Introducing Machine Learning Models for Psychologists - Random Forest

Francisco N. Ramos

Table of contents

1 Introducing Machine Learning Models for Psychologists - Random Forest

In this document, I will walk through a simple tutorial for fitting and analyzing a random forest model of typical psychological data.

Although the workings of random forest models are explained in depth elsewhere (click here for a more detailed but digestible explanation), to be brief, random forest models are built from single decision trees, which predict a response variable/dependent variable (DV) based on binary answers to other questions, which are selected in order to create as homologous of groups as possible. For example, a tree might first decide to check "Is the social anxiety score over or under 10.5?”, and divide the data based on each record’s “answer”. Then, it will continuously repeat the process with new questions designed to improve the decisions, like “Is sex male or female?”, until it has finalized its prediction and new questions do not improve its output. A random forest model is built from averaging the observations of thousands or more trees, all of which ask different “questions” and may assess different variables (Breiman, 2001).

Random forest models usually do well in describing nonlinear/complex relationships between predictor and response variables, are robust (though can technically overfit), and can handle missing and categorical data. However, they are more difficult to interpret than other models like linear regression. Random forests are popular in fields such as biostatistics because they are excellent for problems where the number of predictors p exceeds the number of samples/participants n. Such problems are not as common in psychology, but random forests still model $n>p$ problems well, as should be expected.

This is the first part of a set of documents walking through building simple random forest models for psychological data, including the addenda `Methods of Variable Selection`, `LASSO (Least absolute shrinkage and selection operator)`, and finally `Predictions`.

All references (including the dataset) are at the end of the `LASSO` addendum to this document.

1.1 The Task at Hand

Importantly, I am conducting a **regression** task (predicting a continuous variable), not a **classification** task (predicting a dichotomous or polytomous variable).

The **first** main objective I’ll be attempting to address, and the **only one addressed in this specific document** (the other tasks are addressed in addendum posts, which I link to at the top and bottom of this document as well as throughout), is that I would like to build a suitable predictive model that generalizes well to unseen data of arithmetic performance score in German students from a set of predictor variables including age, sex, self-reported math anxiety, and others. Testing on unseen data is specifically performed in my `Predictions` addendum.

My **second** main objective is to find a “parsimonious” model, which uses a small number of variables to make “good” or “sufficient” prediction. This may be useful insofar as, for example, future researchers do not have to waste resources collecting unimportant variables for no or minimal improvement in prediction. This can be found in my `Methods for Variable Selection` and `LASSO` addenda.

A related **subgoal** is another common objective, that being I would like to have a good sense of which predictor variables appear useful in predicting arithmetic performance and which ones do not. In other words, I would like to identify which variables are important to predicting the response, regardless of their correlation with other predictors. This will also be addressed in my `Methods for Variable Selection` and `LASSO` addenda.

I have not seen the data ahead of time. The analyses described hereafter are entirely novel.

1.2 The Data

The dataset I will investigate in this document is characteristic of many psychology datasets — it has a solid but not too large sample size (735 N before being split into test/training sets), a good but not excessive number of potential predictors (20+ collected as part of the survey, many of which are likely unhelpful in prediction), and what is likely middling to low effect sizes for predictors.

For what it's worth, this dataset does not scream "use random forest". The appeal of random forests is often their proficiency in modeling $p \ggg n$ situations, particularly because random forest models often do best on sizeable datasets. However, random forest models are able to model many situations, including $n>p$, well and often do so competitively with other models, such as linear regression or other machine learning models. I will find out if this holds for our dataset in my `Prediction` addendum.

For simplicity and didactic purposes, I have pulled twenty variables from the full dataset for my analysis. Some will have a relationship with the DV, others will not. *A priori* variable selection according to expert knowledge of what variables are likely to have predictive relevance would likely be the best variable selection method, though I have retained all suitable variables for this demonstration. Variable selection for this dataset is the focus of my `Methods for Variable Selection` addendum.

Below, I define the single DV as well as the twenty candidate predictor variables followed by the syntax for their object in R.

1.2.1 Dependent Variable Definition

1. Arithmetic performance/`sum_arith_perf`, as measured by "the number of correctly solved problems in order as instructed" on a simple arithmetic speed test.

1.2.2 Predictors Definitions

1. Age/`age`, as measured in years.
2. Sex/`sex`, where 1 = male, 2 = female, and 3=other. Participants who ignored this question were removed.
3. Neuroticism/`score_BFI_N`, as measured by the sum score of the 8 items of the Big Five Inventory (short version) pertaining to neuroticism.

4. Math anxiety/`score_AMAS_total`, as measured by the sum score on the Abbreviated Math Anxiety Scale.

5. General trait anxiety/`score_GAD`, as measured by sum score on the Generalized Anxiety Disorder Screener (GAD-7).

6. Math self-efficacy/`score_PISA_ME`, as measured in the PISA 2012 study using the sum score of six items.

7. General state anxiety/`score_STAI_state_short`, as assessed by the sum of the five-item scale STAI-SKD.

8. Test anxiety/`score_TAI_short`, as measured by the sum score of the 5 items on the short version of the Test Anxiety Inventory.

9. Math self-concept/`score_SDQ_M`, as measured by the sum score of the four math-related statements on the Self-Description Questionnaire III. Evaluates variables such as one's comfort/enjoyment/pride with math, whereas self-efficacy evaluates one's self-confidence in math abilities.

10. Language self-concept/`score_SDQ_L`, as measured by the sum score of the four language-related statements on the Self-Description Questionnaire III.

11. Total duration of survey/`total_time_minutes`, as measured by the number of full minutes a participant took to complete the survey (excluding the timed arithmetic performance section).

12. Last math grade at school/`math_grade`, as measured on a scale ranging from 1-6, where 1 is the best grade and 6 is the worst. (I will reverse this in data cleaning)

13. Participant subject's relation to math/`math_load`, where a "1" means a participant's subject of study has low relation to math, a "2" means a participant's subject has medium relation to math, and a "3" means a participant's subject has high relation to math.

14. Influence of math load on participant's subject choice/`math_inf_program_choice`, as measured by a 9-point Likert scale where a "1" means "chosen because I wanted to avoid math subjects", a "5" means "no role" and a "9" means "chosen because I wanted to take math subjects".

15. Level of liking math/`liking_math`, as measured by a 5-point Likert scale where a "1" means "not at all" and a 5 means "absolutely".

16. Level of liking science/`liking_science`, as measured by a 5-point Likert scale where a "1" means "not at all" and a 5 means "absolutely".

17. Level of liking humanities/`liking_humanities`, as measured by a 5-point Likert scale where a "1" means "not at all" and a 5 means "absolutely".

18. Persistence in solving tasks in mathematics/persistence_math, as measured by a 5-point Likert scale where a "1" means "I get discouraged very fast" and a "5" means "I am very persistent".
19. Persistence in solving tasks in science/persistence_science, as measured by a 5-point Likert scale where a "1" means "I get discouraged very fast" and a "5" means "I am very persistent".
20. Persistence in solving tasks in humanities/persistence_humanities, as measured by a 5-point Likert scale where a "1" means "I get discouraged very fast" and a "5" means "I am very persistent".

1.3 Cleaning the Data and a Little Exploratory Data Analysis

I'll start with importing and cleaning the data to make sure it fits the task. I load the most basic libraries - I will load other libraries at the start of later code chunks when needed. The package required for each function in this document is specified in the function call, unless it is available in base R.

```r
library(here)
library(tidyverse)
library(readr)
```

The `here()` function is convenient for creating paths and directories that won't break when files are moved or when you reproduce the work on another computer. More information is available at the following URL: https://here.r-lib.org/

Below, I'll use the `here()` function to tell the `readr()` function where to go to get the dataset. The data was stored in my "Main Script" folder, in a subfolder called "OSF archive", and named "AMATUS_dataset.csv". Notice how I use a `normalizePath()` function to convert relative path names to absolute path names, which helps when switching OSes or from cloud to local storage.

```r
here::i_am("Dr. Lai Feature Selection Project 9.13.24.rproj")
## here() starts at /Users/frankie/Desktop/Dr. Lai Feature Selection Project 9.13.24
# I manually set "here" to the folder containing my rproj file.

file_path <- normalizePath(here::here(
  "Main Script/OSF archive/AMATUS_dataset.csv"
))
# Also adjust this as needed. Function from base R.

# Importing dataset
```

```r
amatus <- readr::read_csv2(file_path, c("", "NA"), col_names = TRUE)
## i Using "','" as decimal and "'.'" as grouping mark. Use `read_delim()` for more control.
# Note that the `c()` function is defining
# what NA values look like for the import function.

View(amatus)
# View the data.

amatusclean <- amatus[!is.na(amatus$sum_arith_perf), ]
# Removing the individuals who did not complete the performance test in order
# as instructed. Using simple subsetting.
amatusclean <- amatusclean[!(amatusclean$sample %in%
  c("german_teachers", "belgian_teachers")), ]
# Removing the 2 teacher samples from the dataset, as I am only # interested in
# the student sample.

View(amatusclean)

# Data cleaning (using tidyverse syntax this time around):
amatusclean <- amatusclean %>%
  dplyr::filter(!is.na(sum_arith_perf)) %>%
  dplyr::mutate(across(c(
    sex, age_range, breaks, honesty, native_speaker, noise
  ), as.factor))
# `filter()` removes the individuals who did not complete the performance test
# in order as instructed. `mutate()` applies the `as.factor()` function to all
# categorical/factor variables.

# I also want to reverse the order of the math grade variable so a 1 is the
# worst grade for simplicity to match every other variable, where lower
# values correspond to having "less" of or being "worse" at the concept.

amatusclean <- amatusclean %>%
  dplyr::mutate(math_grade = recode(math_grade,
    `1` = 6,
    `2` = 5,
    `3` = 4,
    `4` = 3,
    `5` = 2,
    `6` = 1
  ))
View(amatusclean)
```

```r
invisible(colSums(is.na(amatusclean)))
# It appears there is only one NA value in the variables of interest from the
# dataset - in the math load variable. Remove the `invisible()` function from
# the line above to print the results and see for yourself. For simplicity,
# I will just drop this value. There are 734 data points now.
amatusclean <- subset(amatusclean, !is.na(math_load))
nrow(amatusclean)
```

[1] 734

```r

# Let's also isolate the variables in objects now for easier use later on.
numeric_predictors <- c(
  "score_BFI_N", "score_AMAS_total", "age", "score_GAD",
  "score_PISA_ME", "score_STAI_state_short",
  "score_TAI_short", "score_SDQ_L", "score_SDQ_M",
  "total_time_minutes", "math_grade", "math_load",
  "math_inf_program_choice", "liking_math",
  "liking_science", "liking_humanities",
  "persistence_math", "persistence_science",
  "persistence_humanities"
)
categorical_variable <- "sex"
response_variable <- "sum_arith_perf"
```

Now, I'll just take a look at a simple histogram of the distribution of the dependent variable, arithmetic math performance.

1.3.1 Dependent Variable Distribution

```r
library(ggplot2)

ggplot(
  amatusclean,
  aes(x = sum_arith_perf)
) +
  geom_density(fill = "lightblue", alpha = 0.7) +
  theme_minimal()
```

```
# I set up the theme for all plots in hidden code, so I
# won't include it again after this.
```

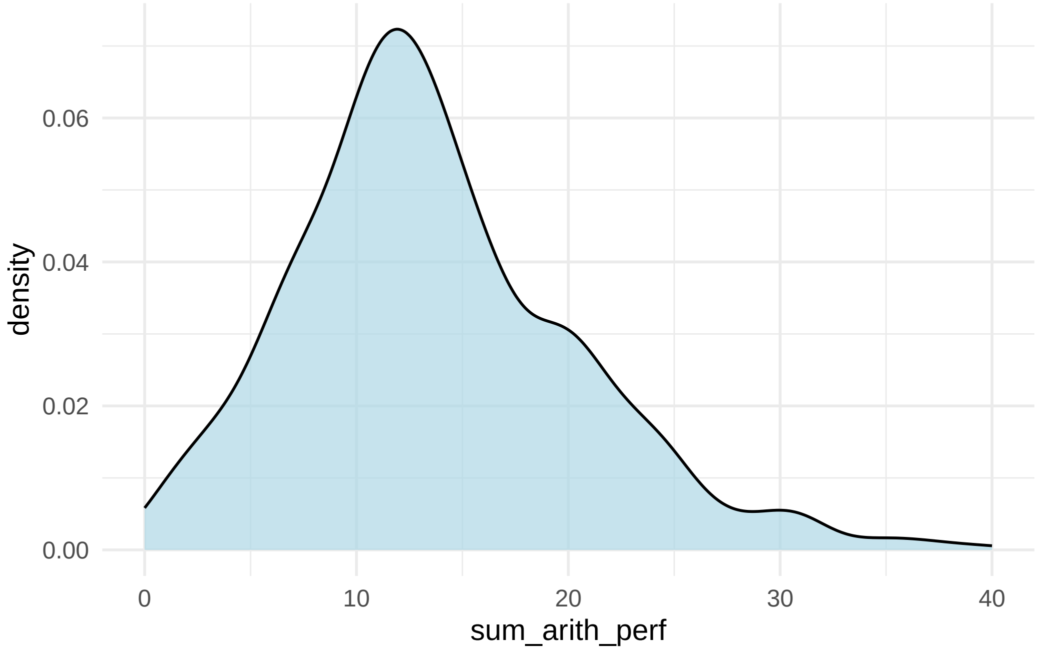

Figure 1: Distribution of response variable: Arithmetic Math Performance

This distribution is clearly skewed right (Figure 1). This indicates that most people scored around the "low" end of all possible values.

Let's look at the distributions of our predictor variables.

1.3.2 Predictor Distributions

```
# The `facet_wrap()` function from `ggplot2` will create a single-image for the
# multiple graphs I create below. However, it only works with "long" data.
# Thus, we use the `pivot_longer()` function from the `dplyr` package to switch
# the data from wide to long format first below.
long_data <- amatusclean %>%
  tidyr::pivot_longer(
    cols = dplyr::all_of(
      numeric_predictors
    ), names_to = "Predictor", values_to = "Value"
  )

# Now, we create all 19 graphs using one function.
facet_distribution_plots <- ggplot2::ggplot(
  # All subsequent functions here are from `ggplot2`, as well.
  long_data, aes(x = Value)
) +
  geom_density(fill = "lightblue", alpha = 0.7) +
  facet_wrap(~Predictor,
    ncol = 4, nrow = 5, scales = "free",
    # I specify rows and columns for 19-20 graphs. Scales vary per predictor.
    strip.position = "bottom"
  ) + # Facet labels on bottom of graphs.
  theme(
    strip.text = element_text(size = 12, face = "bold"), # text settings.
    strip.placement = "outside", # moves labels below x-axes.
    strip.background = element_blank() # removes ugly gray background
  )

facet_distribution_plots
```

Some predictors are approximately normal, others clearly aren't. Luckily, distribution of predictors doesn't matter that much for random forest models. For fun, we'll also look at some continuous predictors graphed against our DV.

```
# Again, we create all 19 graphs using one function.
facet_bivariate_plots <- ggplot2::ggplot(
  # All subsequent functions here are from `ggplot2`, as well.
  long_data, aes(x = Value, y = .data[[response_variable]])
) +
  # This time, we graph predictors against the outcome variable.
  geom_point(alpha = 0.7, color = "lightblue") +
  geom_smooth(method = "lm", color = "red", se = FALSE) +
  # This creates a linear trend line for visualization.
  geom_smooth(method = "loess", color = "blue", se = FALSE) +
  # This creates a nonlinear trend line for visualization.
  facet_wrap(
    ~Predictor,
    ncol = 4, nrow = 5, scales = "free",
    strip.position = "bottom"
  ) +
  theme(
```

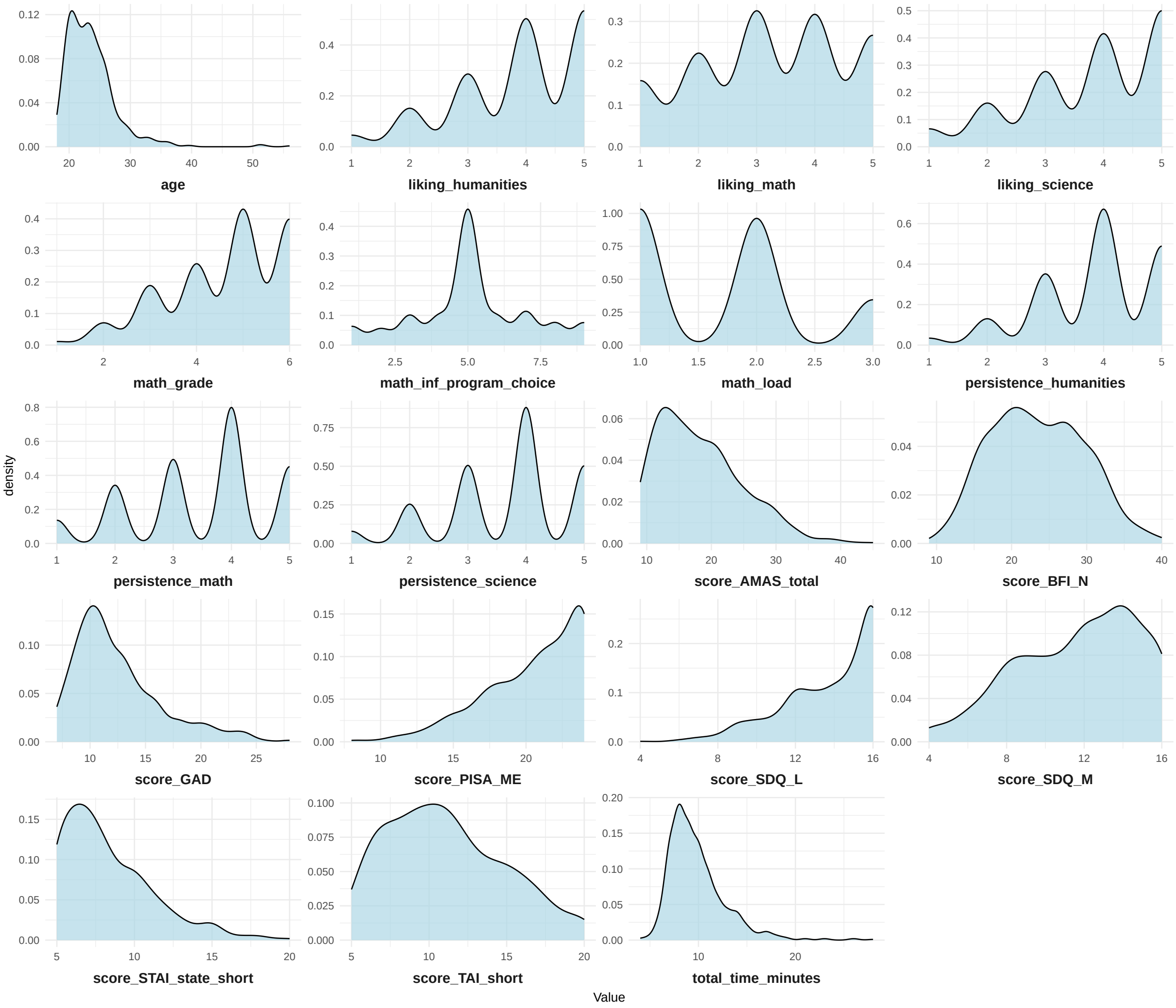

Figure 2: Distribution of predictors

```
    strip.text = element_text(size = 12, face = "bold"),
    strip.placement = "outside",
    strip.background = element_blank()
  )

facet_bivariate_plots
```

The graphs created in the code above are shown on the next page, Page 11, in Figure 3.

Like much of psychological data, most of our predictors (and our DV) are not truly continuous. Our DV for example, `sum_arith_perf`, takes a fairly wide range of values, but it is clear from

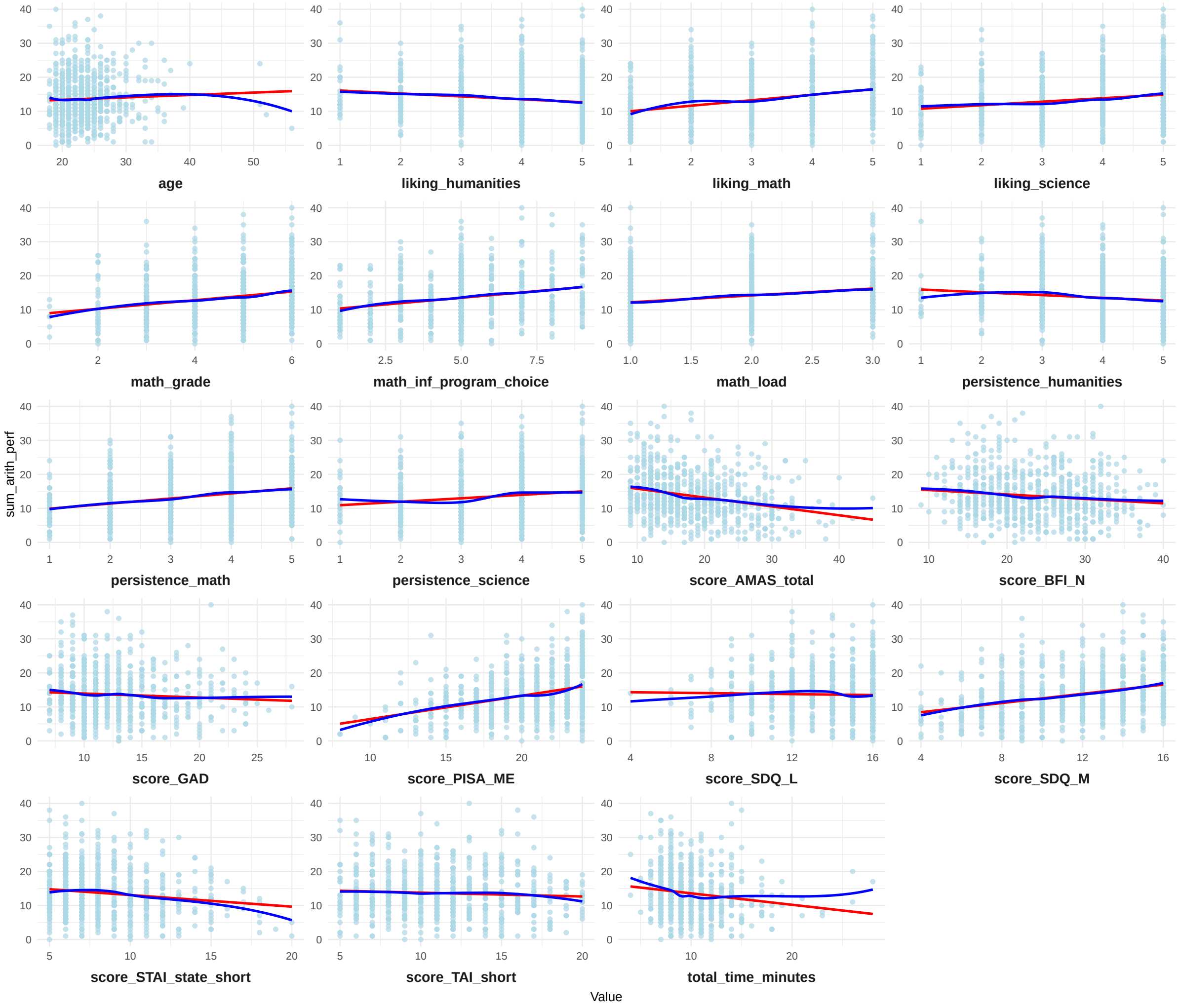

Figure 3: Distribution of predictors against the DV (sum_arith_perf)

its distribution that is truly ordinal because it is measured on a scale. Furthermore, we are currently treating a few ordinal variables with few levels, such as the `liking` and `persistence` variables, as continuous, assuming there is a linear relationship between each level. You would need to check this in a real data project before proceeding. Otherwise, you may turn the ordinal variables into categorical variables (or use different models), though you would lose some information.

Click here for a link to an image of the correlation table provided by the original authors of the dataset, which contains the DV and most of the predictors I am evaluating. Many of the continuous predictors depicted are decently intercorrelated, which is fine because random forest handles this better than many other models (and there aren't any correlations over .8 or .9, which would sound alarm bells). Furthermore, correlations go in the "expected" directions - e.g., arithmetic performance correlates positively with liking math and math self-efficacy correlates negatively with all anxiety measures.

This exploratory data analysis is light, but that is only because I want to focus on the model building and tuning. For a real project, graph all of your variables, transform predictor or outcome variables if needed, and use alternatives to omitting missing data in order to keep statistical power as high as possible (such as multiple imputation).

1.4 Standardization and Training/Test Sets

Because they simply partition data to make predictions at thresholds, random forest models do not *require* you to standardize your variables. However, it has been shown by Strobl and colleagues (2007) that random forest variable importance measures are biased by the range of values a variable can take, tending to produce higher importance scores for variables with a greater range of possible values, even when both are uninformative.

Permutation importance is unbiased with regard to variable importance regarding the average value, although permutation importance can display variance of variable importance for variables with many categories such that this measure can be unreliable in single trials. Importance measures are still commonly utilized in random forest variable selection with solid results (Speiser et al. 2019) - we will review automated methods as well as alternative variable importance metrics in my `Methods for Variable Selection` addendum. Furthermore, the data contains categorical and ordinal data that I do not want to standardize if I do not need to, as this could bias the underlying variance of the noncontinuous data.

I will now create a training set and a test set for model validation. Although validation methods like repeated cross-validation account for the need of a "validation set", I want to have train/test data for the simple purpose of testing built models on "unseen data" to evaluate how well it generalizes to new data. This also comes with concerns (such as the fact that the dataset the data is trained on gets smaller), but this should still help me work out which of the models I construct are truly the best at prediction. (See the `Predictions` addendum for the results).

1.4.1 Setting up the Train/Test Split

```
library(caret)
## Loading required package: lattice
##
## Attaching package: 'caret'
## The following object is masked from 'package:purrr':
##
##     lift
# needed first for the inTrain function below.
set.seed(39)
# I'm going to set the random seed BEFORE every
# random generation, including model fitting, for clarity.
inTrain <- caret::createDataPartition(amatusclean$sum_arith_perf,
  p = 0.7, list = FALSE
)
# 0.7 will sort 70% of the records into training set, 30% into the test
# set. I selected 0.7 to have a decent number of N in the test set.

# Create training vs test data using subsetting.
training <- amatusclean[inTrain, ]
test <- amatusclean[-inTrain, ]
View(training)
View(test)
```

1.4.2 A Note on Feature/Variable Selection

For the purposes of this particular document, I retain all 20 predictor variables to build models and pretend all of them are theorized to have some level of predictive power. However, this is simply for demonstrative purposes and to compare the models I build with all variables later on with better, more parsimonious models. In reality, *this* is the point where you would want to perform variable selection and remove highly correlated/irrelevant predictors from the model to improve performance. I perform this process in the `Methods for Variable Selection` addendum. If you suspect your dataset contains irrelevant/highly correlated features, I recommend switching to the `Methods for Variable Selection` addendum now. If you are merely interested in creating and tuning a random forest model, proceed.

1.5 Tuning your Hyperparameters for your Random Forest Model

Hyperparameters are simply model parameters that, rather than being calculated by the model, you set prior to building the model. In most cases, random forest works pretty well with the "default values" in software packages (Probst et al. 2019), but tuning can improve performance above and beyond that.

Below, I describe four hyperparameters - `mtry`, node size, number of trees and sample size. Most of what I summarize below is explained in greater detail in a paper by Probst and colleagues (2019), which is an incredibly helpful resource for tuning random forest hyperparameters.

1. The hyperparameter `mtry` defines the number of randomly selected predictors to consider splitting at each node of the decision tree, ranging from 1 to p, the number of predictor variables. If you expect many relevant predictors with varying effect sizes, mtry should likely be set low since less influential variables will still be chosen and could provide performance gains. If you expect only a few relevant variables out of many (possibly hundreds), mtry should likely be set high so the algorithm can better find relevant variables (Probst et al, 2019).

    i) The default value for regression in many packages is $p/3$, which has been shown to perform well enough. However, Genuer and colleagues (2008) have shown that in low dimensional regression problems (such as the one at hand), the square root of p performs better, which would be about 4.472 for my analysis. I will tune the `mtry` value, but will also create models with "default" values for hyperparameters for comparison.

2. The hyperparameter `nodesize` describes the minimum number of data points in a leaf/terminal node. It sets the "depth" of trees, meaning that a lower setting leads to more splits being performed until reaching the terminal nodes, which contain fewer observations. The default value is 5 for regression in most statistical packages. Increasing the number of noise variables has been shown to lead to higher optimal node size (Segal 2004) and a higher value is computationally preferable. It has also been shown that higher `nodesize` values lead to overfitting on probability estimation (Barrenada et al. 2024), though this only applies to probability estimation. Again, though I will tune these hyperparameters, I will build other models with default values.

3. The hyperparameter "number of trees" (or `ntree`) refers to the number of individual trees needed to optimize the results of the random forest. The default value is often 500 in statistical packages. This parameter is not really "tunable", as more trees is always better for unbiased error terms. However, more trees directly leads to longer computation time. I will use large values for ntree for the purposes of this document, such as 8000. A number even around a few thousand is likely sufficient for my purposes, though a greater `ntree` can only improve performance (though minimally after a given point).

4. The hyperparameter "sample size" decides how many observations are sampled when training each decision tree. The default value is usually n. If you have a lower value than that, whether you are sampling with or without replacement also becomes a key matter (discussed later). Martinez-Munoz and Suarez (2010) observed that in most datasets, a sample size lower than the standard option tended to produce better performance.

A side note: There are several different validation strategies for a given algorithm that you can use in `caret` or `ranger` or whichever package you use, including bootstrapping, cross-validation, and repeated cross-validation. For random forest, you can also use out-of-bag predictions, which evaluates model performance on the data not selected when the data was bootstrapped as part of the bootstrap/aggregation process. Repeated cross-validation is often optimal when computationally appropriate. However, I will use out-of-bag error to save time and because it is appropriate and reliable for many datasets, as results appear to only be biased in extreme situations (e.g., $n < 20$ with hundreds/thousands of predictors; Janitza & Hornung 2018).

1.6 Tuning Packages in R

Machine learning packages, such as `caret`, contain plenty of functions to help tune models like random forest. For example, the `trainControl()` function is used to define the strategy used, such as repeated cross-validation. You also define if you'd like to search from a grid of predetermined values (`search = "grid"`) or from a random set (`search = "random"`) However, these methods of search can be quite time-consuming, not to mention rather basic (though I recommend reading more about this at the link here.

Instead, I am going to use a slightly more advanced method (sequential model-based optimization, or SMBO) implemented in a simple package, `tuneRanger`, for tuning. Probst and colleagues (2019), in their introduction to the package, state that SMBO is "a very successful tuning strategy that iteratively tries to find the best hyperparameter settings based on evaluations of hyperparameters that were done beforehand". More detail on the process can also be found in papers by Bischl and colleagues (2017) and Hutter and colleagues (2011). The implementation of SMBO in tuneRanger is computationally quick, simultaneously tunes 3 "tunable" parameters, and requires relatively few lines of code.

Please note that tuneRanger uses bootstrap without replacement in the aggregation of the trees. Choosing to sample with or without replacement is an argument in most functions that build a random forest model. This only matters if your sample size hyperparameter is lower than n. Sampling with vs. without replacement does not change the model much particularly when the sample size parameter is "optimal" (Martinez-Munoz and Suarez (2010), but in situations with categorical variables with varying number of categories, a small variable selection bias may be present. I will stick to without replacement because `tuneRanger` does.

(I am optimizing hyperparameters to have the model with the least mean squared error, given that I am conducting regression.)

Let's go through the code to tune these hyperparameters now. Notice that you may need to install and load the `mlr` package, or "Machine Learning in R", for initial setup of the model (or the "regression task" in `mlr` notation). Note that `mlr` is depreciated and `mlr3` is the new and improved package, but I will stick to `mlr` in this demonstration since the tuneRanger package was written for the use of `mlr`.

```r
library(ranger)
library(tuneRanger)
library(mlr)

# Here's where the helpful defined objects I set earlier come in handy.

# Subset the original training data using dplyr
training_subset <- training %>% dplyr::select(all_of(
  c(response_variable, numeric_predictors, categorical_variable)
))




# Ensure it's a dataframe
training_subset <- as.data.frame(training_subset)

# use the `makeRegrTask()` function in mlr to create the "task"
tuning_task1 <- mlr::makeRegrTask(
  data = training_subset,
  target = "sum_arith_perf"
)

# Estimate how long tuning will take with default parameters
tuneRanger::estimateTimeTuneRanger(tuning_task1)
```

Approximated time for tuning: 1M 18S

```r
# This first run will involve a large number of trees and more iterations
# than default since I have leftover computing power. Adjust the values
# as needed.
set.seed(40)

rf_firsttuneranger <- tuneRanger::tuneRanger(
  tuning_task1,
  num.trees = 8000, num.threads = 8, iters = 100,
  iters.warmup = 50
)
# The `measure` argument defines the measures to be optimized. Since we have
# not specified a value, it will use the default for regression, mean squared
# error. `num.trees` means number of trees, `num.threads` is for parallel
# processing (optional), `iters` is the number of iterations the SMBO will go
# through (default value is 70) and `iters.warmup` is the number of warmup
# steps for the initial design (default value of 30). Lastly, `tune.parameters`
# can manually specify the list of tuned parameters (`mtry`, node size,
# and sample size by default).
rf_firsttuneranger$model
```

Model for learner.id=regr.ranger; learner.class=regr.ranger Trained on: task.id = training_subset; obs = 515; features = 20 Hyperparameters: num.threads=8,verbose=FALSE,respect.unordered.facto

```
# This second run has fewer trees.  Perhaps realistic in terms of computing
# power for most and will be slightly different than the previous run.
set.seed(41)
rf_secondtuneranger <- tuneRanger::tuneRanger(
  tuning_task1,
  num.trees = 6000, num.threads = 8, iters = 70,
  iters.warmup = 30
)
rf_secondtuneranger$model
```

Model for learner.id=regr.ranger; learner.class=regr.ranger Trained on: task.id = training_subset; obs = 515; features = 20 Hyperparameters: num.threads=8,verbose=FALSE,respect.unordered.facto

1.7 Training the Models

Now that I have pulled some hopefully optimized settings for the parameters, I demonstrate building two random forest models with these parameters. I will use the `ranger` package, which is essentially a speedier implementation of the older `randomForest` package, which is what `caret` uses to create RF models. It is also a bit simpler to specify hyperparameters like minimum node size and sample fraction. However, if you have used `caret` or `mlr` to build machine learning models in the past, those are excellent options too.

```
# Train the actual Random Forest model using ranger
set.seed(42)

rf_rangertunedmodel <- ranger::ranger(
```

```r
  formula = sum_arith_perf ~ ., # Target variable, all features
  data = training_subset, # Subsetted Training dataset
  num.trees = 8000, # Number of trees. 8000 for strong performance.
  mtry = 20, # Number of variables sampled at each split
  min.node.size = 38, # Minimum node size.
  sample.fraction = 0.267, # Fraction of n used for training model
  importance = "permutation", # Permutation feature importance.
  replace = FALSE, # Bootstrapping without replacement.
  splitrule = "variance", # standard splitting rule for random forest.
  write.forest = TRUE, # saves individual trees just in case you want them.
  verbose = TRUE # Show training in a bit more detail
)
print(rf_rangertunedmodel)
```

Ranger result

Call: ranger::ranger(formula = sum_arith_perf ~ ., data = training_subset, num.trees = 8000, mtry = 20, min.node.size = 38, sample.fraction = 0.267, importance = "permutation", replace = FALSE, splitrule = "variance", write.forest = TRUE, verbose = TRUE)

Type: Regression Number of trees: 8000 Sample size: 515 Number of independent variables: 20 Mtry: 20 Target node size: 38 Variable importance mode: permutation Splitrule: variance OOB prediction error (MSE): 37.27925 R squared (OOB): 0.1441785

```r
set.seed(42)

rf_rangertunedmodel2 <- ranger::ranger(
  formula = sum_arith_perf ~ .,
  data = training_subset,
  num.trees = 8000,
  mtry = 16,
  min.node.size = 38,
  sample.fraction = .311,
  importance = "permutation",
  replace = FALSE,
  splitrule = "variance",
  verbose = TRUE
)
print(rf_rangertunedmodel2)
```

Ranger result

Call: ranger::ranger(formula = sum_arith_perf ~ ., data = training_subset, num.trees = 8000, mtry = 16, min.node.size = 38, sample.fraction = 0.311, importance = “permutation”, replace = FALSE, splitrule = “variance”, verbose = TRUE)

Type: Regression Number of trees: 8000 Sample size: 515 Number of independent variables: 20 Mtry: 16 Target node size: 38 Variable importance mode: permutation Splitrule: variance OOB prediction error (MSE): 37.24817 R squared (OOB): 0.144892

Fairly similar results from the two models, which is expected given that one is likely just a bit more refined given extra iterations and trees. The tuned value for node size is pretty high, which means the trees are not fitting too deeply. Similarly, the `mtry` values are relatively high, indicating that my models seem to fit better when more variables are being sampled at each split. High values of `mtry` appear to be better for datasets containing a few relevant variables out of many (Goldstein et al. 2011), which may be the case here. Lastly, a lower sample size means the individual trees are more diverse and the out of the bag predictions should be solid in terms of estimating the optimal parameter.

You will also notice that the out-of-bag r-squared for each model is not great - 0.146 and 0.145, respectively. This is likely for a few reasons - first, there are undoubtedly some noise variables in the predictors (possibly even constituting a majority). This means the model will train in part on random noise and a more parsimonious model would likely produce more accurate predictions, especially with tuned parameters (more on that in my `Methods for Variable Selection` addendum). Furthermore, this actually probably isn’t actually too far from a usable R-squared in psychology research. Models of human behavior are often far from exact - for example, Wallert et al. (2018) used a random forest classifier model to predict adherence to internet cognitive behavioral therapy treatments for patients who suffered myocardial infarction. Their finalized model (after variable selection with the LASSO algorithm, which I go over in my `LASSO` addendum) reported an accuracy of 0.64. Such a low accuracy value might be surprising, but numbers in this range are usually decent for psychological research, given that human behavior is far more difficult to predict accurately than say, the genes that predict future occurrence of a disease.

Next, I’ll take a look at random forest variable importance scores, which are useful for identifying important predictors in a given model. Recall that I am using permutation importance. However, keep in mind that such scores are relative to the model and can be wildly different from other measures of “importance” to predicting a DV (like zero-order correlation, for example). The scores I evaluate are unscaled.

```r
# Pulling importance scores for model 1, to be called `importance_df`
importance_scores <- rf_rangertunedmodel$variable.importance
importance_df <- data.frame(
  Variable = names(importance_scores), Importance = importance_scores
)
importance_df <- importance_df[order(-importance_df$Importance), ]
print(importance_df)
```

```
Variable   Importance
```

score_PISA_ME score_PISA_ME 3.589547753 score_SDQ_M score_SDQ_M 1.941928038 sex sex 1.475072763 liking_math liking_math 1.444666776 score_AMAS_total score_AMAS_total 1.010055063 persistence_math persistence_math 0.541036820 total_time_minutes total_time_minutes 0.501312651 math_grade math_grade 0.491543991 math_inf_program_choice math_inf_program_choice 0.221200686 liking_science liking_science 0.174240678 score_STAI_state_short score_STAI_state_short 0.152486418 persistence_science persistence_science 0.092768977 score_BFI_N score_BFI_N 0.065190642 math_load math_load 0.063423876 score_TAI_short score_TAI_short 0.051604622 score_GAD score_GAD 0.032831793 age age 0.024770114 persistence_humanities persistence_humanities 0.022385539 liking_humanities liking_humanities 0.021394592 score_SDQ_L score_SDQ_L -0.002609613

```r
# Pulling importance scores for model 2, to be called `importance_df2`
importance_scores2 <- rf_rangertunedmodel2$variable.importance
importance_df2 <- data.frame(
  Variable = names(importance_scores2), Importance = importance_scores2
)
importance_df2 <- importance_df2[order(-importance_df2$Importance), ]
print(importance_df2)
```

```
Variable  Importance
```

score_PISA_ME score_PISA_ME 3.48700624 score_SDQ_M score_SDQ_M 1.92734268 sex sex 1.57027956 liking_math liking_math 1.53459657 score_AMAS_total score_AMAS_total 0.94575458 persistence_math persistence_math 0.56207837 math_grade math_grade 0.51245712 total_time_minutes total_time_minutes 0.50277400 math_inf_program_choice math_inf_program_choice 0.21250082 score_STAI_state_short score_STAI_state_short 0.18198110 liking_science liking_science 0.13511296 score_TAI_short score_TAI_short 0.09819293 score_BFI_N score_BFI_N 0.08915444 math_load math_load 0.07695278 persistence_science persistence_science 0.04826767 score_GAD score_GAD 0.04481476 liking_humanities liking_humanities 0.02784193 persistence_humanities persistence_humanities 0.01774066 age age 0.00722506 score_SDQ_L score_SDQ_L -0.01572412

It appears that `score_PISA_ME` is far and away the most important variable in each model. Besides the dropoff after `score_PISA_me`, there is also a major dropoff around the 5th/4th variable in the models respectively. Recall that a lot of the predictors are intercorrelated -

I may think, for example, "is math persistence giving me a lot of unique information that something like math self-efficacy is not?"

The variable importance metrics suggest to me that the dataset likely has somewhere around 4-5 variables that actually truly contribute to prediction and are not mostly noise variables, perhaps even fewer being truly strong predictors of the dependent variable. However, this measure of variable importance is relative, and simply selecting the top X many predictors is a simplistic and suboptimal method of variable selection. This is why I will conduct and compare several methods of feature selection on this dataset in my `Methods for Variable Selection` addendum, then retune the hyperparameters and refit random forest models using the selected feature list to test their predictions in the `Predictions` addendum.

Lastly, as additional comparison models, I will create two 20-variable random forest models with "default" values for hyperparameters. In other words, minimum node size will be 5 (default for regression), `mtry` will equal $p/3$ as default for regression (as well as the square root of p), and sample size will be n. These models will just serve as a point of comparison for showing how tuning model parameters can lead to better prediction.

I hope this document has been helpful with regards to better understanding the concept and implementation of building random forest models and tuning their hyperparameters. Please read my `Methods for Variable Selection` and `Predictions` addenda to get the "full story" of this data analysis, as I will be selecting a more refined set of features to model and testing the predictions of all of the models I have and will build on unseen test data, respectively.

```r
# Train model using default hyperparameters
set.seed(43)
rf_default_model <- ranger::ranger(
  formula = sum_arith_perf ~ ., # Target variable and features
  data = training_subset, # Training dataset
  num.trees = 8000, # Number of trees
  mtry = 4.47, # Number of variables randomly sampled at each split
  importance = "permutation", # Computes feature importance
  splitrule = "variance", # For regression (use "gini" for classification)
  verbose = TRUE # Show training details
)

print(rf_default_model)
```

Ranger result

Call: ranger::ranger(formula = sum_arith_perf ~ ., data = training_subset, num.trees = 8000, mtry = 4.47, importance = "permutation", splitrule = "variance", verbose = TRUE)

Type: Regression Number of trees: 8000 Sample size: 515 Number of independent variables: 20 Mtry: 4 Target node size: 5 Variable importance mode: permutation Splitrule: variance OOB prediction error (MSE): 37.84268 R squared (OOB): 0.1312439

```r
default_importance_scores <- rf_default_model$variable.importance
importance_scores3 <- rf_default_model$variable.importance
default_importance_df <- data.frame(
  Variable = names(importance_scores3), Importance = importance_scores3
)
default_importance_df <- default_importance_df[
  order(-default_importance_df$Importance),
]
print(default_importance_df)
```

```
                                    Variable  Importance
```

score_PISA_ME score_PISA_ME 2.49654711 liking_math liking_math 2.16155753 score_SDQ_M score_SDQ_M 1.88547592 score_AMAS_total score_AMAS_total 1.58219510 persistence_math persistence_math 1.54847319 math_grade math_grade 1.38496868 sex sex 1.32869094 total_time_minutes total_time_minutes 0.81925869 liking_humanities liking_humanities 0.76350819 liking_science liking_science 0.65735862 score_STAI_state_short score_STAI_state_short 0.61467080 persistence_science persistence_science 0.53049509 math_load math_load 0.51647010 persistence_humanities persistence_humanities 0.51088914 score_GAD score_GAD 0.51076088 score_BFI_N score_BFI_N 0.34438627 score_TAI_short score_TAI_short 0.31063735 age age 0.15123593 score_SDQ_L score_SDQ_L 0.10091545 math_inf_program_choice math_inf_program_choice -0.00304766

```r

set.seed(44)
rf_default_model2 <- ranger::ranger(
  formula = sum_arith_perf ~ ., # Target variable and features
  data = training_subset, # Training dataset
  num.trees = 8000, # Number of trees
  mtry = 6.666, # Number of variables randomly sampled at each split
  importance = "permutation", # Computes feature importance
  splitrule = "variance", # For regression (use "gini" for classification)
  verbose = TRUE # Show training details
)

print(rf_default_model2)
```

Ranger result

Call: ranger::ranger(formula = sum_arith_perf ~ ., data = training_subset, num.trees = 8000, mtry = 6.666, importance = “permutation”, splitrule = “variance”, verbose = TRUE)

Type: Regression Number of trees: 8000 Sample size: 515 Number of independent variables: 20 Mtry: 6 Target node size: 5 Variable importance mode: permutation Splitrule: variance OOB prediction error (MSE): 38.13992 R squared (OOB): 0.1244199

```r
importance_scores4 <- rf_default_model2$variable.importance
default_importance_df2 <- data.frame(
  Variable = names(importance_scores4), Importance = importance_scores4
)
default_importance_df2 <- default_importance_df2[
  order(-default_importance_df2$Importance),
]
print(default_importance_df2)
```

```
                    Variable  Importance
```

score_PISA_ME score_PISA_ME 3.04865606 liking_math liking_math 2.12716398 score_SDQ_M score_SDQ_M 1.87608742 score_AMAS_total score_AMAS_total 1.60184531 persistence_math persistence_math 1.55521674 sex sex 1.35839263 math_grade math_grade 1.27416425 total_time_minutes total_time_minutes 0.95407078 score_STAI_state_short score_STAI_state_short 0.61972107 liking_humanities liking_humanities 0.58438631 liking_science liking_science 0.53773137 score_GAD score_GAD 0.50994630 persistence_science persistence_science 0.45964434 math_load math_load 0.45875730 persistence_humanities persistence_humanities 0.39978225 score_TAI_short score_TAI_short 0.39717869 score_BFI_N score_BFI_N 0.34839674 age age 0.15914036 score_SDQ_L score_SDQ_L 0.01485572 math_inf_program_choice math_inf_program_choice -0.29177447

The End Links to other Documents: Methods for Variable Selection LASSO Addendum Predictions