# Explore data

Our modeling goal is to predict the capacity of wind turbines in Canada

based on other characteristics of the turbines from this week’s #TidyTuesday dataset.

Let’s start by reading in the data.

library(tidyverse)

turbines <- read\_csv("https://raw.githubusercontent.com/rfordatascience/tidytuesday/ master/data/2020/2020-10-27/wind-turbine.csv")

turbines

## # A tibble: 6,698 x 15

## objectid province\_territ… project\_name total\_project\_c… turbine\_identif…

##

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ## | 1 | 1 | | Alberta | Optimist Wi… |  | 0.9 | OWE1 |
| ## | 2 | 2 | | Alberta | Castle Rive… |  | 44 | CRW1 |
| ## | 3 | 3 | | Alberta | Waterton Wi… |  | 3.78 | WWT1 |
| ## | 4 | 4 | | Alberta | Waterton Wi… |  | 3.78 | WWT2 |
| ## | 5 | 5 Alberta | | | Waterton Wi… | 3.78 | WWT3 |  |
| ## | 6 | 6 | | Alberta | Waterton Wi… |  | 3.78 | WWT4 |
| ## | 7 | 7 | | Alberta | Cowley North |  | 19.5 | CON1 |
| ## | 8 | 8 | | Alberta | Cowley North |  | 19.5 | CON2 |
| ## | 9 | 9 | | Alberta | Cowley North |  | 19.5 | CON3 |
| ## | 10 | 10 | | Alberta | Cowley North |  | 19.5 | CON4 |
| ## | # | … | with 6,688 more rows, and 10 more variables: | | | | |  |
| ## | # |  | turbine\_number\_in\_project , turbine\_rated\_capacity\_k\_w | | | | | , |
| ## | # |  | rotor\_diameter\_m , hub\_height\_m , manufacturer , | | | | |  |
| ## | # |  | model , commissioning\_date , latitude , longitude , | | | | |  |
| ## | # |  | notes | | | | |  |

Let’s do a bit of data cleaning and preparation.

turbines\_df <- turbines %>% transmute(

turbine\_capacity = turbine\_rated\_capacity\_k\_w, rotor\_diameter\_m,

hub\_height\_m,

commissioning\_date = parse\_number(commissioning\_date), province\_territory = fct\_lump\_n(province\_territory, 10), model = fct\_lump\_n(model, 10)

) %>%

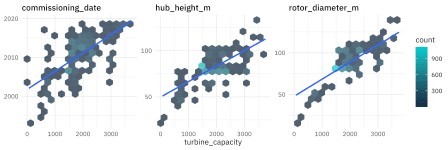
filter(!is.na(turbine\_capacity)) %>% mutate\_if(is.character, factor)

How is the capacity related to other characteristics like the year of commissioning or size of the

turbines?

turbines\_df %>% select(turbine\_capacity:commissioning\_date) %>% pivot\_longer(rotor\_diameter\_m:commissioning\_date) %>% ggplot(aes(turbine\_capacity, value)) +

geom\_hex(bins = 15, alpha = 0.8) + geom\_smooth(method = "lm") + facet\_wrap(~name, scales = "free\_y") + labs(y = NULL) + scale\_fill\_gradient(high = "cyan3")



# Build a model

We can start by loading the tidymodels metapackage, splitting our data into training and testing sets, and creating cross-validation samples.

library(tidymodels)

set.seed(123)

wind\_split <- initial\_split(turbines\_df, strata = turbine\_capacity) wind\_train <- training(wind\_split)

wind\_test <- testing(wind\_split)

set.seed(234)

wind\_folds <- vfold\_cv(wind\_train, strata = turbine\_capacity) wind\_folds

## # 10-fold cross-validation using stratification ## # A tibble: 10 x 2

|  |  |  |
| --- | --- | --- |
| ##  ## |  | splits |
| ## | 1 | Fold01 |
| ## | 2 | Fold02 |
| ## | 3 | Fold03 |
| ## | 4 | Fold04 |
| ## | 5 | Fold05 |
| ## | 6 | Fold06 |
| ## | 7 | Fold07 |
| ## | 8 | Fold08 |
| ## | 9 | Fold09 |

id

## 10 Fold10

Next, let’s create a tunable decision tree model specification.

tree\_spec <- decision\_tree( cost\_complexity = tune(), tree\_depth = tune(), min\_n = tune()

) %>%

set\_engine("rpart") %>% set\_mode("regression")

tree\_spec

## Decision Tree Model Specification (regression) ##

## Main Arguments:

## cost\_complexity = tune() ## tree\_depth = tune()

## min\_n = tune() ##

## Computational engine: rpart

We need a set of possible parameter values to try out for the decision tree.

tree\_grid <- grid\_regular(cost\_complexity(), tree\_depth(), min\_n(), levels = 4)

tree\_grid

## # A tibble: 64 x 3

## cost\_complexity tree\_depth min\_n ##

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ## | 1 | 0.0000000001 | 1 | 2 |
| ## | 2 | 0.0000001 | 1 | 2 |
| ## | 3 | 0.0001 | 1 | 2 |
| ## | 4 | 0.1 | 1 | 2 |
| ## | 5 | 0.0000000001 | 5 | 2 |
| ## | 6 | 0.0000001 | 5 | 2 |
| ## | 7 | 0.0001 | 5 | 2 |
| ## | 8 | 0.1 | 5 | 2 |
| ## | 9 | 0.0000000001 | 10 | 2 |
| ## | 10 | 0.0000001 | 10 | 2 |
| ## | # … | with 54 more rows |  |  |

Now, let’s try out all the possible parameter values on all our resampled datasets. Let’s use some non-default metrics, while we’re at it.

doParallel::registerDoParallel()

set.seed(345)

tree\_rs <- tune\_grid( tree\_spec,

turbine\_capacity ~ ., resamples = wind\_folds, grid = tree\_grid,

metrics = metric\_set(rmse, rsq, mae, mape)

)

tree\_rs

## # Tuning results

## # 10-fold cross-validation using stratification ## # A tibble: 10 x 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ##  ## | splits | id | .metrics | .notes |
| ## 1 | Fold01 |  |  |  |
| ## 2 | Fold02 |  |  |  |
| ## 3 | Fold03 |  |  |  |
| ## 4 | Fold04 |  |  |  |
| ## 5 | Fold05 |  |  |  |
| ## 6 | Fold06 |  |  |  |
| ## 7 | Fold07 |  |  |  |
| ## 8 | Fold08 |  |  |  |
| ## 9 | Fold09 |  |  |  |
| ## 10 | Fold10 |  |  |  |

Notice that we aren’t tuning a workflow() here, as I have often shown how to do. Instead we are tuning the model specification (accompanied by a formula preprocessor); this is so we can use the bare model in some model evaluation activities.

# Evaluate model

Now let’s check out how we did. We can collect or visualize the metrics.

collect\_metrics(tree\_rs)

## # A tibble: 256 x 9 std\_err

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ## | cost\_complexity | tree\_depth | min\_n | .metric | .estimator | mean | n |

##

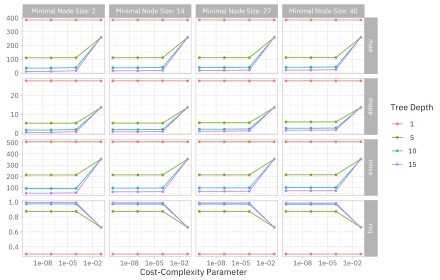
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ## | 1 | 0.0000000001 | 1 | 2 | mae | standard 386. 10 | | |
| 1.50 |  |  |  |  |  |  | | |
| ## | 2 | 0.0000000001 | 1 | 2 | mape | standard 27.7 10 | | |
| 1.30 |  |  |  |  |  |  | | |
| ## | 3 | 0.0000000001 | 1 | 2 | rmse | standard 508. 10 | | |
| 1.44 |  |  |  |  |  |  | | |
| ## | 4 | 0.0000000001 | 1 | 2 | rsq | standard 0.303 10 | | |
| 0.0134 | | | | | | | | |
| ## | 5 | 0.0000001 | 1 | 2 | mae | standard | 386. | 10 |
| 1.50 |  |  |  |  |  |  |  |  |
| ## | 6 | 0.0000001 | 1 | 2 | mape | standard | 27.7 | 10 |
| 1.30 |  |  |  |  |  |  |  |  |
| ## | 7 | 0.0000001 | 1 | 2 | rmse | standard | 508. | 10 |
| 1.44 |  |  |  |  |  |  |  |  |

## 8 0.0000001 1 2 rsq standard 0.303 10

0.0134

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ## | 9 | 0.0001 | 1 | 2 | mae | standard | 386. | 10 |
| 1.50 |  |  |  |  |  |  |  |  |
| ## 1 | 0 | 0.0001 | 1 | 2 | mape | standard | 27.7 | 10 |
| 1.30 |  |  |  |  |  |  |  |  |

## # … with 246 more rows, and 1 more variable: .config autoplot(tree\_rs) + theme\_light(base\_family = "IBMPlexSans")



Looks like this data needs a fairly complex tree!

We can examine or select the best sets of parameter options, chosen by whichever metric we want.

show\_best(tree\_rs, "mape")

## # A tibble: 5 x 9 ## cost\_complexity std\_err

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| tree\_depth | min\_n | .metric | .estimator | mean | n |
| 15 | 2 | mape | standard | 0.564 | 10 |
| 15 | 2 | mape | standard | 0.564 | 10 |
| 15 | 14 | mape | standard | 0.823 | 10 |
| 15 | 14 | mape | standard | 0.823 | 10 |
| 15 | 2 | mape | standard | 0.885 | 10 |

##

## 1 0.0000000001

0.0592

## 2 0.0000001

0.0591

## 3 0.0000000001

0.0547

## 4 0.0000001

0.0547

## 5 0.0001

0.0705

## # … with 1 more variable: .config select\_best(tree\_rs, "rmse")

## # A tibble: 1 x 4

## cost\_complexity tree\_depth min\_n .config

##

## 1 0.0000000001 15 2 Preprocessor1\_Model13

Next, let’s use one of these “best” sets of parameters to update and finalize our model.

final\_tree <- finalize\_model(tree\_spec, select\_best(tree\_rs, "rmse")) final\_tree

## Decision Tree Model Specification (regression) ##

## Main Arguments:

## cost\_complexity = 1e-10 ## tree\_depth = 15

## min\_n = 2 ##

## Computational engine: rpart

This model final\_tree is updated and finalized (no longer tunable) but it is not *fit*. It has all its hyperparameters set but it has not been fit to any data. We have a couple of options for how to fit this model. We can either fit final\_tree to training data using fit() or to the testing/training split using last\_fit(), which will give us some other results along with the fitted output.

final\_fit <- fit(final\_tree, turbine\_capacity ~ ., wind\_train) final\_rs <- last\_fit(final\_tree, turbine\_capacity ~ ., wind\_split)

We can predict from either one of these objects.

predict(final\_fit, wind\_train[144, ])

## # A tibble: 1 x 1 ## .pred

##

## 1 1800

predict(final\_rs$.workflow[[1]], wind\_train[144, ]) ## # A tibble: 1 x 1

## .pred ##

## 1 1800

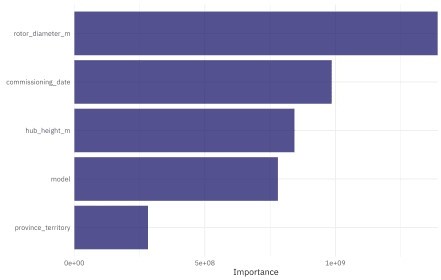
What are the most important variables in this decision tree for predicting turbine capacity?

library(vip)

final\_fit %>%

vip(geom = "col", aesthetics = list(fill = "midnightblue", alpha = 0.8)) +

scale\_y\_continuous(expand = c(0, 0))



I really like the

parttree package for visualization decision tree results. It only works for models with one or two predictors, so we’ll have to fit an example model that isn’t quite the same as our full model. It can still help us understand how this decision tree is working, but keep in mind that it is not the same as our full model with more predictors.

library(parttree)

ex\_fit <- fit( final\_tree,

turbine\_capacity ~ rotor\_diameter\_m + commissioning\_date, wind\_train

)

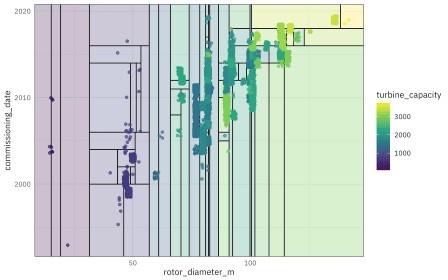
wind\_train %>%

ggplot(aes(rotor\_diameter\_m, commissioning\_date)) + geom\_parttree(data = ex\_fit, aes(fill = turbine\_capacity), alpha =

0.3) +

geom\_jitter(alpha = 0.7, width = 1, height = 0.5, aes(color = turbine\_capacity)) +

scale\_colour\_viridis\_c(aesthetics = c("color", "fill"))



Finally, let’s turn to the testing data! These results are stored in final\_rs, along with the fitted output there. We can see both metrics on the testing data and predictions.

collect\_metrics(final\_rs)

## # A tibble: 2 x 4

## .metric .estimator .estimate .config ##

## 1 rmse standard 73.6 Preprocessor1\_Model1

## 2 rsq standard 0.985 Preprocessor1\_Model1

final\_rs %>% collect\_predictions() %>%

ggplot(aes(turbine\_capacity, .pred)) +

geom\_abline(slope = 1, lty = 2, color = "gray50", alpha = 0.5) + geom\_point(alpha = 0.6, color = "midnightblue") +

coord\_fixed()

