Appendix A The Random Forests Classification Algorithm

A.1 Decision trees

A.1.1 Overview and terminology

An explanation of Random Forests (RF) analysis must start with the *decision trees* that comprise them. Decision trees are a simple but powerful tool for performing statistical classification[2]. Like other classification methods, the goal of decision tree analysis is to model a categorical response as a function of predictor variables. Decision trees are particularly flexible in the patterns they can identify, requiring no assumptions about the shape of predictor-response relationships. They also benefit from an ability to efficiently handle large multivariate datasets.

In general terms, the decision tree algorithm proceeds by repeatedly splitting the dataset into two groups based on predictor variables. At each branching point (*node*), the data may be divided at any location on any predictor variable; the algorithm considers all possibilities and chooses the one that provides the best separation between the resulting groups (*children*), where "best" may be defined by various criteria. This process proceeds until the final groups (*leaves*) contain observations of a single class. As discussed below, the RF algorithm essentially consists of producing many such trees for a single problem, and averaging the results to obtain a robust solution.

A.1.2 Decision tree learning

In this analysis the criterion for splitting decision tree nodes is based on the Gini Impurity Index (G) of the parent and potential child nodes. If the node under consideration consists of n observations in P classes, where each class corresponds to a value of the response variable, and n_p is the number of observations in the pth class, then the Gini Index G for that node is defined as:

$$G = \sum_{p=1}^{P} \frac{n_p}{n}\left(1 - \frac{n_p}{n}\right). \quad (1)$$

Based on this definition, G is minimized (and equal to 0) when all observations in the node are of the same class, and increases when the observations in the node are spread more evenly among the different classes. Other splitting criteria may be used in decision tree classification (e.g. statistical permutation tests), but they all represent a measure differentiating pure from well-mixed classes.

The Gini-based splitting process begins by computing an initial Gini Index for the *parent* node (that is, the node to be split). Then, for all predictor variables available, every unique binary

---

[2] Decision trees and Random Forests can also be used for regression problems when the response variable is continuous. However, in the use case described herein the response of interest is categorical; for simplicity, discussion is therefore restricted to classification problems throughout this appendix.

partition is considered—that is, splitting between the first and second lowest values of the variable, then between the second and third lowest values, etc. For each proposed split, samples with values below the split value are assigned to one new child node, and samples above the split value to a second child node. The Gini Index for each child node is then computed, and the final Gini score for the proposed split is the average of these two values, weighted by the number of observations in each node. After considering all possible variable and value splits, the one that makes the greatest reduction in the Gini Index is retained. The splitting process is then repeated until all nodes contain samples of a single class (and thus total Gini Index is 0).

A.2 Random Forests

A.2.1 Motivation of the Random Forests approach

While decision trees are very effective at modeling complex data relationships, they have two notable drawbacks. First, individual decision trees may have a tendency to be overfit—in other words, they conform too closely to the particular samples represented in the training dataset. While this makes their fit (in terms of R^2 or similar metrics) very good, it renders them less accurate at classifying new samples. In addition, decision trees on their own can only be used in classification problems; they cannot be applied to clustering problems like the one addressed in this study, where the class identities of training samples are unknown.

RF is an extension of the decision tree framework that addresses both of these limitations. Put simply, RF produces a collection of numerous decision trees (i.e., a forest), each based on a different random permutation of the training data. Results from all trees in the collection are averaged to make predictions, rather than allowing any one tree to dictate the analysis. The randomness in the procedure reduces overfitting, leading to more robust predictions. In addition, the RF algorithm provides a powerful measure of relatedness or *proximity* among samples. As discussed below, this last feature allows RF to be used in clustering problems.

This study employs the RF algorithm first introduced by Breiman (2001), which is the most widely used variant today. In his foundational introduction of the method, Breiman showed that it has accuracy at least as good as other ensemble decision tree algorithms, is robust to outliers and noise, and is computationally efficient compared to alternatives. The basic algorithm is implemented using the *randomForest* R package (Liaw *et al.* 2015), and the RF clustering approach is based on recommendations of Shi and Horvath (2006).

A.2.2 The Random Forests algorithm

Consider a dataset of N training samples consisting of the $N \times 1$ response vector of categorical class labels $\boldsymbol{y}$, along with K predictor variables contained in the $N \times K$ matrix $\boldsymbol{X}$. To create a model for predicting $\boldsymbol{y}$ from $\boldsymbol{X}$, the RF algorithm first creates M trees (where M is typically on the order of 10^3–10^4), with randomness injected among them at two different levels. First, each tree is constructed from a random draw of N observations selected with replacement from the training dataset. That is, for each tree a vector $\boldsymbol{\varphi}$ of N indices is first drawn randomly (with replacement) from the integers [1, N]. Then, the responses and predictors to be fit by the tree, $\tilde{\boldsymbol{y}}$ and $\tilde{\boldsymbol{X}}$ (respectively), are defined as:

$$\tilde{y} = \left[y_{\varphi_1}, y_{\varphi_2}, \ldots y_{\varphi_N}\right] \quad (2)$$

$$\tilde{\boldsymbol{X}} = \begin{bmatrix} x_{\varphi_1,1} & \cdots & x_{\varphi_1,K} \\ \vdots & \ddots & \vdots \\ x_{\varphi_N,1} & \cdots & x_{\varphi_N,K} \end{bmatrix} \quad (3$$

This scheme allows individual training samples to contribute more or less to model fit in each of the *M* random decision trees. This property prevents unusual samples or spurious relationships among a few particular samples from dominating the final outcome of the model.

The second random element in the Breiman RF algorithm enters at the node level. Unlike the single-tree analysis described above in Section A1.2, individual trees in an RF analysis are not formed by exhaustively searching for the best split at every node. Instead, for each split the algorithm considers K' predictor variables randomly selected from the *K* available predictors in $\boldsymbol{X}$ (typically, $K' \sim \sqrt{K}$), and chooses the best possible split from among these *K'* predictors. The result of this additional randomization is that different variables are considered in each tree and each node, which allows for a wider range of influences from each variable on the final model. For example, early splits on the most important predictors might dominate trees built with the full $\boldsymbol{X}$, consistently overwhelming substantial but comparatively subtle effects of one or more other predictors. By randomly omitting the dominant predictors from some splits, these effects will be evident in at least some of the trees in the forest.

A.2.3 Products of the Random Forests algorithm

A.2.3.1 Predictions

Once an RF model has been constructed, it can be used to make predictions of class membership for samples from the training data or other datasets. Given an observed sample with predictors $\boldsymbol{x}^*$, the class of the sample y^* (which may or may not be actually known) can be predicted by any individual decision tree by passing the sample through every split in the tree according to the values in $\boldsymbol{x}^*$. By following the splitting rules in the tree, the new sample will end up in the leaf containing the sample from the training data that it is most similar to. The class of that sample is thus the obvious prediction for the new sample's predicted y^*. When predicting from an RF model, the most common approach is to assign y^* to the most frequently predicted class across all trees in the forest (i.e. a "majority wins" vote for the predicted class). Alternatively, the proportion of votes for each class can be retained, and interpreted as probabilities of class membership that reflect uncertainty in the model.

RF predictions are clearly useful when data are available with known values for predictor variables, but whose class membership is unknown. Such scenarios arise when attempting to extrapolate or forecast in space or time, or when predictors are easy to measure but class membership is difficult to ascertain. In addition, predictions on the training set itself offer insights into the accuracy of RF classification, by comparing the predicted and known classes of observations in the training data to obtain various error metrics (e.g. percentage of samples classified incorrectly). This analysis can be conducted using the complete training dataset to produce a classical error estimate. Alternatively, predictions can be made for only those samples

that were omitted from each tree due to randomization[3], leading to an *out-of-bag* (OOB) error rate akin to cross-validation in other analyses.

A.2.3.2 Proximity

Another important output of an RF analysis—especially in the context of unsupervised learning problems—is the proximity matrix, which represents the similarity among observations in the training data. To construct this matrix, an $N \times N$ matrix $\boldsymbol{Q}$ is first initialized with zeros in every entry. After building each tree in the forest, any training sample used in constructing the tree will already have been placed into a leaf node; for the purpose of calculating $\boldsymbol{Q}$, the OOB samples are sorted through the tree as well, as if they were being predicted (*as described above*). Then, for each pair of samples the entry q_{ij} in $\boldsymbol{Q}$ is incremented by 1 if the i^{th} and j^{th} sample have been placed into the same leaf node. Finally, at the end of the analysis, the entire matrix is divided by the number of trees in the forest. In other words,

$$q_{ij} = \frac{1}{M} \sum_{m=1}^{M} I_{sameleaf}(i,j) \tag{4}$$

where $I_{sameleaf}(i,j)$ is an indicator function returning 1 if observations i and j share the same leaf node in tree m, and 0 otherwise.

As indicated, at the end of the RF algorithm, $\boldsymbol{Q}$ contains the proportion of trees in which each pair of observations shared a leaf node during the analysis. This matrix represents similarity among the samples that automatically takes into account the relationships among variables that have been identified by the RF model. Thus, this approach offers a powerful means to obtain a measure of proximity among samples that—like RF itself—reflects complex and nonlinear data structure, is robust to skewed or otherwise unusual data distributions, and has no requirements for the scale of the predictor variables.

A.3 Imputing missing data using Random Forests

Most clustering methods cannot include samples with missing values. Thus, in a real dataset where values are missing, typically entire rows and/or columns must be removed until all missing values have been omitted from the analysis. The clustering approach based on RF proximity described herein is no exception to this rule. However, the RF framework includes an approach for filling in or *imputing* missing data. This analysis employs a method introduced by Stekhoven and Buehlmann (2012) and implemented in the R package *missForest*. The approach has proven to be quite accurate in tests on real datasets with known values artificially removed.

[3] The distribution for the number of occurrences of each training sample in a single randomly constructed tree is *Binomial* (N, $1/N$). The probability of being omitted entirely from a given tree converges fairly rapidly to $e^{-1} \sim 0.37$ as N increases. Thus, in any particular tree in a Random Forests analysis approximately one third of samples are naturally left out by the randomization procedure.

Thus the potential inaccuracies from imputation are considered preferable to the guaranteed loss of information that results when missing values are simply omitted.

The first RF imputation algorithm was introduced by Breiman (2001) along with the RF method itself. In this approach, missing values in each $\boldsymbol{x}_{\cdot k}$ (i.e., the k^{th} column of $\boldsymbol{X}$) are filled in with an initial guess of the median or mode of $\boldsymbol{x}_{\cdot k}$ (for continuous or categorical $\boldsymbol{x}_{\cdot k}$, respectively). RF analysis is conducted using the completed $\boldsymbol{X}$, and then each filled-in (i.e. initially missing) entry is replaced by the proximity-weighted mean of all non-missing values for that variable. That is, the imputed value of a particular missing entry x_{ik} in $\boldsymbol{X}$ is updated to:

$$x_{ik} = \frac{\sum_{j \in j_{obs,k}} q_{ij} x_{jk}}{\sum_{j \in j_{obs,k}} q_{ij}} \tag{5}$$

where $j_{obs,k}$ is the set of indices of all non-missing values of $\boldsymbol{x}_{\cdot k}$, and the weights q_{ij} are entries in the proximity matrix $\boldsymbol{Q}$ (*see above*) resulting from the RF analysis. Thus, the samples that are most similar to $\boldsymbol{x}_{i\cdot}$ (i.e., the i^{th} row of $\boldsymbol{X}$), contribute the most to the imputed value for missing x_{ik}. After all imputed values have been thus updated, a new RF is constructed and the imputation process repeated. These iterations are repeated a user-specified number of times, and in practice tend towards stable imputed values within ~5 iterations.

A limitation of this approach is that it requires a response variable **y** with no missing values. This is problematic in unsupervised learning problems, which by definition have no response variable. This study therefore implements an updated variant introduced by Stekhoven (2013). The method again begins by initially filling in missing values for any variable with the median of the non-missing entries for that variable. Then, over multiple iterations, the imputed values for each variable are updated based on an RF model that attempts to predict those values from the other variables in the dataset. Specifically, in each iteration the following steps are performed:

1. Select a variable in $\boldsymbol{X}$ to update. Denote that variable $\boldsymbol{x}^*$, and denote the matrix of all other variables as $\boldsymbol{X}^*$.
2. Define the vector $\boldsymbol{\varphi}$ as the set of all indices to *non-imputed* values in $\boldsymbol{x}^*$ (i.e., the values that were not missing in the original dataset), and define the complementary vector $\boldsymbol{\varphi'}$ as indices to the *imputed* values in $\boldsymbol{x}^*$.
3. Define a training dataset in which the response variable is the set of non-imputed values in $\boldsymbol{x}^*$, denoted $\boldsymbol{x}^*_{\boldsymbol{\varphi}}$. The predictors are the corresponding rows of $\boldsymbol{X}^*$, denoted $\boldsymbol{X}^*_{\boldsymbol{\varphi}}$. Note that $\boldsymbol{X}^*_{\boldsymbol{\varphi}}$. may contain imputed values, but $\boldsymbol{x}^*_{\boldsymbol{\varphi}}$ consists of actually observed data by definition.
4. Use RF to construct a model for the training dataset, i.e., an RF model that uses $\boldsymbol{X}^*_{\boldsymbol{\varphi}}$. to predict $\boldsymbol{x}^*_{\boldsymbol{\varphi}}$ in the training data. Since RF handles any combination of continuous and categorical data, no special treatment of any of the variables is necessary.
5. Use the fitted RF to predict $\boldsymbol{x}^*_{\boldsymbol{\varphi'}}$ (the originally missing values in $\boldsymbol{x}^*$) from $\boldsymbol{X}^*_{\boldsymbol{\varphi'}}$. (the corresponding rows in $\boldsymbol{X}^*$, which again may contain both observed and imputed data). These predictions become the new imputed values for $\boldsymbol{x}^*$.
6. Repeat steps 1–5 for all variables in $\boldsymbol{X}$. Variables are updated in order from greatest to least number of imputed (originally missing) values.

These iterations are repeated until the imputed values converge, as indicated by a stopping criterion based on the difference between imputed values from one iteration to the next.

A.4 Random Forests for unsupervised learning

A.4.1 Rationale and overview

As discussed above, the RF algorithm is designed to model known class labels as a function of predictors, i.e. problems of *supervised learning*. However, RF (and some other supervised learning techniques) may also be used for *unsupervised learning* (clustering) problems. To do this, synthetic data are constructed with specific properties based on real samples from the unlabeled data of interest. The real and synthetic data are labeled as such, and combined into a single dataset. The RF algorithm is then run on this combined dataset in attempt to classify real vs. synthetic data on the basis of available predictors. In effect, the synthetic data serve as a null hypothesis about the relationships among predictor variables in the real dataset. If these relationships are strong, then the real and synthetic data will be easily distinguishable and the attempt to classify them using RF will have high accuracy. The proximity matrix generated by the RF will reflect these relationships, which can then be fed into additional analyses, such as clustering.

A.4.2 Generating synthetic data to facilitate unsupervised learning with the Random Forests algorithm

In order to serve as a meaningful null hypothesis, the synthetic data should have a similar distribution as the real data, but lack the covariance structure that ultimately indicates which samples are most similar. Thus one reasonable approach for generating the synthetic data is to draw randomly and uniformly from the hyperrectangle that bounds all predictor variable values in the real dataset. However, Shi and Horvath (2006) obtained better results from the simpler approach of randomly drawing synthetic observations from independent marginal distributions of each observed predictor. Thus, this latter approach is followed here.

Specifically, to generate a synthetic dataset corresponding to N real samples with known values for K predictors (but unknown class labels), the first step is to draw an $N \times K$ matrix of indices $\boldsymbol{\Phi}$ randomly (with replacement) from the integers [1, N]. A synthetic data matrix $\tilde{\boldsymbol{X}}$ may then be defined based on these indices and the $N \times K$ matrix $\boldsymbol{X}$ of real samples:

$$\tilde{\boldsymbol{X}} = \begin{bmatrix} x_{\phi_{11},1} & \cdots & x_{\phi_{1K},K} \\ \vdots & \ddots & \vdots \\ x_{\phi_{N1},1} & \cdots & x_{\phi_{NK},K} \end{bmatrix} \tag{6}$$

Thus, as desired the synthetic and real data have the same marginal distributions (each $\tilde{\boldsymbol{x}}_{\cdot k}$ has the same distribution as the corresponding $\boldsymbol{x}_{\cdot k}$ from which it was derived), but the former by definition will fail to capture any dependence structure among the real $\boldsymbol{x}_{\cdot k}$.

Finally, the complete dataset for RF analysis is constructed by appending the real and synthetic data into a $(2N) \times K$ predictor matrix $\boldsymbol{X}^*$, and generating a corresponding $(2N) \times 1$ class label vector $\boldsymbol{y}^*$:

$$\boldsymbol{X}' = \begin{bmatrix} \boldsymbol{X} \\ \widetilde{\boldsymbol{X}} \end{bmatrix} \tag{7}$$

$$\boldsymbol{y}' = [y_1 \;\; y_2 \;\; \ldots \;\; y_{2N}] \tag{8}$$

$$y_i = \begin{cases} 1, & i \leq N \text{ (i.e., } \boldsymbol{x}_i'. \text{ is a real data row)} \\ 0, & i > N \text{ (i.e., } \boldsymbol{x}_i'. \text{ is a synthetic data row)} \end{cases} \tag{9}$$

Since the RF modeling of $\boldsymbol{y}'$ as a function of $\boldsymbol{X}'$ depends on randomly chosen values in $\widetilde{\boldsymbol{X}}$, unsupervised learning results will differ from one analysis to the next depending on the particular realization of $\widetilde{\boldsymbol{X}}$. To avoid spurious outcomes, it is therefore recommended to repeat the analysis for multiple forests, each with a newly randomized $\widetilde{\boldsymbol{X}}$. Predictions of class membership and proximities can then be averaged across the multiple forests, avoiding undue influence of any one synthetic dataset. Shi and Horvath (2006) found that analysis of ~10 forests was sufficient to give robust results.

A.5 Random Forests proximity as a distance metric

The basis of any clustering method is a distance matrix, which contains the distances between every pair of samples being clustered. Conceptually, the distance metric used in this context can be any quantity that represents how dissimilar observations are. The most common metric is the Euclidean distance. Continuing from above, in which the data being analyzed are contained in an N x K matrix $\boldsymbol{X}$, the Euclidean distance between samples $\boldsymbol{x}_i$.and $\boldsymbol{x}_j$. (i.e., the i^{th} and j^{th} rows of $\boldsymbol{X}$) is defined as

$$d(\boldsymbol{x}_i., \boldsymbol{x}_j.) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - \boldsymbol{x}_{j2})^2 + \cdots + (x_{iK} - x_{jK})^2} \tag{10}$$

That is, Euclidean distance is the K-dimensional extension of Pythagoras' formula for physical distance measured in 2- or 3-dimensional space.

Euclidean distance is a common and intuitive metric on which to base cluster analysis, but it comes with one important caveat: it is not straightforward to compute when predictors include binary or factor data, or when they are measured on different scales. The former may be coded into numeric values, and the latter may be transformed in a variety of ways to normalize scale. However, these options require subjective choices that inevitably influence Euclidean distance calculations.

As an alternative to Euclidean distance, the RF proximity matrix $\boldsymbol{Q}$ can be converted to the dissimilarity matrix $\boldsymbol{Q'}$ by a simple transformation of each entry:

$$q'_{ij} = \sqrt{1 - q_{ij}} \tag{11}$$

This formulation[4] yields a matrix that meets all the criteria of a distance matrix—namely, all elements are nonnegative, and elements on the diagonal are equal to zero. Unlike other distance metrics, however, it inherits benefits of the RF approach including scale invariance, robustness to extreme values, and the ability to incorporate a mix of numerical and factor variables.

A.6 Clustering data by Partitioning Around Medoids

A.6.1 Overview and motivation

Mathematically, the RF dissimilarity matrix ***Q'*** described above is in fact a Euclidean distance matrix[5], which means that the simple and common *k*-means clustering method could be applied to it. However, this project implements the alternative Partitioning Around Medoids (PAM) approach. PAM is a variant of the *k*-medoids algorithm, and as suggested by the name it is conceptually similar to *k*-means, but more analogous to a statistical median than mean. In support of this analogy, the main advantage of PAM over *k*-means is that it is more robust to outliers, and thus should in general result in predictions (e.g., when assigning newly obtained samples to a background vs. non-background cluster) that are accurate more often. The drawback of PAM is that it relies on a computationally expensive algorithm, but it has been found to be tractable for this project.

The PAM algorithm creates clusters by seeking samples from a dataset that will serve as the central points, or *medoids*, for *k* distinct clusters. For any set of medoids, the remaining samples may be partitioned by adding them to whichever cluster they are nearest to, in terms of distance to the medoid. The distance metric can be any measure; this analysis uses the RF dissimilarity matrix ***Q'*** defined above. The goal of PAM is to identify clusters such that the total distance of all samples to their cluster's medoid is as small as possible.

A.6.2 Mathematical formulation

Formally, Kaufman and Rousseeuw (1987) defined the problem by letting ***u*** and ***V*** be binary variables such that:

$$u_i = \begin{cases} 1, & u_i \text{ is a medoid} \\ 0, & u_i \text{ is not a medoid} \end{cases} \tag{12}$$

[4] Some analysts omit the square root from the definition of elements in ***Q'*** (for example, the *randomForest* R package (Liaw *et al.* 2015) computes dissimilarity this way). It is unclear why this difference exists in the literature. This analysis uses the definition provided here, but in general it has very little effect on clustering outcomes (in this analysis, <0.5% of classifications change if the square root is omitted).

[5] This can be shown by demonstrating that ***Q*** is positive-semidefinite with entries in the range [0, 1] and diagonal elements equal to 1. For any such matrix, the transformation in Eqn. **(11)** yields a dissimilarity matrix that meets the definition of Euclidean (Gower & Legendre 1986).

$$v_{ij} = \begin{cases} 1, & v_{ij} \text{ is assigned to the cluster with medoid } u_i \\ 0, & v_{ij} \text{ is assigned to a different cluster} \end{cases} \quad (13)$$

Prescribing the rules:

$$\sum_i u_i = k \quad (14)$$

$$\sum_i v_{ij} = 1 \quad (15)$$

$$v_{ij} \leq u_i \quad (16)$$

In other words, there are exactly *k* clusters, each sample belongs to exactly one of them, and a sample's associated medoid can only be another sample that is in fact a medoid.

The goal is then simply to minimize the quantity

$$\sum_i \sum_j q'_{ij} v_{ij} \quad (17)$$

where q'_{ij} are entries in ***Q'*** representing the dissimilarity between samples *i* and *j*. This quantity represents the total dissimilarity for a given partitioning of the dataset.

A.6.3 The PAM algorithm for minimizing dissimilarity

For a given choice of *k* medoids, minimizing total dissimilarity is simply a matter of assigning each sample to the cluster whose medoid is least dissimilar to itself, which can be obtained directly from ***Q'***. Thus, the real computational challenge of the PAM algorithm is choosing the best set of medoids to use. In practice, this is achieved in two phases.

In the *build* phase of the PAM approach, initial guesses for the *k* medoids are made sequentially. The first medoid chosen is the one that minimizes the total dissimilarity between itself and all other samples. That is, letting i_1 denote the index of the sample selected as the first medoid, i_1 is chosen such that

$$\sum_j q'_{i_1 j} = \min_i \sum_j q'_{ij} \quad (18)$$

The second medoid is likewise chosen to minimize the sum of distances from each sample to the nearer of the two selected medoids, and so on until *k* medoids have been selected.

The *build* procedure is designed to find good initial medoid guesses, but it is unlikely to find the best set. Thus, in the second phase of the algorithm, called the *swap* phase, each medoid is considered in turn to assess whether it should be replaced by a different sample (i.e., set u_i for the currently selected medoid to 0, and set another u_i to 1 in its place). All possible swaps are considered, and the one that makes the greatest reduction in total dissimilarity is retained. The swap phase is repeated until there are no swaps for any medoid that would further reduce total dissimilarity.

A.6.4 Selecting the number of clusters

Finally, as with any cluster analysis, the choice of k is somewhat subjective. Various quantitative criteria may be used for selecting the number of clusters to use for any particular dataset, but none is universally considered a best practice and the different criteria may often lead to conflicting values for k. Moreover, an "automatic" method may suggest a number of clusters that is inappropriate for the problem at hand.

A preferable alternative approach is to select k based on *a priori* knowledge. In this project, for example, the goal is to separate two types of water samples—background vs. non-background—so the decision was made to partition the data into two clusters. A sensitivity test explored partitions with three and four clusters. However, this generally served only to subdivide the two clusters identified in the primary analysis. Assuming that all samples ultimately need to be classified as background or not (i.e., when there are more than two clusters, some must eventually be merged to form only two categories), partitioning the data further during the PAM clustering step in the analysis has little impact on the final results.