Introduction to Decision Trees

Jaime Davila

4/18/2021

Introduction

On today's class we will be using a dataset collated from the popular animated series, Scooby Doo:

```
(scooby.tbl <- read_csv("~/Mscs 341 S22/Class/Data/scooby.csv") %>%
  mutate(monster_real=factor(monster_real)))
```

```
## # A tibble: 501 x 4
##    year_aired  imdb monster_real title
##         <dbl> <dbl> <fct>        <chr>
##  1       1969   8.1 fake         What a Night for a Knight
##  2       1969   8.1 fake         A Clue for Scooby Doo
##  3       1969   8   fake         Hassle in the Castle
##  4       1969   7.8 fake         Mine Your Own Business
##  5       1969   7.5 fake         Decoy for a Dognapper
##  6       1969   8.4 fake         What the Hex Going On?
##  7       1969   7.6 fake         Never Ape an Ape Man
##  8       1969   8.2 fake         Foul Play in Funland
##  9       1969   8.1 fake         The Backstage Rage
## 10       1969   8   fake         Bedlam in the Big Top
## # ... with 491 more rows
```

```
table(scooby.tbl$monster_real)
```

```
##
## fake real
##  389  112
```

In particular we are interested in predicting whether or not the monster in the episode is a real or fake based on the year that the episode was aired and how well liked it was on imdb. A preliminary plot shows the relationship across variables:

```
scooby.tbl %>%
   ggplot(aes(imdb, year_aired))+
    geom_jitter(alpha = 0.7, width = 0.05,
              height = 0.2, aes(color = monster_real))
```

And as usual we will set-up or training/testing dataset:

```
set.seed(123)
scooby.split <- initial_split(scooby.tbl)
scooby.train.tbl <- training(scooby.split)
scooby.test.tbl <- testing(scooby.split)
```

Recursive partitioning trees

Decisions trees introduce a completely new idea for making predictions. The fundamental idea, as the name implies, is to use a **tree** as the means of making a decisions. The tree is built on a sequence of decisions based on the predictor variables.

The easiest way to show a decision is to do an example using our dataset. A couple of things to notice from our code are:

- We use the library `rpart`
- We will be making use of trees with 2 only levels (notice the parameter `tree_depth` below)

```
scooby.model <-
  decision_tree(tree_depth=2) %>%
  set_mode("classification") %>%
  set_engine("rpart")

scooby.recipe <- recipe(monster_real ~ imdb+year_aired,
                 data=scooby.train.tbl)

scooby.wflow <- workflow() %>%
    add_recipe(scooby.recipe) %>%
    add_model(scooby.model)

scooby.fit <- fit(scooby.wflow, scooby.train.tbl)
```

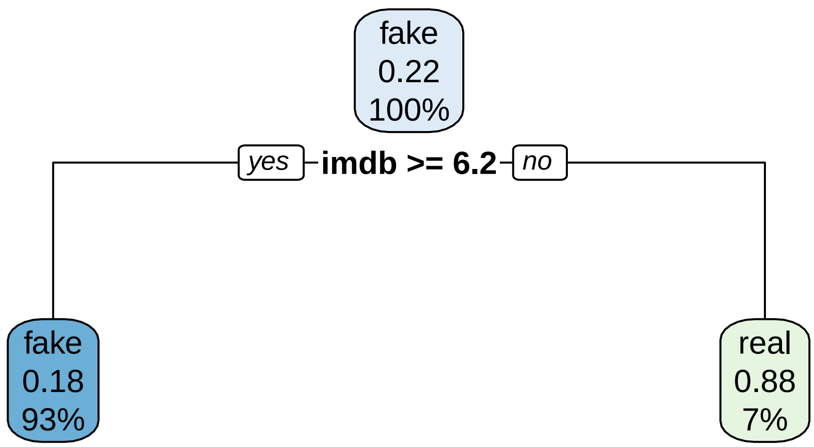
Finally notice that we can get a text output of our tree by using `scooby.fit`

```
scooby.fit
```

```
## == Workflow [trained] ==========================================================
## Preprocessor: Recipe
## Model: decision_tree()
##
## -- Preprocessor ----------------------------------------------------------------
## 0 Recipe Steps
##
## -- Model -----------------------------------------------------------------------
## n= 375
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 375 84 fake (0.7760000 0.2240000)
##   2) imdb>=6.15 350 62 fake (0.8228571 0.1771429) *
##   3) imdb< 6.15 25  3 real (0.1200000 0.8800000) *
```

A better way to visualize our decision tree is to use the library `rpart.plot`

```
library(rpart.plot)
scooby.fit %>%
  extract_fit_engine() %>%
  rpart.plot()
```

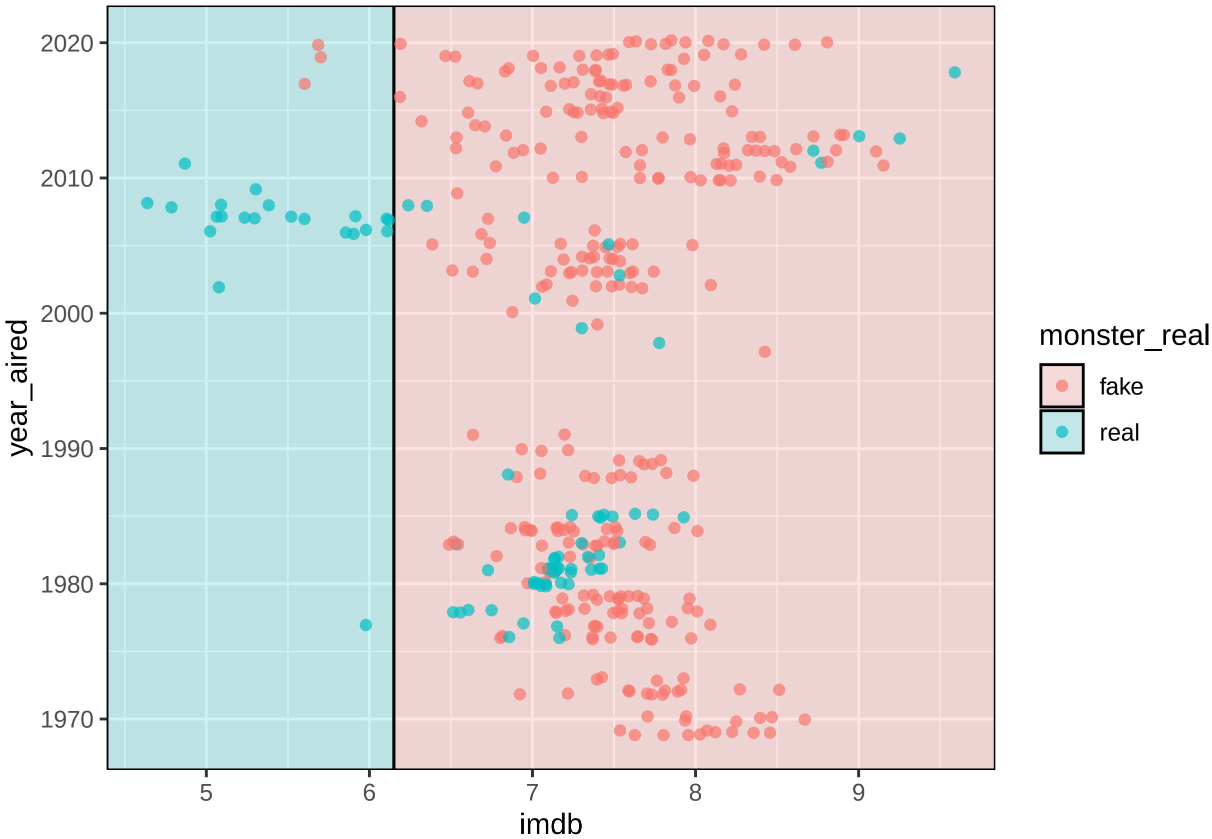


1. a. Talk for a couple of minutes to the people in your group about how to interpret all of the elements of the previous visualization.

   b. Look in page 336 of ISLR for definition of the Gini index and entropy. Why are those two quantities a measure of the "purity" of your partition? In what context are those two measures used?

Another way to visualize the data when you have only two predictors is to use the library `partree` as below

```
library(parttree)
scooby.train.tbl %>%
  ggplot(aes(imdb, year_aired)) +
  geom_parttree(data = scooby.fit,
                aes(fill = monster_real), alpha = 0.2) +
  geom_jitter(alpha = 0.7, width = 0.05,
              height = 0.2, aes(color = monster_real))
```



2. Calculate the accuracy and the confusion matrix using your testing dataset

```
augment(scooby.fit, new_data=scooby.test.tbl ) %>%
   accuracy(truth = monster_real, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.817
```

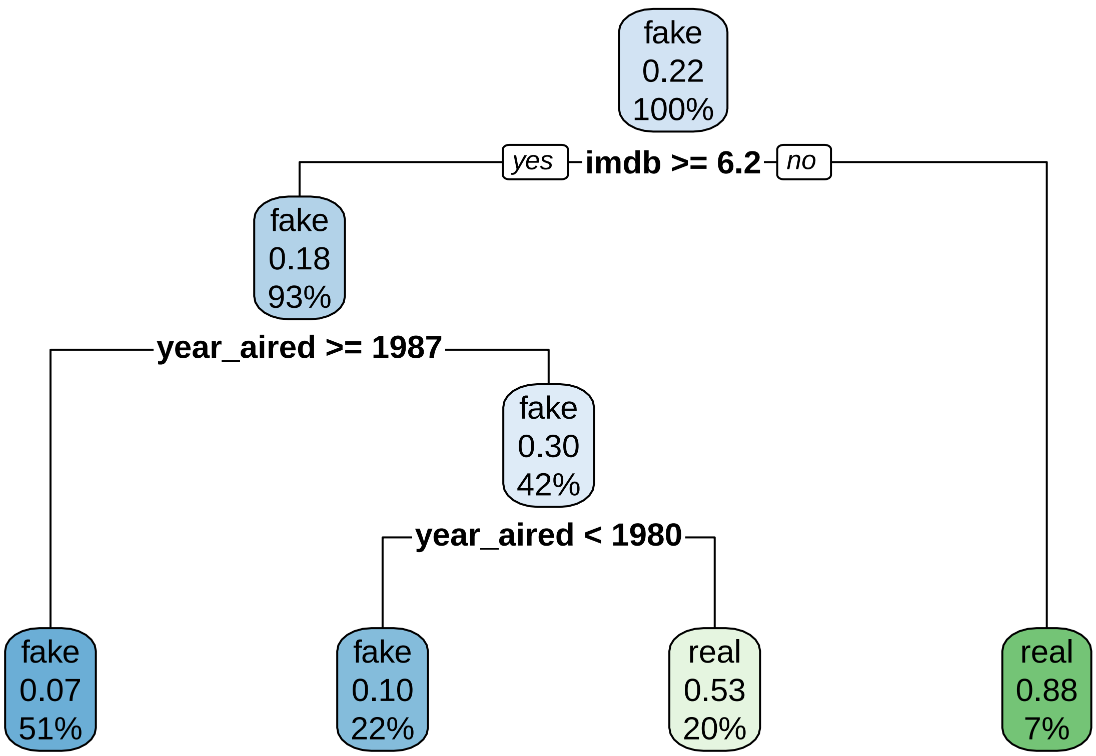
```
augment(scooby.fit, new_data=scooby.test.tbl ) %>%
  conf_mat(truth = monster_real, estimate = .pred_class)
```

```
##           Truth
## Prediction fake real
##       fake   96   21
##       real    2    7
```

3. Calculate the accuracy of your model on your testing dataset for `tree_depth` values of 3, 5, 10 and visualize your models using `rpart.plot`. How do the different models compare to each other? Make sure to define and use a function that takes `tree_depth` as parameter

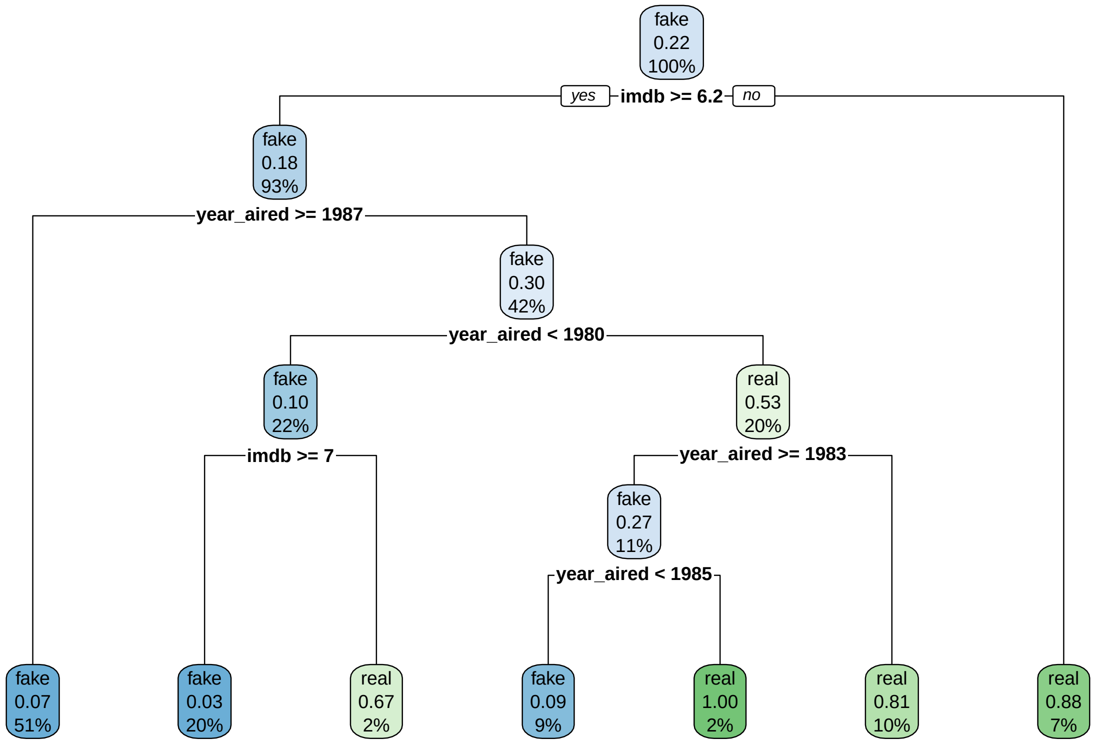
```
test_tree <- function (depth) {
  scooby.model <-
  decision_tree(tree_depth=depth) %>%
  set_mode("classification") %>%
  set_engine("rpart")
```

```
  scooby.recipe <- recipe(monster_real ~ imdb+year_aired,
                 data=scooby.train.tbl)

  scooby.wflow <- workflow() %>%
    add_recipe(scooby.recipe) %>%
    add_model(scooby.model)

  scooby.fit <- fit(scooby.wflow, scooby.train.tbl)

  scooby.fit %>%
  extract_fit_engine() %>%
  rpart.plot()

  augment(scooby.fit, new_data=scooby.test.tbl ) %>%
   accuracy(truth = monster_real, estimate = .pred_class)

}

test_tree(3)
```




```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.857
```

```r
test_tree(5)
```




```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.913
```

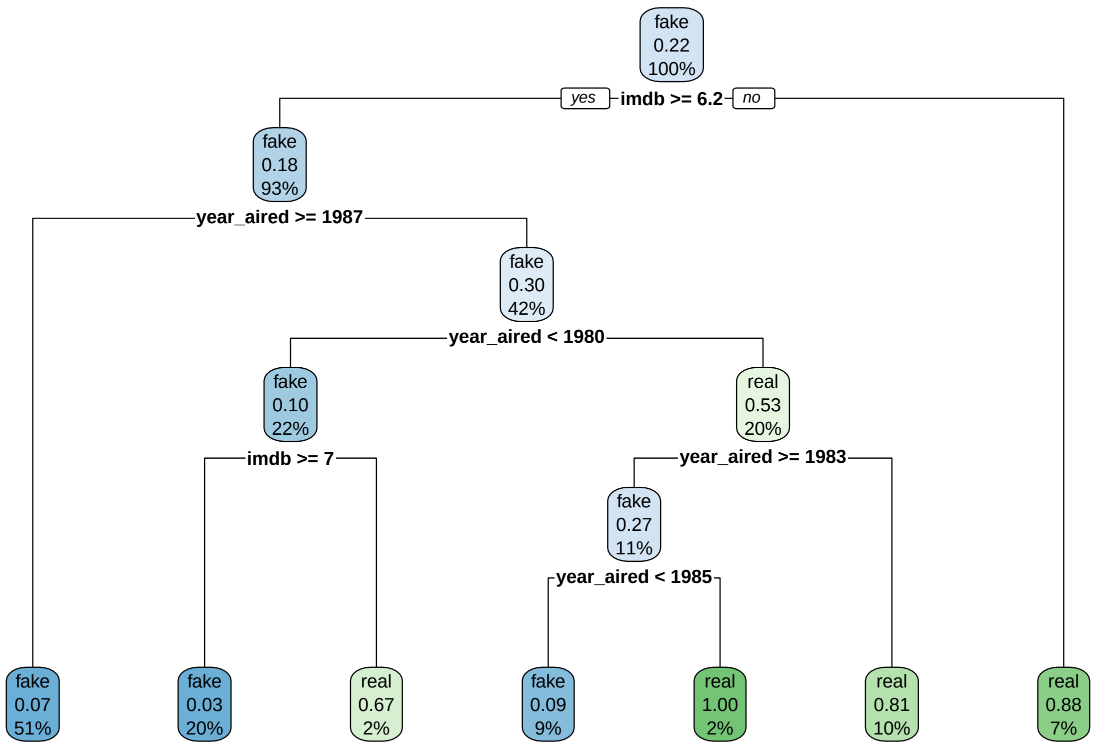
```r
test_tree(10)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.913
```

4. The complexity parameter (`cp` or `cost_complexity`) is a key metric that penalizes the construction of large trees. Create a function `test_cp_tree` which takes as input the complexity parameter and visualizes the model and outputs its accuracy. Test the function for values of `cp`= 0.01, 0.1, 1. What is the effect of the smaller value of `cp` on your tree model?

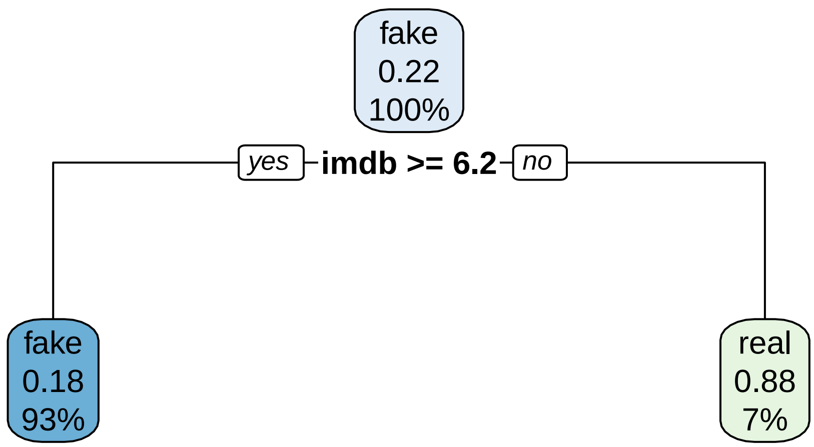
```r
test_cp_tree <- function (cp) {
  scooby.model <-
  decision_tree(cost_complexity=cp) %>%
  set_mode("classification") %>%
  set_engine("rpart")

  scooby.recipe <- recipe(monster_real ~ imdb+year_aired,
                  data=scooby.train.tbl)
```

```
  scooby.wflow <- workflow() %>%
    add_recipe(scooby.recipe) %>%
    add_model(scooby.model)

  scooby.fit <- fit(scooby.wflow, scooby.train.tbl)

  scooby.fit %>%
  extract_fit_engine() %>%
  rpart.plot()

  augment(scooby.fit, new_data=scooby.test.tbl ) %>%
   accuracy(truth = monster_real, estimate = .pred_class)
}

test_cp_tree(0.01)
```



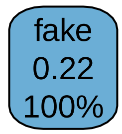

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.913
```

```
test_cp_tree(0.1)
```




```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.817
```

```
test_cp_tree(1)
```




```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.778
```

Optimizing our parameters

We are interested in optimizing our parameters using a cross-validation approach. As before the steps for doing so are:

- Make sure the parameters that you will be optimizing are tuneable.
- Create a cross validation dataset
- Create a grid for searching the parameters in your dataset
- Use `tune_grid` to optimize your function across the values of your grid

```
# Create the model with tuneable parameters
scooby.model <-
  decision_tree(tree_depth=tune(),
                cost_complexity=tune()) %>%
  set_mode("classification") %>%
  set_engine("rpart")

  scooby.recipe <- recipe(monster_real ~ imdb+year_aired,
                 data=scooby.train.tbl)

  scooby.wflow <- workflow() %>%
```

```
    add_recipe(scooby.recipe) %>%
    add_model(scooby.model)
```

```
# Create the cross-validation dataset
set.seed(1234)
scooby.folds <- vfold_cv(scooby.train.tbl, v = 10)
```

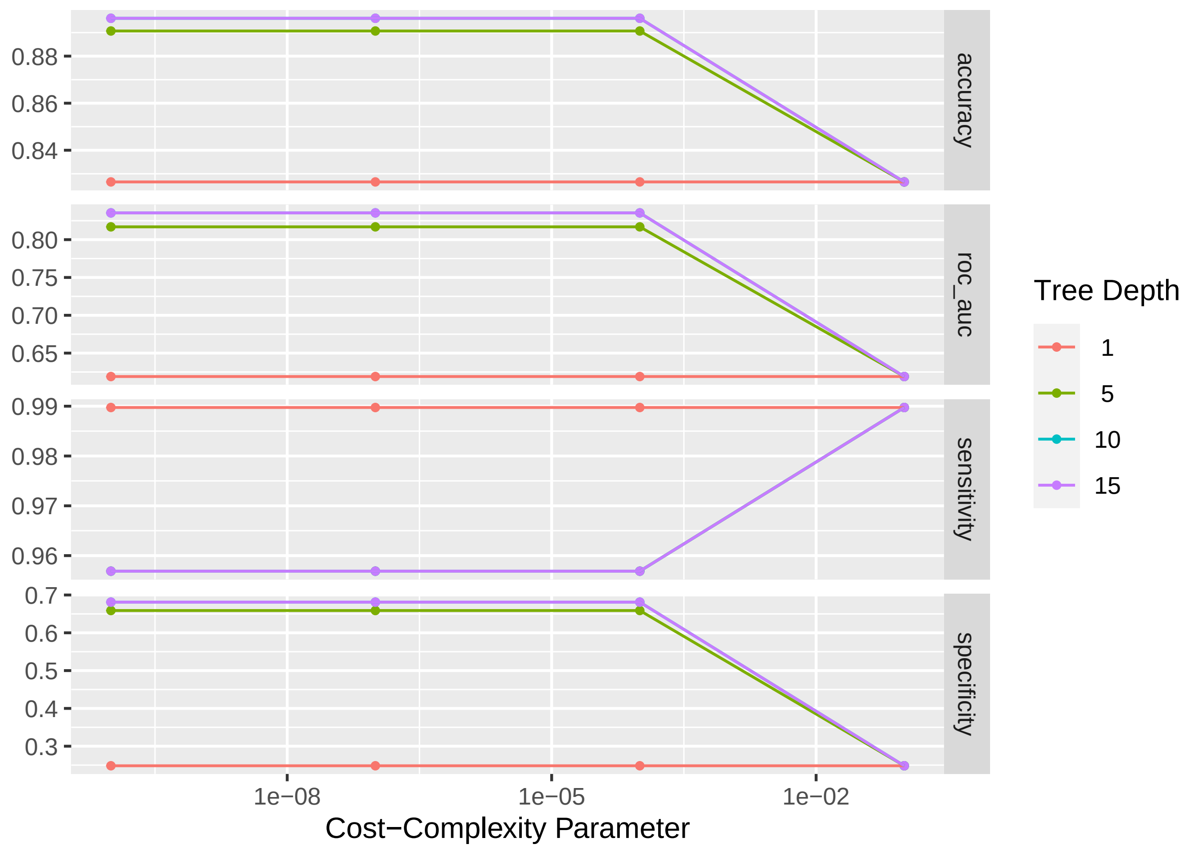
```
#Set up the grid
scooby.grid <-
  grid_regular(cost_complexity(), tree_depth(), levels = 4)

scooby.grid
```

```
## # A tibble: 16 x 2
##    cost_complexity tree_depth
##              <dbl>      <int>
##  1    0.0000000001          1
##  2    0.0000001             1
##  3    0.0001                1
##  4    0.1                   1
##  5    0.0000000001          5
##  6    0.0000001             5
##  7    0.0001                5
##  8    0.1                   5
##  9    0.0000000001         10
## 10    0.0000001            10
## 11    0.0001               10
## 12    0.1                  10
## 13    0.0000000001         15
## 14    0.0000001            15
## 15    0.0001               15
## 16    0.1                  15
```

```
scooby.res <-
  tune_grid(
    scooby.wflow,
    resamples = scooby.folds,
    grid = scooby.grid,
    metrics = metric_set(accuracy, roc_auc, sensitivity, specificity))
```

5. Visualize `scooby.res` and discuss the results with your group members.

```
autoplot(scooby.res)
```



6. Use `select_by_one_std_err` using accuracy as your metric and sorting in descending order the penalty parameter. Check the help of ?select_by_one_std_err and page 236 of ISLR to explain how the "one-standard-error" rule works. Use this parameter to finalize your workflow and fit it. What is your accuracy using your testing dataset?

```
show_best(scooby.res, metric = "accuracy")
```

```
## # A tibble: 5 x 8
##   cost_complexity tree_depth .metric  .estimator  mean     n std_err .config
##             <dbl>      <int> <chr>    <chr>      <dbl> <int>   <dbl> <fct>
## 1    0.0000000001         10 accuracy binary     0.896    10  0.0188 Preprocess~
## 2    0.0000001            10 accuracy binary     0.896    10  0.0188 Preprocess~
## 3    0.0001               10 accuracy binary     0.896    10  0.0188 Preprocess~
## 4    0.0000000001         15 accuracy binary     0.896    10  0.0188 Preprocess~
## 5    0.0000001            15 accuracy binary     0.896    10  0.0188 Preprocess~
```

```
(best.penalty <- select_by_one_std_err(scooby.res,
                                       metric = "accuracy",
                                       -cost_complexity))
```

```
## # A tibble: 1 x 10
##   cost_complexity tree_depth .metric  .estimator  mean     n std_err .config
##             <dbl>      <int> <chr>    <chr>      <dbl> <int>   <dbl> <fct>
## 1          0.0001          5 accuracy binary     0.891    10  0.0197 Preprocess~
## # ... with 2 more variables: .best <dbl>, .bound <dbl>
```

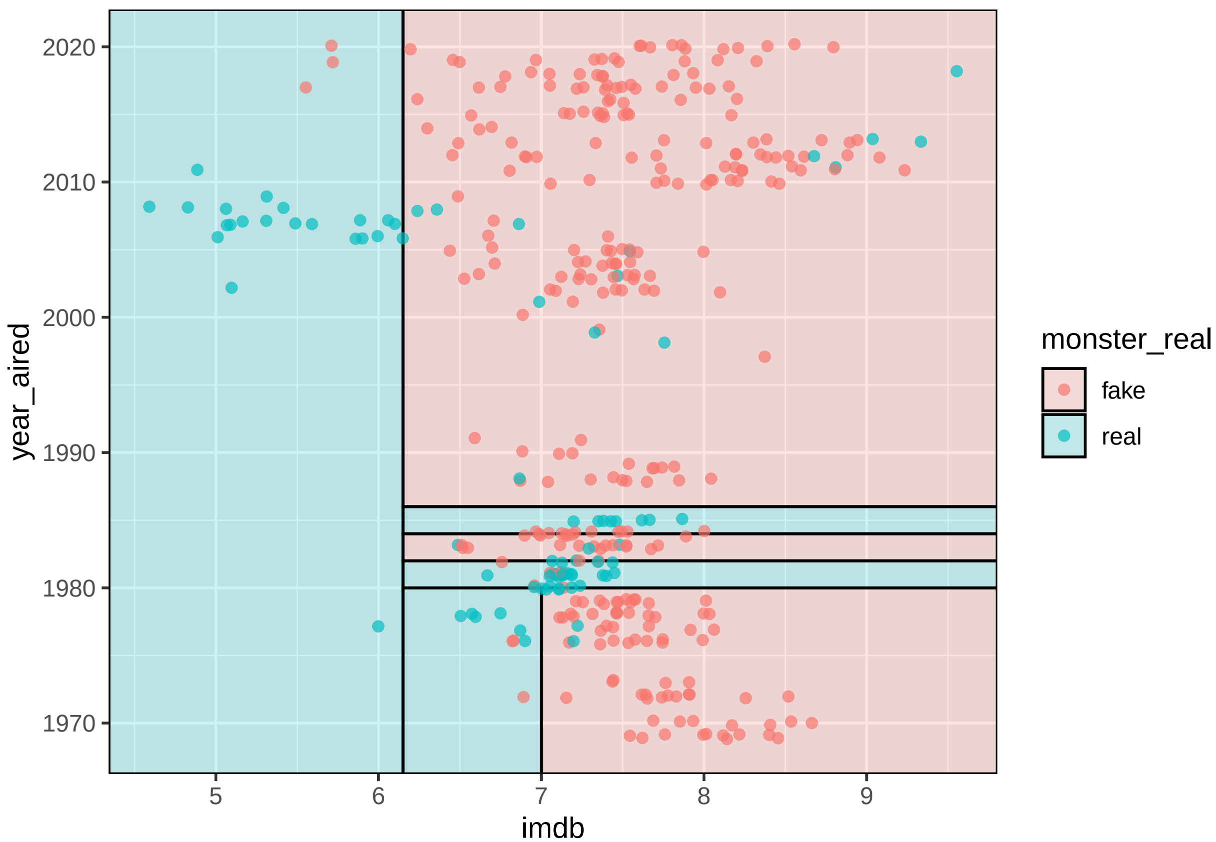
```
scooby.final.wf <- finalize_workflow(scooby.wflow,
                                     best.penalty)
```

```r
scooby.final.fit <- fit(scooby.final.wf,  scooby.train.tbl)

scooby.final.rs <- last_fit(scooby.final.wf,
                         scooby.split)

collect_metrics(scooby.final.rs)
```

```
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <fct>
## 1 accuracy binary         0.913 Preprocessor1_Model1
## 2 roc_auc  binary         0.903 Preprocessor1_Model1
```

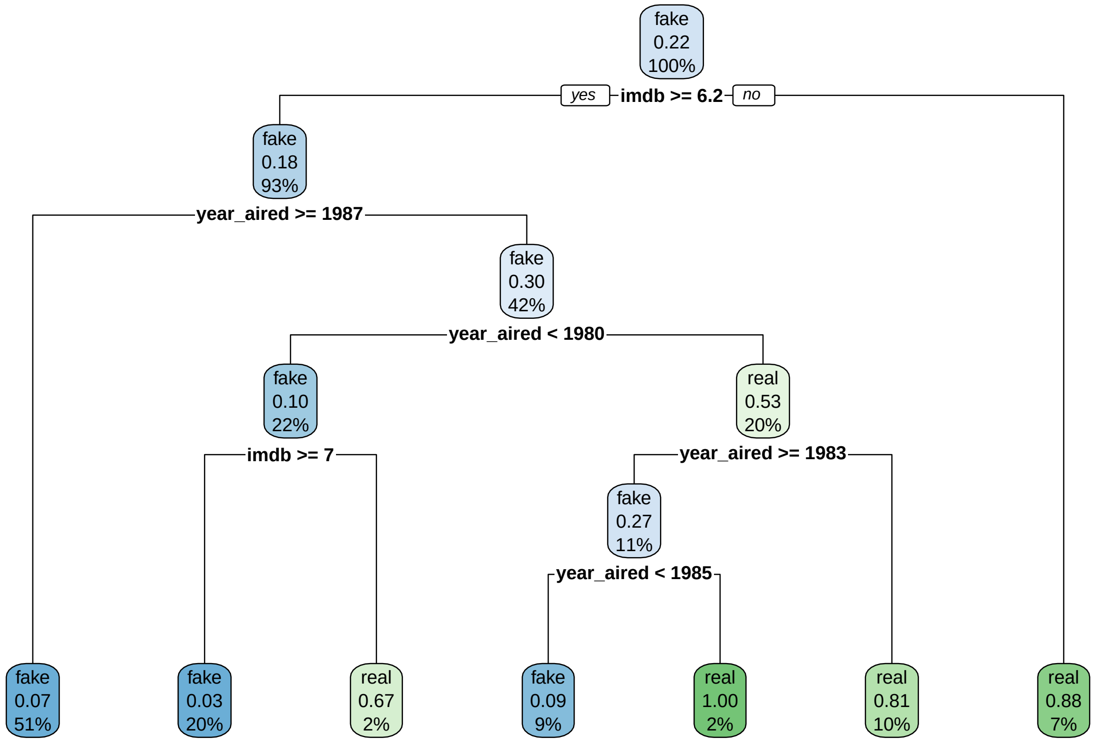
Let's visualize our model on our training dataset using `parttree`

```r
scooby.train.tbl %>%
  ggplot(aes(imdb, year_aired)) +
  geom_parttree(data = scooby.final.fit, aes(fill = monster_real), alpha = 0.2) +
  geom_jitter(alpha = 0.7, width = 0.05, height = 0.2, aes(color = monster_real))
```



And let's look at how the final model looks as a tree

```r
scooby.final.fit %>%
  extract_fit_engine() %>%
  rpart.plot(roundint=FALSE)
```



Acknowledgments

This worksheet draw heavily on the following blog post from Julia Silge: https://juliasilge.com/blog/scooby-doo/