

# Pipelines

14 Mar 2023

## Table of contents

workflow systems . . . . .	1
R/python . . . . .	2
tidymodels . . . . .	2
parsnip . . . . .	2
rsample . . . . .	2
recipes . . . . .	3
more . . . . .	3
example . . . . .	3
build preprocessing recipe . . . . .	4
‘prep’ step . . . . .	5
‘bake’ step (and sampling) . . . . .	6
logistic regression . . . . .	7
digression: experimental design . . . . .	7
sanity check . . . . .	10
tangent: testing the <code>mn_log_loss</code> rule . . . . .	11
conclusions? . . . . .	11
Python . . . . .	11

## workflow systems

- want to *abstract* details of statistical modeling/machine learning
- benefits of abstraction
  - reduce cognitive load
  - shorter code

- costs of abstraction
  - more ‘magic’
  - learning another system
  - harder to dig down for details
  - loss of flexibility/harder to modify in ways not foreseen by designers

## R/python

Materials from [Modeling in R and Python](#)

- `tidymodels`: meta-package for ‘tidy’ modeling in R
- `scikit-learn`: modeling in Python

## tidymodels

### parsnip

- `parsnip` package (`CART` → `caret` → “carrot” → `parsnip`)
- unify modeling interfaces (`lm`, `glmnet`, `randomForest`, etc etc etc)
- specify **model** (algorithm), **mode** (classification or regression), **engine** (implementation/package)
- in principle (???)

### rsample

- resampling, cross-validation, bootstrapping, holdout sets
- ...
- train/test split (`initial_split()`/`training()`/`testing()`)
- cross-validation (`vfold_cv()`), bootstrap (`bootstrap()`)
- blocked/grouped methods! `group_vfold_cv`, `group_bootstraps()`

## recipes

- feature engineering
- preprocessing (centering/scaling, imputation, dimension reduction, etc.)

## more

- workflows: bundle preprocessing/modeling/post-processing
- tune: hyperparameter tuning
- yardstick: assessment

## example

```
library(tidyverse)
library(tidymodels)
library(glmnet)
source("../code/utils.R")

historical <- (read_csv("../code/historical_baseball.csv")
  ## should these be done in the 'prep' process?
  |> mutate(across(inducted, ~fct_rev(factor(.))))
  |> filter(ab > 250)
)
```

Rows: 3235 Columns: 15

-- Column specification -----

Delimiter: ","

chr (1): player\_id

dbl (14): inducted, g, ab, r, h, x2b, x3b, hr, rbi, sb, cs, bb, so, last\_year

i Use `spec()` to retrieve the full column specification for this data.

i Specify the column types or set `show\_col\_types = FALSE` to quiet this message.

```
data_split <- initial_split(historical, prop = 2/3, strata = inducted)
train_data <- training(data_split)
testing_data <- testing(data_split)
```

## build preprocessing recipe

- Basics here.
- Could also do PCA selection, collapse rare factor levels  
(`step_other()`, other filtering ... (see [recipes docs](#))

```
b_recipe <- (
  ## (? why does recipe need data?)
  recipe(inducted ~ ., data = train_data)
  ## no '-' operator in formulas
  ## could use e.g. all_numeric_predictors()
  |> step_rm("last_year")
  ## set player_id to be neither predictor or outcome
  |> update_role(player_id, new_role = "ID")
  ## center, scale, remove zero-variance variables
  |> step_center(all_numeric())
  |> step_scale(all_numeric())
  |> step_nzv(all_numeric())
)
print(b_recipe)
```

-- Recipe -----

-- Inputs

Number of variables by role

```
outcome:    1
predictor: 13
ID:         1
```

```
-- Operations
```

```
* Variables removed: "last_year"
```

```
* Centering for: all_numeric()
```

```
* Scaling for: all_numeric()
```

```
* Sparse, unbalanced variable filter on: all_numeric()
```

### **'prep' step**

- Set any *data-dependent* filtering steps based on the full training data set
- Avoid data leakage

```
b_prepped <- prep(b_recipe)
print(b_prepped)
```

```
-- Recipe -----
```

```
-- Inputs
```

```
Number of variables by role
```

```
outcome:    1
predictor: 13
ID:         1
```

```
-- Training information
```

Training data contained 1776 data points and no incomplete rows.

```
-- Operations
```

```
* Variables removed: last_year | Trained
```

```
* Centering for: g, ab, r, h, x2b, x3b, hr, rbi, sb, cs, bb, so | Trained
```

```
* Scaling for: g, ab, r, h, x2b, x3b, hr, rbi, sb, cs, bb, so | Trained
```

```
* Sparse, unbalanced variable filter removed: <none> | Trained
```

### **'bake' step (and sampling)**

- apply prep to new (maybe) data; sample
- can use `strata` to help balance data, and to avoid data leakage

```
b_prepped |> bake(train_data) |> rsample::vfold_cv(v=10)
```

```
# 10-fold cross-validation
# A tibble: 10 x 2
  splits          id
  <list>        <chr>
1 <split [1598/178]> Fold01
2 <split [1598/178]> Fold02
3 <split [1598/178]> Fold03
4 <split [1598/178]> Fold04
5 <split [1598/178]> Fold05
6 <split [1598/178]> Fold06
7 <split [1599/177]> Fold07
8 <split [1599/177]> Fold08
9 <split [1599/177]> Fold09
10 <split [1599/177]> Fold10
```

## logistic regression

```
lrc_mod <- (
  logistic_reg(mode = "classification",
               penalty = tune(),
               mixture = tune())
  |> set_engine(engine = "glmnet")
)
print(lrc_mod)
```

Logistic Regression Model Specification (classification)

Main Arguments:

```
penalty = tune()
mixture = tune()
```

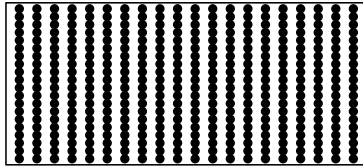
Computational engine: glmnet

## digression: experimental design

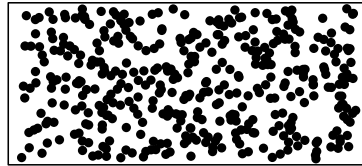
- sample over a multidimensional space?
- grids (easy, inflexible)
- random samples (too clustered)

- space-filling
  - Latin hypercube
  - *Sobol sequences*

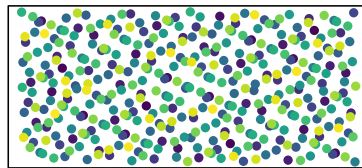
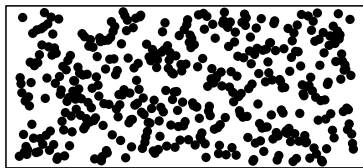
grid



random



Latin hypercube quasirandom (Sobo



Set up parallel processing (foreach package)

```
doParallel::registerDoParallel(cores = 4)

system.time(tt <- tune_grid(
  grid = 100,
  object = lrc_mod,
  preprocessor = b_prepped,
  resamples = vfold_cv(train_data),
  ## decided to use deviance rather than accuracy to train
  metrics = metric_set(mn_log_loss)
  ## , control = control_grid(verbose = TRUE)
))
```

```
user system elapsed
334.612 1.140 102.467
```

```
saveRDS(tt, "tune_grid.rds")
```

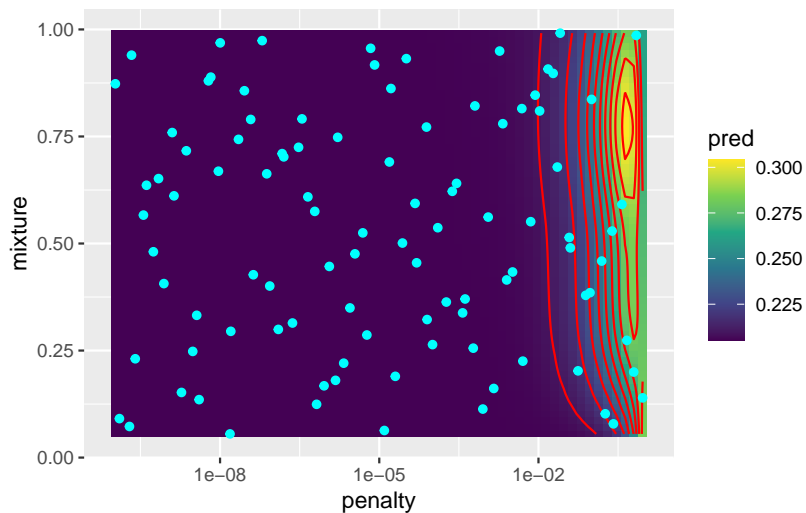
[obligatory xkcd](#)



```

## tt <- readRDS("tune_grid.rds")
cc <- collect_metrics(tt)
## not actually a good design for this case ...
gg0 <- ggplot(cc, aes(penalty, mixture)) + geom_point() + scale_x_log10()
## make regular grid for plotting
dd <- with(cc,
          expand.grid(penalty = sfun(penalty, TRUE),
                     mixture = sfun(mixture))
        )
m1 <- mgcv::gam(mean ~ te(penalty, mixture), data = cc)
dd$pred <- as.numeric(predict(m1, newdata = dd))
gg1 <- ggplot(dd, aes(penalty, mixture)) +
  scale_fill_viridis_c() +
  geom_tile(aes(fill = pred)) +
  geom_contour(aes(z = pred), colour = "red") +
  scale_x_log10() +
  geom_point(data = cc, colour = "cyan")
print(gg1)

```



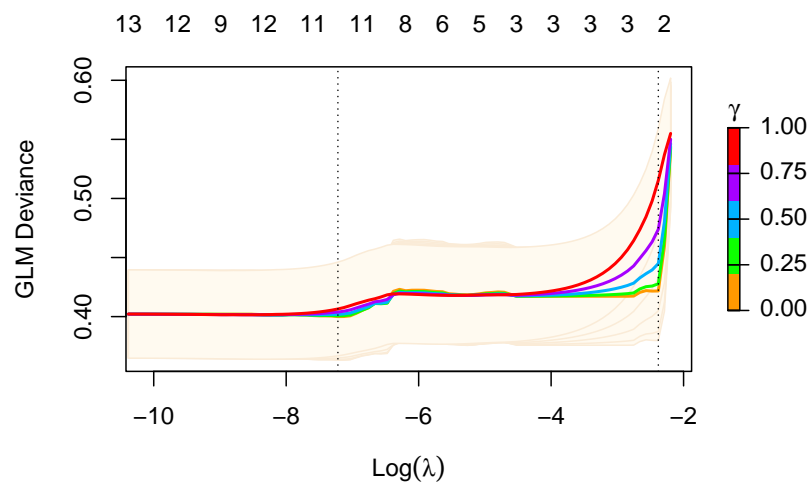
## sanity check

```
system.time(  
  c1 <- cv.glmnet(y = train_data$inducted,  
    x = model.matrix(~ . - inducted -player_id, train_data),  
    family = binomial(),  
    relax = TRUE,  
    data = train_data,  
    parallel = TRUE)  
)
```

```
user  system elapsed  
9.721   0.863   3.995
```

(Why so much faster?? Warm start etc. ...)

```
plot(c1)
```



## tangent: testing the mn\_log\_loss rule

```
x <- factor(1:4)
xmat <- diag(4)
p <- matrix(rgamma(16, shape = 1, scale = 1), 4, 4)
p <- sweep(p, MARGIN=1, STATS=rowSums(p), FUN ="/")
stopifnot(all.equal(rowSums(p), rep(1,4)))
yardstick::mn_log_loss_vec(truth = x, estimate = p)
```

```
[1] 1.42576
```

```
-1 * mean(sapply(1:nrow(p),
  \(i) dmultinom(x = xmat[i,], size = 1, prob = p[i,], log = TRUE)))
```

```
[1] 1.42576
```

## conclusions?

More to do:

- find best model
- show variable importance?
- predictions (with confidence intervals?)
- partial dependence profiles?

```
show_best(tt)
model_profile(select_by_one_std_err(tt, "penalty"))
```

## Python

initial

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
```

```

historical = pd.read_csv("../code/historical_baseball.csv").query("ab>250")

## index var
historical_pidindex = historical.set_index('player_id')
X = historical_pidindex.drop(['inducted', 'last_year'], axis = 1)
y = historical_pidindex.inducted
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0, test_size = 1/3)

from sklearn import preprocessing
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LogisticRegressionCV
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import VarianceThreshold

pipe_scale_lr_lasso = make_pipeline(StandardScaler(),
                                    VarianceThreshold(),
                                    LogisticRegressionCV(Cs = 10,
                                                         penalty = "elasticnet",
                                                         solver = "saga",
                                                         scoring = "neg_log_loss",
                                                         l1_ratios = np.linspace(0, 1, 6),
                                                         cv = 10,
                                                         max_iter = 2000,
                                                         n_jobs = 4))

pipe_scale_lr_lasso.fit(X_train, y_train) # apply scaling on training data

Pipeline(steps=[('standardscaler', StandardScaler()),
                ('variancethreshold', VarianceThreshold()),
                ('logisticregressioncv',
                 LogisticRegressionCV(cv=10,
                                     l1_ratios=array([0. , 0.2, 0.4, 0.6, 0.8, 1. ]),
                                     max_iter=2000, n_jobs=4,
                                     penalty='elasticnet',
                                     scoring='neg_log_loss', solver='saga'))])

```

```

coefs = pipe_scale_lr_lasso.named_steps['logisticregressioncv'].coef_
coef_summary = pd.DataFrame(coefs.transpose(), columns = ['coefs'], index = X_train.columns)
print(coef_summary)

```

```

      coefs
g      2.407943
ab     -4.756825
r       1.834519
h       1.998031
x2b    -0.384117
x3b    -0.023799
hr      0.031622
rbi     0.651412
sb      0.040147
cs     -0.246444
bb     -0.501351
so     -0.005352

```

```

pipe_scale_lr_lasso.score(X_test, y_test)

```

```

-0.22022919720875345

```

**to do:**

- selected penalty, mixture parameters? (expect v. unstable)
- predictions?
- uncertainty of predictions?