8-2018

# A Comparison of R, SAS, and Python Implementations of Random Forests

Breckell Soifua
*Utah State University*

# A COMPARISON OF R, SAS, AND PYTHON IMPLEMENTATIONS OF RANDOM FORESTS

by

Breckell Soifua

A report submitted in partial fulfillment

of the requirements for the degree

of

MASTER OF SCIENCE

in

Statistics

UTAH STATE UNIVERSITY

Logan, Utah

2018

## Contents

**Abstract**

A Comparison of R, SAS and Python Implementations of
Random Forests

by

**Breckell Soifua**

Utah State University, 2018

Major Professor: Adele Cutler

Department: Mathematics and Statistics

The Random Forest method is a useful machine learning tool developed by Leo Breiman. There are many existing implementations across different programming languages; the most popular of which exist in R, SAS, and Python. In this paper, we conduct a comprehensive comparison of these implementations with regards to the accuracy, variable importance measurements, and timing. This comparison was done on a variety of real and simulated data with different classification difficulty levels, number of predictors, and sample sizes. The comparison shows unexpectedly different results between the three implementations.

**List of Figures**

**List of Tables**

A Comparison of R, SAS, and Python Implementations of Random Forests

# 1 INTRODUCTION

The Random Forest method is a useful machine learning tool introduced by Leo Breiman (2001). The method has the ability to perform both classification and regression prediction. Random forests are an improved extension on classification and regression trees (CART) (Liaw and Weiner, 2002) with respect to instability and accuracy. Specifically, random forests remain relatively stable with changes in data due to the combination of many trees. They inherit many advantages of CART. Random forests handle categorical predictors naturally, are computationally simple to fit, there are no formal distributional assumptions, can handle highly non-linear interactions and classification boundaries, perform automatic variable selection, and have the ability to handle missing values.

The Random Forest algorithm is as follows:

1. Grow a forest of *ntree* trees. For each tree from 1 to *ntree*, do the following:
    a. Take a sample of size *N* from the dataset with replacement (bootstrap) to grow a tree.
    b. Select *m* variables, independently for each node, at random out of *M* possible variables. The best split is chosen to split the node using the *m* selected predictors.
    c. Trees are grown until the nodes can no longer be split.

For classification, majority voting is used to get aggregated predictions of the *ntree* trees. For example, to classify a new observation using a random forest with 500 trees, the observation is passed down all 500 trees. Each tree predicts one of the classes and the observation is classified as the class which gets the most votes. Due to the nature of the bootstrapping in Step 1a, there will be approximately 30% of data points that are not

included in the training set. These are used as the “out-of-bag” (OOB) data to measure the accuracy of the model. (Breiman, 2001). In particular, each of the observations will be OOB in around 30% of the trees. The observation is passed down these trees and it is classified as the class which gets the most votes, giving the OOB prediction for the observation. The error rate of all the OOB predictions is the OOB error rate of the forest.

Random forests compute the importance of variables in two different ways. For classification problems, they use the Gini criterion to measure variable importance. For a given tree, the Gini variable importance measure for a particular variable of interest is the weighted average of the decrease in the Gini impurity criteria of the splits based on this variable. This is averaged over the *ntree* trees in the forest to get the Gini importance for the forest. Important variables yield high Gini variable importance measures. The other variable importance calculation is called permutation importance. Permutation importance is based on predictive accuracy. The out-of-bag error rate is computed from both a data set obtained from permuting the values of a particular variable of interest in the OOB data and the original OOB data. The difference between these two OOB error rates gives the permutation variable importance. Important variables yield high permutation variable importance measures. (Boulesteix et al. 2012)

R is a free, open-source software programming environment containing multiple packages with the ability to run Random Forest programs. The ‘randomForest’ package in R is most commonly used among all Random Forest implementations. Leo Breiman and Adele Cutler authored the original randomForest code in Fortran. The R port based on Breiman and Cutler’s code was authored by Andy Liaw and Matthew Weiner (Liaw and Weiner, 2018). The ‘randomForest’ package in R is currently maintained by Liaw. All other implementations of Random Forest refer Breiman and Cutler’s original code as a standard source.

SAS is a commercial software suite developed to perform analytics which is known for its ability to be used in a variety of disciplines. SAS is often preferred for regulatory work and has long been the standard for government work. SAS is optimized for big data using binning which reduces computation time and expense. While SAS is neither free nor open-source, it is still widely used commercially because of its robustness. SAS updates are rare but thorough which is seen as an advantage over the free, open-source programming media. The SAS procedure 'HPFOREST' is used when implementing the Random Forest algorithm. (SAS Institute, 2016)

Python is a free, open-source software programming environment commonly used in web and internet development, scientific and numeric computing, and software and game development. The 'scikit-learn' package is a machine learning package in Python with an implementation of random forests. (Pedregosa et al., 2011)

Boulesteix et al gave a brief overview of some implementations of random forests and their applications in computational biology and bioinformatics. Short descriptions and main features of each implementation are given but limited code is provided for only the standard R implementation. This overview is catered mainly to random forests in bioinformatics and omits other popular implementations, namely those in the SAS and Python languages. (Boulesteix et al., 2012)

## 1.1 IMPLEMENTATIONS

Among the many implementations of random forests, the most popular is the original R version. However, the SAS and Python versions are used often enough to motivate some form of comparison in performance. Assuming R's *randomForest* is used as a benchmark source, the performances should ideally be similar among implementations.

R's *randomForest* is well-documented and built to handle different problems naturally. This package has the ability to perform regression and classification automatically depending on if the specified response variable is of the class 'factor' or otherwise. One advantage of *randomForest* is the ability to handle character variables automatically as factor variables. This is useful as doing this conversion by hand is time consuming. This package outputs, by default, a confusion matrix for classification problems and out-of-bag error rates for each forest that is grown. The default output from this package is concise and useful while more detailed results are easily accessible if needed. These detailed are well-documented in the *randomForest* package documentation. (Liaw and Weiner, 2018)

SAS's *HPFOREST* is well-documented on the SAS website. The documentation for this package, and most other SAS packages, includes descriptions of parameters and options within the procedure. Examples of common uses for the procedure are given. Options for procedure output are extremely thorough with many options to reduce output as well. Many fit statistics offered such as: average squared error (regression) and OOB misclassification rates (classification) which was of particular interest.

Python's *scikit-learn* package is the most poorly-documented of the three implementations being compared. The vernacular used in the official documentation appears to be inconsistent with traditional statistical vernacular which lead to lack of clarity when comparing the three implementations. However, variable importance measures, misclassification rates, and other useful attributes of the forest are easily accessible.

As mentioned before, R's *randomForest* has the ability to take any character variable and convert it to a factor variable automatically. Additionally, R has a useful function *as.factor* that allows the user to convert a variable from a non-factor class to the factor class manually. In contrast, Python's *scikit-learn* does not have the automated

character-to-factor conversion nor the native ability to perform that conversion manually. This created problems in data sets used that have qualitative variables that need to be converted to factors to use in the RF algorithm. Python's *scikit-learn* package needs dummy variables to handle factor variables. We used R to create *k* dummy variables for a *k+1* level factor variable. New datasets with dummy variables included were created and used among the different implementations for consistency. Table 1 in the next section shows the original number of predictors and the number of predictors in the new datasets where levels of qualitative variables were converted to dummy variables. While this issue was resolved easily in R, this has the potential to be computationally expensive for users with different backgrounds/abilities.

Additionally, it should be noted that there is a learning curve present for users unfamiliar with Python programming. Assuming readers are familiar with the more common statistical languages (R and SAS), Python is different from both and is not designed for statistical analyses. Python does not have native support for data frames, but data frame manipulation can be done through the *pandas* package.

## 2 METHODOLOGY

Three measures were taken to compare the three implementations of random forests: accuracy, variable importance, and timing. These measures were taken on real and simulated datasets of varying sizes. Eight datasets were chosen from the UC-Irvine Repository to perform classification tasks using random forests. Additionally, twenty-two simulated datasets were created and used to perform classification using random forests. The R package, *mlbench*, was used to create these simulated data. They are named according to the distribution used to create them. They vary by sample size and number of predictors. Datasets named *TNSimX* were created using the *mlbench.twonorm* function. Datasets named *RNSimX* were created using the

*mlbench.ringnorm* function. Three implementations of random forests were used on these thirty datasets in order to compare performance: *randomForest* (R), *PROC HPFOREST* (SAS), and *scikit-learn* (Python). The datasets are described in the table below.

*Table 1. Datasets used in the comparison. "Extended Predictors" gives the number of predictors after converting factors to dummy variables.*

| Dataset | Observations | Predictors | Extended Predictors |
|---|---|---|---|
| Adult | 32561 | 14 | 100 |
| CreditCard | 30000 | 23 | 23 |
| Seizure | 11500 | 178 | 178 |
| Vegas | 504 | 20 | 111 |
| BreastCancer | 699 | 10 | 80 |
| Glass | 214 | 9 | 9 |
| DNA | 3186 | 180 | 180 |
| Satellite | 6435 | 36 | 36 |
| TNSim1 | 50 | 25 | 25 |
| TNSim2 | 100 | 25 | 25 |
| TNSim3 | 500 | 25 | 25 |
| TNSim4 | 1000 | 25 | 25 |
| TNSim5 | 50 | 100 | 100 |
| TNSim6 | 100 | 100 | 100 |
| TNSim7 | 500 | 100 | 100 |
| TNSim8 | 1000 | 100 | 100 |
| TNSim9 | 5000 | 25 | 25 |
| TNSim10 | 5000 | 100 | 100 |
| TNSim11 | 10000 | 25 | 25 |
| RNSim1 | 50 | 25 | 25 |
| RNSim2 | 100 | 25 | 25 |
| RNSim3 | 500 | 25 | 25 |
| RNSim4 | 1000 | 25 | 25 |
| RNSim5 | 50 | 100 | 100 |
| RNSim6 | 100 | 100 | 100 |
| RNSim7 | 500 | 100 | 100 |
| RNSim8 | 1000 | 100 | 100 |
| RNSim9 | 5000 | 25 | 25 |
| RNSim10 | 5000 | 100 | 100 |
| RNSim11 | 10000 | 25 | 25 |

# 3 RESULTS

We present results in terms of accuracy, variable importance, and timing. All timing results are performed on the same machine.

## 3.1 ACCURACY

The out-of-bag error rate was recorded and used to compare the accuracy of the implementations. To ensure robustness, each dataset was run twenty times and an average of the out-of-bag misclassification rates was taken. Figures 1 through 4 show a comparison of the average out-of-bag error rate for the simulated data. The simulated data were used because the level of difficulty of the classification problem is comparable between the datasets of different sizes, whereas the real datasets yield classification problems with incomparable difficulty levels. Figure 1 shows the out-of-bag error rates for two-norm simulated datasets with 25 predictors and varying sample sizes.

In general, we observe that the error rates on the two-norm and ring-norm datasets are comparable, with the R implementation doing somewhat better than the others for two-norm and somewhat worse than the others for ring-norm.

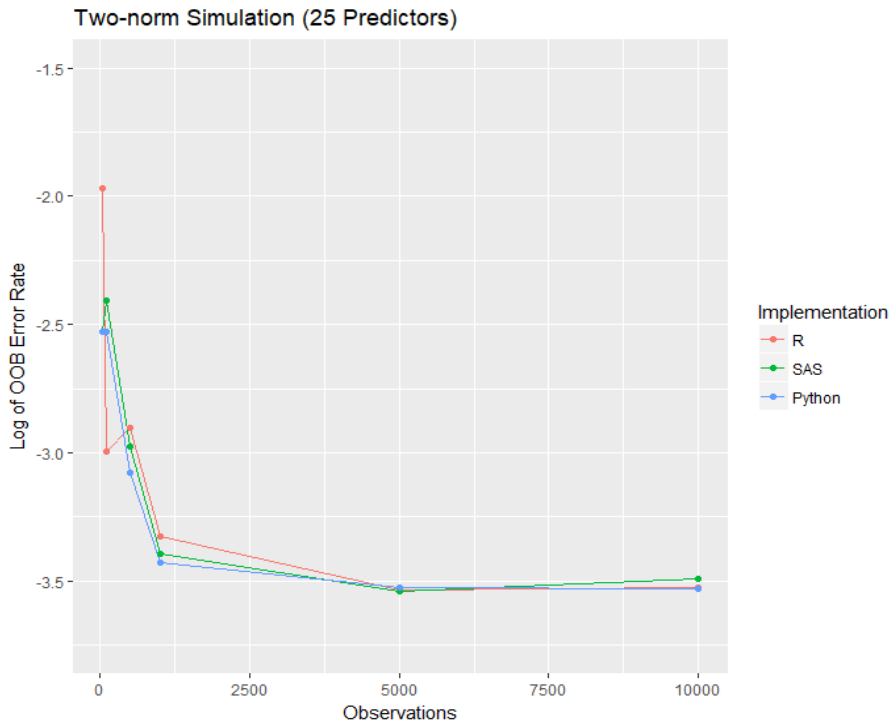



*Figure 1. Log of the out-of-bag error rate by observation size for all implementations. Two-norm simulation datasets with 25 predictors were used.*

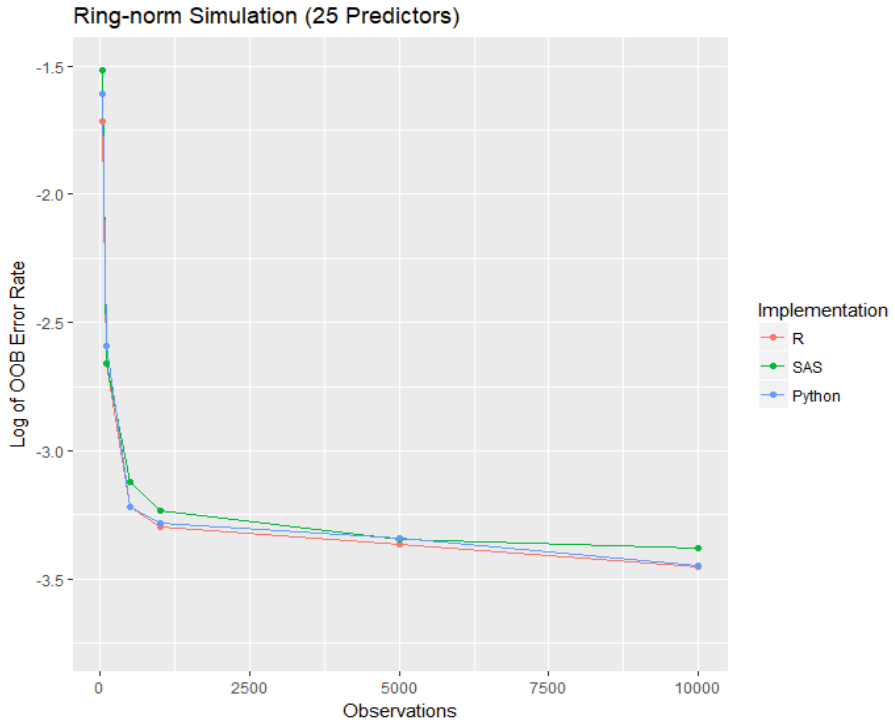



*Figure 2. Log of the out-of-bag error rate by observation size for all implementations. Ring-norm simulation datasets with 25 predictors were used.*

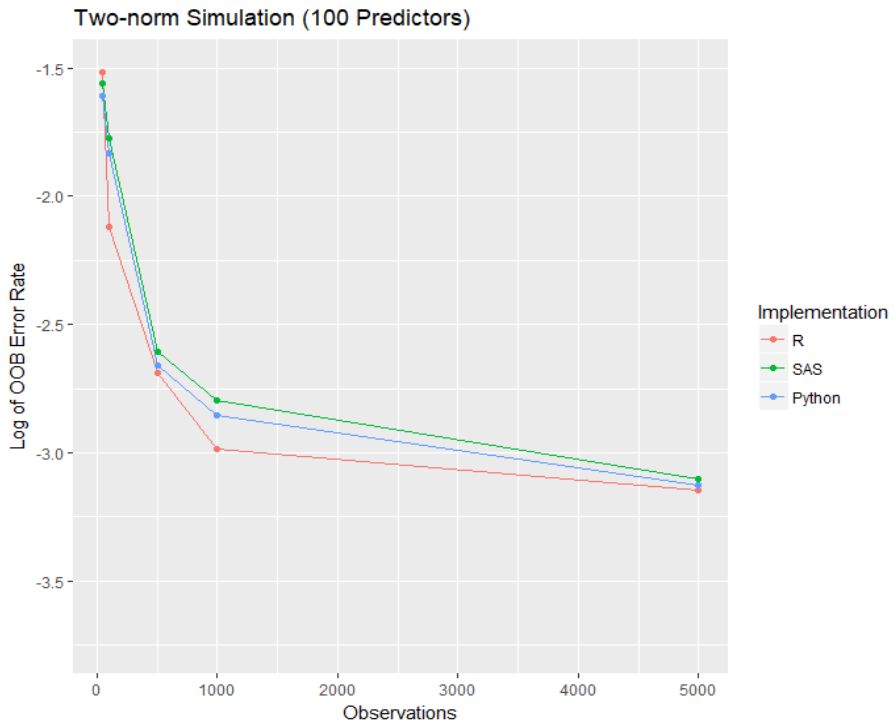



*Figure 3. Log of the out-of-bag error rate by observation size for all implementations. Two-norm simulation datasets with 100 predictors were used.*

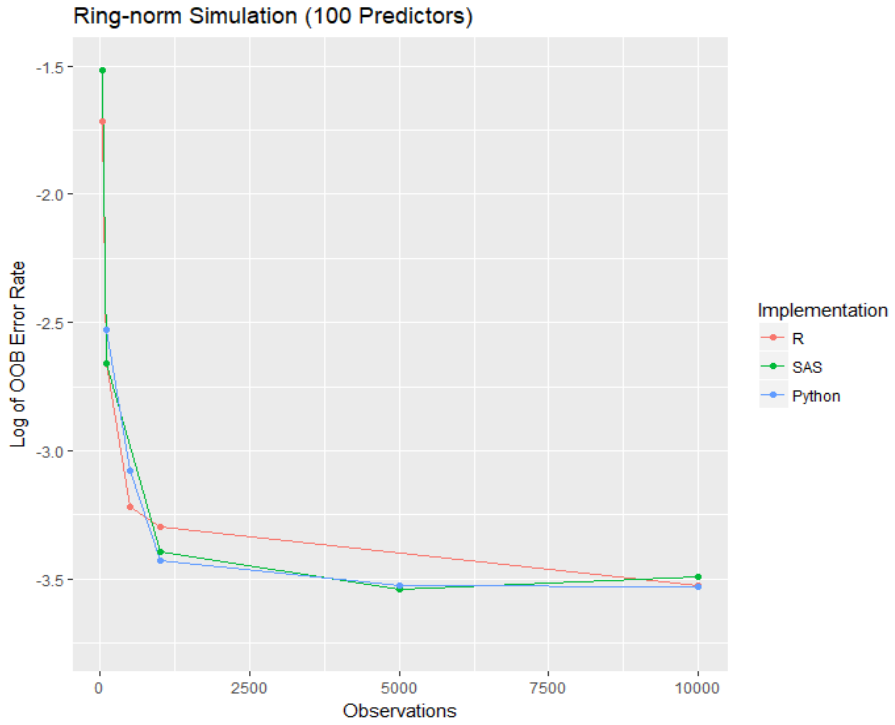



*Figure 4. Log of the out-of-bag error rate by observation size for all implementations. Ring-norm simulation datasets with 100 predictors were used.*

### 3.2 VARIABLE IMPORTANCE

Variable importance was found using the Gini criterion and the fifteen most important variables were investigated for each implementation. R's top fifteen variables were used as the standard to compare to. It was expected that variables considered important in R would have the same degree of importance in SAS and Python. Variable importance between the three implementations was visualized on the *breastcancer* data set from the UCI Data Repository. A real-data example was chosen because all variables are identically distributed in the simulated datasets, so the variable importance measures are unstable. All results are the median of 20 repetitions. In the figures below, it is apparent that all three methods agree for most of the variables and R disagrees with both SAS and Python for four variables.

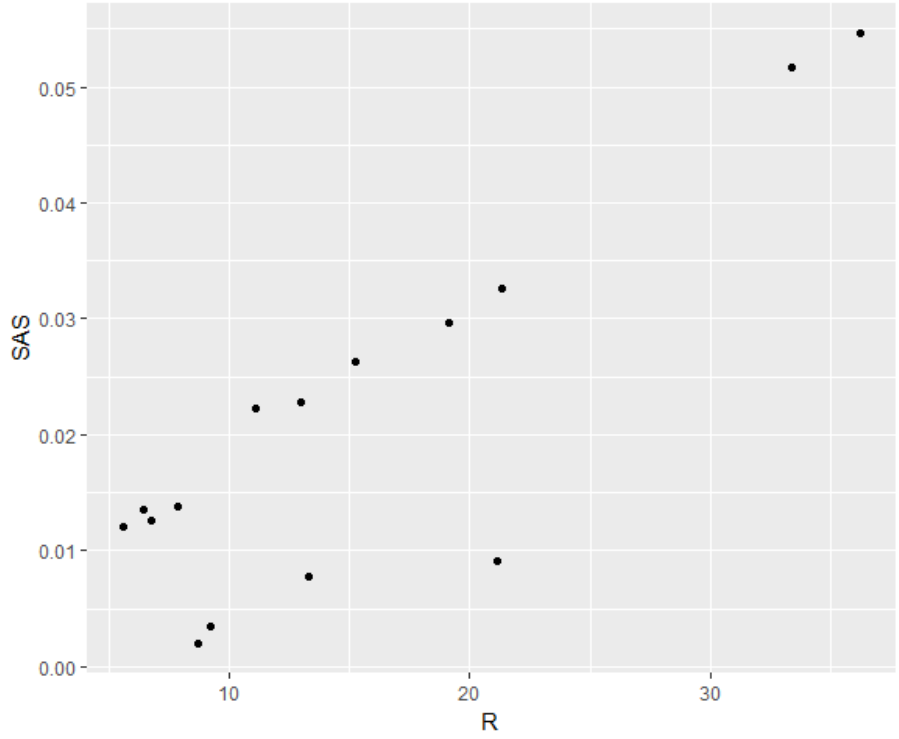



*Figure 5. Gini variable importance measures on the breastcancer dataset for R and SAS implementations. Each point represents the average Gini measure over 20 runs of random forests.*

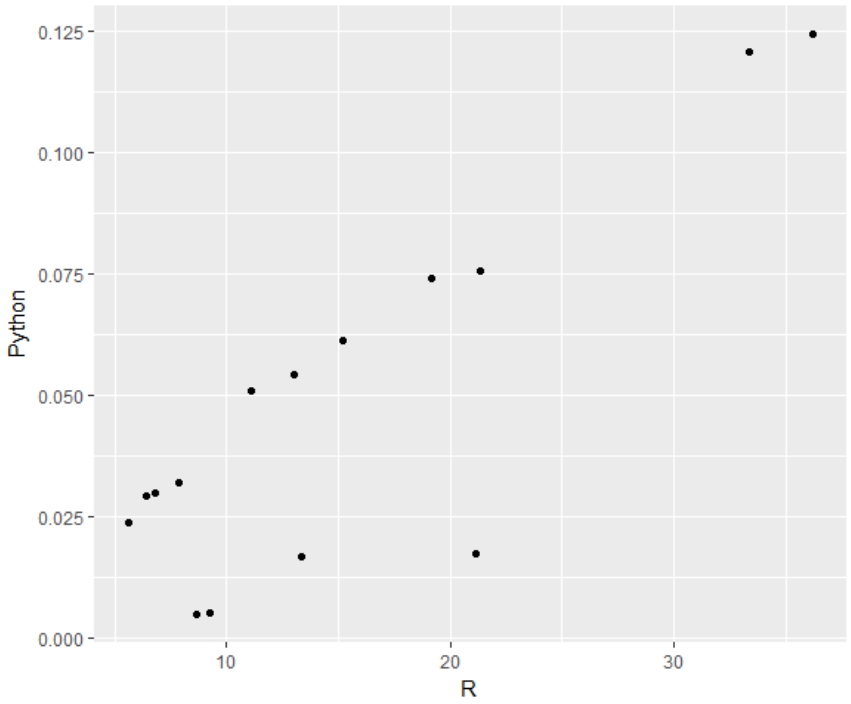



*Figure 6. Gini variable importance measures on the breastcancer dataset for R and Python implementations. Each point represents the average Gini measure over 20 runs of random forests.*

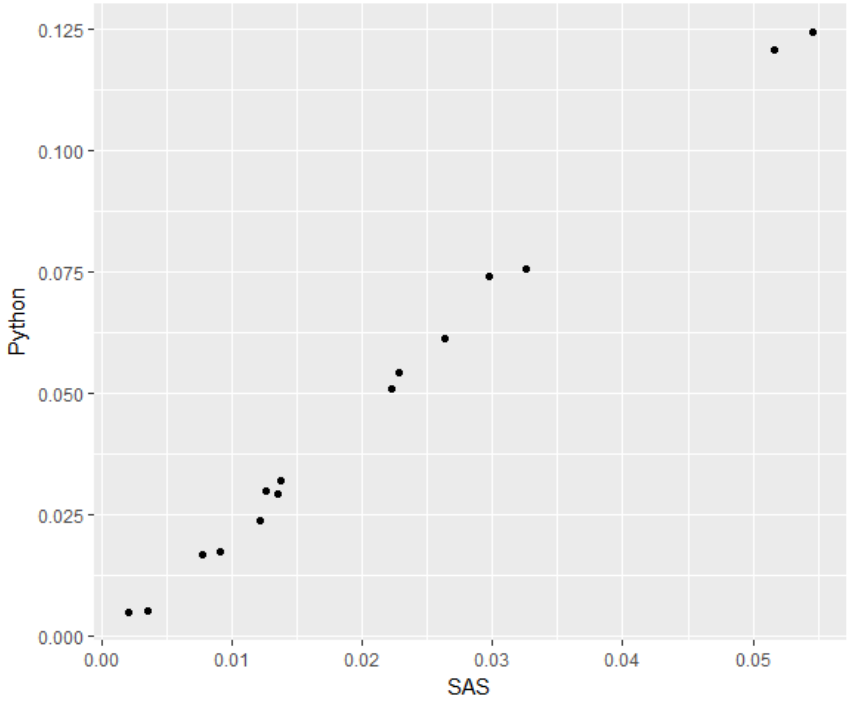



*Figure 7. Gini variable importance measures on the breastcancer dataset for SAS and Python implementations. Each point represents the average Gini measure over 20 runs of random forests.*

It seems more reasonable to conclude that SAS and Python chose the correct "important" variables and R does not choose the correct "important" variables, than SAS and Python both choosing the incorrect "important" variables and R choosing correctly. The four variables disagreed upon by R were investigated further by running random forests after excluding R's four "least important" variables then again excluding SAS and Python's four "least important" variables. We will refer to the former data as $RF_A$ and the latter as $RF_B$. Table 2 shows the average OOB error rate for $RF_A$ and $RF_B$.

*Table 2. Out-of-bag error rates for Breast Cancer data after removing the least important variables according to Python/SAS ($RF_A$) and after removing the least important variables according to R ($RF_B$). Each model has 76 predictors out of the original 80.*

| Data | Out-of-bag error rate |
|---|---|
| $RF_A$ | 4.056% |
| $RF_B$ | 4.263% |

These results agree with the results in the graphs above and suggest that Python and SAS are measuring variable importance in a better way. We cautiously conclude that there may be an issue with R's Gini variable importance calculations in the *randomForest* code.

### 3.3 TIMING

The computation time for each implementation was measured on the two-norm and ring-norm simulated data of different sizes. The following graphs show the computation times for datasets with 25 and 100 predictors and multiple observation sizes. These times were averaged over 20 repetitions.

SAS was consistently the slowest implementation. Note that SAS Studio via SAS OnDemand was used and is likely to be slower than another version of SAS like SAS Enterprise Miner. Additionally, all implementations would have been faster on a faster

machine. R and Python were both faster than SAS, with R being faster than Python in three of the four simulation studies.

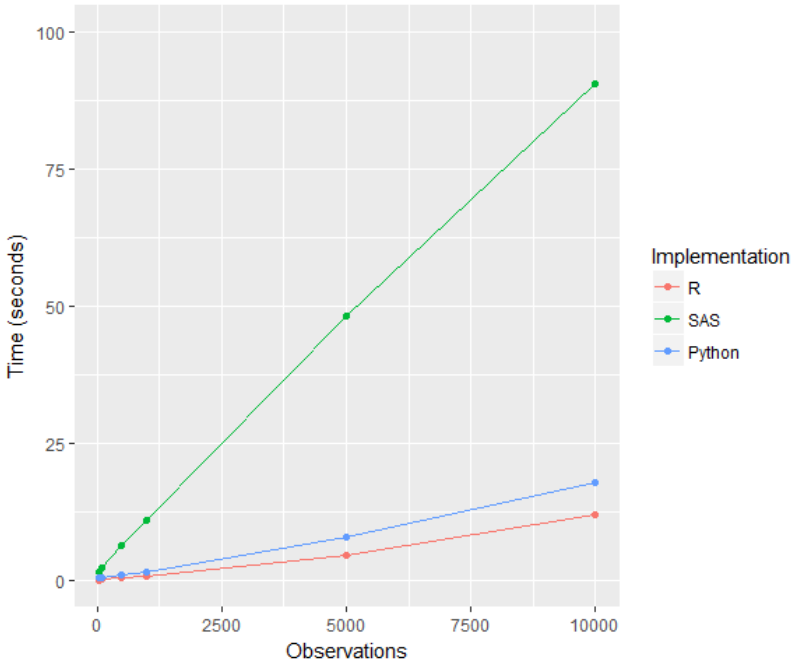



*Figure 8 (above). Computation time in seconds for two-norm simulated data with 25 predictors. Figure 9 (below). Computation time in seconds for two-norm simulated data with 100 predictors. Notice the difference in the horizontal axes.*

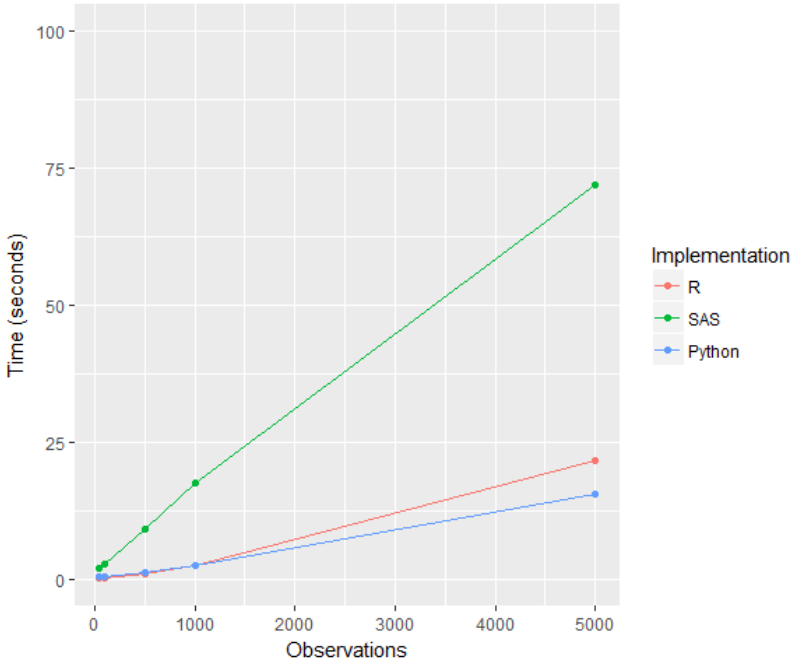



*Figure 9*

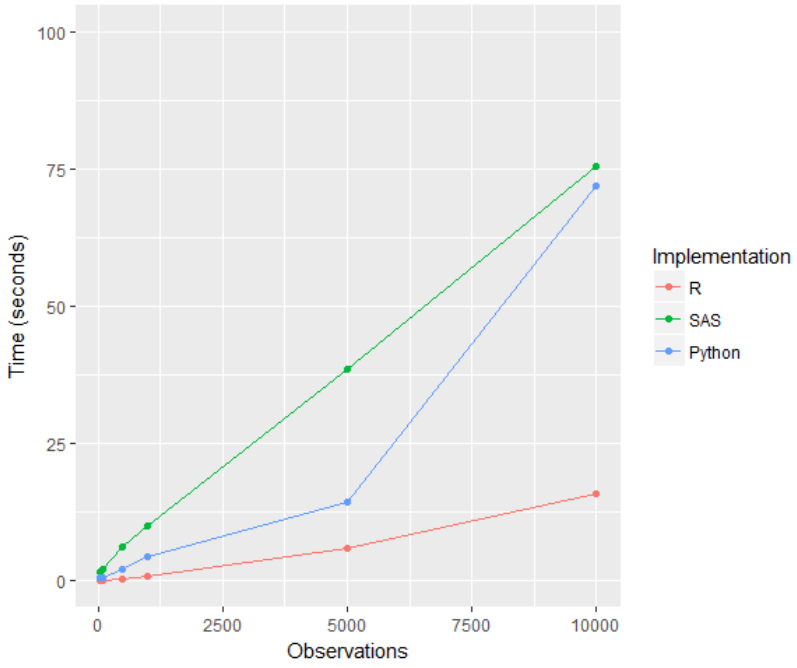



*Figure 10 (above). Computation time in seconds for ring-norm simulated data with 25 predictors. Figure 11 (below). Computation time in seconds for ring-norm simulated data with 100 predictors. Notice the difference in the horizontal axes.*

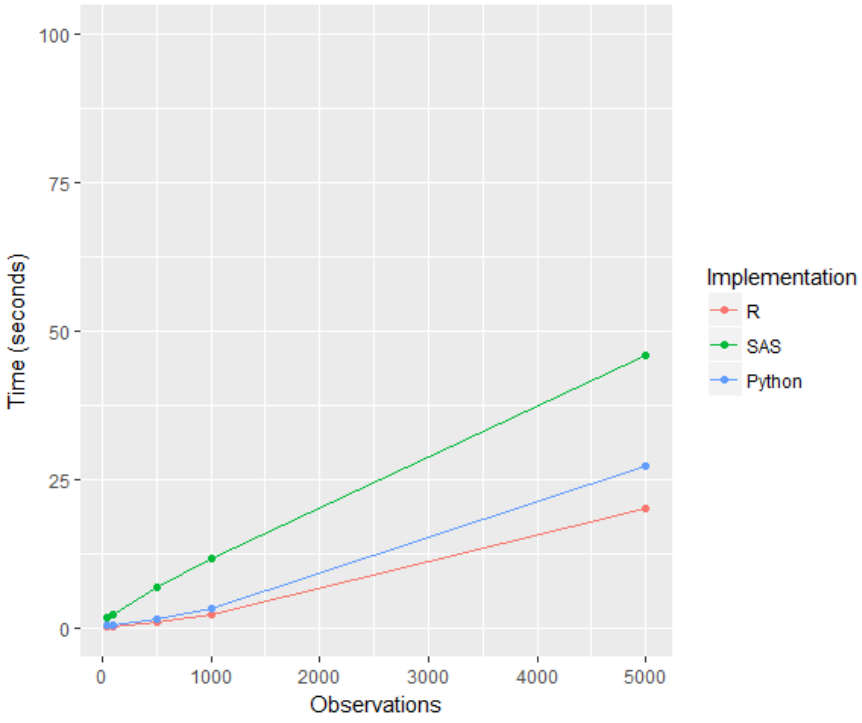



*Figure 11*

## 4 CONCLUSION

Random forests are a useful tool that can be used on many programming platforms. Implementations are not limited to R, arguably the most common statistical programming language. In terms of predictive accuracy, SAS and Python's implementations of random forests are satisfactory in comparison to R's *randomForest*. We discovered a discrepancy between the Gini importance of SAS/Python and that of R. While all three implementations agree on most of the variables studied, R gave a larger variable importance than Python and SAS for four variables. A closer inspection revealed that R's Gini importance may not be performing as well as that of the other two.

R's *randomForest* is fast, user-friendly, and has high accuracy. In general, this implementation recommends important variables correctly but use of caution is advised as mentioned previously. SAS's *PROC HPFOREST* is the most time consuming but has moderate accuracy. This implementation recommends important variables correctly and has detailed, useful output options. Finally, Python's *scikit-learn* random forest implementation is always computationally quick and chooses correct important variables.

## APPENDIX A: CODE

R Code:

```
# random forest on breast cancer data

# load packages needed

library(randomForest)

BC <- read.csv("breastcancer.csv")

start <- Sys.time() # start time

RF1 <- randomForest(formula = y ~ ., data = BC, importance = TRUE)

end <- Sys.time() # end time

end - start # print calculation time


# view variables in decreasing order of importance

imp <- as.data.frame(importance(RF1))

imp[order(imp$MeanDecreaseGini, decreasing = TRUE),]
```

SAS Code:

```
FILENAME REFFILE '/filepath/breastcancer.csv';

PROC IMPORT DATAFILE=REFFILE

        DBMS=CSV

        OUT=WORK.BC;

        GETNAMES=YES;

RUN;

PROC HPFOREST DATA = BC MAXTREES = 500 SEED = 14561;

TARGET Y / LEVEL = BINARY;

INPUT B: M: C: E: N:;

ODS OUTPUT FITSTATISTICS = BCFITSTATS(RENAME = (NTREES = TREES));

RUN;
```

```
/* Timing information can be found in the log as ‘real time’ listed after the PROC HPFOREST
code */

DATA BCFITSTATS;
        SET BCFITSTATS;
        LABEL TREES = 'NUMBER OF TREES';
        LABEL MISCALL = 'FULL DATA';
        LABEL MISCOOB = 'OOB';
RUN;
```

Python Code:

```
# import packages

import pandas as pd
import time
from sklearn.ensemble import RandomForestClassifier
# read in data
bc = pd.read_csv('breastcancer.csv')
# tell Python what the response variable is
bc_Y = bc.pop("y")
rnd_clf = RandomForestClassifier(n_estimators=500,oob_score=True,criterion='gini')
# calculate computation time
start = time.time() # start time
bc_rf = rnd_clf.fit(bc, bc_Y)
print(f'Out-of-bag score estimate:{1-rnd_clf.oob_score_:.3}')
end = time.time() # end time
print(end - start) # print calculation time
```

```
# variable importance measures
bc_varimp = rnd_clf.feature_importances_
headers = ["name", "score"]
values = sorted(zip(bc.columns, rnd_clf.feature_importances_), key=lambda x: x[1] * -1)
# view variables in decreasing order of importance
print(values, headers)
```

## APPENDIX B: WEIGHTED RANDOM FORESTS

The research in this report is the second topic we looked at. The first topic, which took almost nine months, was to try to improve random forests by weighting the trees.

Winham et al. (2013) introduced the Weighted Random Forest, which used the idea of weighting the trees in a random forest using their OOB error rate. The trees were grown as usual, and after growing all the trees, a weight was assigned to each tree. The idea was that by increasing the weights of the trees with low OOB error rates, and decreasing the weights of the trees with high OOB error rates, the Weighted Random Forest could perform better in terms of prediction. They compared different weighting schemes and found that the best weighting they tried was to weight the $j^{th}$ tree by $\text{rank}(1/OOB_j)$ where $OOB_j$ is the out-of-bag error rate for tree j. Unfortunately in their simulations this weighting gave an improvement of only 0.5% and Winham et al. (2013) concluded that the improvements were too modest to be worthwhile in their context (looking for interactions in very high-dimensional genetic data).

*Weighted Random Forests (Winham et al. 2013).*

1. Fit a random forest to the data in the usual way and record the OOB error rate of the tree, $OOB_j$, for the $j^{th}$ tree.
2. Assign weights $w_j = \text{rank}(1/OOB_j)$.
3. The Weighted Random Forest predicts by combining the trees in the random forest from Step 1, using weighted voting (classification) and weighted averaging (regression) with votes $w_j$.

When considering why such a commonsense weighting scheme was not successful, we reasoned that the weights penalized trees with high OOB error. This does not

necessarily mean a tree fits badly, it may just mean that the OOB data for that tree are unusually difficult to predict. So we designed a new weighting method.

*Fairly Weighted Random Forests (Soifua and Cutler 2018).*

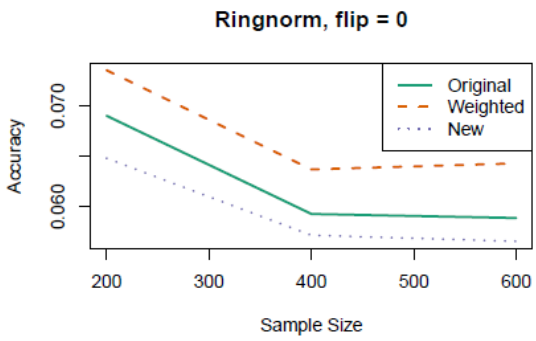
1. Divide the data into 10 random subsets (folds) as in 10-fold cross validation. For each fold k:
    a. Fit a random forest in the usual way to the other 9 folds and record the error rate of the $j^{th}$ tree on fold k. Call this error rate $kCV_j$.
    b. Assign weights $kw_j = \text{rank}(1/kCV_j)$.
2. Compute the prediction on a test set by passing the test set down all 10 random forests from Step 1a and by voting (classification) or averaging (regression) using the weights $kw_j$ for each k and j.

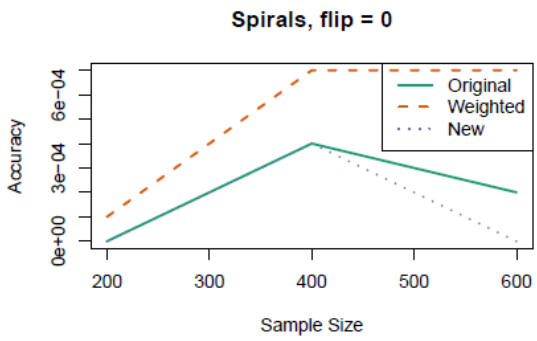
Note that the Fairly Weighted Random Forest method computes the weights for the trees on a common test set (the fold of the data not used to fit the random forest) so that the weights are not confounded with the difficulty of the OOB data. Our initial experiments with the method were not promising, so we moved to the topic presented in the main body of this report. As a public service, we recently completed a more comprehensive set of simulations, which showed a lot more promise, so we include them here.

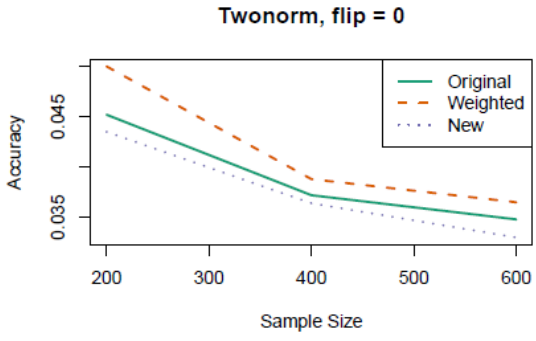
Simulations were performed using R (R Core Team, 2013). R's randomForest package was used for the simulations (Liaw and Weiner, 2002). Simulations were performed using 6 data sets from the R package mlbench (Leisch and Dimitriadou 2010, Newman et al. 1998). The datasets are ringnorm, twonorm, threenorm (each in 20 dimensions), Spirals, 2dnormals and circle (each in 2 dimensions). All of these problems are 2-class problems. Descriptions can be found in the UCI Repository (Newman et al. 1998). Sample sizes of 200, 400 and 600 were used, with a test set of 5000. Default

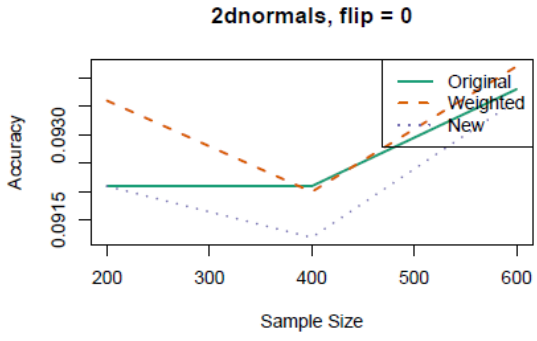
parameters were used for randomForest. To add more challenge, we flipped the signs of the two classes randomly with probabilities 0, 0.1, and 0.2 in both the training and test sets. All results were averaged over 20 runs. The results are summarized in Figures A1, A2 and A3. In these figures we refer to the Weighted Random Forest method (Winham et al. 2013) as “Weighted” and the Fairly Weighted Random Forest method as “New”. The “original” method is unmodified random forest.

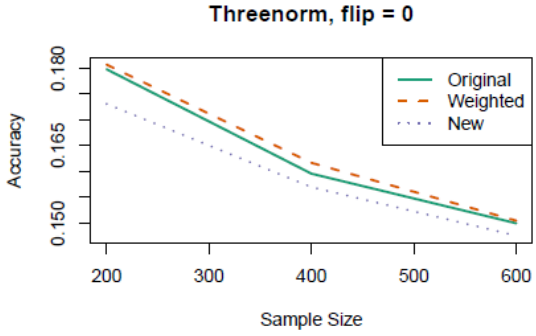
The results show that for the 20-dimensional problems (ring-norm, two-norm and three-norm) Fairly Weighted Random Forests do the best for all sample sizes and all levels of flipping. For the circle problem, Fairly Weighted Random Forests either do the best, or are very close to the best (flip = 0). For the other 2-dimensional problems (spirals, 2dnormals) the patterns are unstable. In fact, these problems do not always exhibit a monotonic nonincreasing pattern as the sample sizes increase, which is somewhat alarming. We probably should have done more repetitions but the simulations took more than 48 hours on a 3.6 GHz Intel Core i7-4790. In fact, Fairly Weighted Random Forests show as much as a 6.5% improvement over random forests (no weighting) for ring-norm, 5.2% for two-norm and 3.7% for three-norm. These results are considerably better than Weighted Random Forests and we plan to explore this new algorithm more thoroughly. One severe disadvantage of the current algorithm is that it fits 10 times as many trees as the other two methods. On the other hand, it is still lightning-fast compared to neural networks.

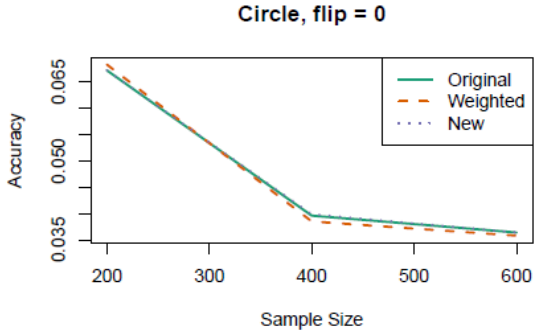

Ringnorm, flip = 0
Accuracy
0.060
0.070
Original
Weighted
New
200
300
400
500
600
Sample Size

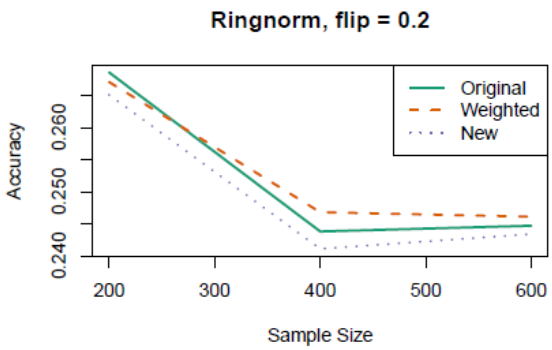

Spirals, flip = 0
Accuracy
0e+00
3e−04
6e−04
Original
Weighted
New
200
300
400
500
600
Sample Size

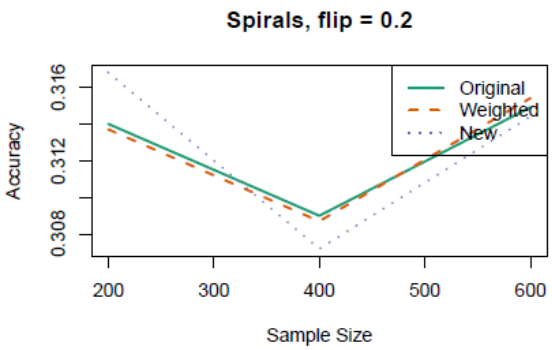

Twonorm, flip = 0
Accuracy
0.035
0.045
Original
Weighted
New
200
300
400
500
600
Sample Size

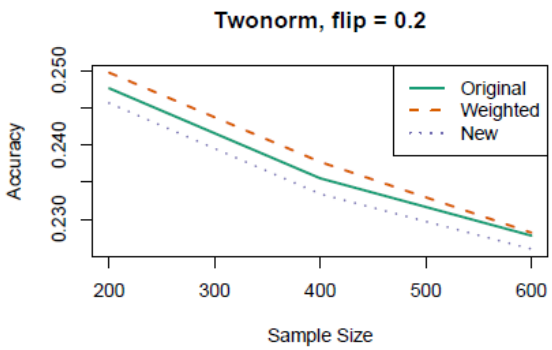

2dnormals, flip = 0
Accuracy
0.0915
0.0930
Original
Weighted
New
200
300
400
500
600
Sample Size

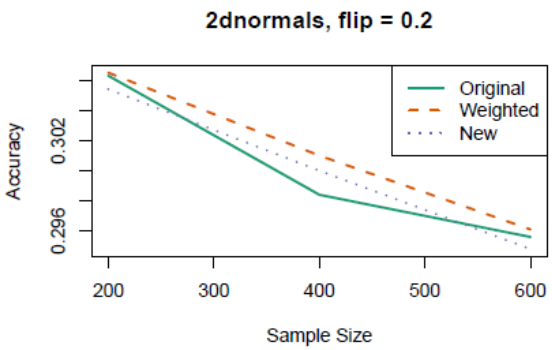

Threenorm, flip = 0
Accuracy
0.150
0.165
0.180
Original
Weighted
New
200
300
400
500
600
Sample Size

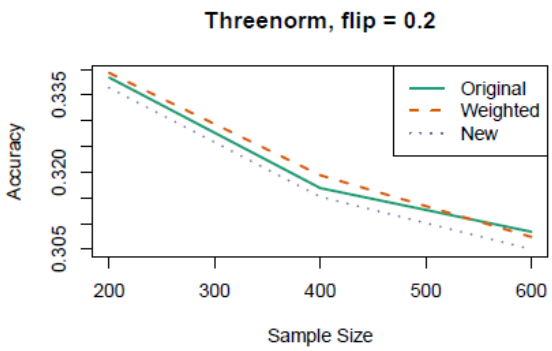

Circle, flip = 0
Accuracy
0.035
0.050
0.065
Original
Weighted
New
200
300
400
500
600
Sample Size

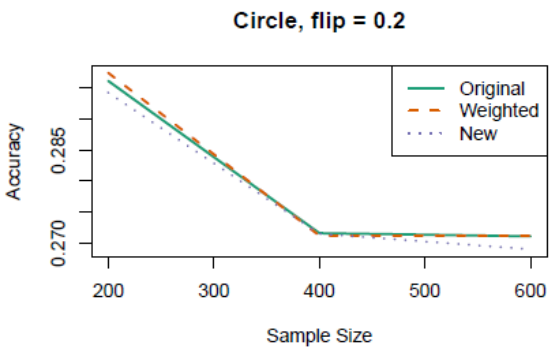

Ringnorm, flip = 0.2
Original
Weighted
New
Accuracy
0.240
0.250
0.260
200
300
400
500
600
Sample Size


Spirals, flip = 0.2
Original
Weighted
New
Accuracy
0.308
0.312
0.316
200
300
400
500
600
Sample Size


Twonorm, flip = 0.2
Original
Weighted
New
Accuracy
0.230
0.240
0.250
200
300
400
500
600
Sample Size


2dnormals, flip = 0.2
Original
Weighted
New
Accuracy
0.296
0.302
200
300
400
500
600
Sample Size


Threenorm, flip = 0.2
Original
Weighted
New
Accuracy
0.305
0.320
0.335
200
300
400
500
600
Sample Size


Circle, flip = 0.2
Original
Weighted
New
Accuracy
0.270
0.285
200
300
400
500
600
Sample Size

**BIBLIOGRAPHY**

Boulesteix, Anne-Laure, et al. (2012). "Overview of random forest methodology and practical guidance with emphasis on computational biology and bioinformatics." *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 2.6. 493-507.

Breiman, Leo. "Random forests." *Machine learning* 45.1 (2001): minig5-32.

Leisch, F. & Dimitriadou, E. (2010). mlbench: Machine Learning Benchmark Problems. R package version 2.1-1.

Liaw, Andy and Matthew Weiner. (2018). “Breiman and Cutler’s Random Forests for Classification and Regression”. 1-29.

Liaw, Andy, and Matthew Wiener. (2002). "Classification and regression by randomForest." *R news* 2.3. 18-22.

Newman, D.J. & Hettich, S. & Blake, C.L. & Merz, C.J. (1998). UCI Repository of machine learning databases [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science.

Pedregosa et al. (2011). “Scikit-learn: Machine Learning in Python”, *JMLR* 12. 2825-2830.

R Core Team (2013). *R: A language and environment for statistical computing. R Foundation for Statistical Computing*. Vienna, Austria. URL http://www.R-project.org/.

SAS Institute Inc. (2016). *SAS® Enterprise Miner™ 14.2: High-Performance Procedures*. Cary, NC: SAS Institute Inc.

Strobl, Carolin, et al. (2007). "Bias in random forest variable importance measures: Illustrations, sources and a solution." *BMC bioinformatics* 8.1. 25.

Winham, S. J., Freimuth, R. R., Biernacka, J.M. (2013) A Weighted Random Forest Approach to Improve Predictive Performance, *Statistical Analysis and Data Mining*, 6(6).