Review #1

SYS 6018 | Spring 2023

review-1.pdf

Contents

---

1 Homework

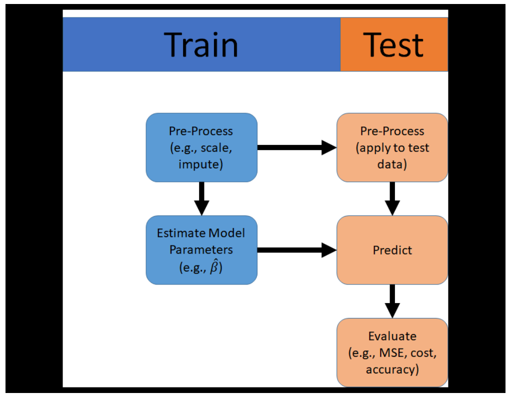
1.1 HW 1

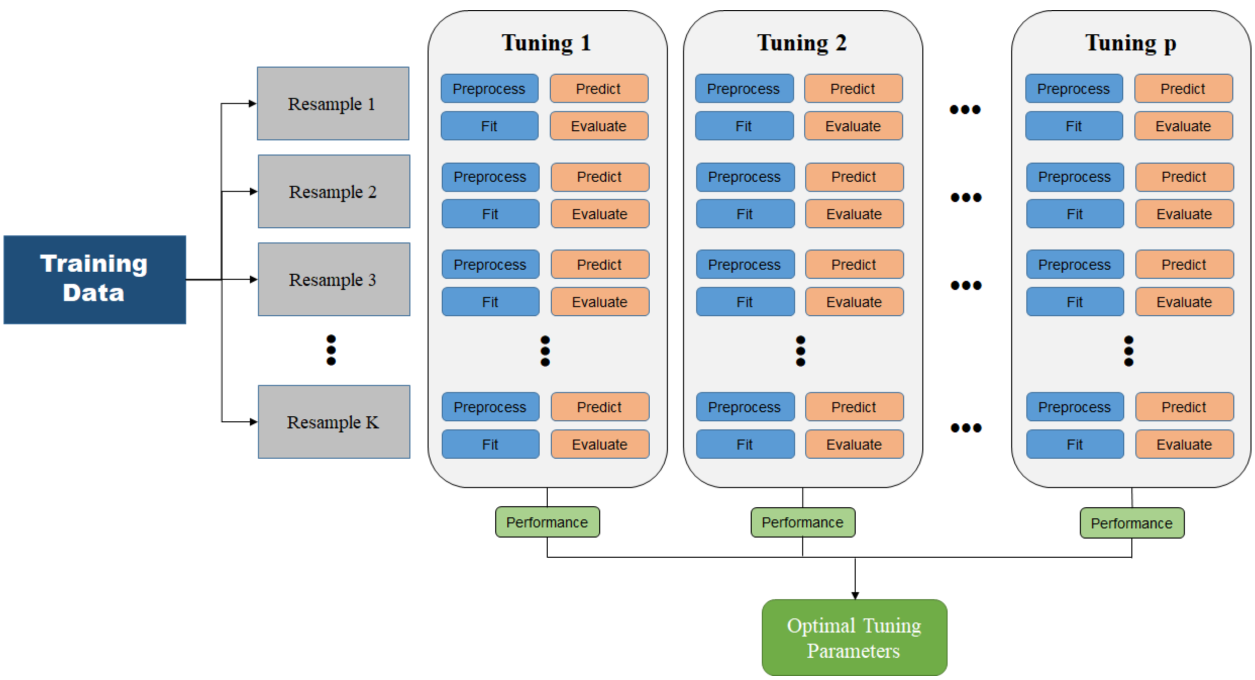
- The best predictive model is not always the true model.
  - Quadratic didn't always make best predictions. Why not?

1.2 HW 2

- Bootstrap Questions

- Cross-validation Questions
  - Cross-validation training data is $(K-1)/K$ of full training data





1.3 HW 3

- lambda min vs. one standard error rule
- Ensure same folds for all tuning parameters
- two model comparison test (paired t-test, permutation test)

1.3.1 Contest Results

1.4 HW 4

1.4.1 Contest Part 1 Results

1.4.2 Contest Part 2 Results

2 Calibration Curves

The textbook *An Introduction to Statistical Learning (ISL)* has a description of a simulated credit card default dataset. The interest is on predicting whether an individual will default on their credit card payment.

```
data(Default, package="ISLR")

#: Create binary column (y)
Default = Default %>% mutate(y = if_else(default == "Yes", 1L, 0L))
```

The variables are:

- *outcome variable* is categorical (factor) `Yes` and `No`, (`default`)
- the categorical (factor) variable (`student`) is either `Yes` or `No`
- the average balance a customer has after making their monthly payment (`balance`)
- the customer's income (`income`)

```
set.seed(11)
Default %>% slice_sample(n=6)
```

default	student	balance	income	y
No	No	396.5	41970	0
No	No	913.6	46907	0
No	Yes	561.4	21747	0
Yes	Yes	1889.3	22652	1
No	No	491.0	37836	0
No	Yes	282.2	19809	0

A risk model is said to be *calibrated* if the predicted probabilities are equal to the true risk (probabilities).

$$\Pr(Y = 1 \mid \hat{p} = p) = p \qquad \text{for all } p$$

Create train/test split

```
#: train/test split
set.seed(2019)
test = sample(nrow(Default), size=2000)
train = -test
```

Fit logistic regression model to training data

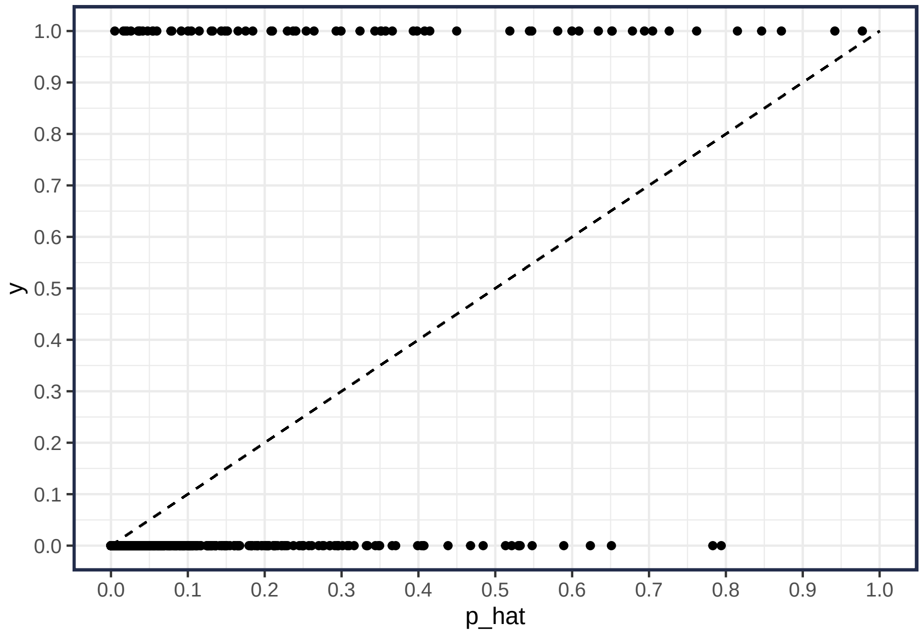
```
#: fit logistic regression on training data
fit.lm = glm(y~student + balance + income, family='binomial',
             data=Default[train, ])
```

Make predictions on test data

```
p_hat = predict(fit.lm, Default[test,], type="response")
preds_test = tibble(
  y = Default$y[test],
  student = Default$student[test],
  p_hat = p_hat
)
```

```
plt = preds_test %>%
  ggplot(aes(p_hat, y)) + geom_point() +
  scale_x_continuous(breaks = seq(0, 1, by=.1)) +
```

```
  scale_y_continuous(breaks = seq(0, 1, by=.1)) +
  coord_cartesian(xlim = c(0,1), ylim=c(0,1)) +
  geom_segment(x = 0, xend = 1, y = 0, yend = 1, lty=2) # perfect calibration
plt
```

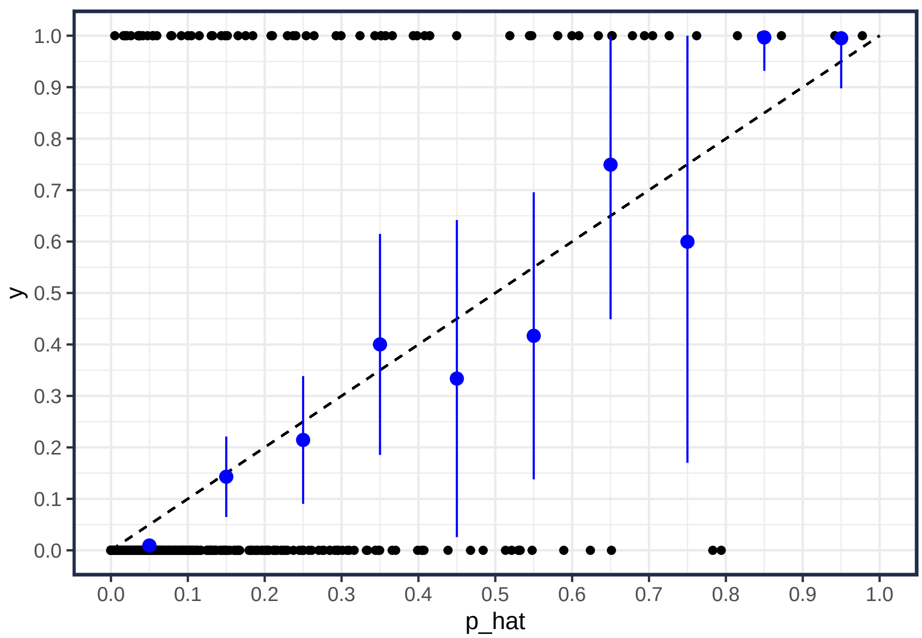


Create bins along the x-axis ($\hat{p}$) and calculate the mean response in each bin. Using Laplace smoothing to avoid extreme $\{0, 1\}$ estimates.

```
bks = seq(0, 1, by = .10)
mids = bks[-1] - diff(bks)/2
binned_data = preds_test %>%
  mutate(
    p_hat_bin = cut(p_hat, breaks = bks, include.lowest = TRUE),
    midpoint = mids[as.integer(p_hat_bin)]
  ) %>%
  group_by(midpoint) %>%
  summarize(
    n = n(),
    n1 = sum(y == 1) + .01, # add .01 defaults to each bin
    n0 = sum(y == 0) + .01, # add .01 non-defaults to each bin
    p = n1 / (n0 + n1),
    se = sqrt(p*(1-p)/n),
    upper = pmin(p + 1.96*se, 1),
    lower = pmax(p - 1.96*se, 0)
  )
binned_data
#> # A tibble: 10 x 8
#>   midpoint     n    n1      n0       p      se  upper   lower
#>      <dbl> <int> <dbl>   <dbl>   <dbl>   <dbl>  <dbl>   <dbl>
#> 1     0.05  1822 17.0  1805.   0.00934 0.00225 0.0138 0.00492
#> 2     0.15    77 11.0    66.0  0.143   0.0399  0.221  0.0648
#> 3     0.25    42  9.01   33.0  0.214   0.0633  0.339  0.0903
#> 4     0.35    20  8.01   12.0  0.400   0.110   0.615  0.185
#> 5     0.45     9  3.01    6.01 0.334   0.157   0.642  0.0256
#> 6     0.55    12  5.01    7.01 0.417   0.142   0.696  0.138
#> # ... with 4 more rows
```

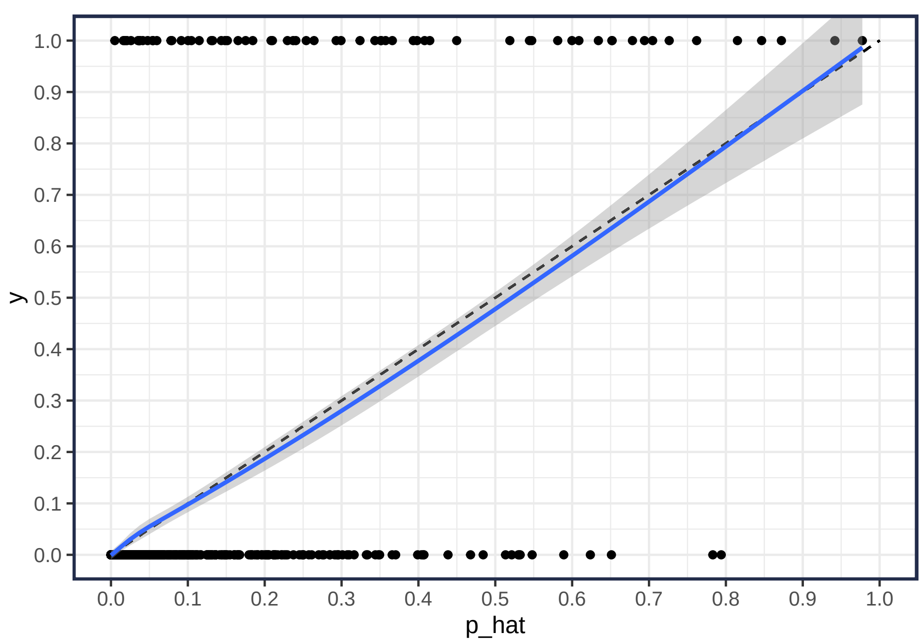
Plot binned estimates.

```
plt +
  geom_pointrange(data = binned_data,
                  aes(midpoint, y=p, ymin=lower, ymax=upper),
                  color = "blue")
```



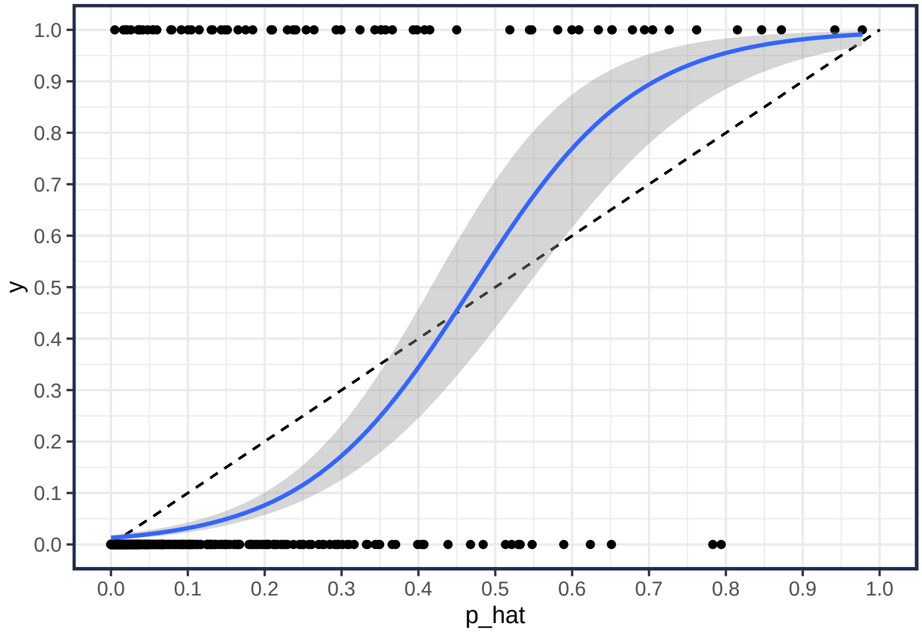
We could have instead added smooth line fit (predictor variable is $\hat{p}$, outcome variable is y). Note that this implements linear regression (squared error loss).

```
plt + geom_smooth()
```



A better way that incorporates the uncertainty that varies with $\hat{p}$ is to use logistic regression. If we try the add directly into `geom_smooth()` it doesn't looks quite right, why?

```
plt + geom_smooth(method = "glm", method.args = list(family = "binomial"))
```



Think of the structure of logistic regression - the linear component captures the *logit* of p (what we referred to as γ in a previous class). I.e.,

$$\text{logit } p(x) = \beta_0 + \beta_1 \hat{p}(x)$$

but we don't want this!

Rather, something like this is what we want

$$\text{logit } p(x) = \beta_0 + \beta_1 \hat{p}(x) + \text{logit } \hat{p}(x)$$

fit on a hold-out set, and check how far β_0 and β_1 are from 0.

```
preds_test %>%
  mutate(gamma = log(p_hat) - log(1-p_hat)) %>%
  glm(y ~ p_hat + offset(gamma), family = "binomial", data = .) %>%
  broom::tidy()
#> # A tibble: 2 x 5
#>   term        estimate std.error statistic p.value
#>   <chr>          <dbl>     <dbl>     <dbl>   <dbl>
#> 1 (Intercept)  0.00981     0.221     0.0444   0.965
#> 2 p_hat       -0.187       0.720    -0.259    0.795
```

Or examine non-linear deviations with B-splines:

```
library(splines)
smooth_fit = preds_test %>%
  mutate(gamma = log(p_hat) - log(1-p_hat)) %>%
  glm(y ~ splines::bs(p_hat) + offset(gamma),
      family = "binomial", data = .)
```

```
smooth_fit %>% broom::tidy()
#> # A tibble: 4 x 5
#>   term                    estimate std.error statistic p.value
#>   <chr>                      <dbl>     <dbl>     <dbl>   <dbl>
#> 1 (Intercept)               0.0224     0.348    0.0645   0.949
```

```
#> 2 splines::bs(p_hat)1   0.0241     1.53     0.0158   0.987
#> 3 splines::bs(p_hat)2  -0.677      2.20    -0.308    0.758
#> 4 splines::bs(p_hat)3   0.558      2.58     0.216    0.829
```