GMM - Gaussian Mixture Models

Tópicos Especiais 2 - Identificação de padrões

César A. Galvão Gabriela Carneiro João Vitor Vasconcelos
Kevyn Andrade de Souza

Contextualização

Classificação supervisionada

- Principal objetivo é montar uma regra de decisão para classificação, uma análise discriminante;

- Desejamos classificar uma observação $\mathbf{x}$ na classe C_k, $k = 1, \ldots, K$ a partir de uma probabilidade a posteriori $\Pr(C_k|\mathbf{x})$ para cada classe k.

Classificação supervisionada

Aprendendo as densidades condicionais das classes $f(\mathbf{x}|C_k)$ e das probabilidades a priori $\Pr(C_k)$ — na prática a frequência relativa de cada grupo — podemos calcular a probabilidade a posteriori $\Pr(C_k|\mathbf{x})$ a partir do teorema de Bayes:

$$\Pr(C_k|\mathbf{x}) = \frac{f(\mathbf{x}|C_k)\Pr(C_k)}{\sum_{g=1}^{K} f(\mathbf{x}|C_k)\Pr(C_k)} \tag{1}$$

Modelos com essa abordagem são chamados de *modelos generativos*.

Classificação supervisionada

Para construção do modelo, parte dos dados disponíveis devem ser usados para calibragem e o restante para teste de seu desempenho.

A validação cruzada evita *overfitting* avaliando o desempenho do modelo em dados não vistos anteriormente, simulando casos "reais".

Classificação baseada em GMM

Esse tipo de modelos assume que a densidade em cada classe segue uma mistura de gaussianas:

$$f(\mathbf{x}|C_k) = \sum_{g=1}^{G_k} \pi_{g,k}\, \phi(\mathbf{x}; \mu_{g,k}, \Sigma_{g,k}) \tag{2}$$

onde, dada uma classe C_k, com k fixo:

- $\pi_{g,k}$ é o "peso" de uma componente gaussiana g na classe k
- $\phi(\mathbf{x}; \mu_{g,k}, \Sigma_{g,k})$ é a densidade gaussiana multivariada com média $\mu_{g,k}$ e matriz de covariância $\Sigma_{g,k}$.

Classificação baseada em GMM

O modelo mais geral com base na Equation 1 é o MclustDA[1], que usa uma mistura finita de gaussianas para as classes, entre as quais o número de componentes e matrizes de covariâncias podem diferir.

As matrizes de covariância para a análise discriminante podem ser decompostas da forma a seguir:

$$\Sigma_k = \lambda_k \mathbf{U}_k \Delta_k \mathbf{U}_k^\top. \tag{3}$$

Classes de modelos com diferentes parametrizações foram vistas em aula (ex: EEE, VVV, etc).

[1] Fraley, C. & Raftery, A. E. Model-based clustering, discriminant analysis, and density estimation. *Journal of the American Statistical Association*; Jun 2002; 97, 458.

Classificação baseada em GMM

Utilizando a Equation 1 e a Equation 2, a classificação pode ser feita a partir de

$$\Pr(C_k|\mathbf{x}) = \frac{f(\mathbf{x}|C_k)\Pr(C_k)}{\sum_{g=1}^{K} f(\mathbf{x}|C_k)\Pr(C_k)} = \frac{\Pr(C_k)\sum_{g=1}^{G_k}\pi_{g,k}\,\phi(\mathbf{x};\mu_{g,k},\Sigma_{g,k})}{\sum_{g=1}^{K}\Pr(C_k)\sum_{k=1}^{G_k}\pi_{g,k}\,\phi(\mathbf{x};\mu_{g,k},\Sigma_{g,k})}.$$

Uma possível estratégia de classificação é escolher a classe C_k que maximiza a probabilidade a posteriori (MAP).

mclust::MclustDA()

O pacote `mclust` implementa, conforme o livro de referência, *Gaussian Mixture Modelling for Model-Based Clustering, Classification, and Density Estimation*[2].

A função `mclust::MclustDA()` implementa o modelo MclustDA (`mclust` *Discriminant Analysis*) com otimização feita via método EM.

[2] https://mclust-org.github.io/mclust/index.html

mclust::MclustDA()

```
MclustDA(data, class, G = NULL, modelNames = NULL,
         modelType = c("MclustDA", "EDDA"),
         prior = NULL,
         control = emControl(),
         initialization = NULL,
         warn = mclust.options("warn"),
         verbose = interactive(),
         ...)
```

mclust::MclustDA()

- `data`: dados de treinamento;
- `class`: vetor de classes;
- `G`: número de componentes da mistura. Default é `G = 1:5`;
- `modelNames`: nomes dos modelos a serem ajustados, ex: "EEE";
- `modelType`: tipo de modelo a ser ajustado, `MclustDA` ou `EDDA`. Default é o primeiro, `EDDA` ocorre quando `G = 1`;
- `prior`: vetor de probabilidades a priori para cada classe;
- `control`: parâmetros de controle do EM;

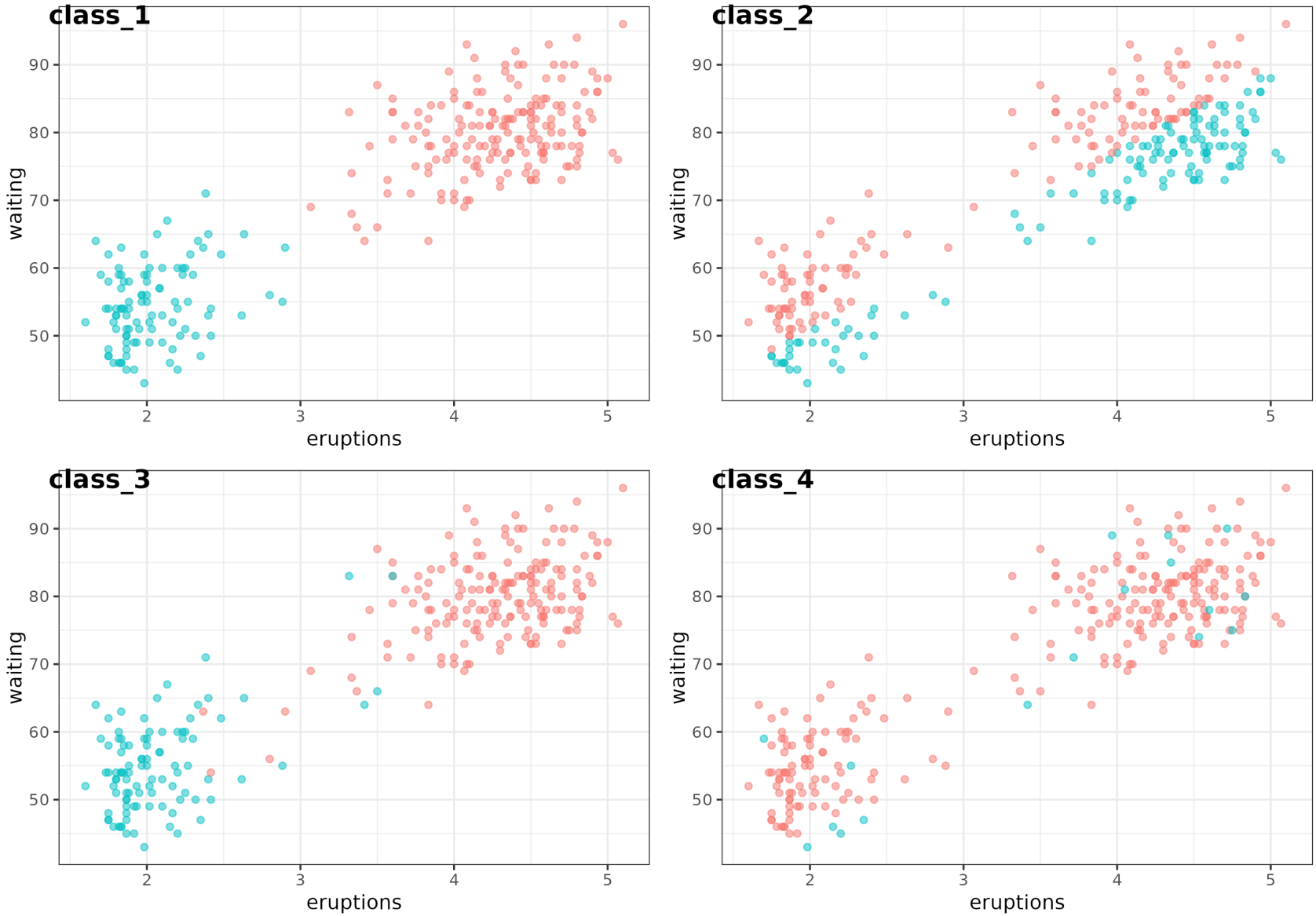
Exemplo de classificação

Dados utilizados

Será utilizada a base de dados `faithful` com classificações arbitrárias `class_1`, `class_2`, `class_3` e `class_4`, assim como a separação entre treino e teste (70/30) e uma variável `lincomb` que representa uma combinação linear entre `eruptions` e `waiting`.

```
Rows: 4
Columns: 8
$ eruptions <dbl> 3.600, 1.800, 3.333, 2.283
$ waiting   <dbl> 79, 54, 74, 62
$ class_3   <chr> "A", "B", "A", "B"
$ class_2   <chr> "A", "A", "A", "A"
$ class_1   <chr> "A", "B", "A", "B"
$ class_4   <chr> "A", "A", "A", "A"
$ lincomb   <dbl> 83.45402, 41.75409, 77.26612, 52.94598
$ treino    <dbl> 1, 0, 1, 0
```

Gráficos dos grupos



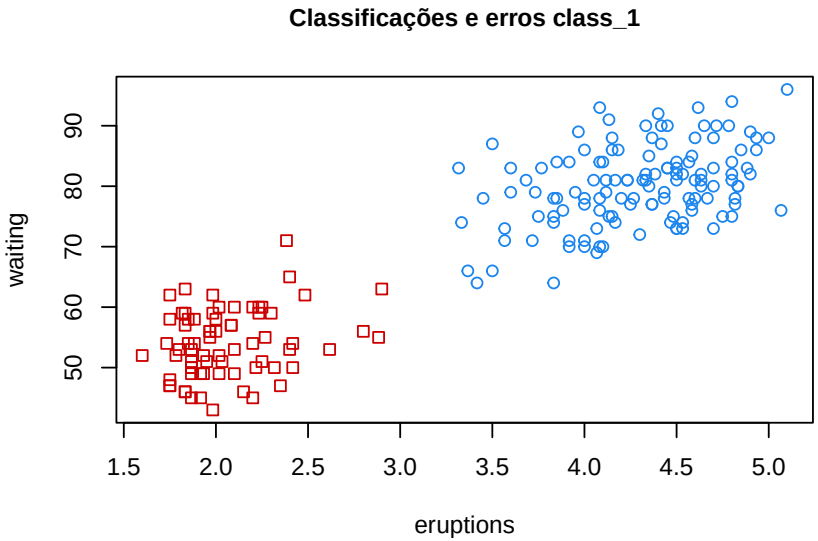
Ajuste do modelo

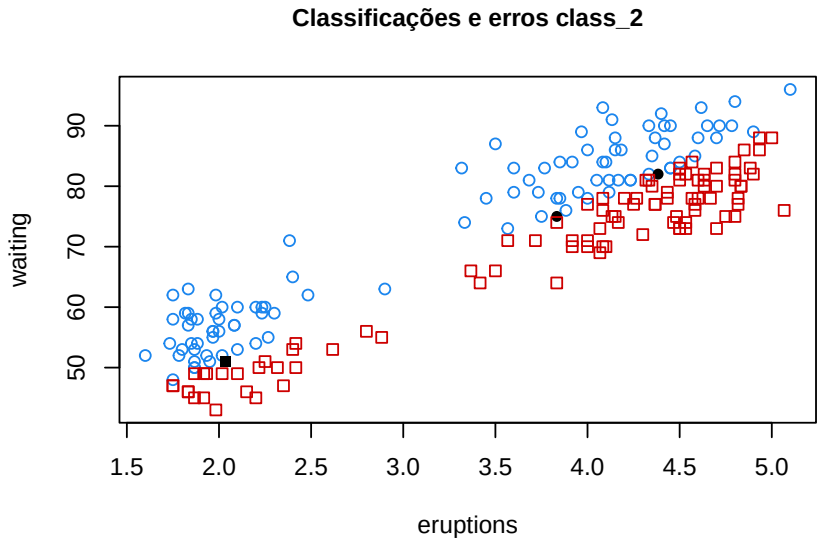
A seguir, um pedaço do código em que ajustamos o modelo MclustDA para as classificações `class_1`, `class_2` , `class_3` e `class_4` da base de treino:

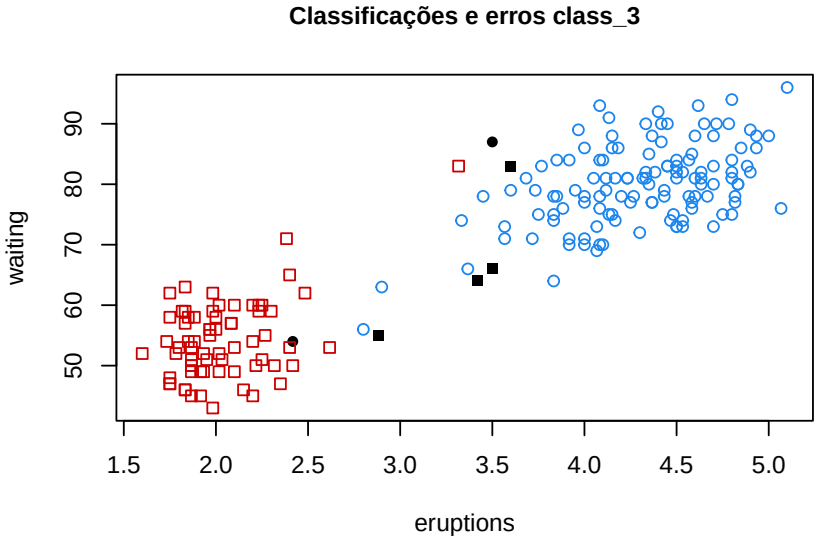
```
mod1 <- dados %>%
  dplyr::filter(treino == 1) %$%
  MclustDA(data = cbind(eruptions, waiting),
           class = class_1)

mod2 <- dados %>%
  dplyr::filter(treino == 1) %$%
  MclustDA(data = cbind(eruptions, waiting),
           class = class_2)

mod3 <- dados %>%
  dplyr::filter(treino == 1) %$%
  MclustDA(data = cbind(eruptions, waiting),
```

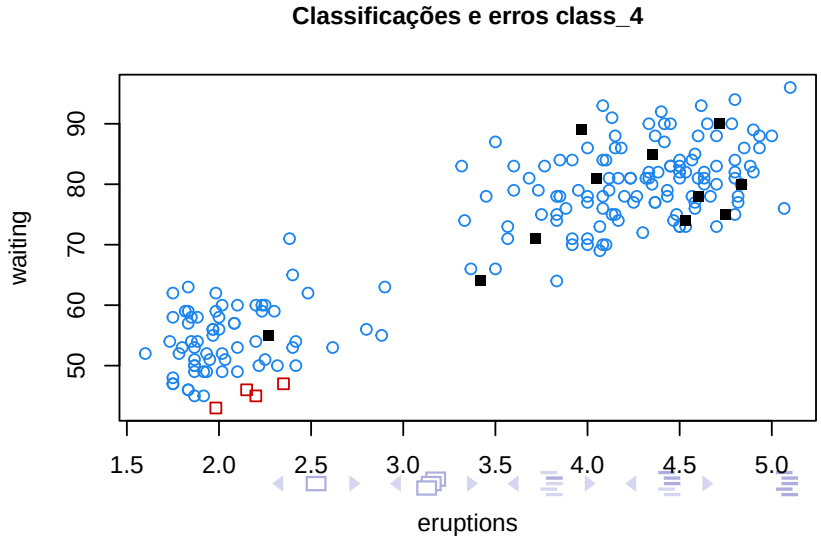
Resultados

Os gráficos a seguir são obtidos usando `plot(modelo, what = "error")`. Pontos pretos indicam erros de classificação.









Desempenho

A função `summary(modelo, newdata, newclass)` permite avaliar o desempenho do modelo em dados não vistos anteriormente, ou seja, de teste/validação.

Os resultados do código abaixo são exibidos no slide seguinte.

```
dados %>%
  dplyr::filter(treino == 0) %$%
  summary(mod1, newdata = tibble(eruptions, waiting),
          newclass = class_1)

dados %>%
  dplyr::filter(treino == 0) %$%
  summary(mod1, newdata = tibble(eruptions, waiting),
          newclass = class_2)
#etc
```

Desempenho

```
----------------------------------------------------
Gaussian finite mixture model for classification
----------------------------------------------------

MclustDA model summary:


Classes   n     % Model G
      A 124 64.92   XXX 1
      B  67 35.08   EEI 2

Training confusion matrix:
     Predicted
Class   A   B
    A 124   0
    B   0  67
Classification error = 0
Brier score          = 0

Test confusion matrix:
     Predicted
Class  A  B
    A 51  0
    B  0 30
Classification error = 0
Brier score          = 0.001
```

```
----------------------------------------------------
Gaussian finite mixture model for classification
----------------------------------------------------

MclustDA model summary:


Classes   n     % Model G
      A 124 64.92   XXX 1
      B  67 35.08   EEI 2

Training confusion matrix:
     Predicted
Class   A   B
    A 124   0
    B   0  67
Classification error = 0
Brier score          = 0

Test confusion matrix:
     Predicted
Class  A  B
    A 25 21
    B 26  9
Classification error = 0.5802
Brier score          = 0.5811
```

Desempenho

Para ambos os modelos não houve erro de classificação durante o treinamento e apresentaram os seguintes resultados:

- Classe A com densidade estimada por uma única componente gaussiana multivariada elipsoidal;
- Classe B com densidade estimada por duas componentes gaussianas EEI, ou seja, parâmetros *volume* e *shape* iguais e matriz de covariância equivalente à identidade.
- Desempenho do modelo 1 é superior ao modelo 2 em termos de erro de classificação dos dados de *teste*.

Avaliação da performance

Erro de classificação

Missclassification error rate é a proporção das predições erradas feitas pelo classificador:

$$\text{CE} = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}(y_i \neq \hat{y}_i),$$

onde y_i é a classe conhecida para a i-ésima observação e $\hat{y}_i$ é a classe predita e $\mathbb{I}$ é a função indicadora que assume valor 1 se $y_i \neq \hat{y}_i$ e 0 caso contrário.

Um bom classificador tem taxa de erro próxima a zero.

Brier score

É computada como o quadrado médio da diferença entre a probabilidade predita e a classe real:

$$\text{BS} = \frac{1}{2n} \sum_{i=1}^{n} \sum_{k=1}^{K} (C_{ik} - \hat{p}_{ik})^2,$$

onde n é o número de observações, K é o número de classes, $C_{ik} = 1$ se a observação i for da classe k e 0 caso contrário, e $\hat{p}_{ik}$ é a probabilidade predita da observação i ser da classe k. A constante $1/2$ garante que o score varie entre 0 e 1.

Um bom classificador tem score próximo a zero.

Validação cruzada

A validação cruzada é uma técnica indicada quando a base de dados não tem tamanho suficiente para garantir que a simples divisão entre conjuntos de treinamento e teste seja suficiente para garantir a generalização do modelo.

Um esquema V-fold de reamostragem utiliza V partições da base de dados, treinando o modelo em $V - 1$ partições e testando em uma. O erro de classificação é a média dos erros obtidos em cada partição.

Quando V é igual ao tamanho da base de dados, temos a técnica *leave-one-out*.

Valores grandes de V reduzem viés do estimador, mas aumentam sua variância. Valores pequenos de V fazem o inverso.

Validação cruzada

A função `cvMclustDA()` realiza a validação cruzada com os seguintes argumentos:

```
cvMclustDA(object, nfold = 10,
           prop = object$prop,
           verbose = interactive(),
           ...)
```

Validação cruzada

Table 1: Resultados da validação cruzada

(a) Erro de classificação

modelo	LOO	5-fold
class_1	0.0000	0.0102
class_2	0.0305	0.0305
class_3	0.0457	0.0406
class_4	0.0660	0.0711

(b) Brier score

modelo	LOO	5-fold
class_1	0.0003	0.0064
class_2	0.0386	0.0391
class_3	0.0304	0.0377
class_4	0.0670	0.0708

Classificação no caso univariado

O caso univariado tem as mesmas restrições dos casos multivariados, com a adição que os discriminantes por decomposição de autovalores (EDDA) só contam com duas parametrizações possíveis: Variâncias iguais entre as classes, nomeada `E`, e variâncias diferentes, caso nomeado como `V`.

```
mod4 <- dados %$% MclustDA(
  lincomb, class_3, modelType = "MclustDA"
)
```

Classificação no caso univariado

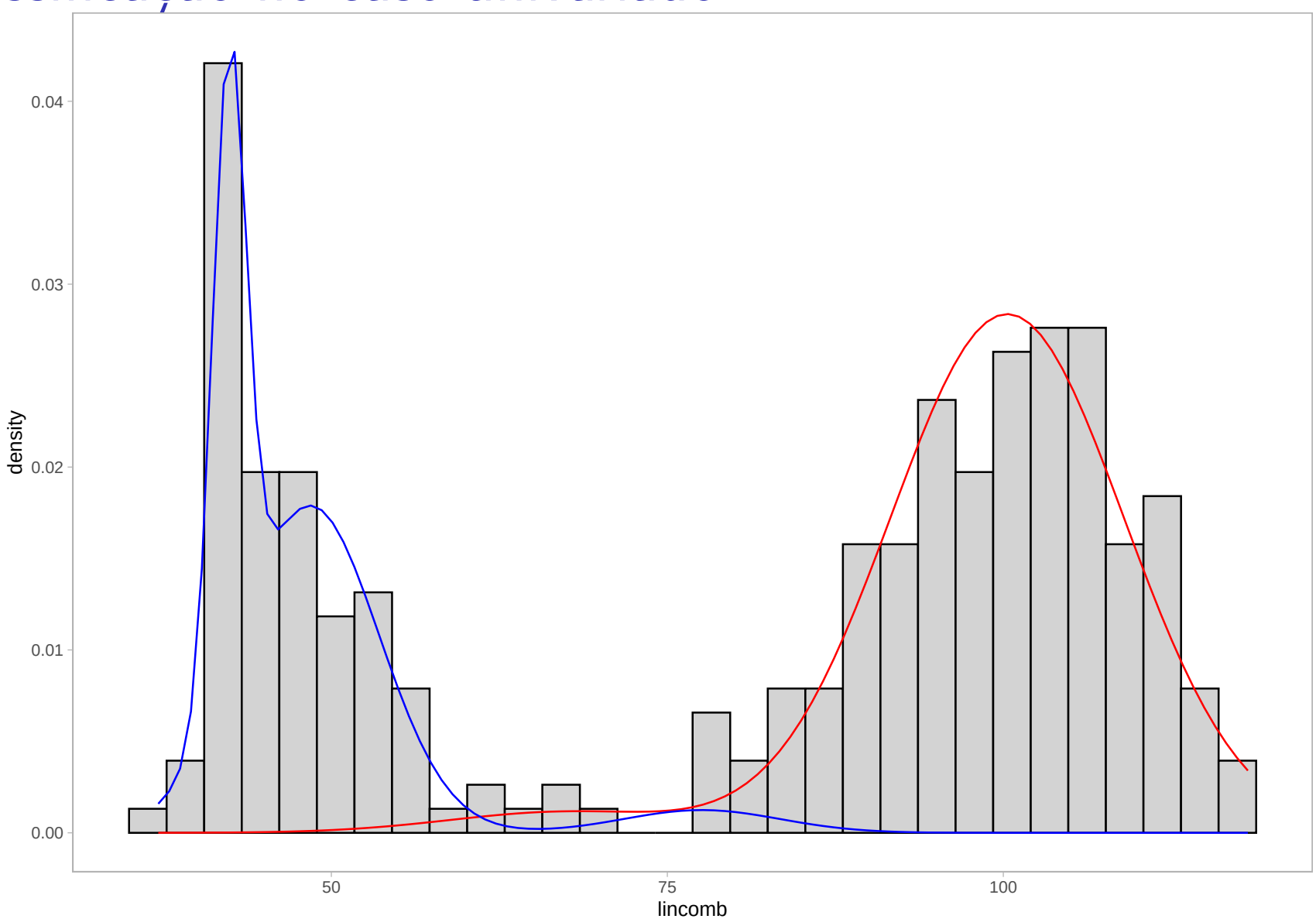


Figure 2: Classes separadas no caso univariado.

Custos diferentes para erros de classificação

Caso cada erro acarrete em custos diferentes, o objetivo se torna minimizar o custo total ao invés de minimizar o erro.

Seja $c(k|j)$ o custo de alocação de um elemento da classe j à classe k, com $k \neq j$, e $c(j|j) = 0$, para qualquer que seja o j. Seja $p(k|j) = Pr(C_k|\mathbf{x} \in C_j)$ a probabilidade de alocar um elemento da classe j na classe k. O custo total dos erros de classificação pode ser expresso por:

$$ECM = \sum_{j=1}^{K} ECM(j) = \sum_{j=1}^{K} \sum_{k \neq j}^{K} c(k|j)p(k|j)$$

Por simplicidade, é possível considerar que $c(k|j) = c(k)$.

Custos diferentes para erros de classificação

Utilizando a `class_3` e a tabela completa, a matriz de confusão de uma classificação que desconsidera os custos é dada por:

Table 2: Matriz de confusão com custos iguais

	A	B
A	172	3
B	5	92

Seja $c(B|A) = 25$ e $c(A|B) = 1$. Nessa configuração, a predição possui custo final de 80, tendo errado 8 observações.

Custos diferentes para erros de classificação

É possível fazer uma predição que considera os custos com o argumento `prop`, da função `predict`.

```
# proporções das classes * custo
pond <- mod3$prop * c(25, 1)
mod3_pred_custos <- predict(mod3,
                            prop = pond / sum(pond))
```

Custos diferentes para erros de classificação

A matriz de confusão assume a forma:

Table 3: Matriz de confusão com custos diferentes

	A	B
A	175	0
B	13	84

Nessa predição, o custo total cai de 80 para 13, mesmo que a quantidade de erros tenha aumentado para 13.

Classificação de classes desbalanceadas

Dado um conjunto de treino $D_{\text{treino}} = (x_1, y_1), \ldots, (x_n, y_n)$ e um novo banco de dados de teste $D_{\text{teste}} = x_1^*, \ldots, x_m^*$, a probababilidade de x_i^* pertencer a classe C_k é

$$\hat{z}_{ik}^* = \hat{\Pr}(C_k | x_i^*), \quad k = 1, \ldots, K.$$

Para lidar com o problema de classes desbalanceadas Saeres et. al (2002) propuseram um algoritmo para estimar as probabilidades condicionais posteriores de um classificador e assim tambem as probabilidades da classe a priori.

Classificação de classes desbalanceadas

Seja $\tilde{t}_k = \sum_{i=1}^{n} I(y_i \in C_k)/n$ a proporção de amostras da classe k nos dados de teste e $\tilde{t}_k^0 = \frac{\sum_{i=1}^{m} \hat{z_{ik}^*}}{m}$ a estimação preliminar das probilidades a priori da classe k. Começando com $s = 1$ e itere

$$\hat{z_{ik}}^{(s)} = \frac{\frac{\hat{t}_k^{(s-1)}}{\tilde{t}_k} \cdot \hat{z_{ik}}^*}{\sum_{g=1}^{K} \frac{\hat{t}_g^{(s-1)}}{\tilde{t}_k} \cdot \hat{z_{ig}}^*}$$

até que $(\hat{t}_1, \ldots, \hat{t}_K)$ estabilize. Tal processo pode ser feito no pacote `mclust` a partir da função `classPriorProbs()`

Classificação de classes desbalanceadas

Fazendo predições do modelo ajustado para a classe 4 e estimando os erros de classificação e o Brier escore.

```
pred <- dados_teste %$%
  predict(mod5, newdata = tibble(eruptions, waiting))

classError(pred$classification,
           dados_teste$class_4)$error
```

```
[1] 0.02666667
```

```
dados_teste %$% BrierScore(pred$z, class_4)
```

```
[1] 0.03744776
```

Classificação de classes desbalanceadas

Utilizando a função `classPriorProbs()`

```
prioprob <- dados_teste %$%
  classPriorProbs(mod5, cbind(eruptions, waiting))
prioprob
```

```
           A            B
9.999998e-01 2.018179e-07
```

Classificação de classes desbalanceadas

```
pred1 <- dados_teste %$%
  predict(mod1,
          newdata=cbind(eruptions,waiting),
          prop = prioprob)

classError(pred1$classification,
           dados_teste$class_4)$error
```

```
[1] 0.2666667
```

```
dados_teste %$% BrierScore(pred1$z,class_4)
```

```
[1] 0.2364875
```

- O brier escore diminiu, o que indica que o ajuste melhorou.
- O erro de classificação continua o mesmo.