionality reduction (i.e., converting your independent variables to Principle Components) and retain the PCs that have eigenvalues greater than 1 (the Kaiser-Guttman criterion). To choose a specific number of PCs to be retained for training data, you can specify an integer in the argument pca.method (e.g., pca.method = 10).

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.

### 6. 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 specified 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.

### 7. Make boxplots for assignment accuracy

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

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

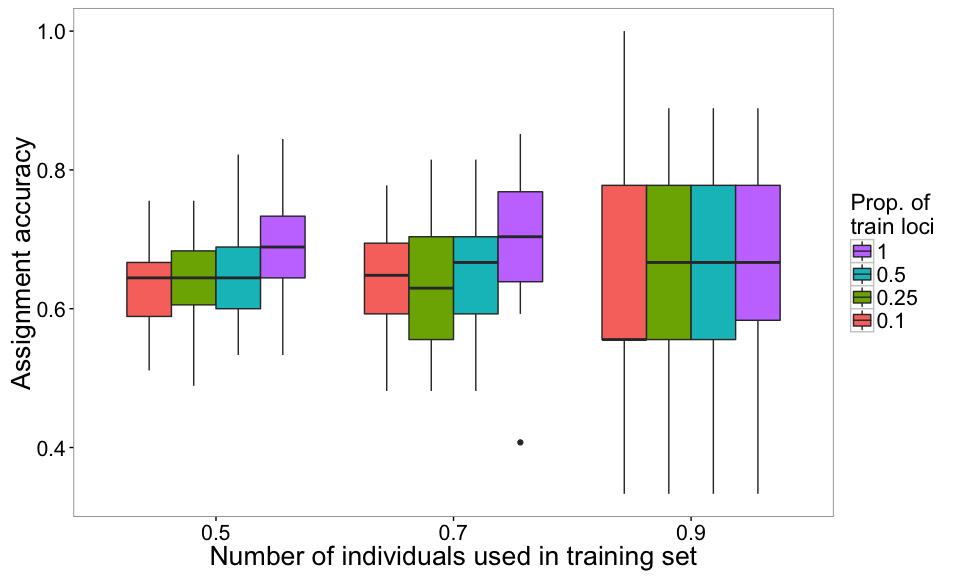


Figure 1. Assignment accuracies estimated via Monte-Carlo cross-validation, with three levels of training individuals (50%, 70% and 90% of individuals from each population, on x-axis) by four levels of training loci (top 10%, 25% and 50% highest Fst loci and all loci in color-coded boxes) by 30 resampling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

Note that the argument pop is used to specify which population to include in the plot. If the population is not specified, assignment accuracy over all populations (arg. pop="all") will be plotted as the default. To see the results of a specific population, use the population name that you provided in the function read.genpop().

The function accuracy.plot() is built based on the [ggplot2](https://cran.r-project.org/web/packages/ggplot2/ggplot2.pdf) library, so you can import the library library(ggplot2) and modify the plot using [ggplot2 functions/arguments](http://docs.ggplot2.org/current/). 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 = "all") +  
 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("Overall population") #Add a plot title

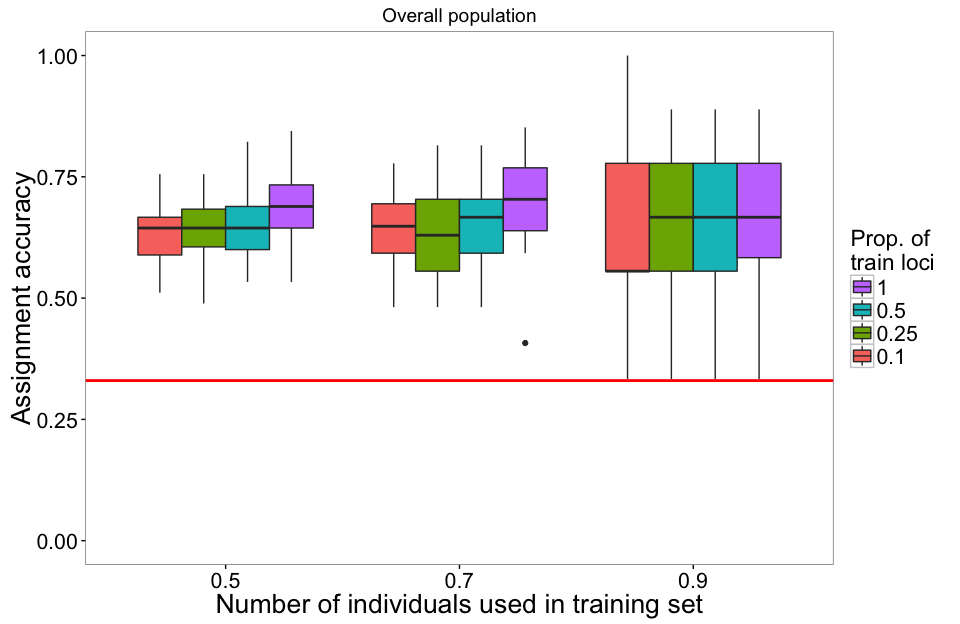


Figure 2. Assignment accuracy results identical to Figure 1 , with modified y-axis limit and a horizontal line indicating 0.33 null assignment rate.

Moreover, to see assignment accuracy for each population (because it may vary among populations), you can make a faceted boxplot with each population in a panel using the following script.

# The first thing to do is to convert numeric variables (train.inds and train.loci) to factors  
accuMC$train.inds <- as.factor(accuMC$train.inds)  
accuMC$train.loci <- as.factor(accuMC$train.loci)  
  
# Next, we use metl() function to reshape the data frame in which the first three columns are kept,  
# assignment accuracies are merged into one column (col. name is "value"),   
# and a new column with population names is created (col. name is "variable").  
library(reshape) #or library(reshape2)  
accuMC\_rs <- melt(accuMC, id = 1:3)  
levels(accuMC\_rs$variable) <- c("Overall", "Pop A", "Pop B", "Pop C") # Change variable name for panel title  
  
# Create a faceted boxplot  
ggplot(accuMC\_rs, aes(x = train.inds, y = value, fill = train.loci )) +  
 geom\_boxplot() +  
 facet\_grid( . ~ variable) +   
 ylim(0,1) +   
 xlab("Proportion of individuals used in training set") + ylab("Assignment accuracy") +  
 scale\_fill\_discrete(name = "Proportion\nof loci\nin training") +  
 # Below are codes to modify plot style  
 theme\_bw() +   
 theme(panel.grid.major = element\_blank(), panel.grid.minor=element\_blank(),  
 strip.text.x = element\_text(size=16, face="bold"),  
 axis.text = element\_text(size=16, colour="black"), axis.title = element\_text(size=20),  
 legend.text = element\_text(size=16), legend.title = element\_text(size=20)) +  
 annotate("segment", x=0.4, xend=3.6, y=0.33, yend=0.33, colour="red", size=1)

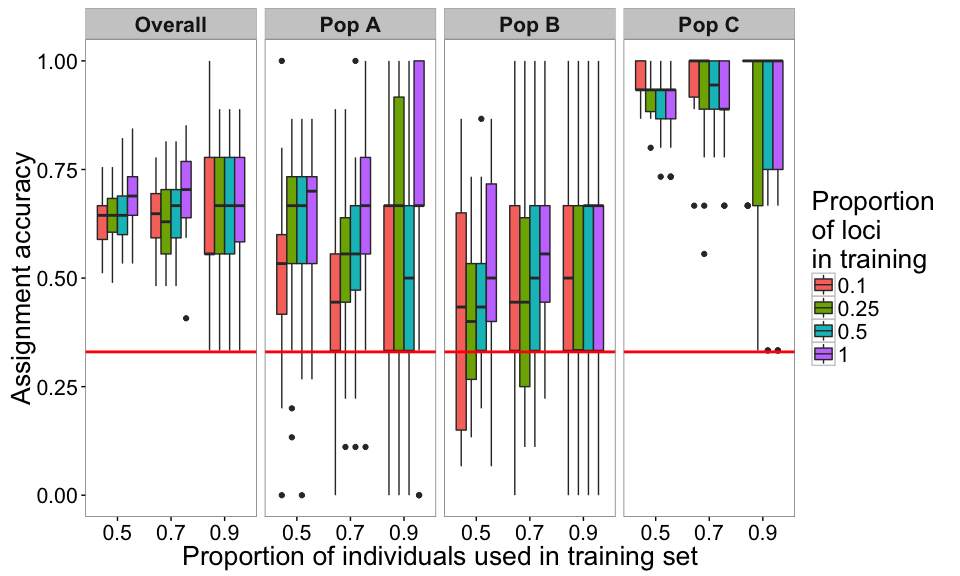


Figure 3. Assignment accuracies estimated via Monte-Carlo cross-validation, with genetic data (724 loci) for three hypothetical populations of 30 individuals.

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 infile\_con returned from compile.data()) to see if using integrated data improves assignment success. (We skip the codes and show the final result below)

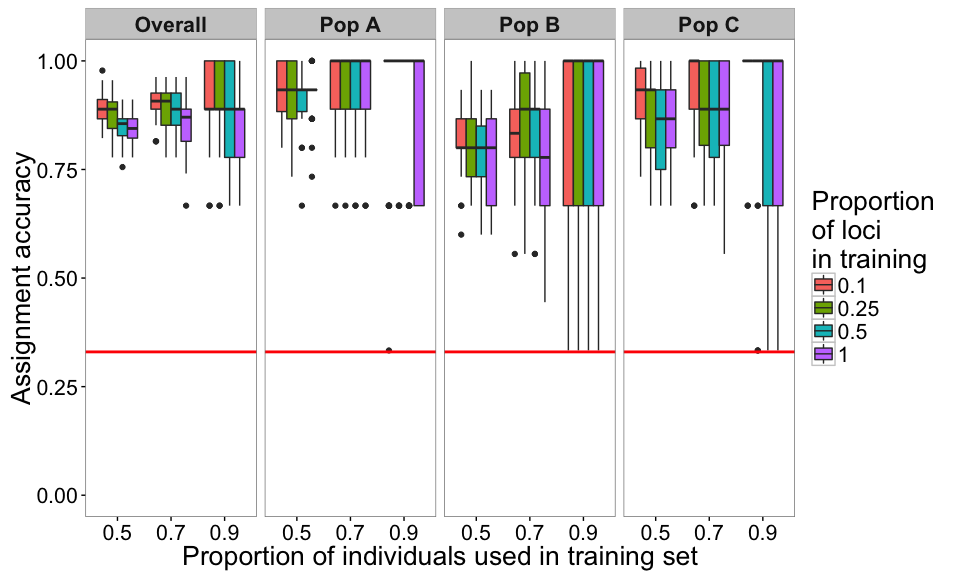


Figure 4. Assignment accuracies estimated via Monte-Carlo cross-validation, using genetic and morphometric data. Each box is the results of using a proportion of training loci plus 4 morphometric variables

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.

### 8. Make stacked bar plots for 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**](http://pritchardlab.stanford.edu/structure.html)) 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.

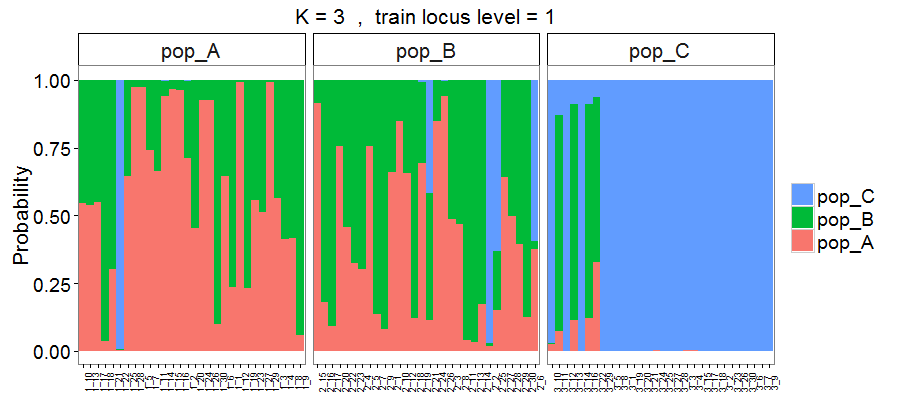


Figure 5. Membership probability of three hypothetical populations, with results estimated via 3-fold cross-validation using overall loci (724 SNPs). Output style = 1.

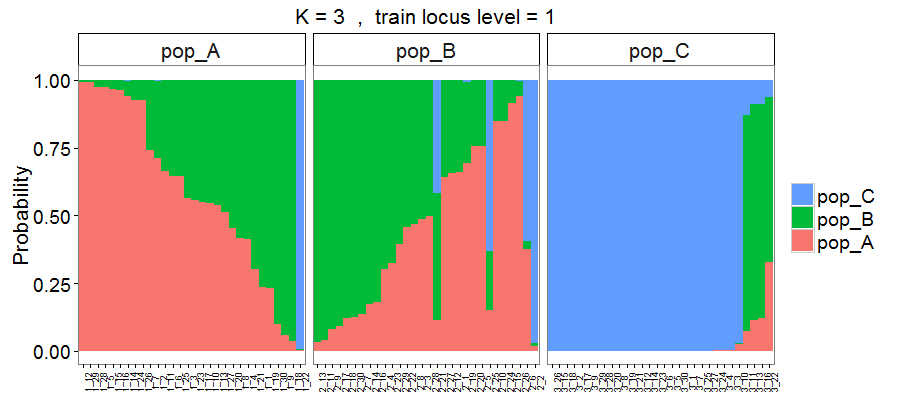


Figure 6. Output style = 2. Individuals are sorted based on the probability of assignment to their original populations.

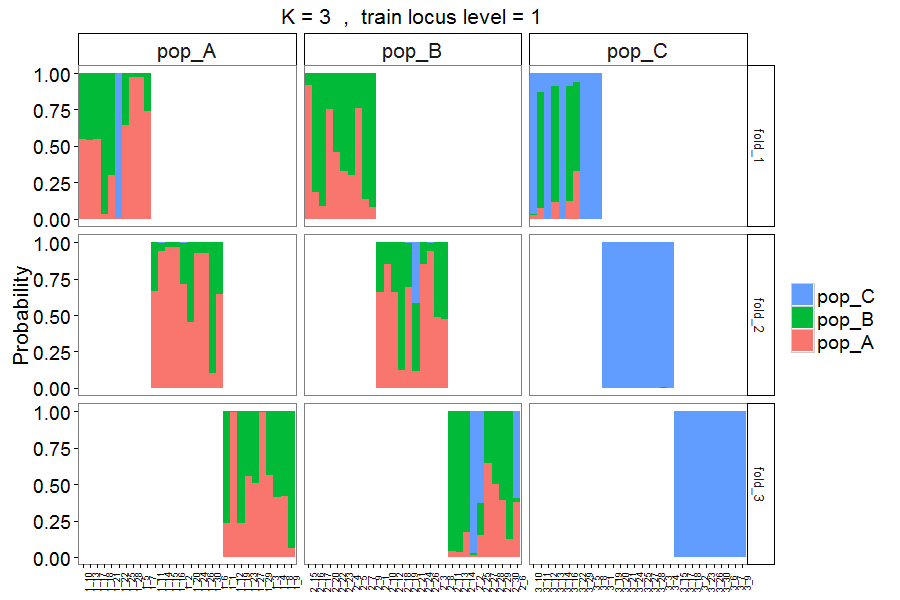


Figure 7. Output style = 3. Similar to Figure 5, but with individuals assigned to different folds in separate panels.

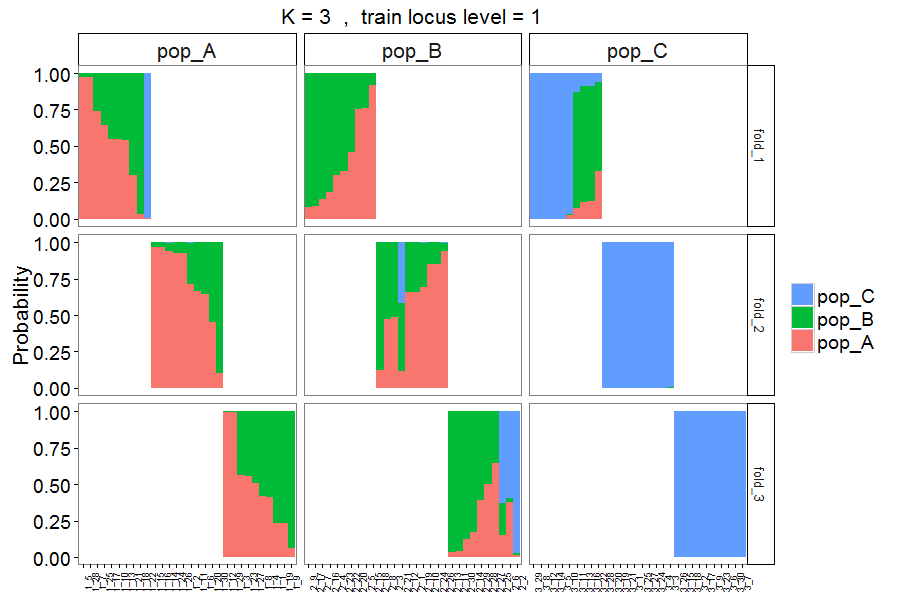


Figure 8. Output style = 4. Similar to Figure 7, but with individuals sorted based on probability of assignment to their original populations.

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.

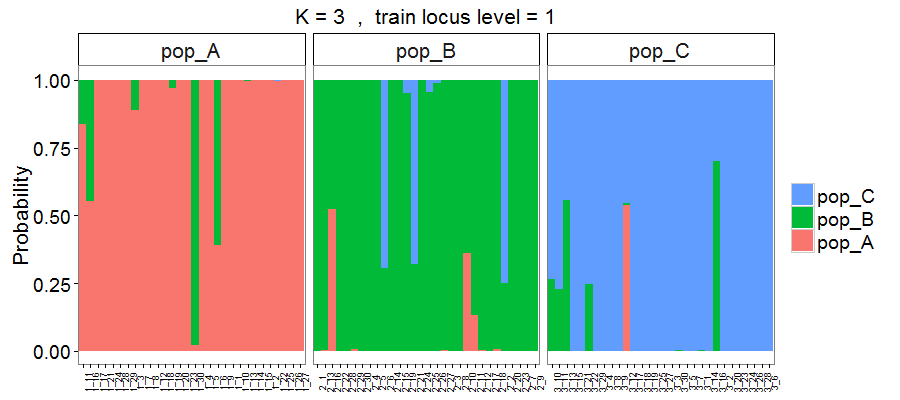


Figure 9. Membership probability of three hypothetical populations, with results estimated based on 724 loci plus 4 morphometric measurements.

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

### 9. Identify informative loci across overall assignment tests

In some cases, using a subset of high *FST* 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 *FST* 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 *FST* 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 *FST* 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 *FST* value (highest *FST* 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(4): Locus\_113(232), Locus\_114(80), Locus\_115(36), Locus\_320(12),   
top.13(6): Locus\_245(224), Locus\_181(76), Locus\_147(36), Locus\_560(12), Locus\_114(8), Locus\_137(4),

The above results show that across our 360 tests, the top 11 highest *FST* 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 *FST* locus across the tests, and Locus\_113 occurs most frequently (232 out of 360 tests) in this rank.

### Report package issue

If you encounter any issue when using the package, please report the issue via the following link. [<https://github.com/alexkychen/assignPOP/issues>](https://github.com/alexkychen/assignPOP/issues)

### Package citation

Chen, K-Y., Marschall, E. A., Sovic M. G., Fries, A. C., Gibbs, H. L., Ludsin, S.A. assignPOP: An R package for population assignment using genomic or integrated data in a machine learning framework.

### Referecnes

* 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>.
* Rousset, François. 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>.
* Waples, Robin 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>.