

# Risk Modeling and Classification

Logistic Regression, ROC Curves, GAM

SYS 6018 | Spring 2023

classification.pdf

## Contents

<b>1</b>	<b>Risk Modeling Intro</b>	<b>2</b>
1.1	Credit Card Default data (Default) . . . . .	2
1.2	Set-up . . . . .	5
1.3	Binary Risk Modeling . . . . .	5
<b>2</b>	<b>Logistic Regression</b>	<b>9</b>
2.1	Basics . . . . .	9
2.2	Estimation . . . . .	9
2.3	Logistic Regression in Action . . . . .	11
2.4	Logistic Regression Summary . . . . .	12
<b>3</b>	<b>Evaluating Binary Risk Models</b>	<b>14</b>
3.1	Common Binary Loss Functions . . . . .	14
3.2	Model Comparison . . . . .	15
3.3	Calibration . . . . .	15
3.4	Area under the ROC curve (AUC or AUROC or C-Statistic) . . . . .	17
<b>4</b>	<b>Risk Scoring vs. Classification</b>	<b>18</b>
<b>5</b>	<b>Binary Classification</b>	<b>19</b>
5.1	Decision Theory . . . . .	19
5.2	Common Binary Loss Functions . . . . .	21
5.3	Performance Metrics . . . . .	21
5.4	Performance over a range of thresholds . . . . .	24
5.5	More than two classes . . . . .	30
5.6	Summary of Classification Evaluation . . . . .	30
<b>6</b>	<b>Generalized Additive Models (GAM)</b>	<b>32</b>
6.1	Generalized Additive Models (GAMs) . . . . .	35
6.2	Estimating $\hat{s}_j(x_j)$ with Backfitting . . . . .	36
<b>7</b>	<b>Appendix: R Code</b>	<b>37</b>

---

# 1 Risk Modeling Intro

## 1.1 Credit Card Default data (Default)

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

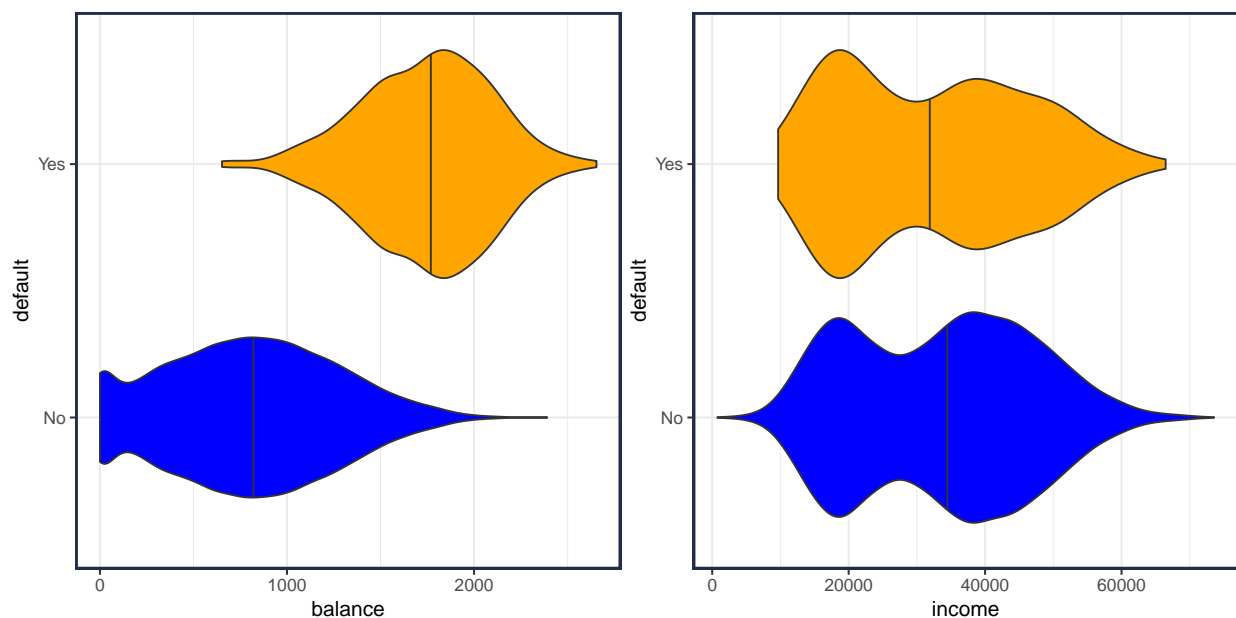
```
data(Default, package="ISLR")
```

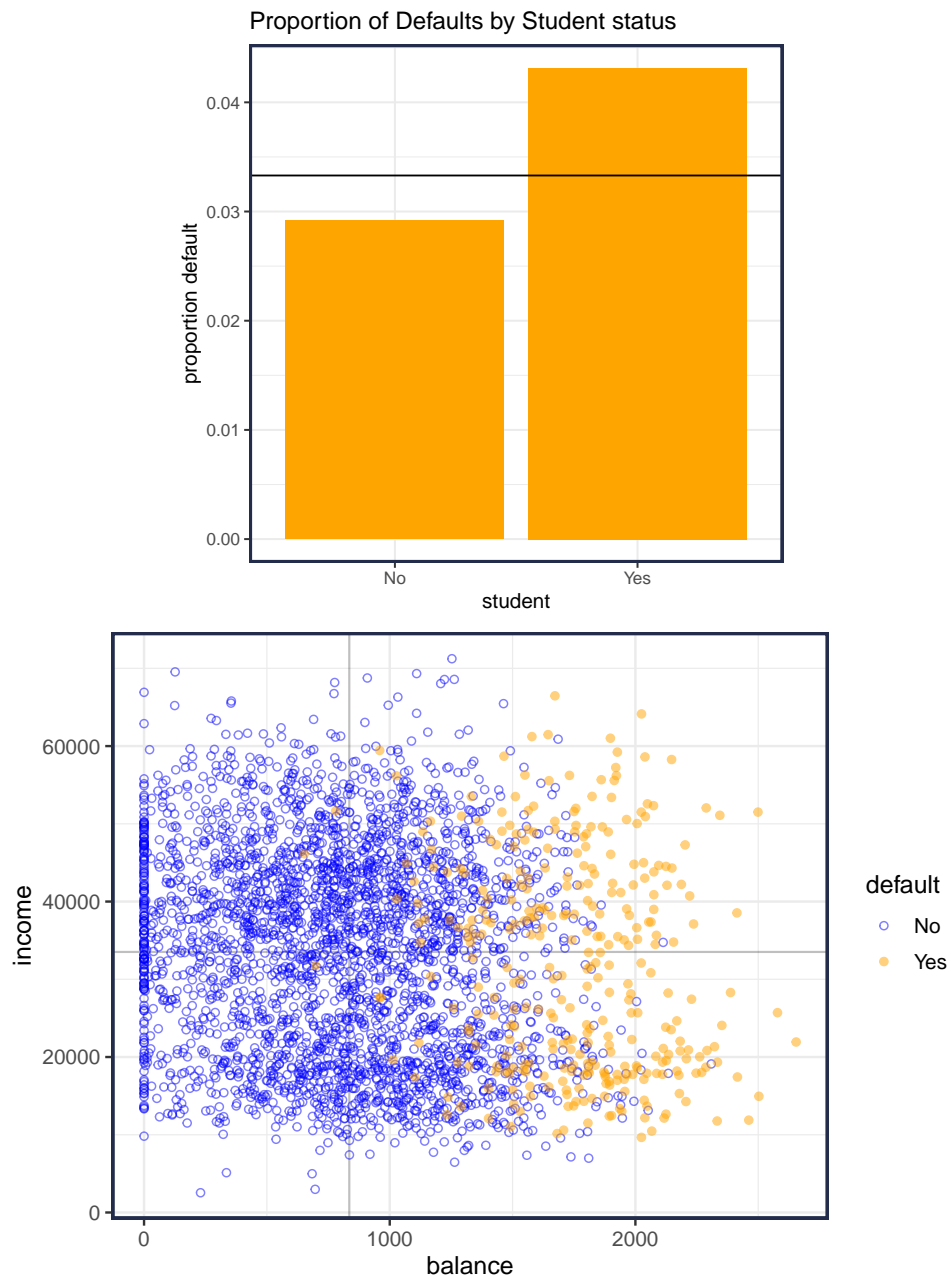
The variables are:

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

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

```
Default %>% summary
#>   default  student      balance      income
#>   No : 9667   No : 7056   Min.   :    0   Min.   :  772
#>   Yes:  333   Yes: 2944   1st Qu.:  482   1st Qu.:21340
#>                                     Median :  824   Median :34553
#>                                     Mean    :  835   Mean    :33517
#>                                     3rd Qu.:1166   3rd Qu.:43808
#>                                     Max.    :2654   Max.    :73554
```





### Your Turn #1 : Credit Card Default Modeling

How would you construct a model to predict the risk of default?

## 1.2 Set-up

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

## 1.3 Binary Risk Modeling

- 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:

$$Y_i = \begin{cases} 1 & G_i = G_1 \quad (\text{outcome of interest}) \\ 0 & G_i = G_2 \end{cases}$$

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

### 1.3.1 Linear Regression

- In this set-up we can run linear regression

$$\hat{y}(\mathbf{x}) = \hat{\beta}_0 + \sum_{j=1}^p \hat{\beta}_j x_j$$

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

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

term	estimate	std.error	statistic	p.value
(Intercept)	-0.08118	0.00838	-9.685	0.00000
studentYes	-0.01033	0.00566	-1.824	0.06817
balance	0.00013	0.00000	37.412	0.00000
income	0.00000	0.00000	1.039	0.29896

### Your Turn #2 : OLS for Binary Responses

1. For the binary  $Y$ , what is linear regression estimating?

2. What is the *loss function* that linear regression is using?
3. How could you create a *hard classification* from the linear model?
4. Does it make sense to use linear regression for binary risk modeling and classification?

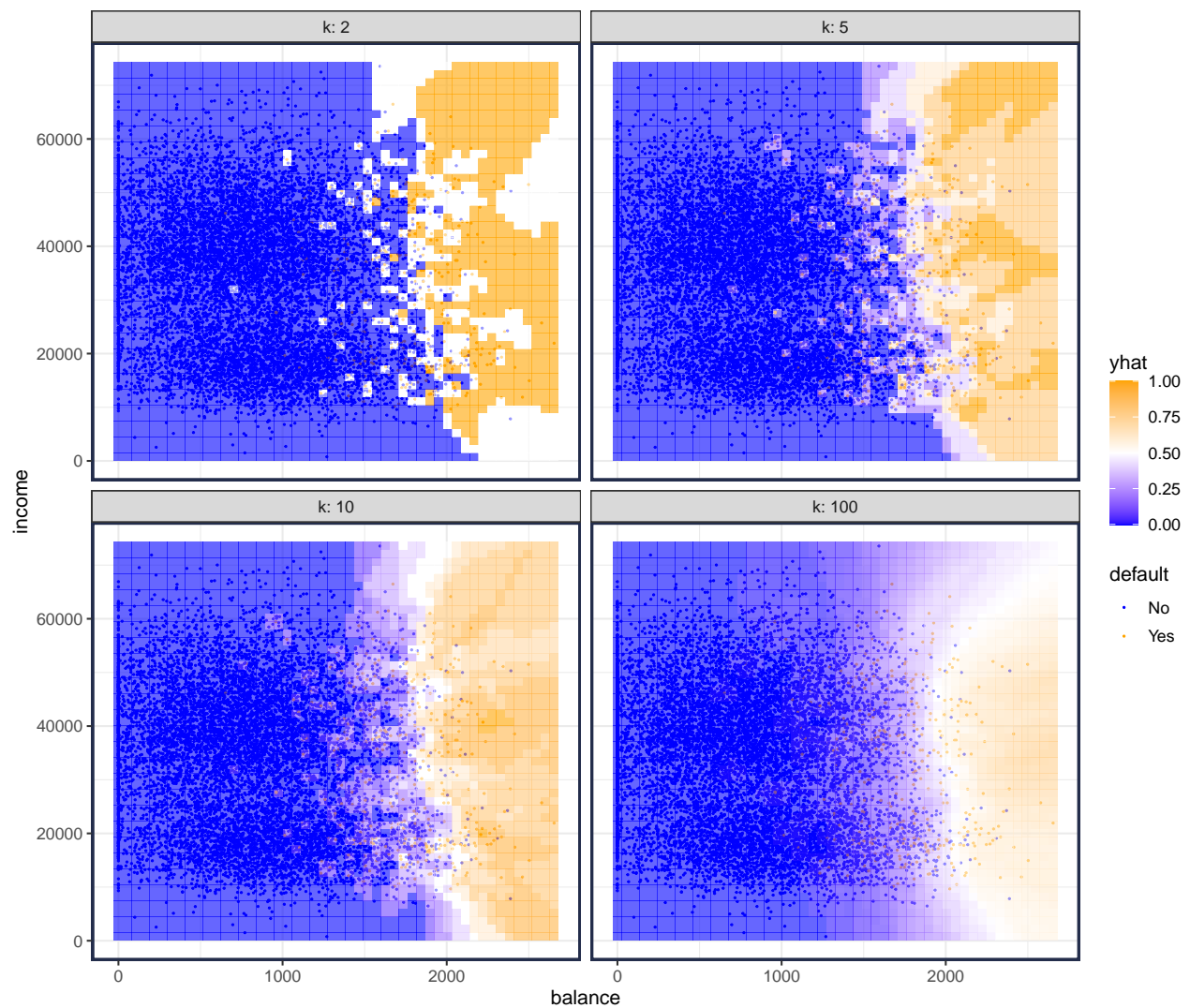
### 1.3.2 *k*-nearest neighbor (kNN)

- The *k*-NN method is a non-parametric *local* method, meaning that to make a prediction  $\hat{y}|x$ , it only uses the training data in the *vicinity* of  $x$ .
  - contrast with OLS linear regression, which uses all  $X$ 's to get prediction.
- The model (for regression and binary classification) is simple to describe

$$\begin{aligned} f_{\text{knn}}(x; k) &= \frac{1}{k} \sum_{i: x_i \in N_k(x)} y_i \\ &= \text{Avg}(y_i \mid x_i \in N_k(x)) \end{aligned}$$

- $N_k(x)$  are the set of  $k$  nearest neighbors
  - only the  $k$  closest  $y$ 's are used to generate a prediction
  - it is a *simple mean* of the  $k$  nearest observations
- When  $y$  is binary (i.e.,  $y \in \{0, 1\}$ ), the kNN model estimates

$$f_{\text{knn}}(x; k) \approx p(x) = \Pr(Y = 1 | X = x)$$



### Your Turn #3 : Thoughts about kNN

The above plots show a kNN model using the *continuous* predictors of *balance* and *income*.

- How could you use kNN with the categorical *student* predictor?

- The  $k$ -NN model also has a more general description when the outcome variables is categorical  $G_i \in \mathcal{G}$

$$f_g^{\text{knn}}(x; k) = \frac{1}{k} \sum_{i: x_i \in N_k(x)} \mathbb{1}(g_i = g)$$

$$= \widehat{\Pr}(G_i = g \mid x_i \in N_k(x))$$

- $N_k(x)$  are the set of  $k$  nearest neighbors

- only the  $k$  closest  $y$ 's are used to generate a prediction
- it is a *simple proportion* of the  $k$  nearest observations that are of class  $g$



## 2 Logistic Regression

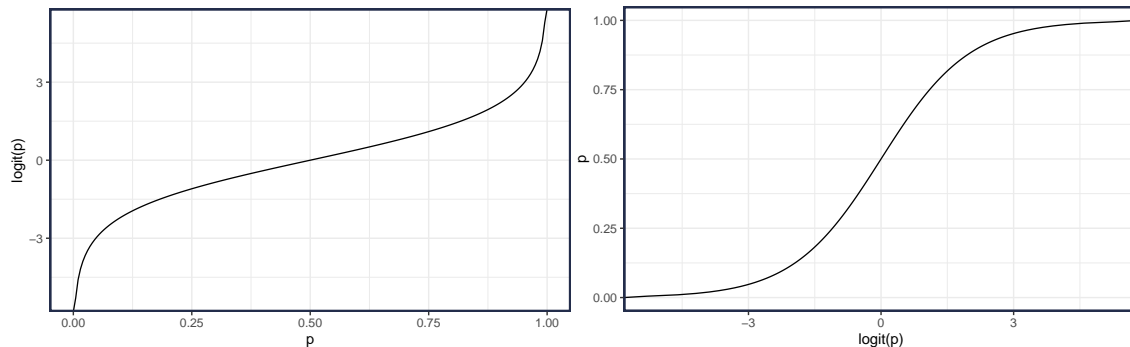
### 2.1 Basics

- Let  $0 \leq p \leq 1$  be a probability.
- The log-odds of  $p$  is called the *logit*

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right)$$

- The inverse logit is the *logistic function*. Let  $f = \text{logit}(p)$ , then

$$\begin{aligned} p &= \frac{e^f}{1 + e^f} \\ &= \frac{1}{1 + e^{-f}} \end{aligned}$$



- For binary outcome variables  $Y \in \{0, 1\}$ , **Linear Regression** models

$$E[Y | X = x] = \Pr(Y = 1 | X = x) = \beta^T x$$

- Alternatively, **Logistic Regression** models

$$\text{logit} \Pr(Y = 1 | X = x) = \log\left(\frac{\Pr(Y = 1 | X = x)}{1 - \Pr(Y = 1 | X = x)}\right) = \beta^T x$$

and thus,

$$\begin{aligned} \Pr(Y = 1 | X = x) &= \frac{e^{\beta^T x}}{1 + e^{\beta^T x}} \\ &= \left(1 + e^{-\beta^T x}\right)^{-1} \end{aligned}$$

### 2.2 Estimation

- The input data for logistic regression are:  $(\mathbf{x}_i, y_i)_{i=1}^n$  where  $y_i \in \{0, 1\}$ ,  $\mathbf{x}_i = (x_{i0}, x_{i1}, \dots, x_{ip})^T$ .
- $y_i | \mathbf{x}_i \sim \text{Bern}(p_i(\beta))$

- $p_i(\beta) = \Pr(Y = 1 | \mathbf{X} = \mathbf{x}_i; \beta) = \left(1 + e^{-\beta^T \mathbf{x}_i}\right)^{-1}$
- where  $\beta^T \mathbf{x}_i = \mathbf{x}_i^T \beta = \beta_0 + \sum_{j=1}^p x_{ij} \beta_j$

- Bernoulli Likelihood Function

$$L(\beta) = \prod_{i=1}^n p_i(\beta)^{y_i} (1 - p_i(\beta))^{1-y_i}$$

$$\begin{aligned} \log L(\beta) &= \sum_{i=1}^n \{y_i \ln p_i(\beta) + (1 - y_i) \ln(1 - p_i(\beta))\} \\ &= \sum_{i=1}^n \begin{cases} \ln p_i(\beta) & y_i = 1 \\ \ln(1 - p_i(\beta)) & y_i = 0 \end{cases} \\ &= \sum_{i:y_i=1} \ln p_i(\beta) + \sum_{i:y_i=0} \ln(1 - p_i(\beta)) \end{aligned}$$

- The usual approach to estimating the Logistic Regression coefficients is *maximum likelihood*

$$\begin{aligned} \hat{\beta} &= \arg \max_{\beta} L(\beta) \\ &= \arg \max_{\beta} \log L(\beta) \end{aligned}$$

- We can also view this as the coefficients that minimize the *loss function*  $\ell(\beta)$ , where the loss function is the negative log-likelihood

$$\hat{\beta} = \arg \min_{\beta} \ell(\beta)$$

using loss  $\ell(\beta) = -C \log L(\beta)$  where  $C > 0$  is some positive constant, e.g.,  $C = 1/n$

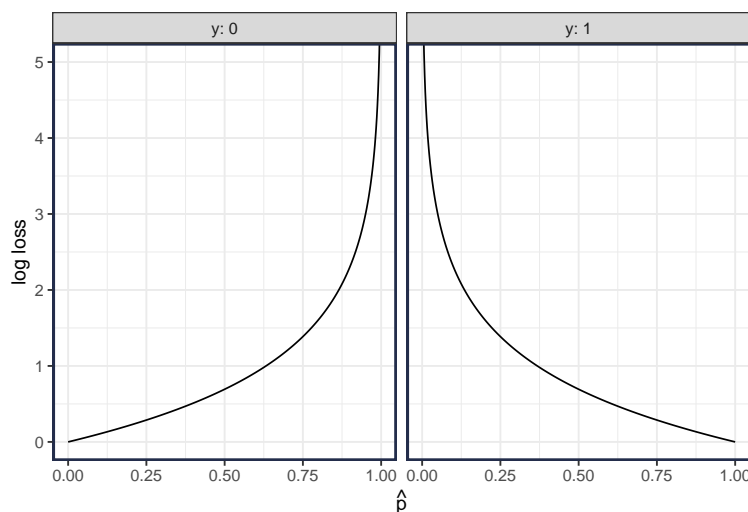
- This view facilitates *penalized logistic regression*

$$\hat{\beta} = \arg \min_{\beta} \ell(\beta) + \lambda P(\beta)$$

---

Ridge Penalty	$P(\beta) = \ \beta\ _2^2 = \sum_{j=1}^p  \beta_j ^2 = \beta^\top \beta$
Lasso Penalty	$P(\beta) = \ \beta\ _1 = \sum_{j=1}^p  \beta_j $
Best Subsets	$P(\beta) = \ \beta\ _0 = \sum_{j=1}^p  \beta_j ^0 = \sum_{j=1}^p 1_{(\beta_j \neq 0)}$
Elastic Net	$P(\beta, \alpha) = (1 - \alpha) \ \beta\ _2^2 / 2 + \alpha \ \beta\ _1 = \sum_{j=1}^p (1 - \alpha)  \beta_j ^2 / 2 + \alpha  \beta_j $

---



## 2.3 Logistic Regression in Action

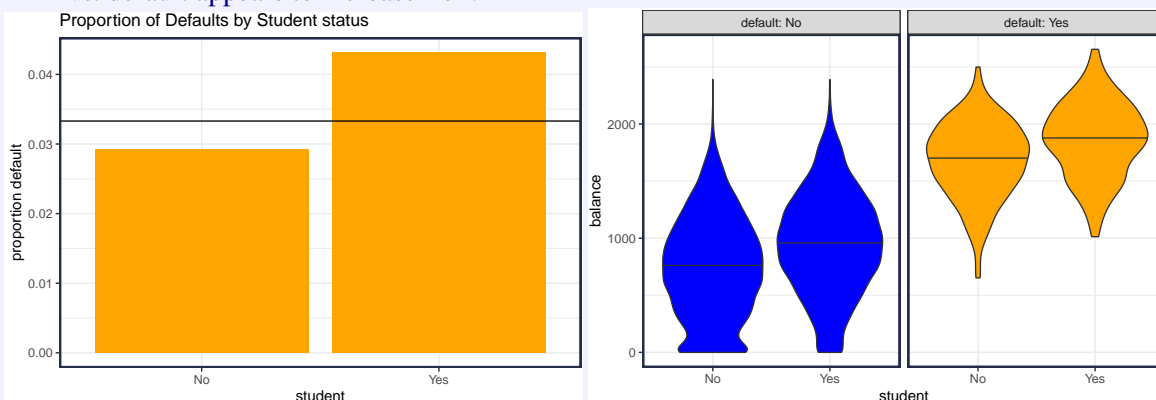
- In **R**, logistic regression can be implemented with the `glm()` function since it is a type of *Generalized Linear Model*.
- Because logistic regression is a special case of *Binomial* regression, use the `family=binomial()` argument

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

term	estimate	std.error	statistic	p.value
(Intercept)	-10.869	0.492	-22.080	0.000
studentYes	-0.647	0.236	-2.738	0.006
balance	0.006	0.000	24.738	0.000
income	0.000	0.000	0.370	0.712

### Your Turn #4 : Interpreting Logistic Regression

1. What is the estimated probability of default for a Student with a balance of \$1000?
2. What is the estimated probability of default for a *Non-Student* with a balance of \$1000?
3. Why does `student=Yes` appear to lower risk of default, when the plot of student status vs. default appears to increase risk?

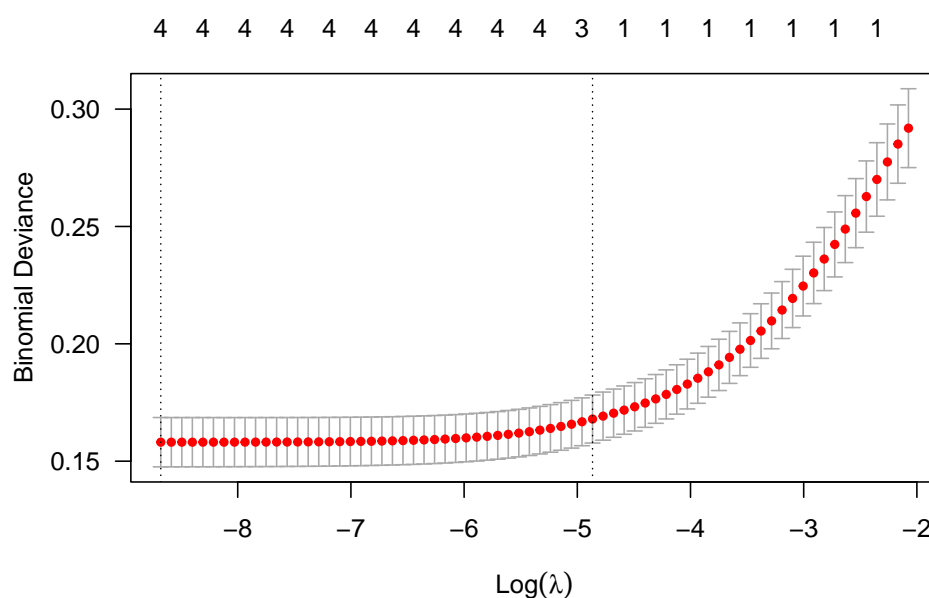


### 2.3.1 Penalized Logistic Regression

- The `glmnet()` package can estimate logistic regression using an elastic net penalty (e.g., ridge, lasso).

```
#: Fit *penalized* logistic regression model
library(glmnet)
set.seed(2020)
X = makeX(select(Default, student, balance, income))
Y = Default$y
fit.enet = cv.glmnet(X, Y,
                     alpha=.5,
                     family="binomial")

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



term	unpenalized	lambda.min	lambda.1se
(Intercept)	-10.869	-11.056	-7.937
studentYes	-0.647	-0.299	-0.041
balance	0.006	0.006	0.004
income	0.000	0.000	0.000
studentNo	NA	0.325	0.044

## 2.4 Logistic Regression Summary

- Logistic Regression (both penalized and unpenalized) estimates a *posterior probability*,  $\hat{p}(x) = \widehat{\Pr}(Y = 1 \mid X = x)$
- This estimate is a function of the estimated coefficients

$$\begin{aligned}\hat{p}(x) &= \frac{e^{\hat{\beta}^\top x}}{1 + e^{\hat{\beta}^\top x}} \\ &= \left(1 + e^{-\hat{\beta}^\top x}\right)^{-1}\end{aligned}$$

**Your Turn #5**

1. Given a person's `student` `status`, `balance`, and `income`, how could you use Logistic Regression to decide if they will `default`? (i.e., make a hard classification)

### 3 Evaluating Binary Risk Models

#### 3.1 Common Binary Loss Functions

- Suppose we are going to predict a binary outcome  $Y \in \{0, 1\}$  with  $\hat{p}(x) \in \{0, 1\}$ .

- Call  $\hat{p}(x)$  the *risk score*

- **Brier Score / Squared Error**

$$\begin{aligned} L(y, \hat{p}) &= (y - \hat{p})^2 \\ &= \begin{cases} (1 - \hat{p})^2 & y = 1 \\ \hat{p}^2 & y = 0 \end{cases} \end{aligned}$$

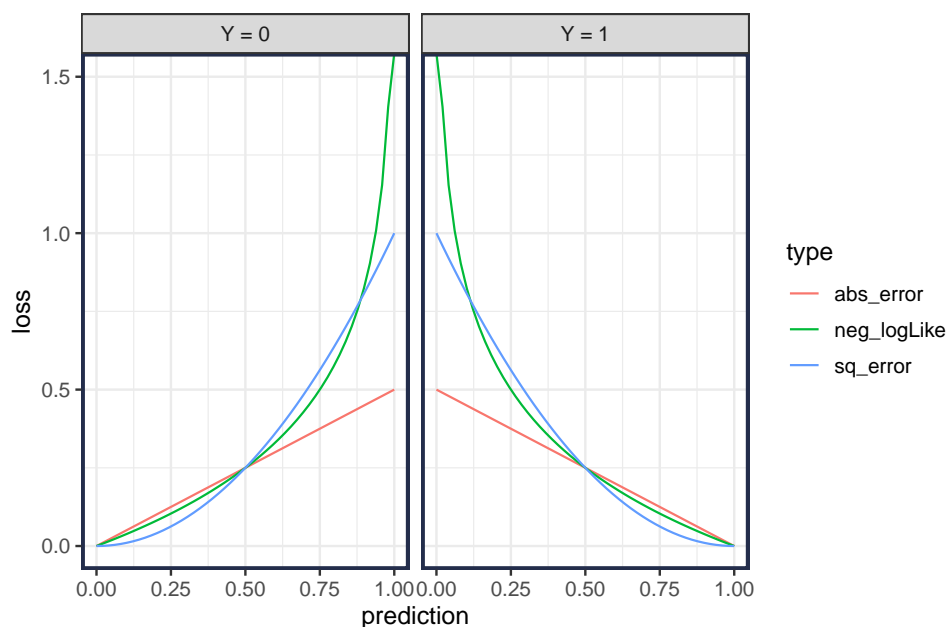
- **Absolute Error**

$$\begin{aligned} L(y, \hat{p}) &= |y - \hat{p}| \\ &= \begin{cases} 1 - \hat{p} & y = 1 \\ \hat{p} & y = 0 \end{cases} \end{aligned}$$

- **Bernoulli negative log-likelihood (Log-Loss)**

- This is the loss function for Logistic Regression

$$\begin{aligned} L(y, \hat{p}) &= -\{y \log \hat{p} + (1 - y) \log(1 - \hat{p})\} \\ &= \begin{cases} -\log \hat{p} & y = 1 \\ -\log(1 - \hat{p}) & y = 0 \end{cases} \end{aligned}$$



## 3.2 Model Comparison

```
#: Evaluation Function
evaluate <- function(p_hat, y){
  tibble(
    mn_log_loss = -mean(dbinom(y, prob = p_hat, size=1, log=TRUE)),
    mse = mean((y-p_hat)^2),
    mae = mean(abs(y-p_hat))
  )
}

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

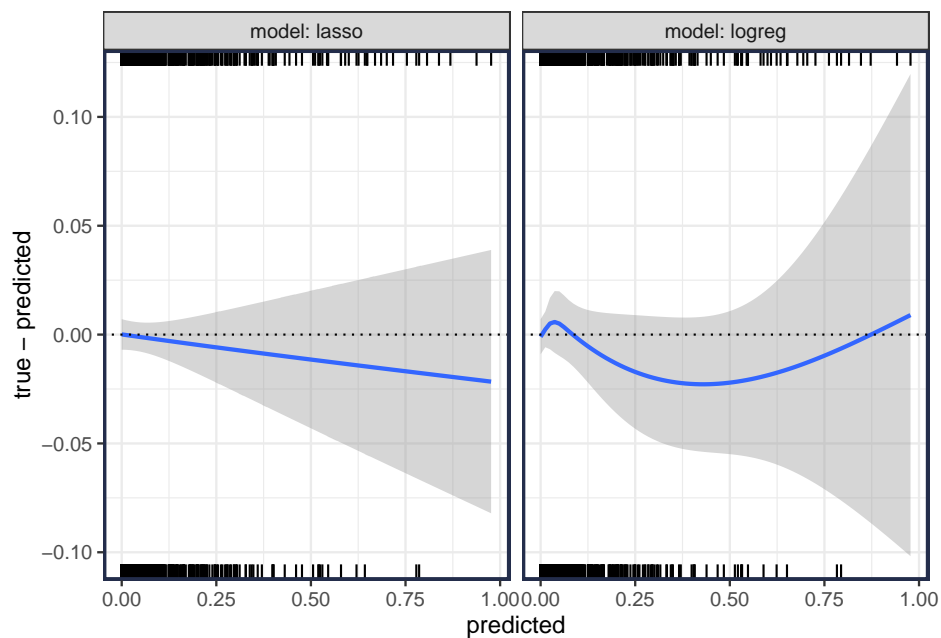
#: fit logistic regression on training data
fit.lm = glm(y~student + balance + income, family='binomial',
             data=Default[train, ])
p_hat.lm = predict(fit.lm, Default[test,], type="response")
evaluate(p_hat.lm, y = Default$y[test])
#> # A tibble: 1 x 3
#>   mn_log_loss    mse    mae
#>   <dbl>    <dbl> <dbl>
#> 1      0.0815 0.0228 0.0457

#: Fit lasso logistic regression (choose lambda with 10-fold cv)
X = glmnet::makeX(select(Default, student, balance, income))
Y = Default$y
fit.lasso = cv.glmnet(X[train,], Y[train], alpha = 1, family = "binomial")
p_hat.lasso = predict(fit.lasso, X[test,], type="response", s = "lambda.min")
evaluate(p_hat.lasso, y=Y[test])
#> # A tibble: 1 x 3
#>   mn_log_loss    mse    mae
#>   <dbl>    <dbl> <dbl>
#> 1      0.0815 0.0228 0.0458
```

## 3.3 Calibration

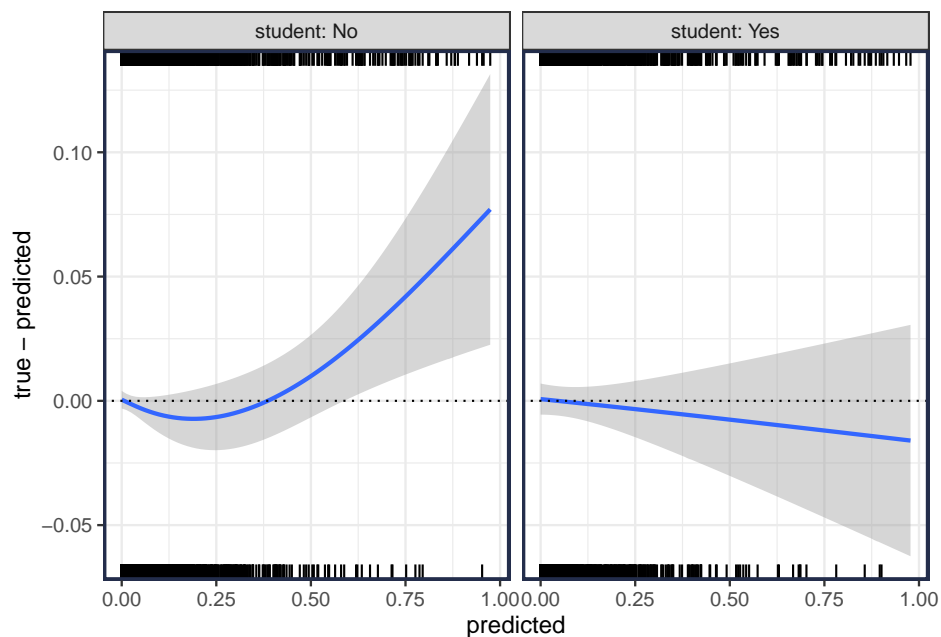
A risk model is said to be *calibrated* if the predicted probabilities are equal to the true risk (probabilities).

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



Calibration plots can be used to measure drift, fairness, and model/algorithmic bias. Consider comparing the predictive performance of our models for Students and Non-Students.

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





### 3.3.1 Estimating Calibration

To measure mis-calibration, we can treat the predictions as features and use the predictions as an offset. E.g., to check for linear deviation

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

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

We will revisit calibration when we cover Support Vector Machines (SVM) to convert a generic score to a probability.

Some additional resources:

- [Platt Scaling](#)
- [Isotonic Regression](#)
- [Calibration: the Achilles heel of predictive analytics](#)
  - Article on clinical decision-making
- [The Measure and Mismeasure of Fairness: A Critical Review of Fair Machine Learning](#)

### 3.4 Area under the ROC curve (AUC or AUROC or C-Statistic)

The AUROC of a risk model is: the probability that the model will rank a randomly chosen positive example ( $Y = 1$ ) higher than a randomly chosen negative example ( $Y = 0$ ), i.e.

$$\text{AUROC} = \Pr(\hat{p}(x_1) > \hat{p}(x_0))$$

- where  $x_k$  is a randomly chosen example from class  $Y = k$ .

To estimate the AUROC you will fit a model to training data and make predictions on hold-out (test) data with known labels. Hopefully the model will assign large probabilities to the outcome of interest ( $Y = 1$ ) and low probabilities to the other class. Then compare the probabilities between all pairs of observations where one comes from the  $Y = 1$  set and the other from the  $Y = 0$  set. The AUROC is the proportion of the pairs where the estimated probability for the outcome of interest is larger than the probability for the other outcome. The extra term is to handle ties in predicted probability.

$$\widehat{\text{AUROC}} = \frac{1}{n_1 n_0} \sum_{i: y_i=1} \sum_{j: y_j=0} \mathbb{1}(\hat{p}_i > \hat{p}_j) + \frac{1}{2} \mathbb{1}(\hat{p}_i = \hat{p}_j)$$

The AUROC assesses the discrimination ability of the model. It gives a different assessment on model performance from calibration. Notice that the AUROC is the same for any monotonic transformation of the estimated probabilities. E.g., we can use  $\hat{p}$  or  $\log(\hat{p})$  or  $\text{logit}(\hat{p})$  or  $\hat{p}/10$  and still get the same AUROC. But calibration assesses how closely the estimated probabilities match the actual probabilities as well as helping to identify the regions in feature space where the predictions are poor.

- A helpful discussion on AUROC: <https://stats.stackexchange.com/questions/145566/how-to-calculate-area-under-the-curve-auc-or-the-c-statistic-by-hand>
- We will say more about ROC curves later in the notes.

## 4 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  $\hat{p}(X)$ , where  $\hat{p}(X) = [\hat{p}_1(X), \dots, \hat{p}_K(X)]$
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.

### 4.0.1 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 #6 : 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?

## 5 Binary Classification

### 5.1 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:  $L(\text{True Label}, \text{Estimated Label}) = L(G, \hat{G})$
- A model's *Expected Prediction Error (EPE)* (also called *Risk*) at input  $X$  is the expected loss on new data with input  $X$ .
- The EPE (for a binary outcome) is:

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

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

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

#### 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.

#### 5.1.1 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.
  - $L(G = 0, \hat{G} = 1) = C_{FP}$ : There is a cost of  $C_{FP}$  units if the model issues a *false positive*, estimating the patient has cancer when they don't.
  - $L(G = 1, \hat{G} = 0) = C_{FN}$ : 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  $C_{FP}$  ( $C_{FN} \gg C_{FP}$ ) 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

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

- The ratio of  $C_{FP}$  to  $C_{FN}$  is all that matters for the decision. Let's say that  $C_{FP} = 1$  and  $C_{FN} = 10$ . 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.

### 5.1.2 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  $p(x)$ :

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

- Table of equivalent representations:

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

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

### 5.1.3 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)

## 5.2 Common Binary Loss Functions

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

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

- This requires that a *hard classification* is made.
- The theoretically optimal prediction is  $G^*(x) = \mathbb{1}(p(x) > C_{FP}/(C_{FP} + C_{FN}))$ .

- 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  $G^*(x) = \mathbb{1}(p(x) > 1 - p(x))$  which is equivalent to  $\mathbb{1}(p(x) > 0.50)$

## 5.3 Performance Metrics

### 5.3.1 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		
		$G = 1$	$G = 0$	
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
```

### 5.3.2 Metrics

There are lots of common evaluation metrics that can be calculated from the confusion matrix:

Metric	Definition	Estimate
Risk/Exp.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)	$P_X(G(X) = 1 \mid \hat{G}_t(X) = 1)$	$\frac{1}{\hat{N}_1(t)} \sum_{i:\hat{G}_t(x_i)=1} \mathbb{1}(G_i = 1)$

$N$  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:

Sources: [13][14][15][16][17][18][19][20] view • talk • edit

		Predicted condition			
		Positive (PP)	Negative (PN)	Informedness, bookmaker informedness (BM) $= \text{TPR} + \text{TNR} - 1$	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 $= \frac{\text{TP}}{P} = 1 - \text{FNR}$	False negative rate (FNR), miss rate $= \frac{\text{FN}}{P} = 1 - \text{TPR}$
	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 $= \frac{\text{FP}}{N} = 1 - \text{TNR}$	True negative rate (TNR), specificity (SPC), selectivity $= \frac{\text{TN}}{N} = 1 - \text{FPR}$
	Prevalence $= \frac{P}{P + N}$	Positive predictive value (PPV), precision $= \frac{\text{TP}}{\text{PP}} = 1 - \text{FDR}$	False omission rate (FOR) $= \frac{\text{FN}}{\text{PN}} = 1 - \text{NPV}$	Positive likelihood ratio (LR+) $= \frac{\text{TPR}}{\text{FPR}}$	Negative likelihood ratio (LR-) $= \frac{\text{FNR}}{\text{TNR}}$
	Accuracy (ACC) $= \frac{\text{TP} + \text{TN}}{P + N}$	False discovery rate (FDR) $= \frac{\text{FP}}{\text{PP}} = 1 - \text{PPV}$	Negative predictive value (NPV) $= \frac{\text{TN}}{\text{PN}}$ $= 1 - \text{FOR}$	Markedness (MK), deltaP ( $\Delta p$ ) $= \text{PPV} + \text{NPV} - 1$	Diagnostic odds ratio (DOR) $= \frac{\text{LR+}}{\text{LR-}}$
	Balanced accuracy (BA) $= \frac{\text{TPR} + \text{TNR}}{2}$	F <sub>1</sub> 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) $= \frac{\sqrt{\text{TPR} \times \text{TNR} \times \text{PPV} \times \text{NPV}}}{\sqrt{\text{FNR} \times \text{FPR} \times \text{FOR} \times \text{FDR}}}$	Threat score (TS), critical success index (CSI), Jaccard index $= \frac{\text{TP}}{\text{TP} + \text{FN} + \text{FP}}$

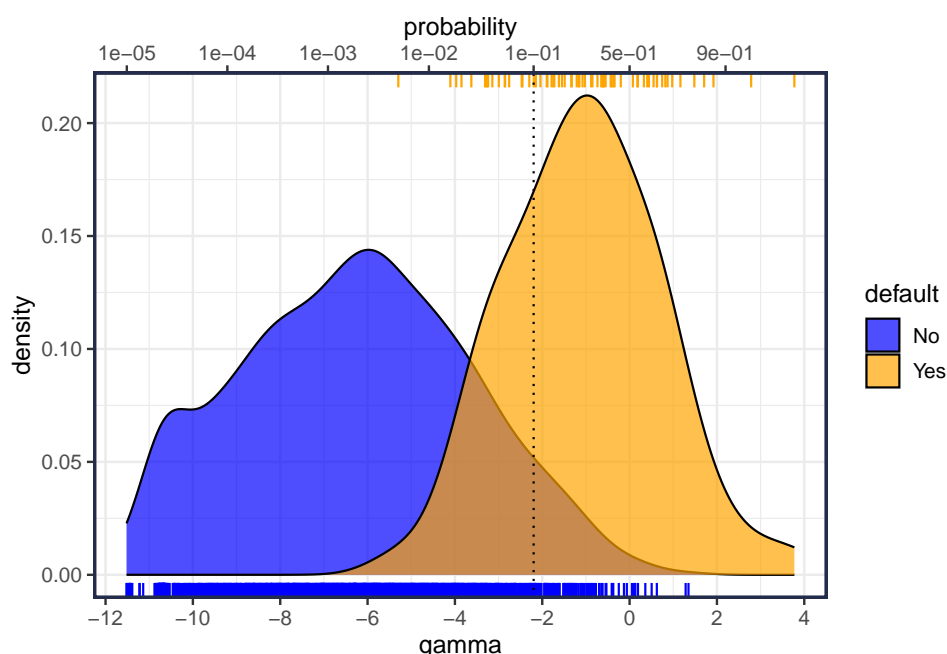
## 5.4 Performance over a range of thresholds

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.

```
#: 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  $\gamma(x) = 0 \rightarrow \Pr(Y = 1 | X = x) = 1/2$
  - $\gamma(x) = \log p(x)/(1 - p(x))$



- 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)
```

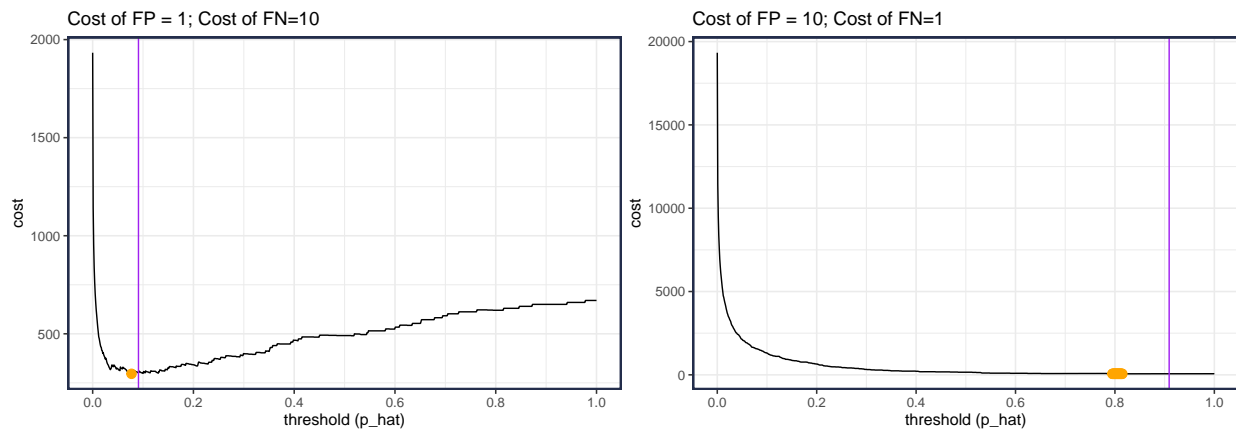
p_hat	gamma_hat	TN	FN	TP	FP	TPR	FPR
0.000	-Inf	0	0	67	1933	1.000	1.000
0.001	-6.906	789	0	67	1144	1.000	0.592
0.002	-6.212	979	0	67	954	1.000	0.494
0.003	-5.805	1095	0	67	838	1.000	0.434
0.004	-5.516	1171	0	67	762	1.000	0.394
0.005	-5.292	1238	1	66	695	0.985	0.360

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 convenient.

#### 5.4.1 Cost Curves

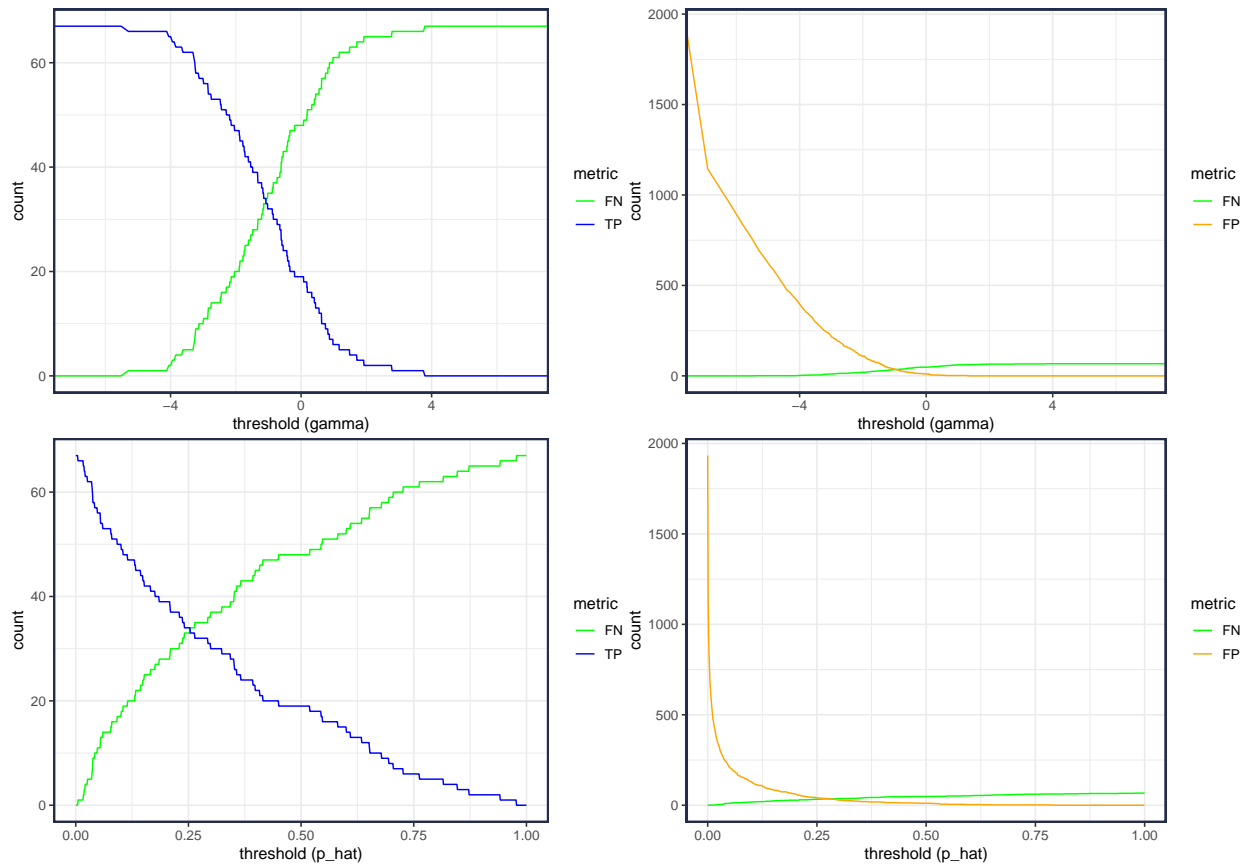
- Under the usual scenario where  $L(0,0) = L(1,1) = 0$ , the cost only depends on the ratio of false positive costs ( $C_{FP}$ ) to false negative costs ( $C_{FN}$ ).
- note: the **purple** is the *theoretical* optimal threshold (using  $t^* = \log C_{FP}/C_{FN}$  for  $\hat{\gamma}(x)$  and  $C_{FP}/(C_{FP}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



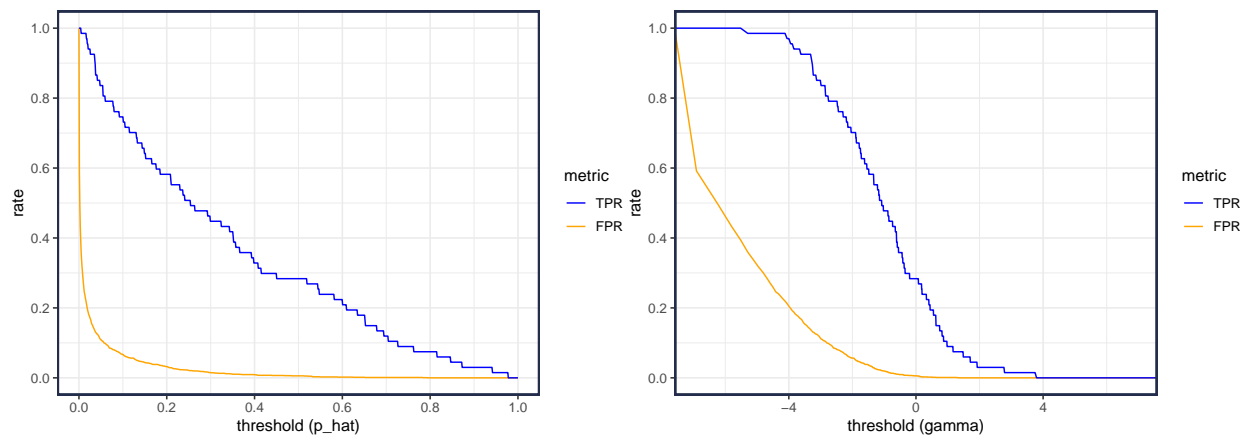
### Optimal Threshold

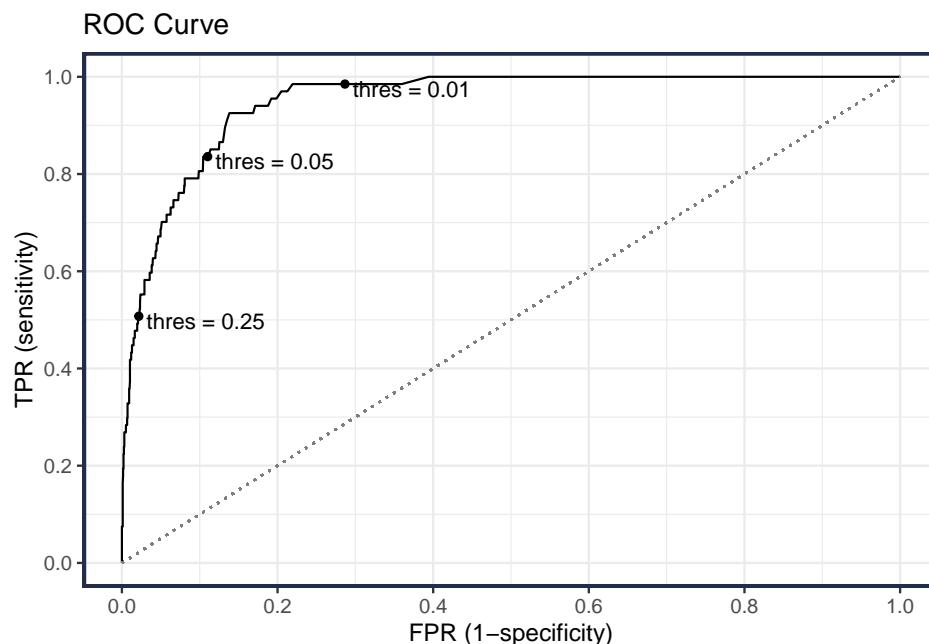
- The *theoretically* optimal threshold is based on the *true*  $\gamma(x) = \log \frac{p(x)}{1-p(x)}$  (for a given cost ratio of FP to FN)
- The observed optimal threshold will differ when the model's estimate  $\hat{\gamma}(x) \neq \gamma(x)$ 
  - Hopefully, they are close and it won't make much difference 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.

### 5.4.2 General Performance as function of threshold (select metrics)



### 5.4.3 ROC Curves (Receiver Operating Characteristic)



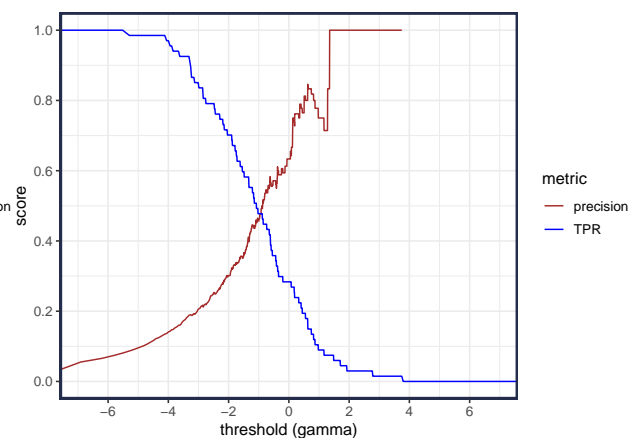
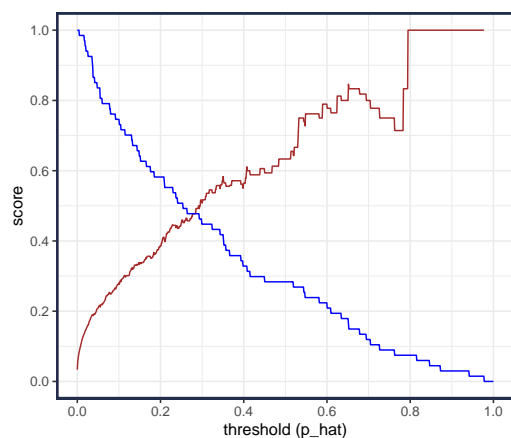


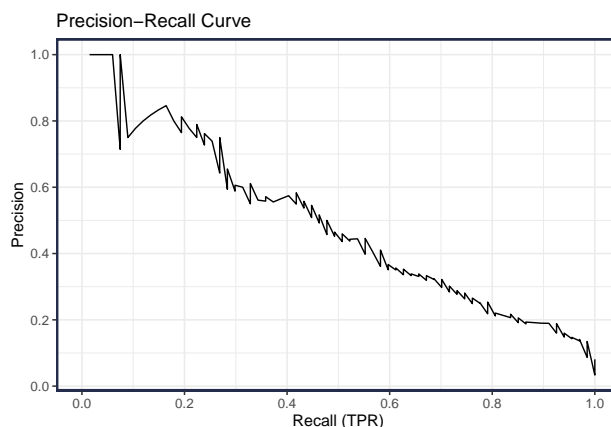
## AUROC

- 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](#)

### 5.4.4 Precision Recall Curves

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





### 5.4.5 R Code

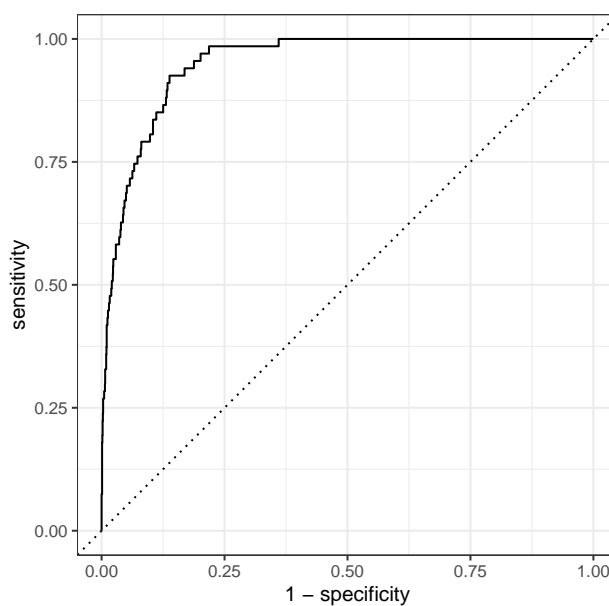
Once we have the FP, TP, TN, FN values for a set of thresholds (like what is in the `perf` object), then we have everything we need to calculate any metric (e.g., gain, lift, F1, ...).

- 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](#)

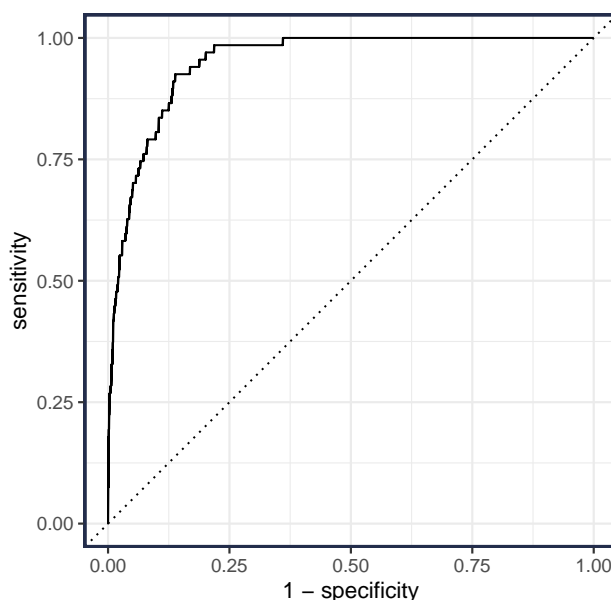
```
library(yardstick) # for evaluation functions

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

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



```
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_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
```

## 5.5 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/Risk:

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

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

## 5.6 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  $\text{cost}(\text{FP})/\text{cost}(\text{FN})$  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

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

- 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.)

## 6 Generalized Additive Models (GAM)

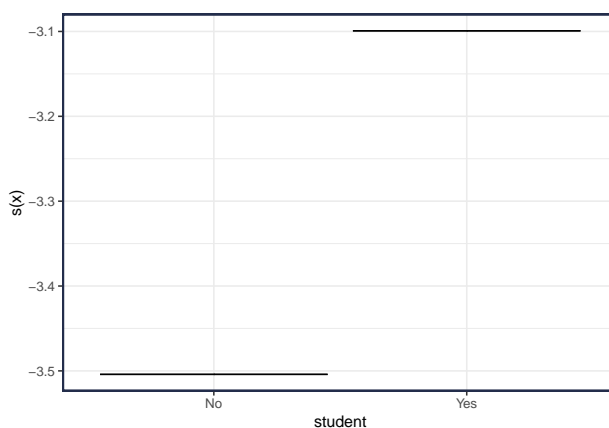
In our discussion of **Basis Expansions**, we covered how the relationship between a single raw predictor  $x$  and the outcome could be made more complex with basis expansions.

### • 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

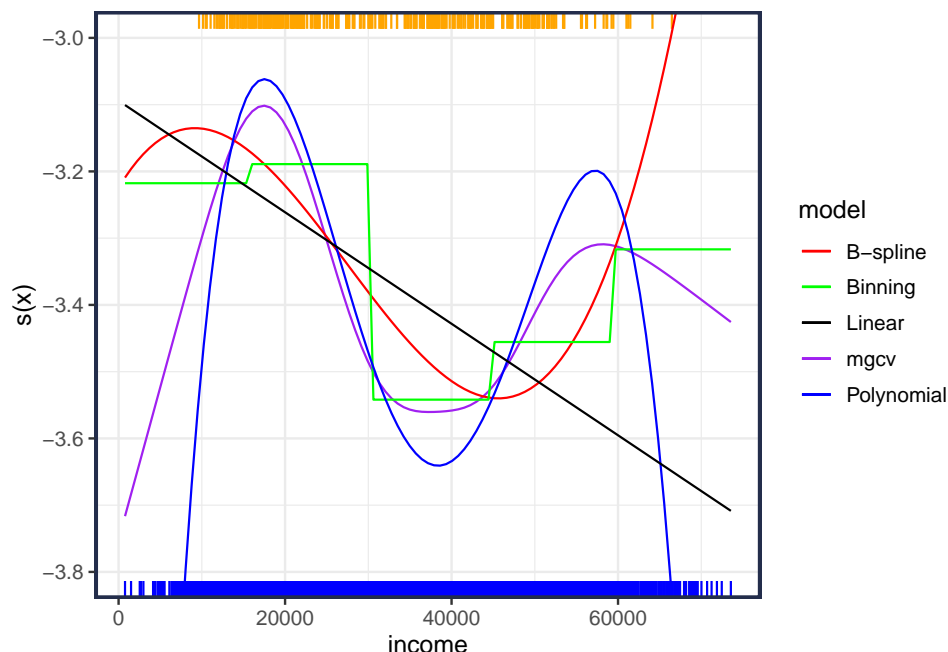
```
library(mgcv)
#: Fit penalized spline, it will select best edf
# specify smooth with s()
fit.5 = gam(y~s(income), data=Default, family="binomial")
summary(fit.5)
#>
```

```
#> 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"))
```



## 6.1 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.

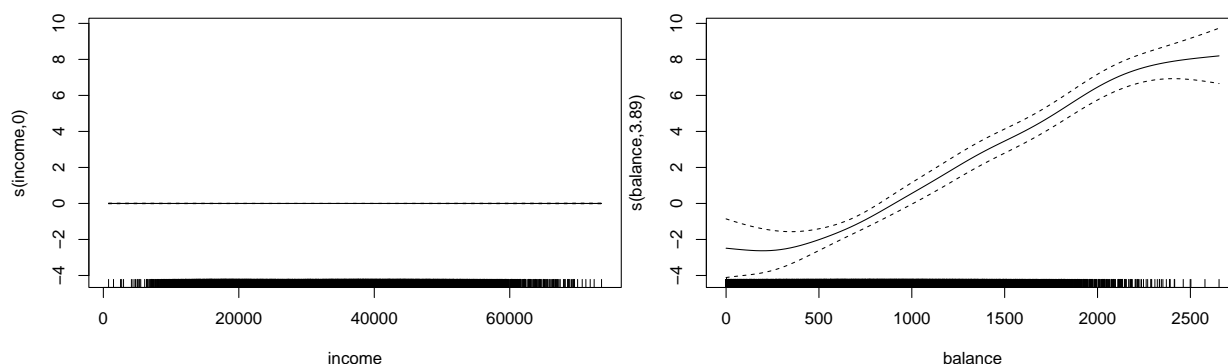
- For binary classification setting:  $\text{logit}p(\mathbf{x}) = f(\mathbf{x})$
- These are *additive* models because each term adds its contribution, although potentially in a non-linear way
  - But interactions can still be accommodated using `s(x1, x2)` or `s(x1, by=fac)`
- These are *generalized* 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)
```



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

The smooth terms of a GAM model can be estimated 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.

## 7 Appendix: R Code

### Set-up

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

#-----#
#: 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

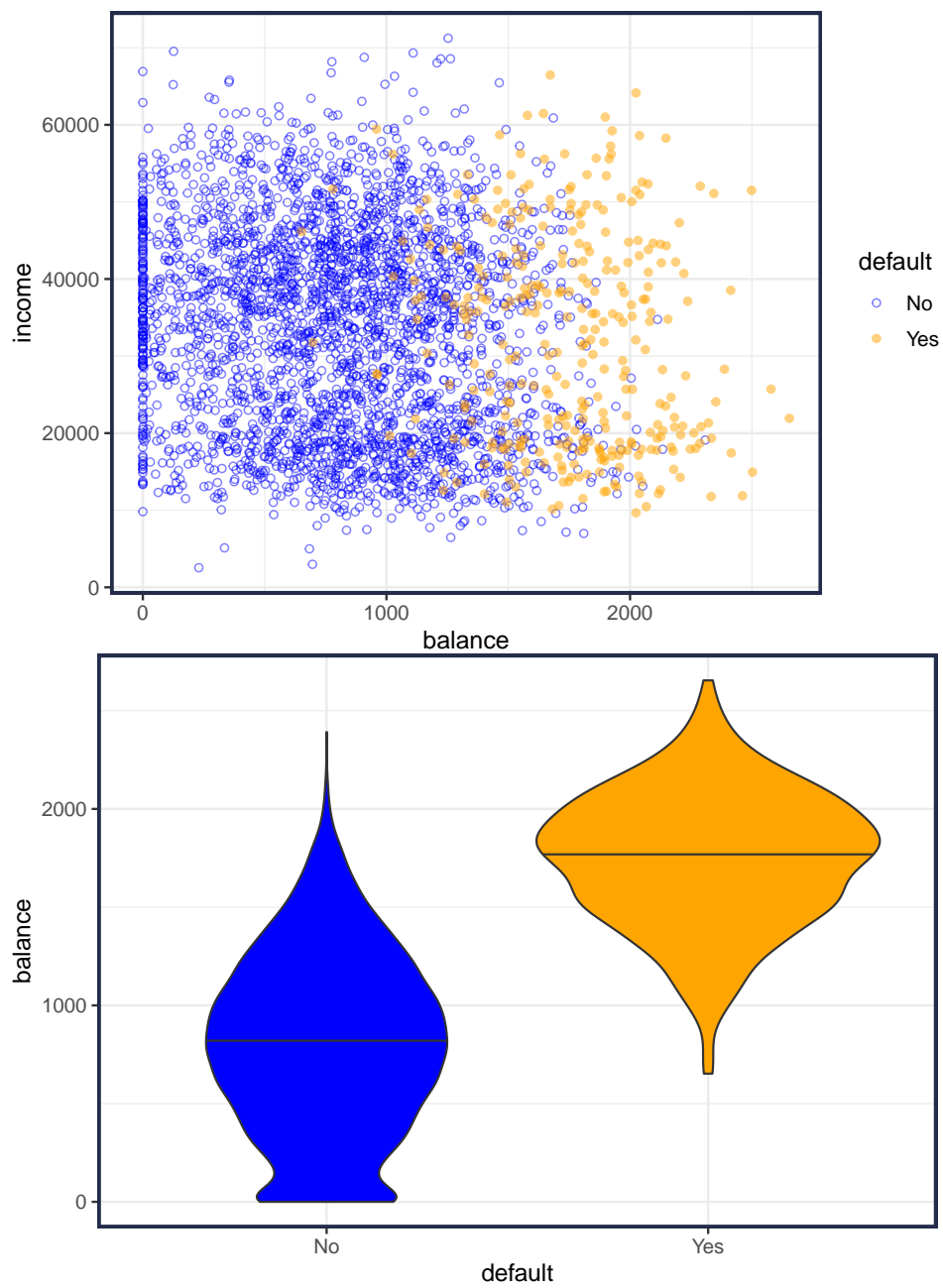
#: Plots
plot_cols = c(Yes="orange", No="blue") # set colors

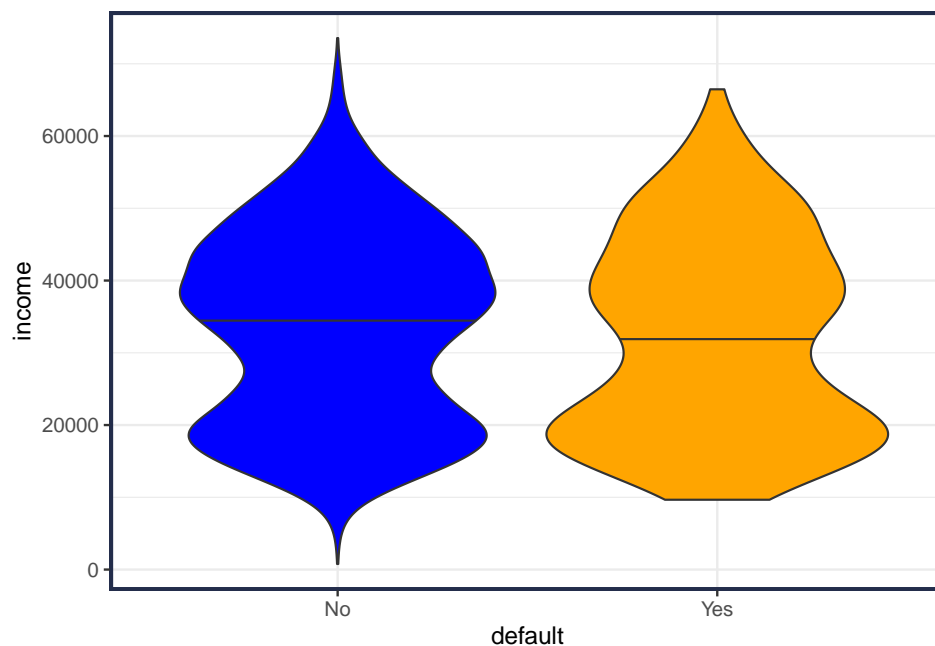
Default %>% group_by(default) %>% slice(1:3000) %>% # choose max of 3000 from each group
ggplot(aes(balance, income, color=default, shape=default)) +
  geom_point(alpha=.5) +
  scale_color_manual(values=plot_cols) +
  scale_shape_manual(values=c(Yes=19, No=1))

ggplot(Default, aes(default, balance, fill=default)) +
  geom_violin(draw_quantiles=.5) + #alternative: geom_boxplot() +
  scale_fill_manual(values=plot_cols, guide=FALSE)

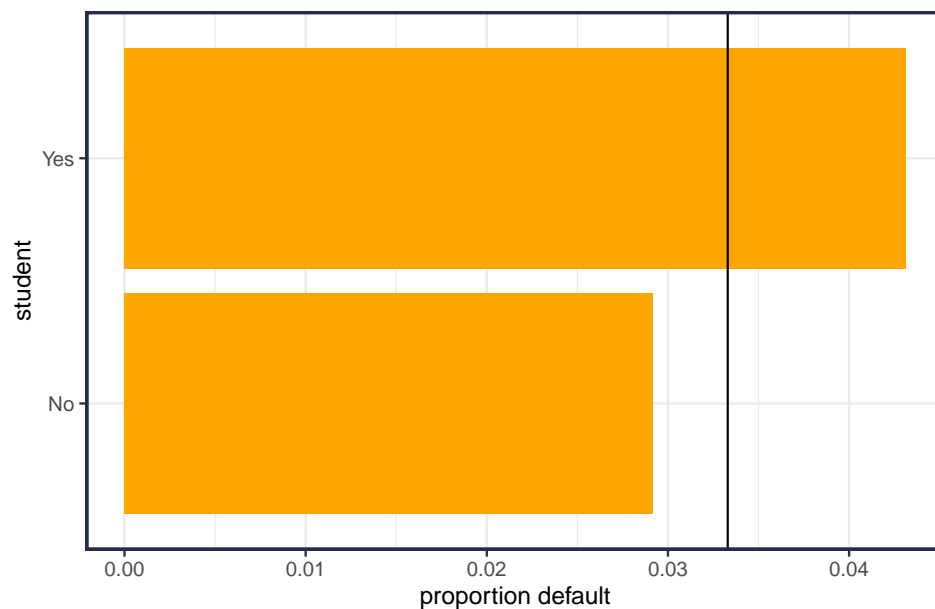
ggplot(Default, aes(default, income, fill=default)) +
  geom_violin(draw_quantiles=.5) +
  scale_fill_manual(values=plot_cols, guide=FALSE)

count(Default, default, student) %>%
  group_by(student) %>% mutate(p=n/sum(n)) %>%
  filter(default == "Yes") %>%
  ggplot(aes(student, p, fill=default)) +
  geom_col() +
  geom_hline(yintercept=mean(Default$default == "Yes")) +
  scale_fill_manual(values=plot_cols, guide="none") +
  labs(title="Proportion of Defaults by Student status",
       y="proportion default") +
  coord_flip()
```





Proportion of Defaults by Student status



## Linear Regression (for binary response)

```
library(broom) # to extract good stuff from models

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

# Extract coefficients
coef(fit.lm) # generic coef function to get coefficients
#> (Intercept) studentYes balance income
#> -8.118e-02 -1.033e-02 1.327e-04 1.992e-07
broom::tidy(fit.lm) # tidy way to get coefficients
#> # A tibble: 4 x 5
```

```
#>   term          estimate std.error statistic    p.value
#>   <chr>          <dbl>    <dbl>    <dbl>    <dbl>
#> 1 (Intercept) -0.0812    0.00838    -9.68 4.37e- 22
#> 2 studentYes  -0.0103    0.00566    -1.82 6.82e-  2
#> 3 balance      0.000133   0.00000355   37.4 8.04e-287
#> 4 income       0.000000199 0.000000192    1.04 2.99e-  1
```

## k nearest neighbor

```
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 ~ balance + income, data = Default) %>% # specify formula
  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(
  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
# 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

```
# 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")

# Get coefficients
tidy(fit.lr)
#> # A tibble: 4 x 5
#>   term          estimate std.error statistic    p.value
#>   <chr>          <dbl>    <dbl>    <dbl>    <dbl>
#> 1 (Intercept) -10.9      0.492    -22.1 4.91e-108
```



```
#> 2 studentYes -0.647 0.236 -2.74 6.19e- 3
#> 3 balance 0.00574 0.000232 24.7 4.22e-135
#> 4 income 0.00000303 0.00000820 0.370 7.12e- 1

# Get predictions (for training data)
prob.lr = predict(fit.lr, type="response") # probabilities
link.lr = predict(fit.lr, type="link") # logit (linear part)

# Interpret
# Notice that Student=Yes has a negative coefficient, but the plot of
# defaults by student status suggests otherwise.
# Reason is because students have more balance on average than non-students,
# and they get over-estimated once balance is in the model

plot_cols = c(Yes="orange", No="blue") # set colors

ggplot(Default, aes(balance, fill=student)) +
  geom_density(alpha=.75) +
  facet_wrap(~default, labeller=label_both) +
  scale_fill_manual(values=plot_cols)

ggplot(Default, aes(student, balance, fill=student)) +
  geom_violin(draw_quantiles = .5) +
  facet_wrap(~default, labeller=label_both) +
  scale_fill_manual(values=plot_cols)

# probability at certain values
eval.pts = tibble(
  student = c("Yes", "No"),
  balance = c(1000, 1000), # balance = 1000
  income = c(40000, 40000) # income set to 40K
)

predict(fit.lr, eval.pts, type="link")
#> 1 2
#> -5.658 -5.011
predict(fit.lr, eval.pts, type="response")
#> 1 2
#> 0.003477 0.006619

# Simpson's Paradox

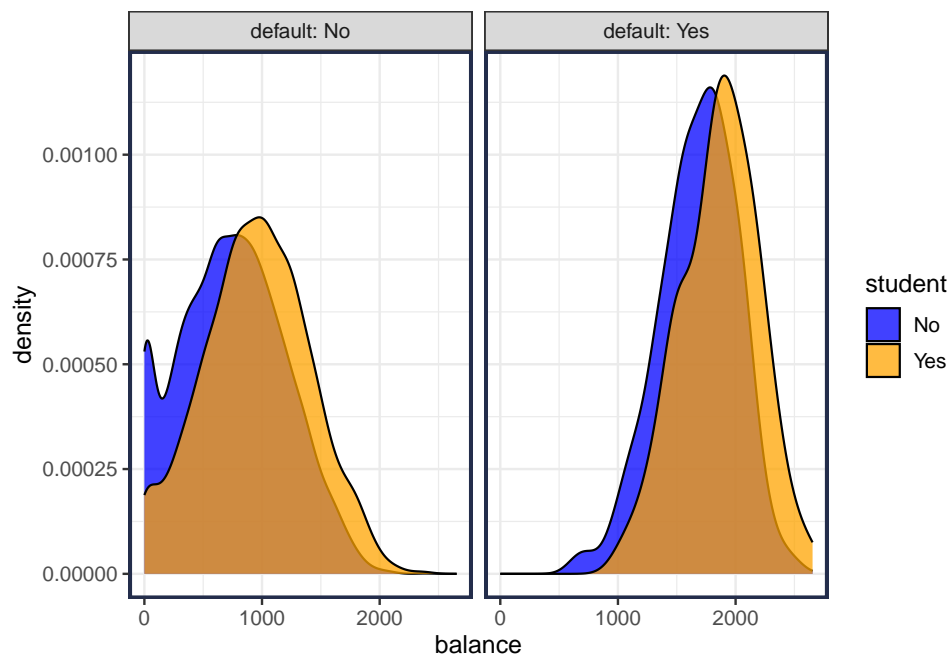
# Students have higher default rate
Default %>%
  group_by(student) %>% summarize(n=n(), p_default = mean(default == 'Yes'))
#> # A tibble: 2 x 3
#> student n p_default
#> <fct> <int> <dbl>
#> 1 No 7056 0.0292
#> 2 Yes 2944 0.0431

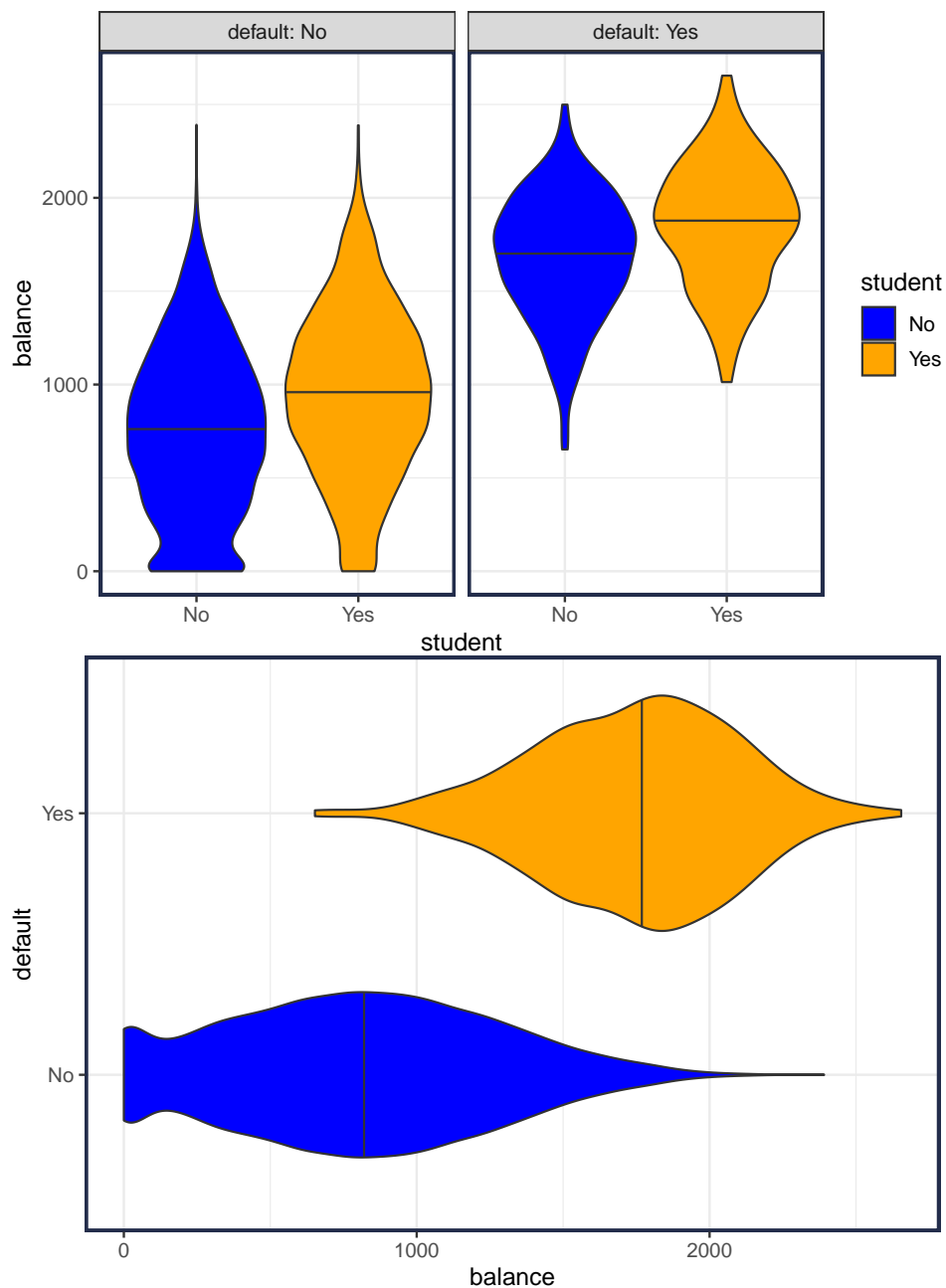
# People with higher balances have higher default rate
ggplot(Default, aes(balance, default, fill=default)) +
  geom_violin(draw_quantiles=.5) + #alternative: geom_boxplot() +
  scale_fill_manual(values=plot_cols, guide=FALSE)
```

```
Default %>%
  group_by(default) %>% summarize(avg_balance = mean(balance))
#> # A tibble: 2 x 2
#>   default avg_balance
#>   <fct>     <dbl>
#> 1 No       804.
#> 2 Yes     1748.

# Students have higher balances on average, so they appear to be more likely
# to default if the balance is not taken into account
Default %>%
  group_by(student) %>% summarize(avg_balance = mean(balance))
#> # A tibble: 2 x 2
#>   student avg_balance
#>   <fct>     <dbl>
#> 1 No       772.
#> 2 Yes     988.

# This is why the logistic regression model correctly adjusts the student status
# negative.
Default %>%
  mutate(p_hat = prob.lr) %>%
  group_by(student) %>%
  summarize(n=n(), default_rate = mean(default == 'Yes'), avg_p = mean(p_hat))
#> # A tibble: 2 x 4
#>   student      n default_rate avg_p
#>   <fct>   <int>     <dbl> <dbl>
#> 1 No     7056      0.0292 0.0292
#> 2 Yes   2944      0.0431 0.0431
```





### 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:

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

```

#-----#
#: 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 tidymodels()
#-----#
# Note: tidymodels is a collection of useful modeling packages (like tidyverse)
library(tidymodels)
# This code only uses the recipe() package
# 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()
#-----#
obj = hardhat::mold(fmla, data=Default)
X.train = obj$predictors
Y.train = obj$outcomes
# X.test = hardhat::mold(fmla, data=test)$predictors

#-----#
#: 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

```

library(tidymodels)
rec = recipe(y~student + balance + income, 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.eval = bake(rec, new_data = eval.pts, all_predictors(), 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

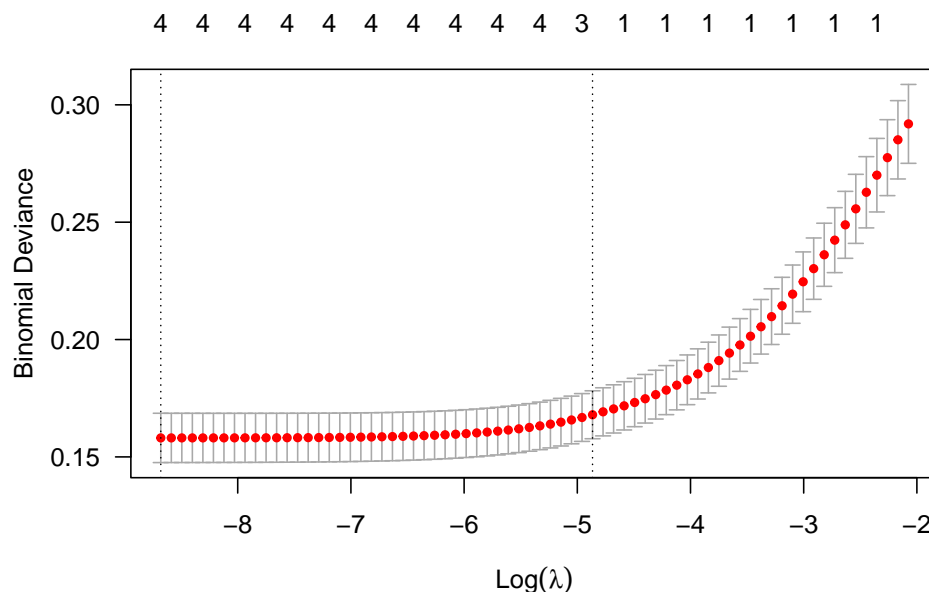
# : Elastic net with alpha = .5. Use CV to select lambda.
library(glmnet)
set.seed(2020)
fit.enet = cv.glmnet(X.train, Y.train,
                    alpha=.5,
                    family="binomial")

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

# : probability at certain values
predict(fit.enet, X.eval, s="lambda.min", type="response")
#>      lambda.min
#> [1,] 0.003722
#> [2,] 0.006929
predict(fit.enet, X.eval, s="lambda.1se", type="response")
#>      lambda.1se
#> [1,] 0.01444
#> [2,] 0.01569

# : Compare with intercept only model. Set large penalty (s large)
predict(fit.enet, X.eval, s=1000, type="response") # intercept only (effectively)
#>      s1
#> [1,] 0.0333
#> [2,] 0.0333
mean(Default$y) # actual intercept only
#> [1] 0.0333

# : Compare with unpenalized logistic regression. Set penalty s=0.
predict(fit.enet, X.eval, s=0, type="response") # unpenalized (effectively)
#>      s1
#> [1,] 0.003722
#> [2,] 0.006929
predict(fit.lr, eval.pts, type="response")
#>      1      2
#> 0.003477 0.006619
```



## 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, ])

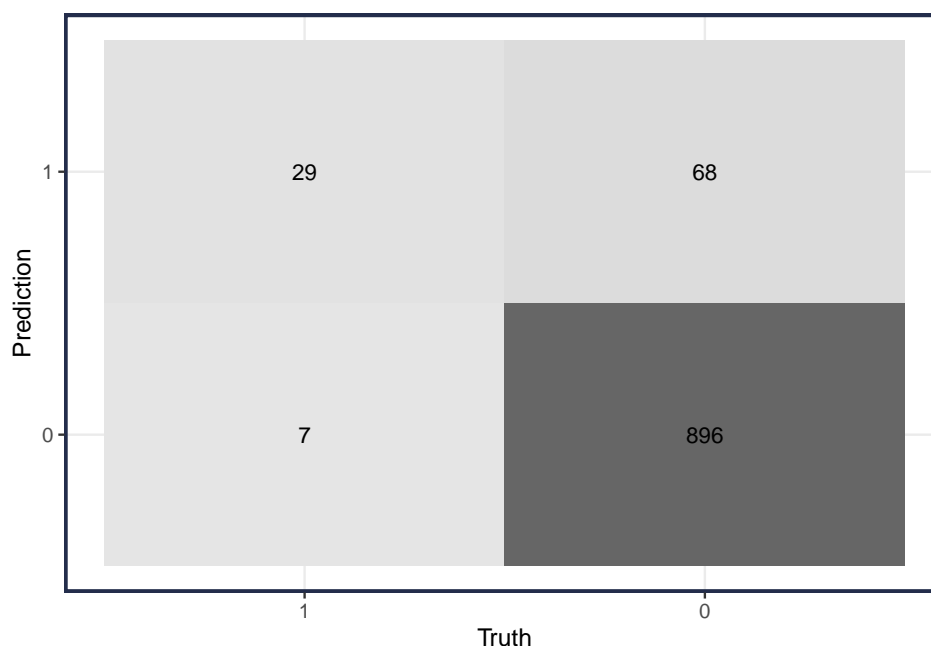
#: Get predictions (of p(x) and gamma(x)) on test data
p.hat = predict(fit.lm, Default[test, ], type='response')
gamma = predict(fit.lm, Default[test, ], type='link')

#: 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(., levels=c("1", "0"))) %>% # conf_mat() requires factors
  conf_mat(truth = G.test, estimate = G.hat)
cm
#>      Truth
#> Prediction    1    0
#>      1   29  68
    
```

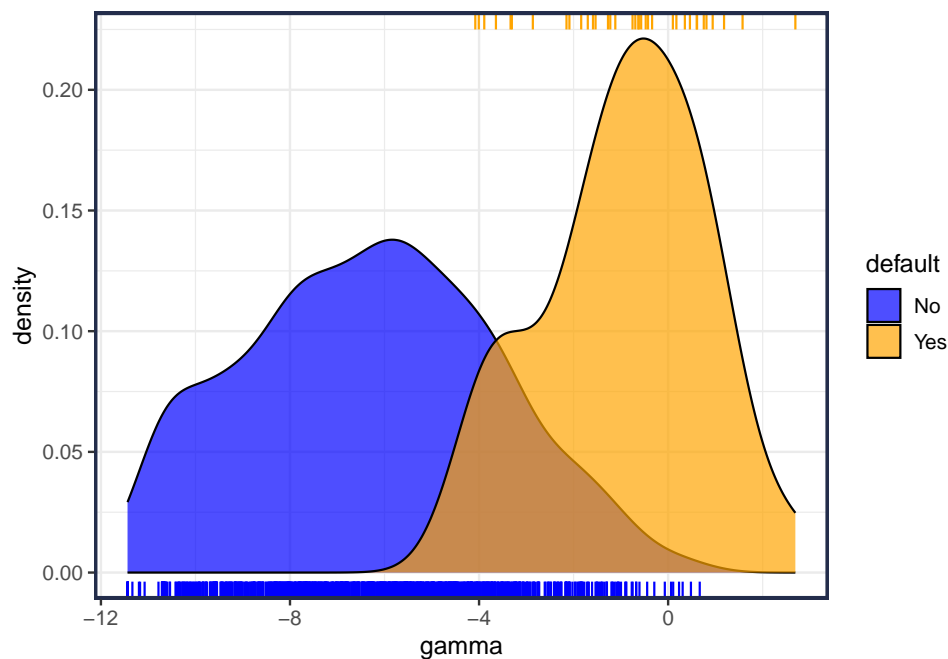
```
#>      0      7 896
autoplot(cm, type = "heatmap")
```



```
#: 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")
```

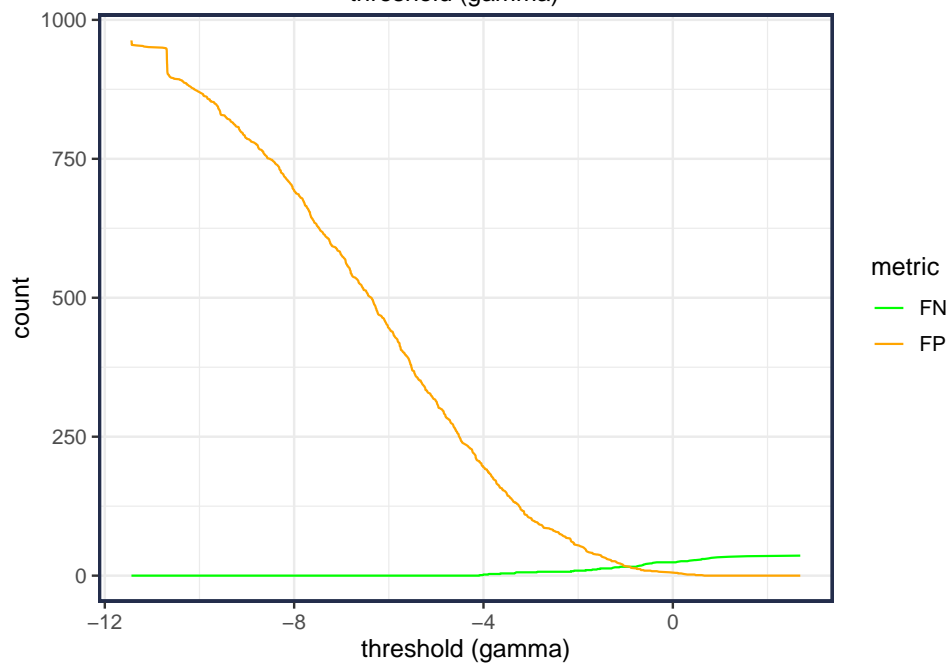
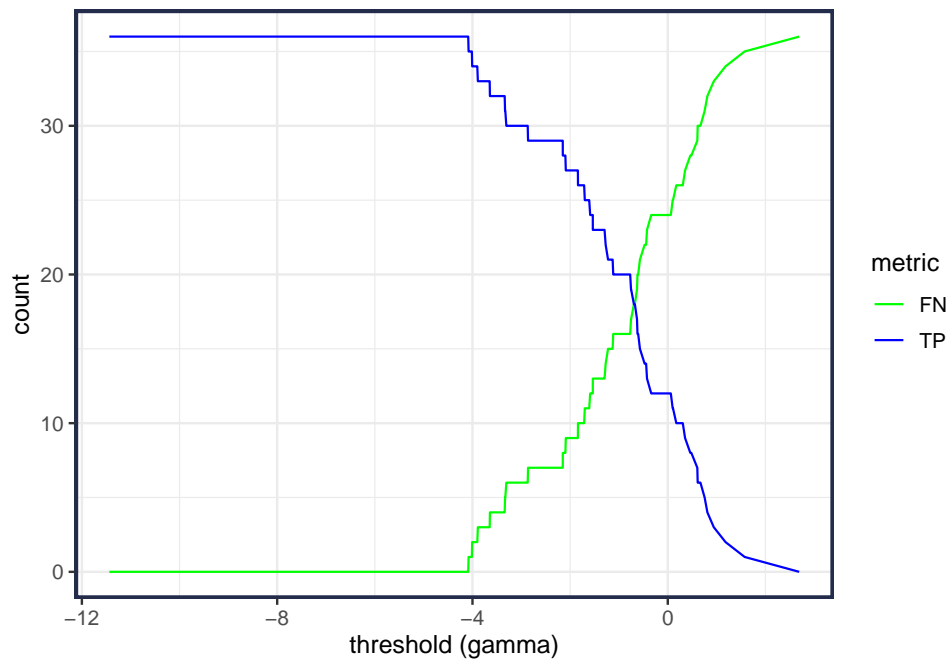
```
#: Make performance curves
perf %>% select(threshold=gamma, FN, TP) %>%
  gather(metric, n, -threshold) %>%
  ggplot(aes(threshold, n, color=metric)) + geom_line() +
  labs(x= "threshold (gamma)", y="count") +
  scale_color_manual(values=col_lines)
```

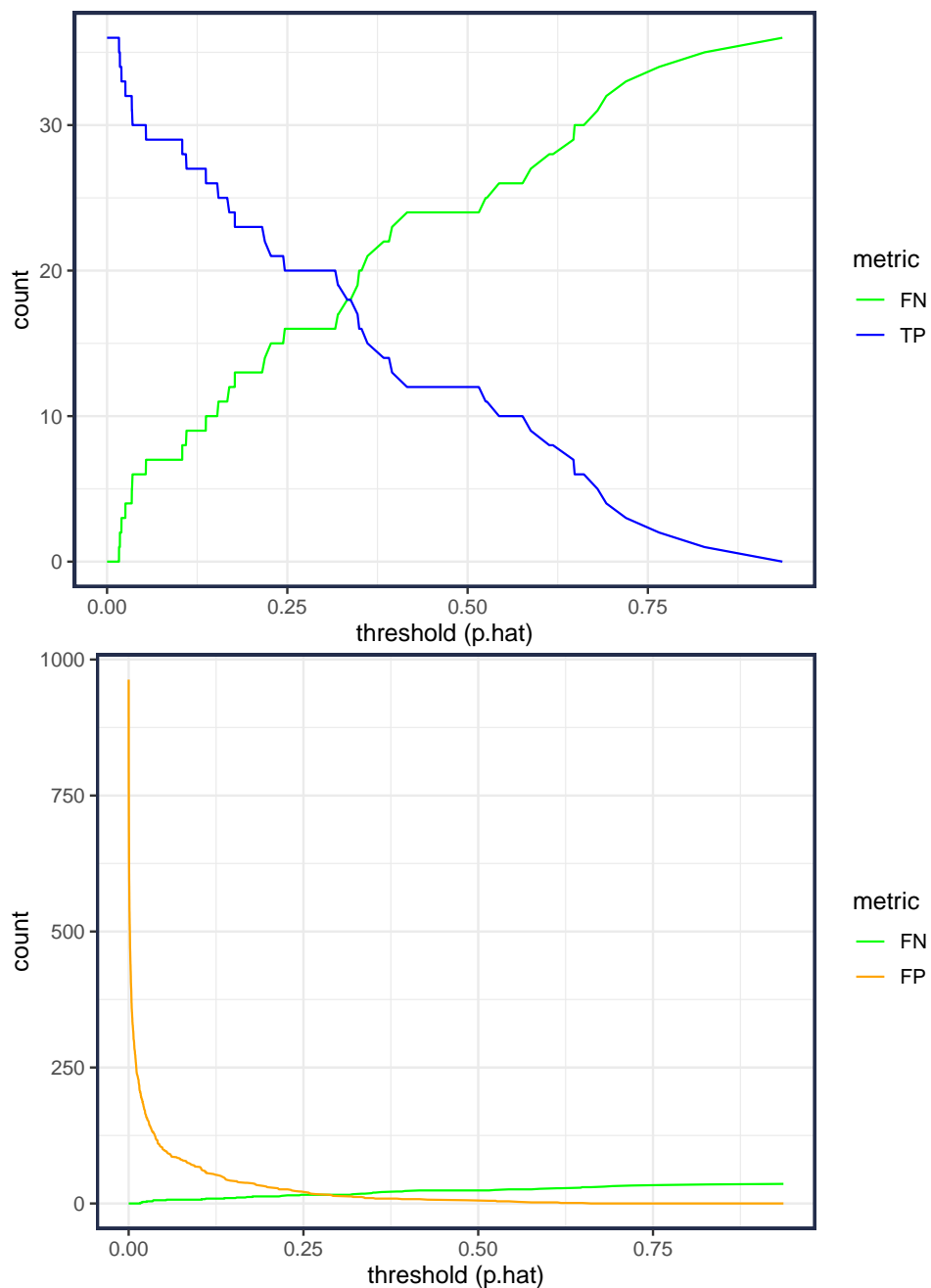
```
perf %>% select(threshold=gamma, FN, FP) %>%
  gather(metric, n, -threshold) %>%
  ggplot(aes(threshold, n, color=metric)) + geom_line() +
  labs(x= "threshold (gamma)", y="count") +
  scale_color_manual(values=col_lines)
```

```
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)
```





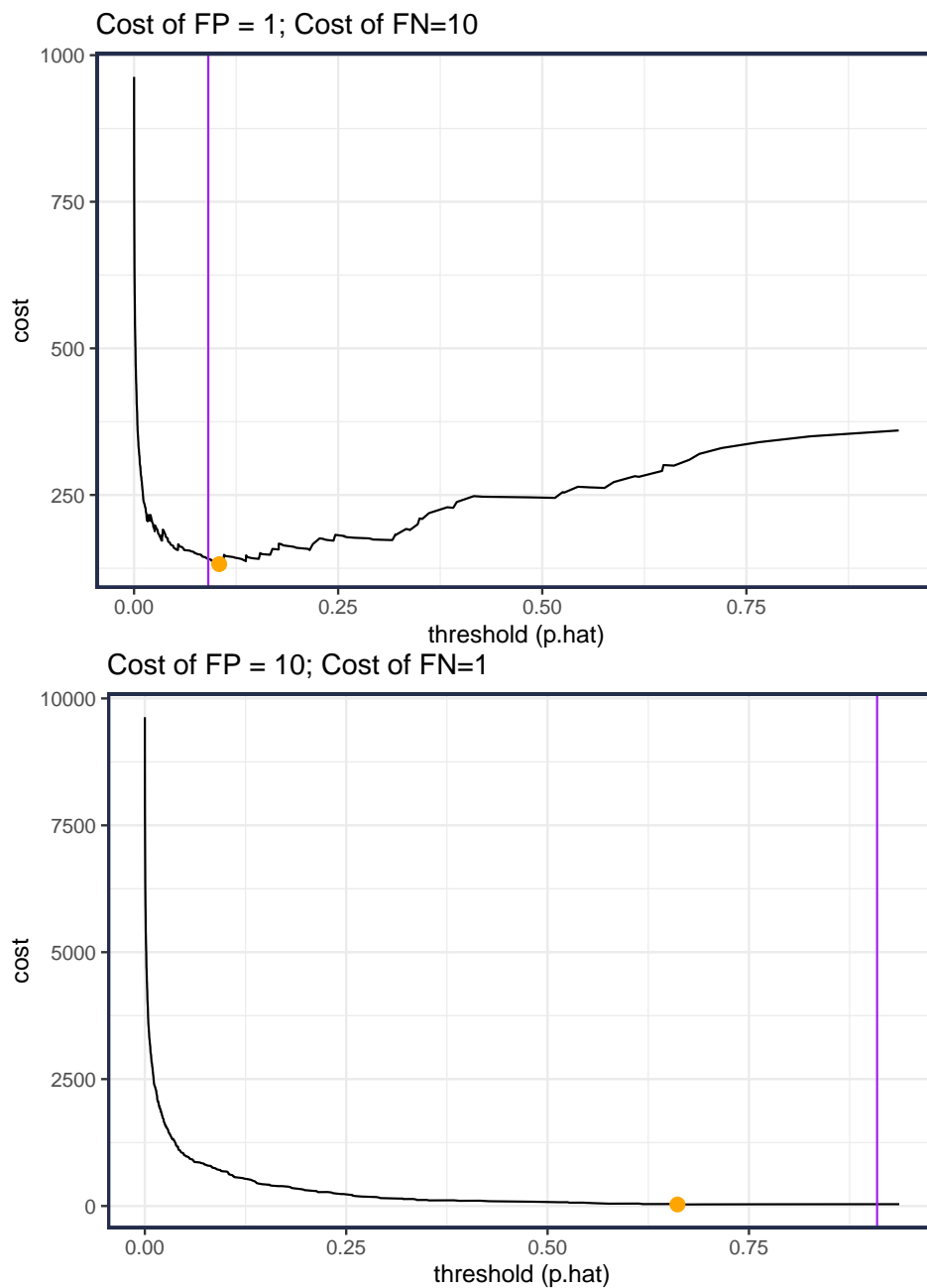


```

#: 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)")

```



```
#: 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 gamm
# - for

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

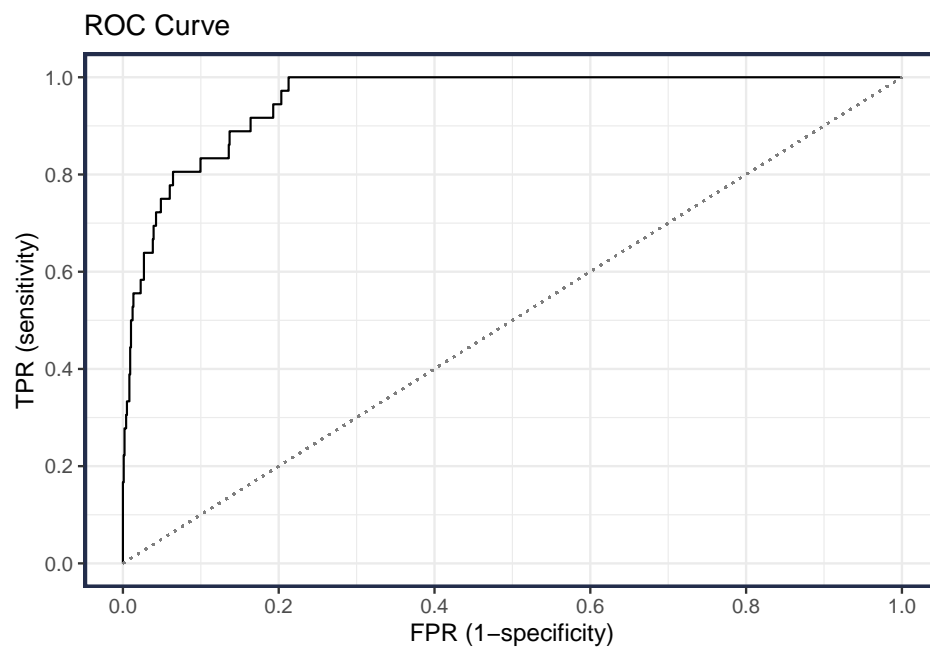
autoplot(ROC) # autoplot() method

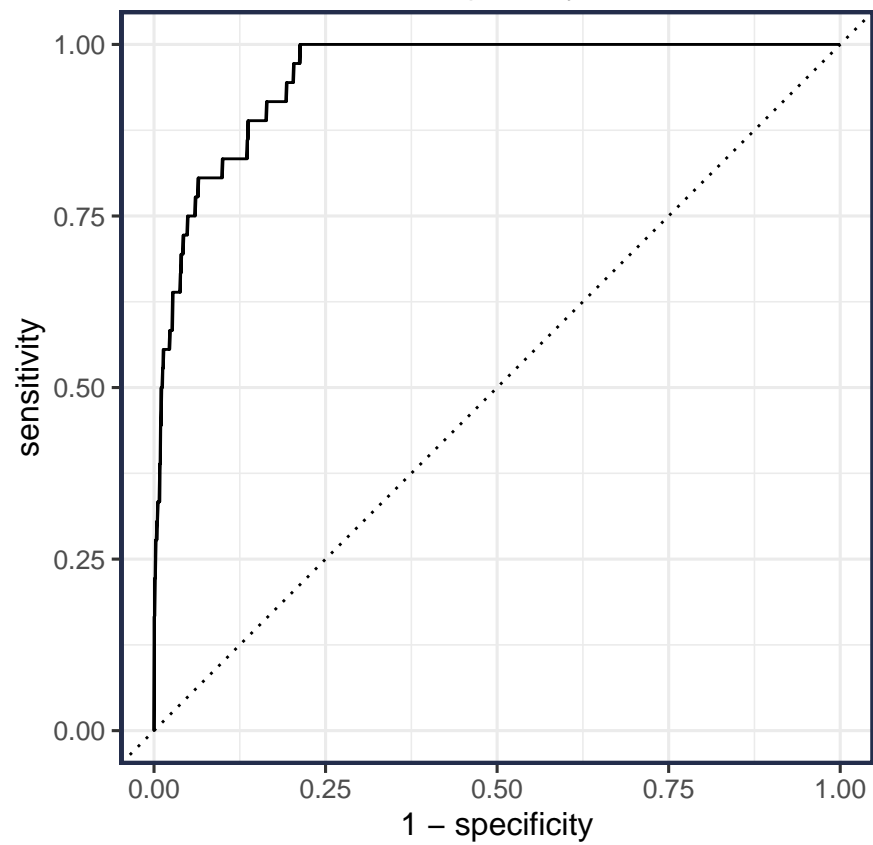
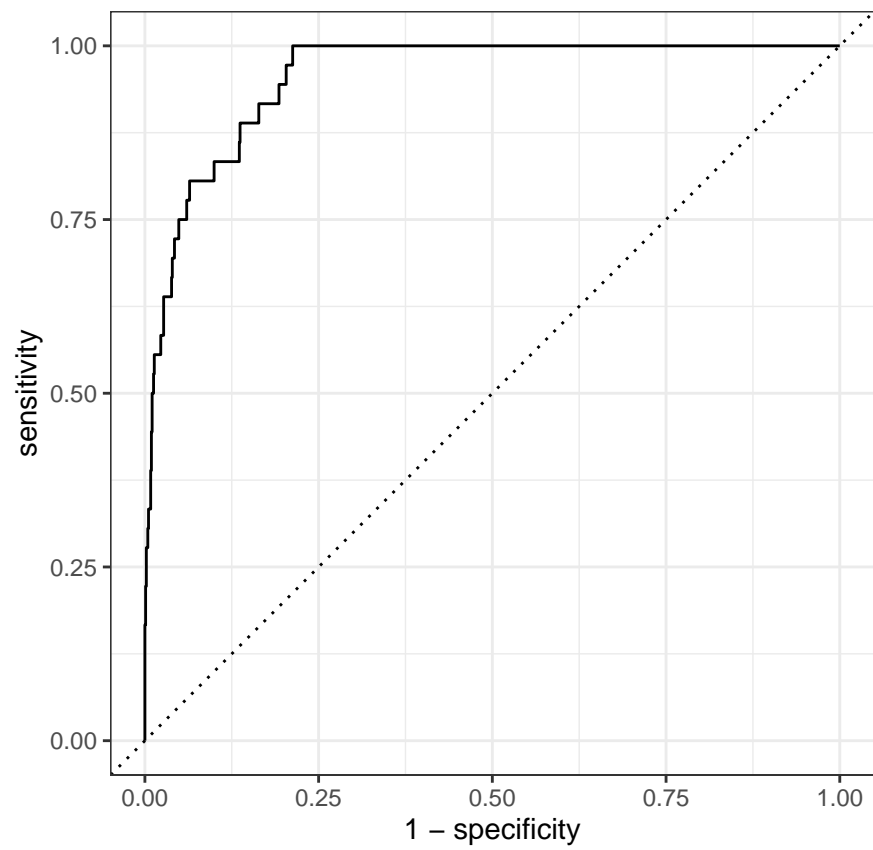
ROC %>%
  # same as autoplot()
  ggplot(aes(1-specificity, sensitivity)) + geom_line() +
  geom_abline(lty=3) +
  coord_equal()

# : 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

```





```

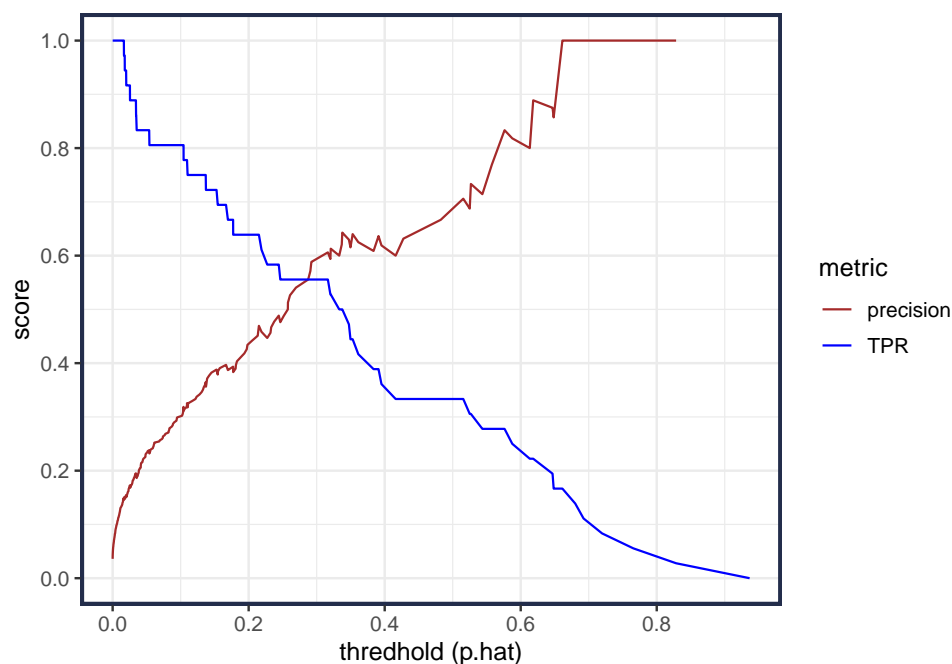
# Log Loss Metric
yardstick::mn_log_loss_vec(factor(G.test, 1:0), p.hat)
#> [1] 0.08113

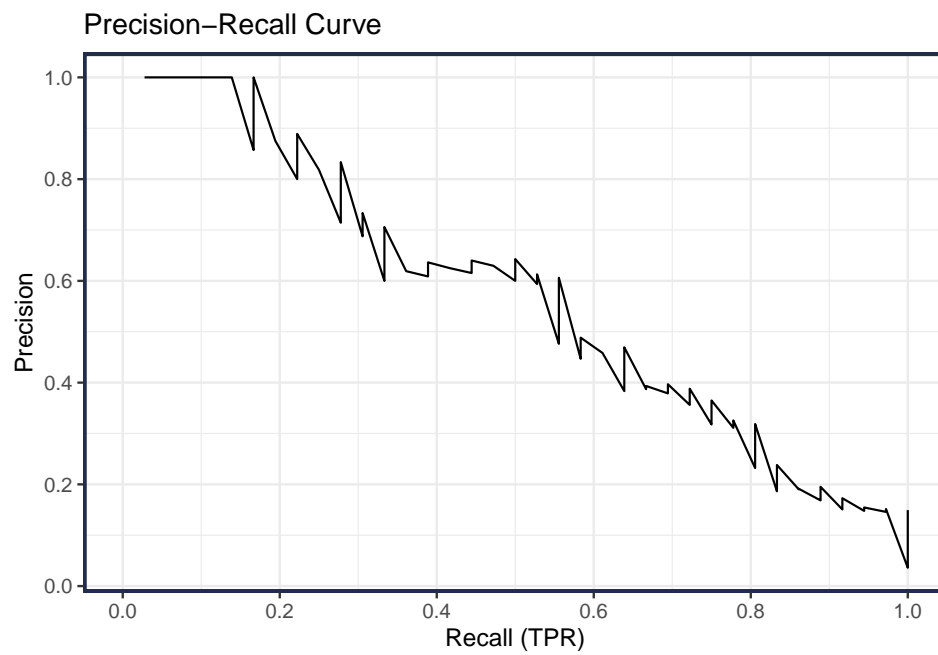
tibble(truth = factor(G.test, levels=c(1,0)), p.hat) %>%
  yardstick::mn_log_loss(truth, p.hat)
#> # A tibble: 1 x 3
#>   .metric      .estimator .estimate
#>   <chr>        <chr>      <dbl>
#> 1 mn_log_loss binary      0.0811

# 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="threshold (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()').

```





```
knitr::knit_exit()
```