Created by Matt Norris on 12/08/15

Email: matthew@mdnorris.com

**Random Forest Technical Documentation for FDI\_Random\_Forest.R**

**Program Purpose**

Many of the methods I will use in this program come from this paper: (<http://zmjones.com/static/papers/rfss_manuscript.pdf>), which I will refer to frequently. As this paper states, machine learning, which typically predicts outcomes better than regression models, is seldom used in the social sciences because substantive interpretation of the relationship between variables is impossible. However, machine learning is especially useful for data exploration, and to some degree, it can be used for model verification.

**Comments on R**

R is open source, and most of its functionality is added through user created packages; for this reason, there is no central repository of R knowledge. If you cannot find a way to do something, Google it. You will probably find the R community has a solution, and often a package will be recommended to improve your program. Remember to Google any error messages and invest some time to understand these messages. As with any programming language, there is usually a way to shorten the code and run more efficiently, but readability is often more important (especially in small programs) than efficiency gains when collaborating on a project.

**Note: there will occasionally be methods in the syntax that should not be used when working with a very large dataset. If this program is ever modified for a large dataset, please modify the code accordingly.**

**Overview of Documentation**

This paper will assume some knowledge of programming in R. In conjunction with the comments in the program, it should possible to easily add variables and write in more analyses/functions. Each header in this paper will correspond with a “###### (Insert section name here) #######” section in the R syntax, and each section will explain the purpose of the syntax, as well as the justification of the methods used and supporting literature where possible.

**Creating Our Random Forest Model**

**Setting WD, Package Installation**

The working directory will be the default folder where files are read and written to. Temporary data is not written to the working directory (for large programs with sizable datasets, it may be a better practice to add in periodic save outs).

Many functions are built into basic R, but several of the functions used in the program require additional packages. A few packages may not be used for the final program, but do add functionality for future runs, so I left them in this section. The *library* and *require* functions load the packages for use. I will include a folder for package documentation.

**Data Prep**

A function is normally used to read a source file into an R data frame. Remember that different versions of a statistical program may save out a file that is not backwards compatible. I typically read in the full dataset and create an Excel file**\*** for easy inspection.

It is often extremely useful to create a string vector with variable names to subset your data. Here it is used to create a new data frame from the original data without having to type each in each variable name. It is much easier to add a variable name or range of variable names to the string vector than find it in each section of the program.

I will always create a random sample of the subsetted data for testing purposes. Do not forget to change the random sample to the full dataset when running your final analysis.

Many essential functions will not work with certain types of data. For example, most missing value imputation will not work with characters, so we must convert to factors.

**Eliminating Missing Values**

Random forests are finicky, and require some data manipulation before they can be run. Missing values must be imputed, and here I use two functions: *na.roughfix()* for quick runs and *mice()* for final runs.

*na.roughfix()* replaces missing values with the median or mode. It runs quickly, but it also performs poorly against multiple imputation methods (<http://repositorio.educacionsuperior.gob.ec/bitstream/28000/287/1/T-SENESCYT-0057.pdf> ), and from the literature (<https://cran.r-project.org/web/packages/CALIBERrfimpute/vignettes/simstudy_survival.pdf>) on multiple imputation methods, multiple imputation by chained equations (MICE) is arguably the best. *mice()* takes much longer to run, so use *na.roughfix()* for all but your final runs.

**Random Forest Model**

After your missing values have been imputed, you may have to change your variable type again. For example, *randomForest* can only handle 52 factor levels, but you can often convert factors to numeric or character data and manipulate afterwards.

I will let Jones provide a more in-depth explanation of classification and regression trees (CART) and the random forests that arise from them. The example of voting outcomes on pages (Jones, 4-6) is the best explanation of CART I have seen. For datasets with many variables, one tree may not determine the proper order of the splits; this means that the algorithm will take the best split at each level, regardless of the possibility that taking a poorer split might end in an overall better fit (Jones, 9). A random forest runs the CART process over small subsamples, with replacement, so that a better fit can be achieved by combining results. The implementation is fairly simple. We choose the outcome that we would like to predict, the predictors we will use for the splits, and the dataset on which we will run the random forest. Additionally, we will create proximity and importance matrices which will be used later.

I would like to note that I considered the conditional random forest method, which often produces less bias in the predictive model. It also has the advantage of handling more factor levels than a random forest, but we would lose the variable importance and proximity matrices. For this reason, I decided to use a random forest.

**Random Forest Variable Importance**

This is probably the most important section of the program. Node purity is a measure of correct assignment of observations to the final leaf of a tree. On the right plot, the order of the variables rank the gain in node purity from each variable in the model. Conversely, the left plot shows the increase in MSE if a variable is removed from the model. Variable importance of a random forest ignores collinearity and other potential interactions, providing insight into which variables to select for an interpretable model.

**Proximity Matrix**

If we run a regression tree on continuous data our response variable will not produce meaningful results, though we can still plot with a predictor variable (for code testing purposes, not for analysis). There is an example in the Jones paper of clustered observations formed from factor data. It’s important to note that any continuous variable, such as an FDI measure, can be converted to up to 52 ranked quantiles, which can then be used as factors in a proximity matrix. If specified correctly, the proximity matrix can reveal similar variables that frequently “travel” together down the tree.

**Partial Dependence Plots**

The last analysis this program performs is a partial dependence plot. This will create a plot that illustrates each individual relationship between an independent variable and the dependent variable. At the very least, this can show the behavior of the variable over the range of its values. Ideally, it will also reveal a non-linearity that can be transformed in the final modeling process.

**Examples, Testing, Sources, and Next Steps**

In the same folder as this documentation is a collection of plots from a full run of the program. These examples will also serve as a loose test of the current program; intuitive examination of these plots and comparisons to the Jones paper may reveal potential areas for improvement. I will also include a folder with many of the papers, articles, and forum posts from experienced R users that I used in the creation of this program, as well as the documentation for the packages we used.

There are two main areas to improve the model. The first is simply adding more variables and running a larger dataset against our response variables. A random forest can distinguish important variables despite how many predictors are included. Additionally, different subsets can be run, especially subsets that might make for interesting clusters in the proximity matrix.

Finally, feature engineering is needed for more accurate results. This is a great article describing the general purpose and practice of feature engineering: <http://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/>. Essentially, the dataset we are using can be transformed, arguments in functions can be improved, and new variables can be created to answer more questions than the initial data might be able to. I’ve included an empty section near the top of the program to add in engineered features.