Build Random Forest for Ancestry estimation

Hannah Meyer

2024-08-22

Contents

Random Forest Classification 1

Parameters **1**

Evaluating/Interpretting the RF 2

References **2**

Random Forest Classification

The previous Ancestry estimation vignette performed PCA from the combined study and reference dataset and identified individuals of European descent based on clustering of data in the first and second PC space. In this two dimensional space, there is no adequate identification of individuals of other ancestries. This vignette will train a random forest classifier that uses up to 20 PCs to sufficiently estimate the study samples' ancestry based on any ancestry in the reference dataset.

We use the .eigenvec file that was created when preparing files for ancestry estimation, and read in its dataframe. The file contains the eigenvectors for the PCA on the LD-pruned genotype data. The .eigenvec file only included the eigenvalues per PC for each individual and did not include any information regarding individual's ancestries. Therefore, the file available in the plinkQC download, which contains a dataframe with each individual's ID, their family ID, and their associated ancestry must be obtained and read in, as obtained from the 1000 Genomes database. These two files are to be joined by ID and filtered to remove any IDs where no genetic data were available.

Parameters

Tuning the parameters for this random forest is crucial in building a machine-learning model to produce the best forest, in this case having the highest accuracy.

Number of Trees

A random forest model is comprised of building several trees and the 'Number of trees' parameter decides exactly how many decision trees will constitute the random forest. Although there exists a positive relationship between the number of trees and predictive performance of the model, there exists a point where continuing to increase the number of trees will no longer find you a better prediction while elongating computational efficiency. There also exists a tradeoff between predictive performance and computational efficiency. Excess

trees also pose a possible issue of overfitting as more trees are trained. It is important for your dataset to test and find the point of diminishing returns. For the example below of the 1000 Genomes set, 500 trees allows accurate predictions to be made, while also being computational eficient.

K-fold Cross Validation

Cross-validation is used to estimate the skill of a machine learning model on unseen data. When tuning this RF model, k-fold cross validation is used to split bootstrapped datasets into random groups, holding one group as test data and training the different models (the individual decision trees) on the remaining groups, where 'k' is the number of folds/groups in the dataset. Using k=5 or k=10 is common and usually arbitrarily chosen, depending on the size of the dataset. A higher number of folds tends to lead to a more accurate model as you are splitting. In the example below, k=10 for 10-fold cross-validation was used, meaning there were 10 iterations of the model for each of the 500 decision trees, each using a different group as the test set and the remaining 9 groups as training data for the model. After all 10 iterations were complete, the resulting iterations are averaged to find the final cross-validation model which is our random forest model.

It is important to test different values of k as there exists a tradeoff between model accuracy and computational expense. Each iteration requires the entire model to run, so this can get very computationally expensive depending on the size of the dataset.

mtry

The mrty parameter controls the number of variables tried at each split. During tre decision tree construcrion process, the confirmation that no two trees look exactly the same must be investiagted. More specifically, this mtry parameter controls the number of input features a decision tree is ablw to consider at any momnet during this tree construction and controls how much randomness is added during the process. Testing different values of mtry is crucial and can greatly impact the model's performance. For every new split that is added as a node to a decision tree, mtry controls how many of the principle component variables are allowed to be considered for each successive split.

If the value of mtry is too large and each model has access to all features as every split, there will not be as much randomness in the model and the decision trees may look very similar to each other, reducing the benefits of building multiple independent decision trees. Two common heuristics for choosing an mtry value is for the value of the parameter to either be the square root of or log base 2 of the total number of features. For the example below, those values end up being about 4.47 and 4.32, respectively; 4 was chosen to avoid the effect of a large mtry that diminishes the randomness in the model.

Evaluating/Interpretting the RF

References