Applications

Of Data Science

Feature Engineering

Applications of Data Science - Class 15

Giora Simchoni

gsimchoni@gmail.com and add #dsapps in subject

Stat. and OR Department, TAU

2022-01-02

What can you possibly teach us about Feature Engineering?

Applications

Of Data Science

Our two previous modeling attempts, with Modeling in the Tidyverse and in The Trees where problematic.

- Using a "sinked" split, not using `set.seed()` - reproducible but wasteful
- "Lazy" feature engineering, not using the entire data, not creative
- No interactions
- Test once, on the same validation set, for everything
- Data "leakage" from training set to validation set (when imputing missing values)
- Sometimes tuning params, sometimes not
- No pre-processing of features, if at all by a hunch
- Not thinking about Multicollinearity where it hurts (regression)
- Almost no treatment of class imbalance in classification
- Chaotic feature selection

1. Predict the income of an OKCupid user (from those who reported income)
2. Predict if an OKCupid user is a Cats or Dogs person, where:

```
cats_categories <- c("has cats", "likes cats", "dislikes dogs and
                     "dislikes dogs and likes cats")
dogs_categories <- c("has dogs", "likes dogs", "has dogs and disli
                     "likes dogs and dislikes cats")
```

And 80% of users are Dogs people, so we have class imbalance.

On our validation set we got...

- Regression Task

Model	RMSE	R
LM	0.35	0.5
LM-L2	0.35	0.5
LM-L1	0.35	0.5
CART	0.36	0.45
Bagging	0.37	0.44
RF	0.35	0.49
GBT-gbm	0.35	0.5
GBT-XGBOOST	0.35	0.5

- Classification Task

Model	AUC	Accuracy	Recall Dogs	Precision Dogs	Recall Cats	Precision Cats
GLM	0.74	0.68	0.68	0.92	0.66	0.26
GLM-EN	0.74	0.67	0.67	0.92	0.67	0.26
CART	0.67	0.75	0.79	0.9	0.49	0.28
GBT+Cutoff	0.74	0.70	0.71	0.9	0.65	0.32

And we were not impressed.

Resampling/Cross-Validation

Applications

Of Data Science

Resampling done right

The main goal of resampling: No Surprises.

(By generating multiple versions of our data, and watching how our model behaves on unseen fresh data again. And again. And again.)

- Resampling is a key principle when developing a ML solution
- All ML practitioners are aware of the menace of overfitting
- But resampling should also be made for:
  - Feature engineering and transformation (should I use the Box-Cox or Yeo-Johnson transformation?)
  - Feature selection (Should I use 2nd-order interactions here?)
  - Choosing imputation strategy (Simply use the median or regress?)
  - Tuning parameters (100 or 500 trees?)
  - Model selection (LM or MARS?)

- A good resampling scheme will:
  - Have more than one train/valid/test samples!*
  - Be stratified (usually by dependent variable distribution)
  - Use a (paired/repeated/within) statistical test on resamples to reach a decision, or at least a proper plot
  - Avoid data "leakage" and not include the test set!
  - Encompass all steps in model building
  - Ideally (when data is Big) perform each step on a different part *of the training data*

💡 * How many resamples?

– The higher the number the lower the variance of performance metric

– The higher the number the less data used in each resample, the higher the bias of performance metric

Ideally (yes, I know)

And don't forget

Only once we decide on the final pre-processing, features, model and parameters, will we fit the model two more times:

1. On the entire training data, then test on the testing data and get a realistic metric of performance
2. On the *entire* data, right before deployment to production

Many approaches to resampling

But two most common:

1. V-fold Cross Validation (+ optionally repeating n times)
2. Bootstrap Samples

V-fold Cross Validation

[Applied Machine Learning Workshop at Rstudio-conf 2019 / Max Kuhn]

Bootstrap Samples

What is the probability of entering a fold (at least once)?

Let's Start from the Very Beginning

Applications

Of Data Science

OkCupid from scratch

```
okcupid <- read_csv("~/okcupid.csv.zip")

cats_categories <- c("has cats", "likes cats", "dislikes dogs and
                     "dislikes dogs and likes cats")
dogs_categories <- c("has dogs", "likes dogs", "has dogs and disli
                     "likes dogs and dislikes cats")

okcupid <- okcupid %>%
  mutate(pets = case_when(
           pets %in% cats_categories ~ "cats",
           pets %in% dogs_categories ~ "dogs",
           TRUE ~ NA_character_)) %>%
  drop_na(pets) %>%
  na_if(-1) %>%
  select(-last_online)

okcupid[which(is.na(okcupid$height)), "height"] <- 70 # you don't
```

Using `rsample::initial_split(okcupid, strata = pets)` would have been better, but if we do it now we won't have the same test data to compare.

```
idx <- read_rds("../data/okcupid3_idx.rda")
train_idx <- idx$train_idx
valid_idx <- idx$valid_idx
test_idx <- idx$test_idx

okcupid_train <- okcupid[train_idx, ]
okcupid_valid <- okcupid[valid_idx, ]
okcupid_test <- okcupid[test_idx, ]
```

And let us bind `okcupid_train` and `okcupid_valid` together because we are going to use resampling anyway:

```
okcupid_train <- bind_rows(okcupid_train, okcupid_valid)

glue("train no. of rows: {nrow(okcupid_train)}
     test no. of rows: {nrow(okcupid_test)}")
```

```
## train no. of rows: 12729
## test no. of rows: 4000
```

See that the class balance is more or less the same:

```
okcupid_train %>% count(pets) %>% mutate(pct = n / sum(n))
```

```
## # A tibble: 2 x 3
##   pets      n   pct
##   <chr> <int> <dbl>
## 1 cats   2097 0.165
## 2 dogs  10632 0.835
```

```
okcupid_test %>% count(pets) %>% mutate(pct = n / sum(n))
```

```
## # A tibble: 2 x 3
##   pets      n   pct
##   <chr> <int> <dbl>
## 1 cats    693 0.173
## 2 dogs   3307 0.827
```

Good enough (we could also drop a few cat people at random if they're too much in the test group).

We stash our test data aside, kiss it good bye, we shall not see it again before the final assessment of our model.

That manager again.

Manager mentions casually: oh, and by the way, this model thingy of yours. Yeah, we need it to run in, like 10 milliseconds per user. Marketing team needs that, don't ask me why, and we can't have the users wait.

You: penalized logistic regression it is 😉

No one mentioned cats/dogs recall/precision to be of any particular importance, so we decide to maximize AUC and take care of cutoff later.

We explore the training data (skipping here at least a couple of hours in which we actually explore the data, I'm assuming we already know quite a bit!):

```
glimpse(okcupid_train)
```

```
## Rows: 12,729
## Columns: 30
## $ age         <dbl> 39, 19, 27, 24, 29, 24, 25, 25, 21, 25, 28, 25, 21,
## $ body_type   <chr> "fit", "curvy", NA, "fit", "fit", "fit", "athletic",
## $ diet        <chr> "mostly anything", "mostly vegetarian", "mostly anytl
## $ drinks      <chr> "socially", "rarely", NA, "socially", "socially", "s
## $ drugs       <chr> "never", "sometimes", NA, NA, "never", NA, "never",
## $ education   <chr> "graduated from college/university", NA, NA, "gradua
## $ essay0      <chr> "i am a very positive person who loves to laugh and
## $ essay1      <chr> "i work in the health and wellness industry and i tr
## $ essay2      <chr> "making people smile or laugh.", "ummmm everything?
## $ essay3      <chr> "my eyes", "my boobs. definitely.<br />\nand the fac
## $ essay4      <chr> "i enjoy malcolm gladwell, blink, tipping point, out
## $ essay5      <chr> "water,steak,chocolate,comfy jeans,transportation,fr
## $ essay6      <chr> "food, naps, women, and health", "caffeine, my baby
## $ essay7      <chr> "doing whatever has come up or has been planned. din
## $ essay8      <chr> "i am totally deaf in my right ear", "i love animals
## $ essay9      <chr> "you enjoy having fun, love to laugh, have a positiv
## $ ethnicity   <chr> "white", "white", NA, "white", "asian, pacific islan
## $ height      <dbl> 70, 60, 72, 65, 69, 72, 70, 67, 66, 67, 63, 63, 62,
## $ income      <dbl> 8e+04, NA, NA, NA, 3e+04, 5e+04, 1e+05, NA, 2e+04, N
## $ job         <chr> "medicine / health", "sales / marketing / biz dev",
## $ location    <chr> "mountain view, california", "pleasant hill, califor
```

```r
library(naniar)

vis_miss(okcupid_train %>%
           sample_frac(0.2) %>%
           select(-starts_with("essay")))
```

Q & A

Q: What type of variables do we have?

A: Numeric: age, height income; Text: essays; Categorical: all others

Q: Can our model(s) handle missing values?

A: Probably gonna use `glmnet` so: NO.

Q: Are missing values a serious issue here?

A: Oh yes. `income` 80+% missing, `offspring` 50+% missing. Gonna have to take care of it.

Q & A

Q: Can our model(s) handle categorical values?

A: Probably gonna use `glmnet` so: NO.

Q: Categorical variables with many levels?

A: Oh yes. `speaks` has over 2K. And there are those `essay`s.

Q: Numeric variables skewed?

A: Income, but all you need is log.

Q: Classes imbalance scenario?

A: Pretty imbalanced, 5 dogs people for every 1 cat person. Gonna have to take care of it.

You have to have a strategy

Strategy: Divide training data into 6 samples for 6 main decision/steps/tunings:

1. Missing data imputation
2. Class imbalance treatment
3. Feature Engineering for `essay` data
4. Looking into interactions
5. Model Tuning
6. Cutoff choice

> 💡 We assume these decisions are additive or independent of each other. Does it have to be and what can we do if we suspect not? (Remember we are Statisticians!)

Looking at our strategy, we decide the 12.7K observations need not split evenly between steps.

- Cutoff choice is "peanuts" compared to the other decisions, let's decide in advance that at this stage we would fit the chosen model on ~12K previously seen observations and choose a cutoff by looking at the remaining ~700 observations
- Missing data imputation, class imbalance and interaction steps will each be given 2K observations
- Model tuning 3K observations
- Text feature engineering and selection 3K observations
- Total: 12,729 training observations

We wish to make sure our samples are stratified by `pets`, so we'll use `initial_split()` in stages:

```
library(tidymodels)

set.seed(42)
ok_cutoff_split <- initial_split(okcupid_train,
                                 strata = pets, prop = (729 - 1)/1
ok_tr_cutoff <- training(ok_cutoff_split)
ok_train <- testing(ok_cutoff_split)

set.seed(42)
ok_missing_split <- initial_split(ok_train,
                                  strata = pets, prop = (2000 - 1)
ok_tr_missing <- training(ok_missing_split)
ok_train <- testing(ok_missing_split)

set.seed(42)
ok_imbalance_split <- initial_split(ok_train,
                                    strata = pets, prop = (2000 - 1)
ok_tr_imbalance <- training(ok_imbalance_split)
ok_train <- testing(ok_imbalance_split)
```

```r
set.seed(42)
ok_interaction_split <- initial_split(ok_train,
                                      strata = pets, prop = (2000
ok_tr_interaction <- training(ok_interaction_split)
ok_train <- testing(ok_interaction_split)

set.seed(42)
ok_tuning_split <- initial_split(ok_train,
                                 strata = pets, prop = (3000 - 1)
ok_tr_tuning <- training(ok_tuning_split)
ok_tr_eng <- testing(ok_tuning_split)
```

```
## # A tibble: 7 x 4
##   name        df                    n_rows cats_rate
##   <chr>       <list>                 <dbl>     <dbl>
## 1 missing     <tibble [1,998 x 30]>   1998     0.165
## 2 imbalance   <tibble [1,999 x 30]>   1999     0.165
## 3 engineering <tibble [3,005 x 30]>   3005     0.165
## 4 interaction <tibble [1,999 x 30]>   1999     0.165
## 5 tuning      <tibble [3,001 x 30]>   3001     0.165
## 6 cutoff      <tibble [727 x 30]>      727     0.164
## 7 test        <tibble [4,000 x 30]>   4000     0.173
```

Dealing with Missing Data

Applications

Of Data Science

Missing Data

We could fill an entire semester on this subject!

- What causes a missing datum?
  - Survey data
  - Merging data sources
  - Failure of measurement

- What is the missing data mechanism?
  - See Little & Rubin (2019) seminal work
  - MCAR: "on some days the weighing scale batteries drained and we couldn't take the mice weight"
  - MAR: "on the 2024 elections poll a massive Facebook campaign called all women to refuse to cooperate with the Seker-Sheker company due to allegations of sexual harassment by the company's head Dani Marom"
  - MNAR: "some of the drug addicts under study missed the final few meetings in the treatment group and their data were discarded"

> 💡 How would you asses the missing data mechanism in your dataset?

Exploring percent missingness

```
gg_miss_var(ok_tr_missing %>%
              select(-starts_with("essay")),
            show_pct = TRUE)
```

Exploring percent missingness by a predictor

```
gg_miss_var(ok_tr_missing %>%
              select(-starts_with("essay")),
            facet = sex, show_pct = TRUE)
```

Exploring missingness between a predictor and the response (categorical) variable

```
ok_tr_missing %>%
    mutate(income_missing = is.na(income)) %>%
    count(income_missing, pets) %>%
  pivot_wider(id_cols = pets,
              names_from = income_missing,
              values_from = n) %>%
  mutate(pct_missing = `TRUE` / (`TRUE` + `FALSE`))
```

```
## # A tibble: 2 x 4
##   pets  `FALSE` `TRUE` pct_missing
##   <chr>   <int>  <int>       <dbl>
## 1 cats       88    241       0.733
## 2 dogs      304   1365       0.818
```

But we can do better

Exploring missingness between two (categorical) predictors

```
library(ggmosaic)

ok_tr_missing %>%
  mutate(body_type_missing = is.na(body_type)) %>%
  ggplot() +
  geom_mosaic(aes(x = product(body_type_missing, status),
                  fill = body_type_missing)) +
  labs(x = "", y = "") + theme_light() + theme(axis.text.y=element
```

You don't always need a plot...

```
ok_tr_missing %>%
    mutate(body_type_missing = is.na(body_type)) %>%
    count(body_type_missing, status) %>%
  pivot_wider(id_cols = status,
              names_from = body_type_missing,
              values_from = n) %>%
  mutate(pct_missing = `TRUE` / (`TRUE` + `FALSE`))
```

```
## # A tibble: 4 x 4
##   status         `FALSE` `TRUE` pct_missing
##   <chr>            <int>  <int>       <dbl>
## 1 available           52      9      0.148
## 2 married              6      2      0.25
## 3 seeing someone      44     14      0.241
## 4 single            1720    151      0.0807
```

The same with `body_type` missingness by `drugs`:

```
ok_tr_missing %>%
    mutate(body_type_missing = is.na(body_type),
           drugs_missing = is.na(drugs)) %>%
    count(body_type_missing, drugs_missing) %>%
  pivot_wider(id_cols = drugs_missing,
              names_from = body_type_missing,
              values_from = n) %>%
  mutate(pct_missing = `TRUE` / (`TRUE` + `FALSE`))
```

```
## # A tibble: 2 x 4
##   drugs_missing `FALSE` `TRUE` pct_missing
##   <lgl>           <int>  <int>       <dbl>
## 1 FALSE            1402    112      0.0740
## 2 TRUE              420     64      0.132
```

But we can do better with a co-occurrence plot or a heatmap

```
ok_tr_missing %>%
  count(drugs, body_type) %>%
  ggplot(aes(drugs, body_type)) +
  geom_point(aes(size = n)) +
  theme_light()
```

Exploring missingness between numerical and categorical predictors

```
ok_tr_missing %>%
  mutate(bodytype_missing = is.na(body_type)) %>%
  ggplot(aes(bodytype_missing, age)) +
  geom_violin() +
  facet_wrap(. ~ sex) +
  theme_light()
```

Exploring missingness between two numerical predictors (borrowing `airquality` data)

```
ggplot(airquality, aes(Ozone, Solar.R)) +
  naniar::geom_miss_point() +
  theme_light()
```

When the data is large you could perform PCA on the binary missing/non-missing matrix:

```
binary_mat <- ok_tr_missing %>%
  select(-pets) %>%
  mutate(across(everything(), is.na))

prcomp(binary_mat) %>%
  pluck("rotation") %>%
  as_tibble() %>%
  select(PC1, PC2) %>%
  bind_cols(
    tibble(
      predictor = colnames(binary_mat),
      pct_missing = binary_mat %>% colSums() / nrow(ok_tr_missing)
    )
  ) %>%
  mutate(label = ifelse(PC1 < -0.3 | PC2 > 0.3, predictor, "")) %>
  ggplot(aes(PC1, PC2)) +
  geom_point(aes(size = pct_missing), color = "lightblue", alpha =
  ggrepel::geom_text_repel(aes(label = label)) +
  theme_light()
```

offspring
diet
religion
essay8
essay7
essay3
essay5
essay6
essay9
0.6
0.4
0.2
0.0
PC2
-0.4
-0.3
-0.2
-0.1
0.0
PC1
pct_missing
0.0
0.2
0.4
0.6
0.8

Definitely not in MCAR land

- Keep the data missing, drop it all or impute values?
  - Very much depends on the mechanism of missing data
  - "Drop it all" only if you're "data rich" and assume MCAR
  - Keep or impute: can your model handle missing values?
  - "It's not a bug, it's a feature!": a new category or an additional variable
  - Use resampling to test the best strategy!

- How to impute?
  - Very much depends on the mechanism of missing data
  - Mean/median value if MCAR
  - kNN imputation
  - Iterative Regression/Classification algos (EM algorithm)
  - Use resampling to test the best strategy!
- What do you do with missing values in the response variable?
  - Semi-supervised Learning
  - Active Learning

Let's choose between 2 x 3 x 3 = 18 strategies:

1. Missing categorical features
   - adding an "unknown" category
   - imputing the most common category
2. The `income` feature:
   - Dropping it (it is 80% missing!!)
   - Log transformation, imputing the mean
   - Log transformation, kNN Imputation
3. The `essay` features:
   - Dropping them
   - Adding `essay` length where a missing `essay` has length 0
   - Adding `essay` length where a missing `essay` has length 0 and adding "is_missing" feature to each `essay`

Use `step_mutate_*()` or add what you need beforehand.

```
ok_tr_missing2 <- ok_tr_missing %>%
  mutate(across(essay0:essay9,
                list(
                  "len" = ~ifelse(is.na(.x), 0, str_length(.x)),
                  "isna" = ~ifelse(is.na(.x), 1, 0)
                  ))) %>%
  rename_with(~paste("len", gsub("_len", "", .), sep = "_"),
              ends_with("_len")) %>%
  rename_with(~paste("isna", gsub("_is_na", "", .), sep = "_"),
              ends_with("_isna")) %>%
  select(-starts_with("essay")) %>%
  mutate(across(where(is.character), as.factor))
```

Defining recipes (without `prep()`!), let's start with 2 to simplify.

```
rec_miss_unk_none_none <- recipe(pets ~ ., data = ok_tr_missing2)
  update_role(income, contains("essay"), new_role = "discarded") %
  step_normalize(all_numeric(), -has_role("discarded")) %>%
  step_other(all_nominal(), -has_role("discarded"), other = "other
  step_novel(all_nominal(), -has_role("discarded"), -all_outcomes
  step_unknown(all_nominal(), -has_role("discarded"), -all_outcome
  step_dummy(all_nominal(), -all_outcomes(),
             -has_role("discarded"), one_hot = FALSE)

rec_miss_mode_none_none <- recipe(pets ~ ., data = ok_tr_missing2)
  update_role(income, contains("essay"), new_role = "discarded") %
  step_normalize(all_numeric(), -has_role("discarded")) %>%
  step_other(all_nominal(), -has_role("discarded"), other = "other
  step_novel(all_nominal(), -has_role("discarded"), -all_outcomes
  step_impute_mode(all_nominal(), -has_role("discarded"), -all_out
  step_dummy(all_nominal(), -all_outcomes(),
             -has_role("discarded"), one_hot = FALSE)
```

(pretty sure `step_rm` is a way to discard predictors altogether)

Defining resamples

```
set.seed(42)
cv_splits_missing <- vfold_cv(ok_tr_missing2, v = 10, strata = pet

cv_splits_missing
```

```
## #  10-fold cross-validation using stratification
## # A tibble: 10 x 2
##    splits             id
##    <list>             <chr>
##  1 <split [1798/200]> Fold01
##  2 <split [1798/200]> Fold02
##  3 <split [1798/200]> Fold03
##  4 <split [1798/200]> Fold04
##  5 <split [1798/200]> Fold05
##  6 <split [1798/200]> Fold06
##  7 <split [1798/200]> Fold07
##  8 <split [1798/200]> Fold08
##  9 <split [1798/200]> Fold09
## 10 <split [1800/198]> Fold10
```

Applying (two) recipes to all splits

```
cv_splits_missing$rec_miss_unk_none_none <- map(
  cv_splits_missing$splits, prepper, recipe = rec_miss_unk_none_no
cv_splits_missing$rec_miss_mode_none_none <- map(
  cv_splits_missing$splits, prepper, recipe = rec_miss_mode_none_r

map_dbl(cv_splits_missing$rec_miss_unk_none_none, ~nrow(.$term_inf
```

```
##  [1] 115 114 113 112 113 113 112 115 114 114
```

```
cv_splits_missing$rec_miss_unk_none_none[[1]]$term_info %>%
  filter(str_detect(variable, "speaks"))
```

```
## # A tibble: 4 x 4
##   variable                  type    role      source
##   <chr>                     <chr>   <chr>     <chr>
## 1 speaks_english..fluently. numeric predictor derived
## 2 speaks_other1             numeric predictor derived
## 3 speaks_new                numeric predictor derived
## 4 speaks_unknown            numeric predictor derived
```

Defining the model specification

```
mod_lr <- logistic_reg(penalty = 0.01) %>%
  set_engine("glmnet")
```

Fitting the model to all splits

```
fit_lr <- function(rec_obj)
  fit(mod_lr, pets ~ .,
      data = bake(rec_obj, NULL, all_predictors(), all_outcomes()))

cv_splits_missing$mod_lr_unk <- map(
  cv_splits_missing$rec_miss_unk_none_none, fit_lr)
cv_splits_missing$mod_lr_mode <- map(
  cv_splits_missing$rec_miss_mode_none_none, fit_lr)
```

Predicting on assessment data of each split

```
pred_lr <- function(split_obj, rec_obj, model_obj) {
  mod_data <- bake(rec_obj,
                   new_data = assessment(split_obj),
                   all_predictors(), all_outcomes())
  out <- mod_data %>% select(pets)
  out$predicted <- predict(model_obj, mod_data, type = "prob",
                           penalty = 0.0001)$.pred_cats
  out
}

cv_splits_missing$pred_unk <-
  pmap(
    lst(
      split_obj = cv_splits_missing$splits,
      rec_obj = cv_splits_missing$rec_miss_unk_none_none,
      model_obj = cv_splits_missing$mod_lr_unk
    ),
    pred_lr)
cv_splits_missing$pred_mode <- pmap(lst(
  split_obj = cv_splits_missing$splits,
  rec_obj = cv_splits_missing$rec_miss_mode_none_none,
  model_obj = cv_splits_missing$mod_lr_mode
),
pred_lr)
```

Measuring the AUC of each split

```
cv_splits_missing$step_unknown <- map_dfr(
  cv_splits_missing$pred_unk, roc_auc, pets, predicted)$.estimate
cv_splits_missing$step_mode <- map_dfr(
  cv_splits_missing$pred_mode, roc_auc, pets, predicted)$.estimate
```

Comparing AUCs with a plot

```
cv_splits_missing %>%
  select(id, step_unknown, step_mode) %>%
  pivot_longer(cols = c(step_unknown, step_mode),
               names_to = "recipe", values_to = "auc") %>%
  ggplot(aes(recipe, auc, group = id, color = id)) +
  geom_line() +
  geom_point() +
  guides(color = "none") + theme_light()
```

Comparing AUCs with a statistical test (paired T test)

```
t.test(cv_splits_missing$step_unknown,
       cv_splits_missing$step_mode, paired = TRUE)
```

```
##
##     Paired t-test
##
## data:  cv_splits_missing$step_unknown and cv_splits_missing$step_mode
## t = -1.6396, df = 9, p-value = 0.1355
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.012471616  0.001989882
## sample estimates:
## mean of the differences
##            -0.005240867
```

Let's move on to 2 x 2 recipes, adding 2 more recipes for treating the `income` variable with mean imputation, with `step_unknown()` and `step_impute_mode()`.

```
rec_miss_unk_mean_none <- recipe(pets ~ ., data = ok_tr_missing2)
  update_role(contains("essay"), new_role = "discarded") %>%
  step_log(income, offset = 1) %>%
  step_impute_mean(income) %>%
  step_normalize(all_numeric(), -has_role("discarded")) %>%
  step_other(all_nominal(), -has_role("discarded"), other = "other
  step_novel(all_nominal(), -has_role("discarded"), -all_outcomes
  step_unknown(all_nominal(), -has_role("discarded"), -all_outcome
  step_dummy(all_nominal(), -all_outcomes(),
             -has_role("discarded"), one_hot = FALSE)

rec_miss_mode_mean_none <- recipe(pets ~ ., data = ok_tr_missing2)
  update_role(contains("essay"), new_role = "discarded") %>%
  step_log(income, offset = 1) %>%
  step_impute_mean(income) %>%
  step_normalize(all_numeric(), -has_role("discarded")) %>%
  step_other(all_nominal(), -has_role("discarded"), other = "other
  step_novel(all_nominal(), -has_role("discarded"), -all_outcomes
  step_impute_mode(all_nominal(), -has_role("discarded"), -all_out
  step_dummy(all_nominal(), -all_outcomes(),
             -has_role("discarded"), one_hot = FALSE)
```

Our approach with 2 recipes does not scale, and we don't need to keep all data just final AUC for each recipe. We are writing a `train_recipe()` function!

```
lst_recs <- list("unknown_none" = rec_miss_unk_none_none,
                 "mode_none" = rec_miss_mode_none_none,
                 "unknown_mean" = rec_miss_unk_mean_none,
                 "mode_mean" = rec_miss_mode_mean_none)

train_recipe <- function(rec, rec_name, splits) {
  splits_prepped <- map(splits, prepper, recipe = rec)
  splits_fit <- map(splits_prepped, fit_lr)
  splits_pred <- pmap(
    lst(
      split_obj = splits,
      rec_obj = splits_prepped,
      model_obj = splits_fit
    ),
    pred_lr
  )
  res <- map_dfr(splits_pred, roc_auc, pets, predicted)$.estimate
  names(res) <- rec_name
  res
}
```

Training multiple recipes, saving the AUC for each recipe, each split.

```
cv_splits_missing <- cv_splits_missing %>%
  bind_cols(
    map2_dfc(lst_recs, names(lst_recs), train_recipe,
             splits = cv_splits_missing$splits)
  )
```

Processing the results into a nice table:

```
cv_res_missing <- cv_splits_missing %>%
    pivot_longer(cols = names(lst_recs),
                 names_to = "recipe", values_to = "AUC") %>%
  select(id, recipe, AUC) %>%
  separate(recipe, c("categorical", "income"))

cv_res_missing
```

```
## # A tibble: 40 x 4
##    id     categorical income   AUC
##    <chr>  <chr>       <chr>  <dbl>
##  1 Fold01 unknown     none   0.691
##  2 Fold01 mode        none   0.694
##  3 Fold01 unknown     mean   0.695
##  4 Fold01 mode        mean   0.702
##  5 Fold02 unknown     none   0.640
##  6 Fold02 mode        none   0.643
```

Comparing AUCs with a plot

```
cv_res_missing %>%
  ggplot(aes(income, AUC, group = id, color = id)) +
  geom_line() + geom_line(data = cv_res_missing_meds, color = "bla
  geom_point() + geom_point(data = cv_res_missing_meds, color = "b
    guides(color = "none") + facet_wrap(. ~ categorical) +
    theme_light()
```

Comparing AUCs with a statistical test (2x2 Repeated measures ANOVA)

```
aov_missing <- aov(
  AUC ~ factor(categorical) * factor(income) + Error(factor(id)),
  data = cv_res_missing)
print(summary(aov_missing))
```

```
##
## Error: factor(id)
##           Df Sum Sq  Mean Sq F value Pr(>F)
## Residuals  9 0.0675 0.007501
##
## Error: Within
##                                  Df    Sum Sq   Mean Sq F value Pr(>F)
## factor(categorical)               1 0.0002915 2.915e-04   6.578 0.0162
## factor(income)                    1 0.0000107 1.070e-05   0.241 0.6271
## factor(categorical):factor(income)  1 0.0000002 2.500e-07   0.006 0.9407
## Residuals                        27 0.0011964 4.431e-05
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
with(cv_res_missing, interaction.plot(categorical, income, AUC))
```

Ok, now let's go back to our original idea of 18 recipes... see code in slides Rmd file.

```
lst_recs <- list("unknown_none_none" = rec_miss_unk_none_none,
                 "mode_none_none" = rec_miss_mode_none_none,
                 "unknown_mean_none" = rec_miss_unk_mean_none,
                 "mode_mean_none" = rec_miss_mode_mean_none,
                 "unknown_knn_none" = rec_miss_unk_knn_none,
                 "mode_knn_none" = rec_miss_mode_knn_none,
                 "unknown_none_len" = rec_miss_unk_none_len,
                 "mode_none_len" = rec_miss_mode_none_len,
                 "unknown_mean_len" = rec_miss_unk_mean_len,
                 "mode_mean_len" = rec_miss_mode_mean_len,
                 "unknown_knn_len" = rec_miss_unk_knn_len,
                 "mode_knn_len" = rec_miss_mode_knn_len,
                 "unknown_none_isna" = rec_miss_unk_none_isna,
                 "mode_none_isna" = rec_miss_mode_none_isna,
                 "unknown_mean_isna" = rec_miss_unk_mean_isna,
                 "mode_mean_isna" = rec_miss_mode_mean_isna,
                 "unknown_knn_isna" = rec_miss_unk_knn_isna,
                 "mode_knn_isna" = rec_miss_mode_knn_isna
                 )
```

```r
set.seed(42)
cv_splits_missing <- vfold_cv(ok_tr_missing2, v = 10, strata = pet
```

```r
cv_splits_missing <- cv_splits_missing %>%
  bind_cols(
    map2_dfc(lst_recs, names(lst_recs), train_recipe,
             splits = cv_splits_missing$splits)
  )
```

```r
cv_res_missing <- cv_splits_missing %>%
    pivot_longer(cols = names(lst_recs),
                 names_to = "recipe", values_to = "AUC") %>%
  select(id, recipe, AUC) %>%
  separate(recipe, c("categorical", "income", "essay"))
```

Again, see the plot (how would you plot a 2x3x3 experiment results?).

```
cv_res_missing %>%
  group_by(id, income, essay) %>%
  summarise(AUC = mean(AUC)) %>%
  ggplot(aes(income, AUC, group = id, color = id)) +
    geom_line() + geom_line(data = cv_res_missing_meds, color = "k
  geom_point() + geom_point(data = cv_res_missing_meds, color = "k
    guides(color = "none") + facet_wrap(. ~ essay) +
    theme_light()
```

Repeated measures ANOVA

```
aov_missing <- aov(
  AUC ~ factor(categorical) * factor(income) + factor(essay) + Er
  data = cv_res_missing)
summary(aov_missing)
```

```
##
## Error: factor(id)
##           Df Sum Sq Mean Sq F value Pr(>F)
## Residuals  9 0.1595 0.01772
##
## Error: Within
##                                   Df   Sum Sq   Mean Sq F value  Pr(>F
## factor(categorical)                1 0.000191 0.0001911   1.975 0.1617
## factor(income)                     2 0.000483 0.0002415   2.497 0.0854
## factor(essay)                      2 0.001101 0.0005503   5.689 0.0040
## factor(categorical):factor(income)  2 0.000007 0.0000036   0.038 0.9630
## Residuals                        163 0.015766 0.0000967
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Or you can just look at the mean numbers...

```
cv_res_missing %>%
  group_by(categorical, income, essay) %>%
  summarise(AUC = mean(AUC)) %>%
  arrange(-AUC) %>%
  head(10)
```

```
## # A tibble: 10 x 4
## # Groups:   categorical, income [6]
##    categorical income essay   AUC
##    <chr>       <chr>  <chr> <dbl>
##  1 mode        mean   len   0.719
##  2 mode        knn    len   0.719
##  3 unknown     knn    len   0.718
##  4 unknown     mean   len   0.718
##  5 mode        mean   none  0.716
##  6 mode        none   len   0.716
##  7 mode        mean   isna  0.715
##  8 unknown     mean   none  0.715
##  9 mode        knn    none  0.715
## 10 unknown     none   len   0.715
```

So what would you do?

Class Imbalance

Applications

Of Data Science

Class Imbalance

(We already gave a decent review in last class)

We'll choose between 3 strategies:

- Upsample
- Downsample
- SMOTE and Downsample

Remember how we got here.

```
handle_essays <- function(ok_df) {
  ok_df %>%
    mutate(across(essay0:essay9,
                  list("len" = ~ifelse(is.na(.x), 0, str_length(.x
    rename_with(~paste("len", gsub("_len", "", .), sep = "_"),
                ends_with("_len"))
}
ok_tr_imbalance2 <- handle_essays(ok_tr_imbalance) %>%
  select(-starts_with("essay")) %>%
  mutate(across(where(is.character), as.factor))
```

```r
library(themis)

rec_imb_upsample <- recipe(pets ~ ., data = ok_tr_imbalance2) %>%
  step_log(income, starts_with("len"), offset = 1) %>%
  step_impute_mean(income) %>%
  step_normalize(all_numeric(), -has_role("discarded")) %>%
  step_other(all_nominal(), -has_role("discarded"), other = "other
  step_novel(all_nominal(), -has_role("discarded"), -all_outcomes
  step_impute_mode(all_nominal(), -has_role("discarded"), -all_out
  step_dummy(all_nominal(), -all_outcomes(),
             -has_role("discarded"), one_hot = FALSE) %>%
  step_upsample(pets, over_ratio = 1, seed = 42)

rec_imb_downsample <- recipe(pets ~ ., data = ok_tr_imbalance2) %>
  step_log(income, starts_with("len"), offset = 1) %>%
  step_impute_mean(income) %>%
  step_normalize(all_numeric(), -has_role("discarded")) %>%
  step_other(all_nominal(), -has_role("discarded"), other = "other
  step_novel(all_nominal(), -has_role("discarded"), -all_outcomes
  step_impute_mode(all_nominal(), -has_role("discarded"), -all_out
  step_dummy(all_nominal(), -all_outcomes(),
             -has_role("discarded"), one_hot = FALSE) %>%
  step_downsample(pets, under_ratio = 1, seed = 42)
```

```
rec_imb_smote <- recipe(pets ~ ., data = ok_tr_imbalance2) %>%
  step_log(income, starts_with("len"), offset = 1) %>%
  step_impute_mean(income) %>%
  step_normalize(all_numeric(), -has_role("discarded")) %>%
  step_other(all_nominal(), -has_role("discarded"), other = "other
  step_novel(all_nominal(), -has_role("discarded"), -all_outcomes
  step_impute_mode(all_nominal(), -has_role("discarded"), -all_out
  step_dummy(all_nominal(), -all_outcomes(),
             -has_role("discarded"), one_hot = FALSE) %>%
  step_downsample(pets, under_ratio = 1.5) %>%
  step_smote(pets, over_ratio = 1, seed = 42)
```

CV splits

```
set.seed(42)
cv_splits_imbalance <- vfold_cv(ok_tr_imbalance2, v = 10, strata =

lst_recs <- list("upsample" = rec_imb_upsample,
                 "downsample" = rec_imb_downsample,
                 "smote" = rec_imb_smote)
```

```
cv_splits_imbalance <- cv_splits_imbalance %>%
  bind_cols(
    map2_dfc(lst_recs, names(lst_recs), train_recipe,
             splits = cv_splits_imbalance$splits)
  )
```

```r
cv_res_imbalance <- cv_splits_imbalance %>%
    pivot_longer(cols = names(lst_recs),
                 names_to = "recipe", values_to = "AUC") %>%
  select(id, recipe, AUC)
```

Repeated measures ANOVA:

```r
aov_imbalance <- aov(
  AUC ~ factor(recipe) + Error(factor(id)),
  data = cv_res_imbalance)
print(summary(aov_imbalance))
```

```
##
## Error: factor(id)
##           Df  Sum Sq  Mean Sq F value Pr(>F)
## Residuals  9 0.07671 0.008523
##
## Error: Within
##                Df   Sum Sq   Mean Sq F value Pr(>F)
## factor(recipe)  2 0.001561 0.0007803   4.034 0.0357 *
## Residuals      18 0.003481 0.0001934
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Plotting...

```
cv_res_imbalance %>%
  ggplot(aes(recipe, AUC, group = id, color = id)) +
    geom_line() + geom_line(data = cv_res_imbalance_meds, color =
  geom_point() + geom_point(data = cv_res_imbalance_meds, color =
    guides(color = "none") +
    theme_light()
```

```r
print(cv_res_imbalance %>%
        group_by(recipe) %>%
        summarise(mean_auc = mean(AUC), .groups="drop"))
```

```
## # A tibble: 3 x 2
##   recipe     mean_auc
##   <chr>         <dbl>
## 1 downsample    0.721
## 2 smote         0.730
## 3 upsample      0.739
```

So what would you do?

Feat. Engineering and Selection

Applications

Of Data Science

If anything, remember:

1. Don't be lazy.
2. Be creative.
3. Use resampling!

Let's divide to:

1. Categorical Features
2. Continuous Features
3. Interactions

Categorical Variables

Applications

Of Data Science

Making dummy variables

(In the case your model can accept categorical levels, skip to the next section - Dealing with categorical variables with many levels)

- One hot encoding: k dummy vars for k levels
- Non one hot encoding: $k - 1$ dummy vars for k levels
- Different Contrasts schemes (e.g. for ordered categorical variables ("low", "medium", "high"))
- Feature Hashing

```r
set.seed(42)
df <- tibble(
  x1 = factor(c(rep("a", 4), rep("b", 4))),
  y = runif(8)
)

one_hot <- recipe(y ~ x1, data = df) %>%
  step_dummy(all_nominal(), one_hot = TRUE) %>%
  prep(df) %>%
  bake(new_data = NULL)

non_one_hot <- recipe(y ~ x1, data = df) %>%
  step_dummy(all_nominal(), one_hot = FALSE) %>%
  prep(df) %>%
  bake(new_data = NULL)
```

```r
head(one_hot)
```

```
## # A tibble: 6 x 3
##       y  x1_a  x1_b
##   <dbl> <dbl> <dbl>
## 1 0.915     1     0
## 2 0.937     1     0
## 3 0.286     1     0
## 4 0.830     1     0
## 5 0.642     0     1
## 6 0.519     0     1
```

```r
head(non_one_hot)
```

```
## # A tibble: 6 x 2
##       y  x1_b
##   <dbl> <dbl>
## 1 0.915     0
## 2 0.937     0
## 3 0.286     0
## 4 0.830     0
## 5 0.642     1
## 6 0.519     1
```

When dealing with linear regression, you should be very careful what you input into the `lm` engine of your choice and how you interpret results!

```
summary(lm(y ~ ., data = non_one_hot))
```

```
##
## Call:
## lm(formula = y ~ ., data = non_one_hot)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.45598 -0.08504  0.11103  0.17826  0.22856
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.7421     0.1434   5.177  0.00206 **
## x1_b         -0.2341     0.2027  -1.155  0.29212
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2867 on 6 degrees of freedom
## Multiple R-squared:  0.1818,    Adjusted R-squared:  0.04546
## F-statistic: 1.333 on 1 and 6 DF,  p-value: 0.2921
```

```
print(mean(df$y) + (mean(df$y[df$x1 == "a"]) - mean(df$y)))
```

This is like:

```
summary(lm(y ~ ., data = df))
```

```
##
## Call:
## lm(formula = y ~ ., data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.45598 -0.08504  0.11103  0.17826  0.22856
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.7421     0.1434   5.177  0.00206 **
## x1b          -0.2341     0.2027  -1.155  0.29212
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2867 on 6 degrees of freedom
## Multiple R-squared:  0.1818,    Adjusted R-squared:  0.04546
## F-statistic: 1.333 on 1 and 6 DF,  p-value: 0.2921
```

```r
summary(lm(y ~ ., data = one_hot))
```

```
##
## Call:
## lm(formula = y ~ ., data = one_hot)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.45598 -0.08504  0.11103  0.17826  0.22856
##
## Coefficients: (1 not defined because of singularities)
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.5080     0.1434   3.544   0.0122 *
## x1_a          0.2341     0.2027   1.155   0.2921
## x1_b              NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2867 on 6 degrees of freedom
## Multiple R-squared:  0.1818,    Adjusted R-squared:  0.04546
## F-statistic: 1.333 on 1 and 6 DF,  p-value: 0.2921
```

```r
summary(lm(y ~ . - 1, data = one_hot))
```

```
##
## Call:
## lm(formula = y ~ . - 1, data = one_hot)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.45598 -0.08504  0.11103  0.17826  0.22856
##
## Coefficients:
##      Estimate Std. Error t value Pr(>|t|)
## x1_a   0.7421     0.1434   5.177  0.00206 **
## x1_b   0.5080     0.1434   3.544  0.01216 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2867 on 6 degrees of freedom
## Multiple R-squared:  0.8677,    Adjusted R-squared:  0.8236
## F-statistic: 19.68 on 2 and 6 DF,  p-value: 0.002314
```

This is like:

```
summary(lm(y ~ . - 1, data = df))
```

```
##
## Call:
## lm(formula = y ~ . - 1, data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.45598 -0.08504  0.11103  0.17826  0.22856
##
## Coefficients:
##     Estimate Std. Error t value Pr(>|t|)
## x1a   0.7421     0.1434   5.177  0.00206 **
## x1b   0.5080     0.1434   3.544  0.01216 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2867 on 6 degrees of freedom
## Multiple R-squared:  0.8677,    Adjusted R-squared:  0.8236
## F-statistic: 19.68 on 2 and 6 DF,  p-value: 0.002314
```

Plot thickens (check your `options("contrasts")`!):

```
summary(lm(y ~ ., data = df, contrasts = list(x1 = "contr.sum")))
```

```
##
## Call:
## lm(formula = y ~ ., data = df, contrasts = list(x1 = "contr.sum"))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.45598 -0.08504  0.11103  0.17826  0.22856
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.6251     0.1014   6.167 0.000835 ***
## x11           0.1170     0.1014   1.155 0.292118
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2867 on 6 degrees of freedom
## Multiple R-squared:  0.1818,    Adjusted R-squared:  0.04546
## F-statistic: 1.333 on 1 and 6 DF,  p-value: 0.2921
```

Two variables...

```
set.seed(42)
df <- tibble(
  x1 = factor(c(rep("a", 6), rep("b", 6))),
  x2 = factor(rep(c(rep("c", 2), rep("d", 2), rep("e", 2)), 2)),
  y = runif(12)
)

one_hot <- recipe(y ~ x1 + x2, data = df) %>%
  step_dummy(all_nominal(), one_hot = TRUE) %>%
  prep(df) %>%
  bake(new_data = NULL)

non_one_hot <- recipe(y ~ x1 + x2, data = df) %>%
  step_dummy(all_nominal(), one_hot = FALSE) %>%
  prep(df) %>%
  bake(new_data = NULL)
```

```r
head(one_hot)
```

```
## # A tibble: 6 x 6
##       y  x1_a  x1_b  x2_c  x2_d  x2_e
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 0.915     1     0     1     0     0
## 2 0.937     1     0     1     0     0
## 3 0.286     1     0     0     1     0
## 4 0.830     1     0     0     1     0
## 5 0.642     1     0     0     0     1
## 6 0.519     1     0     0     0     1
```

```r
head(non_one_hot)
```

```
## # A tibble: 6 x 4
##       y  x1_b  x2_d  x2_e
##   <dbl> <dbl> <dbl> <dbl>
## 1 0.915     0     0     0
## 2 0.937     0     0     0
## 3 0.286     0     1     0
## 4 0.830     0     1     0
## 5 0.642     0     0     1
## 6 0.519     0     0     1
```

```r
summary(lm(y ~ ., data = non_one_hot))
```

```
##
## Call:
## lm(formula = y ~ ., data = non_one_hot)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.48619 -0.08138  0.10650  0.15667  0.19636
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.74071    0.15610   4.745  0.00145 **
## x1_b        -0.11986    0.15610  -0.768  0.46465
## x2_d        -0.06112    0.19118  -0.320  0.75738
## x2_e        -0.09636    0.19118  -0.504  0.62783
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2704 on 8 degrees of freedom
## Multiple R-squared:  0.09602,    Adjusted R-squared:  -0.243
## F-statistic: 0.2832 on 3 and 8 DF,  p-value: 0.8362
```

Same as:

```
summary(lm(y ~ ., data = df))
```

```
##
## Call:
## lm(formula = y ~ ., data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.48619 -0.08138  0.10650  0.15667  0.19636
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.74071    0.15610   4.745  0.00145 **
## x1b         -0.11986    0.15610  -0.768  0.46465
## x2d         -0.06112    0.19118  -0.320  0.75738
## x2e         -0.09636    0.19118  -0.504  0.62783
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2704 on 8 degrees of freedom
## Multiple R-squared:  0.09602,    Adjusted R-squared:  -0.243
## F-statistic: 0.2832 on 3 and 8 DF,  p-value: 0.8362
```

```r
print(mean(df$y) +
        (mean(df$y[df$x1 == "a"]) - mean(df$y)) +
        (mean(df$y[df$x2 == "c"]) - mean(df$y)))
```

```
## [1] 0.7407128
```

```r
print(mean(df$y[df$x1 == "a"]) + mean(df$y[df$x2 == "c"]) - mean(d
```

```
## [1] 0.7407128
```

```r
print(mean(df$y[df$x1 == "b"]) - mean(df$y[df$x1 == "a"]))
```

```
## [1] -0.1198573
```

```r
print(mean(df$y[df$x2 == "d"]) - mean(df$y[df$x2 == "c"]))
```

```
## [1] -0.06112303
```

```r
print(mean(df$y[df$x2 == "e"]) - mean(df$y[df$x2 == "c"]))
```

```
## [1] -0.09636022
```

```r
summary(lm(y ~ ., data = one_hot))
```

```
##
## Call:
## lm(formula = y ~ ., data = one_hot)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.48619 -0.08138  0.10650  0.15667  0.19636
##
## Coefficients: (2 not defined because of singularities)
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.52450    0.15610   3.360  0.00993 **
## x1_a         0.11986    0.15610   0.768  0.46465
## x1_b              NA         NA      NA       NA
## x2_c         0.09636    0.19118   0.504  0.62783
## x2_d         0.03524    0.19118   0.184  0.85836
## x2_e              NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2704 on 8 degrees of freedom
## Multiple R-squared:  0.09602,    Adjusted R-squared:  -0.243
## F-statistic: 0.2832 on 3 and 8 DF,  p-value: 0.8362
```

```r
print(mean(df$y[df$x1 == "b"]) + mean(df$y[df$x2 == "e"]) - mean(c
```

```
## [1] 0.5244952
```

```r
print(mean(df$y[df$x1 == "a"]) - mean(df$y[df$x1 == "b"]))
```

```
## [1] 0.1198573
```

```r
print(mean(df$y[df$x2 == "c"]) - mean(df$y[df$x2 == "e"]))
```

```
## [1] 0.09636022
```

```r
print(mean(df$y[df$x2 == "d"]) - mean(df$y[df$x2 == "e"]))
```

```
## [1] 0.03523718
```

```r
summary(lm(y ~ . - 1, data = one_hot))
```

```
##
## Call:
## lm(formula = y ~ . - 1, data = one_hot)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.48619 -0.08138  0.10650  0.15667  0.19636
##
## Coefficients: (1 not defined because of singularities)
##      Estimate Std. Error t value Pr(>|t|)
## x1_a  0.64435    0.15610   4.128  0.00331 **
## x1_b  0.52450    0.15610   3.360  0.00993 **
## x2_c  0.09636    0.19118   0.504  0.62783
## x2_d  0.03524    0.19118   0.184  0.85836
## x2_e       NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2704 on 8 degrees of freedom
## Multiple R-squared:  0.8914,    Adjusted R-squared:  0.8371
## F-statistic: 16.41 on 4 and 8 DF,  p-value: 0.0006356
```

```r
print(mean(df$y[df$x1 == "a"]) + mean(df$y[df$x2 == "e"]) - mean(c
```

```
## [1] 0.6443525
```

```r
print(mean(df$y[df$x1 == "b"]) + mean(df$y[df$x2 == "e"]) - mean(c
```

```
## [1] 0.5244952
```

```r
print(mean(df$y[df$x2 == "c"]) - mean(df$y[df$x2 == "e"]))
```

```
## [1] 0.09636022
```

```r
print(mean(df$y[df$x2 == "d"]) - mean(df$y[df$x2 == "e"]))
```

```
## [1] 0.03523718
```

Categorical variables with many levels

```
ok_tr_eng %>% count(location)
```

```
## # A tibble: 67 x 2
##    location                          n
##    <chr>                         <int>
##  1 alameda, california              54
##  2 albany, california               10
##  3 ashland, california               1
##  4 atherton, california              7
##  5 belmont, california              11
##  6 belvedere tiburon, california     4
##  7 benicia, california              12
##  8 berkeley, california            188
##  9 brisbane, california              2
## 10 burlingame, california           13
## # ... with 57 more rows
```

Many dangers lurk

- OVERFITTING!
- Unseen levels
- Zero variance variables
- Size of p

We'll talk about

- The "Other" and "New" Strategy
- Clustering
- Feature Hashing
- Supervised Numerical Encoding
- Dimensionality Reduction (Embeddings)
- Text Features
- Won't talk but God knows this is on my mind, e.g. Simchoni and Rosset (2021)

The "Other" and "New" Strategy

- "Other": in many scenarios in which a categorical feature has many levels, some of these are so scarce it is doubtful if they have any predictive power - consider grouping them into one "Other" level
- "New": therefore there might be many levels not seen during training, only during testing - consider adding a (zero variance) "New" level

What % threshold should you use for "Other"?

- `step_other()` default is 5% (i.e. any level with less than 5% occurrence is grouped into one super-level "Other")
- or tune it!

```
recipe(pets ~ ., data = ok_tr_eng %>% select(pets, location)) %>%
  step_other(all_nominal(), threshold = 0.05) %>%
  step_novel(all_nominal()) %>%
  prep(ok_tr_eng) %>%
  bake(new_data = NULL) %>%
  count(location)
```

```
## # A tibble: 4 x 2
##   location                      n
##   <fct>                     <int>
## 1 berkeley, california        188
## 2 oakland, california         344
## 3 san francisco, california  1601
## 4 other                       872
```

You can also insert some creativeness (or plain common sense) in your "Other" strategy:

```
education2 = case_when(
           education == "graduated from high school" ~ "high_schoo
           education == "graduated from two-year college" ~ "colle
           education == "graduated from college/university" ~ "deg
           education == "graduated from masters program" ~ "degree
           education == "graduated from ph.d program" ~ "degree3",
           education == "working on two-year college" ~ "student0"
           education == "working on college/university" ~ "student
           education == "working on masters program" ~ "student2",
           education == "working on ph.d program" ~ "student3",
           is.na(education) ~ NA_character_,
           TRUE ~ "other"
)
```

Clustering

- Manual clustering, using expert knowledge and common sense
- Unsupervised clustering, where it makes sense (e.g. Topic Modeling with LDA on text data)

> 💡 How would you cluster `location`?

The `sign` feature is also a great example:

```
ok_tr_eng %>% count(sign)
```

```
## # A tibble: 47 x 2
##    sign                                          n
##    <chr>                                     <int>
##  1 aquarius                                     44
##  2 aquarius and it&rsquo;s fun to think about   71
##  3 aquarius but it doesn&rsquo;t matter         65
##  4 aries                                        51
##  5 aries and it matters a lot                    1
##  6 aries and it&rsquo;s fun to think about      74
##  7 aries but it doesn&rsquo;t matter            71
```

```r
signs <- c("leo", "libra", "cancer", "virgo", "scorpio", "gemini",
           "taurus", "aries", "pisces", "aquarius", "sagittarius",

recipe(pets ~ ., data = ok_tr_eng %>% select(pets, sign)) %>%
  step_mutate(sign = str_extract(sign, str_c(signs, collapse = "|'
  step_unknown(sign) %>%
  prep(ok_tr_eng) %>%
  bake(new_data = NULL) %>%
  count(sign)
```

```
## # A tibble: 13 x 2
##    sign            n
##    <fct>       <int>
##  1 aries         197
##  2 virgo         231
##  3 leo           224
##  4 taurus        248
##  5 cancer        205
##  6 scorpio       209
##  7 libra         230
##  8 aquarius      180
##  9 gemini        225
## 10 pisces        223
## 11 capricorn     188
## 12 sagittarius   208
## 13 unknown       437
```

Feature Hashing

Suppose I told you "God, just group the locations by the number of times the letter 'l' appears in them!"

```
recipe(pets ~ ., data = ok_tr_eng %>% select(pets, location)) %>%
  step_mutate(location = factor(str_count(location, "l"))) %>%
  prep(ok_tr_eng) %>%
  bake(new_data = NULL) %>%
  count(location)
```

```
## # A tibble: 5 x 2
##   location     n
##   <fct>    <int>
## 1 1         1943
## 2 2          878
## 3 3          141
## 4 4           22
## 5 5           21
```

```r
recipe(pets ~ ., data = ok_tr_eng %>% select(pets, location)) %>%
  step_mutate(locationL = factor(str_count(location, "l"))) %>%
  step_dummy(locationL) %>%
  prep(ok_tr_eng) %>%
  bake(new_data = NULL) %>%
  select(starts_with("location"))
```

```
## # A tibble: 3,005 x 5
##    location                  locationL_X2 locationL_X3 locationL_X4 loca
##    <fct>                            <dbl>        <dbl>        <dbl>
##  1 palo alto, california                0            1            0
##  2 san leandro, california              1            0            0
##  3 hayward, california                  0            0            0
##  4 vallejo, california                  0            1            0
##  5 san francisco, california            0            0            0
##  6 oakland, california                  1            0            0
##  7 berkeley, california                 1            0            0
##  8 alameda, california                  1            0            0
##  9 berkeley, california                 1            0            0
## 10 san francisco, california            0            0            0
## # ... with 2,995 more rows
```

It's not a good idea because

- Differenet `location`s map to the same bin ("collision"), masking possible effect
- The distribution isn't even

But there are advantages too!

- I *did* reduce the dimensionality from 70 to 6 dummy variables
- Fast
- Generalizes well: any new `location` can be mapped this way
- Scales: I can do this for millions of `location`s, no dictionary needed to store in memory

- Feature Hashing (or "hashing trick") does something similar more intelligently
- It lets you choose the hashing algo and no. of features you'd like (the more - the less chance of collision)

```
library(textrecipes)

recipe(pets ~ ., data = ok_tr_eng %>% select(pets, location)) %>%
  step_tokenize(location) %>%
  step_texthash(location, signed = FALSE, num_terms = 2^4) %>%
  prep(ok_tr_eng) %>%
  bake(new_data = NULL)
```

```
## # A tibble: 3,005 x 17
##    location_hash01 location_hash02 location_hash03 location_hash04
##              <dbl>           <dbl>           <dbl>           <dbl>
##  1               0               0               1               0
##  2               0               0               1               0
##  3               0               0               1               0
##  4               0               0               1               0
##  5               0               0               1               0
##  6               0               1               1               0
##  7               0               0               1               0
##  8               0               0               1               0
##  9               0               0               1               0
## 10               0               0               1               0
```

- Particularly useful for fast and furious feature engineering of text (see later)
- Beware of columns with zero variance, consider binning them to a "New" category
- Powerful methods like `xgboost` can handle 2^20+ no. of terms easily

> 💡 Why would this work in spite of collisions?
>
> What is still a major disadvantage? Hint: 🎁.

The Washington Post
Democracy Dies in Darkness

Education

‘Creative ... motivating’ and fired

Sarah Wysocki was out of work for only a few days after she was fired by DCPS last year. She is now teaching at Hybla Valley Elementary School in Fairfax County. (Jahi Chikwendiu/The Washington Post)

By **Bill Turque**
March 6, 2012

[Bill Turque, 2012, Washington Post]

Supervised Numerical Encoding

Here's another crazy idea for you:

- For regression, replace a category with y's mean in this category
- For classification, replace a category with the log odds of y in this category

Thus, changing a categorical variable with many levels into a single numerical variable (or a few for a few classes).

```
library(embed)

recipe(pets ~ ., data = ok_tr_eng %>% select(pets, location)) %>%
  step_mutate(location2 = location) %>%
  step_lencode_glm(location, outcome = vars(pets)) %>%
  prep(ok_tr_eng) %>%
  bake(new_data = NULL)
```

```
## # A tibble: 3,005 x 3
##    location pets  location2
##       <dbl> <fct> <fct>
##  1    -1.44 dogs  palo alto, california
##  2    -1.13 dogs  san leandro, california
##  3    -1.70 cats  hayward, california
##  4    -2.20 dogs  vallejo, california
##  5    -1.82 dogs  san francisco, california
##  6    -1.38 dogs  oakland, california
##  7    -1.22 dogs  berkeley, california
##  8    -1.25 dogs  alameda, california
##  9    -1.22 cats  berkeley, california
## 10    -1.82 cats  san francisco, california
## # ... with 2,995 more rows
```

Make sure you get what just happened

```
mv <- ok_tr_eng$location == "mountain view, california"
p <- mean(ok_tr_eng$pets[mv] == "cats")
log(p / (1 - p))
```

```
## [1] -1.609438
```

But...

- What if a specific location has just 1-2 samples in training set?
- "self-fulfilling prophecy"

The first issue can be somewhat alleviated using a Bayesian approach, *shrinking* somewhat coefficients towards their prior when n is small:

```
recipe(pets ~ ., data = ok_tr_eng %>% select(pets, location)) %>%
  step_mutate(location2 = location) %>%
  step_lencode_bayes(location, outcome = vars(pets)) %>%
  prep(ok_tr_eng %>% head(100)) %>%
  bake(new_data = NULL)
```

```
## # A tibble: 100 x 3
##    location pets  location2
##       <dbl> <fct> <fct>
##  1    -1.45 dogs  palo alto, california
##  2    -1.41 dogs  san leandro, california
##  3    -1.28 cats  hayward, california
##  4    -1.46 dogs  vallejo, california
##  5    -1.54 dogs  san francisco, california
##  6    -1.36 dogs  oakland, california
##  7    -1.46 dogs  berkeley, california
```

Dimensionality Reduction (Embeddings)

See our `word2vec` detour when we learned about [Networks Community Detection](#).

If you have `tensorflow` and `keras` installed, you could train your own embeddings to turn each `location` into a `num_terms` long vector of features:

```
recipe(pets ~ ., data = ok_tr_eng %>% select(pets, location)) %>%
  step_mutate(location2 = location) %>%
  step_embed(location, outcome = vars(pets), num_terms = 10,
             options = embed_control(epochs = 10)) %>%
  prep(ok_tr_eng) %>%
  bake(new_data = NULL)
```

Alternatively you could use pre-trained embeddings for text variables such as `essay`s using `step_word_embedding()` from the `textrecipes` package (where `my_embeddings` are your pre-trained embeddings:

```
recipe(pets ~ ., data = ok_tr_eng %>% select(pets, location)) %>%
  step_tokenize(location) %>%
  step_word_embeddings(location, embeddings = my_embeddings) %>%
  prep(ok_tr_eng) %>%
  bake(new_data = NULL)
```

Or, perform PCA on the dummy variables matrix, keeping only a few of the first PCs as features:

```
recipe(pets ~ ., data = ok_tr_eng %>% select(pets, location)) %>%
  step_mutate(old_location = location) %>%
  step_dummy(location) %>%
  step_normalize(all_numeric()) %>%
  step_pca(starts_with("location"), num_comp = 10) %>%
  prep(ok_tr_eng) %>%
  bake(new_data = NULL)
```

```
## # A tibble: 3,005 x 12
##    pets  old_location   PC01    PC02    PC03     PC04     PC05     PC06
##    <fct> <fct>         <dbl>   <dbl>   <dbl>    <dbl>    <dbl>    <dbl>
##  1 dogs  palo alto, ~ -1.26  -0.790  -1.26    1.36     6.94    -2.97
##  2 dogs  san leandro~ -1.26  -0.784  -1.24    1.20    -3.98    -6.62
##  3 cats  hayward, ca~ -1.25  -0.765  -1.18    0.889   -0.697    2.47
##  4 dogs  vallejo, ca~ -1.25  -0.740  -1.09    0.640   -0.312    0.809
##  5 dogs  san francis~  1.16   0.0656  0.0427 -0.00625  0.00154 -0.00328
##  6 dogs  oakland, ca~ -1.60   2.51    0.606  -0.0647   0.0153  -0.0322
##  7 dogs  berkeley, c~ -1.39  -1.93    3.29   -0.153    0.0340  -0.0708
##  8 dogs  alameda, ca~ -0.437 -0.0618 -0.0441  0.00686 -0.00171  0.00364
##  9 cats  berkeley, c~ -1.39  -1.93    3.29   -0.153    0.0340  -0.0708
## 10 cats  san francis~  1.16   0.0656  0.0427 -0.00625  0.00154 -0.00328
## # ... with 2,995 more rows, and 3 more variables: PC08 <dbl>, PC09 <dbl>
## #   PC10 <dbl>
```

Text Features

Extracting textual features could be the subject of an entire class or even a course.

Just to demonstrate how worthwhile being creative can get here let us just add two textual features here, `n_cat` and `n_dog` in all `essay`s:

```
ok_tr_eng2 <- handle_essays(ok_tr_eng)

str_count_na <- function(string, pattern = "") {
  n <- str_count(string, pattern)
  ifelse(is.na(n), 0, n)
}
```

```
rec_eng_notext <- recipe(pets ~ ., data = ok_tr_eng2) %>%
  update_role(starts_with("essay"), new_role = "discarded") %>%
  step_log(income, starts_with("len"), offset = 1) %>%
  step_impute_mean(income) %>%
  step_normalize(all_numeric(), -has_role("discarded")) %>%
  step_other(all_nominal(), -has_role("discarded"), other = "other
  step_novel(all_nominal(), -has_role("discarded"), -all_outcomes
  step_impute_mode(all_nominal(), -has_role("discarded"), -all_out
  step_dummy(all_nominal(), -all_outcomes(),
             -has_role("discarded"), one_hot = FALSE) %>%
  step_upsample(pets, over_ratio = 1, seed = 42)
```

```
rec_eng_catdog <- recipe(pets ~ ., data = ok_tr_eng2) %>%
  step_mutate(essays = pmap_chr(
    list(essay0, essay1, essay2, essay3, essay4, essay5,
         essay6, essay7, essay8, essay9), str_c),
              n_cat = str_count_na(essays, "cat"),
              n_dog = str_count_na(essays, "dog")) %>%
  update_role(starts_with("essay"), new_role = "discarded") %>%
  step_log(income, starts_with("len"), offset = 1) %>%
  step_impute_mean(income) %>%
  step_normalize(all_numeric(), -has_role("discarded")) %>%
  step_other(all_nominal(), -has_role("discarded"), other = "other
  step_novel(all_nominal(), -has_role("discarded"), -all_outcomes
  step_impute_mode(all_nominal(), -has_role("discarded"), -all_out
  step_dummy(all_nominal(), -all_outcomes(),
             -has_role("discarded"), one_hot = FALSE) %>%
  step_upsample(pets, over_ratio = 1, seed = 42)
```

0.80
0.75
0.70
0.65
AUC
catdog
notext
recipe

Text Features: some ideas for you

- not just words: do cat people tend to use more the letter "z"? no. of commas, no. of dot-dot-dots, no. of links, no. of hashtags, no. of extra spaces, no. of digits
- to stem or not to stem?
  - "plays", "playing", "playa"
- sentiment features
  - not just no. of "hate" vs. "love"
  - also no. of "you" and "I"
- stop words or not to stop words?
- TF-IDF or not to TF-IDF
- Feature Hashing
- n-grams: "I love my cats"
- topics

Using `textrecipes` + `textfeatures`

- The `textrecipes` package lets you calculate a few features with a single `step_textfeature()`
- The `extract_functions` parameter accepts a named list of functions you wish to define on your text data
- By default it will accept `count_functions` list from the `textfeatures` package by Mike Kearney, which gets you basic counts like `n_words`, `n_digits` etc.

But `textfeatures` also has via the `textfeatures()` function:

- basic sentiment features like `sent_bing`
- first word2vec 2-200 dimensions!

```
print(textfeatures:::sentiment_bing("I hate cats!"))
```

```
## [1] -1
```

```
print(textfeatures:::sentiment_bing("I love cats! Hate dogs though
```

```
## [1] 0
```

```
print(textfeatures::word_dims(
  "Why would Kim Jong-un insult me by calling me \"old,\""))
```

```
## INFO  [12:22:13.363] early stopping at 20 iteration
## INFO  [12:22:13.409] early stopping at 20 iteration
##   w1 w2   w3   w4  w5 w6   w7  w8 w9  w10
## 1  0  0 0.05 0.05 0.1  0 0.05 0.7  0 0.05
```

```
ok_tr_eng2 <- ok_tr_eng2 %>%
  mutate(essays = pmap_chr(
    list(essay0, essay1, essay2, essay3, essay4, essay5,
         essay6, essay7, essay8, essay9), str_c))

my_text_funs <- c(textfeatures::count_functions,
                  "sent_bing" = textfeatures:::sentiment_bing)
```

```
recipe(pets ~ ., data = ok_tr_eng2 %>% select(pets, essays)) %>%
  update_role(essays, new_role = "discarded") %>%
  step_textfeature(essays, prefix = "t",
                   extract_functions = my_text_funs) %>%
  step_mutate_at(starts_with("t_"), fn = ~ifelse(is.na(.x), 0, .x)
  prep(ok_tr_eng2 %>% head(10)) %>%
  bake(new_data = NULL) %>%
  select(t_essays_n_uq_words, t_essays_n_hashtags, t_essays_sent_b
```

```
## # A tibble: 10 x 3
##    t_essays_n_uq_words t_essays_n_hashtags t_essays_sent_bing
##                  <dbl>               <dbl>              <dbl>
##  1                   0                   0                  0
##  2                 283                   0                  9
##  3                 345                   0                 16
##  4                   0                   0                  0
##  5                   0                   0                  0
##  6                   0                   0                  0
##  7                   0                   0                  0
##  8                 267                   0                 22
##  9                   0                   0                  0
## 10                 193                   0                 -2
```

Text Features: but the words are most important

We'll take Kuhn & Johnson (2019) approach, and:

1. Filter 1-gram tokens (words) with at least X occurrences
2. Calculate the odds ratio $\frac{P(w|cats)}{1-P(w|cats)} / \frac{P(w|dogs)}{1-P(w|dogs)}$ where $P(w|cats)$ is the % of cats people who used the word w at least once in all essays
3. Calculate a p-value with a FDR correction
4. Choose words with ~~OR of at least |2| and~~ FDR value of maximum 10^{-4}

Then use resampling to see if these have any effect on AUC.

```r
library(tidytext)

tidy_essays <- ok_tr_eng2 %>%
  mutate(id = 1:n()) %>%
  select(id, pets, essays) %>%
  unnest_tokens(word, essays) %>%
  filter(!word %in% stop_words$word,
         str_detect(word, "[a-z]"))

tidy_essays %>% head()
```

```
## # A tibble: 6 x 3
##      id pets  word
##   <int> <chr> <chr>
## 1     2 dogs  bay
## 2     2 dogs  raised
## 3     2 dogs  university
## 4     2 dogs  oregon
## 5     2 dogs  schooled
## 6     2 dogs  hip
```

```r
min_n_words <- tidy_essays %>%
  count(id, word) %>%
  count(word) %>%
  filter(n > 50) %>%
  pull(word)

print(c("cat", "dog", "cats", "dogs") %in% min_n_words)
```

```
## [1] TRUE TRUE TRUE TRUE
```

```r
n_cats_people <- sum(ok_tr_eng2$pets == "cats")
n_dogs_people <- sum(ok_tr_eng2$pets == "dogs")

fisher_test <- function(cats, dogs) {
  cont_tab <- matrix(c(cats, n_cats_people - cats,
                       dogs, n_dogs_people - dogs),
                     ncol = 2, nrow = 2)
  ft <- fisher.test(cont_tab)
  list(OR = ft$estimate, p_val = ft$p.value)
}
```

```r
or_res <- tidy_essays %>%
  filter(word %in% min_n_words) %>%
  count(id, word, pets) %>%
  count(word, pets) %>%
  pivot_wider(id_cols = word, names_from = pets,
              values_from = n, values_fill = list(cats = 0, dogs =
  mutate(fisher_obj = map2(cats, dogs, fisher_test))

or_res <- or_res %>%
  bind_cols(
    bind_rows(or_res$fisher_obj)
  ) %>%
  mutate(fdr = p.adjust(p_val, method = "fdr"))

or_res
```

```
## # A tibble: 835 x 7
##    word        cats  dogs fisher_obj          OR  p_val   fdr
##    <chr>      <int> <int> <list>           <dbl>  <dbl> <dbl>
##  1 ability       16    49 <named list [2]> 1.67  0.0893 0.287
##  2 absolutely    16    67 <named list [2]> 1.21  0.456  0.704
##  3 accent        15    45 <named list [2]> 1.71  0.0793 0.263
##  4 action        19    95 <named list [2]> 1.01  1      1
##  5 active        28   150 <named list [2]> 0.941 0.836  0.966
##  6 activities    20    86 <named list [2]> 1.18  0.505  0.743
##  7 activity       9    47 <named list [2]> 0.968 1      1
##  8 add           15    42 <named list [2]> 1.83  0.0687 0.243
##  9 admit         24   122 <named list [2]> 0.995 1      1
## 10 adult         13    41 <named list [2]> 1.62  0.138  0.354
```

```r
cd_words <- or_res %>%
  arrange(fdr) %>%
  filter(fdr < 10e-4)

cd_words
```

```
## # A tibble: 10 x 7
##    word          cats  dogs fisher_obj          OR    p_val      fdr
##    <chr>        <int> <int> <list>           <dbl>    <dbl>    <dbl>
##  1 cat             46    37 <named list [2]> 6.82  2.19e-16 1.83e-13
##  2 dog              8   230 <named list [2]> 0.162 7.01e-11 2.93e- 8
##  3 cats            31    29 <named list [2]> 5.70  2.42e-10 4.74e- 8
##  4 class           72   144 <named list [2]> 2.79  2.41e-10 4.74e- 8
##  5 href            63   116 <named list [2]> 3.00  2.84e-10 4.74e- 8
##  6 ilink           58   108 <named list [2]> 2.94  2.25e- 9 3.14e- 7
##  7 fantasy         27    31 <named list [2]> 4.60  6.51e- 8 7.76e- 6
##  8 book            96   260 <named list [2]> 2.08  1.14e- 7 1.19e- 5
##  9 intellectual    32    46 <named list [2]> 3.69  1.57e- 7 1.46e- 5
## 10 form            23    32 <named list [2]> 3.76  6.29e- 6 5.25e- 4
```

So these will be our four options for handling text:

1. Not
2. Text features (zero for `NA`)
3. Text features + our flagged words counts (zero for `NA`)
4. Feature Hashing

```
count_words <- function(essays) {
  counts <- map(cd_words$word, ~str_count_na(essays, .x))
  names(counts) = janitor::make_clean_names(str_c("w_", cd_words$w
  counts
}

ok_tr_eng3 <- ok_tr_eng2 %>%
  bind_cols(
    map_dfr(ok_tr_eng2$essays, count_words)
  ) %>%
  select(-starts_with("essay"), -starts_with("len_essay")) %>%
  bind_cols(ok_tr_eng2 %>% select(essays)) %>%
  mutate(across(where(is.character), as.factor))
```

```
rec_eng_notext <- recipe(pets ~ ., data = ok_tr_eng3) %>%
  update_role(starts_with("w_"), essays, new_role = "discarded") %
  step_log(income, offset = 1) %>%
  step_impute_mean(income) %>%
  step_other(all_nominal(), -has_role("discarded"), other = "other
  step_novel(all_nominal(), -has_role("discarded"), -all_outcomes
  step_impute_mode(all_nominal(), -has_role("discarded"), -all_out
  step_dummy(all_nominal(), -all_outcomes(),
             -has_role("discarded"), one_hot = FALSE) %>%
  step_upsample(pets, over_ratio = 1, seed = 42)

rec_eng_textfeat <- recipe(pets ~ ., data = ok_tr_eng3) %>%
  update_role(starts_with("w_"), essays, new_role = "discarded") %
  step_textfeature(essays, prefix = "t",
                   extract_functions = my_text_funs) %>%
  step_mutate_at(starts_with("t_"), fn = ~ifelse(is.na(.x), 0, .x)
  step_log(income, starts_with("t_"), -t_essays_sent_bing, offset
  step_impute_mean(income) %>%
  step_zv(all_numeric()) %>%
  step_other(all_nominal(), -has_role("discarded"), other = "other
  step_novel(all_nominal(), -has_role("discarded"), -all_outcomes
  step_impute_mode(all_nominal(), -has_role("discarded"), -all_out
  step_dummy(all_nominal(), -all_outcomes(),
             -has_role("discarded"), one_hot = FALSE) %>%
  step_upsample(pets, over_ratio = 1, seed = 42)
```

```
rec_eng_fulltext <- recipe(pets ~ ., data = ok_tr_eng3) %>%
  update_role(essays, new_role = "discarded") %>%
  step_textfeature(essays, prefix = "t",
                   extract_functions = my_text_funs) %>%
  step_mutate_at(starts_with("t_"), fn = ~ifelse(is.na(.x), 0, .x)
  step_log(income, starts_with("t_"), -t_essays_sent_bing, offset
  step_impute_mean(income) %>%
  step_zv(all_numeric()) %>%
  step_other(all_nominal(), -has_role("discarded"), other = "other
  step_novel(all_nominal(), -has_role("discarded"), -all_outcomes
  step_impute_mode(all_nominal(), -has_role("discarded"), -all_out
  step_dummy(all_nominal(), -all_outcomes(),
             -has_role("discarded"), one_hot = FALSE) %>%
  step_upsample(pets, over_ratio = 1, seed = 42)

rec_eng_hashtext <- recipe(pets ~ ., data = ok_tr_eng3) %>%
  update_role(starts_with("w_"), essays, new_role = "discarded") %
  step_tokenize(essays) %>%
  step_tokenfilter(essays, min_times = 10) %>%
  step_texthash(essays, num_terms = 2^11) %>%
  step_log(income, offset = 1) %>%
  step_impute_mean(income) %>%
  step_zv(all_numeric()) %>%
  step_other(all_nominal(), -has_role("discarded"), other = "other
  step_novel(all_nominal(), -has_role("discarded"), -all_outcomes
  step_impute_mode(all_nominal(), -has_role("discarded"), -all_out
  step_dummy(all_nominal(), -all_outcomes(),
             -has_role("discarded"), one_hot = FALSE) %>%
  step_upsample(pets, over_ratio = 1, seed = 42)
```

```r
set.seed(42)
cv_splits_eng <- vfold_cv(ok_tr_eng3, v = 10, strata = pets)

lst_recs <- list("notext" = rec_eng_notext,
                 "textfeat" = rec_eng_textfeat,
                 "fulltext" = rec_eng_fulltext,
                 "hashtext" = rec_eng_hashtext)
```

```r
cv_res_eng <- cv_splits_eng %>%
  bind_cols(
    map2_dfc(lst_recs, names(lst_recs), train_recipe,
             splits = cv_splits_eng$splits)
  ) %>%
  pivot_longer(cols = names(lst_recs),
               names_to = "recipe", values_to = "AUC") %>%
  select(id, recipe, AUC)

cv_res_eng_meds <- cv_res_eng %>%
  group_by(recipe) %>%
  summarise(AUC = mean(AUC), id = "All")
```

Haven't forgot about our other categorical variables

Let's use some common sense (see code!).

```
cd_words <- cd_words %>%
  bind_rows(
    tibble(word =
             c("my cat", "my dog", "love cat", "love dog", "kitt",
               "have cat", "have dog", "have a cat", "have a dog")
           cats = NULL, dogs = NULL, fisher_obj = NULL, OR = 0,
           p_val = 0, fdr = 0))
```

```
ok_tr_eng4 <- full_features(ok_tr_eng)

colnames(ok_tr_eng4)
```

```
##  [1] "age"                 "body_type"           "diet"
##  [4] "drinks"              "drugs"               "education"
##  [7] "ethnicity"           "height"              "income"
## [10] "job"                 "location"            "orientation"
## [13] "pets"                "religion"            "sex"
## [16] "smokes"              "status"              "len_essay0"
## [19] "len_essay1"          "len_essay2"          "len_essay3"
## [22] "len_essay4"          "len_essay5"          "len_essay6"
## [25] "len_essay7"          "len_essay8"          "len_essay9"
## [28] "essays"              "offspring_no_kids"   "offspring_one_kid"
```

```
rec_eng_commonsense <- recipe(pets ~ ., data = ok_tr_eng4) %>%
  step_textfeature(essays, prefix = "t",
                   extract_functions = my_text_funs) %>%
  update_role(essays, new_role = "discarded") %>%
  step_mutate_at(starts_with("t_"), fn = ~ifelse(is.na(.x), 0, .x)
  step_log(income, starts_with("len_"), starts_with("t_"),
           -t_essays_sent_bing, offset = 1) %>%
  step_impute_mean(income) %>%
  step_other(
    all_nominal(), -has_role("discarded"), -all_outcomes(),
    other = "all_else", threshold = 0.1) %>%
  step_novel(
    all_nominal(), -has_role("discarded"), -all_outcomes()) %>%
  step_impute_mode(all_nominal(), -has_role("discarded"), -all_out
  step_dummy(all_nominal(), -all_outcomes(),
             -has_role("discarded"), one_hot = FALSE) %>%
  step_upsample(pets, over_ratio = 1, seed = 42)
```

```r
set.seed(42)
cv_splits_eng2 <- vfold_cv(ok_tr_eng4, v = 10, strata = pets)

lst_recs <- list("commonsense" = rec_eng_commonsense)
```

```r
cv_res_commonsense <- cv_splits_eng2 %>%
  bind_cols(
    map2_dfc(lst_recs, names(lst_recs), train_recipe,
             splits = cv_splits_eng2$splits)
  ) %>%
  pivot_longer(cols = names(lst_recs),
               names_to = "recipe", values_to = "AUC") %>%
  select(id, recipe, AUC)

cv_res_cs_meds <- cv_res_commonsense %>%
  group_by(recipe) %>%
  summarise(AUC = mean(AUC), id = "All")
```

```
cv_res_eng %>%
  bind_rows(cv_res_commonsense) %>%
  mutate(recipe = fct_relevel(recipe,
                              c("notext", "hashtext", "textfeat",
  ggplot(aes(recipe, AUC, group = id, color = id)) +
  geom_line() + geom_line(data = cv_res_eng_meds %>% bind_rows(cv_
  geom_point() + geom_point(data = cv_res_eng_meds %>% bind_rows(c
  guides(color = "none") + theme_light()
```

Numerical Variables

Applications

Of Data Science

What could the problem be?

The problem almost always depends on the combination between models and variables:

- Varied scale (and kNN, SVM) --> `step_center()`, `step_scale()`,`step_normalize()`,`step_range()`
- Bounded range of outcome (e.g. percentage and, well, almost any model) --> `step_logit()`
- Skewed data and outliers (and linear regression, NN, kNN, SVM, but less for Trees)
- Non-linearity (and LR)
- Co-linearity (and LR, NN)

Skewed Data: Common Transformations

Box-Cox

In its simplified form:

$$x_i^{(\lambda)} = \begin{cases} \frac{x_i^{\lambda}-1}{\lambda} & \text{if } \lambda \neq 0 \\ \ln x_i & \text{if } \lambda = 0 \end{cases}$$

Originally the Box-Cox transformation was developed to transform a dependent y variable by maximizing the log-likelihood of seeing $y^{(\lambda)}$ in a `lm` against some X.

But λ could be estimated in an unsupervised way as well:

```
MASS::boxcox(ok_tr_eng$age ~ 1)
```

```
tibble(age = ok_tr_eng$age,
       age_bc = -1/age) %>%
  pivot_longer(starts_with("age"), names_to = "age") %>%
  ggplot(aes(value)) +
  geom_histogram(bins = 30, fill = "red", alpha = 0.5) +
  facet_wrap(~ age, scales = "free") + theme_light()
```

Yeo-Johnson

Revised Box-Cox to allow for zero and negative values:

$$x_i^{(\lambda)} = \begin{cases} \frac{(x_i+1)^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0, y \geq 0 \\ \ln\left(x_i + 1\right) & \text{if } \lambda = 0, y \geq 0 \\ \frac{-[(-x_i+1)^{2-\lambda} - 1]}{2-\lambda} & \text{if } \lambda \neq 0, y < 0 \\ -\ln\left(-x_i + 1\right) & \text{if } \lambda = 0, y < 0 \end{cases}$$

Both transformations have `step`s in `recipes`:

```
recipe(pets ~ ., data = ok_tr_eng) %>%
  step_BoxCox(income)
```

```
recipe(pets ~ ., data = ok_tr_eng) %>%
  step_YeoJohnson(income)
```

Non-Linearity: Basis Expansions and Splines

Let's borrow a continuous y variable like `log(income)` to simplify:

```
ggplot(ok_tr_eng, aes(age, log(income))) +
  geom_point() +
  theme_light()
```

You could do this manually in `lm()` like so

```
mod_lm <- lm(log(income) ~ I(pmax(0, 45 - age)) + I(pmax(0, age -
             data = ok_tr_eng)

summary(mod_lm)
```

```
##
## Call:
## lm(formula = log(income) ~ I(pmax(0, 45 - age)) + I(pmax(0, age -
##     45)), data = ok_tr_eng)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.5353 -0.6694 -0.1082  0.3832  3.1974
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)          11.466138   0.090137 127.208  < 2e-16 ***
## I(pmax(0, 45 - age)) -0.032617   0.005668  -5.755 1.42e-08 ***
## I(pmax(0, age - 45)) -0.027372   0.013186  -2.076   0.0384 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8933 on 565 degrees of freedom
##   (2437 observations deleted due to missingness)
## Multiple R-squared:  0.05605,    Adjusted R-squared:  0.05271
## F-statistic: 16.77 on 2 and 565 DF,  p-value: 8.381e-08
```

Or use `splines::bs()` with `degree = 1` and `knots = 45`.

```
library(splines)
mod_lm_bs <- lm(log(income) ~ bs(age, degree = 1, knots = 45),
                data = ok_tr_eng)

summary(mod_lm_bs)
```

```
##
## Call:
## lm(formula = log(income) ~ bs(age, degree = 1, knots = 45), data = ok_tr
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.5353 -0.6694 -0.1082  0.3832  3.1974
##
## Coefficients:
##                                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)                       10.5855     0.0814 130.044  < 2e-16 **
## bs(age, degree = 1, knots = 45)1   0.8807     0.1530   5.755 1.42e-08 **
## bs(age, degree = 1, knots = 45)2   0.2237     0.2808   0.797    0.426
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8933 on 565 degrees of freedom
##   (2437 observations deleted due to missingness)
## Multiple R-squared:  0.05605,    Adjusted R-squared:  0.05271
## F-statistic: 16.77 on 2 and 565 DF,  p-value: 8.381e-08
```

How will the variables look like?

```
X <- model.matrix(mod_lm)
age_no_na <- ok_tr_eng$age[!is.na(ok_tr_eng$income)]
head(cbind(age_no_na, X), 4)
```

```
##    age_no_na (Intercept) I(pmax(0, 45 - age)) I(pmax(0, age - 45))
## 3         26           1                   19                    0
## 5         24           1                   21                    0
## 10        21           1                   24                    0
## 15        25           1                   20                    0
```

```
X <- model.matrix(mod_lm_bs)
colnames(X)[2:3] <- c("age1", "age2")
head(cbind(age_no_na, X), 4)
```

```
##    age_no_na (Intercept)      age1 age2
## 3         26           1 0.2962963    0
## 5         24           1 0.2222222    0
## 10        21           1 0.1111111    0
## 15        25           1 0.2592593    0
```

`splines::bs()` re-parametrizes the variables but it is the same model!

How will the fit look like?

```
ggplot(ok_tr_eng[!is.na(ok_tr_eng$income), ]) +
  geom_point(aes(age, log(income))) +
  geom_line(aes(age, mod_lm_bs$fit), color = "red", lwd = 2, alpha
  theme_light()
```

Or with a simple cubic spline with one knot in `bs()`

Manually with `step_relu()`

```
recipe(income ~ age, ok_tr_eng) %>%
  step_naomit(income) %>%
  step_relu(age, shift = 45, reverse = TRUE) %>%
  step_relu(age, shift = 45) %>%
  prep() %>%
  bake(new_data = NULL) %>%
  head()
```

```
## # A tibble: 6 x 4
##     age income left_relu_age right_relu_age
##   <dbl>  <dbl>         <dbl>          <dbl>
## 1    26  20000            19              0
## 2    24  60000            21              0
## 3    21  20000            24              0
## 4    25  20000            20              0
## 5    35  80000            10              0
## 6    69  80000             0             24
```

For splines and polynomial regression use `step_bs()`, `step_ns()` and `step_poly()`

```
recipe(income ~ age, ok_tr_eng) %>%
  step_naomit(income) %>%
  step_bs(age, degree = 3) %>%
  prep() %>%
  bake(new_data = NULL) %>%
  head()
```

```
## # A tibble: 6 x 4
##   income age_bs_1 age_bs_2 age_bs_3
##    <dbl>    <dbl>    <dbl>    <dbl>
## 1  20000    0.335  0.0622  0.00386
## 2  60000    0.275  0.0366  0.00163
## 3  20000    0.156  0.00977 0.000204
## 4  20000    0.306  0.0488  0.00259
## 5  80000    0.444  0.222   0.0370
## 6  80000    0      0       1
```

Co-linearity: Consider reducing dimensionality

> A conventional thought process is that the more predictors that are included, the better chance the model will have to find the signal. But this is not the case; including irrelevant predictors can have detrimental effects on the final model. The negative impacts include:
>
> - increasing the computational time required for training a model
> - decreasing the predictive performance
> - complicating the predictor importance calculation.
>
> By including any potential predictor, we end up with a sub-optimal model.

Kuhn & Johnson (2019)

Common *dimensionality reduction* methods (some of which we've talked about!):

- PCA, Kernel PCA, Sparse PCA --> `step_pca()`, `step_kpca()`
- NMF --> `step_nnmf()`
- PLS --> `step_pls()`

All of these will find some projection matrix B_{pxk} where k is the number of columns (e.g. PCs) to choose and would typically be much lower than p, the original dimension.

You would then project your X matrix like so: $X^* = XB$ (in most methods X^* columns would also be linearly independent, or "close" to it), and you would fit your data on this transformed X^*

⚠ No guarantee for improvement in performance! In fact...

💡 What is an alternative to reduce dimensionality?

Interaction Effects

Applications

Of Data Science

What is Interaction?

But seriously

```
lm_int <- lm(log(income) ~ sex + height + sex:height,
             data = ok_tr_interaction %>% filter(between(height, 6
summary(lm_int)
```

```
##
## Call:
## lm(formula = log(income) ~ sex + height + sex:height, data = ok_tr_intera
##     filter(between(height, 60, 80)))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.17391 -0.78673  0.00961  0.48291  3.10249
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  9.299263   1.916753   4.852 1.82e-06 ***
## sexm         0.577743   2.395687   0.241    0.810
## height       0.022803   0.029294   0.778    0.437
## sexm:height -0.007008   0.035666  -0.196    0.844
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9178 on 366 degrees of freedom
##   (1614 observations deleted due to missingness)
```

```
ok_tr_interaction %>% filter(between(height, 60, 80)) %>%
  ggplot(aes(height, log(income), color = sex)) +
  geom_point() +
  geom_smooth(method = "lm")
```

💡 How would you visualize an interaction between two continuous features? Two categorical? Three?...

You get an interaction! You get an interaction!

With the last `rec_eng_commonsense` recipe we reached over 150 features.

So all two-way interactions would mean adding... over 11K features. All three-way interactions would mean adding... over 550K features.

> What danger comes to mind? How would you defend yourself?
>
> Which models surface interactions naturally? What about our model?
>
> Should you include interaction terms before or after pre-processing?

Obviously, we need some control for this exploding no. of features:

- Expert knowledge (e.g. when it comes to income most variables probably interact with...?)
- Heredity principle
- Resampling, Penalize
- When p is huge: Two-Stage Modeling combined with your favorite Feature Selection approach
- Whatever you choose, visualize!

So these will be our four options for handling interactions:

1. Not
2. Adding only interactions between `sex` and everything
3. Two Stage: Screening all "main effects" features selected by a single `glmnet` stage then all two-way interactions of those features
4. Two Stage: Screening all two-way interactions by multiple comparisons of a `glm` main effects model vs. main effects + interaction, combined with FDR adjustment for the p-value
5. All two-way interactions 😎 (with a `step_nzv()` though)

For stages 3 and 4 let's use previously available data.

```
ok_tr_combined <- bind_rows(
  ok_tr_missing,
  ok_tr_imbalance,
  ok_tr_eng
  )
ok_tr_combined <- full_features(ok_tr_combined)
```

For stage 3 need to screen out "main effects" features which passed `glmnet`

```
main_data <- rec_eng_commonsense %>% prep(ok_tr_combined) %>% bake

mod_lr_res <- mod_lr %>%
  fit(pets ~ ., data = main_data) %>%
  pluck("fit") %>%
  tidy()

mod_lr_res
```

```
## # A tibble: 4,730 x 5
##    term          step  estimate lambda dev.ratio
##    <chr>        <dbl>     <dbl>  <dbl>     <dbl>
##  1 (Intercept)      1  2.26e-14 0.0963 -6.41e-16
##  2 (Intercept)      2 -1.42e- 2 0.0877  5.46e- 3
##  3 (Intercept)      3 -2.48e- 2 0.0800  1.44e- 2
##  4 (Intercept)      4 -3.38e- 2 0.0728  2.42e- 2
##  5 (Intercept)      5 -2.74e- 2 0.0664  3.54e- 2
##  6 (Intercept)      6 -2.71e- 2 0.0605  4.79e- 2
##  7 (Intercept)      7  5.58e- 3 0.0551  6.19e- 2
##  8 (Intercept)      8  7.41e- 2 0.0502  7.59e- 2
##  9 (Intercept)      9  1.18e- 1 0.0458  8.94e- 2
## 10 (Intercept)     10  1.63e- 1 0.0417  1.02e- 1
## # ... with 4,720 more rows
```

We can plot no. of features selected by `glmnet` by lambda

```
mod_lr_res %>%
  count(lambda) %>%
  ggplot(aes(lambda, n)) +
  geom_line(lwd = 2, color = "red", alpha = 0.5) +
  labs(y = "N features selected by glmnet") +
  theme_light()
```

We see a plateau around 12-15 features, let's choose those.

```
mod_lr_res %>%
  filter(step == 6) %>%
  select(term, estimate) %>%
  arrange(-abs(estimate))
```

```
## # A tibble: 10 x 2
##    term                estimate
##    <chr>                  <dbl>
##  1 w_my_cat            -0.474
##  2 religion_christian   0.348
##  3 w_my_dog             0.266
##  4 status_all_else     -0.170
##  5 body_type_all_else -0.0656
##  6 w_dog                0.0450
##  7 (Intercept)         -0.0271
##  8 drinks_socially      0.0207
##  9 w_cat               -0.0141
## 10 w_cats              -0.00126
```

These make sense, let's choose them as our main effects from which we pick all two-way interactions.

```
main_effects <- mod_lr_res %>%
  filter(step == 6, term != "(Intercept)") %>%
  pull(term)
```

For stage 4 we need to iterate over all possible two-way interactions, each time adding one into a full main effects `glm` model, getting the p-value, and applying FDR adjustment.

```r
mod_glm_base <- glm(pets ~ .,
                    data = main_data %>% select(-ends_with("_new")
                    family = "binomial")

extract_interaction_p_val <- function(var1, var2) {
  pb$tick()$print()
  int_term <- str_c(var1, ":", var2)
  int_term2 <- str_c(var2, ":", var1)
  mod_glm <- quietly(update)(
    mod_glm_base, as.formula(str_c(".~.+", int_term))
  )
  if(length(mod_glm$warnings) > 1) {
    return(NA_real_)
  }
  p_val <- tidy(mod_glm$result) %>%
    filter(term %in% c(int_term, int_term2)) %>%
    pull(p.value)
  if (length(p_val) == 0) {
    return(NA_real_)
  }
  p_val
}

all_vars <- colnames(main_data %>% select(-ends_with("_new")))
all_ints <- t(combn(all_vars[-which(all_vars == "pets")], 2))
colnames(all_ints) <- c("var1", "var2")
all_ints <- as_tibble(all_ints)

pb <- progress_estimated(nrow(all_ints))
all_ints <- all_ints %>%
```

Let's filter only the *most* convincing interactions.

```
pairwise_ints <- all_ints %>%
  arrange(fdr) %>%
  filter(fdr < 10e-7)

pairwise_ints
```

```
## # A tibble: 10 x 4
##    var1                var2                      p_val      fdr
##    <chr>               <chr>                     <dbl>    <dbl>
##  1 ethnicity_other     location_other         2.68e-14 2.06e-10
##  2 offspring_one_kid   speaks_n_fluently      5.06e-13 1.94e- 9
##  3 religion_laughing   ethnicity_white        1.15e-11 2.93e- 8
##  4 len_essay0          education_kind_working 8.94e-11 1.53e- 7
##  5 offspring_one_kid   speaks_spanish         9.97e-11 1.53e- 7
##  6 w_possibly          education_all_else     1.23e-10 1.57e- 7
##  7 w_games             w_possibly             1.86e-10 2.04e- 7
##  8 w_books             w_sci                  3.98e-10 3.81e- 7
##  9 w_fi                t_essays_n_uq_words    4.54e-10 3.87e- 7
## 10 education_all_else  status_all_else        8.89e-10 6.81e- 7
```

First make sure you understand the difference between `other`, `all_else` etc.

```
ok_tr_combined %>% count(ethnicity, sort = TRUE)
```

```
## # A tibble: 6 x 2
##   ethnicity     n
##   <fct>     <int>
## 1 white      3829
## 2 other      1672
## 3 asian       743
## 4 hispanic    403
## 5 black       230
## 6 indian      125
```

```
main_data %>% select(starts_with("ethnicity")) %>% head()
```

```
## # A tibble: 6 x 4
##   ethnicity_other ethnicity_white ethnicity_all_else ethnicity_new
##             <dbl>           <dbl>               <dbl>         <dbl>
## 1               0               1                   0             0
## 2               0               1                   0             0
## 3               0               1                   0             0
## 4               1               0                   0             0
## 5               1               0                   0             0
## 6               0               1                   0             0
```

What are those people with all `ethnicity` columns with zero?

```
main_data %>% select(starts_with("ethnicity")) %>%
  filter(across(everything(), ~.x == 0))
```

```
## # A tibble: 1,016 x 4
##    ethnicity_other ethnicity_white ethnicity_all_else ethnicity_new
##              <dbl>           <dbl>               <dbl>         <dbl>
##  1               0               0                   0             0
##  2               0               0                   0             0
##  3               0               0                   0             0
##  4               0               0                   0             0
##  5               0               0                   0             0
##  6               0               0                   0             0
##  7               0               0                   0             0
##  8               0               0                   0             0
##  9               0               0                   0             0
## 10               0               0                   0             0
## # ... with 1,006 more rows
```

Same with `location`.

```
ok_tr_combined %>% count(location, sort = TRUE)
```

```
## # A tibble: 4 x 2
##   location     n
##   <fct>    <int>
## 1 sf        3742
## 2 other     2040
## 3 oakland    776
## 4 berkeley   444
```

```
main_data %>% select(starts_with("location")) %>% head()
```

```
## # A tibble: 6 x 4
##   location_other location_sf location_all_else location_new
##            <dbl>       <dbl>             <dbl>        <dbl>
## 1              0           0                 1            0
## 2              1           0                 0            0
## 3              0           1                 0            0
## 4              0           1                 0            0
## 5              1           0                 0            0
## 6              0           0                 0            0
```

```
main_data %>% group_by(ethnicity_other, location_other) %>%
  summarise(pct_cats = mean(pets == "cats"), n = n()) %>% ungroup
  mutate(across(c("ethnicity_other", "location_other"), as.factor)
  ggplot(aes(ethnicity_other, pct_cats, color = location_other)) +
  geom_line(aes(group = location_other)) + geom_point(aes(size = n
  scale_y_continuous(labels = percent_format()) +
  theme_light()
```

Try stating what you see in words.

```
main_data %>%
  mutate(speaks_n_fluently = ifelse(speaks_n_fluently >= 3, "3+",
  group_by(speaks_n_fluently, offspring_one_kid) %>%
  summarise(pct_cats = mean(pets == "cats"), n = n()) %>% ungroup
  mutate(across(c("speaks_n_fluently", "offspring_one_kid"), as.fa
  ggplot(aes(speaks_n_fluently, pct_cats, color = offspring_one_ki
  geom_line(aes(group = offspring_one_kid)) + geom_point(aes(size
  scale_y_continuous(labels = percent_format()) +
  theme_light()
```

```
main_data %>%
  group_by(religion_laughing, ethnicity_white) %>%
  summarise(pct_cats = mean(pets == "cats"), n = n()) %>% ungroup
  mutate(across(c("religion_laughing", "ethnicity_white"), as.fact
  ggplot(aes(religion_laughing, pct_cats, color = ethnicity_white)
  geom_line(aes(group = ethnicity_white)) + geom_point(aes(size =
  scale_y_continuous(labels = percent_format()) +
  theme_light()
```

```
ok_tr_interaction2 <- full_features(ok_tr_interaction)

rec_int_noint <- rec_eng_commonsense

rec_int_sex <- recipe(pets ~ ., data = ok_tr_interaction2) %>%
  step_textfeature(essays, prefix = "t",
                   extract_functions = my_text_funs) %>%
  update_role(essays, new_role = "discarded") %>%
  step_mutate_at(starts_with("t_"), fn = ~ifelse(is.na(.x), 0, .x)
  step_log(income, starts_with("len_"), starts_with("t_"),
           -t_essays_sent_bing, offset = 1) %>%
  step_impute_mean(income) %>%
  step_other(
    all_nominal(), -has_role("discarded"), -all_outcomes(),
    other = "all_else", threshold = 0.1) %>%
  step_novel(
    all_nominal(), -has_role("discarded"), -all_outcomes()) %>%
  step_impute_mode(all_nominal(), -has_role("discarded"), -all_out
  step_dummy(all_nominal(), -all_outcomes(),
             -has_role("discarded"), one_hot = FALSE) %>%
  step_interact(~starts_with("sex"):all_numeric()) %>%
  step_nzv(all_numeric(), freq_cut = 99/1) %>%
  step_upsample(pets, over_ratio = 1, seed = 42)
```

```r
# pairwise interactions string code from Max Kuhn's workshop:
# https://github.com/topepo/FES/blob/master/07_Detecting_Interacti

topmain_ints <- t(combn(main_effects, 2))
colnames(topmain_ints) <- c("var1", "var2")
topmain_ints <- topmain_ints %>%
  as_tibble() %>%
  mutate(
    term = str_c(var1, ":", var2)
  ) %>%
  pull(term) %>%
  paste(collapse = "+")

topmain_ints <- str_c("~", topmain_ints)
topmain_ints <- as.formula(topmain_ints)

topmain_ints
```

```
## ~w_cat:w_dog + w_cat:w_cats + w_cat:w_my_cat + w_cat:w_my_dog +
##     w_cat:body_type_all_else + w_cat:drinks_socially + w_cat:religion_chr
##     w_cat:status_all_else + w_dog:w_cats + w_dog:w_my_cat + w_dog:w_my_do
##     w_dog:body_type_all_else + w_dog:drinks_socially + w_dog:religion_chr
##     w_dog:status_all_else + w_cats:w_my_cat + w_cats:w_my_dog +
##     w_cats:body_type_all_else + w_cats:drinks_socially + w_cats:religion_
##     w_cats:status_all_else + w_my_cat:w_my_dog + w_my_cat:body_type_all_e
##     w_my_cat:drinks_socially + w_my_cat:religion_christian +
##     w_my_cat:status_all_else + w_my_dog:body_type_all_else +
##     w_my_dog:drinks_socially + w_my_dog:religion_christian +
##     w_my_dog:status_all_else + body_type_all_else:drinks_socially +
```

```
rec_int_topmain <- recipe(pets ~ ., data = ok_tr_interaction2) %>%
  step_textfeature(essays, prefix = "t",
                   extract_functions = my_text_funs) %>%
  update_role(essays, new_role = "discarded") %>%
  step_mutate_at(starts_with("t_"), fn = ~ifelse(is.na(.x), 0, .x)
  step_log(income, starts_with("len_"), starts_with("t_"),
           -t_essays_sent_bing, offset = 1) %>%
  step_impute_mean(income) %>%
  step_other(
    all_nominal(), -has_role("discarded"), -all_outcomes(),
    other = "all_else", threshold = 0.1) %>%
  step_novel(
    all_nominal(), -has_role("discarded"), -all_outcomes()) %>%
  step_impute_mode(all_nominal(), -has_role("discarded"), -all_out
  step_dummy(all_nominal(), -all_outcomes(),
             -has_role("discarded"), one_hot = FALSE) %>%
  step_interact(topmain_ints) %>%
  step_nzv(all_numeric(), freq_cut = 99/1) %>%
  step_upsample(pets, over_ratio = 1, seed = 42)
```

```
rec_int_topints <- recipe(pets ~ ., data = ok_tr_interaction2) %>%
  step_textfeature(essays, prefix = "t",
                   extract_functions = my_text_funs) %>%
  update_role(essays, new_role = "discarded") %>%
  step_mutate_at(starts_with("t_"), fn = ~ifelse(is.na(.x), 0, .x)
  step_log(income, starts_with("len_"), starts_with("t_"),
           -t_essays_sent_bing, offset = 1) %>%
  step_impute_mean(income) %>%
  step_other(
    all_nominal(), -has_role("discarded"), -all_outcomes(),
    other = "all_else", threshold = 0.1) %>%
  step_novel(
    all_nominal(), -has_role("discarded"), -all_outcomes()) %>%
  step_impute_mode(all_nominal(), -has_role("discarded"), -all_out
  step_dummy(all_nominal(), -all_outcomes(),
             -has_role("discarded"), one_hot = FALSE) %>%
  step_interact(topint_ints) %>%
  step_nzv(all_numeric(), freq_cut = 99/1) %>%
  step_upsample(pets, over_ratio = 1, seed = 42)
```

```
rec_int_all <- recipe(pets ~ ., data = ok_tr_interaction2) %>%
  step_textfeature(essays, prefix = "t",
                   extract_functions = my_text_funs) %>%
  update_role(essays, new_role = "discarded") %>%
  step_mutate_at(starts_with("t_"), fn = ~ifelse(is.na(.x), 0, .x)
  step_log(income, starts_with("len_"), starts_with("t_"),
           -t_essays_sent_bing, offset = 1) %>%
  step_impute_mean(income) %>%
  step_other(
    all_nominal(), -has_role("discarded"), -all_outcomes(),
    other = "all_else", threshold = 0.1) %>%
  step_novel(
    all_nominal(), -has_role("discarded"), -all_outcomes()) %>%
  step_impute_mode(all_nominal(), -has_role("discarded"), -all_out
  step_dummy(all_nominal(), -all_outcomes(),
             -has_role("discarded"), one_hot = FALSE) %>%
  step_interact(~all_numeric():all_numeric()) %>%
  step_nzv(all_numeric(), freq_cut = 95/5) %>%
  step_upsample(pets, over_ratio = 1, seed = 42)
```

```r
set.seed(42)
cv_splits_int <- vfold_cv(ok_tr_interaction2, v = 10, strata = pet

lst_recs <- list("noint" = rec_int_noint,
                 "sex" = rec_int_sex,
                 "topmain" = rec_int_topmain,
                 "topints" = rec_int_topints,
                 "all" = rec_int_all)
```

```r
cv_res_int <- cv_splits_int %>%
  bind_cols(
    map2_dfc(lst_recs, names(lst_recs), train_recipe,
             splits = cv_splits_int$splits)
  ) %>%
  pivot_longer(cols = names(lst_recs),
               names_to = "recipe", values_to = "AUC") %>%
  select(id, recipe, AUC)

cv_res_int_meds <- cv_res_int %>%
  group_by(recipe) %>%
  summarise(AUC = mean(AUC), id = "All")
```

```
cv_res_int %>%
  mutate(recipe = fct_relevel(recipe, c("noint", "sex", "topmain",
  ggplot(aes(recipe, AUC, group = id, color = id)) +
  geom_line() + geom_line(data = cv_res_int_meds, color = "black",
  geom_point() + geom_point(data = cv_res_int_meds, color = "blac
  guides(color = "none") + theme_light()
```

Tuning Params

Everything is Tunable

I can think of at least 5 semi-random quantitative decisions so far, can you?

But let us simply vary Elastic Net α and λ:

$$\min_{\beta_0,\beta} \frac{1}{N} \sum_{i=1}^{N} w_i l(y_i, \beta_0 + \beta^T x_i) + \lambda[(1-\alpha)||\beta||_2^2/2 + \alpha||\beta||_1]$$

```
library(tune)

mod_en_tune <- logistic_reg(penalty = tune(),
                            mixture = tune()) %>%
  set_engine("glmnet")

en_grid <- grid_regular(penalty(range(-6, 0)),
                        mixture(range(0, 1)), levels = c(7, 11))
```

```r
ok_tr_tuning2 <- full_features(ok_tr_tuning)

set.seed(42)
cv_splits_tune <- vfold_cv(ok_tr_tuning2, v = 10, strata = pets)
```

```r
tune_res <- tune_grid(object = mod_en_tune,
                      preprocessor = rec_eng_commonsense,
                      resamples = cv_splits_tune,
                      grid = en_grid,
                      metrics = metric_set(roc_auc),
                      control = control_grid(verbose = TRUE))

collect_metrics_tune_res <- collect_metrics(tune_res)

collect_metrics_tune_res
```

```
## # A tibble: 77 x 7
##     penalty mixture .metric .estimator  mean     n std_err
##       <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl>
##  1 0.000001     0   roc_auc binary     0.775    10  0.0173
##  2 0.000001     0.1 roc_auc binary     0.768    10  0.0168
##  3 0.000001     0.2 roc_auc binary     0.767    10  0.0167
##  4 0.000001     0.3 roc_auc binary     0.767    10  0.0167
##  5 0.000001     0.4 roc_auc binary     0.766    10  0.0167
##  6 0.000001     0.5 roc_auc binary     0.766    10  0.0167
##  7 0.000001     0.6 roc_auc binary     0.767    10  0.0168
##  8 0.000001     0.7 roc_auc binary     0.767    10  0.0168
##  9 0.000001     0.8 roc_auc binary     0.767    10  0.0168
```

```
collect_metrics_tune_res %>%
  mutate(auc = mean, mixture = factor(mixture)) %>%
  ggplot(aes(penalty, auc, color = mixture)) +
  geom_line() + geom_point() +
  theme_light()
```

```r
show_best(tune_res, n = 10, metric = "roc_auc")
```

```
## # A tibble: 10 x 7
##    penalty mixture .metric .estimator  mean     n std_err
##      <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl>
##  1    0.01     0.6 roc_auc binary     0.783    10  0.0166
##  2    0.01     0.7 roc_auc binary     0.783    10  0.0165
##  3    0.01     0.5 roc_auc binary     0.783    10  0.0168
##  4    0.01     0.8 roc_auc binary     0.783    10  0.0165
##  5    0.01     0.9 roc_auc binary     0.783    10  0.0163
##  6    0.01     0.4 roc_auc binary     0.783    10  0.0169
##  7    0.01     1   roc_auc binary     0.782    10  0.0160
##  8    0.01     0.3 roc_auc binary     0.781    10  0.0170
##  9    0.1      0   roc_auc binary     0.781    10  0.0166
## 10    0.1      0.1 roc_auc binary     0.779    10  0.0157
```

So what would you do?

Cutoff choice

The model is there.

Remember it is best to predict a score, let the model's "clients" choose what to do with that score.

Also remember the "I don't know" strategy!

But if we must, let us provide:

- a score to maximize precision of `cats` (while maintaing acceptable levels of recall)
- a score to maximize recall of `cats` (while maintaing acceptable levels of precision)
- a middle score

```
ok_tr_combined <- full_features(
  bind_rows(
    ok_tr_missing,
    ok_tr_imbalance,
    ok_tr_eng,
    ok_tr_interaction,
    ok_tr_tuning))

ok_tr_cutoff2 <- full_features(ok_tr_cutoff)

rec_cutoff <- rec_eng_commonsense %>%
  prep(ok_tr_combined)

ok_tr_for_cutoff <- rec_cutoff %>%
  bake(new_data = NULL)

ok_te_for_cutoff <- rec_cutoff %>%
  bake(ok_tr_cutoff2)

mod_lr_cutoff <- logistic_reg(penalty = 0.01, mixture = 0.5) %>%
  set_engine("glmnet")

mod_lr_cutoff <- mod_lr_cutoff %>%
  fit(pets ~ ., data = ok_tr_for_cutoff)

pred_cutoff <-  mod_lr_cutoff %>%
  predict(new_data = ok_te_for_cutoff, penalty = 0.01, type = "pro
  mutate(truth = ok_te_for_cutoff$pets)
```

```r
library(pROC)

cutoffs <- seq(0.1, 0.9, 0.1)

roc_obj <- roc(pred_cutoff$truth, pred_cutoff$.pred_cats,
               levels = c("dogs", "cats"))
coords(roc_obj, x = cutoffs,
                ret = c("accuracy", "recall", "precision",
                        "specificity", "npv"),
                transpose = TRUE)
```

```
##                    [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
## accuracy    0.20770289 0.2764787 0.4277854 0.6148556 0.7262724 0.8005502
## recall      1.00000000 0.9915966 0.9243697 0.8319328 0.6218487 0.4369748
## precision   0.17122302 0.1835148 0.2127660 0.2757660 0.3245614 0.4000000
## specificity 0.05263158 0.1365132 0.3305921 0.5723684 0.7467105 0.8717105
## npv         1.00000000 0.9880952 0.9571429 0.9456522 0.9098196 0.8877722
##                  [,7]      [,8]      [,9]
## accuracy    0.8308116 0.8610729 0.8583219
## recall      0.2773109 0.2521008 0.1512605
## precision   0.4714286 0.7142857 0.9000000
## specificity 0.9391447 0.9802632 0.9967105
## npv         0.8691020 0.8700730 0.8571429
```

Predict on Test set

Long time no see!

```
set.seed(42)
random_cats_to_get_rid_of <- okcupid_test %>%
  mutate(id = 1:n()) %>%
  filter(pets == "cats") %>%
  sample_n(40) %>%
  pull(id)

okcupid_test2 <- okcupid_test %>%
  mutate(id = 1:n()) %>%
  filter(!id %in% random_cats_to_get_rid_of) %>%
  select(-id)
okcupid_test2 <- full_features(okcupid_test2)

okcupid_test2 %>% count(pets) %>% mutate(pct = n / sum(n))
```

```
## # A tibble: 2 x 3
##   pets      n   pct
##   <fct> <int> <dbl>
## 1 cats    653 0.165
## 2 dogs   3307 0.835
```

```r
okcupid_test_baked <- rec_cutoff %>%
  bake(okcupid_test2)

mod_lr_test <- logistic_reg(penalty = 0.01, mixture = 0.5) %>%
  set_engine("glmnet")

mod_lr_test <- mod_lr_test %>%
  fit(pets ~ ., data = ok_tr_for_cutoff)

pred_test <-  mod_lr_test %>%
  predict(new_data = okcupid_test_baked, penalty = 0.01, type = "p
  mutate(truth = okcupid_test_baked$pets)
```

```r
roc_auc(pred_test, truth, .pred_cats)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.786
```

```r
roc_lr <- roc_curve(
  tibble(truth = pred_test$truth, cats = pred_test$.pred_cats),
  truth, cats)

autoplot(roc_lr) +
  labs(x = "1 - Recall(Dogs)", y = "Recall(Cats)")
```

```r
roc_obj <- roc(pred_test$truth, pred_test$.pred_cats,
               levels = c("dogs", "cats"))

cutoffs <- c(0.3, 0.5, 0.9)

coords(roc_obj, x = cutoffs,
                ret = c("accuracy", "recall", "precision",
                        "specificity", "npv"),
                transpose = TRUE)
```

```
##                  [,1]      [,2]      [,3]
## accuracy    0.4318182 0.7270202 0.8512626
## recall      0.9310873 0.6753446 0.1041348
## precision   0.2161394 0.3366412 0.9444444
## specificity 0.3332325 0.7372241 0.9987904
## npv         0.9607672 0.9200000 0.8495370
```

What *is* the model?

```
tidy(mod_lr_test) %>%
  filter(near(penalty, 0.01, tol = 0.0005)) %>%
  arrange(-abs(estimate)) %>%
  select(term, estimate, penalty) %>%
  head(20)
```

```
## # A tibble: 20 x 3
##    term                 estimate penalty
##    <chr>                   <dbl>   <dbl>
##  1 w_my_cat               -2.13     0.01
##  2 w_my_dog                1.62     0.01
##  3 w_kitt                 -1.32     0.01
##  4 (Intercept)            -1.18     0.01
##  5 w_love_dog              0.879    0.01
##  6 religion_christian      0.817    0.01
##  7 w_cats                 -0.762    0.01
##  8 w_dog                   0.648    0.01
##  9 status_all_else        -0.558    0.01
## 10 w_pupp                  0.551    0.01
## 11 diet_all_else          -0.551    0.01
## 12 drugs_all_else         -0.540    0.01
## 13 body_type_all_else     -0.484    0.01
## 14 drugs_sometimes        -0.476    0.01
## 15 ethnicity_white        -0.459    0.01
```

What did we achieve?

- ✔ Reproducible model
- ✔ Dependable model
- ✔ Explainable model
- ✔ Small and Fast as hell model
- ✕ Impressive model

Is that the end?

No.

> 💡 What *didn't* we do?

Few Excellent Books