Calibrating Risk Scores

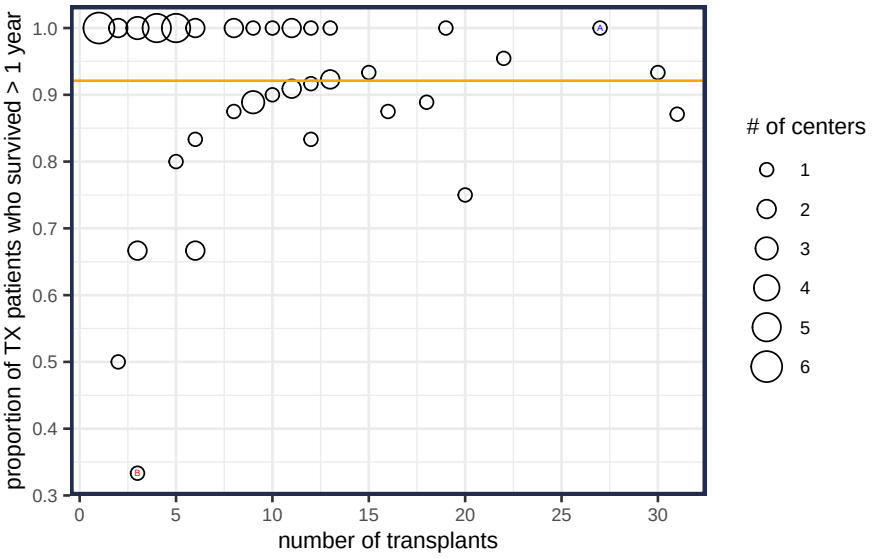
DS-6030 | Spring 2026

calibration.pdf

Table of contents

1 Shrinkage Estimator and Laplace Smoothing

1.1 Transplant Center Performance



In 2019, there were 507 pediatric heart transplants performed in the US. The overall 1-year survival rate was 92.1% (467).

These transplants were performed at 59 different transplant centers. The highest survival center (A) had 100.0% survival (27/27). The center with lowest survival (B) had 33.3% survival (1/3).

Your Turn #1

What is your estimate for both centers' 2020 survival rate?

- Center A:
- Center B:

1.1.1 Laplace (Additive) Smoothing

If we used the empirical proportions (i.e., maximum likelihood point estimates), we would predict 2020 survival as $\hat{p} = 1.000$ for the best center and $\hat{p} = 0.333$ for the worst.

Ignoring the potential of trending performance, regression to the mean suggests that our best and worst performers will probably have closer to the overall survival rate.

Laplace Smoothing

2 Calibrating Risk Scores

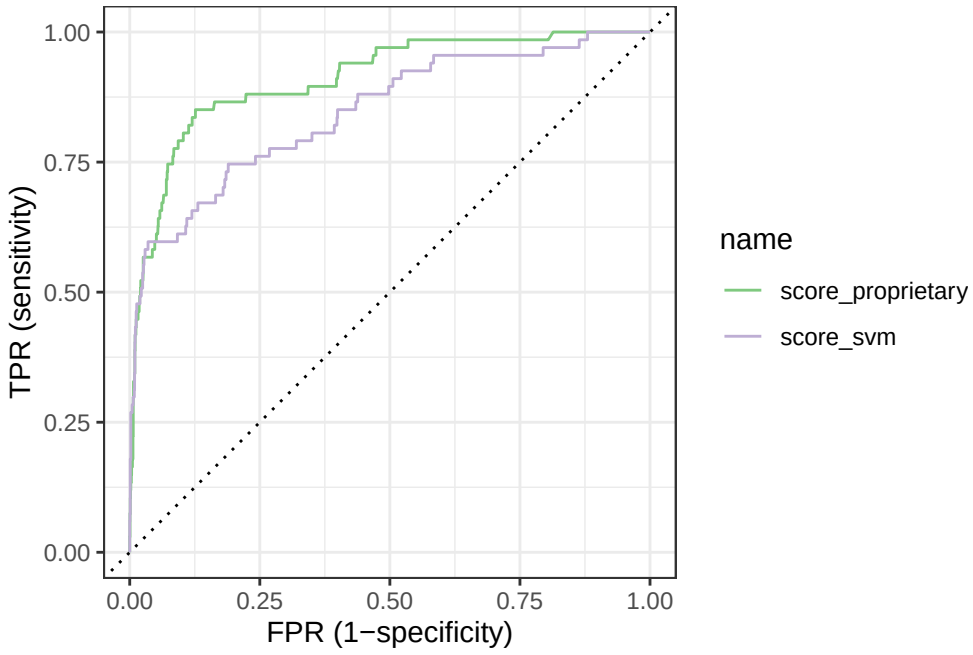
2.1 Data

Here is a sample of the `Default` data. There are two risk scores. One from a SVM and the other from a proprietary model. We are assuming that *higher risk score implies higher risk of default*.

default	y	student	balance	income	score_svm	score_proprietary
No	0	No	1077.0	33071	-1.356	0.014
No	0	Yes	1281.6	14236	-1.461	0.024
No	0	No	427.3	27261	-1.203	0.013
No	0	No	172.0	33505	-1.078	0.012
Yes	1	No	1170.2	46692	-1.334	0.016
Yes	1	Yes	2117.1	12143	-0.171	0.721
Yes	1	Yes	2334.1	19336	0.519	0.809
Yes	1	Yes	1871.9	18077	-0.762	0.408

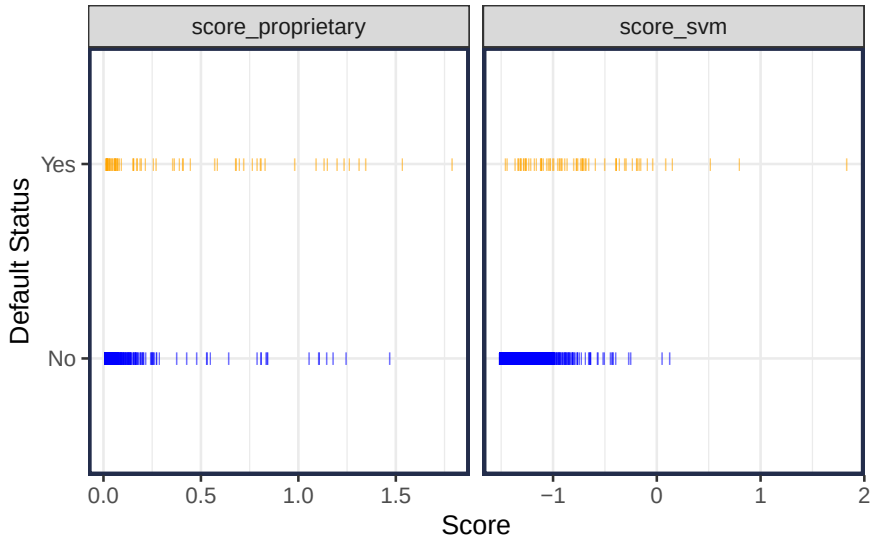
2.2 AUC

The area under the ROC curve (AUROC/AUC) can be created with any score that ranks individual risk.



model	AUC
proprietary	0.910
svm	0.846

2.3 Risk Score Calibration



The key idea is to find a function that converts the risk score to a calibrated probability.

- Data: $\{(s_i, y_i)\}$
- Goal: Estimate $\Pr(Y = 1 \mid s)$.
- This is just a binary regression with a single predictor.

Your Turn #2

What are some ways to convert a risk score to a probability?

2.4 Approach 1: Logistic Regression

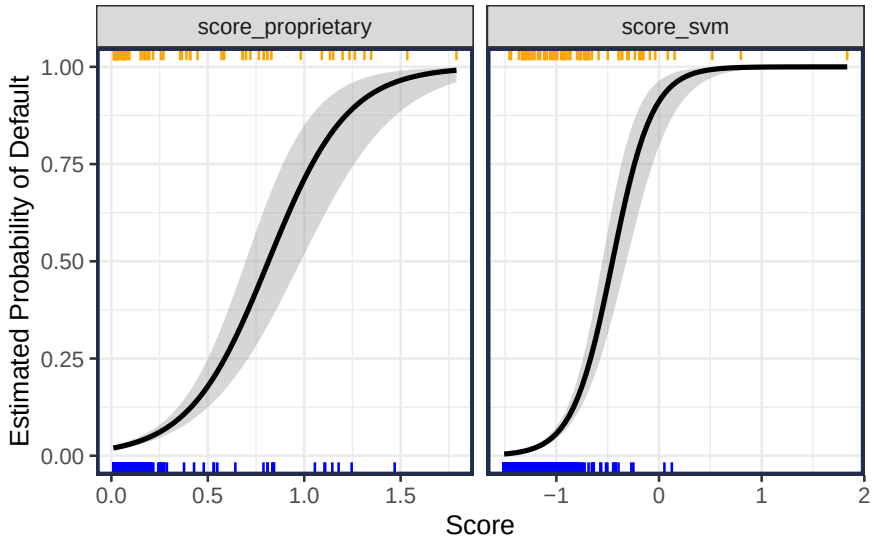
Just like in Platt scaling for SVM, we can use logistic regression to convert an arbitrary risk score to a probability.

- Note: we assume the slope coefficient is positive. If not, go back and check your risk scores.

$$\text{logit}(\hat{p}(x)) = \hat{\beta}_0 + \hat{\beta}_1 s(x)$$
$$\hat{p}(x) = 1/\left(1 + e^{-(\hat{\beta}_0 + \hat{\beta}_1 s(x))}\right)$$

term	estimate	std.error	statistic	p.value
(Intercept)	2.330	0.493	4.723	0
score_svm	5.141	0.465	11.058	0

term	estimate	std.error	statistic	p.value
(Intercept)	-3.952	0.161	-24.61	0
score_proprietary	4.852	0.472	10.28	0



Your Turn #3

How flexible/complex is the logistic regression calibration model?

2.5 Approach 2: Binning/Ordinal

- Sometimes risk scores come on the ordinal scale, e.g., $\{1, 2, ..., 10\}$ or {low, medium, high}.
- Or to get a more flexible relationship, we can bin (i.e., discretize) a continuous risk score.

Let's partition the scores such that there are 10 groups of equal width.

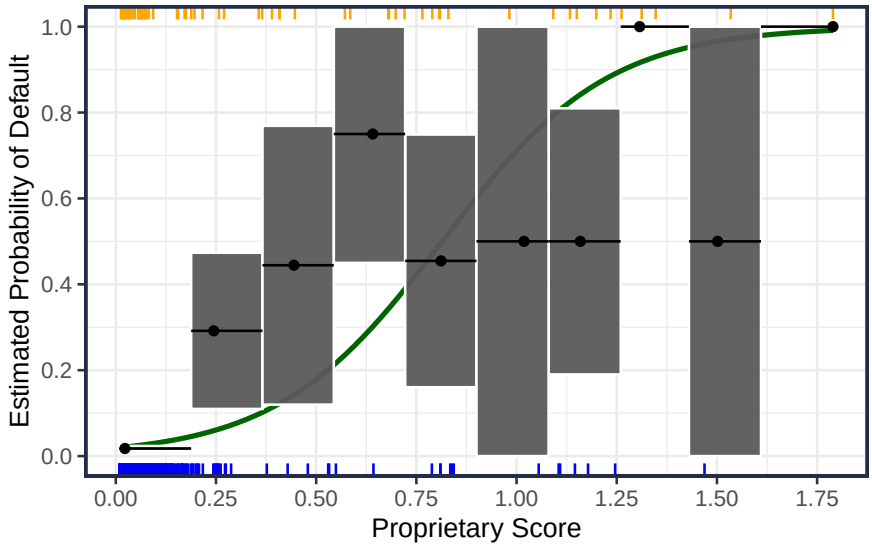
Here's a sample of the data:

default	y	student	balance	income	score_svm	score_bin
No	0	No	1077.0	33071	-1.356	[-1.51,-1.17]
No	0	Yes	1281.6	14236	-1.461	[-1.51,-1.17]
No	0	No	427.3	27261	-1.203	[-1.51,-1.17]
No	0	No	172.0	33505	-1.078	(-1.17,-0.841]
Yes	1	No	1170.2	46692	-1.334	[-1.51,-1.17]
Yes	1	Yes	2117.1	12143	-0.171	(-0.172,0.162]
Yes	1	Yes	2334.1	19336	0.519	(0.496,0.831]
Yes	1	Yes	1871.9	18077	-0.762	(-0.841,-0.506]

The following show the naive estimates.

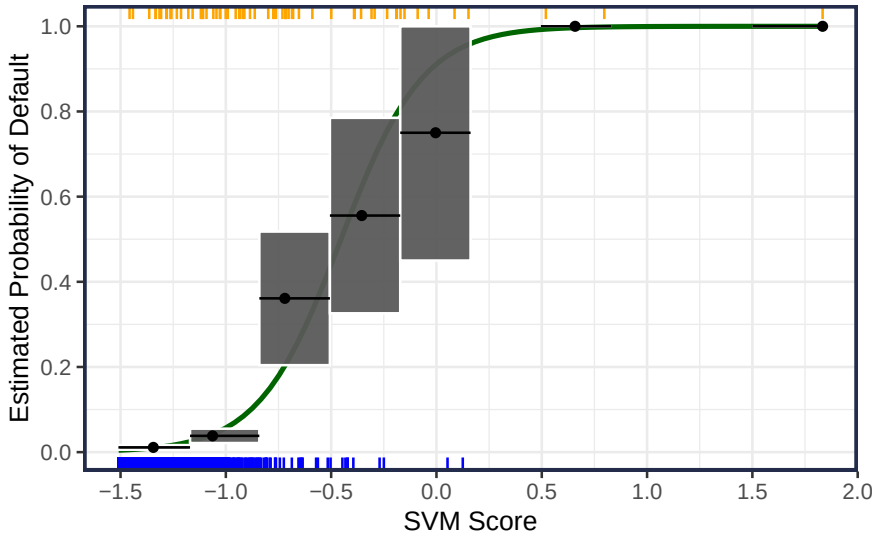
2.5.1 Proprietary Score

bin	score_bin	bin_lower	bin_upper	bin_avg	n	n_1	p_hat	moe
1	[0.0102,0.188]	0.010	0.188	0.023	1930	34	0.018	0.006
2	(0.188,0.366]	0.188	0.366	0.244	24	7	0.292	0.182
3	(0.366,0.544]	0.366	0.544	0.444	9	4	0.444	0.325
4	(1.08,1.26]	1.080	1.260	1.159	10	5	0.500	0.310
5	(1.26,1.43]	1.260	1.430	1.307	3	3	1.000	0.000
6	(0.722,0.9]	0.722	0.900	0.812	11	5	0.455	0.294
7	(0.544,0.722]	0.544	0.722	0.641	8	6	0.750	0.300
8	(0.9,1.08]	0.900	1.080	1.019	2	1	0.500	0.693
9	(1.43,1.61]	1.430	1.610	1.502	2	1	0.500	0.693
10	(1.61,1.79]	1.610	1.790	1.790	1	1	1.000	0.000



2.5.2 SVM Score

bin	score_bin	bin_lower	bin_upper	bin_avg	n	n_1	p_hat	moe
1	[-1.51,-1.17]	-1.510	-1.170	-1.344	1438	16	0.011	0.005
2	(-1.17,-0.841]	-1.170	-0.841	-1.063	497	19	0.038	0.017
3	(-0.841,-0.506]	-0.841	-0.506	-0.720	36	13	0.361	0.157
4	(-0.506,-0.172]	-0.506	-0.172	-0.355	18	10	0.556	0.230
5	(-0.172,0.162]	-0.172	0.162	-0.004	8	6	0.750	0.300
6	(0.496,0.831]	0.496	0.831	0.658	2	2	1.000	0.000
7	(1.5,1.83]	1.500	1.830	1.833	1	1	1.000	0.000



2.5.3 Laplace Smoothing for small bin counts

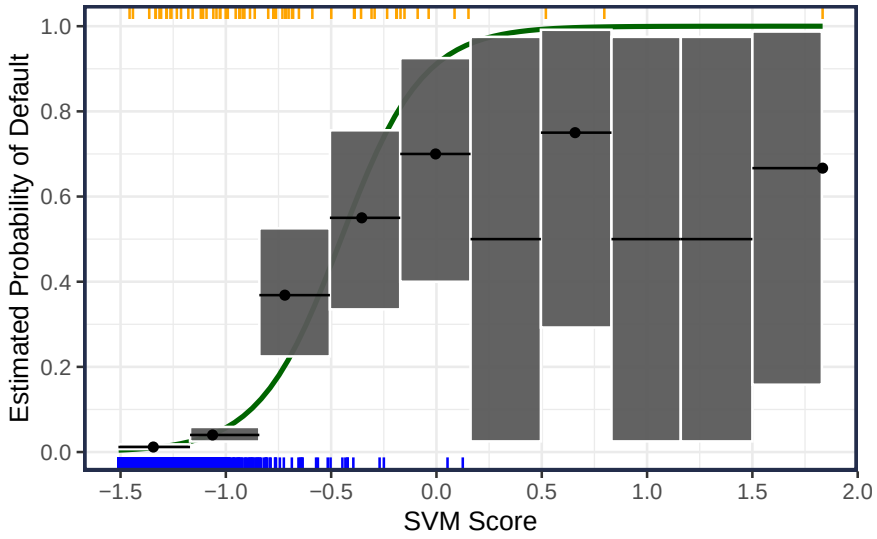
Notice that some of the bins have very few counts. We can use the Laplace Smoothing to regularize the probability estimates.

Let's take the svm scores and add two pseudo observations in each bin: 1 default, 1 non-default.

```
#: get bin limits from cut() object
bin_limits <- function(x, which = "lower"){
  which = match.arg(which, c("lower", "upper"))
  pattern = ".{1}(.+),(.+).{1}" # regex pattern
  if(which == "lower") y = stringr::str_extract(x, pattern , group = 1)
  if(which == "upper") y = stringr::str_extract(x, pattern , group = 2)
  return(as.numeric(y))
}
```

```
#: #: bin the scores (10 equal width bins)
tbl_binning =
  tbl %>%
  mutate(
    score = score_svm,
    score_bin = ggplot2::cut_interval(score, n = 10)
  )

#: get counts in bin
tbl_counts =
 tbl_binning %>%
  group_by(score_bin, .drop = FALSE) %>% # group by score_bin
  summarize(
    bin_avg = mean(score),# average score in bin
    n = n(),              # number of observations in bin
    n_1 = sum(y),         # number of defaults (Y=1) in bin
  ) %>%
  #: add bin ranges
  mutate(
    bin_lower = bin_limits(score_bin, "lower"),
    bin_upper = bin_limits(score_bin, "upper"),
    .before = bin_avg
  )

#: estimated probability and uncertainty intervals
tbl_risk =
  tbl_counts %>%
  mutate(

    #: Frequentist
    p_bar = n_1 / n,
    moe = 1.96 * sqrt(p_bar * (1-p_bar) / n),

    #: Bayesian (uniform prior for each bin)
    p_hat = (n_1 + 1) / (n + 2), # posterior mean
    beta_lower = qbeta(.025, n_1 + 1, n - n_1 + 1),
    beta_upper = qbeta(.975, n_1 + 1, n - n_1 + 1),

  ) %>%
  mutate(
    bin = 1:n(),
    .before = 1
  )
```

bin	score_bin	bin_lower	bin_upper	n	n_1	p_bar	moe	p_hat	beta_lower	beta_upper
1	[-1.51,-1.17]	-1.510	-1.170	1438	16	0.011	0.005	0.012	0.007	0.018
2	(-1.17,-0.841]	-1.170	-0.841	497	19	0.038	0.017	0.040	0.025	0.059
3	(-0.841,-0.506]	-0.841	-0.506	36	13	0.361	0.157	0.368	0.225	0.525
4	(-0.506,-0.172]	-0.506	-0.172	18	10	0.556	0.230	0.550	0.335	0.756
5	(-0.172,0.162]	-0.172	0.162	8	6	0.750	0.300	0.700	0.400	0.925
6	(0.162,0.496]	0.162	0.496	0	0	NaN	NaN	0.500	0.025	0.975
7	(0.496,0.831]	0.496	0.831	2	2	1.000	0.000	0.750	0.292	0.992
8	(0.831,1.16]	0.831	1.160	0	0	NaN	NaN	0.500	0.025	0.975
9	(1.16,1.5]	1.160	1.500	0	0	NaN	NaN	0.500	0.025	0.975
10	(1.5,1.83]	1.500	1.830	1	1	1.000	0.000	0.667	0.158	0.987



2.6 Approach 3: Splines

So far we have treated the score as either:

- a linear predictor in a logistic regression, or
- a set of bins with empirical probabilities.

Both impose a structure:

- Logistic(score) imposes linearity in the log-odds.
- Binning imposes a step function (piecewise-constant).

A more flexible approach is to estimate:

$\Pr(Y = 1|score) = g(score)$

or

$\log \frac{\Pr(Y=1|score)}{\Pr(Y=0|score)} = g(score)$

where $g()$ is a smooth function learned from the data rather than imposed by the modeling. Splines are a great way to do this. Splines can capture non-linear patterns smoothly (instead of binning and step functions).

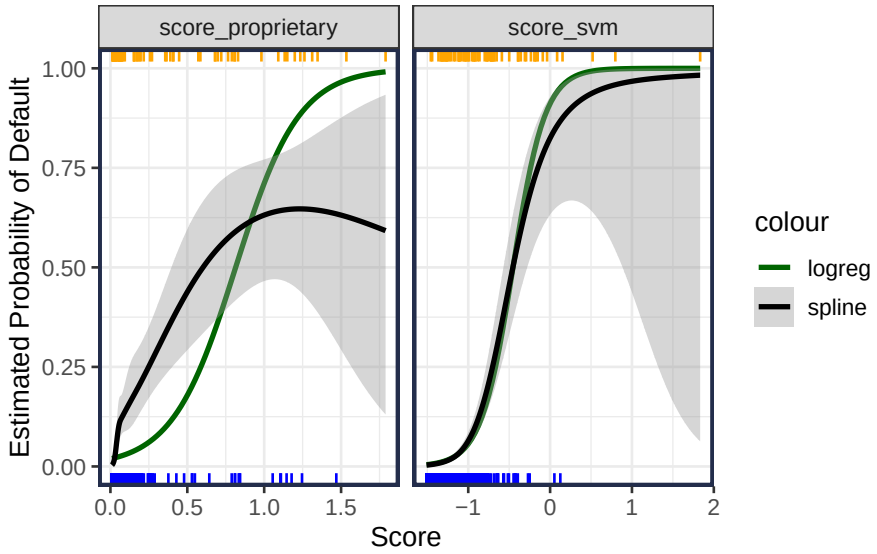
2.6.1 Unrestricted Splines

In R (using `mgcv`)

```r
library(mgcv)

# Assume:
# score = numeric vector
# y     = binary outcome (0/1)

# Fitting
fit = mgcv::gam(y ~ s(score),
           family = binomial(link = "logit"))

# Predicted probabilities
p_hat = predict(fit, type = "response")
```

In python (using `pygam`)

```python
from pygam import LogisticGAM, s
import numpy as np

# Assume:
# score = 1D numpy array
# y     = binary array (0/1)

# Pre-processing (2D array with n rows and 1 column)
X = score[:, None]

# Fitting
gam = LogisticGAM(s(0)).fit(X, y)

# Predicted probabilities
p_hat = gam.predict_proba(X)
```



Your Turn #4

What can do wrong with unrestricted splines? Hint: compare the risk for someone with proprietary score = 2.0 to someone with score of 1.0.

2.6.2 Isotonic Splines

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

- monotonic means that as the risk score increases the probability will increase (technically, the probability will not decrease).

In R (using `scam`):

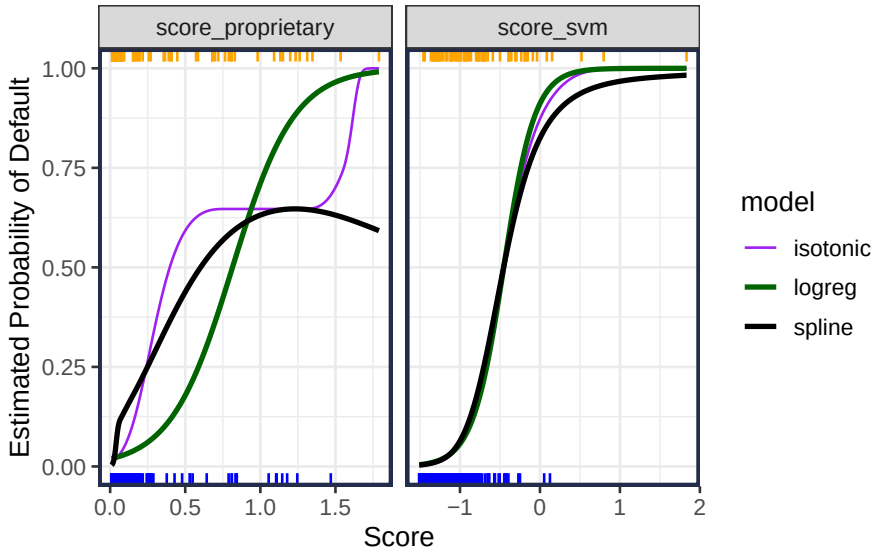
```r
# Assume:
# score = numeric vector
# y     = binary outcome (0/1)

# Fitting
library(scam)
iso = scam(y ~ s(score, bs="miso"),
           family=binomial(link = "logit"))

# Predicted probabilities
p_hat = predict(iso, type = "response")
```

In python (using `sklearn.isotonic`):

```python
from sklearn.isotonic import IsotonicRegression
import numpy as np

# score: 1D numpy array
# y: binary (0/1)

# Fitting
iso = IsotonicRegression(out_of_bounds="clip")

# Predicted probabilities
p_hat = iso.fit_transform(score, y)
```

In python (using `pygam`)

```python
from pygam import LogisticGAM, s
import numpy as np

# Pre-processing (2D array with n rows and 1 column)
X = score[:, None]

# Fitting
gam = LogisticGAM(
    s(0, constraints='monotonic_inc')
).fit(X, y)

# Predicted probabilities
p_hat = gam.predict_proba(X)
```



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.

2.7 Evaluating a Calibrated Probability Risk Score

- Use *training data* to get risk scores
- Use *calibration data* to get calibrated probabilities
- Use *test data* to evaluate probabilities
  - Evaluate probabilities with log-loss or brier scores.