Predictive Pipeline

DS-6030 | Spring 2026

pred-pipe.pdf

Table of contents

1 Linear Models

1.1 Linear Regression

- Linear regression refer to a class of models where the output (predicted value) is a linear combination (weighted sum) of the input variables

$$f(x;\beta) = \beta_0 + \sum_{j=1}^{p} \beta_j x_j$$

where $x = [x_1, \ldots, x_p]^\mathsf{T}$ is a vector of features/variables/attributes and $\hat{Y}|x = f(x;\hat{\beta}) = x^\mathsf{T}\hat{\beta}$ is the predicted outcome at $X = x$.

- the *model parameters* for linear (or coefficients or weights), $\hat{\beta}$ determine how much influence each feature has on the predicted output.

1.1.1 Model Structure

- Outcome variable: $y \in \mathbb{R}$
- Predictor variables $\mathbf{x} \in \mathbb{R}^p$
- Model parameters: $\beta = (\beta_0, \beta_1, \ldots \beta_p)$
- Prediction function: $f(\mathbf{x};\hat{\beta}) = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_1 x_2 + \ldots + \hat{\beta}_1 x_p = \mathbf{x}^\mathsf{T}\beta$

1.1.2 Model Fitting (or Parameter estimation)

- Use *training data*: $D_{\text{train}} = \{(x_i, y_i)\}_{i=1}^n$ to fit the model (i.e., estimate the model parameters).
- The model parameters are often selected by minimizing the sum of squared residuals of the *training data* (as called *ordinary least squares (OLS)*).
  - But, there are other, and better, ways to estimate the parameters in linear regression (e.g., Lasso, Ridge, Elastic Net, Robust, Principal Components).
- Ordinary least squares (OLS) chooses the weights/coefficients that minimize the mean squared error (MSE) loss function over the training data

$$\begin{aligned}
\hat{\beta} &= \arg\min_{\beta} \ \text{MSE}(\beta) \qquad \text{Note: } \beta \text{ is a } \textit{vector} \\
&= \frac{1}{n} \arg\min_{\beta} \ \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i;\beta))^2 \\
&= \frac{1}{n} \arg\min_{\beta} \ \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_{i1} - \beta_2 x_{i2} + \ldots + \beta_p x_{ip})^2
\end{aligned}$$

$$\begin{aligned}
\text{MSE}(\beta) &= \frac{1}{n}\text{SSE}(\beta) \\
\text{RMSE} &= \sqrt{\text{MSE}} \\
&= \sqrt{\text{SSE}}/\sqrt{n}
\end{aligned}$$

1.1.3 Matrix notation

$$f(\mathbf{x};\beta) = \mathbf{x}^\mathsf{T}\beta$$

$$Y = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix} \qquad X = \begin{bmatrix} 1 & X_{11} & X_{12} & X_{13} & \dots & X_{1p} \\ 1 & X_{21} & X_{22} & X_{23} & \dots & X_{2p} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & X_{n1} & X_{n2} & X_{n3} & \dots & X_{np} \end{bmatrix} \qquad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix}$$

$$\text{SSE}(\beta) = (Y - X\beta)^\mathsf{T}(Y - X\beta)$$

$$\begin{aligned} \frac{\partial \text{SSE}(\beta)}{\partial \beta} &= 2X^\mathsf{T}(Y - X\beta) \\ &\implies X^\mathsf{T}Y = X^\mathsf{T}X\beta \\ &\implies \boxed{\hat{\beta} = (X^\mathsf{T}X)^{-1}X^\mathsf{T}Y} \end{aligned}$$

1.2 Poisson Regression

- Poisson regression refer to a class of *linear* models where the output (predicted value) is a **function of the** linear combination (weighted sum) of the input variables

$$\begin{aligned} \eta(x;\beta) &= \beta_0 + \sum_{j=1}^{p} \beta_j x_j \\ f(x;\beta) &= \exp\left(\beta_0 + \sum_{j=1}^{p} \beta_j x_j\right) \\ &= \exp\left(\eta(x;\beta)\right) \end{aligned}$$

where $x = [x_1, \dots, x_p]^\mathsf{T}$ is a vector of features/variables/attributes and $\hat{Y}|x = f(x;\hat{\beta}) = \exp\left(x^\mathsf{T}\hat{\beta}\right)$ is the predicted outcome at $X = x$.

- the *model parameters* for linear (or coefficients or weights), $\hat{\beta}$ determine how much influence each feature has on the predicted output.

1.2.1 Model Structure

- Outcome variable: $y \in \{0, 1, \dots\}$
- Predictor variables $\mathbf{x} \in \mathbb{R}^p$
- Model parameters: $\beta = (\beta_0, \beta_1, \dots \beta_p)$
- Prediction function: $f(\mathbf{x};\hat{\beta}) = \exp\left(\hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_1 x_2 + \dots + \hat{\beta}_1 x_p\right) = e^{\mathbf{x}^\mathsf{T}\beta}$

1.2.2 Model Fitting (or Parameter estimation)

- The model is fitted by minimizing the *Poisson Loss*, which is the negative log-likelihood of the Poisson pmf.
  - Even better, add a penalty to the loss (e.g., elasticnet) to prevent over-fitting.

2 Prediction Pipeline

We will build up a more complete predictive modeling pipeline, but for today we will focus on this version that ignores where the data came from, the purpose of modeling (or problem we are trying to solve), and how to evaluate the predictions so that they help us address the problem.

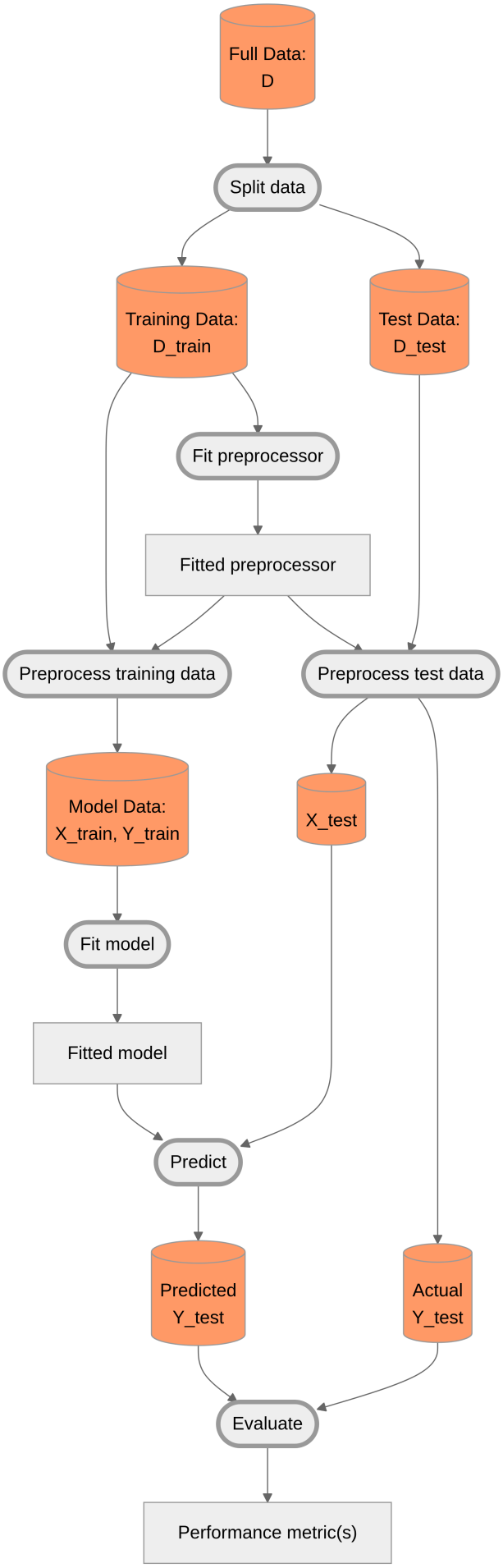
Full Data: D
Split data
Training Data: D_train
Test Data: D_test
Fit preprocessor
Fitted preprocessor
Preprocess training data
Preprocess test data
Model Data: X_train, Y_train
X_test
Fit model
Fitted model
Predict
Predicted Y_test
Actual Y_test
Evaluate
Performance metric(s)

Figure 1

3 Coding the Predictive Pipeline

First we grab the advertising data (modified from ISLR datasets). I added a categorical `promotion` feature and made some of the `newspaper` values to be missing.

```
#: Load Data
dir_data = 'https://mdporter.github.io/teaching/data'
url = file.path(dir_data, "advertising.csv")
advert = read_csv(url)
```

```
#: Show data
advert
#> # A tibble: 200 x 5
#>      TV radio newspaper sales promotion
#>   <dbl> <dbl>     <dbl> <dbl> <chr>
#> 1 230.   37.8      69.2  22.1 No
#> 2  44.5  39.3      45.1  10.4 No
#> 3  17.2  45.9      69.3   9.3 No
#> 4 152.   41.3      NA    18.5 No
#> 5 181.   10.8      58.4  12.9 No
#> 6   8.7  48.9      75     7.2 No
#> # i 194 more rows
```

```
advert %>% mutate_if(is.character, factor) %>% summary()
#>        TV             radio          newspaper          sales       promotion
#>  Min.   :  0.7   Min.   : 0.00   Min.   :  0.3   Min.   : 1.6   No :183
#>  1st Qu.: 74.4   1st Qu.: 9.97   1st Qu.: 12.9   1st Qu.:10.4   Yes: 17
#>  Median :149.8   Median :22.90   Median : 25.6   Median :12.9
#>  Mean   :147.0   Mean   :23.26   Mean   : 30.7   Mean   :14.0
#>  3rd Qu.:218.8   3rd Qu.:36.52   3rd Qu.: 45.1   3rd Qu.:17.4
#>  Max.   :296.4   Max.   :49.60   Max.   :114.0   Max.   :27.0
#>                                  NA's   :8
```

3.1 tidymodels (R)

```
library(tidymodels)


#: Create train/test split of the data
set.seed(20261)    # set random seed
n_test = 50        # set number of test observations
prop = n_test / nrow(advert) # calculate proportion of data to use for training
df =
  advert |>
  initial_split(prop = prop, strata = NULL)

# training(df)
# testing(df)

#: Create the recipe for pre-processing
rec =
  recipe(sales ~ TV + radio + newspaper + promotion,
         data = training(df)) |> # use training data
  step_impute_median(all_numeric_predictors()) |>
  step_ns(TV, deg_free = 5) |>
  step_dummy(all_nominal_predictors())

#: Create the linear regression model specification
linear_model = linear_reg() |> set_engine("lm")

#: Put into a workflow
wf =
```

```r
  workflow(
    spec = linear_model,
    preprocessor = rec
  )

#: Fit the workflow
lm_fit = fit(wf, data = training(df))   # be sure to use training data

#: Make predictions on test data
y_hat = predict(lm_fit, testing(df))

#: Evaluate predictions
rmse_vec(truth = testing(df)$sales, estimate = y_hat$.pred)   # using RMSE
#> [1] 1.681
rsq_vec(truth = testing(df)$sales, estimate = y_hat$.pred)    # using R^2
#> [1] 0.8966
```

3.2 scikit-learn (Python)

Using sample data (will fix later)

```python
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, SplineTransformer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression

np.random.seed(1)
df = pd.DataFrame({
    "y":  np.random.randn(200),
    "x1": np.random.randn(200),
    "x2": np.random.choice(["A", "B"], size=200)
})

# add some missingness in x1
missing_idx = np.random.choice(df.index, size=20, replace=False)
df.loc[missing_idx, "x1"] = np.nan

X = df[["x1", "x2"]]
y = df["y"]

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)

numeric_pipeline = Pipeline(steps=[
    ("impute", SimpleImputer(strategy="median")),
    ("spline", SplineTransformer(n_knots=6, degree=3, include_bias=False))
])

preprocess = ColumnTransformer(
    transformers=[
        ("num", numeric_pipeline, ["x1"]),
        ("cat", OneHotEncoder(drop="first", handle_unknown="ignore"), ["x2"])
    ]
)

pipe = Pipeline(steps=[
    ("prep", preprocess),
    ("model", LinearRegression())
])
```

```
pipe.fit(X_train, y_train)

y_pred = pipe.predict(X_test)
```