Generative Classifiers

LDQ, QDA, Naive Bayes

SYS 6018 | Spring 2025

gen-classifiers.pdf

Contents

1 Classification and Pattern Recognition

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

1.1 Binary Classification

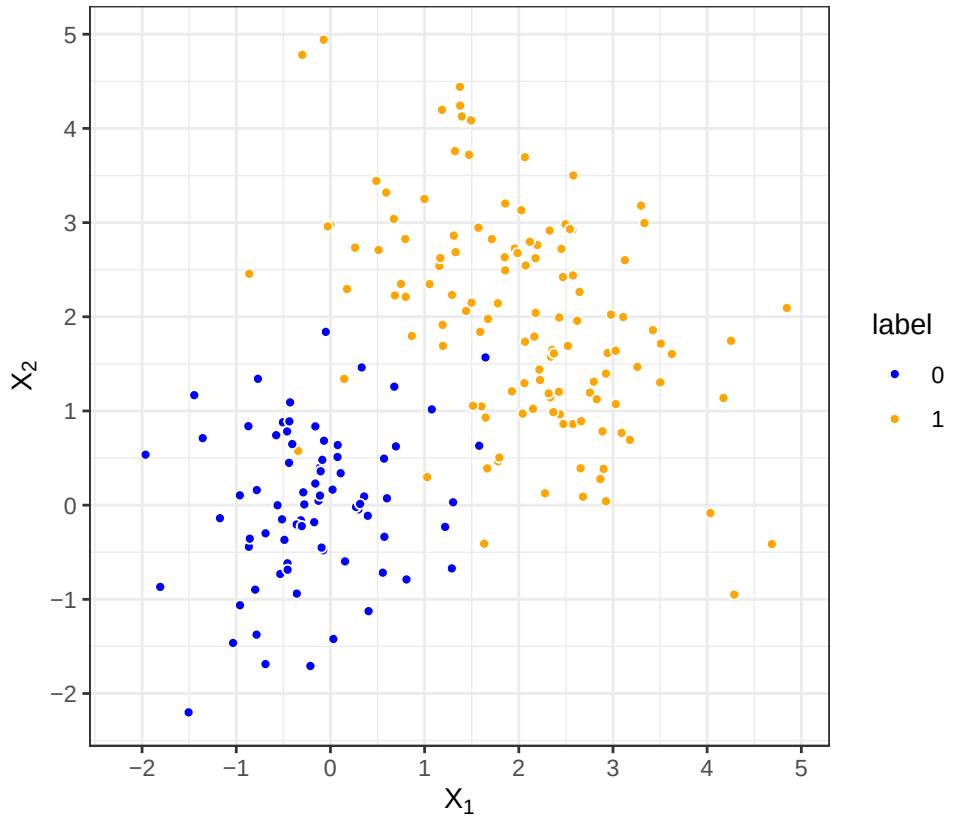
- Classification is simplified when there are only 2 classes.
  - Many multi-class problems can be addressed by solving a set of binary classification problems (e.g., one-vs-rest).
- It is often convenient to transform the outcome variable to a binary $\{0, 1\}$ variable:

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

- Or, like with SVM, as a $\{-1, +1\}$ variable:

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

1.2 Two-Class Example



Your Turn #1

I simulated these data. How do you think I did it?

1.3 Conditional/Discriminative Models

- The classification models we have covered in this course so far (Logistic Regression, SVM, and KNN) attempt to conditionally estimate a score related to the $\Pr(Y = 1 \mid X = x)$ **conditional on** $X = x$. These models are considered *discriminative* models.
- Their goal is to directly estimate $\Pr(Y = 1 \mid X = x)$ **conditional on** $X = x$.

$$p(x) = \Pr(Y = 1 \mid X = x)$$

a. **Linear Regression (for binary outcomes)**

$$\hat{p}(x; \beta) = \hat{\beta}^{\mathsf{T}} x$$

b. **Logistic Regression**

$$\log\left(\frac{\hat{p}(x; \beta)}{1 - \hat{p}(x; \beta)}\right) = \hat{\beta}^{\mathsf{T}} x$$

and thus,

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

c. **kNN (for binary outcomes)**

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

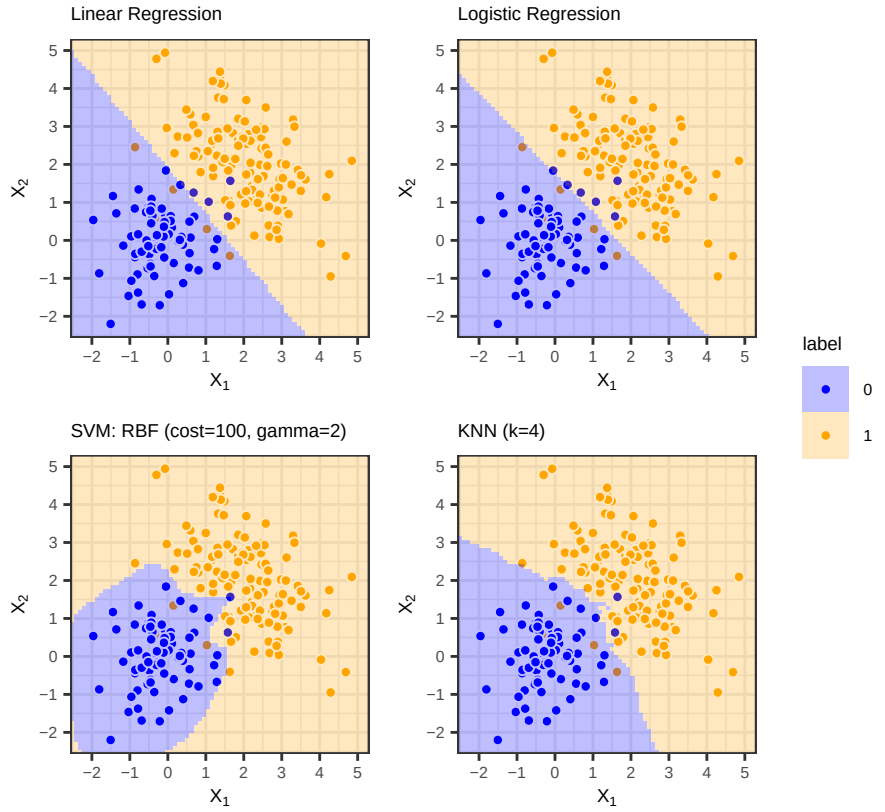
- $N_k(x)$ are the set of k closest training points to x

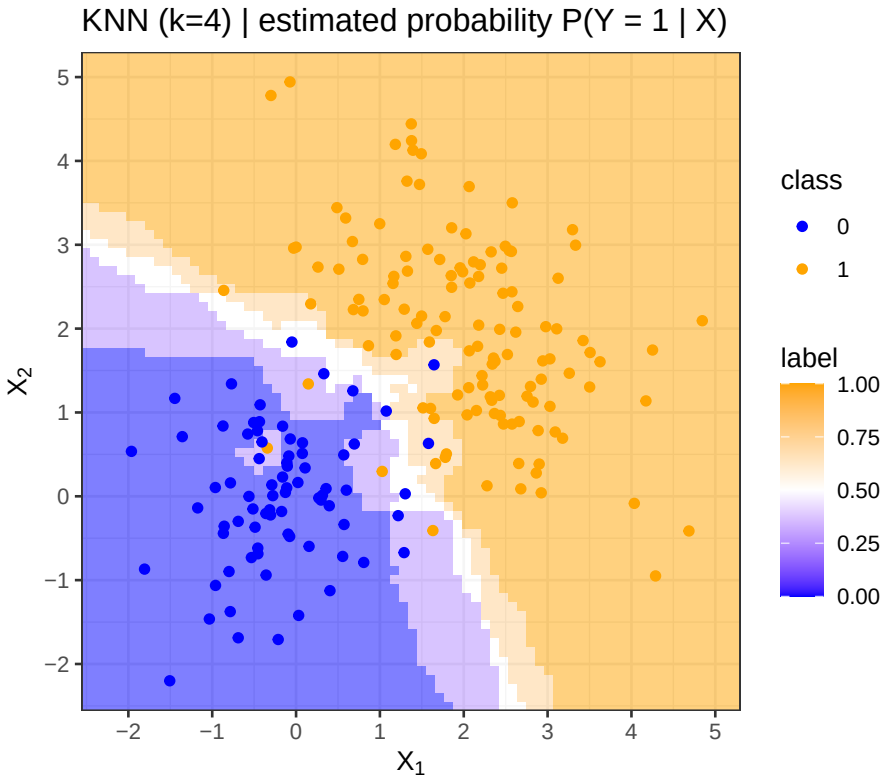
d. **Support Vector Machines (SVM)**

$$\hat{g}(x) = \hat{\alpha}_0 + \sum_{i=1}^{n} \hat{\alpha}_i\, y_i\, K(x, x_i)$$

- Decide $\hat{Y}=1$ if $\hat{g}(x) >$ threshold
- Or calibrated probability: $\log \frac{\tilde{p}(x)}{1-\tilde{p}(x)} = \tilde{\beta}_0 + \tilde{\beta}_1 \hat{g}(x)$
  - I.e., using logistic regression with $\hat{g}(x)$ as the predictor.

$$\begin{aligned}\hat{p}(x;\hat{g}) &= \frac{e^{\tilde{\beta}_0+\tilde{\beta}_1\hat{g}(x)}}{1+e^{\tilde{\beta}_0+\tilde{\beta}_1\hat{g}(x)}} \\ &= \left(1+e^{-\tilde{\beta}_0-\tilde{\beta}_1\hat{g}(x)}\right)^{-1}\end{aligned}$$


Linear Regression
Logistic Regression
SVM: RBF (cost=100, gamma=2)
KNN (k=4)
label
0
1
X1
X2


KNN (k=4) | estimated probability P(Y = 1 | X)
class
0
1
label
1.00
0.75
0.50
0.25
0.00
X1
X2

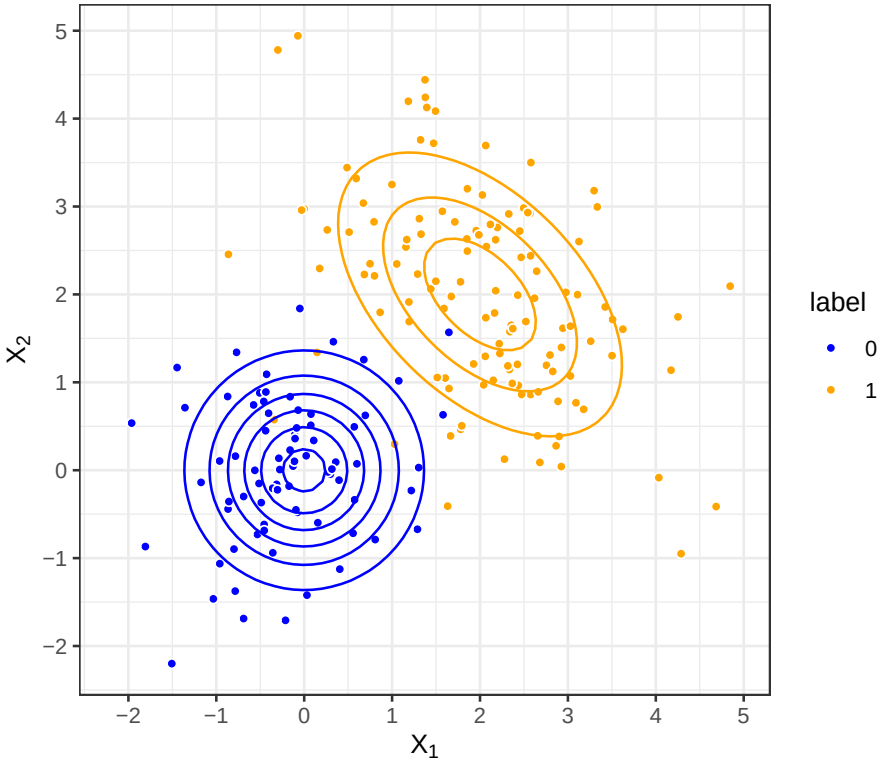
2 Generative Classification Models

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

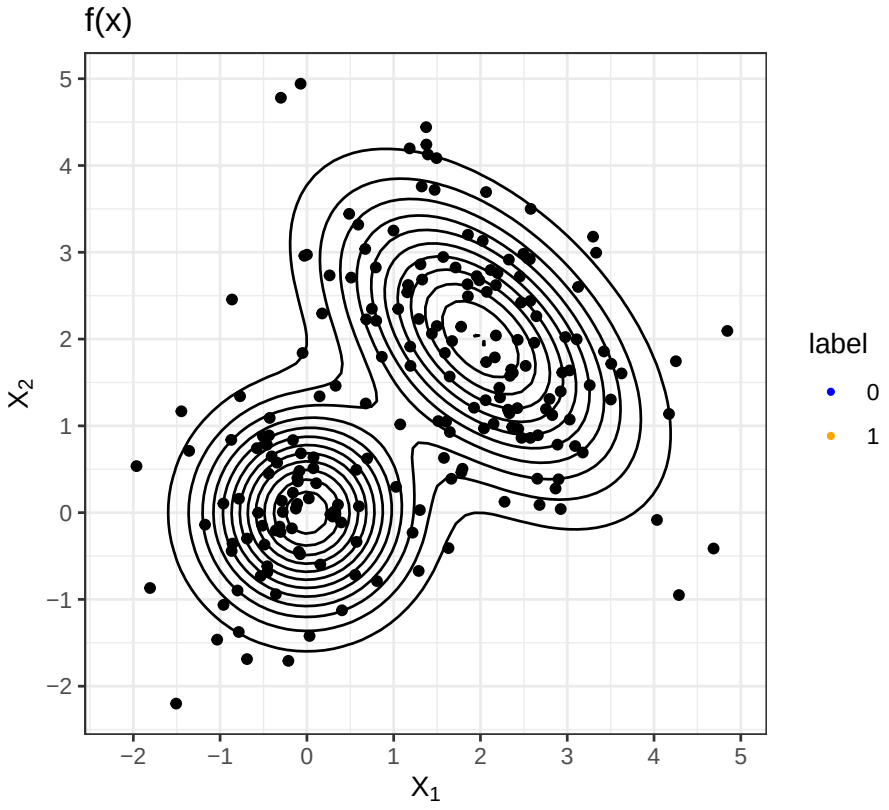
Example

- Two classes, $k \in \{1, 0\}$
  - $\pi_1 = 0.6$, $\pi_0 = 0.4$
  - I *expect* 60% of the observations to be from class 1.
- If $G_i = 1$, then $X \sim N\left(\mu_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}\right)$
- If $G_i = 0$, then $X \sim N\left(\mu_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \Sigma_0 = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}\right)$



Your Turn #2

Use Bayes Theorem to re-write the expression for $\Pr(Y = 1 \mid X = x)$.



2.1 From Discriminative to Generative, and Back Again

- The models we have discussed in this course so far are considered *discriminative* and focused on estimating the **conditional** probability $\Pr(Y = k \mid X = x)$.
  - Or in case of SVM, a score representing the distance to the separating boundary.
- But there is another class of models termed *generative* which try to directly estimate the **joint** probability $\Pr(Y = k, X = x) = \Pr(X = x \mid Y = k) \Pr(Y = k)$.
  - This flips the script; instead of using supervised models to estimate $\Pr(Y = k \mid X = x)$, we use unsupervised density estimation to estimate $\Pr(X = x \mid Y = k)$.

2.1.1 The Bayes Breakdown (Binary Classification)

Bayes Theorem

$$p_k(x) = \Pr(Y = k \mid X = x) = \frac{\Pr(X = x \mid Y = k) \Pr(Y = k)}{\Pr(X = x)} = \frac{f_k(x)\pi_k}{\sum_j f_j(x)\pi_j}$$

- $f_k(x)$ is the *class conditional density* (pdf/pmf)
- $0 \leq \pi_k \leq 1$ are the *prior class probabilities*
- $\sum \pi_k = 1$
- X is distributed as a finite mixture model: $f(x) = \sum_j f_j(x)\pi_j$

2.1.1.1 Special case when $K = 2$ (binary classification)

$$p(x) = \Pr(Y = 1 \mid X = x) = \frac{\Pr(X = x \mid Y = 1) \Pr(Y = 1)}{\Pr(X = x)} = \frac{f_1(x)\pi}{f_1(x)\pi + f_0(x)(1 - \pi)}$$

Recall our notation for the log-odds:

- $\gamma(x) = \log \frac{p(x)}{1-p(x)}$

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

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

2.1.2 Decision-Making (Hard Classification)

- We know that the optimal decision can be based on the density ratios

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

2.1.3 Estimation

- $\hat{\pi}_k = n_k/n$ is a natural estimate for the class priors if we think the testing data will have the same proportions as the training data
- The other term to estimate is the log density ratio: $\log\left(\widehat{\frac{f_1(x)}{f_0(x)}}\right)$
- Generative Models estimate this term by

$$\log\left(\widehat{\frac{f_1(x)}{f_0(x)}}\right) = \log\left(\frac{\hat{f}_1(x)}{\hat{f}_0(x)}\right)$$

- That is, generative models estimate the class conditional densities $\{f_k(\cdot)\}$
- The different generative models take different approaches to estimate these component densities

Generative Models

Generative Classification Models use *density estimation* to make predictions!

2.1.3.1 Linear/Quadratic Discriminant Analysis (LDA/QDA)

- Both LDA and QDA model the class conditional densities $f_k(x)$ with a *Gaussian* density
  - 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

2.1.3.2 Kernel Discriminant Analysis (KDA)

- Model the class conditional densities $f_k(x)$ with a multivariate *kernel density estimate (KDE)*

$$\hat{f}_k(x) = \frac{1}{n_k} \sum_{i:g_i=k} K(x - x_i; H)$$

where H is the $p \times p$ bandwidth matrix.

2.1.3.3 Mixture Discriminant Analysis (MDA)

- Model the class conditional densities $f_k(x)$ with a *finite mixture model*

$$\hat{f}_k(x) = \frac{1}{J} \sum_{j=1}^{J} \pi_j g_j(x; \theta_j)$$

where $\sum_{j=1}^{J} \pi_j = 1$ and $g_j(x)$ is a density function (e.g., Gaussian).

2.1.3.4 Naive Bayes

- **Naive Bayes** ignores potential associations between predictors and estimates the density of each predictor variable independently.

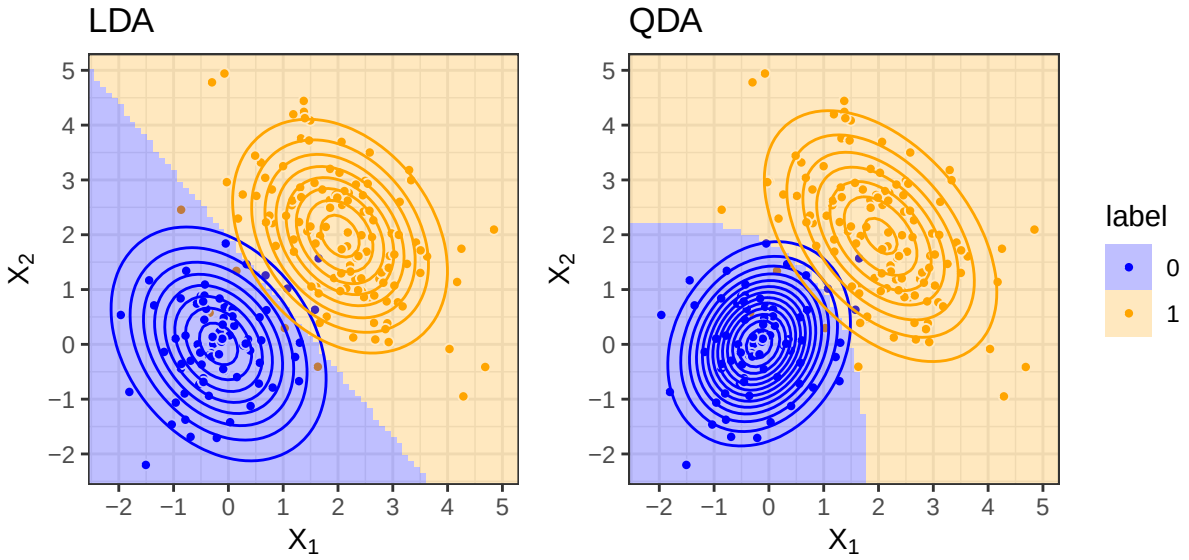
$$\hat{f}_k(x) = \prod_{j=1}^{p} \hat{f}_{jk}(x_j))$$

  - This greatly simplifies the estimation
  - You will often find $\hat{f}_{jk}(u) = \mathcal{N}(u; \hat{\mu}_{jk}, \hat{\sigma}_{jk})$
  - But KDE is a great approach $\hat{f}_{jk}(u) = \frac{1}{n_k} \sum_{\{i:G_i=k\}} K_h(u - x_{ij})$
  - And including mix continuous and discrete variables is very easily

Your Turn #3

How would you estimate the probability for a categorical predictor?

3 Linear/Quadratic Discriminant Analysis (LDA/QDA)



- *Linear Discriminant Analysis (LDA)* finds *linear* boundaries between classes
- *Quadratic Discriminant Analysis (QDA)* finds *quadratic* boundaries between classes

- Setup: $K = |\mathcal{G}|$ classes in the training data, $D = \{(\mathbf{X}_i, G_i)\}_{i=1}^n$
  - where $\mathbf{X}_i \in \mathbf{R}^p$, $G_i \in \mathcal{G}$
- The posterior probability of class g, given $X = x$,

$$\begin{aligned}\Pr(G = g \mid \mathbf{X} = \mathbf{x}) &= \frac{f(x \mid G = g)\Pr(G = g)}{f(x)} \\ &= \frac{f_g(x)\pi_g}{\sum_{k=1}^K f_k(x)\pi_k}\end{aligned}$$

  - $f_k(x)$ is the *class conditional density*
  - $0 \leq \pi_k \leq 1$ are the *prior class probabilities*; $\sum_{k=1}^K \pi_k = 1$

3.1 Estimation

- Both LDA and QDA model the class conditional densities $f_k(x)$ with *Gaussians*
  - Thus, they model the observations as coming from a K component *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

$$f_k(x) = \mathcal{N}(x; \mu_k, \Sigma_k)$$

- LDA: $\hat{\Sigma}_1 = \hat{\Sigma}_2 = \ldots = \hat{\Sigma}_K = \hat{\Sigma}$ Common covariance
- QDA: $\hat{\Sigma}_1 \neq \hat{\Sigma}_2 \neq \ldots \neq \hat{\Sigma}_K$ Different covariances

- **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

Note

In R, the density $f_k(x)$ can be computed with `mvtnorm::dmvnorm()` (from the `mvtnorm` package). It requires the mean vector μ_k and the variance-covariance matrix Σ_k.

Your Turn #4 : Model Complexity

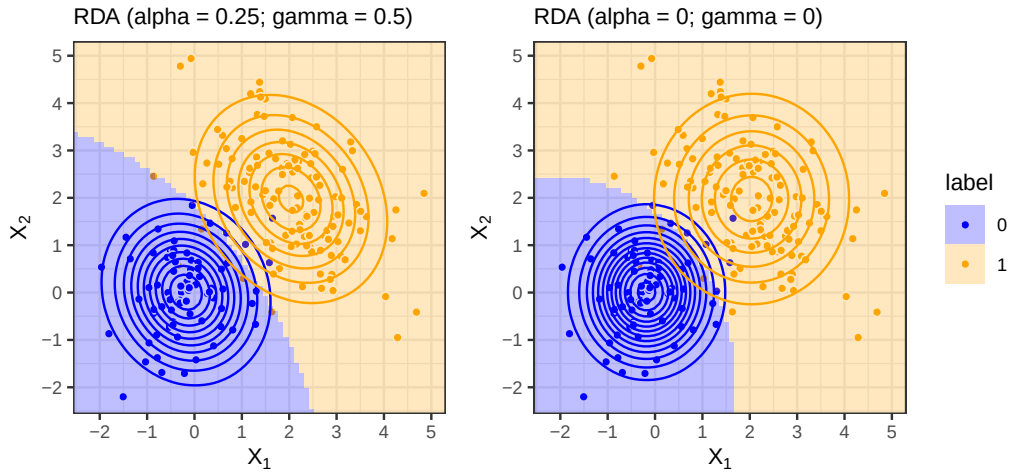
The LDA model uses a common covariance matrix while QDA allows each class to have a different covariance (which permits quadratic boundaries). But this flexibility comes at a cost.

1. How many parameters have to be estimated in an LDA model with K classes and p dimensions?

2. How many parameters have to be estimated in an QDA model with K classes and p dimensions?

- There are a few methods to maintain some flexibility, yet protect the model from high variance
- One is to use a *regularlized covariance matrix* (see ESL 4.3.1). Called Regularlized Discriminant Analysis (RDA)

$$\hat{\Sigma}_k(\alpha,\gamma) = \alpha\hat{\Sigma}_k + (1-\alpha)\{\gamma\hat{\Sigma} + (1-\gamma)\hat{\sigma}^2 I_p\}$$



- A special case of above using diagonal covariance matrices only ($\hat{\Sigma}_k(\alpha=0,\gamma=0)$). This covariance

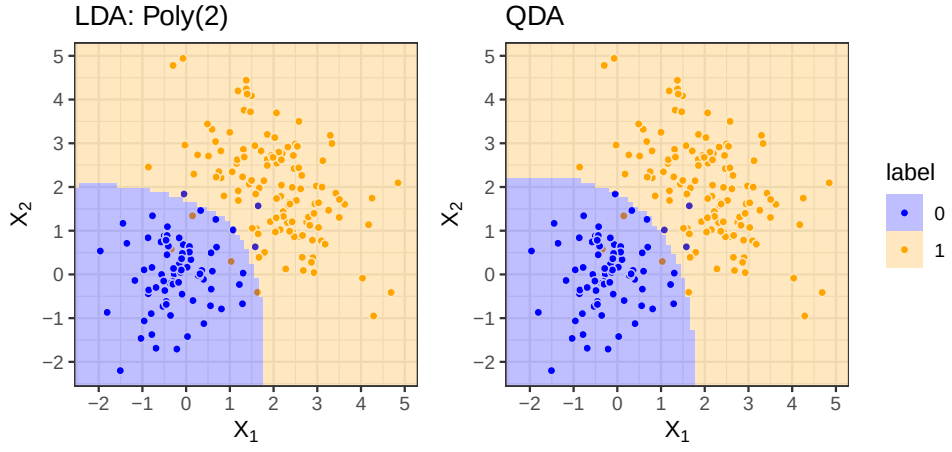
matrix has all off-diagonal terms set to 0.

$$\hat{\Sigma}_k = diag(\hat{\sigma}_1^2, \hat{\sigma}_2^2, \ldots, \hat{\sigma}_p^2) = \begin{bmatrix} \sigma_1^2 & 0 & \ldots & 0 \\ 0 & \sigma_2^2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \sigma_p^2 \end{bmatrix}$$

  - This treats predictors/features as uncorrelated/independent.
  - It is a special case of *Naive Bayes*!
- A more restrictive (less complex) model specifies that variance in all dimensions are equal

$$\hat{\Sigma}_k = \hat{\sigma}^2 I_p = \begin{bmatrix} \sigma^2 & 0 & \ldots & 0 \\ 0 & \sigma^2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \sigma^2 \end{bmatrix}$$

  - This treats predictors/features as uncorrelated/independent.
  - It is a special, special, case of *Naive Bayes*!
  - Models all variances as equal.
- In some settings (large K, small p), edf could be reduced by fitting an LDA model in an *enlarged feature space*
  - E.g., for $p = 2$ dimensions, use $X_1, X_2, X_1 \cdot X_2, X_1^2, X_2^2$ instead of QDA in X_1, X_2.
  - Think basis expansion like what we did with polynomial regression or B-splines
  - Or kernels with SVM



Mahalanobis Distance

Notice that a multivariate normal density is a function of the squared *Mahalanobis* distance from x to the mean.

$$f(\mathbf{x};\mu,\mathbf{\Sigma}) = (2\pi)^{-p/2}|\Sigma|^{-1/2}\exp\left\{-\frac{1}{2}(\mathbf{x}-\mu)^\intercal\Sigma^{-1}(\mathbf{x}-\mu)\right\}$$
$$= (2\pi)^{-p/2}|\Sigma|^{-1/2}\exp\left\{-\frac{1}{2}D^2(x)\right\}$$

where

$$D(x) = \sqrt{(\mathbf{x}-\mu)^\intercal\Sigma^{-1}(\mathbf{x}-\mu)}$$

is the Mahalanobis distance.

3.2 LDA/QDA in Action

- In **R**, LDA and QDA can be implemented with the `MASS::lda()` and `MASS::qda()` functions from the `MASS` package.
  - Note conflicts between `MASS::slice()` and `dplyr::slice()`
- See ISLR 4.7 for details
- Warning: the `MASS` package has a `select()` functions that conflicts with `dplyr`'s `select()`. If you use `tidyverse`, I suggest you use `MASS::lda()` and `MASS::qda()` instead of loading the entire `MASS` package.

3.3 Connections: LDA, QDA, and Logistic Regression

ISL 4.5 and ESL 4.4.5 show more details about the parametric form LDA and QDA take.

Recall the notation for generative models:

$$\hat{\gamma}(x) = \log\left(\frac{\hat{p}(x)}{1-\hat{p}(x)}\right)$$
$$= \log\left(\frac{\hat{\pi}}{1-\hat{\pi}}\right) + \log\left(\frac{\hat{f}_1(x)}{\hat{f}_0(x)}\right)$$

Logistic Regression

$$\hat{\gamma}(x) = \hat{\beta}_0 + \sum_{j=1}^{p}\hat{\beta}_j x_j \qquad \text{Main Effects}$$

$$\hat{\gamma}(x) = \hat{\beta}_0 + \sum_{j=1}^{p}\hat{\beta}_j x_j + \sum_{j=1}^{p}\sum_{k=1}^{p}\hat{\beta}_{jk}x_j x_k \qquad \text{Quadratic Terms}$$

LDA

$$\hat{\gamma}(x) = \hat{\alpha}_0 + \sum_{j=1}^{p} \hat{\alpha}_j x_j$$

$$\begin{aligned}\hat{a}_0 &= \log \frac{\hat{\pi}}{1-\hat{\pi}} - \frac{1}{2}(\hat{\mu}_1 - \hat{\mu}_0)^\mathsf{T} \hat{\Sigma}^{-1}(\hat{\mu}_1 - \hat{\mu}_0) \\ \hat{a}_j &= \text{the } j\text{th element of } \hat{\Sigma}^{-1}(\hat{\mu}_1 - \hat{\mu}_0)\end{aligned}$$

QDA

$$\hat{\gamma}(x) = \hat{a}_0 + \sum_{j=1}^{p} \hat{a}_j x_j + \sum_{j=1}^{p}\sum_{k=1}^{p} \hat{a}_{jk} x_j x_k$$

$$\begin{aligned}\hat{a}_0 &= \log \frac{\hat{\pi}}{1-\hat{\pi}} - \frac{1}{2}\log\frac{|\hat{\Sigma}_1|}{|\hat{\Sigma}_0|} - \frac{1}{2}\left(\hat{\mu}_1^\mathsf{T}\Sigma_1^{-1} - \hat{\mu}_0^\mathsf{T}\Sigma_0^{-1}\right) \\ \hat{a}_j &= \text{the } j\text{th element of } \hat{\Sigma}_1^{-1}\hat{\mu}_1 - \hat{\Sigma}_0^{-1}\hat{\mu}_0 \\ \hat{a}_{jk} &= \text{the } (j,k)\text{th element of } (\hat{\Sigma}_0^{-1} - \hat{\Sigma}_1^{-1})/2\end{aligned}$$

3.3.1 Estimation

LDA and QDA estimates model parameters by maximizing the *joint* likelihood:

$$\begin{aligned}\hat{\alpha} &= \arg\max_{\alpha} \ \Pr(X, Y) \\ &= \arg\max_{\alpha} \ \Pr(X \mid Y)\Pr(Y) \\ &= \arg\max_{\alpha} \ \Pr(Y \mid X)\Pr(X)\end{aligned}$$

Logistic Regression estimates model parameters by maximizing the *conditional* likelihood

$$\hat{\beta} = \arg\max_{\beta} \ \Pr(Y \mid X)$$

4 Kernel Discriminant Analysis (KDA)

- Model the class conditional densities $f_k(x)$ with a multivariate *kernel density estimate (KDE)*

$$f_k(x) = \frac{1}{n_k} \sum_{i:g_i=k} K(x - x_i; H_k)$$

where H_k is the $p \times p$ bandwidth matrix.

There are three primary approaches to multivariate (p dimensional) KDE:

1. Multivariate kernels
   - e.g., $K(u) = N(\mathbf{0}, H_k)$:

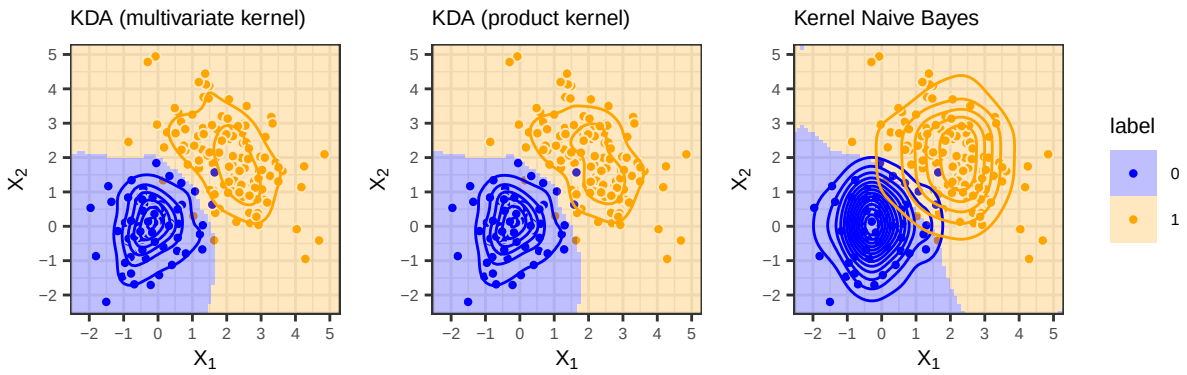
$$\hat{f}_k(x) = \frac{1}{(2\pi)^{d/2}|H_k|^{1/2}n} \sum_{i=1}^{n} \exp\left(-\frac{1}{2}(x - x_i)^\mathsf{T} H_k^{-1}(x - x_i)\right)$$

2. Product Kernels
   - $H_k = diag(h_{k1}, h_{k2}, \ldots, h_{kp})$

$$\hat{f}_k(x) = \frac{1}{n} \sum_{i=1}^{n} \left(\prod_{j=1}^{p} K(x_j - x_{ij}; h_{kj}) \right)$$

3. Independence
   - This is a special case of *Naive Bayes* (Kernel Naive Bayes)!

$$\begin{aligned} \hat{f}_k(x) &= \prod_{j=1}^{p} \hat{f}_{kj}(x) \\ &= \prod_{j=1}^{p} \left(\frac{1}{n} \sum_{i=1}^{n} K(x_j - x_{ij}; h_{kj}) \right) \end{aligned}$$



4.1 KDA with R

- In **R**, the `ks::kda()` function (`ks` package) implements Kernel Discriminant Analysis.

5 Naive Bayes

$$\Pr(G = g \mid X = x) = \frac{\pi_g \prod_{j=1}^{p} \hat{f}_{gj}(x_j)}{\sum_k \pi_k \prod_{j=1}^{p} \hat{f}_{kj}(x_j)}$$

Naive Bayes is a generative model that ignores potential associations between predictors and estimates the density of each predictor variable independently.

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

- This greatly simplifies the estimation
- The densities do *not* have to be Gaussian (e.g., KDE is a good option)
- Categorical densities (i.e., pmfs) can be thrown in the mix without a problem
- Because of the independence, this is easy to implement in parallel (and thus can be fast)

The estimated posterior probability under Naive Bayes becomes

$$\widehat{\Pr}(G = g \mid X = x) = \hat{p}_g(x) = \frac{\hat{\pi}_g \prod_{j=1}^{p} \hat{f}_{gj}(x_j)}{\sum_k \hat{\pi}_k \prod_{j=1}^{p} \hat{f}_{kj}(x_j)}$$

For binary outcomes the decision function is:

$$\begin{aligned}
\hat{\gamma}(x) &= \log\left(\frac{\hat{p}(x)}{1-\hat{p}(x)}\right) \\
&= \log\left(\frac{\hat{\pi}}{1-\hat{\pi}}\right) + \log\left(\frac{\hat{f}_1(x)}{\hat{f}_0(x)}\right) \\
&= \log\left(\frac{\hat{\pi}}{1-\hat{\pi}}\right) + \log\left(\frac{\prod_{j=1}^{p} \hat{f}_{1j}(x_j)}{\prod_{j=1}^{p} \hat{f}_{0j}(x_j)}\right) \\
&= \log\left(\frac{\hat{\pi}}{1-\hat{\pi}}\right) + \log\left(\prod_{j=1}^{p} \frac{\hat{f}_{1j}(x_j)}{\hat{f}_{0j}(x_j)}\right) \\
&= \log\left(\frac{\hat{\pi}}{1-\hat{\pi}}\right) + \sum_{j=1}^{p} \log\left(\frac{\hat{f}_{1j}(x_j)}{\hat{f}_{0j}(x_j)}\right)
\end{aligned}$$

5.1 Gaussian Naive Bayes

- Recall in LDA/QDA, the class conditional densities were estimated as Gaussians:

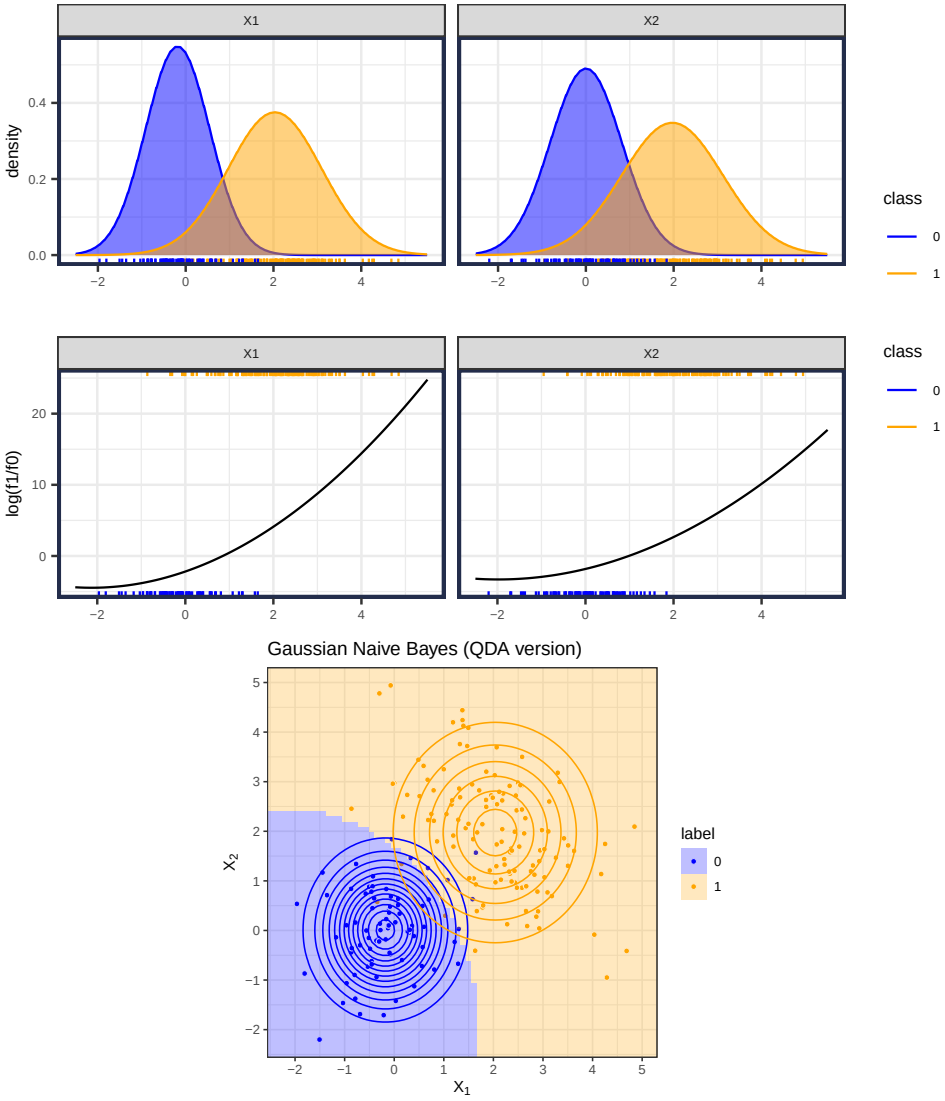
$$\hat{f}_k(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \hat{\mu}_k, \hat{\Sigma}_k)$$

  - But when the dimensionality of $\mathbf{x}$ gets large or there is high correlation, estimation of $\hat{\Sigma}_k$ can be poor
- If we force $\hat{\Sigma}_k$ to be *diagonal* then the densities are product of univariate Gaussians (called Gaussian Naive Bayes)

$$\hat{f}_k(\mathbf{x}) = \prod_{j=1}^{p} \mathcal{N}(x_j; \mu_{kj}, \sigma_{kj})$$

- Even if the data are not independent, this may give better estimates by reducing the variance (at the expense of a bit of bias)
- This is a special case of QDA, where we restrict the off-diagonal terms in the variance-covariance to be 0.

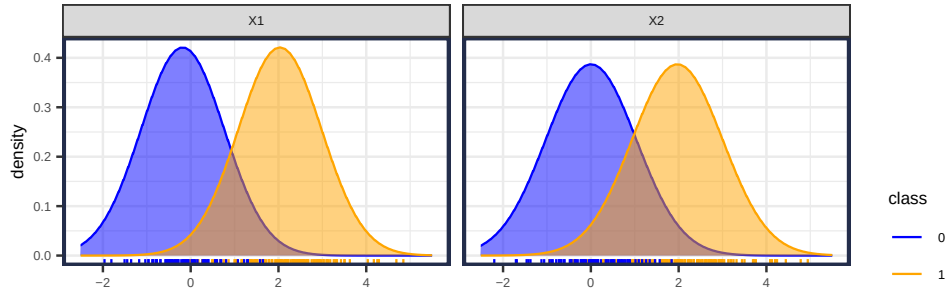
class	predictor	mu	sd	density
0	X1	-0.18	0.73	N(mu = -0.18, sd = 0.73)
1	X1	2.04	1.06	N(mu = 2.04, sd = 1.06)
0	X2	0.01	0.81	N(mu = 0.01, sd = 0.81)
1	X2	1.97	1.15	N(mu = 1.97, sd = 1.15)

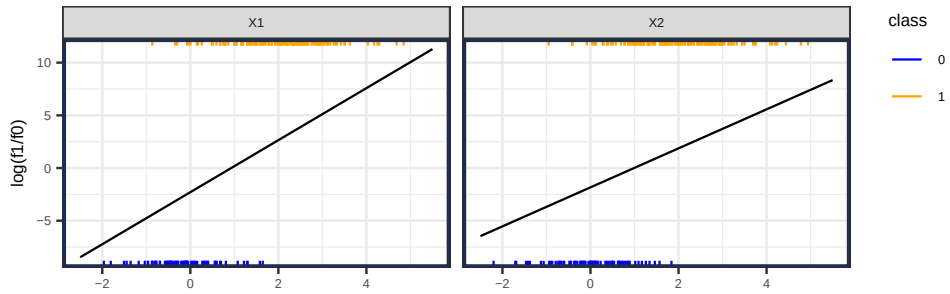


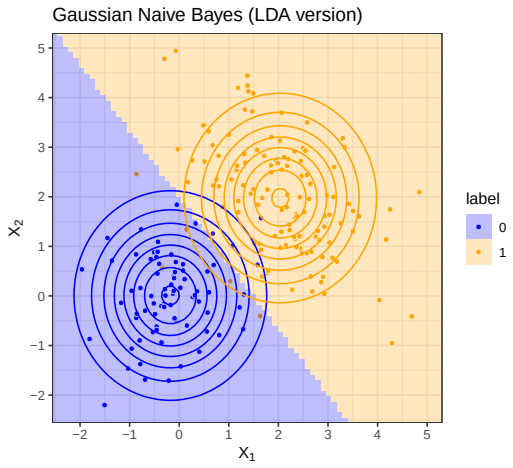
- A simpler model (less complexity/edf) forces a common standard deviation for all class (special case of LDA)

$$\hat{f}_k(\mathbf{x}) = \prod_{j=1}^{p} \mathcal{N}(x_j; \mu_{kj}, \sigma_j)$$

class	predictor	mu	sd	density
0	X1	-0.18	0.95	N(mu = -0.18, sd = 0.95)
1	X1	2.04	0.95	N(mu = 2.04, sd = 0.95)
0	X2	0.01	1.03	N(mu = 0.01, sd = 1.03)
1	X2	1.97	1.03	N(mu = 1.97, sd = 1.03)







5.2 Kernel Naive Bayes

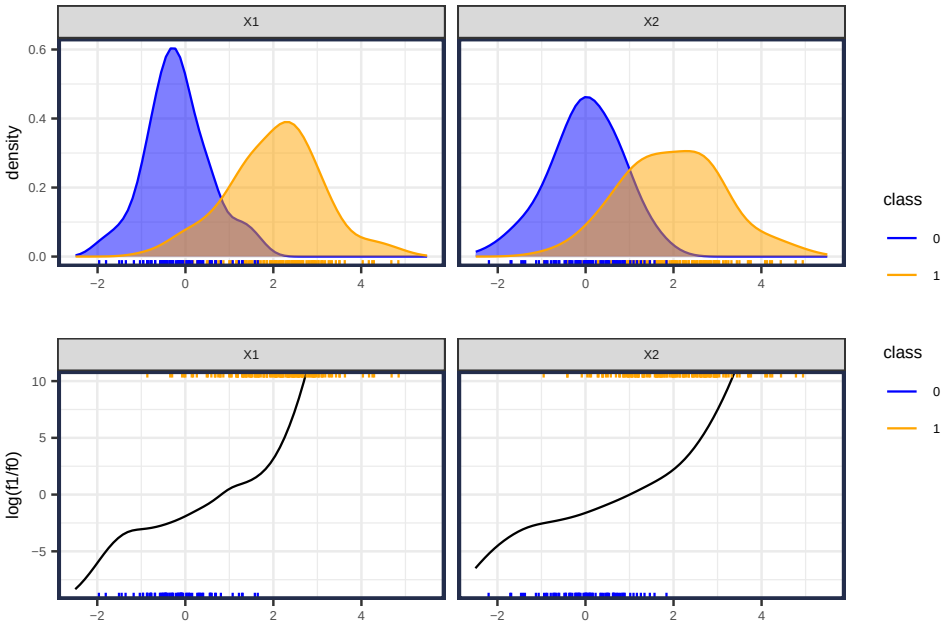
In *kernel density* Naive Bayes, use Kernel Density Estimation (KDE) to estimate each component density:

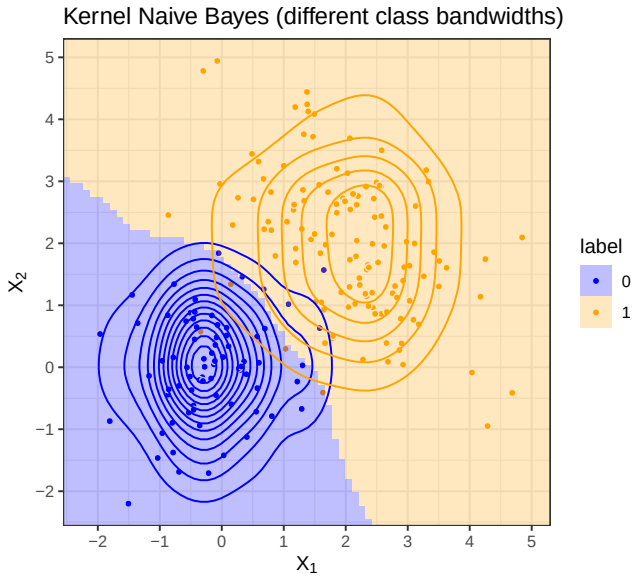
$$\hat{f}_{kj}(x_j) = \frac{1}{n_k} \sum_{i:g_i=k} K(x_j - x_{ij}; h_{kj})$$

with bandwidth parameter h_{kj}.

The density ratio becomes

$$\frac{\hat{f}_{1j}(x_j)}{\hat{f}_{0j}(x_j)} = \frac{\frac{1}{n_1}\sum_{i:g_i=1} K(x_j - x_{ij}; h_{1j})}{\frac{1}{n_0}\sum_{i:g_i=0} K(x_j - x_{ij}; h_{0j})}$$





- for less complex models, use same bandwidth parameter for each class.

Note: this gives a different solution than using KDE with a *product kernel*! (which is not a naive bayes model)

$$\hat{f}_k(\mathbf{x}) = \frac{1}{n_k} \sum_{i:g_i=k} \prod_{j=1}^{p} K(x_j - x_{ij}; h_{kj})$$

6 Connections: Generalized Additive Models (GAM)

It turns out that there is a close connection between Logistic Regression, Naive Bayes, and LDA. To help see this, notice that all three methods can be written:

$$\begin{aligned}\gamma(x) &= \log\left(\frac{\pi}{1-\pi}\right) + \log\left(\frac{f_1(x)}{f_0(x)}\right) \\ &\approx \alpha_0 + \sum_{j=1}^{p} \alpha_j S_j\end{aligned}$$

- **Logistic Regression**

  $\hat{\alpha}_0 = \hat{\beta}_0$
  $\hat{\alpha}_j = \hat{\beta}_j$
  $\hat{S}_j = x_j$

- **LDA**

  $\hat{\alpha}_0 = \log\frac{\hat{\pi}}{1-\hat{\pi}} - \frac{1}{2}(\hat{\mu}_1 + \hat{\mu}_0)^\mathsf{T}\hat{\Sigma}^{-1}(\hat{\mu}_1 - \hat{\mu}_0)$
  $\hat{\alpha}_j = \hat{\Sigma}^{-1}(\hat{\mu}_1 - \hat{\mu}_0)$
  $\hat{S}_j = x_j$

- **Naive Bayes**

  $\hat{\alpha}_0 = \log\frac{\hat{\pi}}{1-\hat{\pi}}$
  $\hat{\alpha}_j = 1$
  $\hat{S}_j = \log\frac{\hat{f}_{1j}(x_j)}{\hat{f}_{0j}(x_j)}$

- **Generalized Additive Models (GAM)**
  - GAM models are made to directly estimate models of this form.

    $$\hat{\gamma}(x) = \hat{\alpha}_0 + \sum_{j=1}^{p} \hat{g}_j(x_j)$$

  - $g_j(x_j)$ is non-linear (usually based on penalized splines)
  - In **R**, the `mgcv` package is worth becoming familiar with to implement GAM.
  - See ESL 9.1 for more details

7 Bag of Words Classification

Before transformers and LLMs, Naive Bayes was popular for classifying text data. For example, classifying emails as spam vs. not spam. Here we will use naive bayes for classifying the author of a tweet.

Following Text Mining with R, Chapter 7, we will get the authors' (Julia Silge and David Robinson) tweets and combine them.

```
url = "https://github.com/dgrtwo/tidy-text-mining/raw/refs/heads/master/data"
julia = read_csv(file.path(url, "tweets_julia.csv"))
david = read_csv(file.path(url, "tweets_dave.csv"))
```

I'm adding the row number from the original data to keep track of everything. We could use `tweet_id`, but since its such a large number it doesn't print out nicely. Also removing all retweets (the `text` begins with `RT`). Replacing all url links with the word "URL".

```
tweets =
  bind_rows(
    julia %>% mutate(author = "julia", row = str_c("j:",row_number())),
    david %>% mutate(author = "david", row = str_c("d:",row_number()))
  ) %>%
  filter(!str_detect(text, "^RT")) %>% # remove all retweets
  # replace any url (http:// or https://) with "URL"
  mutate(
    text = str_replace_all(text,
                           pattern = "http.+? |http.+$",
                           replacement = "URL")
  ) %>%
  select(author, text, row)
```

```
tweets_sample = tweets %>% slice_sample(n=5)
tweets_sample
#> # A tibble: 5 x 3
#>   author text                                                          row
#>   <chr>  <chr>                                                         <chr>
#> 1 julia  All three children asleep! Now I have wine, a HUGE pile of laund~ j:64~
#> 2 julia  Happy 2nd birthday to my littlest! URL                        j:53~
#> 3 julia  Leaving for my ultrasound soon!  After months of being noncommit~ j:12~
#> 4 david  @_inundata For NYC, I'll be at PLOTCON in November and NYR in (p~ d:12~
#> 5 david  I don't think I'd be a good Green Lantern because the ring works~ d:20~
```

The next step is converting a text string into a set of features. The *bag of words* approach is to essentially dummy encode each word/token. The `tidytext` library has a convenience function `unnest_tokens()` to do this as well as converting all letters to lowercase.

```
library(tidytext)

#: Regular Expressions for determining a "token" in twitter data
unnest_reg = "([^A-Za-z_\\d#@']|'(?![A-Za-z_\\d#@]))"

#: unnested tokens
tweets_sample %>%
  unnest_tokens(token, text, token = "regex", pattern = unnest_reg)
#> # A tibble: 93 x 3
#>   author row    token
#>   <chr>  <chr>  <chr>
#> 1 julia  j:6445 all
#> 2 julia  j:6445 three
#> 3 julia  j:6445 children
#> 4 julia  j:6445 asleep
#> 5 julia  j:6445 now
```

```
#> 6 julia  j:6445 i
#> # i 87 more rows
```

Notice that some symbols, like commas and explanation marks are removed. This may not be a good idea for this task!

The most frequent tokens are commonly considered *stop words* that are removed prior to modelling. We are going to ignore.

```
tweets %>%
  unnest_tokens(token, text, token = "regex", pattern = unnest_reg) %>%
  count(token, sort=TRUE)
#> # A tibble: 17,274 x 2
#>   token     n
#>   <chr> <int>
#> 1 i      6906
#> 2 the    5573
#> 3 to     5188
#> 4 a      4669
#> 5 is     3701
#> 6 of     3569
#> # i 17,268 more rows
```

Also, there are many infrequent tokens and we can expect new tokens if we were to load new tweets.

```
tweets %>%
  unnest_tokens(token, text, token = "regex", pattern = unnest_reg) %>%
  count(token) %>% arrange(n)
#> # A tibble: 17,274 x 2
#>   token           n
#>   <chr>       <int>
#> 1 #1064           1
#> 2 #10yearsago     1
#> 3 #1989           1
#> 4 #2011           1
#> 5 #4              1
#> 6 #41             1
#> # i 17,268 more rows
```

There are many other NLP techniques to *clean* the text before modeling. We are going to ignore all of these for now. Our Naive Bayes may be able to do well despite, let's find out.

We can use the `cast_sparse()` function (from `tidytext`) to create the *sparse* model matrix. Each column corresponds to a token.

```
tweets %>%
  unnest_tokens(token, text, token = "regex", pattern = unnest_reg) %>%
  cast_sparse(row = row, column = token)
```

```
#>         all three children asleep now i have wine a huge pile of laundry
#> j:6445    1     1        1      1   1 1    1    1 1    1    1  1       1
#> j:5338    0     0        0      0   0 0    0    0 0    0    0  0       0
#> j:12861   0     0        0      0   0 0    1    0 1    0    0  1       0
#> d:1224    0     0        0      0   0 0    0    0 0    0    0  0       0
#> d:2007    0     0        0      0   0 1    0    0 1    0    0  0       0
```

Overall we want to use the bayes breakdown to estimate the probability that a given tweet was made by Julia (or David). For example,

$$\Pr(Y = \text{julia} \mid \text{“three huge children at \#jsm2016"})$$

Looking at our model matrix from `tweets_sample`, this would correspond to vector "three huge children at #jsm2016" $= [0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, \ldots]$.

In general, we will have p columns and want to estimate:

$$\Pr(Y = \text{julia} \mid X_1 = x_1, X_2 = x_2, \ldots, X_p = x_p) = \frac{\Pr(X_1 = x_1, X_2 = x_2, \ldots, X_p = x_p) \mid Y = \text{julia}) \Pr(Y = \text{julia})}{\Pr(X_1 = x_1, X_2 = x_2, \ldots, X_p = x_p)}$$

where each column is a binary $X_j \in \{0, 1\}$.

7.1 Naive Bayes

To further simply, naive bayes treats each column (in this case token) as independent from all others.

$$\Pr(X_1 = x_1, X_2 = x_2, \ldots, X_p = x_p \mid Y = \text{julia}) \approx \prod_{j=1}^{p} \Pr(X_j = x_j \mid Y = \text{julia})$$

The log density ratio becomes

$$\sum_{j=1}^{p} \log \left(\frac{\Pr(X_j = x_j \mid Y = \text{julia})}{\Pr(X_j = x_j \mid Y = \text{david})} \right)$$

For each column, we want to estimate $\Pr(X_j = x_j \mid Y = \text{julia})$ and $\Pr(X_j = x_j \mid Y = \text{david})$.

The *tidy format* (i.e., long) is convenient for estimating

```
# total number of tweets by author
tweet_counts =
  tweets %>%
  count(author, name = "n_total")
tweet_counts
#> # A tibble: 2 x 2
#>   author n_total
#>   <chr>    <int>
#> 1 david     3013
#> 2 julia    12051
```

```
# number of tweets containing token, by author
token_counts =
  tweets %>%
  unnest_tokens(token, text, token = "regex", pattern = unnest_reg) %>%
  count(author, token, name = "n1") %>%
  complete(author, token, fill = list(n1 = 0))

token_counts %>% slice_sample(n = 10)
#> # A tibble: 10 x 3
#>   author token            n1
#>   <chr>  <chr>         <int>
#> 1 julia  marinated          1
#> 2 julia  preregistered      1
#> 3 david  fuzzyjoin         10
#> 4 david  indenting          0
#> 5 david  betrayal           1
#> 6 david  'downton           0
#> # i 4 more rows
```

```
# p1 = Pr(X = 1 | Y=k); Estimated probability tweet by author contains token.
# p0 = Pr(X = 0 | Y=k); Estimated probability tweet by author not contain token.
token_probs =
  token_counts %>%
  left_join(tweet_counts, by = "author") %>%
  mutate(
    n0 = n_total - n1,
    # p1 = n1/n_total, # MLE will have high variance
    p1 = (n1 + .1) / (n_total + .2), # add shrinkage toward 1/2
    p0 = 1-p1
  ) %>%
  arrange(-p1)


token_probs %>% slice_sample(n=10)
#> # A tibble: 10 x 7
#>   author token        n1 n_total    n0        p1    p0
#>   <chr>  <chr>     <int>   <int> <int>     <dbl> <dbl>
#> 1 david  napa          0    3013  3013 0.0000332 1.00
#> 2 julia  haiti        10   12051 12041 0.000838  0.999
#> 3 david  streams       0    3013  3013 0.0000332 1.00
#> 4 julia  show         37   12051 12014 0.00308   0.997
#> 5 david  tagalongs     0    3013  3013 0.0000332 1.00
#> 6 david  jokes         2    3013  3011 0.000697  0.999
#> # i 4 more rows
```

From this, we can calculate the log density ratio (also known as weight of evidence):

```
log_density_ratio =
  token_probs %>%
  select(author, token, p1) %>%
  pivot_wider(names_from = author, values_from = p1) %>%
  mutate(
    ldr_1 = log(julia) - log(david),
    ldr_0 = log(1-julia) - log(1-david)
  )

# evidence from token in tweet favors julia
log_density_ratio %>%
  slice_max(ldr_1, n = 10)
#> # A tibble: 10 x 5
#>   token           julia     david ldr_1   ldr_0
#>   <chr>           <dbl>     <dbl> <dbl>   <dbl>
#> 1 @selkie1970    0.0473 0.0000332  7.26 -0.0484
#> 2 @skedman       0.0441 0.0000332  7.19 -0.0450
#> 3 @haleynburke   0.0248 0.0000332  6.62 -0.0251
#> 4 @doctormac     0.0226 0.0000332  6.52 -0.0228
#> 5 @katiekrongard 0.0208 0.0000332  6.44 -0.0210
#> 6 3yo            0.0167 0.0000332  6.22 -0.0168
#> # i 4 more rows

# evidence from token in tweet favors david
log_density_ratio %>%
  slice_min(ldr_1, n = 10)
#> # A tibble: 10 x 5
#>   token          julia   david ldr_1   ldr_0
#>   <chr>          <dbl>   <dbl> <dbl>   <dbl>
#> 1 @jtleek   0.00000830 0.0266  -8.07 0.0269
#> 2 @ucfagls  0.00000830 0.0183  -7.70 0.0184
```

```
#> 3 @jkgoya    0.00000830 0.0126  -7.33 0.0127
#> 4 #jsm2016  0.00000830 0.00999 -7.09 0.0100
#> 5 #user2016 0.00000830 0.00999 -7.09 0.0100
#> 6 dev       0.00000830 0.00899 -6.99 0.00903
#> # i 4 more rows

# evidence from *no* token in tweet favors julia
log_density_ratio %>%
  slice_max(ldr_0, n = 10)
#> # A tibble: 10 x 5
#>   token            julia  david  ldr_1  ldr_0
#>   <chr>            <dbl>  <dbl>  <dbl>  <dbl>
#> 1 url            0.153   0.285  -0.622 0.169
#> 2 @hadleywickham 0.00391 0.103  -3.27  0.105
#> 3 you            0.116   0.191  -0.496 0.0884
#> 4 @jennybryan    0.00640 0.0717 -2.42  0.0680
#> 5 @quominus      0.00565 0.0684 -2.49  0.0652
#> 6 data           0.0105  0.0707 -1.91  0.0628
#> # i 4 more rows

# evidence from *no* token in tweet favors david
log_density_ratio %>%
  slice_min(ldr_0, n = 10)
#> # A tibble: 10 x 5
#>   token julia  david ldr_1   ldr_0
#>   <chr> <dbl>  <dbl> <dbl>   <dbl>
#> 1 i     0.497 0.304  0.492 -0.325
#> 2 the   0.393 0.276  0.354 -0.177
#> 3 my    0.177 0.0883 0.694 -0.102
#> 4 to    0.358 0.289  0.213 -0.102
#> 5 and   0.230 0.150  0.430 -0.0992
#> 6 of    0.251 0.183  0.316 -0.0868
#> # i 4 more rows
```

```
# tweet to classify
test_tweet = tibble(text = "three huge children at #jsm2016") %>%
  unnest_tokens(token, text, token = "regex", pattern = unnest_reg) %>%
  pull(token)

# make evidence
evidence =
  log_density_ratio %>%
  mutate(
    X = ifelse(token %in% test_tweet, 1, 0),
    ldr = ifelse(X == 1, ldr_1, ldr_0)
  ) %>%
  arrange(-X, -abs(ldr))
evidence
#> # A tibble: 17,274 x 7
#>   token         julia     david  ldr_1    ldr_0     X    ldr
#>   <chr>         <dbl>     <dbl>  <dbl>    <dbl> <dbl>  <dbl>
#> 1 #jsm2016 0.00000830 0.00999   -7.09   0.0100      1 -7.09
#> 2 children 0.0167     0.0000332  6.22  -0.0168      1  6.22
#> 3 three    0.00366    0.00136    0.989 -0.00230     1  0.989
#> 4 huge     0.00432    0.00169    0.938 -0.00264     1  0.938
#> 5 at       0.0853     0.0571     0.401 -0.0304      1  0.401
#> 6 i        0.497      0.304      0.492 -0.325       0 -0.325
#> # i 17,268 more rows
```

```
evidence %>%
  group_by(X) %>%
  summarize(
    n = n(),
    ldr = sum(ldr)
  )
#> # A tibble: 2 x 3
#>       X     n    ldr
#>   <dbl> <int>  <dbl>
#> 1     0 17269 -0.386
#> 2     1     5  1.46
```

There are 5 tokens in the test tweet. Those 5 give supporting evidence that the tweet was made by *julia*. The 17269 tokens that were not in the test tweet weakly point towards *david* as the author. In total, the evidence favors *julia*

```
evidence %>%
  summarize(evidence = sum(ldr))
#> # A tibble: 1 x 1
#>   evidence
#>      <dbl>
#> 1     1.07
```

7.1.1 Last Steps

Julia has many more tweets than David.

```
tweet_counts %>%
  mutate(prior = n_total / sum(n_total) )
#> # A tibble: 2 x 3
#>   author n_total prior
#>   <chr>    <int> <dbl>
#> 1 david     3013 0.200
#> 2 julia    12051 0.800
```

```
prior_odds =
  tweets %>%
  summarize(
    n_julia = sum(author == "julia"),
    n_david = sum(author == "david"),
    p_julia = (n_julia + 1) / (n_julia + n_david + 2),
    prior_odds = p_julia / (1 - p_julia),
    log_prior_odds = log(p_julia) - log(1-p_julia)
  )

prior_odds
#> # A tibble: 1 x 5
#>   n_julia n_david p_julia prior_odds log_prior_odds
#>     <int>   <int>   <dbl>      <dbl>          <dbl>
#> 1   12051    3013   0.800       4.00           1.39
```

```
tibble(
  log_density_ratio = sum(evidence$ldr),
  log_prior_odds = prior_odds$log_prior_odds,
  log_odds = log_density_ratio + log_prior_odds,
  p_julia = exp(log_odds) / (1 + exp(log_odds)),
  p_david = 1-p_julia
)
#> # A tibble: 1 x 5
```

```
#>   log_density_ratio log_prior_odds log_odds p_julia p_david
#>               <dbl>          <dbl>    <dbl>   <dbl>   <dbl>
#> 1              1.07           1.39     2.46   0.921  0.0791
```