Notas de `tidymodels`

Parte 1: `recipes`

Emmanuel Alcalá

28 febrero, 2022

Capítulo 8 de `tmwr`

Preprocesar datos incluye:

- Manejar los valores perdidos (remover o rellenar con imputación).
- Normalizar o escalar. Modelos que asignan importancia (o peso) a un predictor por su escala pueden crear problemas numéricos.
- Transformaciones, como `log`-transformaciones, para cambiar la forma de una distribución (e.g., par adistrbuciones sesgadas a la derecha).
- Remover predictores redundantes o que producen multicolinealidades en los modelos.

Anteriores actividades suelen ser agrupadas en la llamada ingeniería de factores (*feature engineering*).

Funciones de `recipes`

`recipe()`

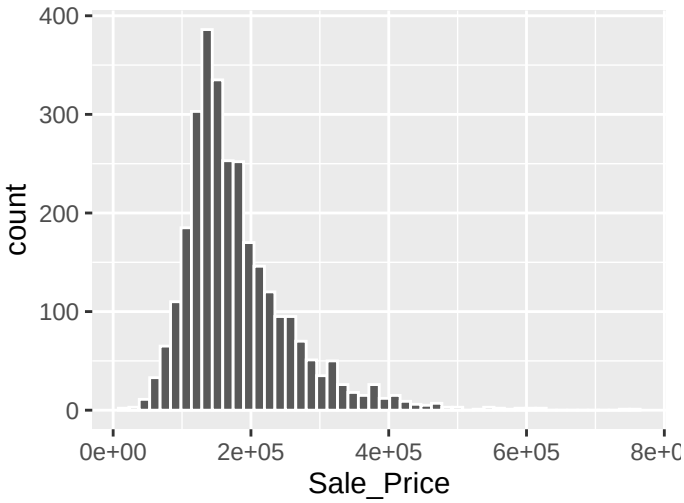
Según la documentación: "Define operaciones en los datos y sus roles asociados". Preprocesar incluye reducir la cantidad de predictores (por ejemplo, si existe correlación entre ellos) mediante extracción de factores (*feature extraction*). También, mediante métodos de imputación, corregir la presencia de valores peridos. Otra es estimar una transformación de los datos originales, en vez de los originales, si alguna característica (e.g., simetría) es necesaria o facilita el modelamiento.

Ames housing data Se carga junto con `tidymodels`. Para ver en qué consiste el conjunto de datos, correr `?ames`.

```
data(ames)
# evitar conflictos de funciones
tidymodels_prefer()

ggplot(ames, aes(x = Sale_Price)) +
  geom_histogram(bins = 50, col = "white")
```

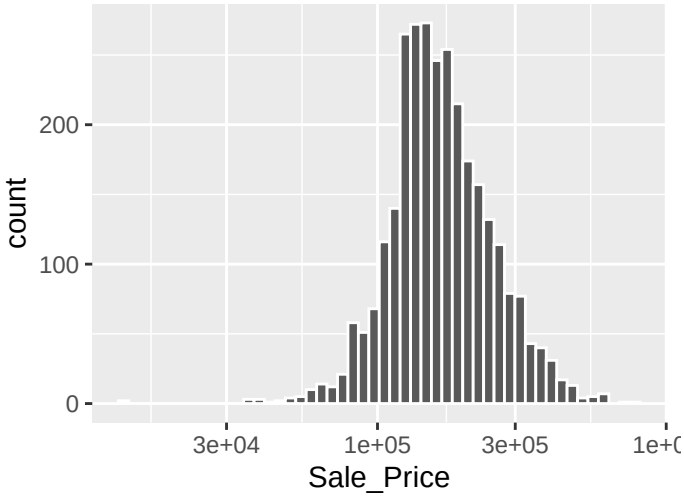
Explorar



Los datos muestran que hay más casas baratas que casas caras. Como se trata de una distribución sesgada a la derecha, una estrategia es log-transformar. Esto asegura que las casas con precios altos (que tienen una baja frecuencia) no tengan un peso elevado. Además, métodos que usan distancias (e.g., euclideana o L_2) requieren que los predictores estén en las mismas unidades.

```
data(ames)
tidymodels_prefer()

ggplot(ames, aes(x = Sale_Price)) +
  geom_histogram(bins = 50, col = "white") +
  scale_x_log10()
```




```
# log transformar Sale_Price
ames <- ames %>% mutate(Sale_Price = log10(Sale_Price))
```

Predictores

- Vecindario ($n = 29$)
- La porción de la casa por encima del piso (*gross above-grade*), `Gr_Liv_Area`.
- Año de construcción (`Year_Built`).
- Tipo de construcción (`Bldg_Type` con niveles `OneFam, TwoFmCon, Duplex, Twnhs y TwnhsE`).

```
lm_ames <- lm(
  Sale_Price ~ Neighborhood + log10(Gr_Liv_Area) + Year_Built + Bldg_Type,
```

```
  data = ames
)
```

Matemáticamente:

$$\begin{aligned}\text{Sale Price} = \beta_{\text{Neighborhood}}\text{Neighborhood} + \\ \beta_{\text{GrLivArea}}\log_{10}(\text{GrLivArea}) + \\ \beta_{\text{YearBuilt}}\text{YearBuilt} + \\ \beta_{\text{BldgType}}\text{BldgType}\end{aligned}$$

Nota: los predictrores cualitativos se descomponen en sus niveles, por lo que realmente tendremos un β_{BldgType} por nivel.

Al usar `lm(y~x)` el `data.frame` se convierte en una matriz de diseño. Con `recipes` hacemos una receta que consista en los pasos para el procesamiento de datos (de ahí el nombre). Es una especificación, no se ejecutan.

También usaremos `workflow`. Un flujo de trabajo debe incluir el preprocesamiento, el modelamiento y cualquier otro proceso post-modelado (por ejemplo, extraer coeficientes de un modelo de regresión). La función `workflow` permite precisamente eso.

```
# Usar funciones initial_split, training y testing de rsamples
ames_split <- initial_split(ames,
  # dividir en prop 80/20
  prop = 0.80,
  # estratigicar por cuartil
  strata = Sale_Price
)
# split for training, 80 % training and 20 % testing
ames_train <- training(ames_split)
ames_test <- testing(ames_split)
# make a recipe
simple_ames <- recipe(
  # formula
  Sale_Price ~ Neighborhood + Gr_Liv_Area + Year_Built + Bldg_Type,
  data = ames_train
) %>%
  # transformar Gr_Liv_Area a log10
  step_log(Gr_Liv_Area, base = 10) %>%
  # usar dummy para los predictores no numericos (cualitativos)
  # y los convierte a dummy
  step_dummy(all_nominal_predictors())
# declarar un modelo lineal
lm_model <- linear_reg() %>%
  set_engine("lm")
# definir un workflow.
lm_wflow <- workflow() %>%
  add_model(lm_model) %>%
  add_recipe(simple_ames)
```

Correr el modelo

```
# usar fit de parsnip (generics)
lm_fit <- fit(lm_wflow, ames_train)
lm_fit %>%
```

Table 1: Codificación binaria para un un predictor cualitativo

Raw Data	TwoFmCon	Duplex	Twnhs	TwnhsE
OneFam	0	0	0	0
TwoFmCon	1	0	0	0
Duplex	0	1	0	0
Twnhs	0	0	1	0
TwnhsE	0	0	0	1

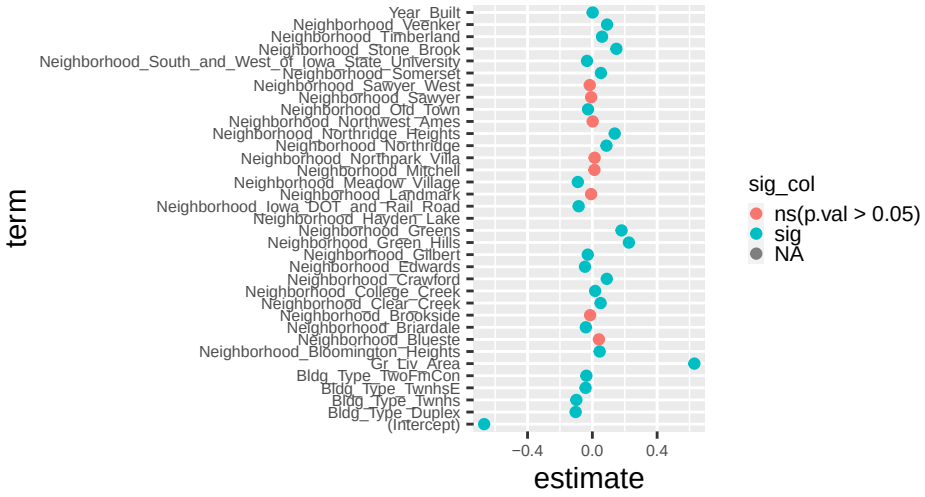
```
  # retorna modelo ajustado
  extract_fit_parsnip() %>%
  # produce una tabla tidy del modelo
  tidy() %>%
  # mostrar primeras 5 filas
  slice(1:5)
```

```
## # A tibble: 5 x 5
##   term                        estimate std.error statistic  p.value
##   <chr>                          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)                -0.668    0.227        -2.94 3.28e- 3
## 2 Gr_Liv_Area                 0.630    0.0136       46.4  0
## 3 Year_Built                  0.00198  0.000115     17.2  8.31e-63
## 4 Neighborhood_College_Creek  0.0175   0.00804       2.18 2.95e- 2
## 5 Neighborhood_Old_Town      -0.0265   0.00820      -3.23 1.26e- 3
```

Graficar los estimadores

```
tidy_fit <-
  lm_fit %>%
  extract_fit_parsnip() %>%
  tidy() %>%
  mutate(sig_col = ifelse(
    p.value >= 0.05, "ns(p.val > 0.05)", "sig"
  ))

ggplot(
  tidy_fit,
  aes(
    x = term,
    y = estimate,
    color = sig_col
  )
) +
  geom_point() +
  coord_flip() +
  theme(
    axis.text = element_text(size = 6),
    legend.key.size = unit(0.2, "cm"),
    legend.key.height = unit(0.2, "cm"),
    legend.key.width = unit(0.2, "cm"),
    legend.title = element_text(size = 8),
    legend.text = element_text(size = 8)
  )
```



Para evaluar el modelo, tenemos que usar `predict` con un nuevo conjunto de datos, que en este caso son `ames_test`. A esta estrategia se le llama *validación empírica*: usar el conjunto de datos no usado en *entrenamiento* para evaluar su efectividad.

Para esto, debemos elegir una métrica (el paquete `yardstick` tiene diferentes métricas para evaluar modelos). Por ejemplo, RMSE mide la precisión, mientras que R^2 mide correlación entre dos variables. optimizar una u otra debe hacerse para diferentes propósitos. Usar RMSE produce resultados variables pero con precisión uniforme en el rango de valores, mientras que R^2 produce una mayor relación lineal entre observados y predichos.