

# Predictive Bias and Calibration

DS 6030 | Fall 2023

calibration.pdf

## Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Probability Modeling</b>            | <b>2</b>  |
| 1.1      | Data . . . . .                         | 2         |
| 1.2      | Logistic Regression . . . . .          | 2         |
| 1.3      | Predictive Performance . . . . .       | 2         |
| 1.4      | Performance by Group . . . . .         | 5         |
| <b>2</b> | <b>Predictive Bias and Calibration</b> | <b>8</b>  |
| 2.1      | Binning . . . . .                      | 8         |
| 2.2      | Nearest Neighbor . . . . .             | 9         |
| 2.3      | Smoothing splines . . . . .            | 9         |
| 2.4      | Summary . . . . .                      | 11        |
| 2.5      | Calibration by group . . . . .         | 11        |
| <b>3</b> | <b>Testing for Calibration</b>         | <b>13</b> |
| 3.1      | Logistic Regression . . . . .          | 13        |
| 3.2      | Linear bias . . . . .                  | 13        |
| 3.3      | Non-linear bias . . . . .              | 14        |
| 3.4      | Bias in predictors . . . . .           | 14        |
| <b>4</b> | <b>SVM Calibration</b>                 | <b>17</b> |
| 4.1      | SVM Modeling . . . . .                 | 17        |
| 4.2      | SVM Calibration . . . . .              | 18        |

---

# 1 Probability Modeling

## 1.1 Data

```
#: Load Data, Create binary column (y)
data(Default, package="ISLR")
Default = Default %>% as_tibble() %>%
  mutate(y = if_else(default == "Yes", 1L, 0L), .after=default)
```

| default | y | student | balance | income |
|---------|---|---------|---------|--------|
| No      | 0 | No      | 1384.9  | 40131  |
| Yes     | 1 | Yes     | 1889.3  | 22652  |
| Yes     | 1 | Yes     | 1740.8  | 18161  |
| Yes     | 1 | Yes     | 2123.4  | 23836  |
| No      | 0 | Yes     | 856.7   | 15523  |
| No      | 0 | No      | 310.1   | 31446  |
| No      | 0 | No      | 1248.9  | 31960  |
| Yes     | 1 | No      | 1823.6  | 44260  |

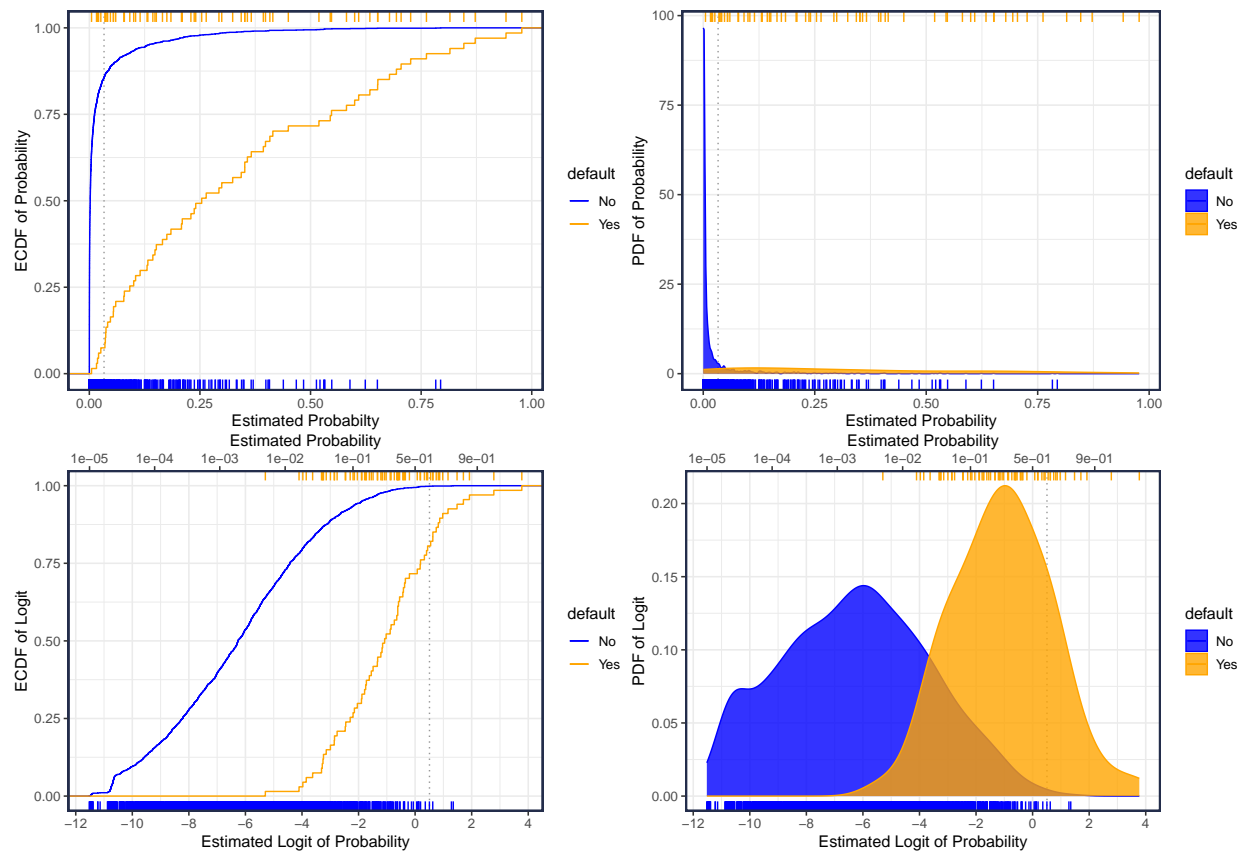
```
#: train/test split
set.seed(2019)
test = sample(nrow(Default), size=2000)
train = -test
```

## 1.2 Logistic Regression

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

## 1.3 Predictive Performance

```
#: table of predictions and true values
tbl_lr = Default[test,] %>%
  mutate(
    g = factor(y, c(0,1)),
    p_hat = predict(fit_lr, ., type="response"),
    gamma_hat = predict(fit_lr, ., type="link"),
  )
```



```

# Threshold of 1/2
tbl_lr %>%
  mutate(Y_hat = ifelse(p_hat > 1/2, 1L, 0L)) %>%
  count(y, Y_hat)
#> # A tibble: 4 x 3
#>   y Y_hat     n
#>   <int> <int> <int>
#> 1     0     0 1922
#> 2     0     1   11
#> 3     1     0   48
#> 4     1     1   19

```

```

# Threshold of 1/4
tbl_lr %>%
  mutate(Y_hat = ifelse(p_hat > 1/4, 1L, 0L)) %>%
  count(y, Y_hat)
#> # A tibble: 4 x 3
#>   y Y_hat     n
#>   <int> <int> <int>
#> 1     0     0 1891
#> 2     0     1   42
#> 3     1     0   33
#> 4     1     1   34

```

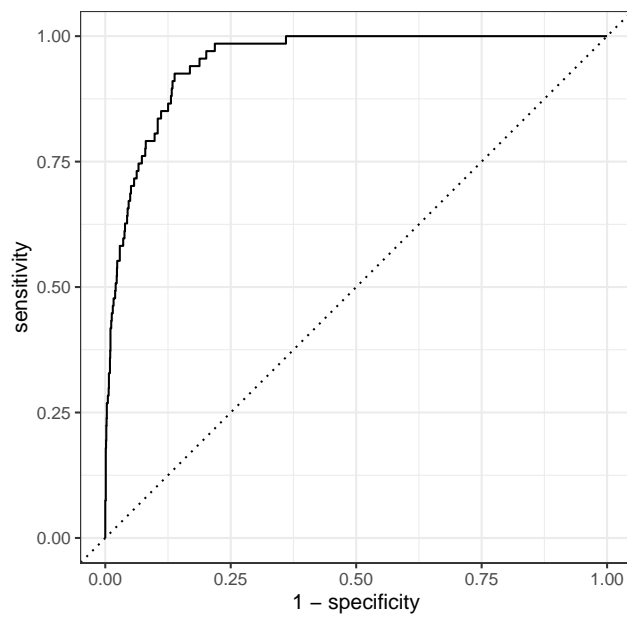
```

library(yardstick)
roc_curve(tbl_lr, truth = g, p_hat, event_level="second") %>%
  autoplot()

```

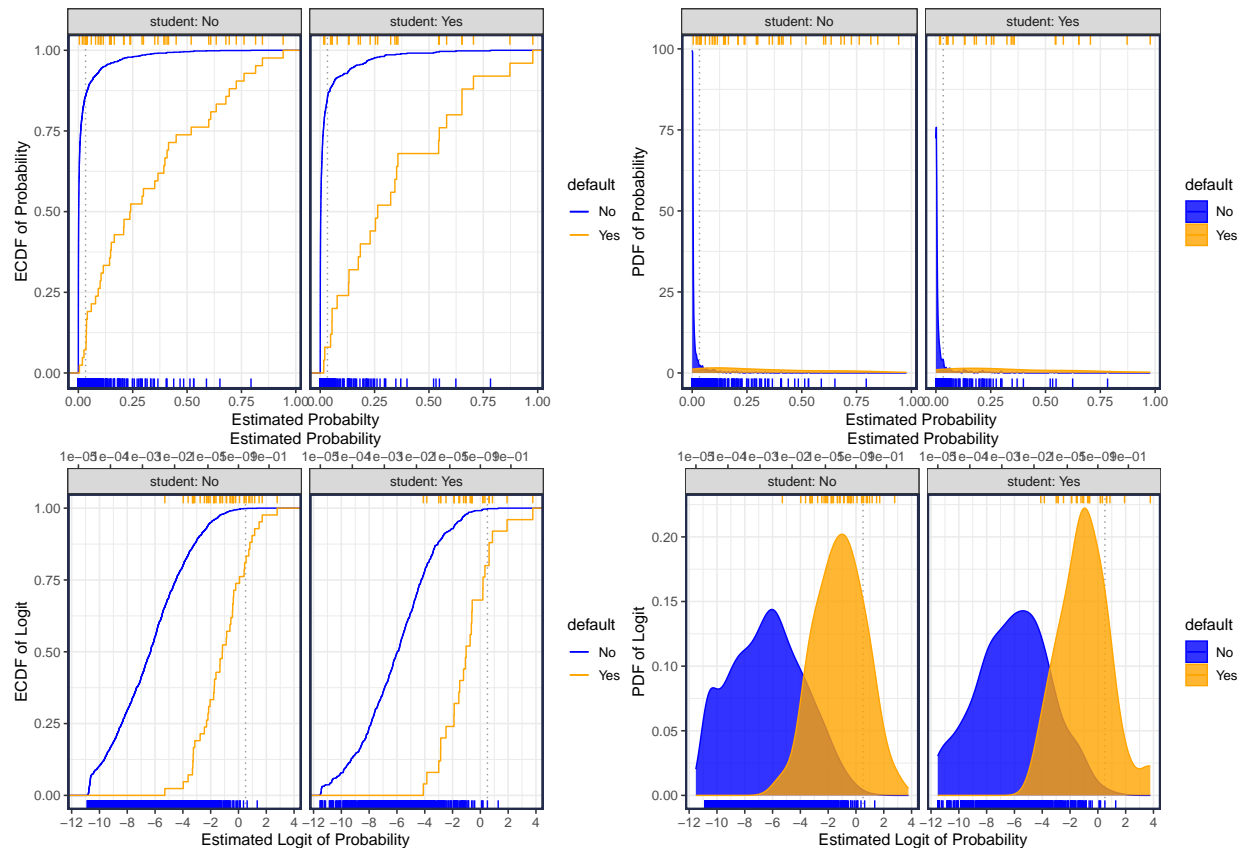
| thres | $\hat{P}$ | $\hat{N}$ | FP  | FN |
|-------|-----------|-----------|-----|----|
| 0.1   | 178       | 1822      | 128 | 17 |
| 0.2   | 101       | 1899      | 62  | 28 |
| 0.3   | 59        | 1941      | 29  | 37 |
| 0.4   | 39        | 1961      | 17  | 45 |
| 0.5   | 30        | 1970      | 11  | 48 |
| 0.6   | 18        | 1982      | 4   | 53 |
| 0.7   | 10        | 1990      | 2   | 59 |
| 0.8   | 5         | 1995      | 0   | 62 |
| 0.9   | 2         | 1998      | 0   | 65 |

| n    | log_loss | brier | mae   | auroc |
|------|----------|-------|-------|-------|
| 2000 | 0.082    | 0.023 | 0.046 | 0.951 |



## 1.4 Performance by Group

All performance scores can also be assessed at the group level (i.e., over subsets of the features). Here we explore the predicted output by Student status.



```
#: Threshold of 1/2
tbl_lr %>%
  mutate(Y_hat = ifelse(p_hat > 1/2, 1L, 0L)) %>%
  count(student, y, Y_hat)
#> # A tibble: 8 x 4
#>   student y Y_hat n
#>   <fct>   <int> <int> <int>
#> 1 No      0      0 1356
#> 2 No      0      1    6
#> 3 No      1      0   31
#> 4 No      1      1   11
#> 5 Yes     0      0  566
#> 6 Yes     0      1    5
#> # i 2 more rows
```

```
#: Threshold of 1/4
tbl_lr %>%
  mutate(Y_hat = ifelse(p_hat > 1/4, 1L, 0L)) %>%
  count(student, y, Y_hat)
#> # A tibble: 8 x 4
#>   student y Y_hat n
#>   <fct>   <int> <int> <int>
#> 1 No      0      0 1334
#> 2 No      0      1   28
#> 3 No      1      0   22
```

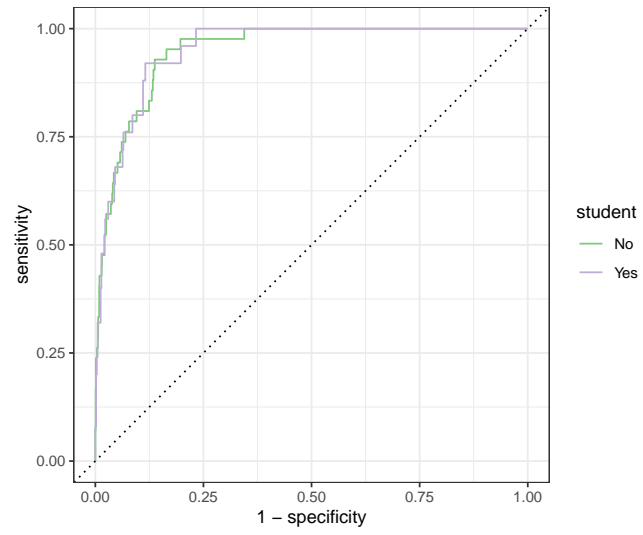
```
#> 4 No      1      1      20
#> 5 Yes     0      0     557
#> 6 Yes     0      1      14
#> # i 2 more rows
```

| thres | student | $\hat{P}$ | $\hat{N}$ | FP | FN |
|-------|---------|-----------|-----------|----|----|
| 0.1   | No      | 114       | 1290      | 83 | 11 |
| 0.2   | No      | 63        | 1341      | 39 | 18 |
| 0.3   | No      | 38        | 1366      | 20 | 24 |
| 0.4   | No      | 26        | 1378      | 12 | 28 |
| 0.5   | No      | 17        | 1387      | 6  | 31 |
| 0.6   | No      | 11        | 1393      | 2  | 33 |
| 0.7   | No      | 6         | 1398      | 1  | 37 |
| 0.8   | No      | 3         | 1401      | 0  | 39 |
| 0.9   | No      | 1         | 1403      | 0  | 41 |

| thres | student | $\hat{P}$ | $\hat{N}$ | FP | FN |
|-------|---------|-----------|-----------|----|----|
| 0.1   | Yes     | 64        | 532       | 45 | 6  |
| 0.2   | Yes     | 38        | 558       | 23 | 10 |
| 0.3   | Yes     | 21        | 575       | 9  | 13 |
| 0.4   | Yes     | 13        | 583       | 5  | 17 |
| 0.5   | Yes     | 13        | 583       | 5  | 17 |
| 0.6   | Yes     | 7         | 589       | 2  | 20 |
| 0.7   | Yes     | 4         | 592       | 1  | 22 |
| 0.8   | Yes     | 2         | 594       | 0  | 23 |
| 0.9   | Yes     | 1         | 595       | 0  | 24 |

| student | n    | log_loss | brier | mae   | auroc |
|---------|------|----------|-------|-------|-------|
| No      | 1404 | 0.076    | 0.021 | 0.042 | 0.949 |
| Yes     | 596  | 0.096    | 0.028 | 0.054 | 0.951 |

```
library(yardstick)
tbl_lr %>% group_by(student) %>%
  roc_curve(truth = g, p_hat, event_level="second") %>%
  autoplot() + scale_color_brewer(type = "qual")
```



## 2 Predictive Bias and Calibration

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

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

To evaluate the calibration of a model's predictions, we need to estimate the proportion of the observations with  $Y = 1$  and  $\hat{p} \approx p$ .

Calibration *plots* can be used to measure drift, fairness, and model/algorithmic bias. We could for example use binning (regressograms/histograms), kNN, smoothing, or isotonic regression.

### 2.1 Binning

Here is an example of binning. I'll partition the predictions such that there are 10 groups of equal width.

```
tbl_grp = tbl_lr %>%
  mutate(grp = cut_width(p_hat, width = .1, boundary = 1)) %>%
  group_by(grp) %>%
  summarize(
    n = n(),
    lower = min(p_hat),
    upper = max(p_hat),
    p_hat = mean(p_hat),
    n_y = sum(y),
    # bayesian (uniform prior)
    p_y = (n_y+1) / (n+2), # posterior mean
    beta_lower = qbeta(.025, n_y+1, n-n_y+1),
    beta_upper = qbeta(.975, n_y+1, n-n_y+1),
    # frequentist
    p_y_bar = mean(y),
    moe = 1.96*sqrt(p_y_bar*(1-p_y_bar)/n)
  )
```

| grp       | n    | lower | upper | p_hat | n_y | p_y   | beta_lower | beta_upper | p_y_bar | moe   |
|-----------|------|-------|-------|-------|-----|-------|------------|------------|---------|-------|
| [0,0.1]   | 1822 | 0.000 | 0.100 | 0.009 | 17  | 0.010 | 0.006      | 0.015      | 0.009   | 0.004 |
| (0.1,0.2] | 77   | 0.100 | 0.199 | 0.141 | 11  | 0.152 | 0.082      | 0.238      | 0.143   | 0.078 |
| (0.2,0.3] | 42   | 0.202 | 0.299 | 0.242 | 9   | 0.227 | 0.118      | 0.360      | 0.214   | 0.124 |
| (0.3,0.4] | 20   | 0.301 | 0.399 | 0.348 | 8   | 0.409 | 0.218      | 0.616      | 0.400   | 0.215 |
| (0.4,0.5] | 9    | 0.404 | 0.484 | 0.431 | 3   | 0.364 | 0.122      | 0.652      | 0.333   | 0.308 |
| (0.5,0.6] | 12   | 0.513 | 0.600 | 0.546 | 5   | 0.429 | 0.192      | 0.684      | 0.417   | 0.279 |
| (0.6,0.7] | 8    | 0.609 | 0.694 | 0.649 | 6   | 0.700 | 0.400      | 0.925      | 0.750   | 0.300 |
| (0.7,0.8] | 5    | 0.705 | 0.794 | 0.754 | 3   | 0.571 | 0.223      | 0.882      | 0.600   | 0.429 |
| (0.8,0.9] | 3    | 0.815 | 0.872 | 0.845 | 3   | 0.800 | 0.398      | 0.994      | 1.000   | 0.000 |
| (0.9,1]   | 2    | 0.942 | 0.978 | 0.960 | 2   | 0.750 | 0.292      | 0.992      | 1.000   | 0.000 |

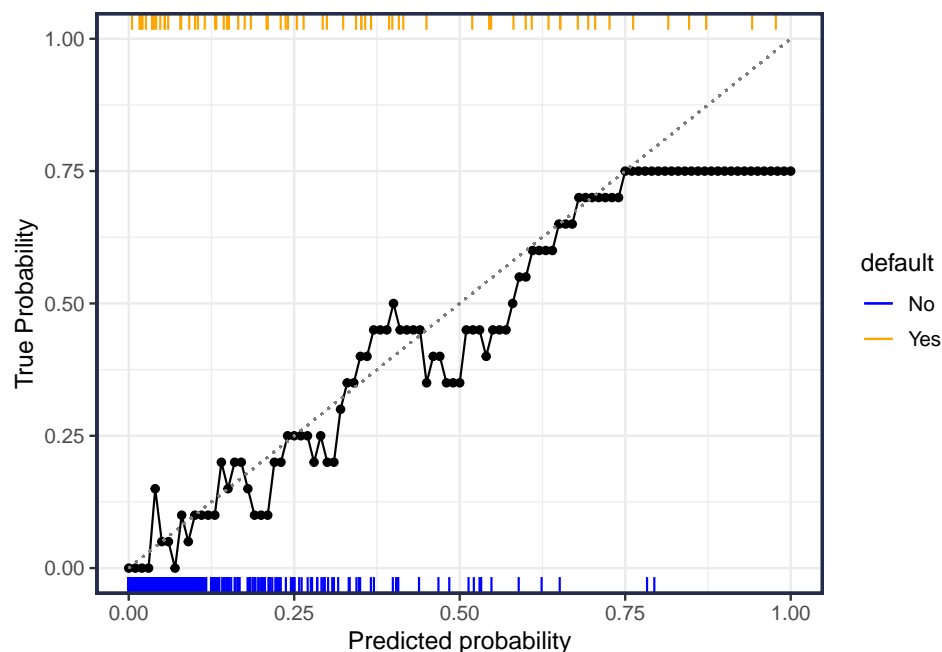
```
tbl_grp %>%
  ggplot(aes(p_hat, p_y)) +
    geom_rect(aes(xmin=lower, xmax=upper,
                  ymin=beta_lower,
                  ymax=beta_upper))
  ) +
  geom_point()
```



```
geom_linerange(aes(xmin=lower, xmax=upper)) +
geom_segment(x=0,xend=1, y=0, yend=1, linetype = 3, color = "grey50") +
labs(x = "Predicted probability", y = "True Probability") +
geom_rug(data = . %>% filter(y==0),
         aes(x=p_hat, color=default), sides="b",
         inherit.aes = FALSE) +
geom_rug(data = . %>% filter(y==1),
         aes(x=p_hat,color=default), sides="t",
         inherit.aes = FALSE) +
scale_color_manual(values=c(Yes="orange", No="blue"))
#> Error in `geom_rug()`:
#> ! Problem while computing layer data.
#> i Error occurred in the 5th layer.
#> Caused by error in `filter()`:
#> i In argument: `y == 0`.
#> Caused by error:
#> ! object 'y' not found
```

## 2.2 Nearest Neighbor

K-nearest neighbor:



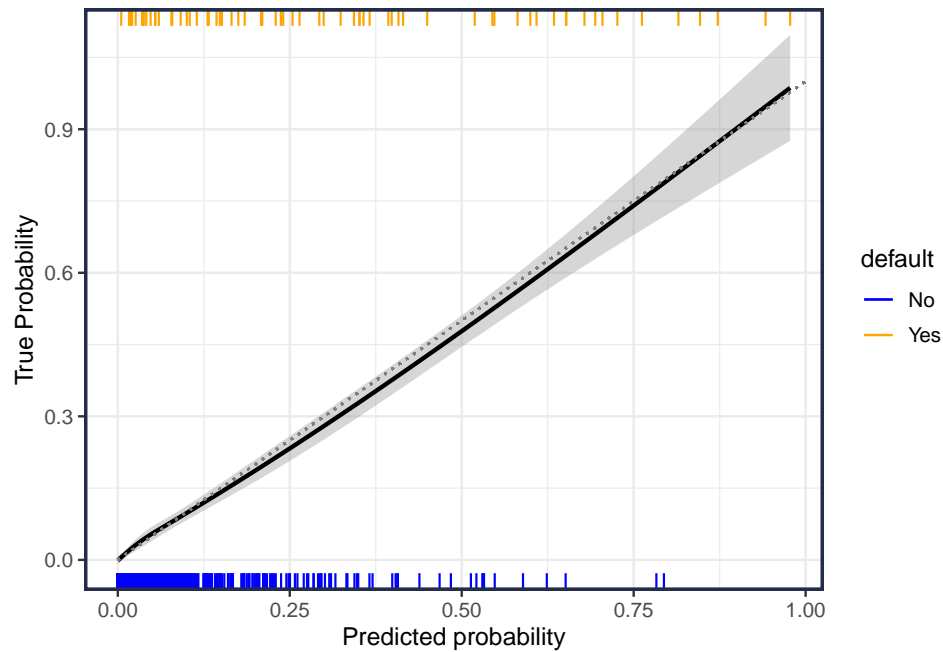
But knn is not expected to work well at the edges ( $\hat{p}$  close to 0 or 1).

## 2.3 Smoothing splines

Using smoothing splines (brier score loss):

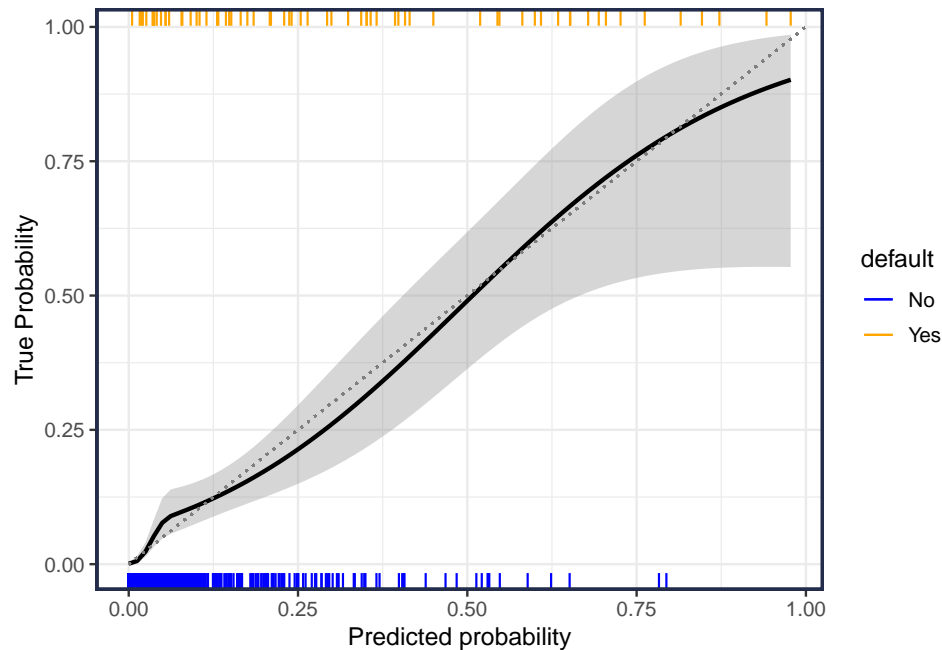
```
#: The Regular geom_smooth() uses an MSE (brier score) loss
tbl_lr %>%
  ggplot(aes(p_hat)) +
  geom_smooth(aes(y = y), color="black") +
  geom_segment(x=0,xend=1, y=0, yend=1, linetype = 3, color = "grey50") +
  geom_rug(data = . %>% filter(y==0), aes(color=default), sides="b") +
  geom_rug(data = . %>% filter(y==1), aes(color=default), sides="t") +
```

```
scale_color_manual(values=c(Yes="orange", No="blue")) +
labs(x = "Predicted probability", y = "True Probability")
```



Using smoothing splines (log-loss):

```
#: Here we use family argument in geom_smooth() to force log-loss
tbl_lr %>%
  ggplot(aes(p_hat)) +
  geom_smooth(aes(y = y),
    method = "gam",
    method.args = list(family = binomial()),
    color="black") +
  geom_segment(x=0,xend=1, y=0, yend=1, linetype = 3, color = "grey50") +
  geom_rug(data = . %>% filter(y==0), aes(color=default), sides="b") +
  geom_rug(data = . %>% filter(y==1), aes(color=default), sides="t") +
  scale_color_manual(values=c(Yes="orange", No="blue")) +
  labs(x = "Predicted probability", y = "True Probability")
```



## 2.4 Summary

It looks like the logistic regression model is decently calibrated for *overall* calibration (i.e., aggregated). The predicted probabilities are suitably close to the observed proportions in a test set. That is, with 2000 test observations there is not enough evidence to suggest clear predictive bias.

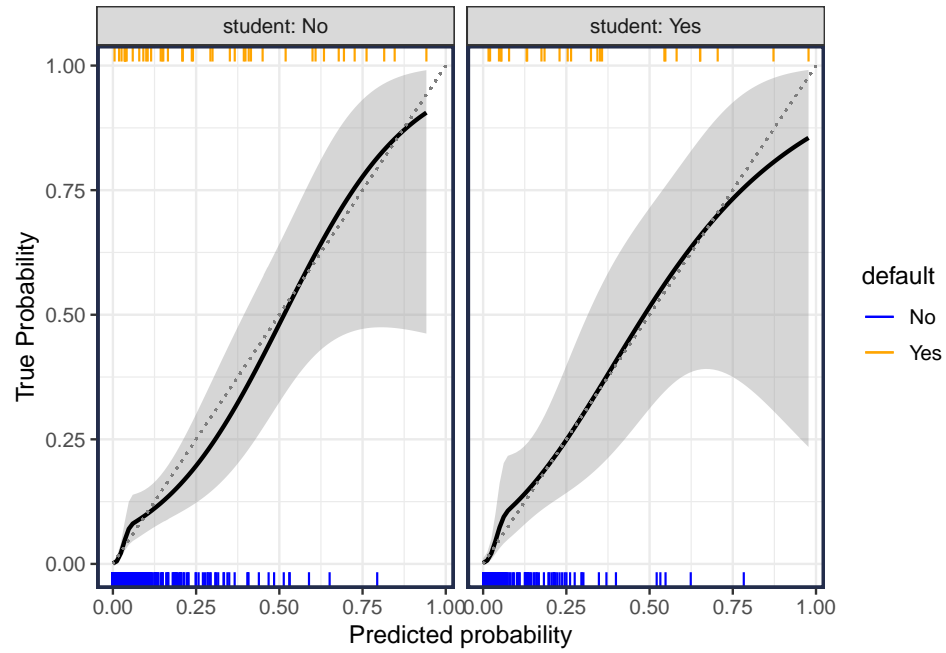
But, does predictive performance and calibration hold for all groups?

## 2.5 Calibration by group

Consider comparing the predictive performance of our models for Students and Non-Students.

$$\Pr(Y = 1 \mid \hat{p}(x) = p, X = x) = p \quad \text{for all } p \text{ and } x$$

```
tbl_lr %>%
  ggplot(aes(p_hat)) +
  geom_smooth(aes(y = y),
    method = "gam",
    method.args = list(family = binomial()),
    color="black") +
  geom_segment(x=0,xend=1, y=0, yend=1, linetype = 3, color = "grey50") +
  labs(x = "predicted probability", y = "true - predicted") +
  geom_rug(data = . %>% filter(y==0), aes(color=default), sides="b") +
  geom_rug(data = . %>% filter(y==1), aes(color=default), sides="t") +
  scale_color_manual(values=c(Yes="orange", No="blue")) +
  labs(x = "Predicted probability", y = "True Probability") +
  facet_wrap(~student, labeller = label_both) # <--- Faceting
```



### 3 Testing for Calibration

The main idea is to test the (null) hypothesis that

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

or equivalently,

$$\Pr(Y = 1 \mid \hat{p}(x)) - \hat{p}(x) = 0 \quad \text{for all } \hat{p}$$

The plots shown above visually explore the hypothesis. We can also take a model-based approach.

#### 3.1 Logistic Regression

For a calibrated model the estimated  $\hat{p}(x)$  should be close to the true  $p(x)$ ,

$$\begin{aligned} p(x) &\approx \hat{p}(x) \\ \text{logit } p(x) &\approx \text{logit } \hat{p}(x) \end{aligned}$$

To test this, we introduce a bias term  $b(x)$  and test for  $b(x) = 0 \forall x$ .

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

Notice the above expression is the same form as logistic regression.

This means we can use logistic regression to test for mis-calibration (predictive bias). To do this, use  $\text{logit } \hat{p}(x)$  as an *offset* in the logistic regression model. An *offset* is a term that has a fixed weight/coefficient of 1.

We also need to specify the form of the bias term before we can test it. We give a few examples below.

#### 3.2 Linear bias

To check for linear deviation, we specify the bias term as  $b(x) = \beta_0 + \beta_1 \text{logit } \hat{p}(x)$ .

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

Fit on a hold-out set, and check how far  $\beta_0$  and  $\beta_1$  are from 0.

```
calibrated = glm(y ~ gamma_hat + offset(gamma_hat),
  family = binomial,
  data = tbl_lr)
# Note that gamma_hat = logit(p_hat)

calibrated %>% summary()
#>
#> Call:
#> glm(formula = y ~ gamma_hat + offset(gamma_hat), family = binomial,
#>      data = tbl_lr)
#>
#> Coefficients:
#>              Estimate Std. Error z value Pr(>|z|)
```

```
#> (Intercept) -0.02237    0.22027   -0.10    0.92
#> gamma_hat    0.00589    0.09235    0.06    0.95
#>
#> (Dispersion parameter for binomial family taken to be 1)
#>
#> Null deviance: 326.02 on 1999 degrees of freedom
#> Residual deviance: 326.02 on 1998 degrees of freedom
#> AIC: 330
#>
#> Number of Fisher Scoring iterations: 8
```

No significance suggests not enough evidence to reject null of no bias against linear bias.

### 3.3 Non-linear bias

We can introduce splines to detect non-linear deviations:

```
glm(y~splines::bs(gamma_hat, df = 3) + offset(gamma_hat),
    family = binomial,
    data = tbl_lr) %>%
  summary()
#> Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
#>
#> Call:
#> glm(formula = y ~ splines::bs(gamma_hat, df = 3) + offset(gamma_hat),
#>      family = binomial, data = tbl_lr)
#>
#> Coefficients:
#>
#> Estimate Std. Error z value Pr(>|z|)
#> (Intercept)          -27.0      22.5  -1.20   0.23
#> splines::bs(gamma_hat, df = 3)1    43.7      36.7   1.19   0.23
#> splines::bs(gamma_hat, df = 3)2    18.4      15.3   1.21   0.23
#> splines::bs(gamma_hat, df = 3)3    30.5      26.1   1.17   0.24
#>
#> (Dispersion parameter for binomial family taken to be 1)
#>
#> Null deviance: 326.02 on 1999 degrees of freedom
#> Residual deviance: 323.50 on 1996 degrees of freedom
#> AIC: 331.5
#>
#> Number of Fisher Scoring iterations: 12
```

Again, no significance suggests not enough evidence to reject null of no bias against non-linear bias (although we could try other smoothing).

### 3.4 Bias in predictors

We can also see if the predictions should be adjusted for regions in features space.

Including an interaction term too.

```
glm(y ~ student + student:gamma_hat + offset(gamma_hat),
    family = binomial,
    data = tbl_lr) %>%
  summary()
#>
#> Call:
#> glm(formula = y ~ student + student:gamma_hat + offset(gamma_hat),
```

```
#> family = binomial, data = tbl_lr)
#>
#> Coefficients:
#> Estimate Std. Error z value Pr(>|z|)
#> (Intercept) -0.07335 0.27616 -0.27 0.79
#> studentYes 0.14052 0.45795 0.31 0.76
#> studentNo:gamma_hat 0.00509 0.11489 0.04 0.96
#> studentYes:gamma_hat 0.00368 0.15511 0.02 0.98
#>
#> (Dispersion parameter for binomial family taken to be 1)
#>
#> Null deviance: 326.02 on 1999 degrees of freedom
#> Residual deviance: 325.81 on 1996 degrees of freedom
#> AIC: 333.8
#>
#> Number of Fisher Scoring iterations: 8
```

```
glm(y ~ splines::bs(balance, 3) + offset(gamma_hat),
     family = binomial,
     data = tbl_lr) %>%
  summary()
#>
#> Call:
#> glm(formula = y ~ splines::bs(balance, 3) + offset(gamma_hat),
#>      family = binomial, data = tbl_lr)
#>
#> Coefficients:
#> Estimate Std. Error z value Pr(>|z|)
#> (Intercept) -15.55 15.59 -1.00 0.32
#> splines::bs(balance, 3)1 26.26 26.97 0.97 0.33
#> splines::bs(balance, 3)2 9.37 9.26 1.01 0.31
#> splines::bs(balance, 3)3 18.40 19.45 0.95 0.34
#>
#> (Dispersion parameter for binomial family taken to be 1)
#>
#> Null deviance: 326.02 on 1999 degrees of freedom
#> Residual deviance: 324.29 on 1996 degrees of freedom
#> AIC: 332.3
#>
#> Number of Fisher Scoring iterations: 12
```

```
glm(y ~ splines::bs(income, 3) + offset(gamma_hat),
     family = binomial,
     data = tbl_lr) %>%
  summary()
#>
#> Call:
#> glm(formula = y ~ splines::bs(income, 3) + offset(gamma_hat),
#>      family = binomial, data = tbl_lr)
#>
#> Coefficients:
#> Estimate Std. Error z value Pr(>|z|)
#> (Intercept) 0.435 1.038 0.42 0.68
#> splines::bs(income, 3)1 -0.756 2.829 -0.27 0.79
#> splines::bs(income, 3)2 -0.299 1.646 -0.18 0.86
#> splines::bs(income, 3)3 -1.104 2.564 -0.43 0.67
#>
#> (Dispersion parameter for binomial family taken to be 1)
#>
```

```
#>      Null deviance: 326.02  on 1999  degrees of freedom  
#> Residual deviance: 325.07  on 1996  degrees of freedom  
#> AIC: 333.1  
#>  
#> Number of Fisher Scoring iterations: 7
```

The logistic regression model appears well-calibrated. This isn't surprising as the log-loss metric encourages good calibration.

However, we could still get poor calibration due to over-fitting, drift, or mis-specification.



## 4 SVM Calibration

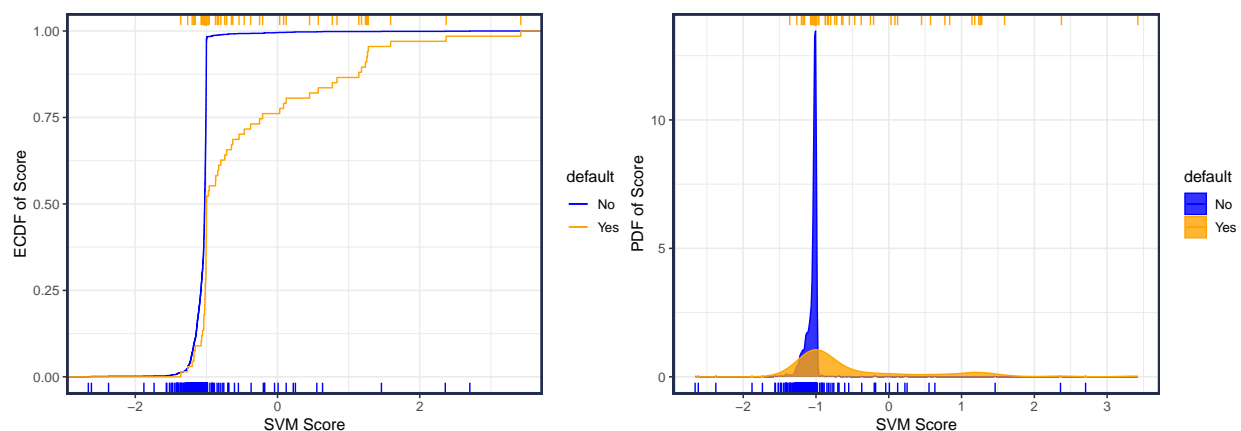
### 4.1 SVM Modeling

```
library(kernlab)
# Radial basis kernel
fit_svm = ksvm(default~student + balance + income,
  data = Default[train, ],
  scaled = TRUE, # be sure to scale the predictors
  kernel = "rbfdot", # kernel
  sigma = .01, C = 100, # tuning parameters
)

# table of predictions and true values
tbl_svm = Default[test,] %>%
  mutate(
    g = factor(y, c(0,1)),
    score = predict(fit_svm, ., type = "decision")[,1],
    class = predict(fit_svm, ., "response"),
  )
```

SVMs don't naturally output a probability, but rather a score (or decision) value that indicates the observations' distance to the decision boundary. We called the SVM score output  $\hat{f}(x)$  in the class notes.

| default | y | student | balance | income | g | score  | class |
|---------|---|---------|---------|--------|---|--------|-------|
| No      | 0 | Yes     | 1026.1  | 16431  | 0 | -1.023 | No    |
| No      | 0 | Yes     | 1378.2  | 20120  | 0 | -1.002 | No    |
| No      | 0 | No      | 487.1   | 55459  | 0 | -1.156 | No    |
| No      | 0 | Yes     | 1252.1  | 13861  | 0 | -0.999 | No    |
| No      | 0 | No      | 0.0     | 30208  | 0 | -1.004 | No    |
| No      | 0 | No      | 921.3   | 25350  | 0 | -1.046 | No    |



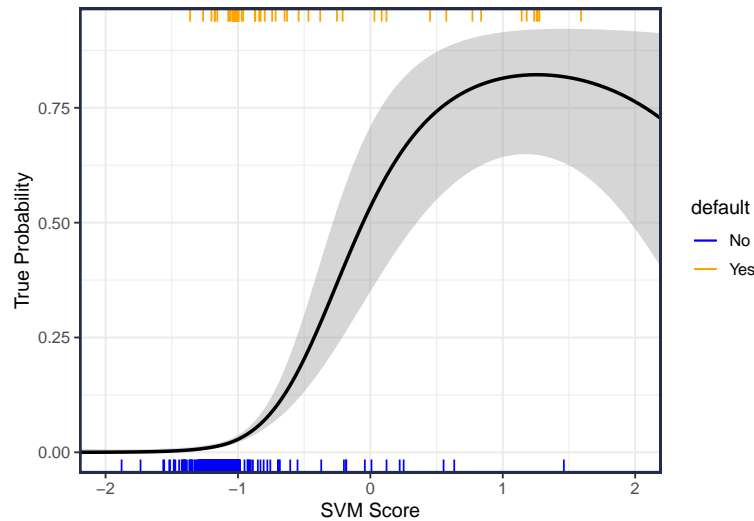
Let's check out how well the SVM scores map to a probability. Here I'll go with the smoothing spline approach (log-loss):

```
# Here we use family argument in geom_smooth() to force log-loss
tbl_svm %>%
  ggplot(aes(score)) +
  geom_smooth(aes(y = y), n=1000,
    method = "gam",
    method.args = list(family = binomial()),
```

```

color="black") +
geom_rug(data = . %>% filter(y==0), aes(color=default), sides="b") +
geom_rug(data = . %>% filter(y==1), aes(color=default), sides="t") +
scale_color_manual(values=c(Yes="orange", No="blue")) +
labs(x = "SVM Score", y = "True Probability") +
scale_x_continuous(breaks = seq(-5, 5, by=1)) +
coord_cartesian(xlim = c(-2, 2))

```



## 4.2 SVM Calibration

Let  $\hat{f}_i = \hat{f}(x_i)$  be the score output from an SVM model. Platt's idea was to fit a logistic regression model using  $\hat{f}(x)$  as the predictor variable. The *calibrated probabilities* are the predictions from this model. That is:

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

or

$$p(x) = 1 / \left( 1 + e^{-(\beta_0 + \beta_1 \hat{f}(x))} \right)$$

```

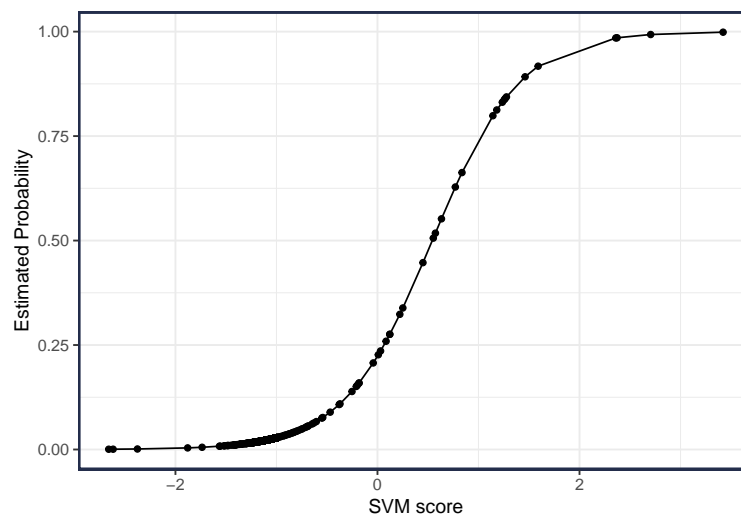
calibrated_svm = glm(y~score, family="binomial", data = tbl_svm)
summary(calibrated_svm)
#>
#> Call:
#> glm(formula = y ~ score, family = "binomial", data = tbl_svm)
#>
#> Coefficients:
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept)  -1.246      0.294   -4.24  2.2e-05 ***
#> score         2.297      0.289    7.95  1.9e-15 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for binomial family taken to be 1)
#>
#>      Null deviance: 586.82  on 1999  degrees of freedom
#> Residual deviance: 492.90  on 1998  degrees of freedom

```

```
#> AIC: 496.9
#>
#> Number of Fisher Scoring iterations: 6

tbl_svm = tbl_svm %>%
  mutate(
    p_hat = predict(calibrated_svm, ., type="response"),
    gamma_hat = log(p_hat) - log(1-p_hat)
  )

tbl_svm %>%
  ggplot(aes(score, p_hat)) +
  geom_point() +
  geom_line() +
  labs(x = "SVM score", y = "Estimated Probability")
```



Notice that when the SVM score is zero ( $\hat{f}(x) = 0$ ), the estimated probability is 0.25; this is pretty far from the expected 0.50.

### Sketch of SVM Calibration System

#### 4.2.1 Testing Calibration

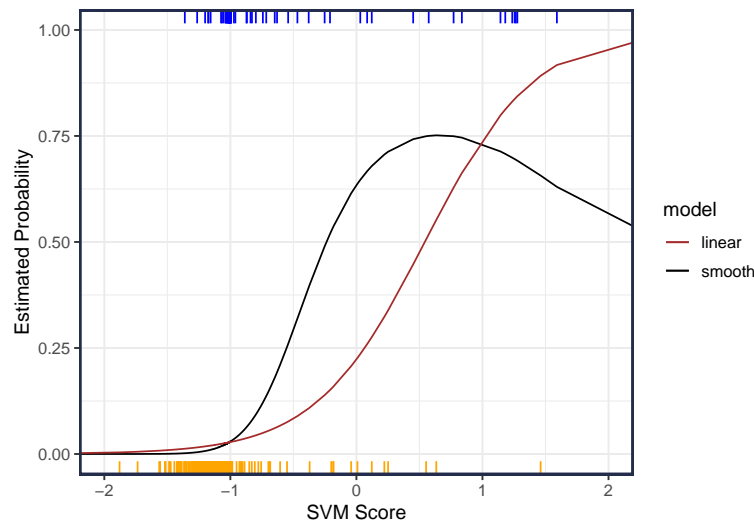
How well did Platt's method work? Is the 2 parameter logistic transformation sufficiently complex?

We can for test this, just like we did earlier. However, we are going to see a problem with using splines!

```
calibrated_svm_2 = glm(y~splines::bs(score, df=3),
                      family="binomial", data = tbl_svm)
summary(calibrated_svm_2)
#>
#> Call:
#> glm(formula = y ~ splines::bs(score, df = 3), family = "binomial",
#>      data = tbl_svm)
#>
#> Coefficients:
#>
#>               Estimate Std. Error z value Pr(>|z|)
#> (Intercept)      -24.41      4.51   -5.41  6.4e-08 ***
#> splines::bs(score, df = 3)1    39.84      9.89    4.03  5.7e-05 ***
#> splines::bs(score, df = 3)2    19.36      2.33    8.31 < 2e-16 ***
#> splines::bs(score, df = 3)3    25.87      5.87    4.41  1.0e-05 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for binomial family taken to be 1)
#>
#>      Null deviance: 586.82  on 1999  degrees of freedom
#> Residual deviance: 463.51  on 1996  degrees of freedom
#> AIC: 471.5
#>
#> Number of Fisher Scoring iterations: 7
```

Notice that the bspline with 3 df (4 edf with intercept) has statistically significant terms. But what does this non-linear calibration look like?

```
tbl_svm %>%
  ggplot(aes(score)) +
  geom_line(aes(y = p_smooth, color = "smooth")) +
  geom_line(aes(y = p_hat, color = "linear")) +
  geom_rug(data = . %>% filter(y==0), color = "orange", sides="b") +
  geom_rug(data = . %>% filter(y==1), color = "blue", sides="t") +
  scale_color_manual(name = "model", values=c(smooth="black", linear="brown")) +
  labs(x = "SVM Score", y = "Estimated Probability") +
  scale_x_continuous(breaks = seq(-5, 5, by=1)) +
  coord_cartesian(xlim = c(-2, 2))
```



That dip doesn't look good! It suggests a non-monotonic transformation which is not appealing. It would mean an individual that SVM classifies as a default ( $\hat{f} < 0$ ) could actually be given a higher probability that someone that SVM scored higher!

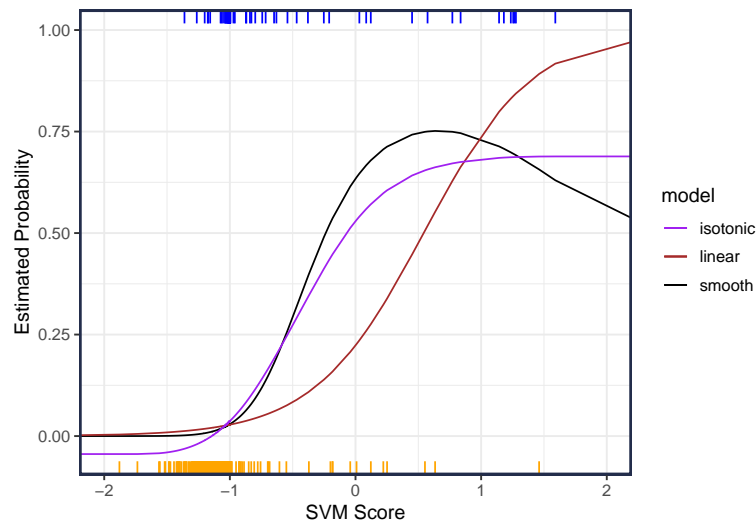
#### 4.2.2 Isotonic Calibration

Enter isotonic (or monotonic) splines. These are special splines that produce a monotonic prediction; something perfect for our application.

```
library(scam)
iso = scam(y~s(score, bs="mpi"), data=tbl_svm)
summary(iso)
#>
#> Family: gaussian
#> Link function: identity
#>
#> Formula:
#> y ~ s(score, bs = "mpi")
#> <environment: 0x5591a451aca8>
#>
#> Parametric coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept) -0.0443      0.0080  -5.54 3.5e-08 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Approximate significance of smooth terms:
#>             edf Ref.df    F p-value
```

```
#> s(score)      3      3 157 <2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> R-sq.(adj) =  0.1905   Deviance explained = 19.2%
#> GCV score = 0.026274   Scale est. = 0.026222   n = 2000
```

```
tbl_svm %>%
  ggplot(aes(score)) +
  geom_line(aes(y = p_smooth, color = "smooth")) +
  geom_line(aes(y = p_hat, color = "linear")) +
  geom_line(aes(y = p_iso, color = "isotonic")) +
  geom_rug(data = . %>% filter(y==0), color = "orange", sides="b") +
  geom_rug(data = . %>% filter(y==1), color = "blue", sides="t") +
  scale_color_manual(name = "model",
    values=c(smooth="black", linear="brown",
              isotonic = "purple")) +
  labs(x = "SVM Score", y = "Estimated Probability") +
  scale_x_continuous(breaks = seq(-5, 5, by=1)) +
  coord_cartesian(xlim = c(-2, 2))
```



It does appear that the more complex isotonic smoother deviates from the logistic, but with only 3 edf in the smoothing portion, it isn't too different.

#### 4.2.3 Hold-out data

### Calibration with Validation Data

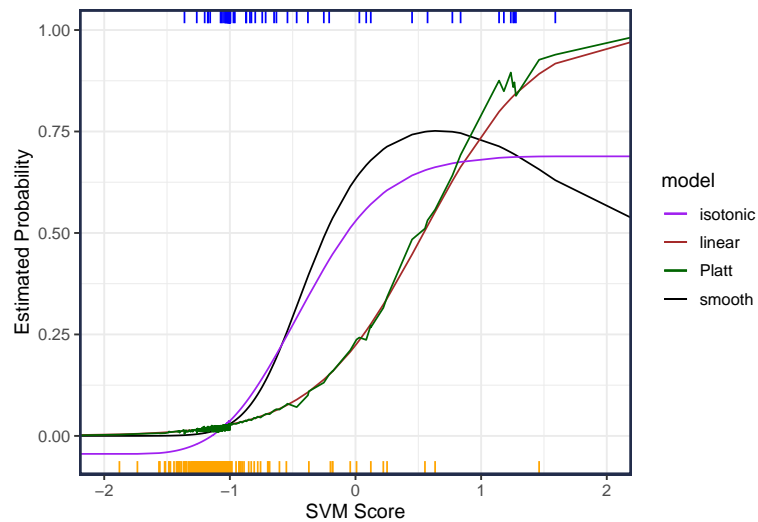
#### 4.2.4 Built-in Calibration

Set `prob.model=TRUE` in the `ksvm()` function to get probability estimates. Note that this uses 3-fold cross-validation to predict out-of-sample scores. Then uses weighted logistic regression to estimate the calibration function.

```
library(kernlab)
# Radial basis kernel
fit_svm = ksvm(default~student + balance + income,
               data = Default[train, ],
               prob.model = TRUE, # ** Need to set to TRUE to get probabilities
               scaled = TRUE,    # be sure to scale the predictors
               kernel = "rbfdot", # kernel
               sigma = .01, C = 100, # tuning parameters
               )

# table of predictions and true values
tbl_svm = tbl_svm %>%
  mutate(
    p_ksvm = predict(fit_svm, ., type = "probabilities")[, "Yes"]
  )
```

```
tbl_svm %>%
  ggplot(aes(score)) +
  geom_line(aes(y = p_smooth, color = "smooth")) +
  geom_line(aes(y = p_hat, color = "linear")) +
  geom_line(aes(y = p_iso, color = "isotonic")) +
  geom_line(aes(y = p_ksvm, color = "Platt")) +
  geom_rug(data = . %>% filter(y==0), color = "orange", sides="b") +
  geom_rug(data = . %>% filter(y==1), color = "blue", sides="t") +
  scale_color_manual(name = "model",
    values=c(smooth="black", linear="brown",
             isotonic = "purple", Platt="darkgreen")) +
  labs(x = "SVM Score", y = "Estimated Probability") +
  scale_x_continuous(breaks = seq(-5, 5, by=1)) +
  coord_cartesian(xlim = c(-2, 2))
```



It matches pretty closely to just implementing logistic regression on the in-sample data.