tidymodels and recipes

Max Kuhn

tidymodels

> The tidymodels framework is a collection of packages for modeling and machine learning using tidyverse principles.

Our goals:

- Enable users to fall into the "pit of success".
- Provide a pleasant and rational interface to many types of models.
- Provide guardrails, sometimes invisibly, that can prevent methodology errors.

tidymodels.org

Tidy Modeling with R (tmwr.org)

applicable
Baguette
broom
butcher
dials
discrim
embed
finetune
hardhat
parsnip
recipes
rsample
RULES
spatialsample
stacks
TEXT RECIPES
tidymodels
tidyposterior
TUNE
%>%
WORKFLOWS
yardstick

An example - thyroid disorder

A very old data of 3,772 data points for binary classification to predict people who are sick from thyroid disease.

- Mostly binary predictors
- Class imbalance of about 6%
- Significant amounts of missing data

We will hold back 25% of the data for testing model performance.

Getting started

```r
library(tidymodels)
```

```
## ── Attaching packages ────────────────────────────────────── tidymodels 0.1.4 ──
```

```
## ✓ broom        0.7.11     ✓ recipes      0.1.17
## ✓ dials        0.1.0      ✓ rsample      0.1.1
## ✓ dplyr        1.0.8      ✓ tibble       3.1.6
## ✓ ggplot2      3.3.5      ✓ tidyr        1.2.0
## ✓ infer        1.0.0      ✓ tune         0.1.6
## ✓ modeldata    0.1.1      ✓ workflows    0.2.4
## ✓ parsnip      0.1.7      ✓ workflowsets 0.1.0
## ✓ purrr        0.3.4      ✓ yardstick    0.0.9
```

```
## ── Conflicts ───────────────────────────────────────── tidymodels_conflicts() ──
## x purrr::accumulate() masks foreach::accumulate()
## x purrr::discard()    masks scales::discard()
## x dplyr::filter()     masks stats::filter()
## x dplyr::lag()        masks stats::lag()
## x recipes::step()     masks stats::step()
## x purrr::when()       masks foreach::when()
## • Search for functions across packages at https://www.tidymodels.org/find/
```

```r
library(themis)     # <- for class imbalance sampling
```

Getting started

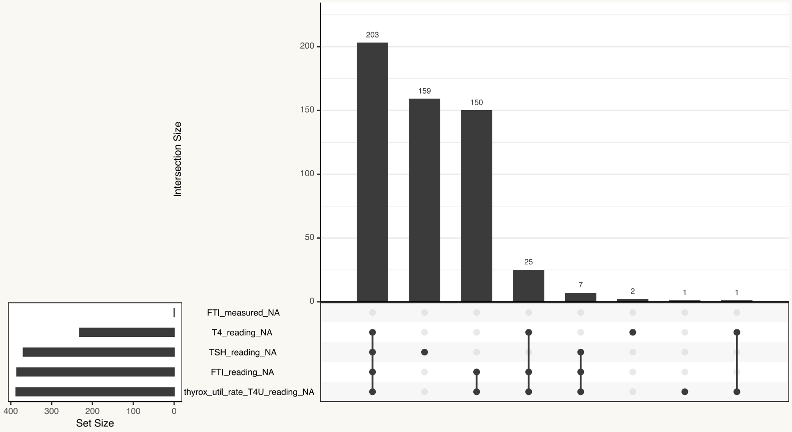
```r
tidymodels_prefer()
theme_set(theme_bw())

# training/test split
set.seed(1)
thy_split <- initial_split(thyroid_disease, strata = class)
thy_train <- training(thy_split)
thy_test  <- testing(thy_split)

# 10-fold cross-validation scheme
thy_folds <- vfold_cv(thy_train, strata = class)

# Columns with missing values
miss_cols <- c("T4_reading", "TSH_reading", "FTI_reading", "thyrox_util_rate_T4U_reading")
```

Missing value patterns



A data recipe

Recipes is a package that can prepare data for modeling.

- Think of it as a mash-up of `model.matrix()` and `dplyr`.
- It is well integrated with the rest of tidymodels.
- See TMwR Chapters 8 and 16

We'll use this to deal with some of the important aspects of these data.

Recipes are also a *great* tool for feature engineering (but we won't do much of that here).

A list of all known recipe steps

What is feature engineering?

First things first: what's a feature?

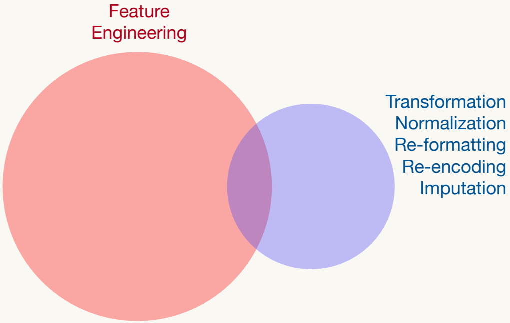
I tend to think of a feature as some representation of a predictor that will be used in a model.

Old-school features:

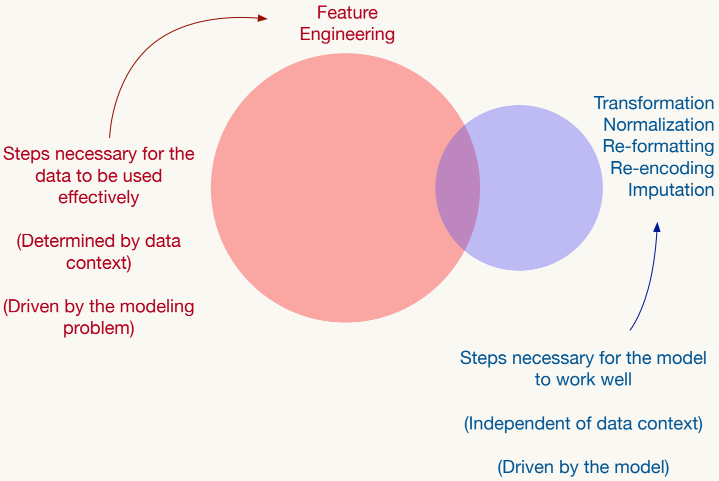
- Interactions
- Polynomial expansions/splines
- PCA feature extraction

"Feature engineering" sounds pretty cool, but let's take a minute to talk about *preprocessing* data.

Two types of preprocessing



Two types of preprocessing


Feature Engineering
Steps necessary for the data to be used effectively
(Determined by data context)
(Driven by the modeling problem)
Transformation
Normalization
Re-formatting
Re-encoding
Imputation
Steps necessary for the model to work well
(Independent of data context)
(Driven by the model)

Original column

Features

(At least that's what we hope the difference looks like.)

Starting the recipe

```r
thyr_rec <-
  recipe(class ~ ., data = thy_train)
```

Based on the formula, the function assigns columns to roles of "outcome" or "predictor"

Estimating a transformation of the data

```r
thyr_rec <-
  recipe(class ~ ., data = thy_train) %>%
  step_YeoJohnson(TSH_reading)
```

This transformation requires a parameter for estimation and can help with skewed data distributions.

Imputing missing data

```r
thyr_rec <-
  recipe(class ~ ., data = thy_train) %>%
  step_YeoJohnson(TSH_reading) %>%
  step_impute_knn(all_of(miss_cols), neighbors = 5)
```

Other possible options: `step_impute_bag()`, `step_impute_linear()`, `step_impute_lower()`, `step_impute_mean()`, `step_impute_median()`, `step_impute_mode()`, `step_impute_roll()`

We might want to *optimize* the number of neighbors to use for imputation. In that case we can tag it using `neighbors = tune()`.

Create dummy variables

```r
thyr_rec <-
  recipe(class ~ ., data = thy_train) %>%
  step_YeoJohnson(TSH_reading) %>%
  step_impute_knn(all_of(miss_cols), neighbors = 5) %>%
  step_dummy(all_nominal_predictors())
```

Recipes do not automatically create indicator variables from categorical (a.k.a. nominal) predictors.

There are a *lot* of tools for this in recipers. See TMwR.

Dealing with the class imbalance

```r
thyr_rec <-
  recipe(class ~ ., data = thy_train) %>%
  step_YeoJohnson(TSH_reading) %>%
  step_impute_knn(all_of(miss_cols), neighbors = 5) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_smote(class, neighbors = 10)
```

SMOTE is a method to synthesize new samples from the classes to enrich the minority class.

It's not a great solution and is fairly controversial (in statistics, at least).

Again, we can optimize the number of SMOTE neighbors too.

The themis package has other tools for imbalances and some models can be used in cost-sensitive learning.

Eliminate zero-variance predictors

```r
thyr_rec <-
  recipe(class ~ ., data = thy_train) %>%
  step_YeoJohnson(TSH_reading) %>%
  step_impute_knn(all_of(miss_cols), neighbors = 5) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_smote(class, neighbors = 10) %>%
  step_zv(all_predictors())
```

This removes columns that have a single unique value (i.e., zero variance).

A basic logistic regression

```
ctrl <- control_resamples(save_pred = TRUE)
lr_res <-
  logistic_reg() %>%
  fit_resamples(thyr_rec, resamples = thy_folds, control = ctrl)

show_best(lr_res, metric = "roc_auc")
```

```
## # A tibble: 1 × 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 roc_auc binary     0.872    10  0.0125 Preprocessor1_Model1
```

Both the recipe and model are re-estimated 10 times since we are using 10-fold CV.

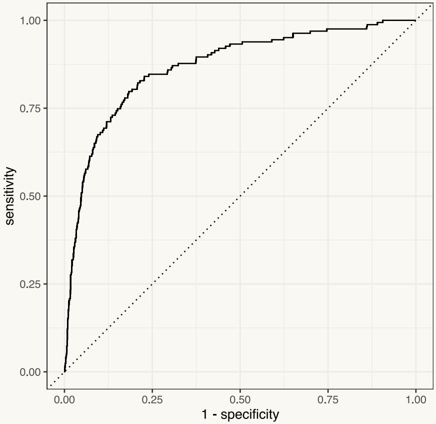
If we had used `tune()` for any of the model or recipe parameters, there are a set of tuning functions: `tune_grid()`, `tune_bayes()`, `tune_race_anova()`, and others.

Other notes

- These models fits are independent of one another. Parallel processing can be used to significantly speed up the training process.
- The individual models can be saved so you can look at variation in the model parameters or recipes steps.
- If you are interested in a validation set, tidymodels considers that a single resample of the data. Everything else in this chapter works the same.
- Formulas can also be used to specify the model terms.

Resampled ROC curve

```
lr_res %>%
  collect_predictions() %>%
  roc_curve(class, .pred_sick) %>%
  autoplot()
```



tidymodels

There's a lot more to talk about with tidymodels, but this example is emblematic of the power and security of the framework.

We'll finish up with David Robinson:

> Some of the resistance I've seen to tidymodels comes from a place of "This makes it too easy- you're not thinking carefully about what the code is doing!" But I think this is getting it backwards. By removing the burden of writing procedural logic, I get to focus on scientific and statistical questions about my data and model.

Thanks

Thanks for the invitation to speak today!

The tidymodels team: Davis Vaughan, Julia Silge, Hanna Frick, and Emil Hvitfeldt.

Special thanks for the other folks who contributed so much to tidymodels: Edgar Ruiz, Alison Hill, Desirée De Leon, Simon Couch, Alex Hayes, Mine Cetinkaya-Rundel, and the tidyverse team.

These slides were made with the `xaringan` package and styled by Alison Hill.