Classification

Logistic Regression, Discriminant Analysis, and Naive Bayes

SYS 6018 | Spring 2021

classification.pdf

Contents

Some of the figures in this presentation are taken from "An Introduction to Statistical Learning, with applications in R" (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.

1 Classification 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.

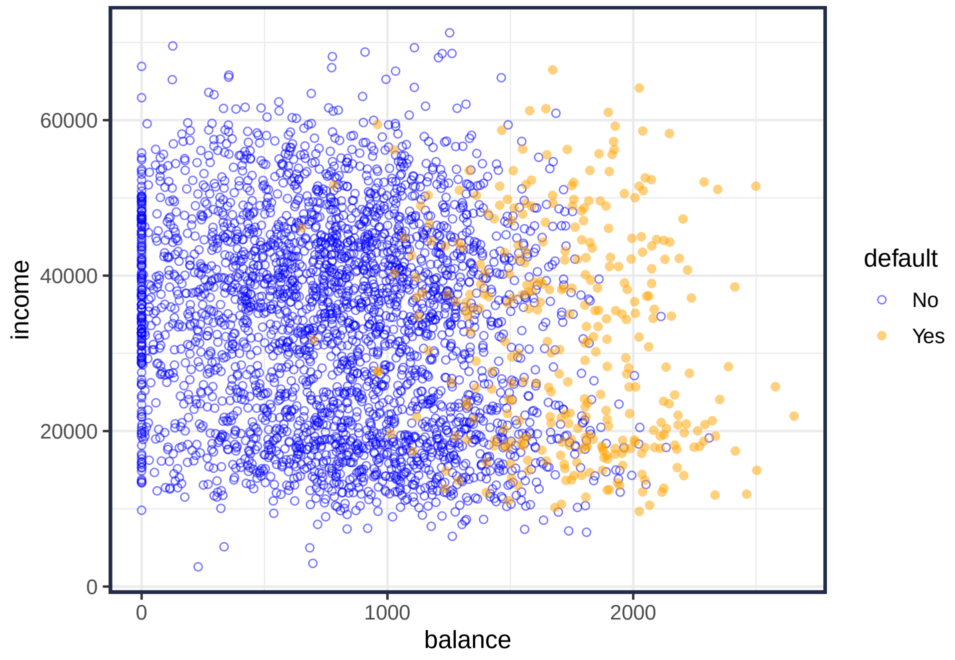
```
data(Default, package="ISLR")
```

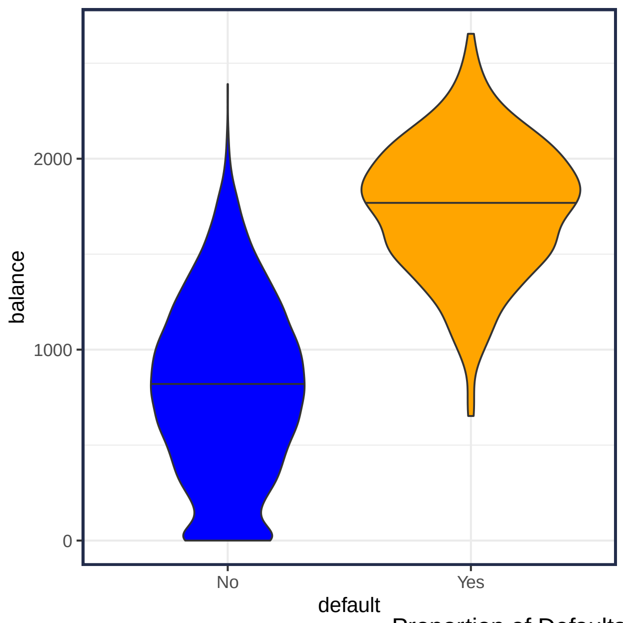
The variables are:

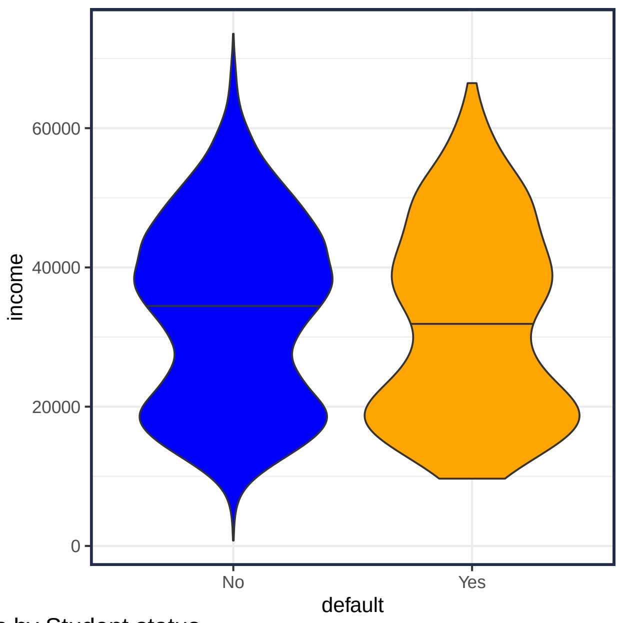
- *response variable* is categorical (factor) `Yes` and `No`, (`default`)
- the categorical (factor) variable (`student`) is either `Yes` or `No`
- the average balance a customer has after making their monthly payment (`balance`)
- the customer's income (`income`)

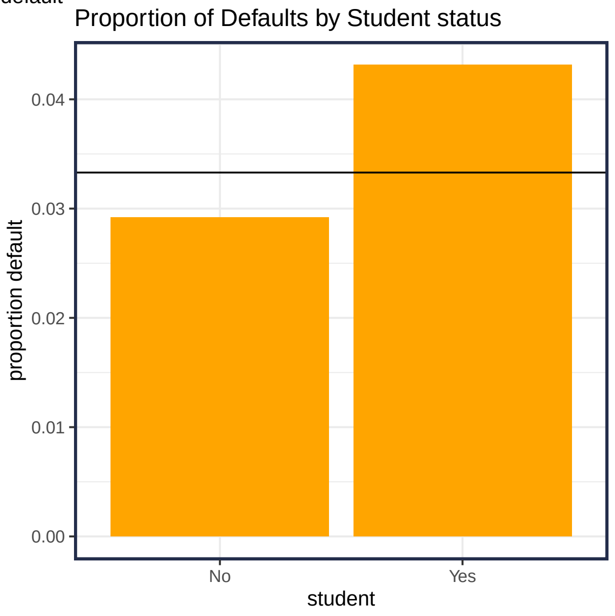
default	student	balance	income
No	No	729.5	44362
No	Yes	817.2	12106
No	No	1073.5	31767
No	No	529.3	35704
No	No	785.7	38463
No	Yes	919.6	7492

```
summary(Default)
#>  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  
```




balance
0
1000
2000
No
Yes
default


income
0
20000
40000
60000
No
Yes
default


Proportion of Defaults by Student status
proportion default
0.00
0.01
0.02
0.03
0.04
No
Yes
student

Your Turn #1 : Credit Card Default Modeling

How would you construct a model to predict defaults?

2 Classification and Pattern Recognition

- The response 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 $\Pr(G = g \mid \mathbf{X} = \mathbf{x})$

2.1 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 *code* the response 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.

2.1.1 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 = ifelse(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)	-0.081	0.008	-9.685	0.000
studentYes	-0.010	0.006	-1.824	0.068
balance	0.000	0.000	37.412	0.000
income	0.000	0.000	1.039	0.299

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 is make sense to use linear regression for binary classification?

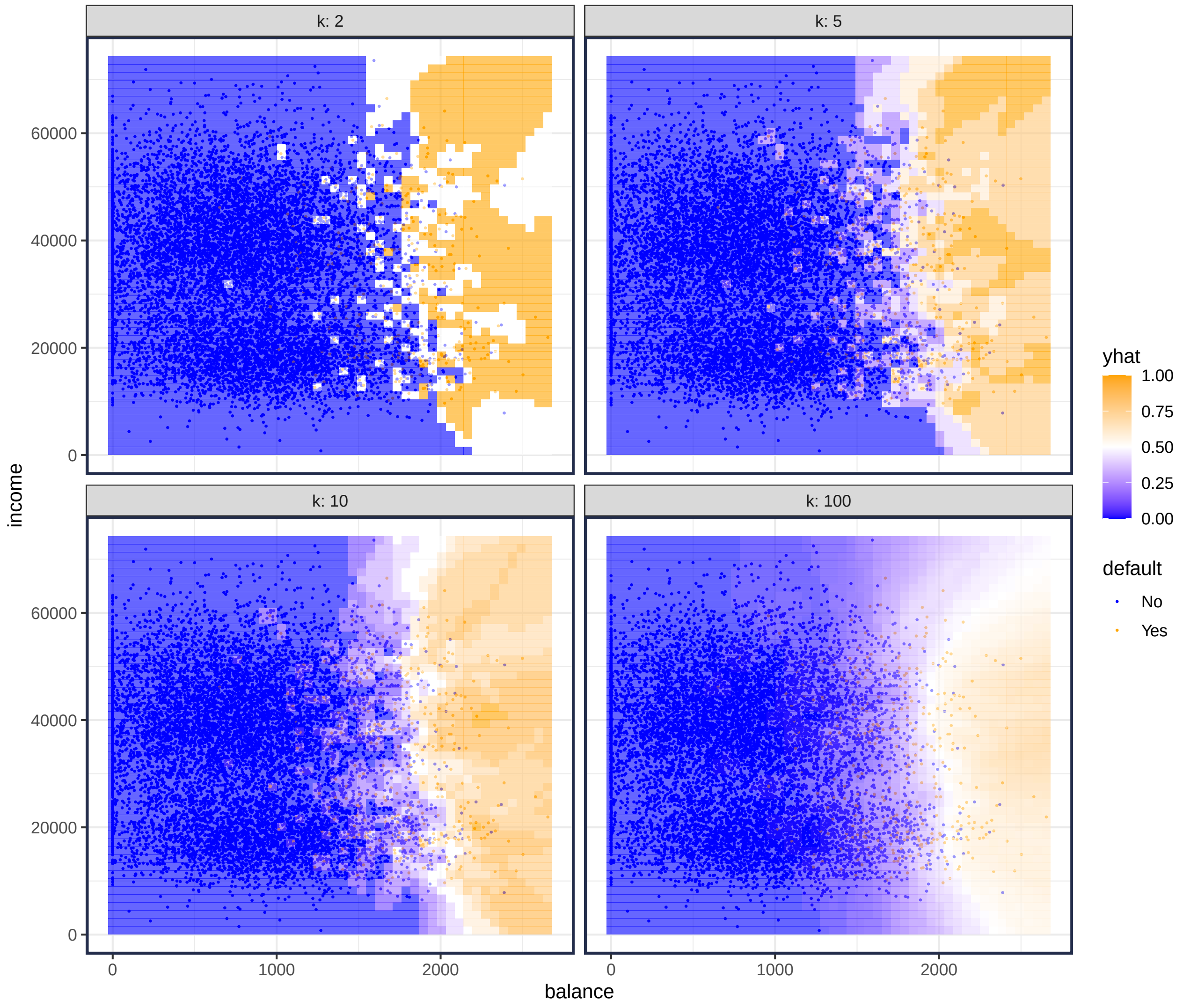
2.1.2 k-nearest neighbor (kNN)

- 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 response variables is categorical $G_i \in \mathcal{G}$

$$\begin{aligned} f_g^{\text{knn}}(x;k) &= \frac{1}{k} \sum_{i:x_i \in N_k(x)} \mathbb{1}(g_i = g) \\ &= \widehat{\Pr}(G_i = g \mid x_i \in N_k(x)) \end{aligned}$$

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

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

3 Logistic Regression

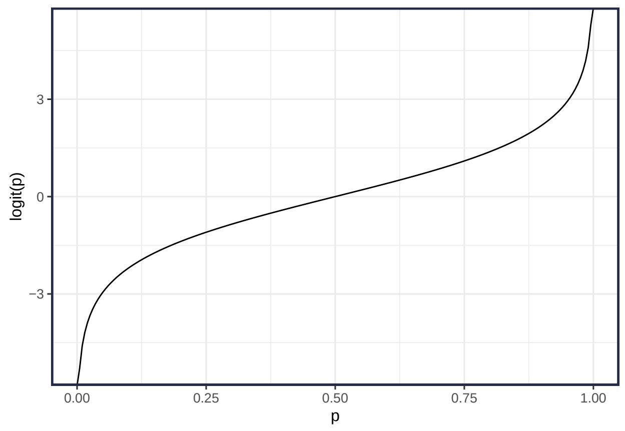
3.1 Basics

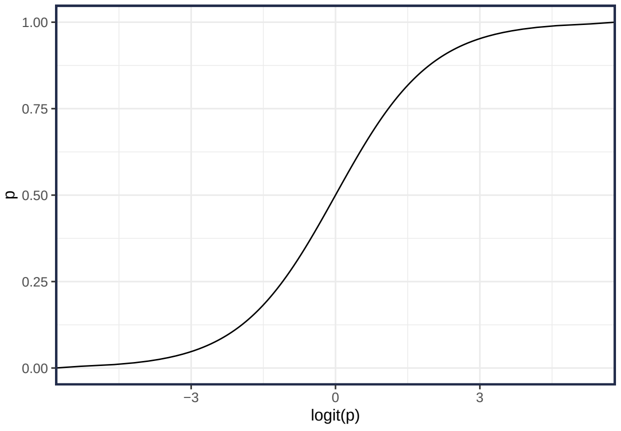
- Let $0 \leq p \leq 1$ be a probability.
- The log-odds of p is called the *logit*

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

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

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





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

$$\mathrm{E}[Y \mid X = x] = \Pr(Y=1 \mid X=x) = \beta^\mathsf{T} x$$

- Logistic Regression models alternatively estimate

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

and thus,

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

3.2 Estimation

- The data for logistic regression is: $(\mathbf{x}_i, y_i)_{i=1}^n$ where $y_i \in \{0,1\}$, $\mathbf{x}_i = (x_{i0}, x_{i1}, \ldots, x_{ip})^\mathsf{T}$.
- $y_i \mid \mathbf{x}_i \sim \operatorname{Bern}(p_i(\beta))$
  - $p_i(\beta) = \Pr(Y=1 \mid \mathbf{X} = \mathbf{x}_i; \beta) = \left(1+e^{-\beta^\mathsf{T}\mathbf{x}_i}\right)^{-1}$

  - $\beta^\mathsf{T}\mathbf{x}_i = \mathbf{x}_i^\mathsf{T}\beta = \beta_0 + \sum_{j=1}^p x_{ij}\beta_j$

- Bernoulli Likelihood Function

$$L(\beta) = \prod_{i=1}^n p_i(\beta)^{y_1}(1-p_i(\beta))^{1-y_i}$$

$$\log L(\beta) = \sum_{i=1}^n \{y_i \ln p_i(\beta) + (1-y_i)\ln(1-p_i(\beta))\}$$

- 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*, where the loss function is the negative log-likelihood

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

  where the loss is

$$\ell(\beta) = -C\sum_{i=1}^n \{y_i \ln p_i(\beta) + (1-y_i)\ln(1-p_i(\beta))\}$$

  - and C is some constant, e.g., $C = 1/n$

- This view facilitates *penalized logistic regression*

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

  - Ridge Penalty

$$P(\beta) = \sum_{j=1}^p |\beta_j|^2 = \beta^\mathsf{T}\beta$$

  - Lasso Penalty

$$P(\beta) = \sum_{j=1}^p |\beta_j|$$

  - Best Subsets

$$P(\beta) = \sum_{j=1}^p |\beta_j|^0 = \sum_{j=1}^p 1_{(\beta_j \neq 0)}$$

  - Elastic Net (glmnet form)

$$P(\beta, \alpha) = \sum_{j=1}^p (1-\alpha)|\beta_j|^2/2 + \alpha|\beta_j|$$

3.3 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

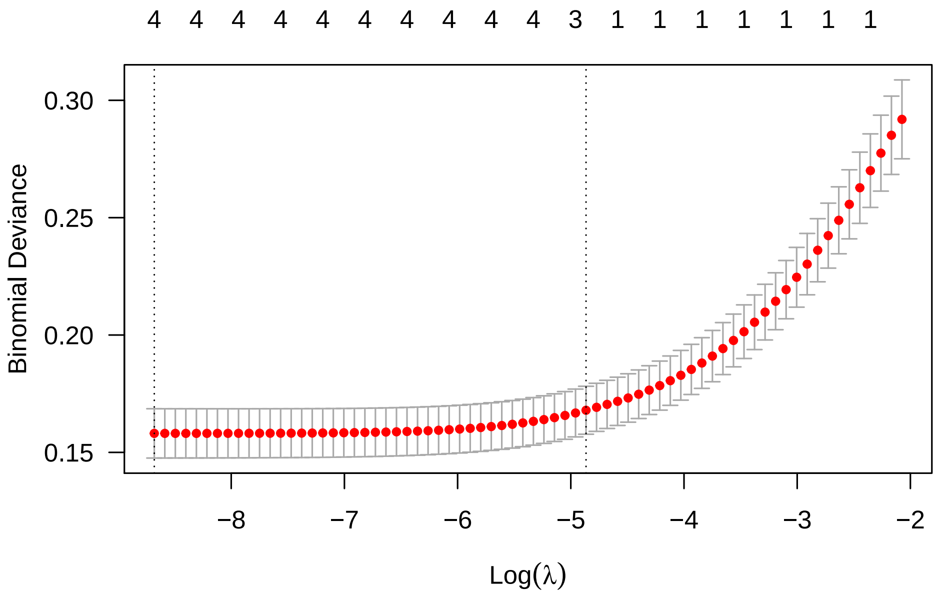
1. What is the estimated probability of default for a Student with a balance of \$1000?
2. What is the estimated probability of default for a *Non-Student* with a balance of \$1000?
3. Why does `student=Yes` appear to lower risk of default, when the plot of student status vs. default appears to increase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

3.3.1 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)
library(glmnetUtils)
set.seed(2020)
```

```r
fit.enet = cv.glmnet(y~student + balance + income, data=Default,
                     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

3.4 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

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

4 Evaluating Classification Models

- Training Data: $\{X_i, G_i\}$
  - $G_i \in \{1, \ldots, K\}$ (i.e., there are K classes)
- Predictor: $\hat{G}(X)$
- Loss function: $L(G, \hat{G}(X))$ is the loss incurred by estimating G with $\hat{G}$
- Risk is the expected loss (or expected prediction error EPE)
  - Expectation is taken wrt future values of (X, G)

$$\begin{aligned}
\text{Risk}(\hat{G}) &= \text{EPE} \\
&= \text{E}_{XG}\left[L(G, \hat{G}(X))\right] \\
&= \text{E}_{X}\left[\text{E}_{G|X}\left[L(G, \hat{G}(X)) \mid X\right]\right] \\
&= \text{E}_{X}\left[R_X(\hat{G})\right]
\end{aligned}$$

- The Risk at input $X = x$ is

$$\begin{aligned}
R_x(\hat{G}) &= \text{E}_{G|X=x}\left[L(G, \hat{G}(x)) \mid X = x\right] \\
&= \sum_{k=1}^{K} L(G = k, \hat{G}(x)) \Pr(G = k \mid X = x) \\
&= \sum_{k=1}^{K} L(G = k, \hat{G}(x)) p_k(x)
\end{aligned}$$

- Thus the optimal class label, given $X = x$, is

$$\hat{G}(x) = \arg\min_{g} R_x(g)$$

- The optimal label can only be obtained if you know $p_k(x) = \Pr(G = k \mid X = x)$ for all k.
- Since we won't know $\{p_k(x)\}_k$, we have to estimate them.

4.1 Evaluation of Binary Classification Models

- We are considering *binary* outcomes, so use the notation $Y \in \{0, 1\}$
- Let $p(x) = \Pr(Y = 1 \mid X = x)$
- The Risk (for a binary outcome) is:

$$\begin{aligned}
R_x(g) &= L(1, g) \Pr(Y = 1 \mid X = x) + L(0, g)(1 - \Pr(Y = 1 \mid X = x)) \\
&= L(1, g)p(x) + L(0, g)(1 - p(x))
\end{aligned}$$

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

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

4.1.1 Example

- 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
  - $L(G = 0, \hat{G} = 1) = FP$: There is a cost of FP units if the model issues a *false positive*, estimating the patient has cancer when they don't
  - $L(G = 1, \hat{G} = 0) = FN$: There is a cost of FN units if the model issues a *false negative*, estimating the patient does not have cancer when they really do
  - In these scenarios FN is often much larer than FP ($FN >> FP$) 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
- Our model will decide to issue a positive indication for cancer if $R_x(1) < R_x(0)$ which occurs when

$$\begin{aligned}
\frac{p(x)}{1-p(x)} &\geq \frac{FP}{FN} \\
p(x) &\geq \frac{FP}{FP + FN}
\end{aligned}$$

- The ratio of FP to FN is all that matters for the decision. Let's say that FP=1 and 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 with cancer), which is usually going to be small.

4.2 Common Binary Loss Functions

- Suppose we are going to estimate a binary reponse $Y \in \{0,1\}$ with a (possibly continuous) predictor $\hat{y}(x)$
- **0-1 Loss** or **Misclassification Error**

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

  - This assumes $L(0,1) = L(1,0)$ (i.e., false positive costs the same as a false negative)

  - Requires that a *hard classification* is made
  - The optimal prediction is $y^*(x) = \mathbb{1}(p(x) > 1 - p(x))$ which is equivalent to $\mathbb{1}(p(x) > 0.50)$

- **Squared Error**

$$L(y, \hat{y}(x)) = (y - \hat{y}(x))^2$$

  - The optimal prediction is $y^*(x) = \mathrm{E}[Y \mid X = x] = \Pr(Y = 1 \mid X = x)$

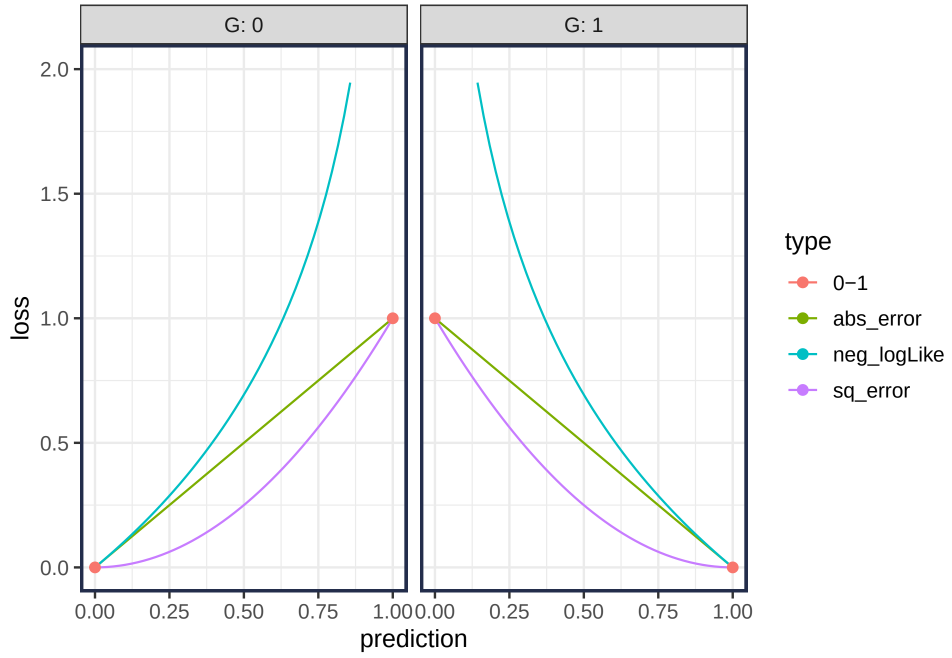
- **Absolute Error**

$$L(y, \hat{y}(x)) = |y - \hat{y}(x)|$$

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

$$\begin{aligned} L(y, \hat{y}(x)) &= -\left\{y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)\right\} \\ &= \begin{cases} -\log \hat{y} & y = 1 \\ -\log(1 - \hat{y}) & y = 0 \end{cases} \end{aligned}$$

  - Requires $\hat{y}(x) \in [0, 1]$



4.3 Evaluating Binary Classification Models

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

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

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

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

- Then we get

$$\begin{aligned} p(x) &= \Pr(G = 1 \mid X = x) \\ &= \frac{e^{\gamma(x)}}{1 + e^{\gamma(x)}} \end{aligned}$$

- And the optimal (*hard classification*) decision can be described in terms of $\gamma(x)$:

$$\boxed{\text{Choose } \hat{G}(x) = 1 \text{ if } \gamma(x) > t, \quad \text{where } t \text{ is a threshold}}$$

- If the losses/costs are known, then

$$t^* = \log\left(\frac{L(0,1) - L(0,0)}{L(1,0) - L(1,1)}\right)$$

  is the optimal threshold.

4.3.1 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 is $\hat{G}_t(x) = \mathbb{1}(\hat{\gamma}(x) \geq t)$
- Note: we can also use $\hat{p}(x)$ to make the decision threshold.
  - Just adjust the threshold accordingly.
- Note: we may also need to estimate the best threshold, t^* (more info on this below).

4.4 Performance Metrics

4.4.1 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_2	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 about 9000 training, 1000 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 $\hat{\gamma}(x) \geq \log(.10) - \log(1 - .10) = -2.1972$

```r
#-- 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)) on test data
p.hat = predict(fit.lm, newdata=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    896    7  903
#>       1     68   29   97
#>       Sum  964   36 1000
```

4.4.2 Metrics

Metric	Definition	Estimate
Risk/Exp.Cost	$\frac{1}{N}\sum_{i=0}^{1}\sum_{j=0}^{1} L(i,j) P_X(G(X)=i, \hat{G}_t(X)=j)$	$\frac{1}{N}\sum_{i=1}^{N} L(G_i, \hat{G}_t(x_i))$
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)	$P_X(G(X)=1 \mid \hat{G}_t(X)=1)$	$\frac{1}{\hat{N}_1(t)}\sum_{i:\hat{G}(x_i)=1} \mathbb{1}(G_i=1)$

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

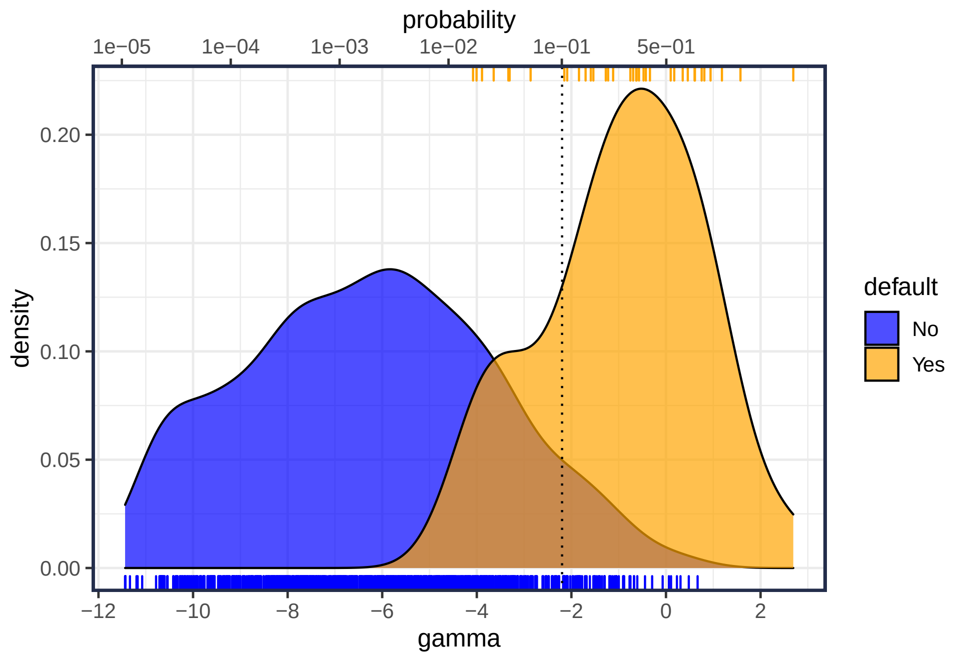
4.5 Performance over a range of thresholds

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.

```r
#-- Get predictions (of gamma(x)) on test data
gamma = predict(fit.lm, newdata=Default[test,], type='link')
```

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



- We can calculate performance over a range of thresholds.
  - Unless the test data is too large, use all unique values of the training data as the thresholds. If too large, manually create threshold sequence.

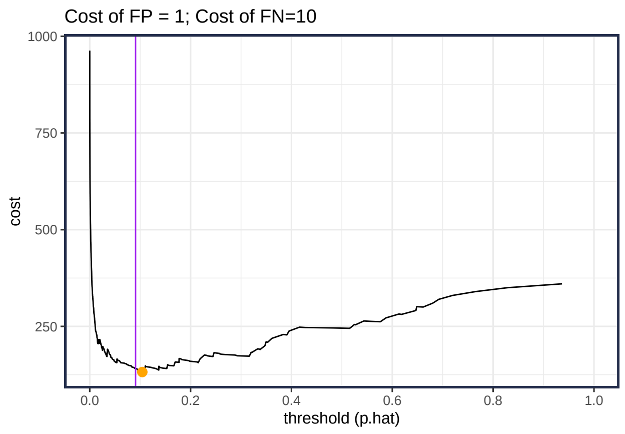
```
#-- Get performance data (by threshold)
perf = tibble(truth = G.test, gamma, p.hat) %>%
  #- group_by() + summarize() in case of ties
  group_by(gamma, p.hat) %>%
  summarize(n=n(), n.1=sum(truth), n.0=n-sum(truth)) %>% ungroup() %>%
  #- calculate metrics
  arrange(gamma) %>%
  mutate(FN = cumsum(n.1),     # 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)
```

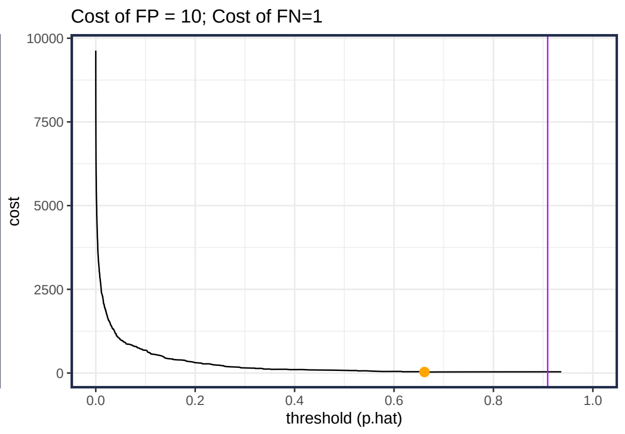
- Note: the `perf` object is *only based on the rank order* of the predictions. This means that the same results would be obtained if we used $\hat{\gamma}(x)$ or $\hat{p}(x)$ to do the ranking.
  - This is because there is a one-to-one monotone relationship between $\hat{\gamma}(x)$ and $\hat{p}(x)$.
  - In the `perf` object I made I grouped by both `gamma` 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.

4.5.1 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 (FP) to false negative costs (FN).

- note: the purple is the *theoretical* optimal threshold (using $t^* = \log FP/FN$ for $\hat{\gamma}(x)$ and $FP/(FP+FN)$ for $\hat{p}(x)$) and the orange point is at the optimal value using the model

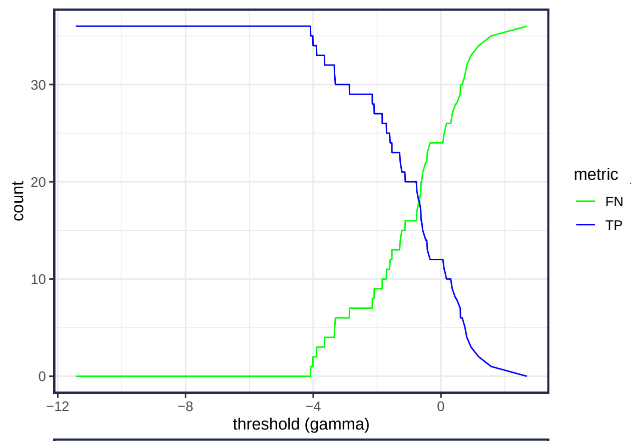


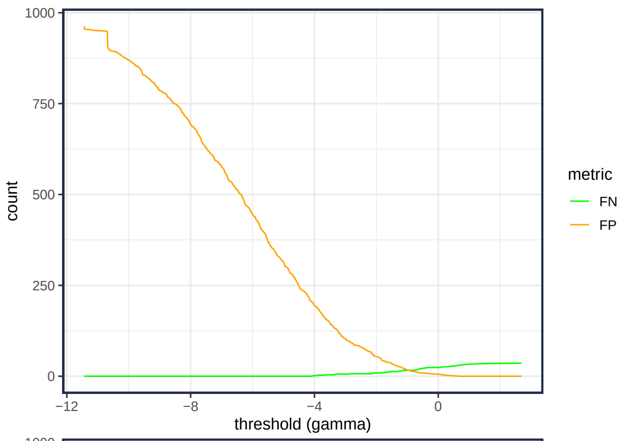


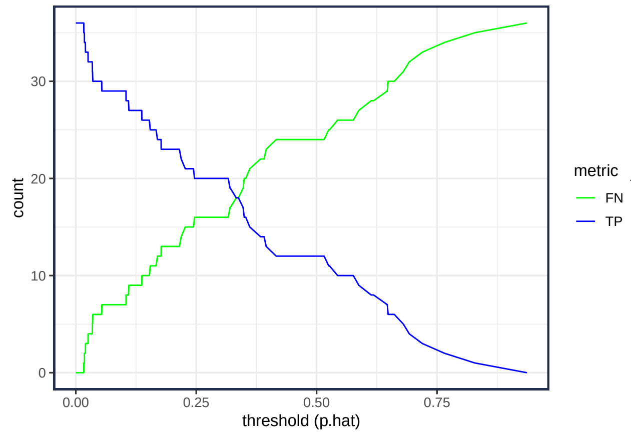
Optimal Threshold

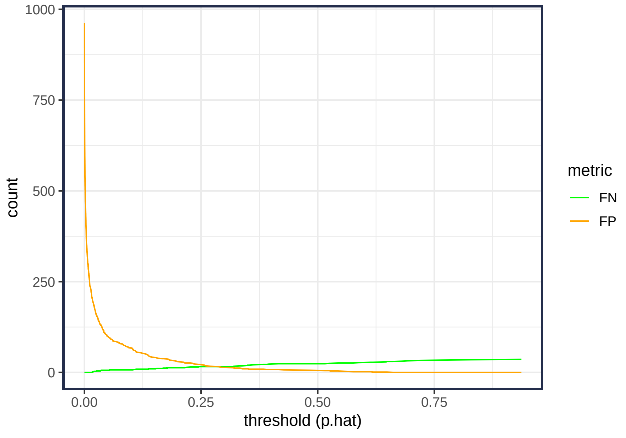
- The *theoretically* optimal threshold is based on the *true* $\gamma(x) = \log \frac{p(x)}{1-p(x)}$ (for a given cost ratio of FP to FN)
- The observed optimal threshold will differ when the model's estimate $\hat{\gamma}(x) \neq \gamma(x)$
  - Hopefully, they are close and it won't make much difference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

4.5.2 General Performance as function of threshold (select metrics)

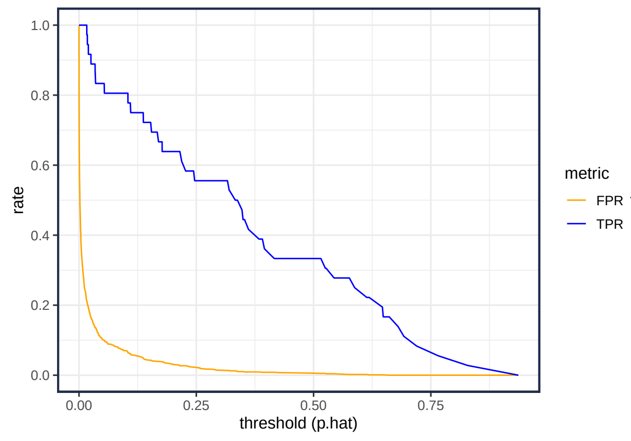


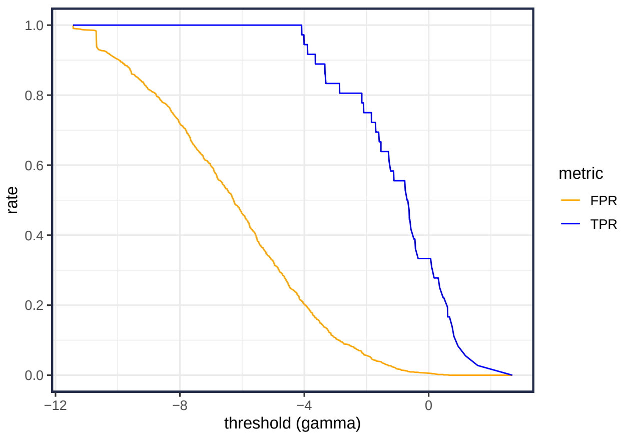


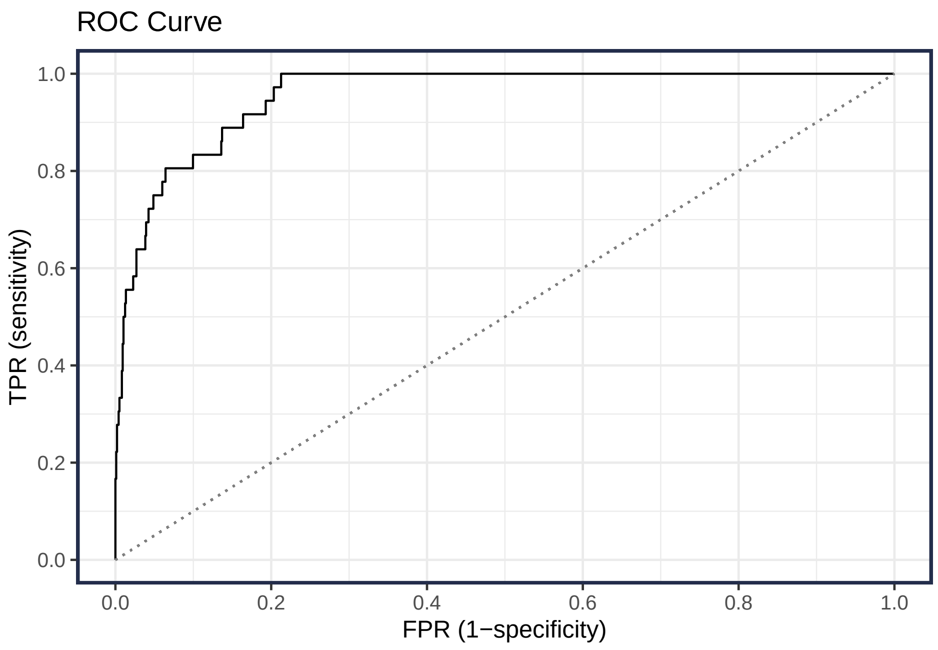




4.5.3 ROC Curves (Receiver Operating Characteristic)





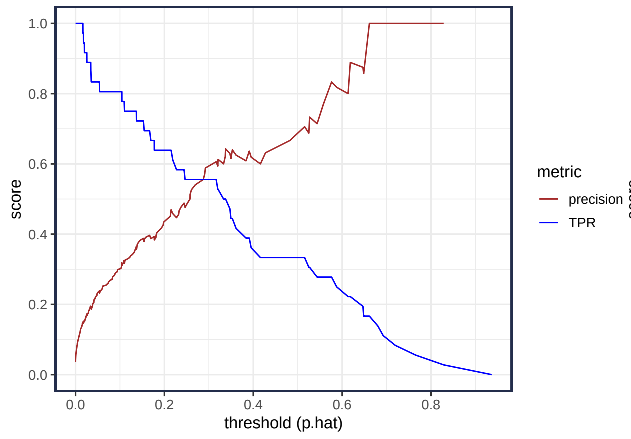


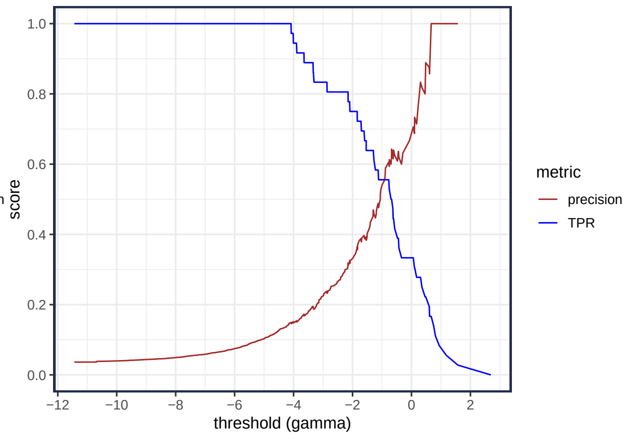
AUROC

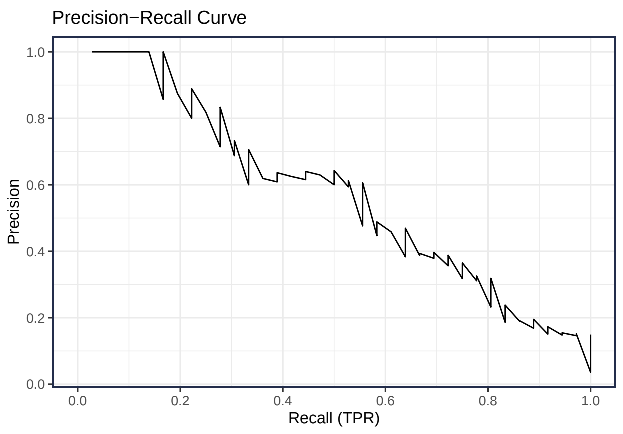
- The *area under the ROC curve* (AUROC) is a popular performance metric
- I don't think it is a great way to compare classifiers for several reasons
  - The main reason is that in a real application you can almost always come up with an estimated cost/loss for the different decisions
  - To say it another 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

4.5.4 Precision Recall Curves

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





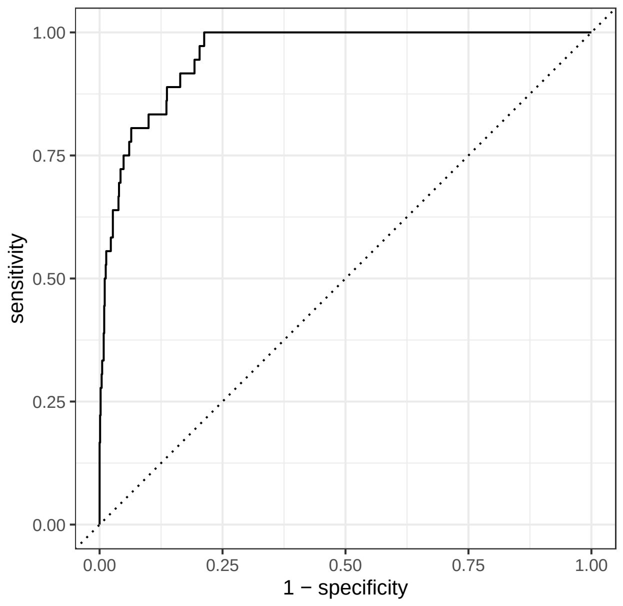


4.5.5 R Code

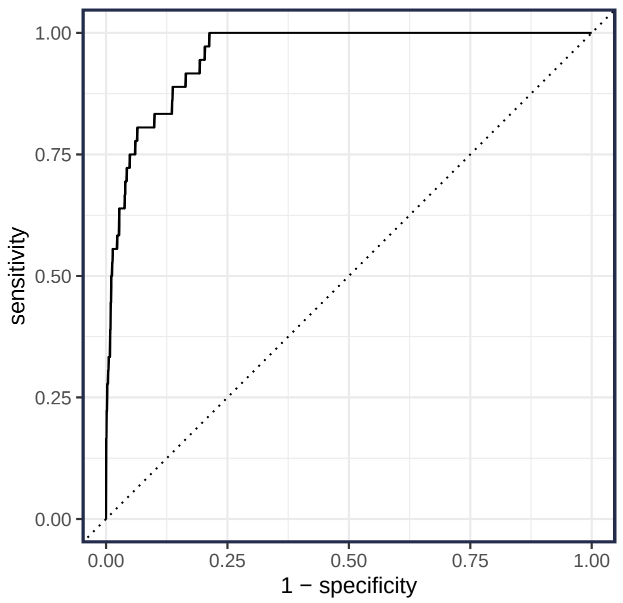
Once we have the FP, TP, TN, FN values for a set of thresholds (like what is in the `perf` object), then we have everything we need to calculate any metric (e.g., gain, lift, F1, . . .).

- 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) %>%
  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

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

4.6 Summary of Classification Evaluation

- Use cost! The other metrics are probably not going to give you what you really want.
  - Resist the urge 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

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

- 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).

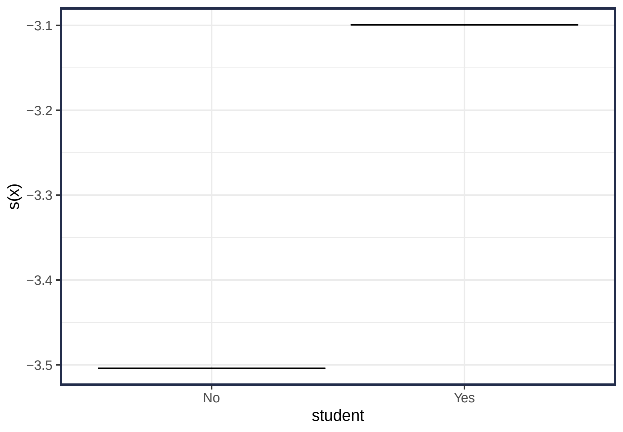
  - Reason: its difficult to estimate model parameters with such loss functions (e.g., non-differentiable, non-unique, etc.)

5 Generalized Additive Models (GAM)

In our discussion of Basis Expansions, we covered how the relationship between a single raw predictor x and the response 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)
```

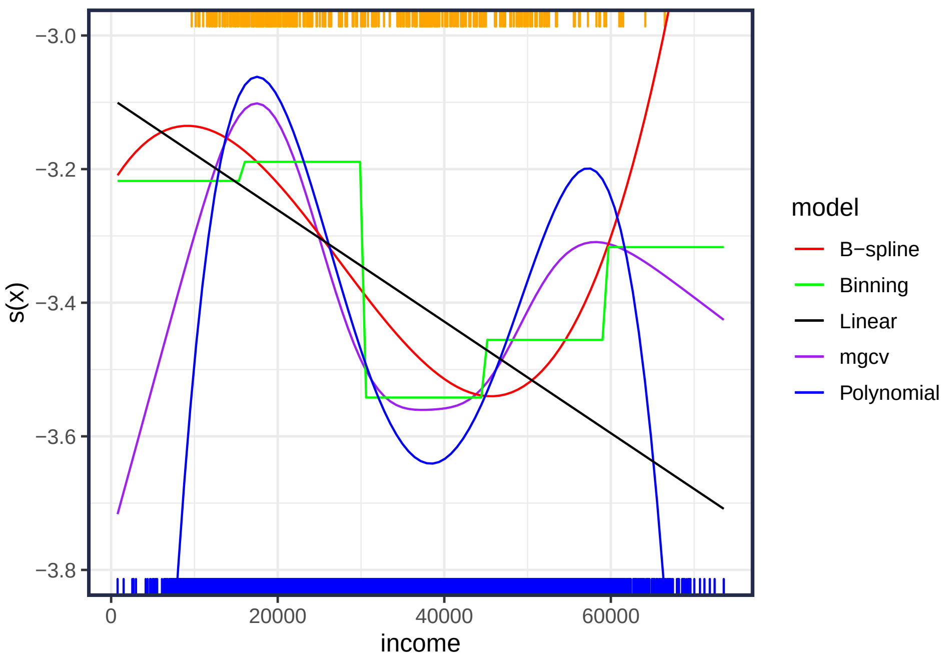
- **Example 4: Continuous Predictor with B-Splines Basis**

```
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"))
```



5.1 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

$$f(\mathbf{x}) = \beta_0 + \sum_j s_j(x_j)$$

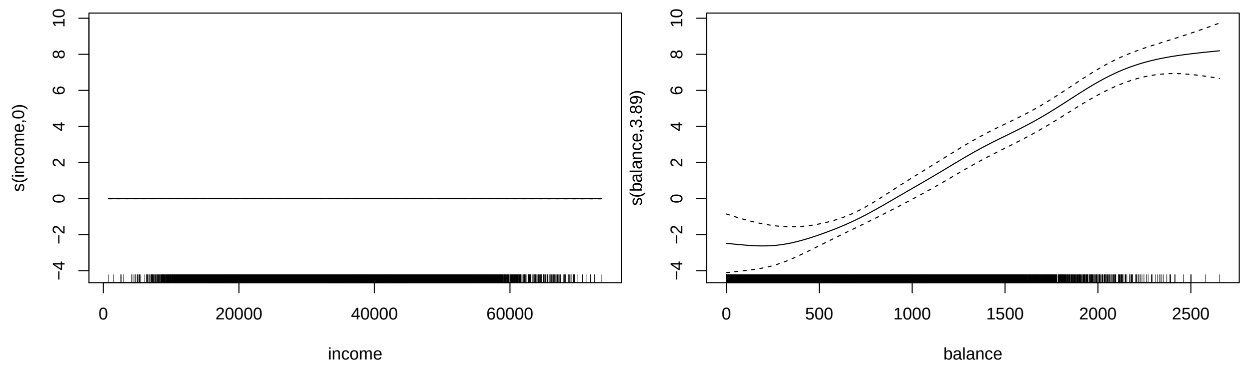
where $s_j(x_j)$ can allow non-linear (e.g., smooth) forms, like B-splines.

- 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)
```



5.2 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 jth predictor
   b. Fit jth 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. - Gauss-Seidel algorithm if linear terms (no smoothing)

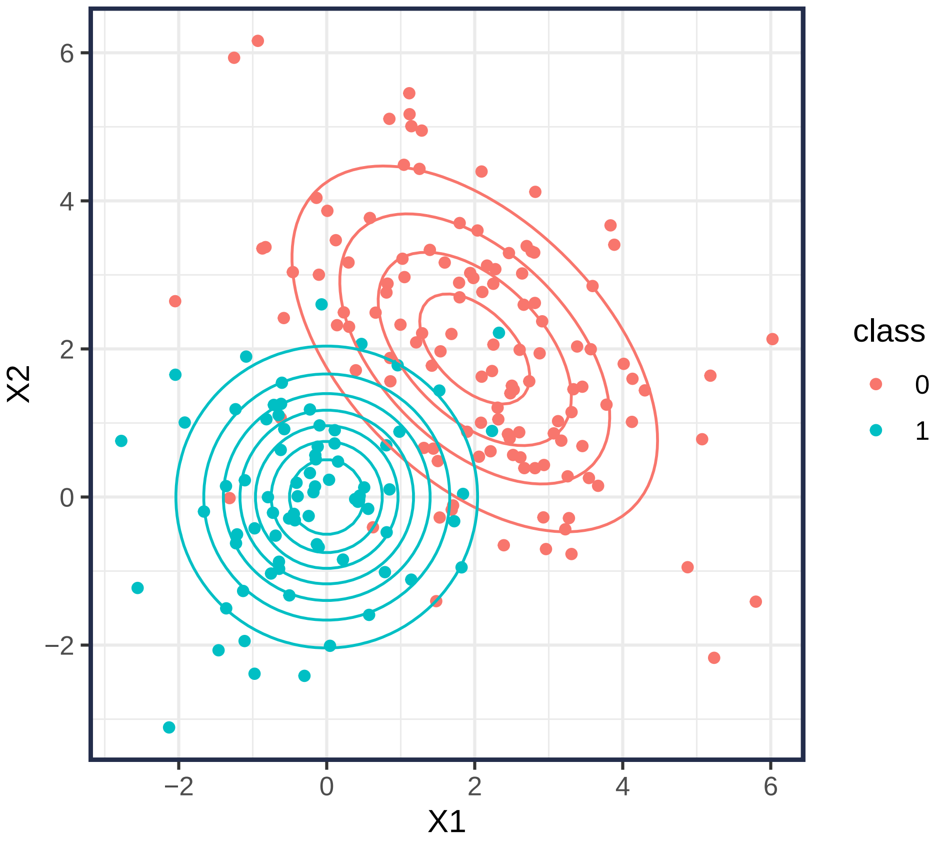
6 Generative Classification Models

Consider how the data $D = \{(X_1, G_1), (X_2, G_2), \ldots, (X_n, G_n)\}$ could be generated.

1. First, the class label is selected according to the *prior probabilities* $\pi = [\pi_1, \ldots, \pi_K]$.
   - That is, $\Pr(G_i = k) = \pi_k$
2. Given the class is k, the X value is generated $X \mid G = k \sim f_k$
   - Let $f_k(\mathbf{x})$ be the (pdf/pmf/mixed) of the predictors from class k.
3. Repeat n times

Example

- Two classes, $k \in \{0, 1\}$
  - $\pi_0 = .60$, $\pi_1 = .40$
  - I expect 40% of the observations to be from class 1.
- X is two-dimensional ($X \in \mathbb{R}^2$)



6.1 From Discriminative to Generative, and Back Again

- The models we have discussed so far are considered *discriminative* and focused on estimating the **conditional** probability $\Pr(Y = k \mid X = x)$
  - Logistic Regression: logit $\hat{p}(x) = f(x)$
- But there is another class of models termed *generative* which try to directly estimate the **joint** probability $\Pr(Y = k, X = x) \propto \Pr(X = x \mid Y = k) \Pr(Y = k)$

6.1.1 The Bayes Breakdown (Binary Classification)

Recall our notation for binary classification problems

- $p(x) = \Pr(Y = 1 \mid X = x) = \frac{f_1(x)\pi}{f_1(x)\pi + f_0(x)(1-\pi)}$
- $\gamma(x) = \frac{p(x)}{1-p(x)}$
- $f_k(x)$ is the *class conditional density*
- $0 \leq \pi_k \leq 1$ are the *prior class probabilities*
- $\pi_0 + \pi_1 = 1$

The log-odds reduces to a combination of prior odds and density (likelihood) ratios

$$\begin{aligned}\gamma(x) &= \log\left(\frac{p(x)}{1-p(x)}\right)\\ &= \underbrace{\log\left(\frac{\pi}{1-\pi}\right)}_{\text{log prior odds}} + \underbrace{\log\left(\frac{f_1(x)}{f_0(x)}\right)}_{\text{log density ratio}}\end{aligned}$$

- Note: $\frac{f_1(x)}{f_0(x)}$ is usually called a *likelihood ratio* when estimated via MLE and *Bayes Factor* when integrating over the model parameters
- We can see that the optimal decision can be based on the density ratios

$$\begin{aligned}&\text{Choose } \hat{G}(x) = 1 \text{ if:}\\ &\hat{\gamma}(x) > \log\left(\frac{L(0,1)-L(0,0)}{L(1,0)-L(1,1)}\right)\\ &\log\left(\frac{1-\hat{\pi}}{\hat{\pi}}\right) + \log\left(\widehat{\frac{f_1(x)}{f_0(x)}}\right) > \log\left(\frac{L(0,1)-L(0,0)}{L(1,0)-L(1,1)}\right)\\ &\log\left(\widehat{\frac{f_1(x)}{f_0(x)}}\right) > \log\left(\frac{1-\hat{\pi}}{\hat{\pi}}\right) + \log\left(\frac{L(0,1)-L(0,0)}{L(1,0)-L(1,1)}\right)\end{aligned}$$

- The challenge in this set-up is to estimate the densities $\{f_k(\cdot)\}$
  - Note: $\hat{\pi}_k = n_k/n$ is a natural estimate for the class priors if we think the testing data will have the same proportions as the training data

6.1.2 Estimation

- Both LDA and QDA model the class conditional densities $f_k(x)$ with *Gaussians*
  - Thus, they model the observations as coming from a *Gaussian mixture model*
  - Each class has its own mean vector μ_k
  - The difference between LDA and QDA is what they use for their covariance matrix
- **LDA**

$$f_k(x) = (2\pi)^{-p/2}|\Sigma|^{-1/2}\exp\left\{-\frac{1}{2}(\mathbf{x}-\mu_k)^\mathsf{T}\Sigma^{-1}(\mathbf{x}-\mu_k)\right\}$$

  - $\Sigma_k = \Sigma \quad \forall k$ (*uses the same variance-covariance for all classes*)

- **QDA**

$$f_k(x) = (2\pi)^{-p/2}|\Sigma_k|^{-1/2} \exp\left\{-\frac{1}{2}(\mathbf{x}-\mu_k)^\mathsf{T}\Sigma_k{}^{-1}(\mathbf{x}-\mu_k)\right\}$$

  - Σ_k is *different* for each classes
- **Naive Bayes** ignores potential associations between predictors and estimates the density of each predictor variable independently.

$$\hat{f}_k(x) = \sum_{j=1}^{p} \hat{f}_{jk}(x_j)$$

  - This greatly simplifies the estimation
  - You will often find $f_{jk} = \mathcal{N}(x_j; \mu_{jk}, \sigma_{jk})$
  - But can mix continuous and discrete variables very easily

We will describe this in more depth after we cover density estimation!