

Pipelines

5 Mar 2023

Table of contents

workflow systems	1
R/python	2
tidymodels	2
parsnip	2
rsample	2
recipes	2
more	3
example	3
build preprocessing recipe	3
‘prep’ step	5
‘bake’ step (and sampling)	6
logistic regression	7
digression: experimental design	7
sanity check	9
tangent: testing the <code>mn_log_loss</code> rule	10

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)

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:

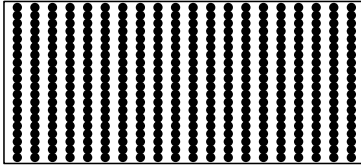
```
penalty = tune()  
mixture = tune()
```

Computational engine: glmnet

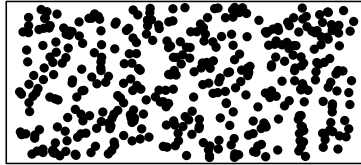
digression: experimental design

- sample over a multidimensional space?
- grids (easy, inflexible)
- random samples (too clustered)
- **space-filling**
 - Latin hypercube
 - *Sobol sequences*

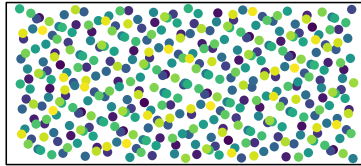
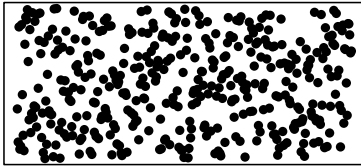
grid



random



Latin hypercubequasirandom (Sobo



```
doMC::registerDoMC(cores = 4)
```

```
tt <- tune_grid(
  grid = 100,
  object = lrc_mod,
  preprocessor = b_prepped,
  resamples = vfold_cv(train_data),
  metrics = metric_set(mn_log_loss)
  ## , control = control_grid(verbose = TRUE)
)
saveRDS(tt, "tune_grid.rds")
```

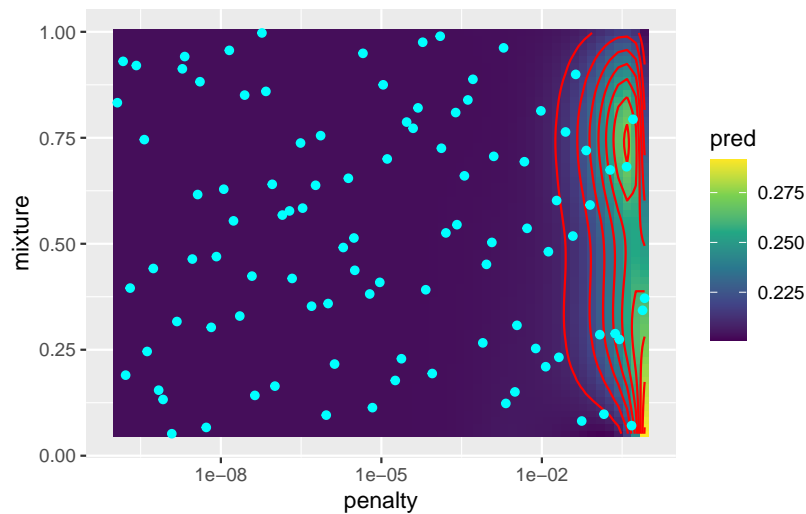
```
cc <- collect_metrics(tt)
gg0 <- ggplot(cc, aes(penalty, mixture)) + geom_point() + scale_x_log10()
sfun <- function(x, log = FALSE, n = 61) {
  if (log) x <- log(x)
  s <- seq(min(x), max(x), length.out = n)
  if (log) exp(s) else s
}
dd <- with(cc,
  expand.grid(penalty = sfun(penalty, TRUE),
             mixture = sfun(mixture))
)
if (FALSE) gg0 %+% dd
```



```

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)

```



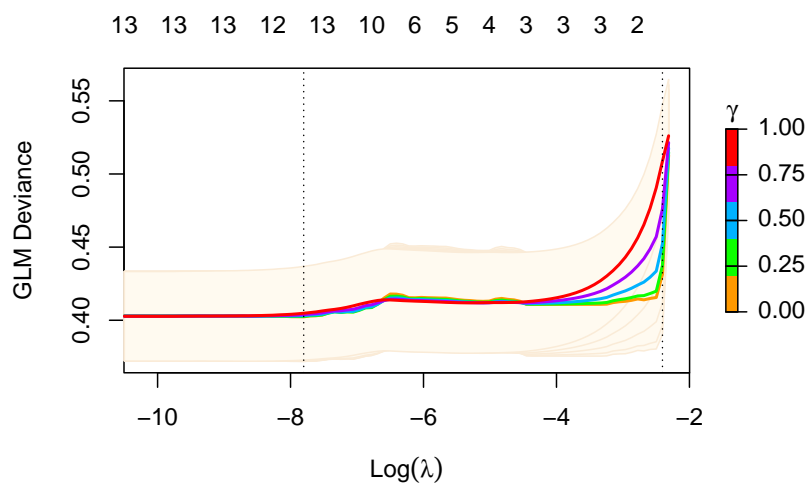
sanity check

```

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

```

```
plot(c1)
```



tangent: testing the `mn_log_loss` rule

```
data(iris)
svm.model <- e1071::svm(Species ~ ., data = iris, probability = TRUE)
pred <- predict(svm.model, iris, probability = TRUE)
prob <- attr(pred, "probabilities")
spmat <- matrix(0, ncol = length(levels(iris$Species)),
               nrow = length(iris$Species))
spmat[cbind(1:nrow(prob), as.numeric(iris$Species))] <- 1
-1 * mean(sapply(1:nrow(prob),
                 \ (i) dmultinom(x = spmat[i,], size = 1, prob[i,], log = TRUE))))
```

```
[1] 0.08061148
```

```
yardstick::mn_log_loss_vec(truth = iris$Species, estimate = prob)
```

```
[1] 0.08061148
```