Aprendizaje Estadístico Supervisado

Natalia da Silva

2024

Muchos flujos de trabajos en uno

- Esta clase se base en el capítulo 7 (A Model Wrokflow), capítulo 11 (Comparing Models with Resampling) y capítulo 15 (Screening many models)
- Ya hemos trabajado con `workflow()` para ajustar modelos en `tidymodels`.
- `workflow` encapsula las piezas principales del proceso de modelado.

Importante:

- Porque nos permite trabajar con una buena metodología ya que es un solo punto de entrada para estimar los componentes del análisis de datos
- Permite a los usuarios organizar mejor el proyecto

Ejemplo: Datos Ames

Particiono los datos

```
library(modeldata)
data(ames)
library(tidymodels)
tidymodels_prefer()

set.seed(501)

# Save the split information for an 80/20 split of the dat
ames_split <- initial_split(ames, prop = 0.80)
ames_split
```

```
<Training/Testing/Total>
<2344/586/2930>
```

```
ames_train <- training(ames_split)
ames_test  <-  testing(ames_split)
```

Ejemplo: Datos Ames

Bosque

```
lm_model <-
  linear_reg() %>%
  set_engine("lm")

lm_wflow <-
  workflow() %>%
  add_model(lm_model) |>
  add_formula(Sale_Price ~ Longitude + Latitude)


lm_fit <- fit(lm_wflow, ames_train)
lm_fit
```

```
== Workflow [trained]
==========================================================================
Preprocessor: Formula
Model: linear_reg()
```

```
── Preprocessor ────────────────────────────────────────────────────────────────
Sale_Price ~ Longitude + Latitude

── Model ───────────────────────────────────────────────────────────────────────
```

```
1 predict(lm_fit, ames_test %>% slice(1:3))
```

```
# A tibble: 3 × 1
    .pred
    <dbl>
1 210673.
2 207465.
3 207175.
```

Creando muchos workflows en uno

En muchas situaciones los datos requieren varios intentos para encontrar el modelo apropiado. Por ejemplo:

- Modelos predictivos, queremos evaluar una variedad de distintos tipos de modelos. Esto requiere crear multiples especificaciones de modelos.
- El testeo secuencial de modelos tipicamente comienza expandiendo el conjunto de predictores. Comienzo con un modelo completo y genero un secuencia de modelos que va sacando predictores.

Este proceso se puede hacer con conjuntos de workflow, usamos el paquete `workflowset`

Creando muchos workflows en uno

Ejemplo: queremos enfocarnos en diferentes formas que la locación de la casa es representada en los datos de Ames. Creamos un conjunto de fórmulas que capturan estos predictores:

```
location <- list(
  longitude = Sale_Price ~ Longitude,
  latitude = Sale_Price ~ Latitude,
  coords = Sale_Price ~ Longitude + Latitude,
  neighborhood = Sale_Price ~ Neighborhood
)
```

Creando muchos workflows en uno

Estas representaciones pueden ser cruzadas con una o más representaciones usando la función `workflow_set()`. Usamos solo la especificacion del modelo lineal anterior

- `preproc` lista con objetos de preprocesamiento, formulas, recetas o `workflow_variables()`
- `models` lista o o modelo especificado con `parsnip`

```
library(workflowsets)
location_models <- workflow_set(preproc = location, models
location_models
```

```
# A workflow set/tibble: 4 × 4
  wflow_id        info             option    result
  <chr>           <list>           <list>    <list>
1 longitude_lm    <tibble [1 × 4]> <opts[0]> <list [0]>
2 latitude_lm     <tibble [1 × 4]> <opts[0]> <list [0]>
```

Creando muchos workflows en uno

Ajustamos un modelo para cada fórmula y lo salvamos en una nueva columna llamada fit. Acá se usan operaciones básicas de `dplyr` y `purrr`

`map` aplica una función a cada elemento de un vector

```
location_models <-
   location_models %>%
   mutate(fit = map(info, ~ fit(.x$workflow[[1]], ames_tra
location_models
```

```
# A workflow set/tibble: 4 × 5
  wflow_id         info             option    result      fit
  <chr>            <list>           <list>    <list>      <list>
1 longitude_lm     <tibble [1 × 4]> <opts[0]> <list [0]> <workflow>
2 latitude_lm      <tibble [1 × 4]> <opts[0]> <list [0]> <workflow>
3 coords_lm        <tibble [1 × 4]> <opts[0]> <list [0]> <workflow>
4 neighborhood_lm <tibble [1 × 4]> <opts[0]> <list [0]> <workflow>
```

```
location_models$fit[[1]]
```

```
== Workflow [trained] ==========================================================
Preprocessor: Formula
Model: linear_reg()

-- Preprocessor ----------------------------------------------------------------
Sale_Price ~ Longitude

-- Model -----------------------------------------------------------------------
```

Evaluando con el conjunto tes

Cuando terminamos el desarrollo del modelo y queremos el modelo final. Podemos usar convenientemente la función `last_fit()` que ajusta el modelo al conjunto de entrenamiento entero y lo evalua en el conjunto test.

usando `lm_wflow`, podemos pasarle al modelo la particion inicial en training/test

```
1 final_lm_res <- last_fit(lm_wflow, ames_split)
2 final_lm_res
```

```
# Resampling results
# Manual resampling
# A tibble: 1 × 6
  splits               id               .metrics .notes   .predictions
.workflow
  <list>               <chr>            <list>   <list>   <list>
<list>
1 <split [2344/586]> train/test split <tibble> <tibble> <tibble>
<workflow>
```

workflow results

```
fitted_lm_wflow <- extract_workflow(final_lm_res)
```

metricas de performance

```
collect_metrics(final_lm_res)
```

```
# A tibble: 2 × 4
  .metric .estimator .estimate .config
  <chr>   <chr>          <dbl> <chr>
1 rmse    standard   74464.    Preprocessor1_Model1
2 rsq     standard       0.152 Preprocessor1_Model1
```

```
collect_predictions(final_lm_res) %>% slice(1:5)
```

```
# A tibble: 5 × 5
    .pred id                .row Sale_Price .config
    <dbl> <chr>            <int>      <int> <chr>
1 210673. train/test split     7     213500 Preprocessor1_Model1
2 207465. train/test split     9     236500 Preprocessor1_Model1
3 207175. train/test split    11     175900 Preprocessor1_Model1
4 193826. train/test split    25     149900 Preprocessor1_Model1
5 192687. train/test split    27     126000 Preprocessor1_Model1
```

Comparando modelos con remuestreo

- Una vez que uno crea dos o más modelos, los queremos comparar para ver cuál es el mejor.
- En algunos casos la comparación es dentro del mismo modelo pero en otros pueden usar distintas variables o preprocesamieto
- También puedo querer comparar entre distintos tipos de modelos, regresión lineal, rf, etc.
- En cualquier caso los resultados son en base a medidas de resumen basadas en remuesteo para cada modelo.

Comparando modelos con remuestreo

- Veremos como workflow set puede ser usado para ajustar muchos modelos
- Veremos que preprocesamiento y o modelos pueden ser generados combinadamente
- Ch 10, se usan recetas para incluir interacciones y splines para longitude y latitude.
- Ejemplo de workflow sets, se crean tres modelos lineales distintos que agregan estos pasos de preprocesamiento incrementalmente
- Vemos si estos términos adicionales mejoran el resultado de los modelos
- Se crean tres recetas y se combinan en un workflow set

Comparando modelos con remuestreo

Ejemplo Ch 11

```
basic_rec <-
  recipe(Sale_Price ~ Neighborhood + Gr_Liv_Area + Year_Bu
         Latitude + Longitude, data = ames_train) %>%
  step_log(Gr_Liv_Area, base = 10) %>%
  step_other(Neighborhood, threshold = 0.01) %>%
  step_dummy(all_nominal_predictors())

interaction_rec <-
  basic_rec %>%
  step_interact( ~ Gr_Liv_Area:starts_with("Bldg_Type_") )

spline_rec <-
  interaction_rec %>%
  step_ns(Latitude, Longitude, deg_free = 50)

```

```
# A workflow set/tibble: 3 × 4
  wflow_id    info             option    result
```

```
  <chr>       <list>           <list>    <list>
1 basic_lm    <tibble [1 × 4]> <opts[0]> <list [0]>
2 interact_lm <tibble [1 × 4]> <opts[0]> <list [0]>
3 splines_lm  <tibble [1 × 4]> <opts[0]> <list [0]>
```

Comparando modelos con remuestreo

Nos gustaría remuestrear cada uno de estos modelos, para ello se usa la función `workflow_map()`. Esta toma un argumento inicial para aplicar al workflows, seguido de opciones a la misma

Comparando modelos con remuestreo

```
set.seed(1001)
ames_folds <- vfold_cv(ames_train, v = 10)
ames_folds
```

```
#  10-fold cross-validation
# A tibble: 10 × 2
   splits             id
   <list>             <chr>
 1 <split [2109/235]> Fold01
 2 <split [2109/235]> Fold02
 3 <split [2109/235]> Fold03
 4 <split [2109/235]> Fold04
 5 <split [2110/234]> Fold05
 6 <split [2110/234]> Fold06
 7 <split [2110/234]> Fold07
 8 <split [2110/234]> Fold08
```

Comparando modelos con remuestreo

```
keep_pred <- control_resamples(save_pred = TRUE, save_work


lm_models <-
  lm_models %>%
  workflow_map("fit_resamples",
               # Options to `workflow_map()`:
               seed = 1101, verbose = TRUE,
               # Options to `fit_resamples()`:
               resamples = ames_folds, control = keep_pred
collect_metrics(lm_models) %>%
  filter(.metric == "rmse")
```

```
# A tibble: 3 × 9
  wflow_id    .config      preproc model .metric .estimator   mean
n std_err
  <chr>       <chr>        <chr>   <chr> <chr>   <chr>       <dbl>
<int>   <dbl>
1 basic_lm    Preprocesso… recipe  line… rmse    standard   38814.
```

```
10   1321.
2 interact_lm Preprocesso… recipe  line… rmse    standard   38476.
10   1286.
3 splines_lm  Preprocesso… recipe  line… rmse    standard   36487.
10   1171.
```

Comparando modelos con remuestreo

Podemos agregar un modelo adicional al conjunto convirtiendolo en un workflow propio y luego agregar las filas. Es necesiario que cuando el modelos fue remuestreado `save_workflow = TRUE`

```
rf_model <-
  rand_forest(trees = 1000) %>%
  set_engine("ranger") %>%
  set_mode("regression")

rf_wflow <-
  workflow() %>%
  add_formula(
    Sale_Price ~ Neighborhood + Gr_Liv_Area + Year_Built +
      Latitude + Longitude) %>%
  add_model(rf_model)

rf_fit <- rf_wflow %>% fit(data = ames_train)
rf_res <-
  rf_wflow %>%
```

```
# Resampling results
# 10-fold cross-validation
# A tibble: 10 × 5
   splits             id     .metrics          .notes
.predictions
   <list>             <chr>  <list>            <list>            <list>
 1 <split [2109/235]> Fold01 <tibble [2 × 4]> <tibble [0 × 3]>
<tibble>
 2 <split [2109/235]> Fold02 <tibble [2 × 4]> <tibble [0 × 3]>
<tibble>
 3 <split [2109/235]> Fold03 <tibble [2 × 4]> <tibble [0 × 3]>
<tibble>
```

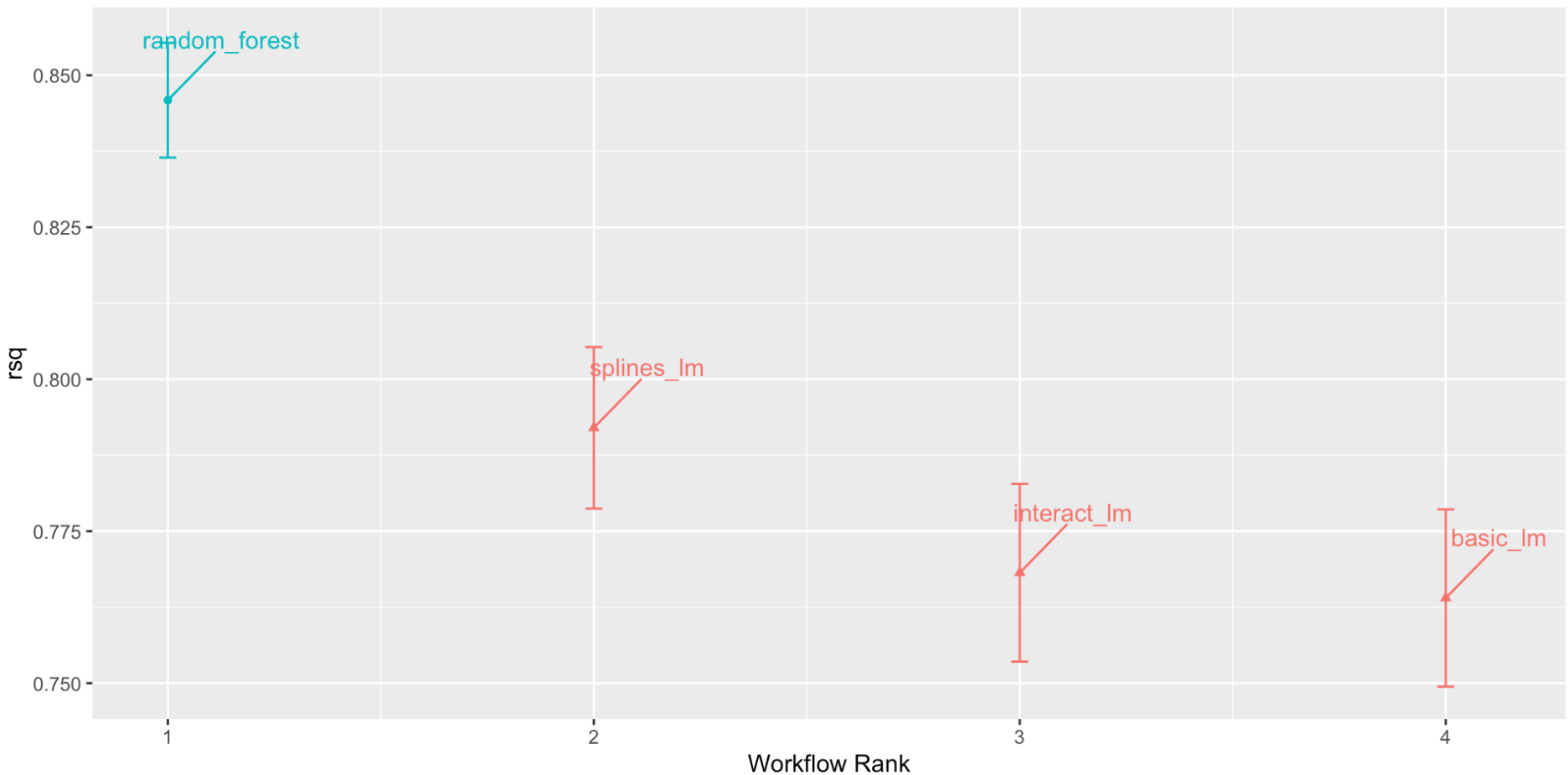
Comparando modelos con remuestreo

```r
four_models <-
  as_workflow_set(random_forest = rf_res) %>%
  bind_rows(lm_models)
four_models
```

```
# A workflow set/tibble: 4 × 4
  wflow_id      info             option    result
  <chr>         <list>           <list>    <list>
1 random_forest <tibble [1 × 4]> <opts[0]> <rsmp[+]>
2 basic_lm      <tibble [1 × 4]> <opts[2]> <rsmp[+]>
3 interact_lm   <tibble [1 × 4]> <opts[2]> <rsmp[+]>
4 splines_lm    <tibble [1 × 4]> <opts[2]> <rsmp[+]>
```

Comparando modelos con remuestreo

```
1 library(ggrepel)
2 autoplot(four_models, metric = "rsq") +
3   geom_text_repel(aes(label = wflow_id), nudge_x = 1/8, nu
```



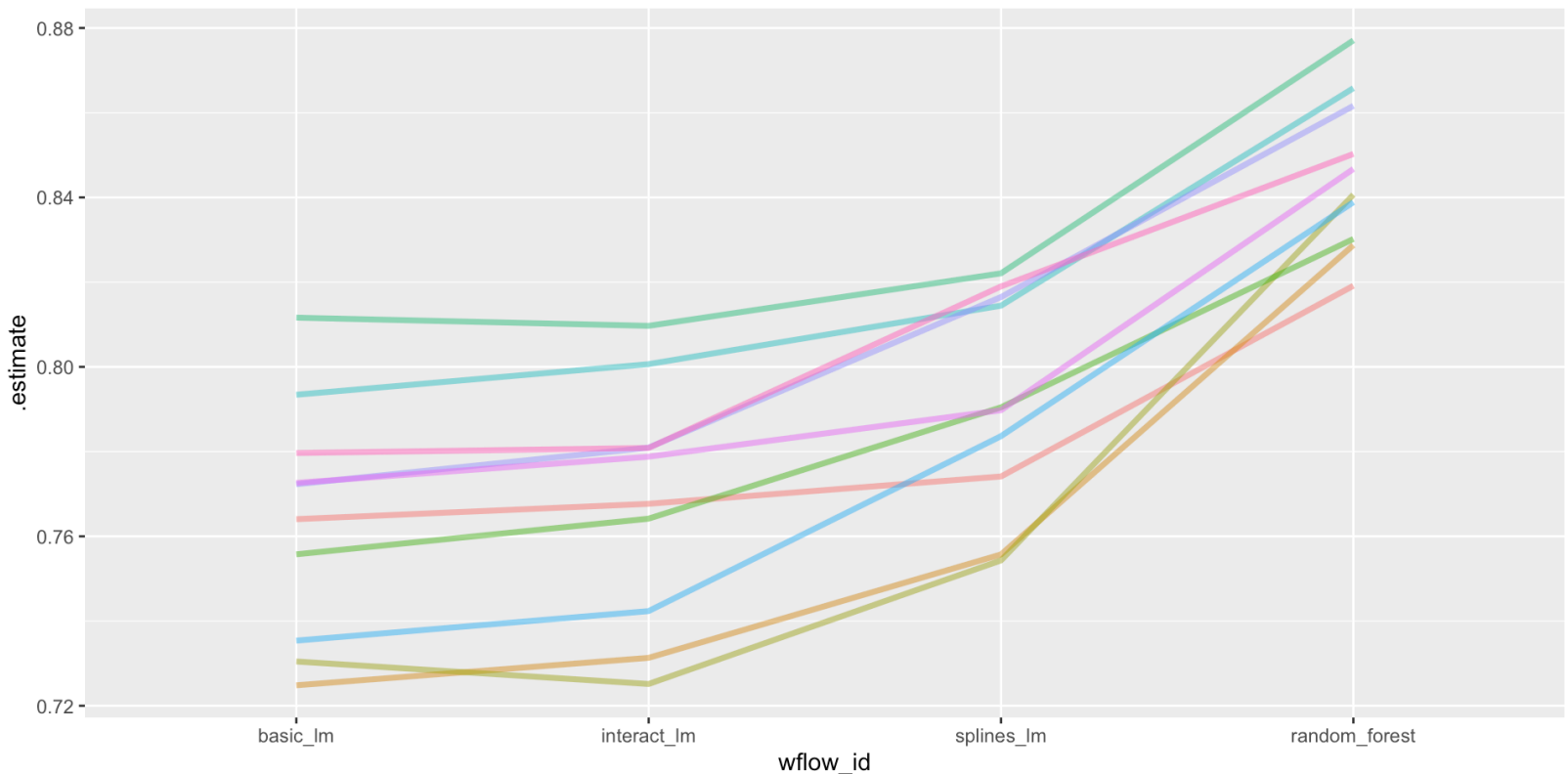
Comparando modelos con remuestreo

```
rsq_indiv_estimates <-
  collect_metrics(four_models, summarize = FALSE) %>%
  filter(.metric == "rsq")

rsq_wider <-
  rsq_indiv_estimates %>%
  select(wflow_id, .estimate, id) %>%
  pivot_wider(id_cols = "id", names_from = "wflow_id", val

corrr::correlate(rsq_wider %>% select(-id), quiet = TRUE)
```

```
# A tibble: 4 × 5
  term          random_forest basic_lm interact_lm splines_lm
  <chr>                 <dbl>    <dbl>       <dbl>      <dbl>
1 random_forest        NA        0.754       0.723      0.783
2 basic_lm              0.754   NA           0.986      0.875
3 interact_lm           0.723    0.986      NA          0.898
4 splines_lm            0.783    0.875       0.898     NA
```

Comparando modelos con remuestreo

```
rsq_indiv_estimates %>%
  mutate(wflow_id = reorder(wflow_id, .estimate)) %>%
  ggplot(aes(x = wflow_id, y = .estimate, group = id, colo
  geom_line(alpha = .5, linewidth = 1.25) +
  theme(legend.position = "none")
```



Ch 15

Miramos el ejemplo donde se ajustan y tunean muchos modelos.

Hacer esto en el proyecto!