Classification

DS 6030 | Fall 2023

classification.pdf

Contents

1 Classification Intro

1.1 Set-up

- The outcome variable is categorical and denoted $G \in \mathcal{G}$
  - Default Credit Card Example: $\mathcal{G} = \{$"Yes", "No"$\}$
  - Medical Diagnosis Example: $\mathcal{G} = \{$"stroke", "heart attack", "drug overdose", "vertigo"$\}$
- The training data is $D = \{(X_1, G_1), (X_2, G_2), \ldots, (X_n, G_n)\}$
- The optimal decision/classification is often based on the posterior probability $\Pr(G = g \mid \mathbf{X} = \mathbf{x})$

1.2 Binary Classification

- Classification is simplified when there are only 2 classes.
  - Many multi-class problems can be addressed by solving a set of binary classification problems (e.g., one-vs-rest).
- It is often convenient to transform the outcome variable to a binary $\{0, 1\}$ variable:

$$Y_i = \begin{cases} 1 & G_i = \mathcal{G}_1 \quad \text{(outcome of interest)} \\ 0 & G_i = \mathcal{G}_2 \end{cases}$$

- In the `Default` data, it would be natural to set `default=Yes` to 1 and `default=No` to 0.

2 Risk Scoring vs. Classification

Most of the models we will encounter can output a predicted probability $\hat{p}_k(x) = \widehat{\Pr}(Y = k \mid X = x)$ for every class $k \in \mathcal{G}$.

Sometimes a *hard classification* needs to be made, i.e., decide on single label/class to assign the observation.

1. Hard Classification:
   - Use training data to estimate the *label* $\hat{G}(X)$
   - The loss/cost $L(G, \hat{G}(X))$ is the loss incurred by estimating G with $\hat{G}$
2. Risk Scoring (Soft-Classification):
   - Use training data to estimate the *probability* $\hat{p}_k(X)$
   - The loss/cost $L(G, \hat{p}(X))$ is the loss incurred by estimating G with $\hat{p}_k(X)$, where $\hat{p}(X) = [\hat{p}_1(X), \ldots, \hat{p}_K(X)]$
3. Ranking:
   - Use the training data to rank the test observations according to estimated risk level.
   - The loss/cost is based on the number of outcomes of interest in the top proportion of risk.

2.0.1 Example: Recidivism Prediction

Recently the National Institute of Justice hosted a Recidivism Forecasting Challenge which challenged contestants to predict if a parolee would be arrested for another offense within the next few years. The motivation is not to determine who should be released on parole, but rather which parolees should get additional assistance/supervision.

Objective	Model Output
Classification	Predict {Yes, No} for re-offending
Scoring	Predict probability of re-offending
Ranking	Order from highest risk level to lowest

Your Turn #1 : Recidivism Prediction

1. How could you use the probability/score to make a hard classification?
2. Do you think a hard classification or probability/score is better for this scenario?
3. If there were limited resources (e.g., only N parolees could get extra assistance), which type of model output would be more useful?

3 Binary Classification

3.1 Decision Theory

- We are considering *binary* outcomes, so the outcomes $G \in \{0, 1\}$
- Let $p(x) = \Pr(G = 1 \mid X = x)$
- Loss Function: $L(\text{True Label}, \text{Estimated Label}) = L(G, \hat{G})$

Loss	Description	Name
$L(G = 0, \hat{G} = 0)$	True class is 0, Predicted class is 0	True Negative
$L(G = 1, \hat{G} = 1)$	True class is 1, Predicted class is 1	True Positive
$L(G = 0, \hat{G} = 1)$	True class is 0, Predicted class is 1	False Positive
$L(G = 1, \hat{G} = 0)$	True class is 1, Predicted class is 0	False Negative

- A model's *Expected Prediction Error (EPE)* at input X is the expected loss on new data with input X.
- The EPE (for a binary outcome) is:

$$\begin{aligned}
\text{EPE}_x(g) &= \text{E}_{G|X=x}\left[L(G, \hat{G}(x) = g) \mid X = x\right] \\
&= L(1, g)\Pr(G = 1 \mid X = x) + L(0, g)(1 - \Pr(G = 1 \mid X = x)) \\
&= L(1, g)p(x) + L(0, g)(1 - p(x))
\end{aligned}$$

- Hard Decision ($\hat{G}(x) \in \{0, 1\}$): choose $\hat{G}(x) = 1$ if

$$\begin{aligned}
\text{EPE}_x(1) &< \text{EPE}_x(0) \\
L(1,1)p(x) + L(0,1)(1 - p(x)) &< L(1,0)p(x) + L(0,0)(1 - p(x)) \\
p(x)\left(L(1,1) - L(1,0)\right) &< (1 - p(x))\left(L(0,0) - L(0,1)\right) \\
p(x)\left(L(1,0) - L(1,1)\right) &\geq (1 - p(x))\left(L(0,1) - L(0,0)\right) \qquad \textit{(multiply both sides by -1)} \\
\frac{p(x)}{1 - p(x)} &\geq \frac{L(0,1) - L(0,0)}{L(1,0) - L(1,1)}
\end{aligned}$$

Note

In most cases, there will be no loss/cost for making a correct classification. Thus it is convention to set $L(0, 0) = L(1, 1) = 0$ in these scenarios.

3.1.1 Example: Cancer Diagnosis

- Say we have a goal of estimating if a patient has cancer using medical imaging
  - Let $G = 1$ for cancer and $G = 0$ for no cancer
- Suppose we have solicited a loss function with the following values
  - $L(G = 0, \hat{G} = 0) = 0$: There is no loss for correctly diagnosis a patient without cancer.
  - $L(G = 1, \hat{G} = 1) = 0$: There is no loss (for our model) for correctly diagnosis a patient with cancer.

- $L(G=0, \hat{G}=1) = C_{\text{FP}}$: There is a cost of C_{FP} units if the model issues a *false positive*, estimating the patient has cancer when they don't.
- $L(G=1, \hat{G}=0) = C_{\text{FN}}$: There is a cost of C_{FN} units if the model issues a *false negative*, estimating the patient does not have cancer when they really do.
- In these scenarios C_{FN} is often much larger than C_{FP} ($C_{\text{FN}} >> C_{\text{FP}}$) because the the effects of not promptly treating (or further testing, etc) a patient is more severe than starting a treatment path for patients that don't actually have cancer.

• The optimal decision is to issue a positive indication for cancer if EPE$_x(1) <$ EPE$_x(0)$. This occurs when

$$\frac{p(x)}{1-p(x)} \geq \frac{C_{\text{FP}}}{C_{\text{FN}}} \quad \text{OR} \quad p(x) \quad \geq \frac{C_{\text{FP}}}{C_{\text{FP}} + C_{\text{FN}}} \quad \text{OR} \quad \log\left(\frac{p(x)}{1-p(x)}\right) \geq \log\left(\frac{C_{\text{FP}}}{C_{\text{FN}}}\right)$$

• The ratio of C_{FP} to C_{FN} is all that matters for the decision. Let's say that $C_{\text{FP}} = 1$ and $C_{\text{FN}} = 10$. Then if $p(x) \geq 1/11$, our model will diagnose cancer.

- Note: $p(x) = \Pr(Y=1|X=x)$ is affected by the class prior $\Pr(Y=1)$ (e.g., the portion of the population tested who have cancer), which is usually going to be small.

3.1.2 Optimal Threshold

• Recall, the optimal *hard classification* decision is to choose $\hat{G} = 1$ if:

$$\frac{p(x)}{1-p(x)} \geq \frac{L(0,1) - L(0,0)}{L(1,0) - L(1,1)}$$

• It can be convenient to use model output other than $p(x)/(1-p(x))$ to make decisions

• Some models directly output $\hat{p}(x)$

• Other models, like GLMs, naturally work with the link function (linear part)

- Denote $\gamma(x)$ as the *logit* of $p(x)$:

$$\gamma(x) = \log\frac{p(x)}{1-p(x)} = \log\frac{\Pr(G=1 \mid X=x)}{\Pr(G=0 \mid X=x)}$$

• Table of equivalent representations:

Score	Threshold	Threshold (simplified)
$\frac{p(x)}{1-p(x)}$	$\frac{L(0,1) - L(0,0)}{L(1,0) - L(1,1)}$	$\frac{C_{\text{FP}}}{C_{\text{FN}}}$
$\gamma(x) = \log\frac{p(x)}{1-p(x)}$	$\log\left(\frac{L(0,1) - L(0,0)}{L(1,0) - L(1,1)}\right)$	$\log\left(\frac{C_{\text{FP}}}{C_{\text{FN}}}\right)$
$p(x)$	$\log\left(\frac{L(0,1) - L(0,0)}{L(0,1) - L(0,0) + L(1,0) - L(1,1)}\right)$	$\frac{C_{\text{FP}}}{C_{\text{FP}} + C_{\text{FN}}}$

The *Threshold (simplified)* assumes $L(0,0) = L(1,1) = 0$

3.1.3 Using estimated values

- We will never have the actual $p(x)$ or $\gamma(x)$, so replace them with the estimated values.
- For a given threshold t and input x, the hard classification rule is $\hat{G}_t(x) = \mathbb{1}(\hat{\gamma}(x) \geq t)$

Note

Because we have to estimate $\hat{p}(x)$ or $\hat{\gamma}(x)$, the best threshold t^* may differ from the theoretical optimal and need to be estimated. (more info about this below)

3.2 Common Binary Loss Functions

- Setting: estimate a binary outcome $G \in \{0, 1\}$ with a predicted label $\hat{G}(x)$
- **Mis-Classification Cost**

$$L(G, \hat{G}(x)) = \begin{cases} C_{\text{FP}} & G = 0, \hat{G}(x) = 1 \\ C_{\text{FN}} & G = 1, \hat{G}(x) = 0 \\ 0 & \text{otherwise} \end{cases}$$

  - This requires that a *hard classification* is made.
  - The theoretically optimal prediction is:

$$G^*(x) = \mathbb{1}(p(x) > C_{\text{FP}}/(C_{\text{FP}} + C_{\text{FN}}))$$

- **0-1 Loss** or **Misclassification Error**

$$L(G, \hat{G}(x)) = \mathbb{1}(y \neq \hat{G}(x)) = \begin{cases} 0 & G = \hat{G}(x) \\ 1 & G \neq \hat{G}(x) \end{cases}$$

  - This assumes $L(0, 1) = L(1, 0)$ (i.e., false positive costs the same as a false negative)
  - This requires that a *hard classification* is made.
  - The theoretically optimal prediction is:

$$\begin{aligned} G^*(x) &= \mathbb{1}(p(x) > 1 - p(x)) \\ &= \mathbb{1}(p(x) > 0.50) \end{aligned}$$

3.3 Performance Metrics

3.3.1 Confusion Matrix

- Given a threshold t, we can make a *confusion matrix* to help analyze our model's performance on data
  - Data = $\{(X_i, G_i)\}_{i=1}^N$ (ideally this is hold-out/test data)
  - N_k is number of observations from class k ($N_0 + N_1 = N$)

		True Outcome		
		$G = 1$	$G = 0$	**total**
Model Outcome	$\hat{G}_t = 1$	True Positive (TP)	False Positive (FP)	$\hat{N}_1(t)$
	$\hat{G}_t = 0$	False Negative (FN)	True Negative (TN)	$\hat{N}_0(t)$
	total	N_1	N_0	N

Table from: https://tex.stackexchange.com/questions/20267/how-to-construct-a-confusion-matrix-in-latex

To illustrate a confusion table in practice let's go back to the `Default` data and see how the basic logistic regression models performs.

- In order to evaluate on hold-out data, split the data into train/test (used 8000 training, 2000 testing), fit a logistic regression model on training data, and make predictions on the test data
- Note that only 3.3% of the data is default.
  - Using a threshold of $\hat{p}(x) \geq 0.10$ to make a hard classification.
  - Equivalent to $\hat{\gamma}(x) \geq \log(.10) - \log(1 - .10) = -2.1972$

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

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

#: Get predictions (of p(x)) on test data
p_hat = predict(fit.lm, Default[test, ], type='response')

#: Make Hard classification (use .10 as cut-off)
G.hat = ifelse(p_hat >= .10, 1, 0)

#: Make Confusion Table
G.test = Default$y[test]   # true values

table(predicted=G.hat, truth = G.test) %>% addmargins()
#>          truth
#> predicted    0    1  Sum
#>       0   1805   17 1822
#>       1    128   50  178
#>       Sum 1933   67 2000
```

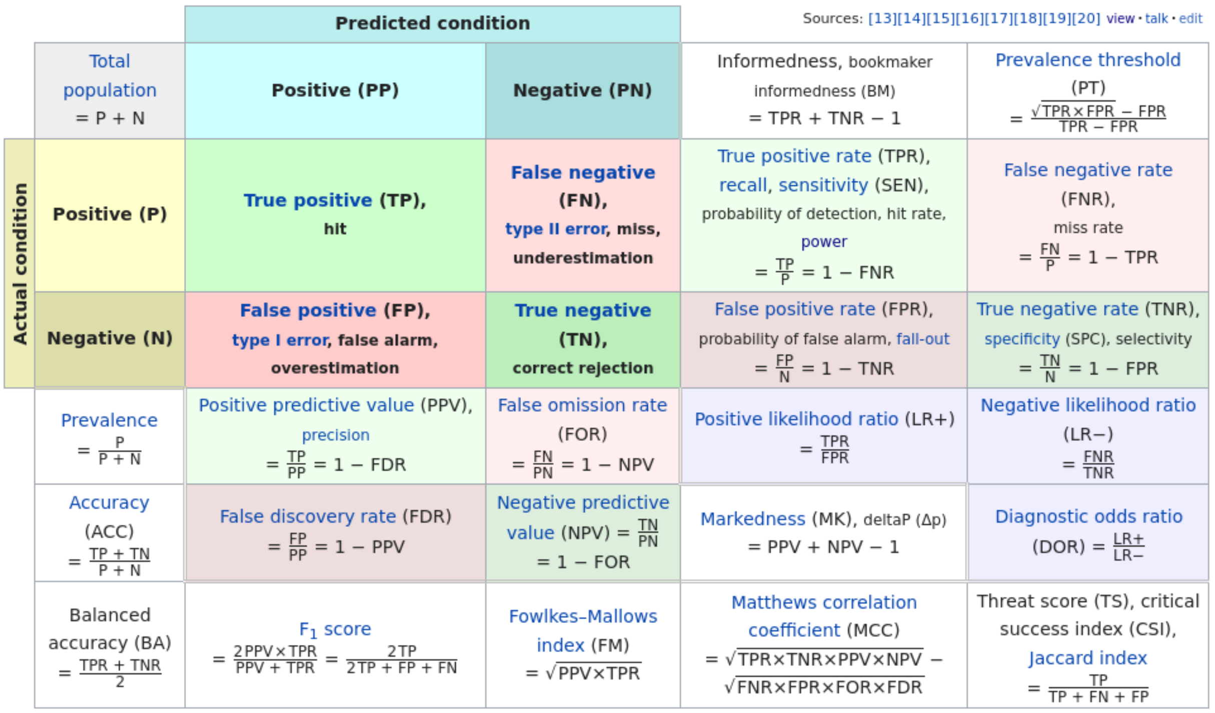
3.3.2 Metrics

There are several standard evaluation metrics that can be calculated from the confusion matrix:

Metric	Definition	Estimate
Expected Cost	$\sum_{i=0}^{1}\sum_{j=0}^{1} L(i,j) P_X(G(X)=i, \hat{G}_t(X)=j)$	$\frac{1}{N}\sum_{i=1}^{N} L(G_i, \hat{G}_t(x_i))$
Mis-classification Rate	$P_{XG}(\hat{G}_t(X) \neq G(X)) = P_X(\hat{G}_t(X)=0, G(X)=1) + P_X(\hat{G}_t(X)=1, G(X)=0)$	$\frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(\hat{G}_t(x_i) \neq G_i)$
False Positive Rate (FPR) {1-*Specificity*}	$P_X(\hat{G}_t(X)=1 \mid G(X)=0)$	$\frac{1}{N_0}\sum_{i:G_i=0} \mathbb{1}(\hat{G}_t(x_i)=1)$
True Positive Rate (TPR) {*Hit Rate, Recall, Sensitivity*}	$P_X(\hat{G}_t(X)=1 \mid G(X)=1)$	$\frac{1}{N_1}\sum_{i:G_i=1} \mathbb{1}(\hat{G}_t(x_i)=1)$
Precision TP/(TP + FP)	$P_X(G(X)=1 \mid \hat{G}_t(X)=1)$	$\frac{1}{\hat{N}_1(t)}\sum_{i:\hat{G}(x_i)=1} \mathbb{1}(G_i=1)$

N is the total number of predictions/observations, N_0 is the number of true class 0's in the data ($N_0 = \sum_{i=1}^{N} \mathbb{1}(y_i = 0)$), N_1 is the number of true class 1's in the data ($N_1 = \sum_{i=1}^{N} \mathbb{1}(y_i = 1)$), $\hat{N}_1(t)$ is the number of *predicted* class 1's using a threshold of t ($\hat{N}_1(t) = \sum_{i=1}^{n} \mathbb{1}(\hat{p}_i \geq t)$).

- Note: Performance estimates are best carried out on *hold-out* data!
- See Wikipedia Page: Confusion Matrix for more metrics:



		Predicted condition			Sources: [13][14][15][16][17][18][19][20] view · talk · edit
	Total population = P + N	**Positive (PP)**	**Negative (PN)**	Informedness, bookmaker informedness (BM) = TPR + TNR − 1	Prevalence threshold (PT) = $\frac{\sqrt{\text{TPR} \times \text{FPR}} - \text{FPR}}{\text{TPR} - \text{FPR}}$
Actual condition	**Positive (P)**	**True positive (TP), hit**	**False negative (FN), type II error, miss, underestimation**	True positive rate (TPR), recall, sensitivity (SEN), probability of detection, hit rate, power = $\frac{\text{TP}}{\text{P}}$ = 1 − FNR	False negative rate (FNR), miss rate = $\frac{\text{FN}}{\text{P}}$ = 1 − TPR
	Negative (N)	**False positive (FP), type I error, false alarm, overestimation**	**True negative (TN), correct rejection**	False positive rate (FPR), probability of false alarm, fall-out = $\frac{\text{FP}}{\text{N}}$ = 1 − TNR	True negative rate (TNR), specificity (SPC), selectivity = $\frac{\text{TN}}{\text{N}}$ = 1 − FPR
	Prevalence = $\frac{\text{P}}{\text{P} + \text{N}}$	Positive predictive value (PPV), precision = $\frac{\text{TP}}{\text{PP}}$ = 1 − FDR	False omission rate (FOR) = $\frac{\text{FN}}{\text{PN}}$ = 1 − NPV	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Negative likelihood ratio (LR−) = $\frac{\text{FNR}}{\text{TNR}}$
	Accuracy (ACC) = $\frac{\text{TP} + \text{TN}}{\text{P} + \text{N}}$	False discovery rate (FDR) = $\frac{\text{FP}}{\text{PP}}$ = 1 − PPV	Negative predictive value (NPV) = $\frac{\text{TN}}{\text{PN}}$ = 1 − FOR	Markedness (MK), deltaP (Δp) = PPV + NPV − 1	Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR−}}$
	Balanced accuracy (BA) = $\frac{\text{TPR} + \text{TNR}}{2}$	F_1 score = $\frac{2 \text{PPV} \times \text{TPR}}{\text{PPV} + \text{TPR}} = \frac{2 \text{TP}}{2 \text{TP} + \text{FP} + \text{FN}}$	Fowlkes–Mallows index (FM) = $\sqrt{\text{PPV} \times \text{TPR}}$	Matthews correlation coefficient (MCC) = $\sqrt{\text{TPR} \times \text{TNR} \times \text{PPV} \times \text{NPV}} - \sqrt{\text{FNR} \times \text{FPR} \times \text{FOR} \times \text{FDR}}$	Threat score (TS), critical success index (CSI), Jaccard index = $\frac{\text{TP}}{\text{TP} + \text{FN} + \text{FP}}$

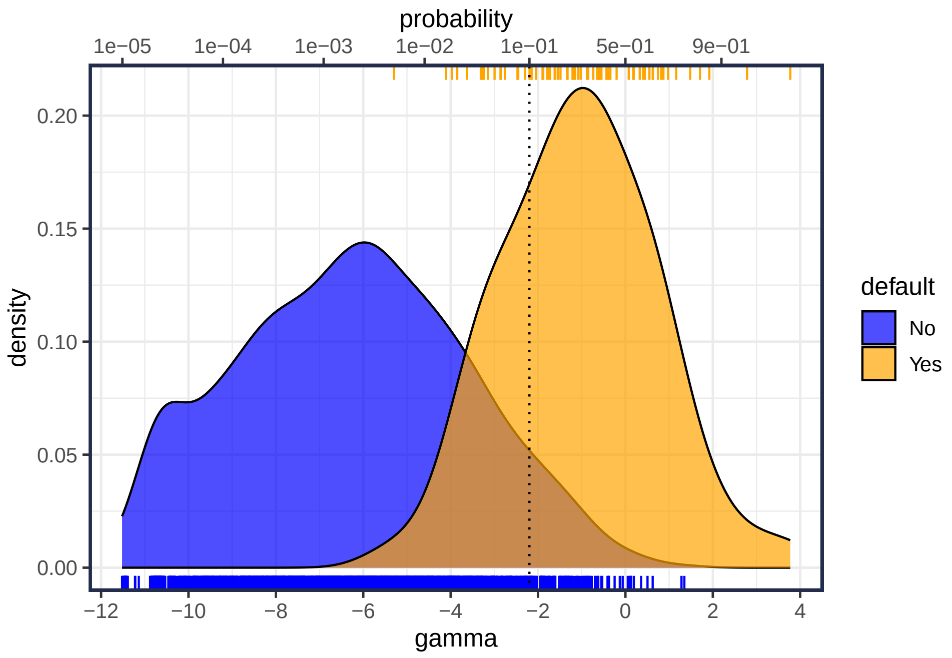
3.4 Performance over a range of thresholds

In the previous example, a hard classification was made using a threshold of $\hat{p}(x) \geq 0.10$. But performance varies as we adjust the threshold. Let's explore!

I'll use $\hat{\gamma}(x)$ instead of $\hat{p}(x)$ for this illustration because it better separates the classes.

```
#: Get predictions (of gamma(x)) on test data
gamma_hat = predict(fit.lm, Default[test,], type='link')
p_hat = predict(fit.lm, Default[test,], type='response')
```

- The model is unable to perfectly discriminate between groups, but the *defaults* do get scored higher in general:
  - As a reference point, note that $\gamma(x) = 0 \rightarrow \Pr(Y = 1 \mid X = x) = 1/2$
  - $\gamma(x) = \log p(x)/(1 - p(x))$



- We can calculate performance over a range of thresholds:

```
# truth: {0,1} vector
# score: risk score with larger values correspond to label = 1.
# thres: vector of thresholds at which to calculate metric.
# Note: decision is 1 if score > thres, 0 if score <= thres.
perf_table <- function(truth, score, thres=NULL){
  if(is.null(thres)) thres = seq(min(score, max(score), length=1000))

  x = c(-Inf, thres, Inf) %>% unique() %>% sort() # expand and clean thresholds
  tibble(truth, score) %>%
    # create groups by threshold
    mutate(
      bin = findInterval(score, x, left.open = TRUE),
      val = x[bin+1]
    ) %>%
    # counts by group/threshold
    group_by(val) %>%
      summarize(n = n(), n.1 = sum(truth), n.0 = n-n.1) %>%
    ungroup() %>%
    complete(val = thres, fill = list(n=0L, n.1=0L, n.0 = 0L)) %>%
    # calculate metrics
    arrange(val) %>%
    mutate(
      TN = cumsum(n.0),    # True negatives
      FN = cumsum(n.1),    # False negatives
      TP = sum(n.1) - FN,  # True positives
      FP = sum(n.0) - TN,  # False positives
      TPR = TP/sum(n.1),   # True positive rate (TP / Positives)
      FPR = FP/sum(n.0)    # False positive rate (FP / Negatives)
    ) %>%
    # drop values outside of stated thresholds
    filter(val %in% thres)  %>%
    # retain relevant metrics
    select(-n, -n.1, -n.0, score = val)
}
```

```
thresholds = seq(0, 1, length = 1000)
perf = perf_table(truth = G.test, score = p_hat, thres = thresholds) %>%
  mutate(p_hat = score, gamma_hat = log(p_hat) - log(1-p_hat), .before=1) %>%
  select(-score)
```

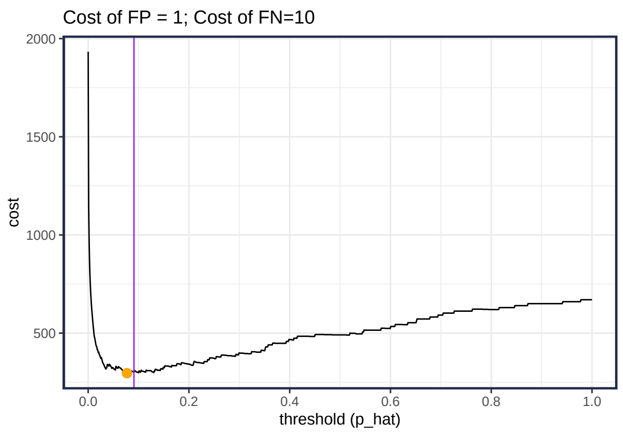
p_hat	gamma_hat	TN	FN	TP	FP	TPR	FPR
0.000	-Inf	0	0	67	1933	1.000	1.000
0.001	-6.906	789	0	67	1144	1.000	0.592
0.002	-6.212	979	0	67	954	1.000	0.494
0.003	-5.805	1095	0	67	838	1.000	0.434
0.004	-5.516	1171	0	67	762	1.000	0.394
0.005	-5.292	1238	1	66	695	0.985	0.360

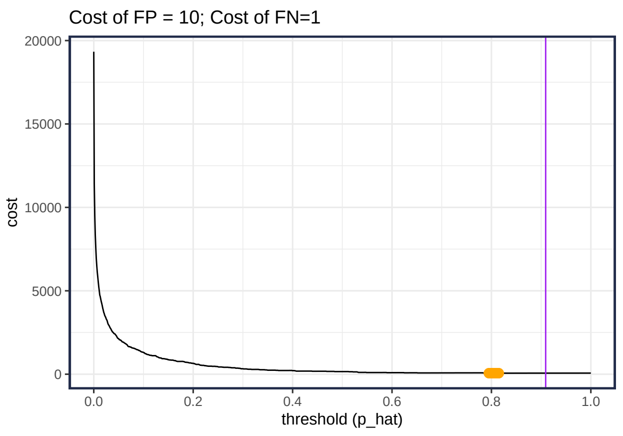
p_hat	gamma_hat	TN	FN	TP	FP	TPR	FPR
0.995	5.292	1933	67	0	0	0	0
0.996	5.516	1933	67	0	0	0	0
0.997	5.805	1933	67	0	0	0	0
0.998	6.212	1933	67	0	0	0	0
0.999	6.906	1933	67	0	0	0	0
1.000	Inf	1933	67	0	0	0	0

- Note: the `perf` object is *only based on the rank order* of the predictions. This means that the same results would be obtained if we used $\hat{\gamma}(x)$ or $\hat{p}(x)$ to do the ranking.
  - This is because there is a one-to-one monotone relationship between $\hat{\gamma}(x)$ and $\hat{p}(x)$.
  - The `perf` object grouped by both `gamma_hat` and `p_hat` so both thresholds are available. But we can switch back and forth from the relationship $\log(p/(1-p)) = \gamma$, so its easy to switch between the two depending on what is most convienient.

3.4.1 Cost Curves

- Under the usual scenario where $L(0,0) = L(1,1) = 0$, the cost only depends on the ratio of false positive costs (C_{FP}) to false negative costs (C_{FN}).
- note: the purple is the *theoretical* optimal threshold (using $t^* = \log C_{\text{FP}}/C_{\text{FN}}$ for $\hat{\gamma}(x)$ and $C_{\text{FP}}/(C_{\text{FP}} + C_{\text{FN}})$ for $\hat{p}(x)$) and the orange point is at the optimal value for the test data.

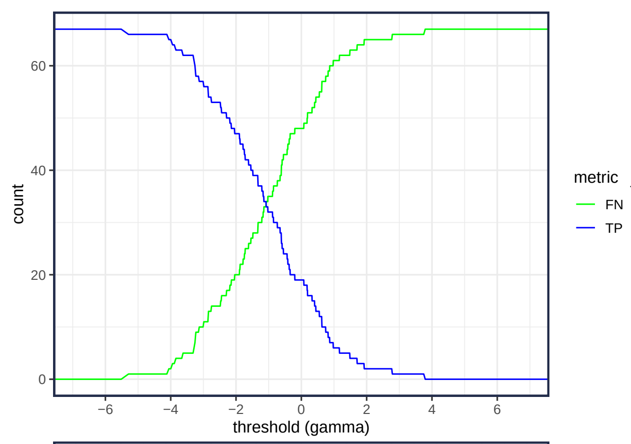


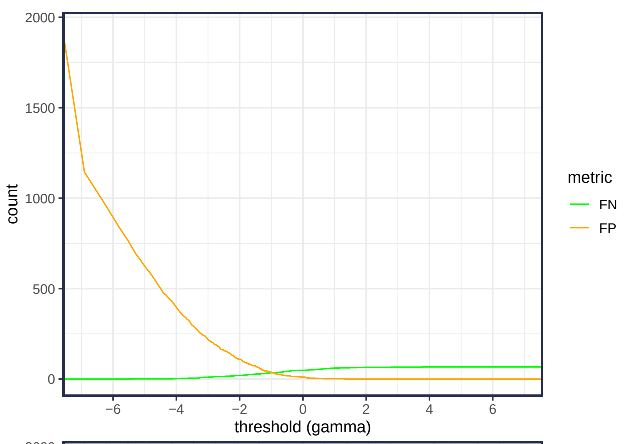


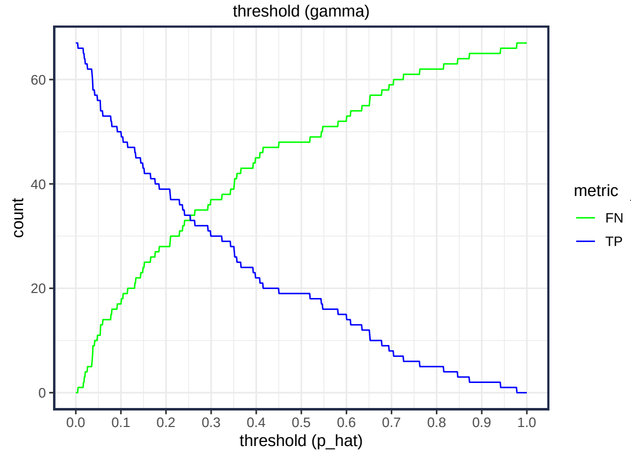
Optimal Threshold

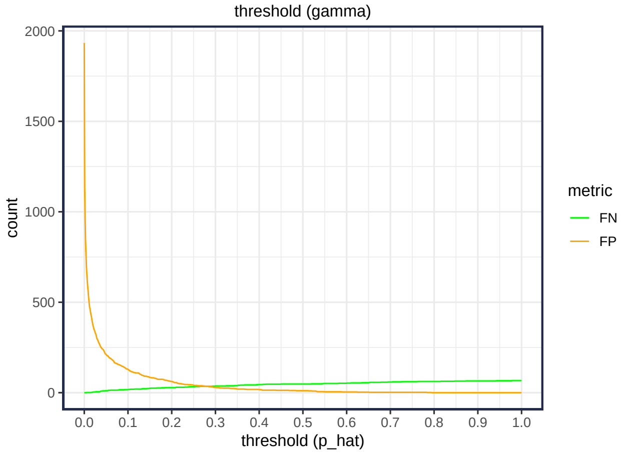
- The *theoretically* optimal threshold is based on the *true* $\gamma(x) = \log \frac{p(x)}{1-p(x)}$ (for a given cost ratio of FP to FN)
- The observed optimal threshold will differ when the model's estimate $\hat{\gamma}(x) \neq \gamma(x)$
  - Hopefully, they are close and it won't make much difference which one you use. But I'd take the estimated threshold if I had sufficient data.
- Note that the estimated values depend on the prior class probabilities. If you suspect these may differ in the future, then you should adjust the threshold.

3.4.2 General Performance as function of threshold (select metrics)

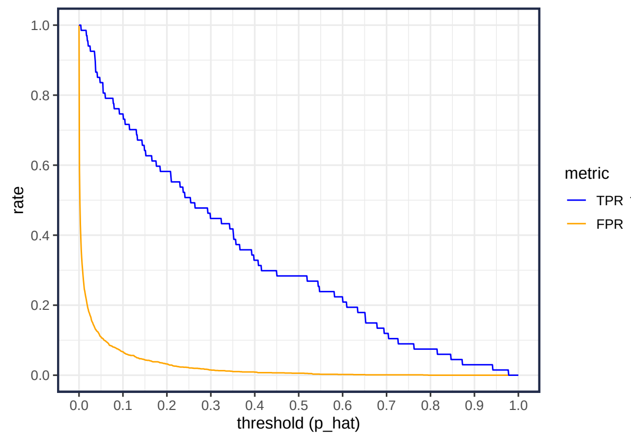


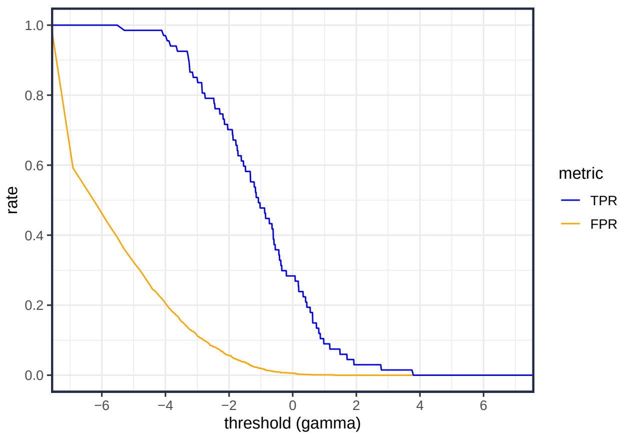


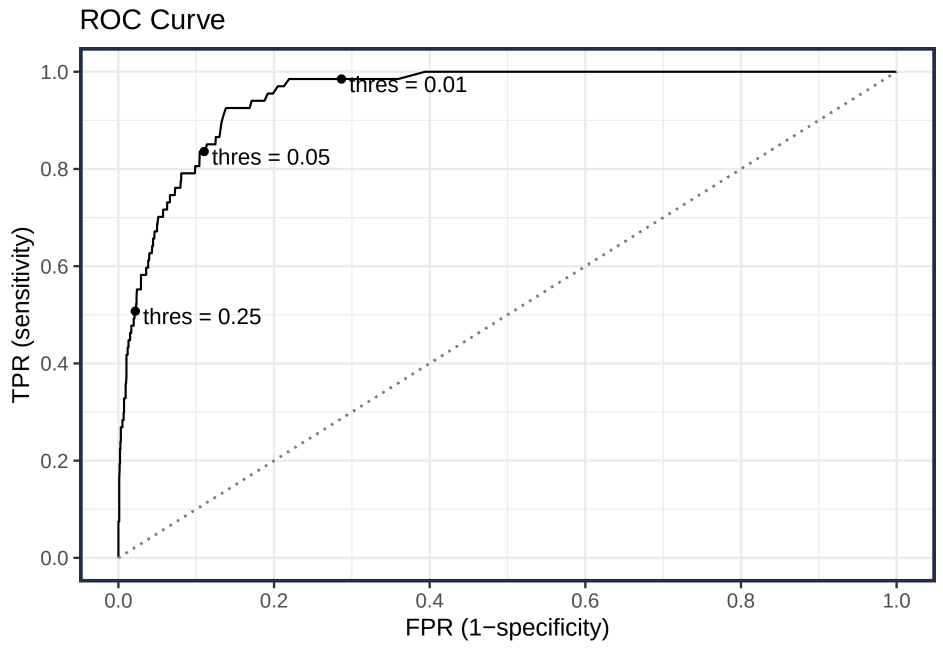




3.4.3 ROC Curves (Receiver Operating Characteristic)





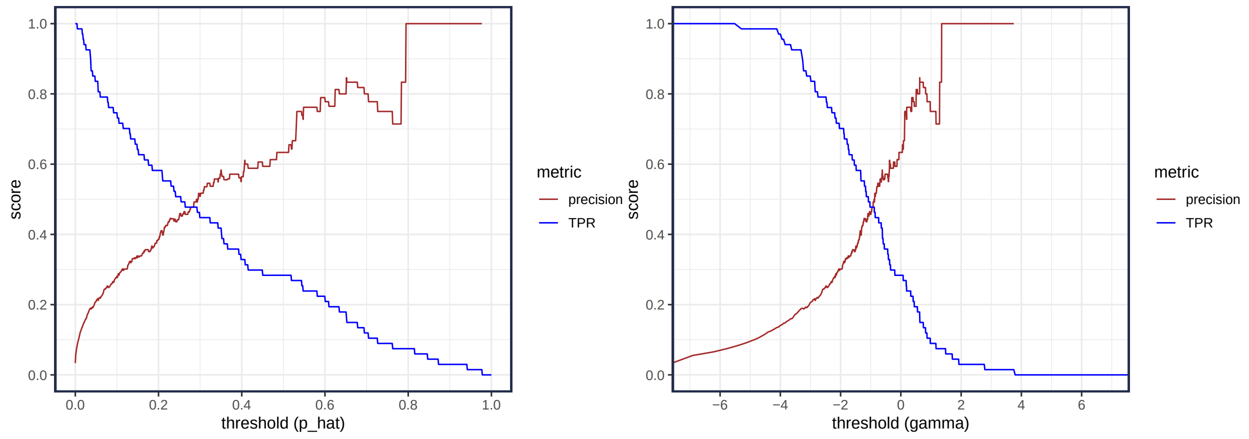


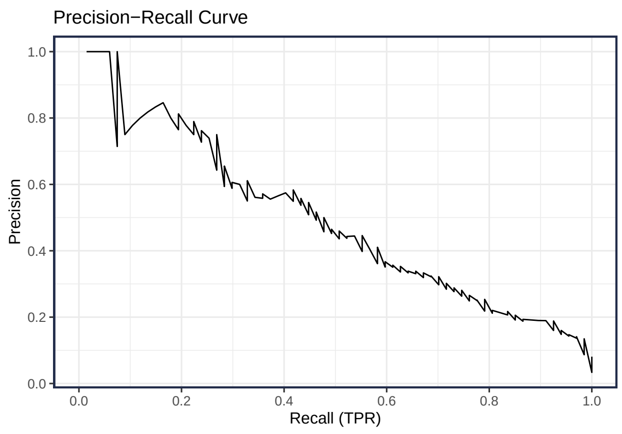
AUROC

- The *area under the ROC curve* (AUROC or AUC) is a popular performance metric
- I don't think it is a great way to compare classifiers for several reasons
  - The main reason is that in a real application you can almost always come up with an estimated cost/loss for the different decisions
  - To say it another way, comparisons should be made at a single point on the curve; the entire FPR region should not factor into the comparison.
- The AUROC is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one.
  - AUROC is proportional to the Mann-Whitney U statistic

3.4.4 Precision Recall Curves

- Popular for information retrieval/ranking
- The *precision* metric is not monotonic wrt threshold, hence the sawteeth pattern.



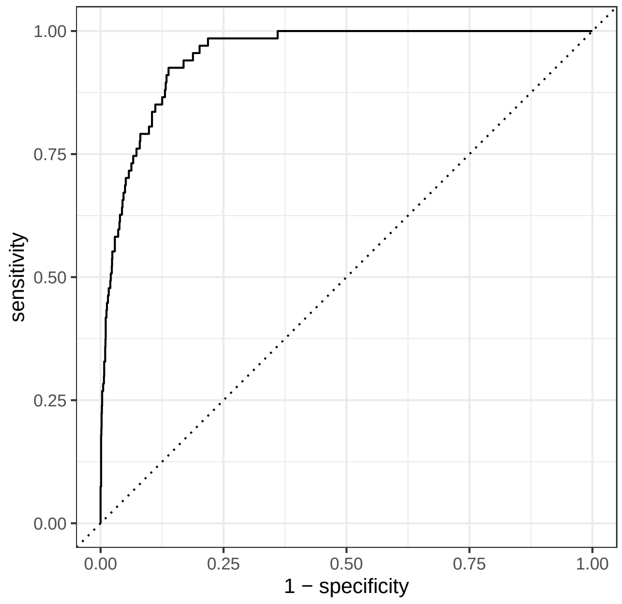


3.4.5 R Code

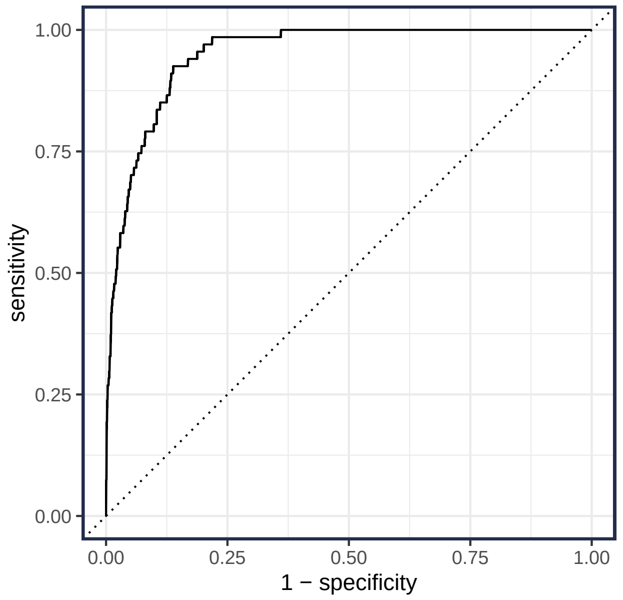
Once we have the FP, TP, TN, FN values for a set of thresholds (like what is in the `perf` object), then we have everything we need to calculate any metric (e.g., gain, lift, F1, . . .).

- But I will mention the `yardstick` R package which offers some functionality you may find convenient
- List of the metrics included in the `yardstick` package

```r
library(yardstick)   # for evaluation functions

#: ROC plots
ROC = tibble(truth = factor(G.test, levels=c(1,0)), gamma_hat) %>%
  yardstick::roc_curve(truth, gamma_hat)

autoplot(ROC)   # autoplot() method
```



```r
ROC %>%          # same as autoplot()
  ggplot(aes(1-specificity, sensitivity)) + geom_line() +
  geom_abline(lty=3) +
  coord_equal()
```



```r
#: Area under ROC (AUROC)
tibble(truth = factor(G.test, levels=c(1,0)), gamma_hat) %>%
  roc_auc(truth, gamma_hat)
#> # A tibble: 1 x 3
#>   .metric .estimator .estimate
#>   <chr>   <chr>          <dbl>
#> 1 roc_auc binary         0.951

roc_auc_vec(factor(G.test, 1:0), gamma_hat)
#> [1] 0.9505
```

3.5 More than two classes

Nothing really changes when there are more than two classes; we still want to choose the (hard) label that has minimum EPE:

$$\begin{aligned}\mathrm{EPE}_x(g) &= \mathrm{E}_{G|X=x}\left[L(G, \hat{G}(x) = g) \mid X = x\right] \\ &= \sum_k L(G = k, \hat{G} = g)\,\Pr(G = k \mid X = x)\end{aligned}$$

$$\begin{aligned}G^*(x) &= \arg\max_g \widehat{\mathrm{EPE}}_x(g) \\ &= \arg\max_g \sum_k L(G = k, \hat{G} = g)\widehat{\Pr}(G = k \mid X = x)\end{aligned}$$

3.6 Summary of Classification Evaluation

- Ask yourself: do I really need to make a hard classification? Or are risk scores/probabilities better for end user?
- Use cost! The other metrics are probably not going to give you what you really want.
  - Resist the pressure to use AUROC, Accuracy, F1.

  - If you don't know cost(FP)/cost(FN) ratio, then report performance for a reasonable range of values.

- For Binary Classification Problems, the optimal decision is to choose $\hat{G}(x) = 1$ if

$$\begin{aligned}\frac{p(x)}{1-p(x)} &\geq \frac{L(0,1) - L(0,0)}{L(1,0) - L(1,1)} \\ &= \frac{\text{FP – TN}}{\text{FN – TP}}\end{aligned}$$

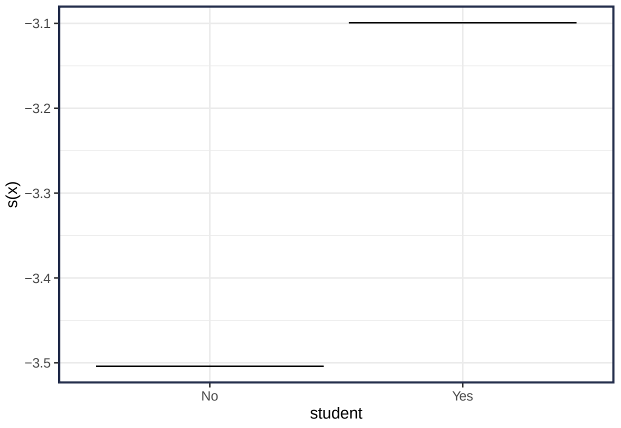
- Consider the connection to Decision Theory, make the decision that maximizes *expected utility*. The losses define the utility.
- In practice, we need to use an *estimated* $\hat{p}(x)$ or $\hat{\gamma}(x)$ and *estimated* threshold.
- Model parameters are usually estimated with a different metric than what's used for evaluation.
  - E.g., Estimate logistic regression parameters by minimizing Log-loss (i.e., maximum likelihood)
  - E.g., Hinge Loss for Support Vector Machines (SVM)
  - But Total Cost, MAE, F1, AUROC are used for evaluation (and tuning parameter estimation).
  - Reason: its difficult to estimate model parameters with such loss functions (e.g., non-differentiable, non-unique, etc.)

4 Generalized Additive Models (GAM)

In our discussion of Basis Expansions, we covered how the relationship between a single raw predictor x and the outcome could be made more complex with basis expansions.

- **Example 1: Categorical Predictor One-Hot Encoded**

```
#: One-hot Basis
X1 = model.matrix(~student-1, data=Default)
head(X1, 4)
#>   studentNo studentYes
#> 1         1          0
#> 2         0          1
#> 3         1          0
#> 4         1          0

#: Fit
fit.1 = glm(y ~ student-1, family="binomial", data=Default)

#: Plot
Default %>%
  mutate(pred = predict(fit.1, newdata=Default, type="link")) %>%
  distinct(student, pred) %>%
  ggplot(aes(student, pred)) + geom_errorbar(aes(ymin=pred, ymax=pred)) +
  labs(y="s(x)")
```



- **Example 2: Continuous Predictor with Polynomial Basis**

```
#: Fit linear
fit.lm = glm(y~income, data=Default, family="binomial")

#: Polynomial Basis
X2 = model.matrix(y~poly(income, degree=4)-1, data=Default)
head(X2, 4)
#>   poly(income, degree = 4)1 poly(income, degree = 4)2 poly(income, degree = 4)3
#> 1                  0.008132                 -0.003807                -0.0080610
#> 2                 -0.016055                  0.016202                -0.0138049
#> 3                 -0.001312                 -0.009300                 0.0057123
#> 4                  0.001640                 -0.009414                 0.0009502
#>   poly(income, degree = 4)4
#> 1                 0.0006076
#> 2                 0.0052431
#> 3                 0.0063483
#> 4                 0.0083087
```

```
#: Polynomial Model (edf=5)
fit.2 = glm(y~poly(income, degree=4), family="binomial", data=Default)
# equivalent to:  glm(y~X2, family="binomial", data=Default)
```

- **Example 3: Continuous Predictor with Binning (Regressograms)**

```
#: Binning Basis
X3 = model.matrix(~cut(income, 5)-1, data=Default)
head(X3, 4)
#>   cut(income, 5)(699,1.53e+04] cut(income, 5)(1.53e+04,2.99e+04]
#> 1                            0                                  0
#> 2                            1                                  0
#> 3                            0                                  0
#> 4                            0                                  0
#>   cut(income, 5)(2.99e+04,4.44e+04] cut(income, 5)(4.44e+04,5.9e+04]
#> 1                                 1                                 0
#> 2                                 0                                 0
#> 3                                 1                                 0
#> 4                                 1                                 0
#>   cut(income, 5)(5.9e+04,7.36e+04]
#> 1                                0
#> 2                                0
#> 3                                0
#> 4                                0

#: Binning Model (edf=5)
fit.3 = glm(y~cut(income, 5)-1, data=Default, family="binomial")
# equivalent to:  glm(y~X3-1, family="binomial", data=Default)
```

- **Example 4: Continuous Predictor with B-Splines Basis**

```
library(splines) # for bs() function

#: B-spline Basis
X4 = model.matrix(~bs(income, df=4)-1, data=Default)
head(X4, 4)
#>   bs(income, df = 4)1 bs(income, df = 4)2 bs(income, df = 4)3
#> 1              0.1204              0.4351            0.428565
#> 2              0.5755              0.1229            0.008137
#> 3              0.3521              0.4809            0.166404
#> 4              0.2625              0.4994            0.238160
#>   bs(income, df = 4)4
#> 1           1.591e-02
#> 2           0.000e+00
#> 3           0.000e+00
#> 4           2.576e-05

#: Binning Model (edf=5)
fit.4 = glm(y~bs(income, df=3), data=Default, family="binomial")
# equivalent to:  glm(y~X4, family="binomial", data=Default)
```

- **Example 5: Continuous Predictor with Penalized Spline**

```
library(mgcv)
#: Fit penalized spline, it will select best edf
#    specify smooth with s()
fit.5 = gam(y~s(income), data=Default, family="binomial")
summary(fit.5)
#>
```

```
#> Family: binomial
#> Link function: logit
#>
#> Formula:
#> y ~ s(income)
#>
#> Parametric coefficients:
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept)  -3.3819     0.0564     -60   <2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Approximate significance of smooth terms:
#>            edf Ref.df Chi.sq p-value
#> s(income) 4.31   5.37   10.8    0.06 .
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> R-sq.(adj) =  0.00098   Deviance explained = 0.466%
#> UBRE = -0.70823  Scale est. = 1         n = 10000
```

```
#: Plot of Fit
Default %>%
  ggplot(aes(income)) +
  geom_rug(data=. %>% filter(y==1), aes(color=default), sides="t", color = plot_cols[["Yes"]]) +
  geom_rug(data=. %>% filter(y==0), aes(color=default), sides="b", color = plot_cols[["No"]]) +
  scale_color_manual(values=c(mgcv = "purple", `B-spline`="red" ,
                              Binning="green", Polynomial="blue", Linear="black"), name="model") +
  coord_cartesian(ylim=c(-3.8, -3)) +
  labs(y="s(x)") +
  geom_function(fun = ~predict(fit.5, newdata=tibble(income=.)), aes(color="mgcv")) +
  geom_function(fun = ~predict(fit.4, newdata=tibble(income=.)), aes(color="B-spline")) +
  geom_function(fun = ~predict(fit.3, newdata=tibble(income=.)), aes(color="Binning")) +
  geom_function(fun = ~predict(fit.2, newdata=tibble(income=.)), aes(color="Polynomial")) +
  geom_function(fun = ~predict(fit.lm, newdata=tibble(income=.)), aes(color="Linear"))
#> Error in eval(expr, envir, enclos): object 'plot_cols' not found
```

4.1 Generalized Additive Models (GAMs)

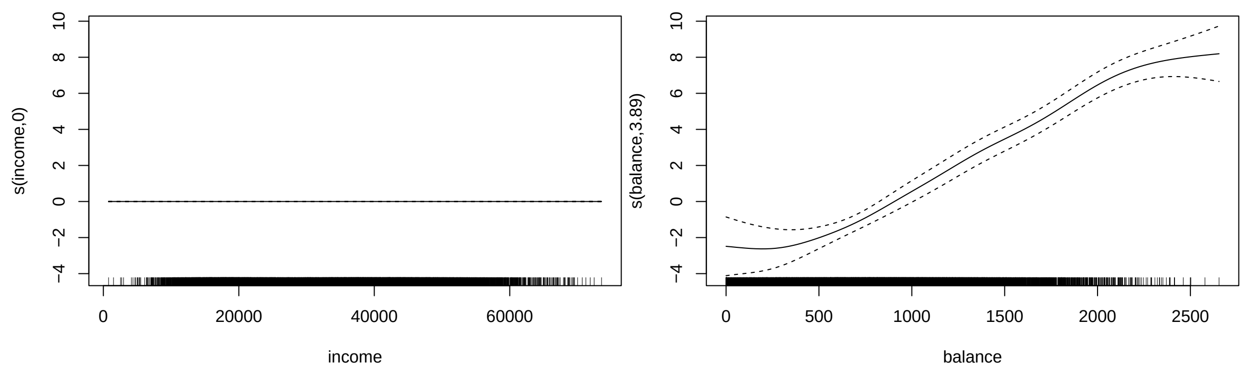
All of the above models are for a *single* predictor. The extension to multiple predictors is called **Generalized Additive Models (GAMs)**.

Instead of the linear form
$$f(\mathbf{x}) = \beta_0 + \sum_j \beta_j x_j,$$
use non-linear bases for each predictor
$$f(\mathbf{x}) = \beta_0 + \sum_j s_j(x_j)$$
where $s_j(x_j)$ can allow non-linear (e.g., smooth) forms, like B-splines.

- For binary classification setting: $\text{logit} p(\mathbf{x}) = f(\mathbf{x})$
- These are *additive* models because each term adds its contribution, although potentially in a non-linear way

  - But interactions can still be accommodated using `s(x1, x2)` or `s(x1, by=fac)`
- These are *generalized* models following the GLM notation. You can use different distributions with the `family=` argument
- GAMs retain the interpretability of a linear additive model (linear regression, logistic regression), but can add complexity to predictors where needed
  - Drawback: can be slow, especially for high dimensional data
- In **R**, the `mgcv` package is excellent for implementing GAM models.
  - It used Generalized Cross-validation to select optimal smoothing for each component
  - It also has a `select=TRUE` argument to further shrink entire components toward 0
  - Can handle low dimension interactions (even factor-continuous)
- See ISL 7.7 or ESL 9.1 for more details

```
library(mgcv)

fit.gam = gam(y ~ student + s(income) + s(balance),  # smooth main effects
              select = TRUE,                           # shrink components toward 0
              family="binomial", data=Default)

summary(fit.gam)
#>
#> Family: binomial
#> Link function: logit
#>
#> Formula:
#> y ~ student + s(income) + s(balance)
#>
#> Parametric coefficients:
#>              Estimate Std. Error z value Pr(>|z|)
#> (Intercept)    -5.611      0.318  -17.62  < 2e-16 ***
#> studentYes     -0.711      0.149   -4.78  1.7e-06 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Approximate significance of smooth terms:
#>                 edf Ref.df Chi.sq p-value
#> s(income)  0.000806      9      0    0.85
#> s(balance) 3.888128      9    641  <2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> R-sq.(adj) =  0.339   Deviance explained = 46.3%
#> UBRE = -0.84185  Scale est. = 1         n = 10000
plot(fit.gam)
```



4.2 Estimating $\hat{s}_j(x_j)$ with Backfitting

The smooth terms of a GAM model can be estimating using an iterative approach called *backfitting*.

Algorithm: Backfitting for GAM (Squared Error Loss / Linear Regression)

```
Model: $\hat{y}(\mathbf{x}) = \beta_0 + \sum_{j=1}^{p} \hat{s}_j(x_j)$
    1. Start with intercept-only model. All smooth terms set to zero: $s_j(x_j) = 0$.
    2. Iterate over all $p$ predictor variables:
        a. Construct partial residuals $r_i = y_i - \hat{\beta}_0 - \sum_{k \neq j} \hat{s}_k(x_{ik})$ holding out the $j$th predictor
        b. Fit $j$th smoother to residuals: Estimate $\hat{s}(x_j)$ from $\{(r_i, x_{ij})\}_{i=1}^{n}$
    3. Repeat many times stopping when converged (i.e., smooth fits no longer changing very much)
Note: There are more details (see ESL 9.1), but this is the main (and simple) idea.
```

5 Appendix: R Code

Set-up

```r
#: Load Required Packages
library(ISLR)
library(FNN)
library(broom)
library(yardstick)
library(tidyverse)
```

```r
#----------------------------------------------------------------------------#
#: Default Data
#  From the ISLR package
#  The outcome variable is `default`
#----------------------------------------------------------------------------#
library(ISLR)
data(Default, package="ISLR")     # load the Default Data

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

#: Summary Stats (Notice only 333 (3.3%) have defaulted)
summary(Default)
#>  default    student       balance         income             y
#>  No :9667   No :7056   Min.   :   0   Min.   :  772   Min.   :0.0000
#>  Yes: 333   Yes:2944   1st Qu.: 482   1st Qu.:21340   1st Qu.:0.0000
#>                        Median : 824   Median :34553   Median :0.0000
#>                        Mean   : 835   Mean   :33517   Mean   :0.0333
#>                        3rd Qu.:1166   3rd Qu.:43808   3rd Qu.:0.0000
#>                        Max.   :2654   Max.   :73554   Max.   :1.0000
```

Linear Regression (for binary response)

```r
#: Fit Linear Regression Model
fit.lm = lm(y~student + balance + income, data=Default)
```

k nearest neighbor

```r
library(FNN)           # for knn.reg() function
library(tidymodels)    # for recipe functions

#: center and scale predictors so Euclidean distance makes more sense
library(tidymodels)
pre_process = recipe(y ~ student + balance + income, data = Default) %>% # specify formula
  step_dummy(student) %>%                        # use dummy coding
  step_normalize(all_predictors()) %>%           # center and scale
  prep()                                         # estimate means and sds

#: apply the transformation to the predictors
X.scale = bake(pre_process, Default, all_predictors())
Y = bake(pre_process, Default, all_outcomes(), composition = "data.frame")
# Y = Default$y


#: Evaluation Points
eval.pts = expand_grid(
  student = levels(Default$student),
  balance = seq(min(Default$balance), max(Default$balance), length=50),
  income = seq(min(Default$income), max(Default$income), length=50)
  )

X.eval = bake(pre_process, eval.pts)  # scale eval pts too
#> Warning:  There was 1 column that was a factor when the recipe was prepped:
#>  'student'.
#>  This may cause errors when processing new data.
#  Note: this uses the same center and scale from the *training data*.
#        This is important!
#        Don't rescale the hold-out data separately!

#: fit knn model
knn5 = FNN::knn.reg(X.scale, y=Y, test=X.eval, k=5)
```

Logistic Regression

```r
#  More details in ISL_v2 4.7.2

#: Fit logistic regression model
fit.lr = glm(y ~ student + balance + income, data=Default,
             family="binomial")

## Alternatively, you can replace the binary y with a factor/character column
# fit.lr = glm(default~student + balance + income, data=Default,
#              family="binomial")
```

Convert data frame to model matrix

The `glmnet` package only handles model matrix and not data frames, so we have to convert the data into model matrix. When all predictors are numeric, this is easy (e.g., data.matrix() or model.matrix() if formula), but categorical/factor data needs to be handled separately and consistently if there are multiple data sets (e.g., train and test).

Here are a few options:

```r
#: Create model formula
fmla = as.formula(y ~ student + balance + income)
vars = fmla %>% terms() %>% labels()
```

```r
#-----------------------------------------#
#: Option 1: using model.matrix()
#-----------------------------------------#
X.train = model.matrix(fmla, data = Default)[,-1] # remove intercept term
Y.train = Default$y
# X.test = model.matrix(fmla, data = test)[,-1] # remove intercept term

#-----------------------------------------#
#: Option 2: using recipe() from recipe (tidymodels)
#-----------------------------------------#
library(tidymodels)  # library(recipe)
# recipe() sets the variables: predictor, outcome, case_weight, or ID
# prep() performs (i.e., fits) the pre-processing/transformations; set training=data_train
# bake() applies the pre-processing to new data
rec = recipe(fmla, data=Default) %>%
  step_dummy(all_nominal(), one_hot = TRUE) %>%
  prep()
X.train = bake(rec, new_data = Default, all_predictors(), composition="matrix")
Y.train = bake(rec, new_data = Default, all_outcomes()) %>% pull()
# X.test = bake(rec, new_data = test, all_predictors(), composition="matrix")

#-------------------------------------------#
#: Option 3: using hardhat::mold()
#-------------------------------------------#
library(tidymodels)  # library(hardhat)
obj = hardhat::mold(fmla, data=Default)
X.train = obj$predictors %>% as.matrix()
Y.train = obj$outcomes
# X.test = hardhat::mold(fmla, data=test)$predictors %>% as.matrix()

#-------------------------------------------#
#: Option 4: Manually using dplyr::select() + as.matrix()
#-------------------------------------------#
# Note: must manually transform, dummy, interactions, etc.
X.train = select(Default, !!vars) %>%
  mutate(dummy=1) %>%
  pivot_wider(names_from = student, values_from = dummy, values_fill = 0L) %>%
  as.matrix()                                    # make X matrix
Y.train = Default$y             # make Y vector
# X.test = select(test, !!vars) %>%
#   mutate(dummy=1) %>% pivot_wider(names_from = student, values_from = dummy, values_fill = 0L) %>%
#   as.matrix()

#-------------------------------------------#
#: Option 5: using glmnet::makeX
#-------------------------------------------#
X.train = glmnet::makeX(select(Default, !!vars))
# if test data is available
# X = glmnet::makeX(train = select(Default, -y), test = select(train, -y))
# X.train = X$x
# X.test = X$xtest
```

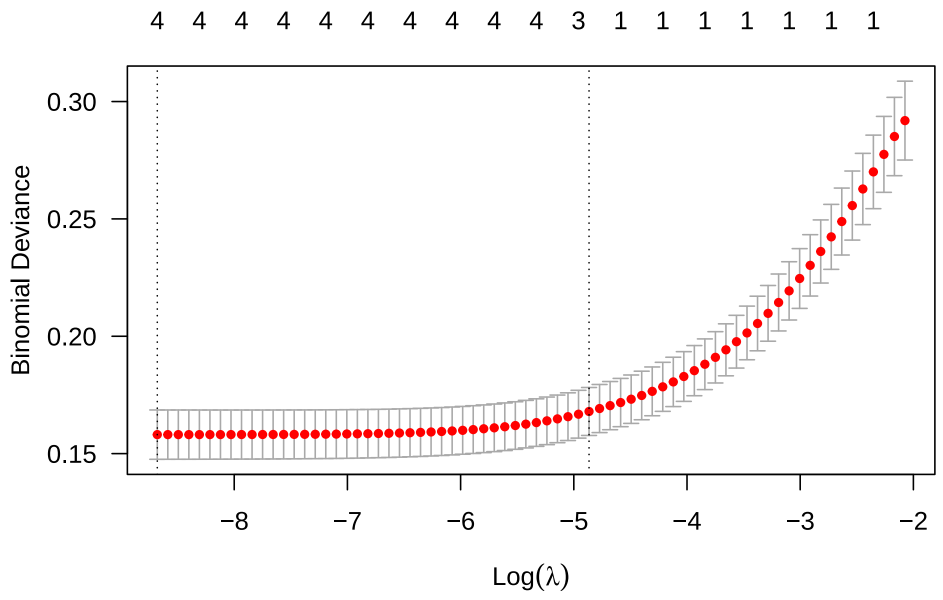
Penalized Logistic Regression

First need to convert data to numeric model matrix

```
# using the recipe/bake method
library(tidymodels)
rec = recipe(y~student + balance + income, data = Default) %>%
  step_dummy(all_nominal(), one_hot = TRUE)
rec_fit = rec %>% prep()
X.train = rec_fit %>%
  bake(all_predictors(), new_data = Default, composition="matrix")
Y.train = rec_fit %>% bake(all_outcomes(), new_data = Default) %>% pull()
X.eval = rec_fit %>%
  bake(all_predictors(), new_data = eval.pts, composition="matrix")
#> Warning:  There was 1 column that was a factor when the recipe was prepped:
#>   'student'.
#>   This may cause errors when processing new data.

# library(glmnet)
# vars = c("student", "balance", "income")
# X = glmnet::makeX(select(Default, !!vars), select(eval.pts, !!vars))
# X.train = X$x
# Y.train = Default$y
# X.eval = X$xtest
```

Now fit the elastic net

```
#: Elastic net with alpha = .5. Use CV to select lambda.
library(glmnet)
set.seed(2020)    # seed controls the cross-validation splits
fit.enet = cv.glmnet(X.train, Y.train,
                     alpha=.5,
                     family="binomial")

#: CV performance plot
plot(fit.enet, las=1)
```

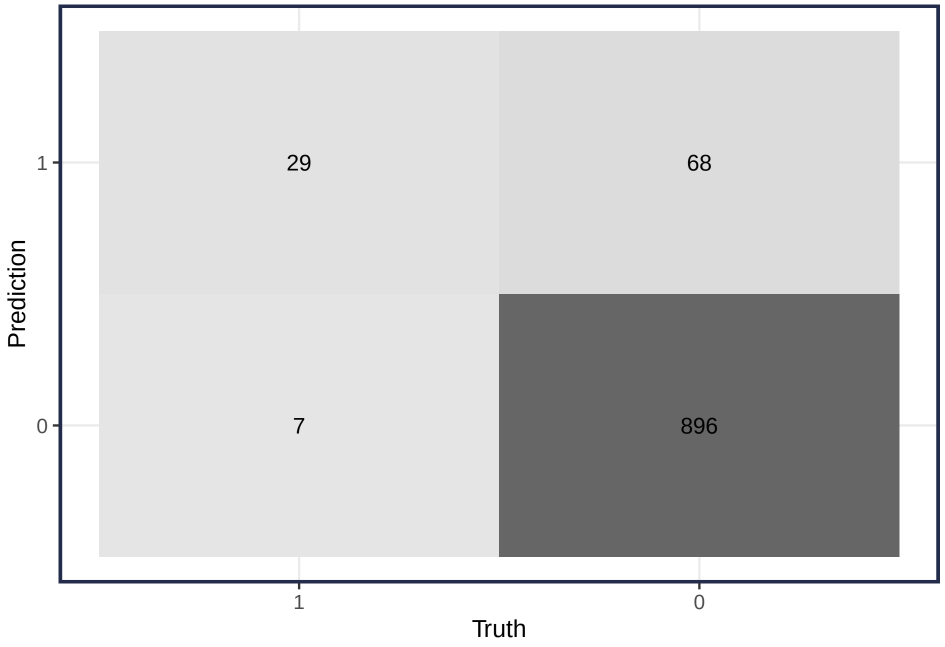


Performance Metrics and Curves

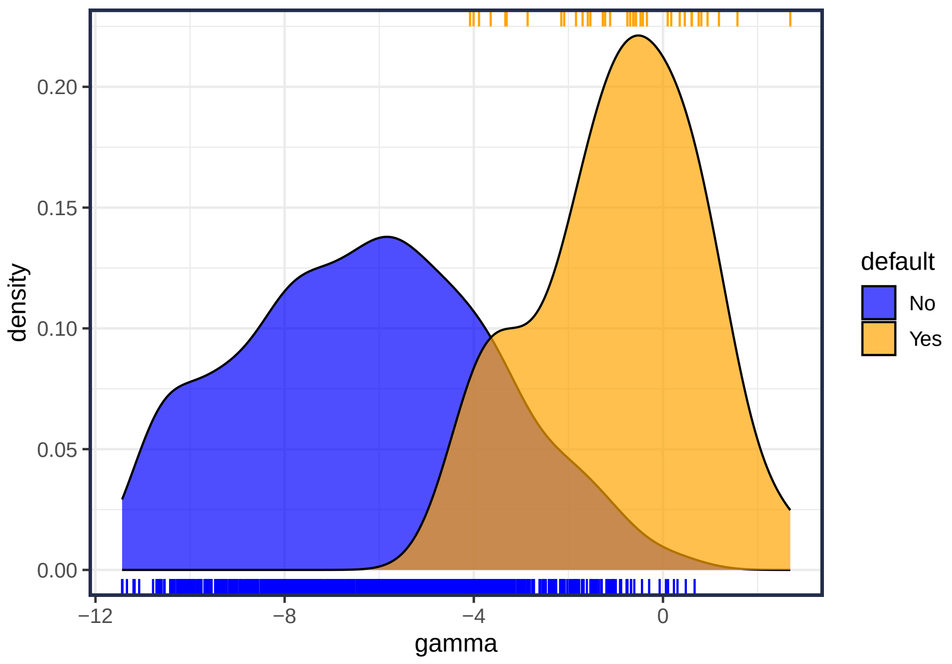
```
#: train/test split
set.seed(2019)
test = sample(nrow(Default), size=1000)
train = -test
```

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

#: Get predictions (of p(x) and gamma(x)) on test data
p.hat = predict(fit.lm, Default[test, ], type='response')
gamma = predict(fit.lm, Default[test, ], type='link')

#: Make Hard classification (use .10 as cut-off)
G.hat = ifelse(p.hat >= .10, 1, 0)
```

```
#: Make Confusion Table (base R)
G.test = Default$y[test]   # true values
table(predicted=G.hat, truth = G.test) %>% addmargins()
#>            truth
#> predicted    0    1   Sum
#>        0     896     7   903
#>        1      68    29    97
#>        Sum   964    36 1000

#: Make Confusion Table (yardstick)
library(yardstick)
# Note: the yardstick package functions, like conf_mat(), requires that hard
#  classifications have *factor* inputs (instead of *character*)
cm = tibble(G.test, G.hat) %>%
  mutate_all(~factor(.x, levels=c("1", "0"))) %>% # conf_mat() requires factors
  conf_mat(truth = G.test, estimate = G.hat)
cm
#>            Truth
#> Prediction   1   0
#>           1  29  68
#>           0   7 896
autoplot(cm, type = "heatmap")
```

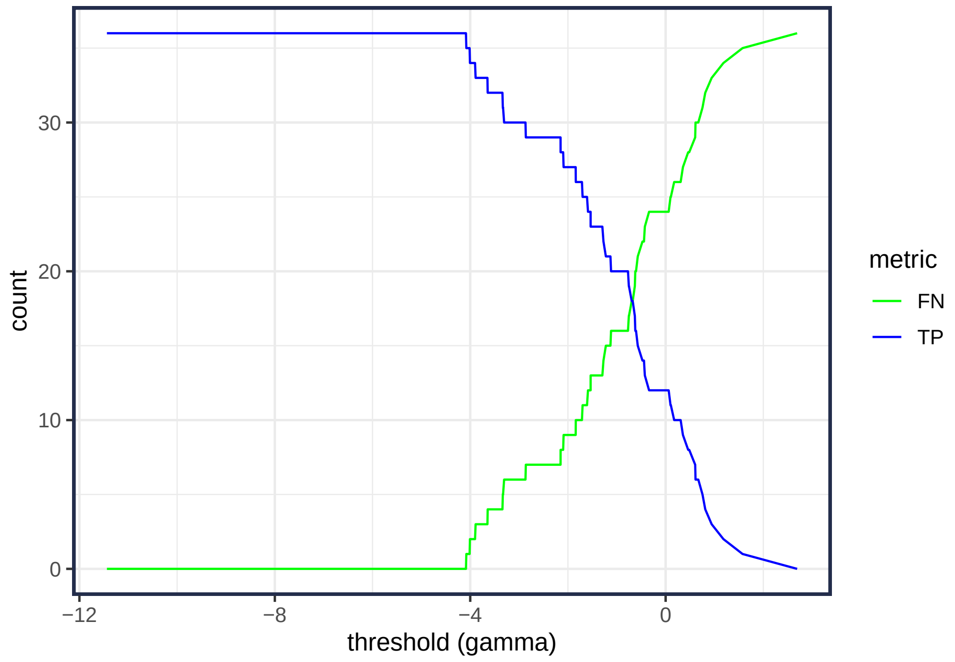


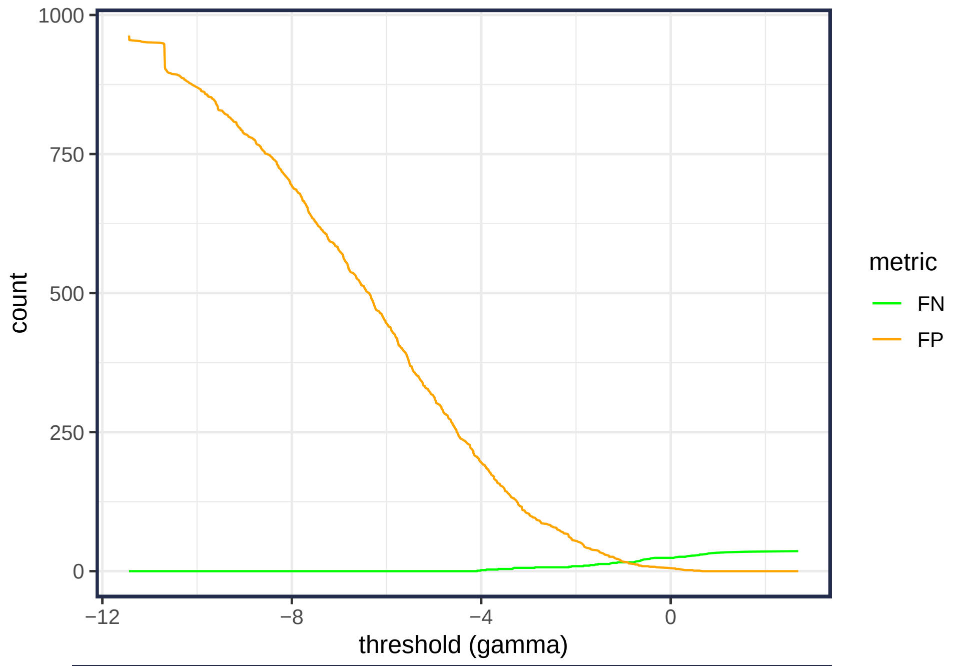
```
#: Visualize Performance by score
Default[test,] %>%
  mutate(gamma = gamma) %>%
  ggplot(aes(gamma, fill=default)) +
  geom_density(alpha=.70) +                        # add kernel density estimates
  geom_rug(data=. %>% filter(default == 'Yes'), # add rug to top
           aes(color=default), sides='t') +
  geom_rug(data=. %>% filter(default == 'No'),  # add rug to bottom
           aes(color=default), sides='b') +
  scale_fill_manual(values=c(Yes="orange", No="blue")) + # modify fill colors
  scale_color_manual(values=c(Yes="orange", No="blue"), guide="none") # modify colors
```



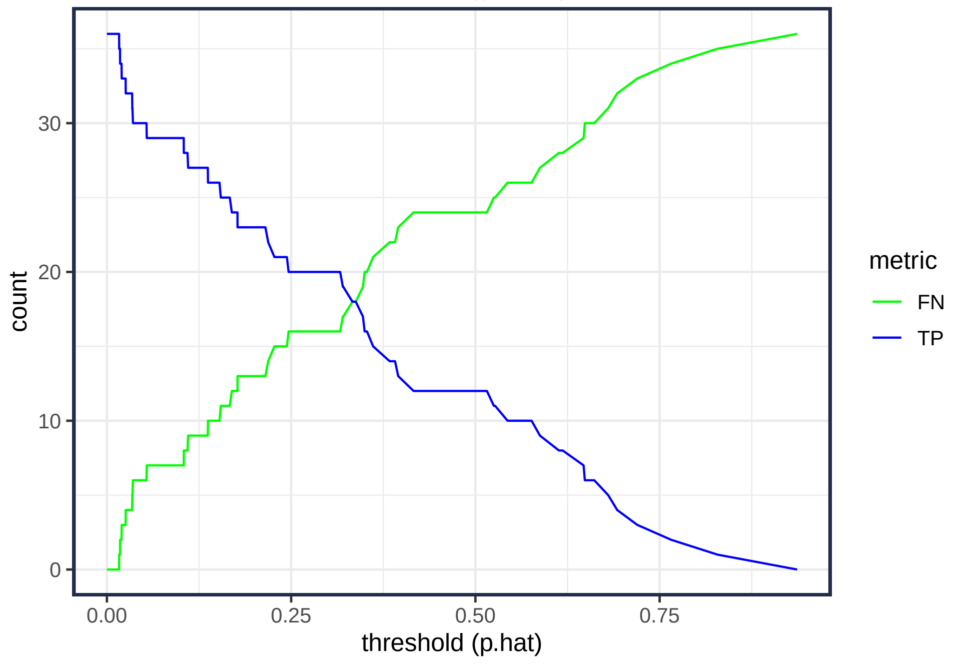
```
#: Get performance data (by threshold)
#   This table has one row for every threshold. The columns are the elements
#   of the confusion table plus FPR, TPR
perf = tibble(truth = G.test, gamma, p.hat) %>%
  #- group_by() + summarize() in case of ties
  group_by(gamma, p.hat) %>%
  summarize(n=n(), n.1=sum(truth), n.0=n-sum(truth)) %>% ungroup() %>%
  #- calculate metrics
  arrange(gamma) %>%
  mutate(FN = cumsum(n.1),     # false negatives
         TN = cumsum(n.0),     # true negatives
         TP = sum(n.1) - FN,   # true positives
         FP = sum(n.0) - TN,   # false positives
         N = cumsum(n),        # number of cases predicted to be 1
         TPR = TP/sum(n.1), FPR = FP/sum(n.0)) %>%
  #- only keep relevant metrics
  select(-n, -n.1, -n.0, gamma, p.hat)


## Note: gamma = log(p.hat) - log(1-p.hat) = log(p.hat / (1-p.hat))
```

```
#: Make performance curves
col_lines = c(TP = "blue", FP="orange", FN="green", TN="brown",
              TPR = "blue", FPR="orange", FNR="green", TNR="brown")

#: Make performance curves
perf %>% select(threshold=gamma, FN, TP) %>%
  gather(metric, n, -threshold) %>%
  ggplot(aes(threshold, n, color=metric)) + geom_line() +
  labs(x= "threshold (gamma)", y="count") +
  scale_color_manual(values=col_lines)

perf %>% select(threshold=gamma, FN, FP) %>%
  gather(metric, n, -threshold) %>%
  ggplot(aes(threshold, n, color=metric)) + geom_line() +
  labs(x= "threshold (gamma)", y="count") +
  scale_color_manual(values=col_lines)

perf %>% select(threshold=p.hat, FN, TP) %>%
  gather(metric, n, -threshold) %>%
  ggplot(aes(threshold, n, color=metric)) + geom_line() +
  labs(x= "threshold (p.hat)", y="count") +
  scale_color_manual(values=col_lines)

perf %>% select(threshold=p.hat, FN, FP) %>%
  gather(metric, n, -threshold) %>%
  ggplot(aes(threshold, n, color=metric)) + geom_line() +
  labs(x= "threshold (p.hat)", y="count") +
  scale_color_manual(values=col_lines)
```

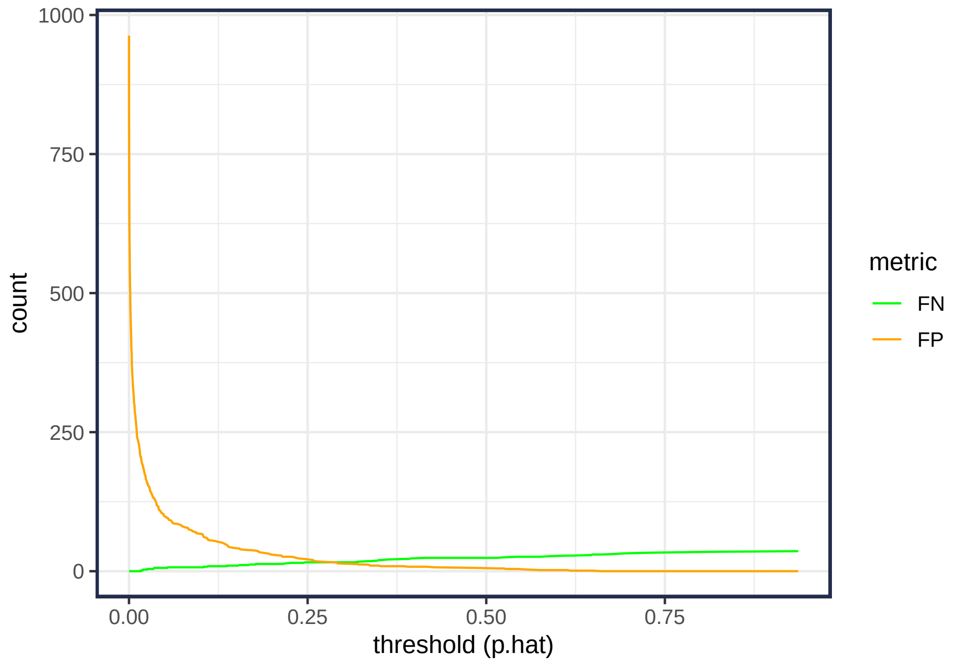




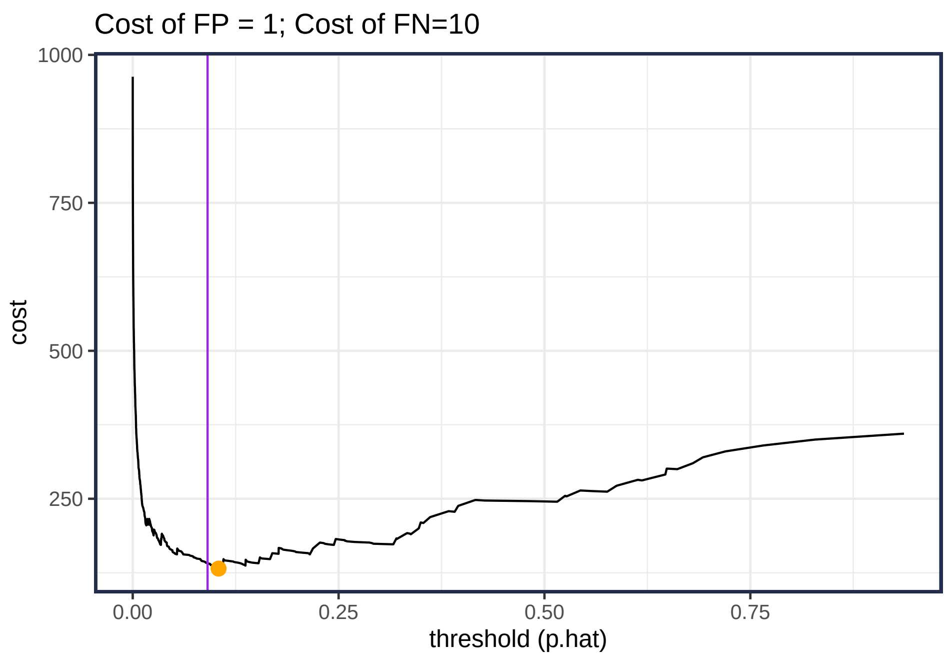
count
1000
750
500
250
0
−12
−8
−4
0
threshold (gamma)
metric
FN
FP

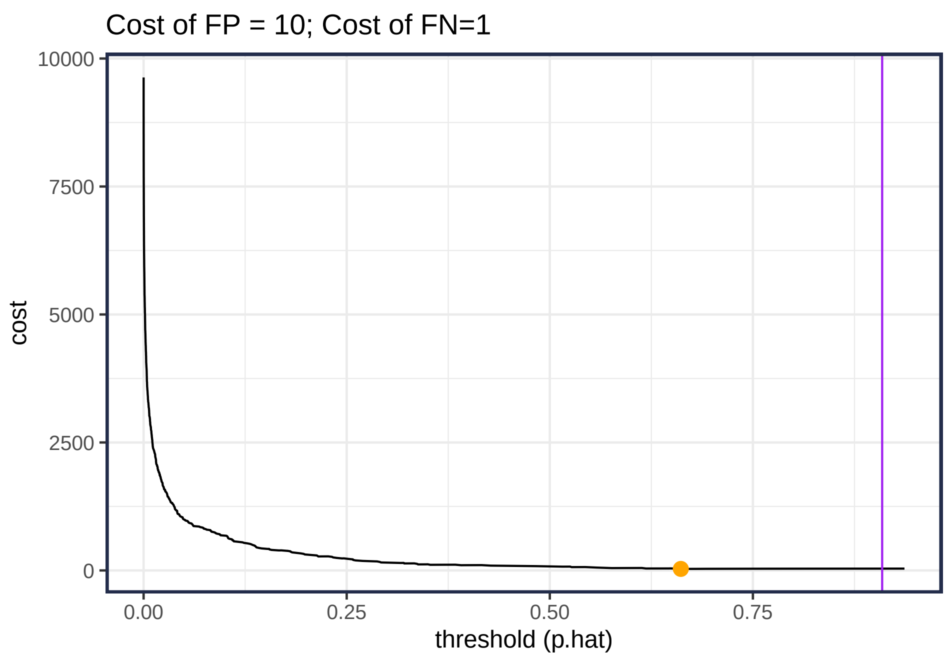


count
30
20
10
0
0.00
0.25
0.50
0.75
threshold (p.hat)
metric
FN
TP

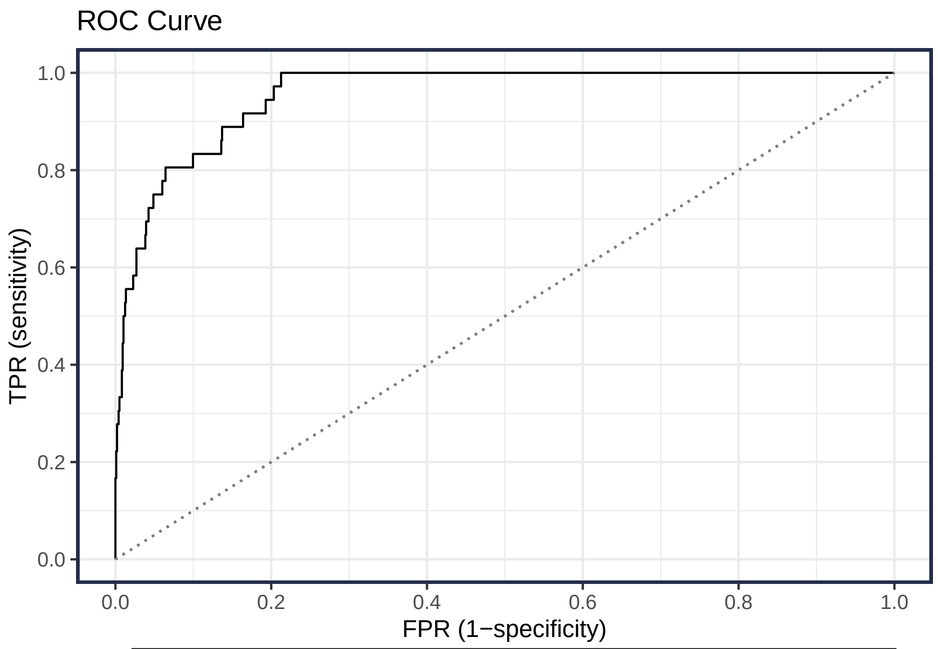


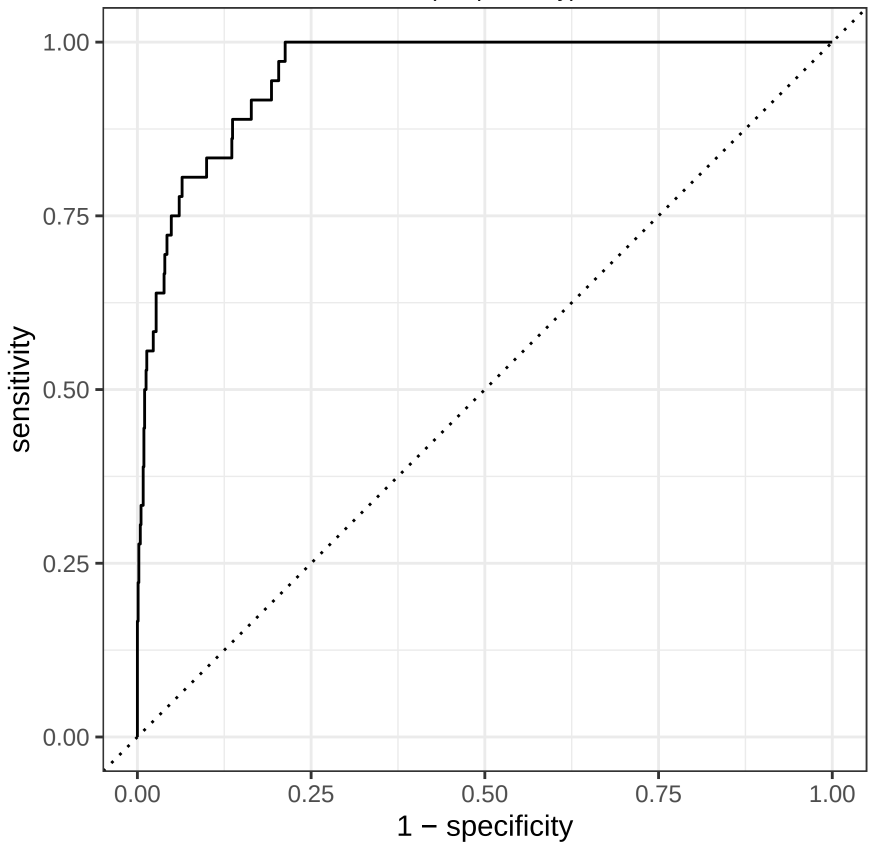
```r
#: Make Cost curves
perf %>% mutate(cost = 1*FP + 10*FN) %>%   # use 1:10 costs
  ggplot(aes(p.hat, cost)) + geom_line() +
  geom_point(data=. %>% filter(cost==min(cost)), size=3, color='orange') + # # optimal from test data
  geom_vline(xintercept = 1/11, color='purple') +  # theoretical optimal
  ggtitle('Cost of FP = 1; Cost of FN=10') +
  labs(x="threshold (p.hat)")

perf %>% mutate(cost = 10*FP + 1*FN) %>%   # use 10:1 costs
  ggplot(aes(p.hat, cost)) + geom_line() +
  geom_point(data=. %>% filter(cost==min(cost)), size=3, color='orange') + # optimal from test data
  geom_vline(xintercept = 10/11, color='purple') + # theoretical optimal
  ggtitle('Cost of FP = 10; Cost of FN=1') +   labs(x="threshold (p.hat)")
```

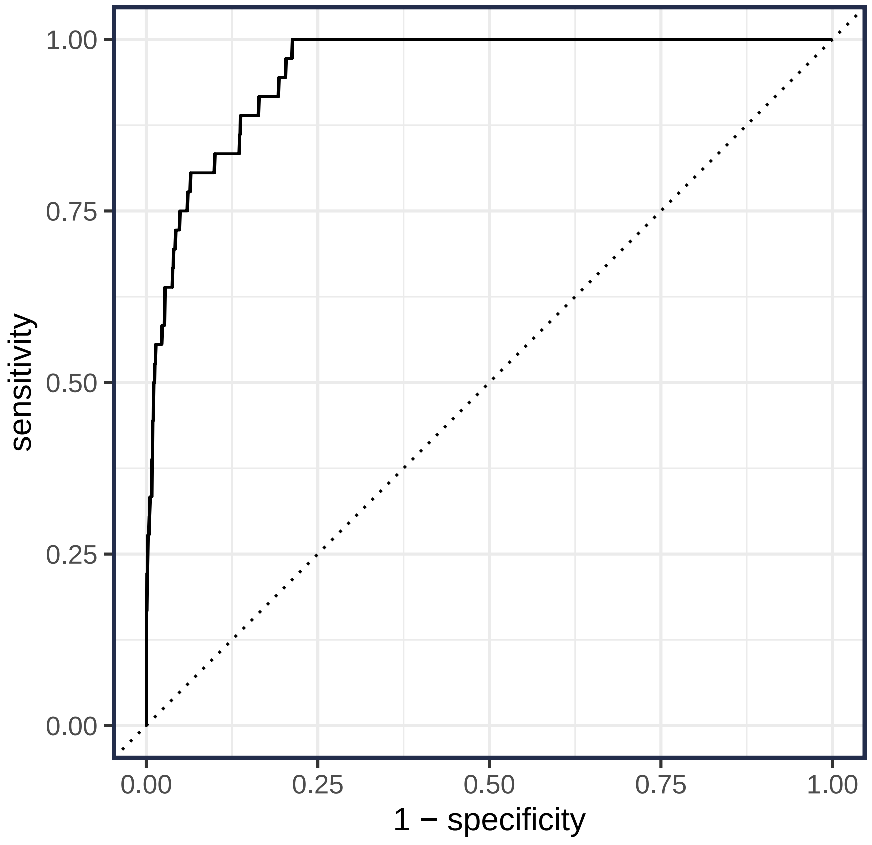




```r
#: Make ROC curve
perf %>%
  ggplot(aes(FPR, TPR)) + geom_path() +
  labs(x='FPR (1-specificity)', y='TPR (sensitivity)') +
  geom_segment(x=0, xend=1, y=0, yend=1, lty=3, color='grey50') +
  scale_x_continuous(breaks = seq(0, 1, by=.20)) +
  scale_y_continuous(breaks = seq(0, 1, by=.20)) +
  ggtitle("ROC Curve")


## Using yardstick package
library(yardstick)  # for evaluation functions
# Notes:
# - for ROC curve and AUROC, it doesn't matter if the estimates/predictions
#   are p.hat or gamma
# - for


#: ROC plots
ROC = tibble(truth = factor(G.test, levels=c(1,0)), gamma) %>%
  yardstick::roc_curve(truth, gamma)

autoplot(ROC)  # autoplot() method

ROC %>%        # same as autoplot()
  ggplot(aes(1-specificity, sensitivity)) + geom_line() +
  geom_abline(lty=3) +
  coord_equal()


#: Area under ROC (AUROC)
tibble(truth = factor(G.test, levels=c(1,0)), gamma) %>%
  roc_auc(truth, gamma)
#> # A tibble: 1 x 3
#>   .metric .estimator .estimate
#>   <chr>   <chr>          <dbl>
#> 1 roc_auc binary         0.955

yardstick::roc_auc_vec(factor(G.test, 1:0), gamma)
#> [1] 0.9552
```

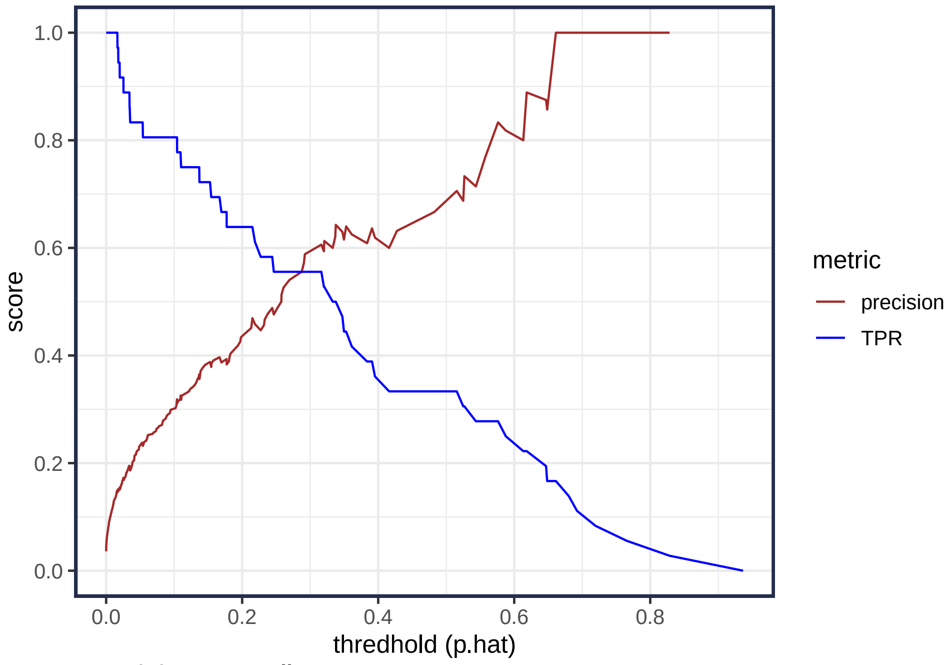


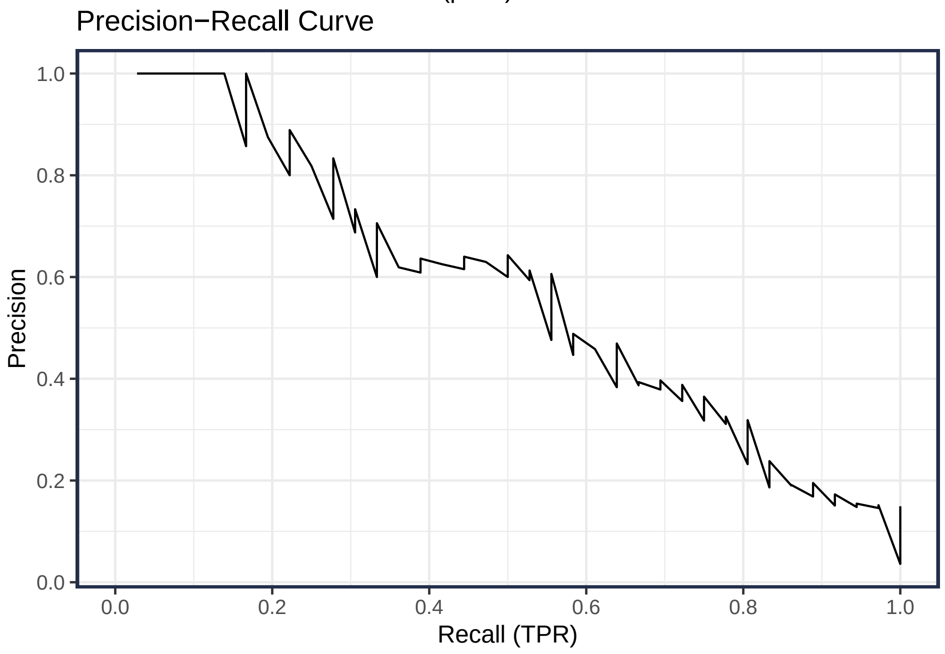




```
#: Log Loss Metric
yardstick::mn_log_loss_vec(factor(G.test, 1:0), p.hat)
#> [1] 0.08113

tibble(truth = factor(G.test, levels=c(1,0)), p.hat) %>%
  yardstick::mn_log_loss(truth, p.hat)
#> # A tibble: 1 x 3
#>   .metric     .estimator .estimate
#>   <chr>       <chr>          <dbl>
#> 1 mn_log_loss binary        0.0811
```

```
#: Precision-Recall
perf %>% mutate(threshold = p.hat, precision = TP/(TP + FP)) %>%
  select(threshold, TPR, precision) %>%
  gather(metric, n, -threshold) %>%
  ggplot(aes(threshold, n, color=metric)) + geom_line() +
  scale_x_continuous(breaks = seq(0, 1, by=.20)) +
  scale_y_continuous(breaks = seq(0, 1, by=.20)) +
  scale_color_manual(values = c(TPR="blue", precision="brown")) +
  labs(x="thredhold (p.hat)", y="score")
#> Warning: Removed 1 row containing missing values (`geom_line()`).

perf %>%
  mutate(threshold=p.hat, precision = TP/(TP + FP)) %>%
  ggplot(aes(TPR, precision)) + geom_line() +
  scale_x_continuous(breaks = seq(0, 1, by=.20)) +
  scale_y_continuous(breaks = seq(0, 1, by=.20)) +
  labs(x='Recall (TPR)', y='Precision', # (TP/(TP+FP))
       title="Precision-Recall Curve")
#> Warning: Removed 1 row containing missing values (`geom_line()`).
```


score
thredhold (p.hat)
metric
precision
TPR
0.0
0.2
0.4
0.6
0.8
1.0


Precision−Recall Curve
Precision
Recall (TPR)
0.0
0.2
0.4
0.6
0.8
1.0