lab2c_estes

Andrew Estes

1/28/2022

Installing libraries, packages, and datasets

```
#install.packages("pacman", "cvms", "hablar", ranger")
pacman::p_load(tidymodels,
               tidyverse,
               ggplot2,
               MASS,
               leaps,
               car,
               caret,
               rpart,
               rpart.plot,
               randomForest,
               pROC,
               factoextra,
               dplyr,
               hablar,
               cvms,
               ranger)
```

Part 1

1A) Read in the ObesityDataSet_raw_and_data_sinthetic.csv data set. Make sure the variables come in as factors, perhaps using stringsAsFactors=TRUE.

1B) We won't want to use Height and Weight in the modeling. The easiest way to avoid that is to remove them from the data set.

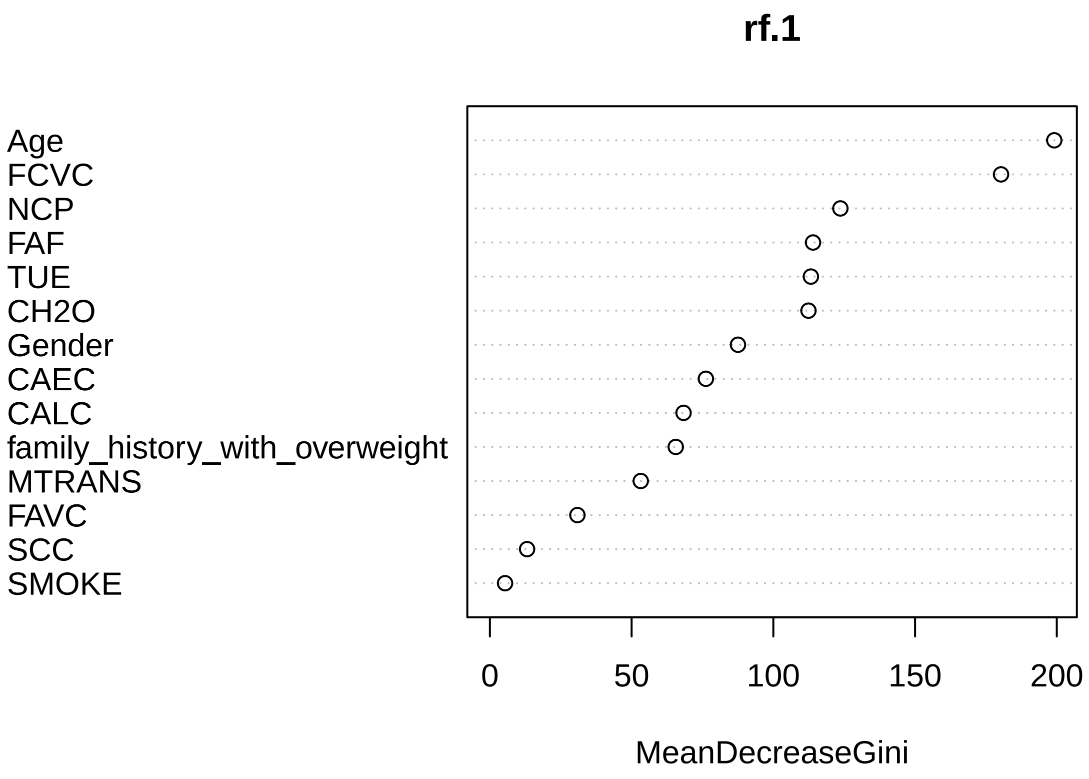
1C) Create a training and test set from the original data. Use set.seed to make sure you get reproducible results (you can choose your own seed, however).

```
#loading the data
df_original <- read.csv("ObesityDataSet_raw_and_data_sinthetic.csv",
                        stringsAsFactors = TRUE)

#copying dataframe and removing two columns
df <- subset(df_original, select=-c(Height, Weight))

#creating training/test datasets
```

```r
set.seed(1234)
split <- sample(1:nrow(df), 0.75*floor(nrow(df)))
split.train <- df[split, ]
split.test <- df[-split, ]
```

1D) Create an untuned random forest model to predict obesity level, and calculate its accuracy on the test set.

```r
#ATTEMPT 1.0
rf.1 <- randomForest(NObeyesdad ~ ., data = split.train)
pred.rf.1 <- predict(rf.1, newdata=split.test)

table(pred.rf.1)
varImpPlot(rf.1)
```



```r
confusionMatrix(pred.rf.1, split.test$NObeyesdad)
```

The Confusion Matrix shows an accuracy level of 84.28%.

The Variable Importance Plot shows Age and FCV are the two most important variables. The next grouping of important variables includes NCP, FAF, TUE, and CH20. The remaining variables are all less than 100 in the Mean Decrease Gini.

Part 2

2A) Create a tuned model, using mtry and min_n as tuning parameters.

Note: In the grid_regular command, you can use mtry(c(1, 14))and min_n(). For whatever reason, mtry needs bounds specified by you, but min_n will pick its own defaults.

Further Note: If you want to have min_n=1, you can actually use the same syntax as with mtry: min_n(c(1, something)). By default, min_n=1 doesn't seem to be selected otherwise.

Since it can take a long time to run the code, I'd suggest using levels=1 in the grid_regular command while you test your code, then change it to something like levels=5 (or more) once you know your code runs.

2B) Make a graph that shows the cross-validated accuracy ("accuracy" is the name of the command) of the model as a function of mtry and min_n. I'd put mtry on the x axis.

2C) Describe in words what the graph tells you about the best model parameters.

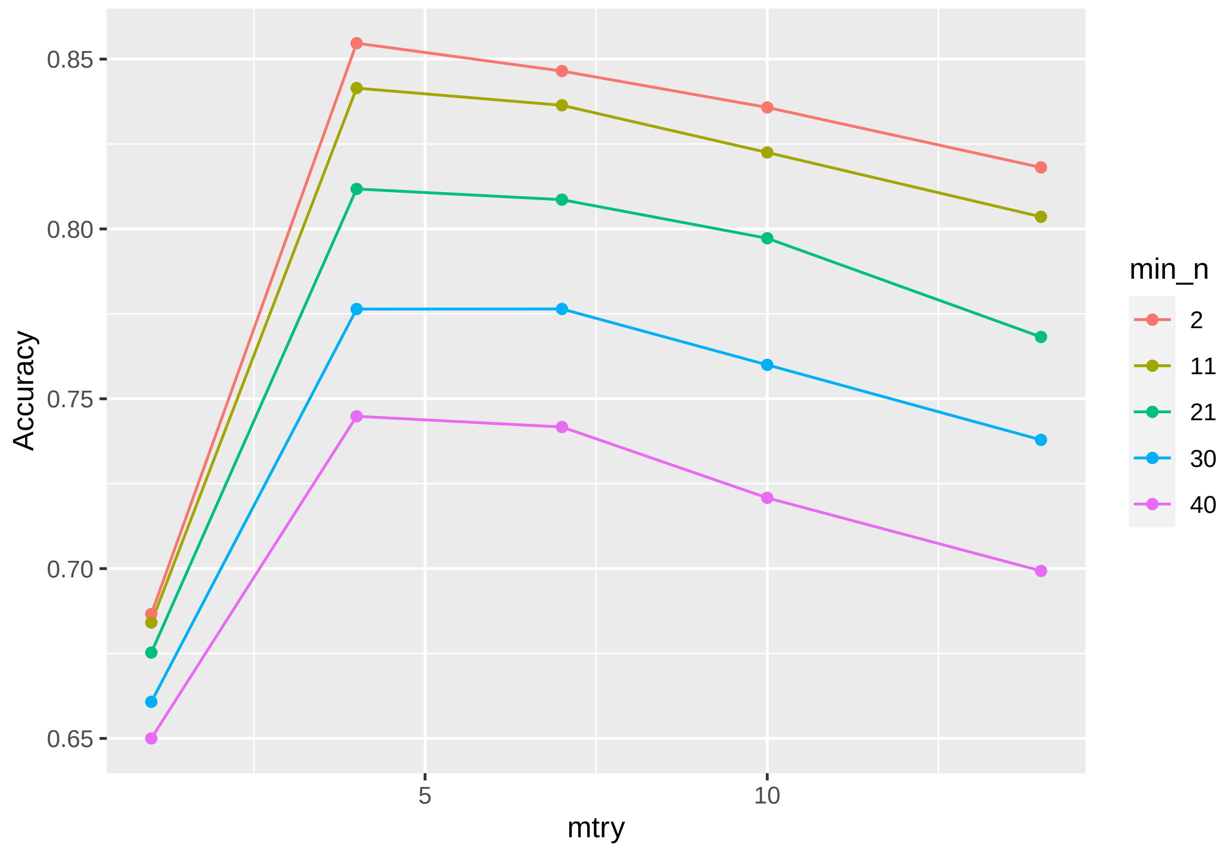
2D) Does it appear that you've captured the range of parameter values that contain the best values?

```r
#setting up tree grid
tree.grid <- grid_regular(mtry(c(1, 14)),
                          min_n(),
                          levels = 5)

#creating cross-validation folds
folds <- vfold_cv(split.train, v = 10)
```

```r
#ATTEMPT 2.4
tune.rf <-
  rand_forest(
    mtry = tune(),
    min_n = tune(),
  ) %>%
  set_engine("randomForest") %>%
  set_mode("classification")
#tune.rf

tune.rf.results <-
  tune.rf %>%
  tune_grid(NObeyesdad ~ ., resamples = folds, grid = tree.grid)

tune.rf.results %>% collect_metrics() %>% head()
```

```
## # A tibble: 6 x 8
##    mtry min_n .metric  .estimator  mean     n std_err .config
##   <int> <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1     1     2 accuracy multiclass 0.687    10 0.0126  Preprocessor1_Model01
## 2     1     2 roc_auc  hand_till  0.951    10 0.00423 Preprocessor1_Model01
## 3     4     2 accuracy multiclass 0.855    10 0.0109  Preprocessor1_Model02
## 4     4     2 roc_auc  hand_till  0.977    10 0.00259 Preprocessor1_Model02
## 5     7     2 accuracy multiclass 0.846    10 0.0101  Preprocessor1_Model03
## 6     7     2 roc_auc  hand_till  0.976    10 0.00296 Preprocessor1_Model03
```

```
tune.rf.results %>% collect_metrics() %>%
  filter(.metric == "accuracy") %>%
  mutate(min_n = factor(min_n)) %>%
  ggplot(aes(x=mtry, y=mean, color=min_n)) +
  geom_line() + geom_point() + ylab("Accuracy")
```



The graph shows that the optimal number of variables to maximize accuracy is 4. It also shows that reducing the number of node splits is more accurate than a higher number of nodes.

I do believe this captures the range of parameter values that contains the best values.

Part 3

3A) Finalize your tuned model with the best parameters that your cross-validation found. Then calculate the prediction accuracy on the test set for this tuned model.

3B) Based on your graph, is model tuning important?

3C) Comparing your tuned results and the untuned model, comment on whether the default parameters from the untuned randomForest seem to be good or bad in this case.

```r
#ATTEMPT 3.1
best.params <- tune.rf.results %>% select_best(metric="accuracy")

best.model <- tune.rf %>%
  finalize_model(best.params) %>%
  fit(NObeyesdad ~ ., data=split.train)

#untuned confusion matrix
untuned.cf <- confusionMatrix(pred.rf.1, split.test$NObeyesdad)
untuned.cf

#tuned confusion matrix
tuned <- predict(best.model, new_data = split.test) %>% bind_cols(split.test)
c_predicted <- c(tuned$.pred_class)
d_actual <- c(tuned$NObeyesdad)
cd <- data.frame(c_predicted, d_actual)

tuned.cf <- confusionMatrix(cd$c_predicted, cd$d_actual)
tuned.cf
```

The accuracy for the tuned model is 85.04% (compared to 84.28% for the untuned model). Even though the accuracy was practically the same, I still do beleive that model tuning is important based upon the graph. Reducing the number of variables and reducing the number of tree node splits is crucially important as it prevents over-fitting (even in this case where it didn't make a significant difference).