Model_Fitting

Patrick Seebold

```
Warning: package 'tidymodels' was built under R version 4.3.3

-- Attaching packages -------------------------------------- tidymodels 1.2.0 --

v broom        1.0.5     v recipes      1.1.0
v dials        1.3.0     v rsample      1.2.1
v dplyr        1.1.4     v tibble       3.2.1
v ggplot2      3.5.1     v tidyr        1.3.1
v infer        1.0.7     v tune         1.2.1
v modeldata    1.4.0     v workflows    1.1.4
v parsnip      1.2.1     v workflowsets 1.1.0
v purrr        1.0.2     v yardstick    1.3.1

Warning: package 'dials' was built under R version 4.3.3

Warning: package 'scales' was built under R version 4.3.3

Warning: package 'dplyr' was built under R version 4.3.3

Warning: package 'ggplot2' was built under R version 4.3.3

Warning: package 'infer' was built under R version 4.3.3

Warning: package 'modeldata' was built under R version 4.3.3

Warning: package 'parsnip' was built under R version 4.3.3

Warning: package 'purrr' was built under R version 4.3.3
```

```
Warning: package 'recipes' was built under R version 4.3.3

Warning: package 'rsample' was built under R version 4.3.3

Warning: package 'tidyr' was built under R version 4.3.3

Warning: package 'tune' was built under R version 4.3.3

Warning: package 'workflows' was built under R version 4.3.3

Warning: package 'workflowsets' was built under R version 4.3.3

Warning: package 'yardstick' was built under R version 4.3.3

-- Conflicts ----------------------------------------- tidymodels_conflicts() --
x purrr::discard() masks scales::discard()
x dplyr::filter()  masks stats::filter()
x dplyr::lag()     masks stats::lag()
x recipes::step()  masks stats::step()
* Dig deeper into tidy modeling with R at https://www.tmwr.org
```

Introduction

This is Part 2 of this project, which seeks to predict diabetes status from features of the person, using data from an online data set. In our previous document, we performed an exploratory data analysis (EDA). In this document, we will train our model and select the best fit.

The goal of predictive modeling is to construct a mathematical model based on prior data in order to predict some variable of interest in new data. In this case, we have a data set in which we know the ground truth for our variable of interest (i.e., whether each participant has diabetes) as well as data representing various features of each individual's health. Our goal in this document is to use this data to create such a model, tune it so that we can be confident we have the best performing model that we can, and then validate that it works on novel data. In this case, we will use self-report data of the participant's mental, physical, and general health alongside their age and education level to try and predict their diabetes status.

Note that this is inherently an experimental approach - more standard methods would use features such as blood pressure and other physiological variables. For this reason, our model may not perform well. However, if we *are* able to predict diabetes status from these subjective values, this would be a very interesting finding! Plus, it keeps things interesting instead of iterating the more commonly used approach.

Our Model Types

In this workflow, we will use two different types of models. First, let's discuss how these models work before we start working with the data.

The first model we will use is a Classification Tree. In a Classification Tree, we create a series of progressive splitting criteria leading to different terminal classifications. For example, a tree might begin with the splitting criteria where for participants with General Health of 3 or greater, split left; else split right. Each of these splits will then go through subsequent splitting conditions, further subsetting the data into smaller subgroups. At the end of this chain of splits, each branch will terminate in a final classification. When a new data point is fed into the model to have its class predicted, it follows the iterative if-then splits until it eventually lands at one of these terminal nodes - whatever node it lands at becomes the classification for that point. These trees are handy because they are easy to understand, but they can be quite variable between training attempts.

The second model type is the Random Forest model, which is essentially an ensemble approach using multiple Classification trees. Instead of using just one tree, which can be quite variable between training attempts, a Random Forest involves training a whole bunch of trees (ending up with a forest!). We then treat each tree's classification as a 'vote' in the final model for new data, and the majority vote becomes that data point's classification. Alternate 'voting' mechanisms are possible, but this is the simplest approach. The Random Forest model has benefits over individual Trees since it helps reduce the variance of the predictions, although it can be more computationally intensive.

Prepping the Data

```
# First, we repeat the data loading steps from the EDA, just in case our user didn't already
data = read.csv("diabetes_binary_health_indicators_BRFSS2015.csv")

# grab only our variables of interest
data_sub = data[, c("Diabetes_binary", "Education", "Sex", "GenHlth", "MentHlth", "PhysHlth"
summary(data_sub)
```

```
 Diabetes_binary    Education          Sex              GenHlth
 Min.   :0.0000   Min.   :1.00   Min.   :0.0000   Min.   :1.000
 1st Qu.:0.0000   1st Qu.:4.00   1st Qu.:0.0000   1st Qu.:2.000
 Median :0.0000   Median :5.00   Median :0.0000   Median :2.000
 Mean   :0.1393   Mean   :5.05   Mean   :0.4403   Mean   :2.511
 3rd Qu.:0.0000   3rd Qu.:6.00   3rd Qu.:1.0000   3rd Qu.:3.000
 Max.   :1.0000   Max.   :6.00   Max.   :1.0000   Max.   :5.000
```

```
    MentHlth         PhysHlth
 Min.   : 0.000   Min.   : 0.000
 1st Qu.: 0.000   1st Qu.: 0.000
 Median : 0.000   Median : 0.000
 Mean   : 3.185   Mean   : 4.242
 3rd Qu.: 2.000   3rd Qu.: 3.000
 Max.   :30.000   Max.   :30.000
```

```
# set up our factors
data_sub$Diabetes_binary = factor(data_sub$Diabetes_binary, levels = c('1','0'),
                                     labels = c("Diabetes", "No Diabetes"))
data_sub$Sex = factor(data_sub$Sex, levels = c('0','1'),
                      labels = c("Female","Male"))
data_sub$Education = factor(data_sub$Education,
                               levels = c('1','2','3','4','5','6'),
                               labels = c("Never attended school or only kindergarten",
                               "Grades 1 through 8 (Elementary)",
                               "Grades 9 through 11 (Some high school)",
                               "Grade 12 or GED (High school graduate)",
                               "College 1 year to 3 years (Some college or technical school)"
                               "College 4 years or more (College graduate)"))
data_sub$GenHlth = factor(data_sub$GenHlth, levels = c('1','2','3','4','5'),
                             labels = c("Excellent", "Very Good", "Good",
                                        "Fair","Poor"))
```

Now, we will split our data into the train/test sets (70/30):

```
# Start out by doing our initial splits
set.seed(42)
splits = initial_split(data_sub, prop = 0.70)
train = training(splits)
test = testing(splits)
```

Now we can prepare our recipes for training the models:

```
# Recipe 1: all numeric variables, no interactions needed for classificaiton
rec1 = recipe(Diabetes_binary ~., data = train) |>
  step_normalize(all_numeric_predictors()) |>
  step_dummy(Education, Sex, GenHlth) #|>
  #prep(training = train) |>
  #bake(train) # Checking that it worked; for validation only
```

Now that we have our recipe, we can explore fitting the training data to them. We'll look at log-loss as our metric in order to select the best recipe for our single classification tree. First, we'll set up our five-fold CV, then will create the workflows, engines, and models for each recipe. We'll need to specify our log loss so we get the right metric!f

```r
set.seed(42)
cv5 = vfold_cv(train, 5) # same CV for all models
metric = metric_set(mn_log_loss)

tree1 = decision_tree(tree_depth = tune(), min_n = 10, cost_complexity = tune()) |>
  set_engine("rpart") |>
  set_mode("classification")

wkf1 = workflow() |>
  add_recipe(rec1) |>
  add_model(tree1)

grid1 = grid_regular(tree_depth(), cost_complexity())

fits1 = wkf1 |>
  tune_grid(resamples = cv5, grid = grid1, metrics = metric)

params1 = select_best(fits1, metric = "mn_log_loss")

params1
```

```
# A tibble: 1 x 3
  cost_complexity tree_depth .config
            <dbl>      <int> <chr>
1    0.0000000001          8 Preprocessor1_Model2
```

Now that we have our best parameters for the single Classification Tree, let's go ahead and train the Random Forest model and find the best mn_log_lost there too:

```r
set.seed(42)
rf_spec = rand_forest(mtry = tune()) |>
  set_engine("ranger",importance = "impurity") |>
  set_mode("classification")

rf_wkf = workflow() |>
  add_recipe(rec1) |>
  add_model(rf_spec)
```

```r
rf_fit = rf_wkf |>
  tune_grid(resamples = cv5,
  grid = grid_regular(mtry(range=c(1,10))), metrics = metric)
```

```
Warning: package 'ranger' was built under R version 4.3.3
```

```r
rf_fit |>
  collect_metrics() |>
  filter(.metric == "mn_log_loss") |>
  arrange(mean)
```

```
# A tibble: 3 x 7
   mtry .metric     .estimator  mean     n  std_err .config
  <int> <chr>       <chr>      <dbl> <int>    <dbl> <chr>
1     5 mn_log_loss binary     0.358     5 0.000656 Preprocessor1_Model2
2    10 mn_log_loss binary     0.367     5 0.000397 Preprocessor1_Model3
3     1 mn_log_loss binary     0.375     5 0.000594 Preprocessor1_Model1
```

```r
params_rf = select_best(rf_fit, metric = "mn_log_loss")
```

We see that mtry = 5 is our best outcome. We now will compare the performance of the best versions of these two model types on the test set to determine our best overall model:

```r
wkf1_final = wkf1 |>
  finalize_workflow(params1) |>
  last_fit(splits, metrics = metric)
wkf1_final |>
  collect_metrics()
```

```
# A tibble: 1 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 mn_log_loss binary         0.378 Preprocessor1_Model1
```

```r
rf_wkf_final = rf_wkf |>
  finalize_workflow(params_rf) |>
  last_fit(splits, metrics = metric)
rf_wkf_final |>
  collect_metrics()
```

```
# A tibble: 1 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 mn_log_loss binary         0.357 Preprocessor1_Model1
```

We see that the log loss is better (lower) in the random forest model, so this is our overall winner! That wraps up our work for this document - next we will create an API file and then host this model in a Dockerfile to allow others to submit data for new observations and get a prediction for whether that participant has diabetes or not.