

Rafael Izbicki

Departamento de Estatística - UFSCar

Tiago Mendonça dos Santos

Insper Instituto de Ensino e Pesquisa

Machine Learning sob a ótica estatística

Uma abordagem preditivista para a estatística com
exemplos em R

Atenção: versão preliminar em preparação

Comentários, sugestões e correções:
rafaelizbicki@gmail.com

24 de janeiro de 2019

Prefácio

Esta é uma versão em desenvolvimento, por favor me envie sugestões, críticas e erros encontrados.

Recomendo que a leitura do Capítulo 1 seja feita antes dos demais capítulos, pois ele apresenta a nomenclatura e principais paradigmas abordados no restante do livro.

Este livro está dividido da seguinte maneira:

- Partes I e II: Aprendizado supervisionado (regressão e classificação, respectivamente).
- Parte III: Aprendizado não supervisionado (métodos de redução de dimensionalidade e análise de agrupamento)
- Parte IV: Sistemas de recomendação
- Parte V: Apêndice

Agradecimentos. Grato pelas sugestões de Lucas Leite Cavalaro, Luiz Gabriel Fernandes Cotrim, Victor Vinicius Fernandes, João Carlos Poloniato Ferreira, Lucas Pereira Lopes, Paula Ianishi, Marco Henrique de Almeida Inacio, Juliana Maia, Taís Roberta Ribeiro, Ana Paula Jorge do Espírito Santo, Marcia Maria Barbosa da Silva, Gilson Shimizu, João Flávio Andrade Silva, Rafael Bassi Stern, Afonso Fernandes Vaz.

Sumário

Parte I Regressão

1	Introdução	3
1.1	Predição versus Inferência: Por que estimar $r(\mathbf{x})$?	4
1.2	As Duas Culturas	8
1.3	A Função de Risco	9
1.4	Seleção de Modelos: Super e Sub-Ajuste	11
1.4.1	Data Splitting e Validação Cruzada	12
1.4.2	Balanço entre Viés e Variância	22
1.5	Tuning Parameters	23
1.6	Resumo	24
2	Métodos Paramétricos	25
2.1	O Método dos Mínimos Quadrados	25
2.1.1	Mínimos Quadrados quando a Suposição de Linearidade Falha	26
2.1.2	Regressão Linear no R	27
2.2	Resumo	28
3	Métodos Paramétricos em Dimensões Altas	29
3.1	Busca pelo Melhor Subconjunto de Covariáveis	29
3.2	Regressão Stepwise	31
3.3	Lasso	32
3.3.1	Garantias Teóricas	34
3.4	Regressão Ridge	35
3.5	Formulação Alternativa	36
3.6	Interpretação Bayesiana	36
3.7	Regressão Ridge e Lasso no R	37

3.8 Exemplos	38
3.9 Resumo	46
4 Métodos Não Paramétricos	49
4.1 Séries Ortogonais	49
4.2 Splines	52
4.3 Método dos k Vizinhos Mais Próximos	52
4.4 Nadaraya-Watson	54
4.5 Regressão Polinomial Local	56
4.6 Penalização em RKHSs	57
4.6.1 Penalização em Reproducing Kernel Hilbert Spaces (RKHS)	59
4.6.2 Solução	60
4.6.3 Exemplo 1: Kernel Ridge Regression	62
4.6.4 Exemplo 2: Support Vector Regression Machines	66
4.7 Modelos Aditivos	67
4.8 Árvores de Regressão	69
4.9 Bagging e Florestas Aleatórias	74
4.9.1 Florestas Aleatórias	76
4.10 Boosting	77
4.11 Redes Neurais Artificiais	78
4.11.1 Estimação: Backpropagation	81
4.11.2 Deep Learning	82
4.12 Um Pouco de Teoria	82
4.12.1 k -vizinhos Mais Próximos	82
4.12.2 Séries Ortogonais	84
4.13 Exemplos	86
4.13.1 Esperança de Vida e PIB per Capita	86
4.13.2 Exemplo em Duas Dimensões	88
4.14 Resumo	90
5 Métodos Não Paramétricos em Dimensões Altas	93
5.1 Taxas de convergência e a maldição da dimensionalidade	93
5.1.1 Esparsidade	95
5.1.2 Redundância	96
5.2 k Vizinhos Mais Próximos e Regressão Linear Local	96
5.3 Support Vector Regression	96
5.4 Séries	97
5.4.1 Bases Espectrais	97
5.4.2 O Estimador	99

5.5	Florestas Aleatórias	100
5.6	SpAM - Modelos Aditivos Esparsos	100
5.7	Exemplos	103
5.7.1	Isomap face data	103
5.8	Resumo	104
6	Outros Aspectos de Regressão	105
6.1	Interpretabilidade	105
6.2	Individual Sequence Predictions	106
6.3	Estimação de Densidades	107
6.4	Estimação de Densidades Condicionais	107
Parte II Classificação		
7	Introdução	113
7.1	Função de Risco	114
7.2	Estimação do Risco e Seleção de Modelos	115
7.3	Balanço entre Viés e Variância	117
7.4	Outras medidas de performance	117
8	Métodos de classificação	119
8.1	Classificadores Plug-in	119
8.1.1	Métodos de regressão	119
8.1.2	Regressão logística	120
8.1.3	Bayes Ingênuo	121
8.1.4	Análise Discriminante	123
8.2	Support Vector Machines (SVM)	128
8.3	Árvores de Classificação	131
8.4	Bagging e Florestas Aleatórias	134
8.5	Boosting	134
8.6	Método dos k Vizinhos Mais Próximos	135
8.7	Redes Neurais Artificiais	135
8.8	Exemplos	137
9	Outros Aspectos de Classificação	145
9.1	Assimetria na Função de Perda e Conjuntos de Dados Desbalanceados	145
9.2	Dataset Shift e Viés de Seleção	147
9.2.1	Covariate Shift	148
9.2.2	Prior Shift	151
9.3	Combinando classificadores	152

9.4 Teoria do Aprendizado Estatístico	155
9.4.1 Prova do teorema VC	159

Parte III Aprendizado não supervisionado

10 Redução de Dimensionalidade	167
10.1 Componentes Principais (PCA)	168
10.1.1 Interpretação alternativa: escalonamento multidimensional	170
10.1.2 Aplicação: compressão de imagens	171
10.2 Kernel PCA (KPCA)	175
10.3 Principal Curves	177
10.4 t-SNE	177
10.5 Projeções Aleatórias	177
10.6 Autoencoders	179
10.7 Quantos componentes utilizar?	179

11 Análise de Agrupamento	181
11.1 K-Médias	182
11.2 Métodos Hierárquicos	183
11.3 Análise de Agrupamento Espectral	186
11.4 Clustering com base em modas	186

Parte IV Sistemas de Recomendação

12 Regras de Associação	189
13 Sistemas de Recomendação	195
13.1 Filtro colaborativo baseado no usuário	196
13.2 Filtro colaborativo baseado no produto	197
13.3 FunkSVD	198
13.4 Seleção de Modelos	198

Parte V Apêndice

A Apêndice	205
A.1 Imagens	205
A.1.1 Lendo Imagens no R	207
A.2 Textos	208
A.3 Representação de Matrizes Esparsas	209
A.3.1 Word2vec	210

Sumário	xii
Referências	210
Índice Remissivo	215

Parte I

Régressão

Capítulo 1

Introdução

Métodos de regressão surgiram há mais de dois séculos com [Legendre \(1805\)](#) e [Gauss \(1809\)](#), que exploraram o método dos mínimos quadrados com o objetivo de prever órbitas ao redor do Sol. Hoje em dia, o problema de estimação de uma função de regressão possui papel central em estatística.

Apesar das primeiras técnicas para solucionar tal problema datarem de ao menos 200 anos, os avanços computacionais recentes permitiram que novas metodologias fossem exploradas. Em particular, com a capacidade cada vez maior de armazenamento de dados, métodos que fazem menos suposições sobre o verdadeiro estado da natureza ganham cada vez mais espaço. Com isso, vários desafios surgiram: por exemplo, métodos tradicionais não são capazes de lidar de forma satisfatória com bancos de dados em que há mais covariáveis que observações, uma situação muito comum nos dias de hoje. Similarmente, são frequentes as aplicações em que cada observação consiste em uma imagem ou um documento de texto, objetos complexos que levam a análises que requerem metodologias mais elaboradas. Aqui apresentamos diversos avanços recentes na área de regressão sob uma ótica preditivista, assim como uma revisão de modelos tradicionais sob esta mesma ótica.

De modo geral, o objetivo de um modelo de regressão é determinar a relação entre uma variável aleatória $Y \in \mathbb{R}$ e um vetor $\mathbf{x} = (x_1, \dots, x_d) \in \mathbb{R}^d$. Mais especificamente, busca-se estimar a função de regressão

$$r(\mathbf{x}) := \mathbb{E}[Y | \mathbf{X} = \mathbf{x}]$$

como forma de descrever tal relação. Note que Y é uma variável quantitativa em um problema de *regressão* (Parte I do livro); quando Y é qualitativa temos um problema de *classificação*, veja a Parte II deste livro.

A variável Y frequentemente recebe o nome de variável resposta, variável dependente ou rótulo (*label* em inglês). Já $\mathbf{x} = (x_1, \dots, x_d)$ são em geral chamadas de variáveis

explicativas, variáveis independentes, características (*features* em inglês), preditores ou covariáveis.

O objetivo da Parte I deste livro é descrever algumas técnicas para estimar (ou *treinar*, como dito na literatura de aprendizado de máquina) $r(\mathbf{x})$.

Atenção: Vamos assumir em todo o livro, a menos que seja dito o contrário, que a estimativa será feita com base em uma amostra i.i.d. $(\mathbf{X}_1, Y_1), \dots, (\mathbf{X}_n, Y_n) \sim (\mathbf{X}, Y)$.

Observação 1.1 Denotamos por $x_{i,j}$ o valor da j -ésima covariável na i -ésima amostra, ou seja, $\mathbf{x}_i = (x_{i,1}, \dots, x_{i,d})$, $i = 1, \dots, n$; veja a notação utilizada na Tabela 1.1.

□

Tabela 1.1: Notação utilizada no livro para as variáveis envolvidas em um problema de regressão.

Resposta Covariáveis	
Y_1	$X_{1,1} \dots X_{1,d}$ ($= \mathbf{X}_1$)
\vdots	$\vdots \quad \ddots \quad \vdots$
Y_n	$X_{n,1} \dots X_{n,d}$ ($= \mathbf{X}_n$)

1.1 Predição versus Inferência: Por que estimar $r(\mathbf{x})$?

A seguir veremos alguns exemplos reais nos quais a estimativa de $r(\mathbf{x})$ possui papel central.

Exemplo 1.1 [Esperança de vida e PIB per Capita] A Figura 1.1 mostra o PIB per Capita e Esperança de vida em 211 países em 2012¹. Uma possível pergunta de interesse é como podemos usar estes dados para estabelecer uma relação entre essas duas variáveis. Tal relação pode ser utilizada para estimar a esperança de vida de países cujo PIB per Capita é conhecido, mas a esperança não o é. Para tanto, pode-se estimar $\mathbb{E}[Y|\mathbf{x}]$, em que Y é a esperança de vida em um dado país, e \mathbf{x} é seu PIB per Capita. Uma estimativa de $\mathbb{E}[Y|\mathbf{x}]$ também pode ser utilizada para testar a hipótese de que não há relação entre essas duas variáveis.

¹ Dados obtidos em <http://data.worldbank.org/>

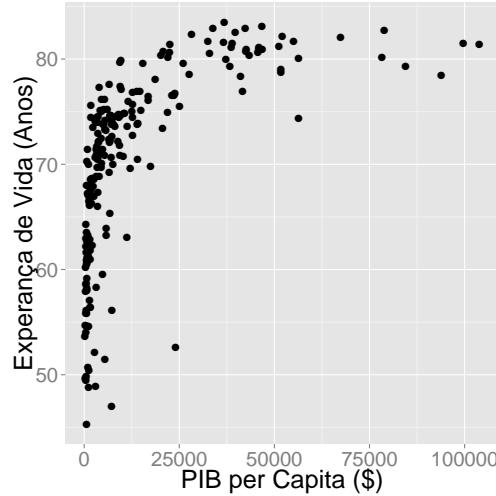


Figura 1.1: PIB per Capita e Esperança de vida de 211 países.

□

Exemplo 1.2 [Distância de uma galáxia até a Terra] Em cosmologia, uma variável de extrema importância é o *desvio para o vermelho* (*redshift*, em inglês) de uma galáxia, que quantifica o quanto longe tal objeto encontra-se da Terra. A *espectroscopia* permite que o desvio para o vermelho seja determinado com grande acurácia, contudo ela é extremamente cara e demanda muito tempo para ser feita. Assim, um grande interesse na atualidade é como estimar tal quantidade usando-se apenas imagens das galáxias ao invés da espectroscopia. Para tanto, pode-se usar uma amostra $(\mathbf{X}_1, Y_1), \dots, (\mathbf{X}_n, Y_n)$, em que \mathbf{X}_i é a imagem da i -ésima galáxia e Y_i seu respectivo desvio para o vermelho obtido via espectroscopia, para estimar a função de regressão $r(\mathbf{x})$ para predizer as respostas em imagens de novas galáxias cuja distância até a Terra é desconhecida. Veja o Apêndice [A.1](#) para uma breve exposição sobre imagens.

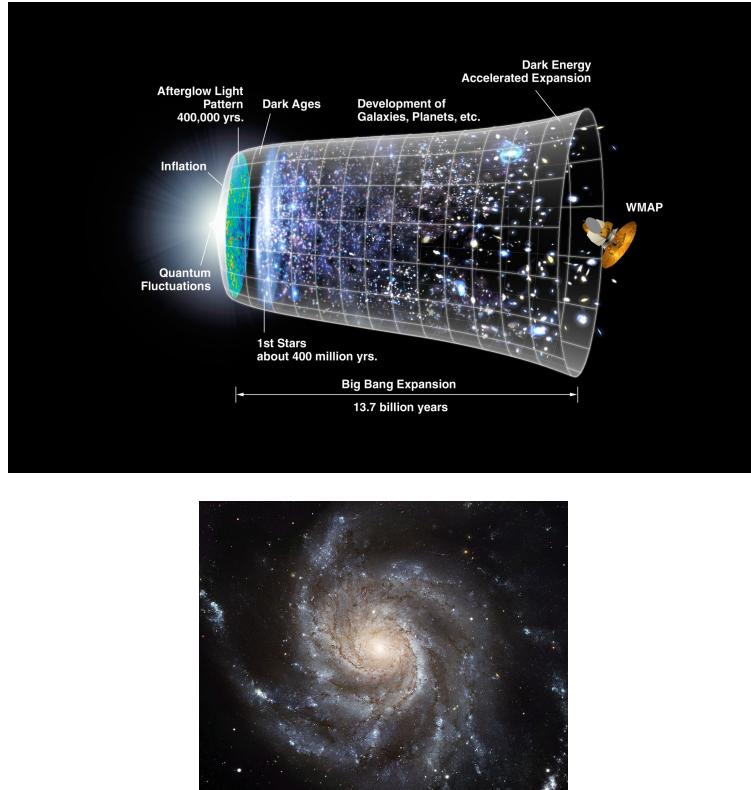


Figura 1.2: Acima: representação gráfica do universo e sua expansão. Embaixo: imagem de uma galáxia (crédito das imagens: ESA/Hubble).

□

Exemplo 1.3 [Isomap face data] Neste conjunto de dados, proveniente de [Tenenbaum et al. \(2000\)](#), o objetivo é estimar a direção em que uma pessoa está olhando (a resposta y) com base em uma imagem desta pessoa (covariáveis \mathbf{x}). Uma forma de se fazer isso é estimando a função de regressão $r(\mathbf{x})$ com base em uma amostra. Uma vez ajustada, $r(\mathbf{x})$ pode ser utilizada para prever a direção em novas imagens \mathbf{x} .

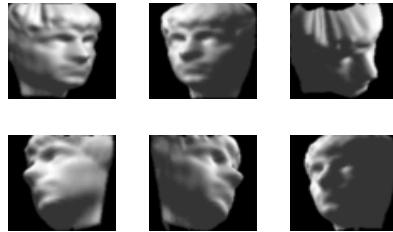


Figura 1.3: *Isomap face data*: cada amostra consiste na imagem de um indivíduo (\mathbf{x}) juntamente com qual direção ele está olhando (y). Nosso objetivo é estimar $r(\mathbf{x})$ para predizer a direção em novas imagens \mathbf{x} .

□

Exemplo 1.4 [Predição do ano de lançamento de músicas] Os dados do Year-PredictionMSD² contém informações sobre diversas covariáveis sobre certas músicas do banco *Million Song Dataset* (e.g., informações sobre timbre, o “quão energéticas” são cada uma das músicas, o “quão dançáveis” elas são etc), assim como o ano de lançamento de cada uma delas. Com esse banco de dados, é possível utilizar uma estimativa de $r(\mathbf{x})$ (em que \mathbf{x} são as covariáveis medidas, e Y é o ano de lançamento de uma dada música) para (i) predizer o ano de lançamento de músicas com base apenas nas covariáveis e (ii) entender que covariáveis estão relacionadas ao ano de lançamento, e como se dá esta relação (por exemplo, nos anos 70 as músicas eram mais “dançáveis”?).

□

Grosso modo, os objetivos destes exemplos podem ser divididos em duas classes:

- **Objetivo inferencial:** Quais preditores são importantes? Qual a relação entre cada preditor e a variável resposta? Qual o efeito da mudança de valor de um dos preditores na variável resposta?
- **Objetivo preditivo:** Como podemos criar uma função

$$g : \mathbb{R}^d \longrightarrow \mathbb{R}$$

que tenha bom poder preditivo? Isto é, como criar g tal que, dadas *novas* observações i.i.d. $(\mathbf{X}_{n+1}, Y_{n+1}), \dots, (\mathbf{X}_{n+m}, Y_{n+m})$, tenhamos

$$g(\mathbf{x}_{n+1}) \approx y_{n+1}, \dots, g(\mathbf{x}_{n+m}) \approx y_{n+m}?$$

² <https://archive.ics.uci.edu/ml/datasets/YearPredictionMSD>.

Note que enquanto que em alguns dos problemas o objetivo é claramente inferencial ou preditivo, em outros temos uma mistura de ambos. Por exemplo, no Exemplo 1.3 temos um problema claramente preditivo: a relação entre cada pixel da imagem e a resposta em geral não é de interesse imediato. Já no Exemplo 1.4 temos um objetivo misto: desejamos criar uma função para descobrir o ano de lançamento de músicas com base em suas covariáveis e ao mesmo tempo também queremos entender como o perfil de músicas mudou ao longo dos anos.

1.2 As Duas Culturas

Breiman (2001) argumenta que existem duas culturas no uso de modelos estatísticos (em especial modelos de regressão). Grosso modo, a primeira cultura, chamada de *data modeling culture* por Breiman, é a que domina a comunidade estatística. Em geral, nela se assume que o modelo utilizado para $r(\mathbf{x})$ – por exemplo, $r(\mathbf{x}) = \beta_0 + \sum_{i=1}^d \beta_i x_i$ – é correto. Isso ocorre pois o principal objetivo está na interpretação dos parâmetros envolvidos no modelo (neste caso, β_i 's); em particular há interesse em testes de hipóteses e intervalos de confiança para eles. Sob esta abordagem, testar se as suposições do modelo (por exemplo, normalidade dos erros, linearidade, homocedasticidade etc) são válidas é de fundamental importância. Ainda que predição muitas vezes faça parte dos objetivos, o foco em geral está na inferência.

A segunda cultura, chamada de *algorithmic modeling culture* por Breiman, é a que domina a comunidade de aprendizado de máquina (*machine learning*). Neste meio, o principal objetivo é a predição de novas observações. Não se assume que o modelo utilizado para os dados é correto; o modelo é utilizado apenas para criar bons algoritmos para prever bem novas observações. Muitas vezes não há nenhum modelo probabilístico explícito por trás dos algoritmos utilizados.

Observação 1.2 Ainda que não seja tradicional na literatura de estatística, nem todo método inferencial assume que o modelo utilizado é correto (veja a Seção 6.1 para algumas abordagens). Assim, mesmo na comunidade de *machine learning*, os modelos ajustados muitas vezes são utilizados para se fazer inferência, ainda que sob uma perspectiva diferente daquela usada na comunidade estatística.

□

O foco deste livro está na segunda abordagem. Buscaremos estimar $r(\mathbf{x})$ sem assumir que nossos modelos englobam a verdadeira função de regressão.

Apesar desta divisão de culturas, Breiman foi um estatístico que fez um grande trabalho para unir a área de estatística com aprendizado de máquina. Acreditamos que tal união é mutuamente benéfica. Assim, esse livro segue esta ideologia.

1.3 A Função de Risco

O primeiro passo para construir boas funções de predição é criar um critério para medir a performance de uma dada função de predição $g : \mathbb{R}^d \rightarrow \mathbb{R}$. Aqui, isso será feito através de seu risco quadrático, embora essa não seja a única opção:

$$R_{pred}(g) = \mathbb{E} [(Y - g(\mathbf{X}))^2],$$

em que (\mathbf{X}, Y) é uma nova observação que não foi usada para estimar g . Quanto menor o risco, melhor é a função de predição g .

Observação 1.3 A função $L(g; (\mathbf{X}, Y)) = (Y - g(\mathbf{X}))^2$ é chamada de *função de perda quadrática*. Outras funções de perda podem ser utilizadas, como por exemplo a *função de perda absoluta*; $L(g; (\mathbf{X}, Y)) = |Y - g(\mathbf{X})|$. Em geral, o risco é definido como a esperança de uma função de perda.

□

Quando medimos a performance de um estimador com base em seu risco quadrático, criar uma boa função de predição $g : \mathbb{R}^d \rightarrow \mathbb{R}$ equivale a encontrar um bom estimador para a função de regressão $r(\mathbf{x})$. Isto é, responder à pergunta do porquê estimar a função de regressão é, nesse sentido, o melhor caminho para se criar uma função para predizer novas observações Y com base em covariáveis observadas \mathbf{x} . Isto é mostrado no seguinte teorema:

Teorema 1.1 Suponha que definimos o risco de uma função de predição $g : \mathbb{R}^d \rightarrow \mathbb{R}$ via perda quadrática: $R_{pred}(g) = \mathbb{E} [(Y - g(\mathbf{X}))^2]$, em que (\mathbf{X}, Y) é uma nova observação que não foi usada para estimar g . Suponhamos também que medimos o risco de um estimador da função de regressão via perda quadrática: $R_{reg}(g) = \mathbb{E} [(r(\mathbf{X}) - g(\mathbf{X}))^2]$. Então

$$R_{pred}(g) = R_{reg}(g) + \mathbb{E}[\mathbb{V}[Y|\mathbf{X}]].$$

Demonstração.

$$\begin{aligned} \mathbb{E} [(Y - g(\mathbf{X}))^2] &= \mathbb{E} [(Y - r(\mathbf{X}) + r(\mathbf{X}) - g(\mathbf{X}))^2] \\ &= \mathbb{E} [(r(\mathbf{X}) - g(\mathbf{X}))^2] + \mathbb{E}[(Y - r(\mathbf{X}))^2] + 2\mathbb{E}[(r(\mathbf{X}) - g(\mathbf{X}))(Y - r(\mathbf{X}))] \\ &= \mathbb{E} [(r(\mathbf{X}) - g(\mathbf{X}))^2] + \mathbb{E}[\mathbb{V}[Y|\mathbf{X}]] \end{aligned}$$

□

Assim, visto que o termo $\mathbb{E}[\mathbb{V}[Y|\mathbf{X}]]$ ³ não depende de $g(\mathbf{x})$, estimar bem a função de regressão $r(\mathbf{x})$ é fundamental para se criar uma boa função de predição sob a ótica do

³ Para uma interpretação deste termo, veja a Seção 1.4.2.

risco quadrático, já que a melhor função de predição para Y é a função de regressão $r(\mathbf{x})$:

$$\arg \min_g R_{pred}(g) = \arg \min_g R_{reg}(g) = r(\mathbf{x}).$$

Observação 1.4 A função g no Teorema 1.1 pode ser entendida tanto como sendo fixa (neste caso $R(g)$ é chamado de *risko conditional*) quanto como sendo aleatória (função dos dados). Neste último caso, a esperança é tomada com relação à observação (\mathbf{X}, Y) e à amostra $(\mathbf{X}_1, Y_1), \dots, (\mathbf{X}_n, Y_n)$, e o risco é chamado de *risko esperado*. Em outras palavras, o risco esperado é a esperança do risco condicional sob todas as possíveis amostras usadas para criar g . Assim, ele pode ser entendido como uma garantia sobre o processo de criação de g ao invés de uma garantia sobre a particular função g criada para um dado problema. Por exemplo, se o risco esperado de uma regressão linear vale C em um dado problema, isso implica que, em média, uma regressão linear para dados que seguem a mesma distribuição desse conjunto e com o mesmo tamanho amostral possui risco condicional C . Dependendo do contexto, utilizaremos uma ou outra definição.

□

Daqui em diante, denotaremos por R o risco preditivo R_{pred} .

Observação 1.5 Em um contexto de predição, a definição de risco condicional (i.e., considerando g fixo; veja a Observação 1.4) possui grande apelo frequentista. Digamos que observamos um novo conjunto, $(\mathbf{X}_{n+1}, Y_{n+1}), \dots, (\mathbf{X}_{n+m}, Y_{n+m})$, i.i.d à amostra observada. Então, pela lei dos grandes números, sabemos que, se m é grande,

$$\frac{1}{m} \sum_{i=1}^m (Y_{n+i} - g(\mathbf{X}_{n+i}))^2 \approx R(g) := \mathbb{E} [(Y - g(\mathbf{X}))^2].$$

Em outras palavras, se $R(g)$ possui valor *baixo*, então

$$g(\mathbf{x}_{n+1}) \approx y_{n+1}, \dots, g(\mathbf{x}_{n+m}) \approx y_{n+m},$$

e portanto teremos boas predições em novas observações. Note que essa conclusão vale qualquer que seja a função de perda L utilizada para definir o risco, uma vez que

$$\frac{1}{m} \sum_{i=1}^m L(g, (\mathbf{X}_{n+i}, Y_{n+i})) \approx R(g) := \mathbb{E} [L(g; (\mathbf{X}, Y))].$$

□

O objetivo dos métodos de regressão sob a ótica preditivista é fornecer, em diversos contextos, métodos que costumam produzir bons estimadores de $r(\mathbf{x})$, isto é, estimadores que possuam risco baixo.

1.4 Seleção de Modelos: Super e Sub-Ajuste

Em problemas reais, é comum ajustar vários modelos para a função de regressão $r(\mathbf{x})$ e buscar qual deles possui maior poder preditivo, isto é, qual possui menor risco. Isto é exemplificado no exemplo a seguir.

Exemplo 1.5 [Esperança de vida e PIB per Capita] Revisitamos aqui o Exemplo 1.1. A Figura 1.4 mostra o ajuste de três modelos distintos para $r(\mathbf{x})$:

$$r(x) = \beta_0 + \sum_{i=1}^p \beta_i x^i, \text{ para } p \in \{1, 4, 50\}.$$

Em outras palavras, ajustamos três regressões polinomiais: uma linear, uma de 4° e uma de 50° grau. Os ajustes foram feitos (i.e., os coeficientes β_i foram estimados) por meio do Método dos Mínimos Quadrados (veja Capítulo 2 para uma revisão). Observa-se que, enquanto o modelo linear é simplista demais para os dados, o modelo com $p = 50$ polinômios é extremamente complexo, e parece fornecer uma função g que não produzirá boas previsões em novas amostras. O modelo com $p = 4$ parece ser o mais razoável neste caso. Dizemos que o modelo com $p = 1$ sofre de *sub-ajuste*, ou *underfitting* (não é suficiente para explicar bem os dados), enquanto que o modelo com $p = 50$ sofre de *super-ajuste*, ou *overfitting* (ele se ajusta demais à esta amostra específica, de modo que possui poder de generalização baixo). O objetivo desta seção é descrever um método para escolher o modelo com $p = 4$ graus automaticamente.

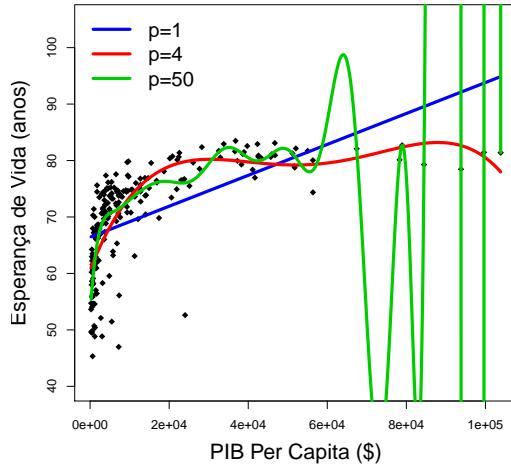


Figura 1.4: Comparação dos modelos $r(x) = \beta_0 + \sum_{i=1}^p \beta_i x^i$, para $p \in \{1, 4, 50\}$ nos dados de esperança de vida (Y) versus PIB per Capita (X).

□

O objetivo de um método de seleção de modelos é selecionar g que não sofra nem de sub nem de super-ajuste e, assim, que possua bom poder preditivo. De forma mais geral, gostaríamos de escolher uma função g dentro de uma classe de candidatos \mathbb{G} que possua bom poder preditivo. Assim, é necessário ter um critério para medir a qualidade de cada $g \in \mathbb{G}$. Como discutido na seção anterior, faremos isso através do risco quadrático. Uma vez que $R(g)$ é desconhecido, é necessário estimá-lo, o que pode ser feito conforme descrito na próxima seção.

1.4.1 Data Splitting e Validação Cruzada

O *risco observado* (também chamado de erro quadrático médio no conjunto de treinamento), definido por

$$EQM(g) := n^{-1} \sum_{i=1}^n (Y_i - g(\mathbf{X}_i))^2 \quad (1.1)$$

é um estimador muito otimista do real risco; ele leva ao super-ajuste, um ajuste perfeito dos dados. Isto ocorre pois g foi escolhida de modo a ajustar bem $(\mathbf{X}_1, Y_1), \dots, (\mathbf{X}_n, Y_n)$.

Uma maneira de solucionar este problema é dividir o conjunto de dados em duas partes, *treinamento* e *validação*:

$$\underbrace{(\mathbf{X}_1, Y_1), (\mathbf{X}_2, Y_2), \dots, (\mathbf{X}_s, Y_s)}_{\text{Treinamento (e.g., 70\%)}} \quad \underbrace{(\mathbf{X}_{s+1}, Y_{s+1}), \dots, (\mathbf{X}_n, Y_n)}_{\text{Validação (e.g., 30\%)}}.$$

Usamos o conjunto de treinamento para estimar g (e.g., estimar os coeficientes da regressão linear), mas usamos o conjunto de validação *apenas para estimar $R(g)$* via

$$R(g) \approx \frac{1}{n-s} \sum_{i=s+1}^n (Y_i - g(\mathbf{X}_i))^2 := \hat{R}(g),$$

isto é, avaliamos o erro quadrático médio no conjunto de validação.

Observação 1.6 Uma boa prática para escolher quais amostras serão utilizadas para compor o conjunto de treinamento e quais serão utilizadas para compor o conjunto de validação é fazê-lo **aleatoriamente**. Isto é, usar um gerador de números aleatórios para escolher quais amostras serão usadas para o treinamento, e quais serão usadas para a validação. Este procedimento evita problemas quando o banco de dados está previamente ordenado segundo uma covariável (por exemplo, quem coletou o banco pode ter organizado este em ordem crescente da variável resposta).

□

Como o conjunto de validação não foi usado para estimar os parâmetros de g , o estimador acima é consistente pela lei dos grandes números.

Observação 1.7 Para uma função de perda L qualquer, o risco pode ser aproximado por

$$R(g) \approx \frac{1}{n-s} \sum_{i=s+1}^n L(g; (\mathbf{X}_i, Y_i)) := \hat{R}(g).$$

□

Observação 1.8 O procedimento acima descrito é chamado de *data splitting*. O termo *validação cruzada* em geral é usado para designar uma variação deste estimador que faz uso de toda a amostra. Por exemplo, no *leave-one-out cross validation* (Stone, 1974), o estimador usado é dado por

$$R(g) \approx \frac{1}{n} \sum_{i=1}^n (Y_i - g_{-i}(\mathbf{X}_i))^2,$$

em que g_{-i} é ajustado utilizando-se todas as observações exceto a i -ésima delas, i.e., utilizando-se

$$(\mathbf{X}_1, Y_1), \dots, (\mathbf{X}_{i-1}, Y_{i-1}), (\mathbf{X}_{i+1}, Y_{i+1}), \dots, (\mathbf{X}_n, Y_n).$$

Alternativamente, pode-se usar o *k-fold cross validation*; veja, por exemplo, [Wasserman \(2006\)](#).

□

Observação 1.9 Segue da lei dos grandes números que a estimativa do risco baseada na divisão treinamento versus validação fornece um estimador consistente para o *erro condicional*. Por outro lado, o *leave-one-out cross validation* é aproximadamente não-viesado para o *erro esperado* (o risco descrito na Observação 1.4). De fato, se $R(g)$ é o erro esperado do método de construção de g , então

$$\begin{aligned} \mathbb{E} \left[\frac{1}{n+1} \sum_{i=1}^{n+1} (Y_i - g_{-i}(\mathbf{X}_i))^2 \right] &= \frac{1}{n+1} \sum_{i=1}^{n+1} \mathbb{E} [(Y_i - g_{-i}(\mathbf{X}_i))^2] \\ &= \frac{1}{n+1} \sum_{i=1}^{n+1} R(g) = R(g), \end{aligned}$$

em que a segunda igualdade segue do fato que g_{-i} tem a mesma distribuição de g . Assim, o estimador *leave-one-out cross validation* calculado em um conjunto com $n+1$ observações é não viesado para o erro esperado do estimador obtido com n amostras.

□

Desta maneira, uma forma de selecionar um modelo g dentro de uma classe de modelos \mathbb{G} consiste em usar validação cruzada para estimar $R(g)$ para cada $g \in \mathbb{G}$, e então escolher g com o menor risco estimado.

Observação 1.10 Do mesmo modo que o EQM no conjunto de treinamento é muito otimista pois cada $g \in \mathbb{G}$ é escolhida de modo a minimizá-lo, o EQM no conjunto de validação *avaliado no g com menor EQM no conjunto de validação* também é otimista. Uma forma de contornar isso, é dividir o conjunto original em três partes: treinamento, validação e teste. Os conjuntos de treinamento e validação são usados como descrito anteriormente, e o de teste é usado para estimar o erro do melhor estimador da regressão encontrado segundo o conjunto de validação.

□

Observação 1.11 Utilizando o conjunto de teste, podemos também fazer um intervalo de confiança para o risco. Assuma que $(\tilde{\mathbf{X}}_1, \tilde{Y}_1), \dots, (\tilde{\mathbf{X}}_s, \tilde{Y}_s)$ são elementos do conjunto de teste, não utilizados nem para o treinamento, nem para a validação do modelo. Um estimador não viesado para o risco de g (que foi estimada com base nos conjuntos de treinamento e validação) é dado por

$$\widehat{R}(g) = \frac{1}{s} \sum_{k=1}^s \underbrace{\left(\tilde{Y}_k - g(\tilde{\mathbf{X}}_k) \right)^2}_{W_k}.$$

Como se trata de uma média de variáveis i.i.d., sabemos pelo Teorema do Limite Central que

$$\widehat{R}(g) \approx \text{Normal} \left(R(g), \frac{1}{s} \mathbb{V}[W_1] \right).$$

Como W_1, \dots, W_s são i.i.d.'s, podemos estimar $\mathbb{V}[W_1]$ com

$$\widehat{S^2} = \frac{1}{s} \sum_{k=1}^s (W_k - \bar{W})^2,$$

em que $\bar{W} = \frac{1}{s} \sum_{k=1}^s W_k$. Assim, um IC aproximado para $R(g)$ (com confiança 95%) é dado por

$$\widehat{R}(g) \pm 2\sqrt{\frac{1}{s} \widehat{S^2}}.$$

□

Exemplo 1.6 [Esperança de vida e PIB per Capita] Revisitamos os Exemplos 1.1 e 1.5. Aqui nosso objetivo é selecionar o melhor estimador dentro da classe

$$\mathbb{G} = \left\{ g(x) : g(x) = \beta_0 + \sum_{i=1}^p \beta_i x^i, \text{ para } p \in \{1, 2, \dots, 50\} \right\}$$

A Figura 1.5a mostra o erro quadrático médio *no conjunto de treinamento* para cada $g \in \mathbb{G}$. Observamos que, de fato, tal medida não é um bom estimador do erro e em particular leva ao super-ajuste. Isto é, quanto maior o número de parâmetros, menor o erro quadrático médio no conjunto de treinamento, mas não necessariamente melhor é o modelo. Por outro lado, a Figura 1.5b mostra o risco estimado para cada $g \in \mathbb{G}$ via EQM no conjunto de validação. Como os riscos para $p > 11$ são demasiadamente altos, mostramos apenas o comportamento para $p \leq 11$. O menor risco estimado é obtido tomando-se $p = 8$ (i.e., o modelo com 9 parâmetros). Este modelo apresenta bom poder preditivo; compare as Figuras 1.6a e 1.6b. Assim, o *data splitting* nos leva à escolha de um modelo adequado para predizer novas observações.

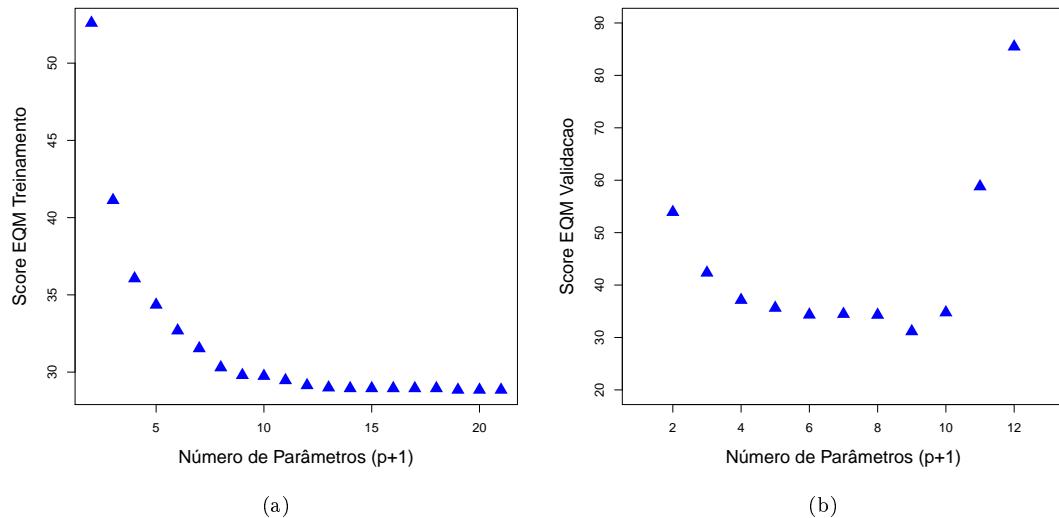


Figura 1.5: Comparaçāo dos modelos $r(x) = \beta_0 + \sum_{i=1}^p \beta_i x^i$, para $p \in \{1, 2, \dots, 50\}$ nos dados de esperança de vida (Y) versus PIB per Capita (X). EQM do conjunto de validação (esquerda) e EQM do conjunto de treinamento (direita). A segunda forma de fazer a seleção de modelos leva ao superajuste.

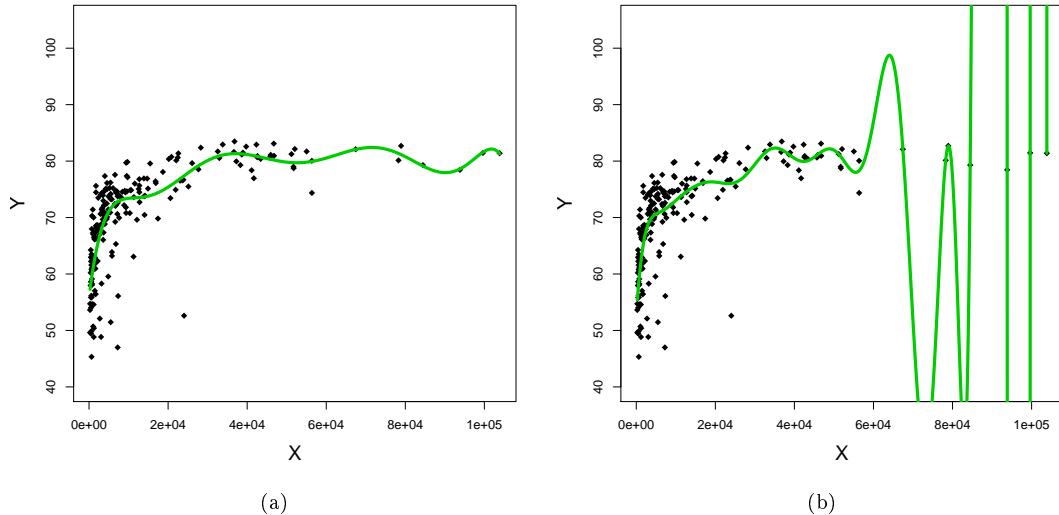


Figura 1.6: Comparação dos modelos ajustados segundo melhor EQM do conjunto de treinamento (esquerda) e segundo EQM do conjunto de validação (direita).

Retomaremos este exemplo na Seção 4.13.1.

□

Exemplo 1.7 [A especificação do modelo estar correta não garante melhor poder preditivo.]

Dentro da cultura do *data modeling*, é comum checar se as suposições feitas por um modelo estão corretas. Contudo, a especificação do modelo estar correta não garante melhor poder preditivo. Isto é, um modelo especificado incorretamente pode ter melhor desempenho. Nesta seção isso é ilustrado via um exemplo simulado. Os dados são gerados de modo que $Y|\mathbf{x} \sim N(\beta^t \mathbf{x}, 1)$, com $\beta = (0, 3, 2, 0.2, 0.1, 0.1)$ e $\mathbf{x} = (1, x, x^2, \dots, x^5)$. Isto é, a regressão real é um polinômio de quinto grau. Ajustamos então um polinômio de quinto grau e um polinômio de segundo grau.

```

gerar.modelo=function(n,beta)
{
  # X1 ~ U(0,1) (unica covariavel)
  # y=(X1,X1^2,...,X1^d)'*beta+N(0,1), com d=length(beta)

  # unica covariavel:

```

```

x=as.matrix(runif(n))
# calcular polinomios: X=(X1,X1^2,...,X1^d), com d=length(beta):
X=t(apply(x,1,function(xx)xx^(1:length(beta))))

y=X%*%beta+rnorm(n)
return(list(x=x,y=y))
}

ajustar.modelo=function(x,y,d)
{
  # ajusta polinomio de grau d
  return(lm(y~poly(x,degree = d, raw=TRUE)))
}

predizer.observacoes=function(ajuste,x)
{
  x=as.data.frame(x)
  colnames(x)="x"
  return(predict(ajuste,newdata=x))
}

```

A Figura 1.7 indica que, apesar de o modelo correto ser um polinômio de quinto grau, um ajuste de segundo grau leva a melhores resultados. Isso ocorre pois o modelo mais simples possui variância muito menor (ainda que ele seja viesado) e, assim, evita o *overfitting*.

```

set.seed(400)
beta=c(3,2,0.2,0.1,0.1)
dados.treinamento=gerar.modelo(10,beta)

plot(dados.treinamento$x,dados.treinamento$y,pch=16,xlim=c(0,1),
      xlab="x",ylab="y",cex.lab=1.5,ylim=c(-1.2,5))
grid.x=seq(0,1,length.out = 1000)
lines(grid.x,poly(grid.x,degree = length(beta),raw=TRUE)%*%beta,
      lwd=3)

modelo.5=ajustar.modelo(dados.treinamento$x,dados.treinamento$y,
                        d=5)
predito.modelo.5=predizer.observacoes(modelo.5,grid.x)
lines(grid.x,predito.modelo.5,lwd=3,col=2)

```

```

modelo.2=ajustar.modelo(dados.treinamento$x, dados.treinamento$y,
                        d=2)
predito.modelo.2=predizer.observacoes(modelo.2,grid.x)
lines(grid.x,predito.modelo.2, lwd=3, col=4)

legend("topleft",c("Real","Grau 5","Grau 2"),col=c(1,2,4),lwd=3,
       cex=1.3,bty = "n")

```

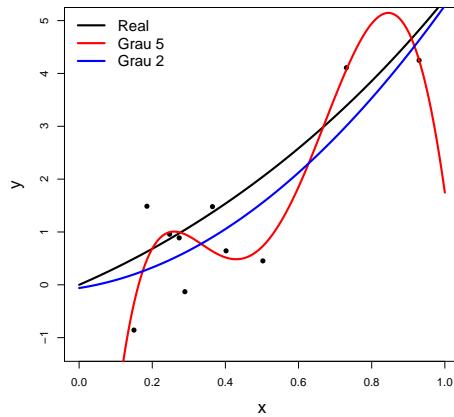


Figura 1.7: Apesar de o modelo correto ser um polinômio de quinto grau, um ajuste de segundo grau levou a um resultado melhor.

A seguir replicamos o experimento acima para 1000 conjuntos de treinamento diferentes.

```

B=1000 # numero de vezes para repetir experimento
beta=c(3,2,0.2,0.1,0.1)

simular.risco=function()
{
  dados.treinamento=gerar.modelo(10,beta) # para ajustar modelo
  dados.teste=gerar.modelo(1e5,beta) # para avaliar risco

  modelo.5=ajustar.modelo(dados.treinamento$x,
                         dados.treinamento$y,

```

```

d=5)
predito.modelo.5=predictor.observacoes (modelo.5,dados.teste$x)
risco.modelo.5=mean ( (predito.modelo.5-dados.teste$y)^2)

modelo.2=ajustar.modelo (dados.treinamento$x,
                        dados.treinamento$y,
                        d=2)
predito.modelo.2=predictor.observacoes (modelo.2,dados.teste$x)
risco.modelo.2=mean ( (predito.modelo.2-dados.teste$y)^2)

return (c (risco.modelo.2,risco.modelo.5))
}

# repetir experimento B vezes, calculando o risco em cada um deles
riscos=replicate (B,simular.risco ())

```

O erro esperado do modelo com 2 graus de fato é menor que o erro esperado do modelo especificado corretamente (i.e., com 5 graus):

```

# risco esperado 2 graus vs 5 graus:
apply (riscos,1,mean)

## [1] 1.700045 22354.077400

```

Além disso, para a grande maioria dos conjuntos de treinamento, o risco condicional do modelo mais simples é menor:

```

# para qual proporcao de conjutos de dados o modelo com
# 2 graus eh melhor que o modelo com 5 graus?
mean (riscos[1,]<=riscos[2,])

## [1] 0.978

```

□

1.4.1.1 Penalização: uma alternativa

Uma forma alternativa de se estimar o risco de um certo modelo g é utilizando uma medida de penalização ou complexidade. Quanto mais parâmetros no modelo, mais o erro quadrático médio observado, $EQM(g)$ (Eq. 1.1), subestima $R(g)$, isto é, maior a diferença

entre $EQM(g)$ e $R(g)$. A ideia por trás de métodos de penalização é criar uma medida de complexidade para g , $\mathcal{P}(g)$, que é utilizada para corrigir essa diferença. Por exemplo, $\mathcal{P}(g)$ pode ser o número de parâmetros que há no modelo. Podemos então compensar o quanto subestimado $R(g)$ é adicionando estas duas quantidades:

$$R(g) \approx EQM(g) + \mathcal{P}(g).$$

Existem diversas funções de penalização, com diferentes motivações teóricas. Dois exemplos populares são:

- $\mathcal{P}(g) = 2/n * p * \hat{\sigma}^2$ (conhecido como AIC);
- $\mathcal{P}(g) = \log(n)/n * p * \hat{\sigma}^2$ (conhecido como BIC).

Aqui, p é o número de parâmetros de g e $\hat{\sigma}^2$ é uma estimativa de $\mathbb{V}[Y|\mathbf{x}]$.

Note que, se g tem muitos parâmetros, $EQM(g)$ é em geral muito baixo (super-ajuste), mas em compensação $\mathcal{P}(g)$ é alto. Analogamente, se g tem poucos parâmetros, $EQM(g)$ é em geral muito alto (sub-ajuste), mas em compensação $\mathcal{P}(g)$ é baixo. Assim, espera-se que $EQM(g) + \mathcal{P}(g)$ seja uma boa estimativa para $R(g)$.

A Figura 1.8 ilustra o uso do AIC para seleção de modelos no Exemplo 1.6. Note como o comportamento deste gráfico é similar ao da Figura 1.5b. Em particular, os melhores valores encontrados para p em ambos são muito próximos.

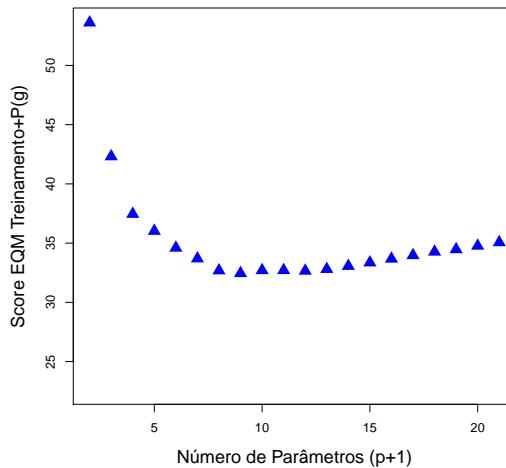


Figura 1.8: Comparação dos modelos $r(x) = \beta_0 + \sum_{i=1}^p \beta_i x^i$, para $p \in \{1, 2, \dots, 50\}$ nos dados de esperança de vida (Y) versus PIB per Capita (X) via penalização AIC.

1.4.2 Balanço entre Viés e Variância

Um grande apelo para o uso do risco quadrático é sua grande interpretabilidade: o risco quadrático (condicional no novo \mathbf{x} observado) pode ser decomposto como

$$\mathbb{E} [(Y - \hat{g}(\mathbf{X}))^2 | \mathbf{X} = \mathbf{x}] = \mathbb{V}[Y | \mathbf{X} = \mathbf{x}] + (r(\mathbf{x}) - \mathbb{E}[\hat{g}(\mathbf{x})])^2 + \mathbb{V}[\hat{g}(\mathbf{x})].$$

Note que estamos trabalhando com o erro esperado (veja a Observação 1.4), em que a aleatoriedade está tanto em Y quanto na amostra de treinamento e, por esta razão, estamos usando a notação \hat{g} para denotar o estimador de g .

Assim, o risco pode ser decomposto em três termos:

- $\mathbb{V}[Y | \mathbf{X} = \mathbf{x}]$ é a variância intrínseca da variável resposta, que não depende da função \hat{g} escolhida e, assim, não pode ser reduzida
- $(r(\mathbf{x}) - \mathbb{E}[\hat{g}(\mathbf{x})])^2$ é o quadrado do viés do estimador \hat{g}
- $\mathbb{V}[\hat{g}(\mathbf{x})]$ é sua variância; ambos podem ser reduzidos se escolhemos \hat{g} adequado.

Grosso modo, modelos com muitos parâmetros possuem viés relativamente baixo, mas variância alta, já que é necessário estimar todos eles. Já modelos com poucos parâmetros possuem variância baixa, mas viés muito alto, já que são demasiado simplistas para descrever o modelo gerador dos dados. Assim, com a finalidade de obter um bom poder preditivo deve-se escolher um número de parâmetros nem tão alto, nem tão baixo. A Figura 1.9 mostra qualitativamente o balanço (também chamado de *tradeoff*) entre viés e variância.

Tal *tradeoff* é justamente o que ocorre no Exemplo 1.5: enquanto que $p = 50$ induz um modelo com viés relativamente baixo mas variância alta (há muitos parâmetros para serem estimados), $p = 1$ leva a um viés extremamente alto, mas variância muito baixa. Ao se selecionar o melhor modelo usando, por exemplo, *data splitting* (veja Figura 1.5b) estamos justamente buscando a melhor combinação viés-variância possível de modo a obter um modelo com um risco baixo.

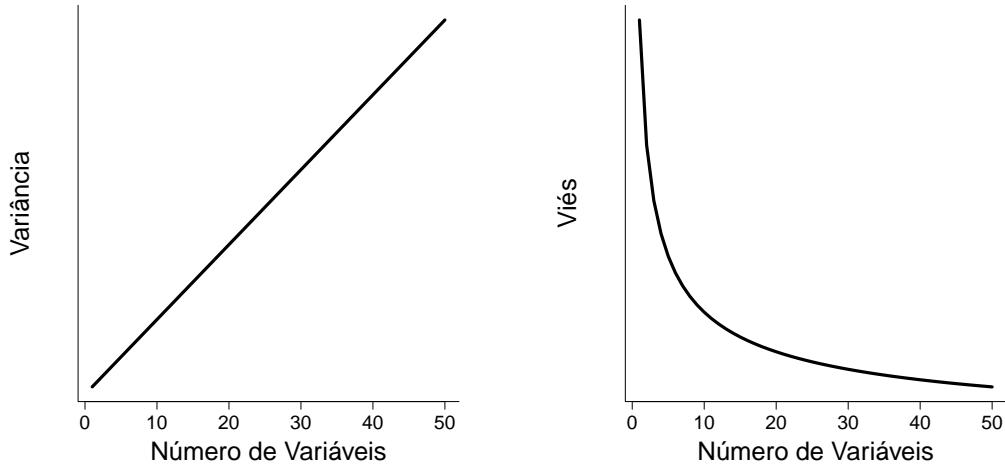


Figura 1.9: O balanço entre viés e variância.

Note que, enquanto que em inferência paramétrica tradicionalmente buscam-se estimadores não viesados para os parâmetros de interesse, em regressão moderna é comum abrir mão de se ter um estimador viesado para, em troca, conseguir-se uma variância menor e, assim, um risco menor. Talvez essa seja uma das grandes mudanças de paradigma do ponto de vista metodológico em regressão moderna quando comparada a análises tradicionais.

1.5 Tuning Parameters

A função do parâmetro p no Exemplo 1.6, o grau do polinômio utilizado, é controlar o balanço entre viés e variância. O valor ótimo de p depende de n e de $r(\mathbf{x})$. O parâmetro p é dito um *tuning parameter* (i.e., um parâmetro de sintonização).

Vários dos métodos de regressão possuem um ou mais *tuning parameters*. Como feito no Exemplo 1.6, neste livro sempre iremos escolhê-los via validação cruzada ou *data splitting*, ainda que esta não seja a única maneira de fazer esta seleção (veja por exemplo Wasserman 2006).

1.6 Resumo

Neste capítulo vimos que há ao menos dois motivos para se fazer uma análise de regressão: o *motivo inferencial*, no qual estamos interessados em tirar conclusões sobre a população à qual os dados pertencem, e o *motivo preditivista*, no qual estamos interessados em ter boas previsões.

Vimos também que existem duas culturas em análise de dados: *data modeling culture* e *algorithmic modeling culture*. Enquanto que na primeira há uma preocupação em se testar suposições dos modelos utilizados, na segunda o foco está em se obter modelos com bom poder preditivo.

Mostramos também que nosso objetivo nesta Parte I deste livro é mostrar diversos métodos que permitem estimar funções $g(\mathbf{x})$ com risco $R(g) = \mathbb{E}[(Y - g(\mathbf{X}))^2]$ baixo. Vimos que encontrar g com risco baixo equivale a encontrar uma boa estimativa da função de regressão. Vimos também que o risco pode ser decomposto em viés e variância. Modelos complexos possuem variância alta, mas viés baixo, enquanto que modelos simples possuem variância baixa, mas viés alto. Nossa objetivo é encontrar a complexidade que equilibre este balanço, de modo a termos um risco baixo.

Finalmente, estudamos dois métodos para estimar o risco de uma dada função g : a validação cruzada/*data splitting* e a penalização. Estes métodos podem ser usados para fazer seleção de modelos e, assim, equilibrar o balanço entre viés e variância.

Capítulo 2

Métodos Paramétricos

Métodos paramétricos assumem que a função de regressão pode ser parametrizada com um número finito de parâmetros. Neste capítulo iremos nos restringir apenas a modelos de regressão linear. Assumimos que a maior parte do conteúdo apresentado aqui é uma revisão de conceitos já conhecidos, mas sob uma perspectiva possivelmente diferente.

A regressão linear assume que

$$r(\mathbf{x}) = \boldsymbol{\beta}^t \mathbf{x} = \beta_0 x_0 + \beta_1 x_1 + \dots + \beta_d x_d, \quad (2.1)$$

em que adotamos a convenção $x_0 \equiv 1$, e onde $\boldsymbol{\beta} = (\beta_0, \dots, \beta_d)$. Note que x_i não precisa necessariamente ser a i -ésima variável original; podemos criar novas covariáveis que são funções das originais (ex: $x_i^2, x_i x_j$ etc; veja também a Seção 4.1 e Exemplo 1.5).

2.1 O Método dos Mínimos Quadrados

Uma forma de estimar os coeficientes $\boldsymbol{\beta}$ da regressão linear é utilizando o método dos mínimos quadrados, que propõe o estimador dado por

$$\widehat{\boldsymbol{\beta}} = \arg \min_{\boldsymbol{\beta}} \sum_{i=1}^n (Y_i - \beta_0 - \beta_1 x_{i,1} - \dots - \beta_d x_{i,d})^2. \quad (2.2)$$

A solução para tal problema é dada por

$$\widehat{\boldsymbol{\beta}} = (\widehat{\beta}_0, \widehat{\beta}_1, \dots, \widehat{\beta}_d) = (\mathbb{X}^t \mathbb{X})^{-1} \mathbb{X}^t \mathbb{Y}, \quad (2.3)$$

em que

$$\mathbb{X} = \begin{pmatrix} X_{1,0} & \dots & X_{1,d} \\ \vdots & \ddots & \vdots \\ X_{n,0} & \dots & X_{n,d} \end{pmatrix}$$

e $\mathbb{Y} = (Y_1, \dots, Y_n)$ (aqui usamos a convenção de que um vetor, quando utilizado como matriz, é um vetor *coluna*.)

Assim, a função de regressão é estimada por

$$g(\mathbf{x}) = \hat{\beta}^t \mathbf{x}.$$

Boa parte da literatura estatística é voltada para a justificativa do método dos mínimos quadrados sob um ponto de vista de estimadores de máxima verossimilhança, assim como para testes de aderência e métodos para a construção de intervalos de confiança para os parâmetros β_i . Indicamos para o leitor interessado nestes aspectos o livro de Neter et al. (1996).

Assumir que a verdadeira regressão é, de fato, linear, em geral é uma suposição muito forte. Existe, contudo, uma literatura substancial que tenta justificar o método dos mínimos quadrados mesmo quando $r(\mathbf{x})$ não satisfaz essa suposição. Na seção que segue mostramos uma dessas ideias.

2.1.1 Mínimos Quadrados quando a Suposição de Linearidade Falha

A suposição de que $r(\mathbf{x})$ é linear muitas vezes não é válida. Contudo, mesmo quando esta suposição falha, frequentemente existe um vetor β_* tal que $g_{\beta_*}(\mathbf{x}) = \beta_*^t \mathbf{x}$ tem bom poder preditivo. Neste caso, o método dos mínimos quadrados tende a produzir estimadores com baixo risco. Isto ocorre pois $\hat{\beta}$ converge para o melhor preditor linear,

$$\beta_* = \arg \min_{\beta} R(g_{\beta}) = \arg \min_{\beta} \mathbb{E} [(Y - \beta^t \mathbf{X})^2], \quad (2.4)$$

em que (\mathbf{X}, Y) é uma nova observação, mesmo quando a verdadeira regressão $r(\mathbf{x})$ não é linear. Além disso, o risco associado a $\hat{\beta}$ converge para o risco associado a β_* ¹. Isto é mostrado no teorema a seguir.

Teorema 2.1 *Seja β_* o melhor preditor linear (Equação 2.4), e $\hat{\beta}$ o estimador de mínimos quadrados (Equação 2.3). Assuma que $\Sigma = \mathbb{E}[\mathbf{X}\mathbf{X}^t]$ está bem definido. Então*

$$\hat{\beta} \xrightarrow[n \rightarrow \infty]{P} \beta_* \text{ e } R(g_{\hat{\beta}}) \xrightarrow[n \rightarrow \infty]{P} R(g_{\beta_*})$$

¹ β_* é chamado de *oráculo*.

Demonstração. Primeiramente, notamos que se minimizamos $\mathbb{E}[(Y - \beta^t \mathbf{X})^2]$, obtemos que

$$\beta_* = \Sigma^{-1}\alpha,$$

em que $\alpha = (\mathbb{E}[YX_0], \dots, \mathbb{E}[YX_d])$. Também notamos que podemos reescrever $\hat{\beta}$ como

$$\hat{\beta} = \hat{\Sigma}^{-1}\hat{\alpha},$$

em que $\hat{\Sigma} = n^{-1} \sum_{i=1}^n \mathbf{X}_i \mathbf{X}_i^t$ e $\hat{\alpha} = (\hat{\alpha}_0, \dots, \hat{\alpha}_d)$, com $\hat{\alpha}_j = n^{-1} \sum_{i=1}^n Y_i X_{i,j}$. Pela lei fraca dos grandes números, temos que

$$\hat{\Sigma} \xrightarrow[n \rightarrow \infty]{P} \Sigma$$

e

$$\hat{\alpha} \xrightarrow[n \rightarrow \infty]{P} \alpha,$$

de modo que, pelo Teorema de Mann-Wald,

$$\hat{\beta} \xrightarrow[n \rightarrow \infty]{P} \beta_*$$

e, assim, novamente pelo Teorema de Mann-Wald,

$$R(g_{\hat{\beta}}) \xrightarrow[n \rightarrow \infty]{P} R(g_{\beta_*})$$

□

O teorema anterior mostra, portanto, que $\hat{\beta}$ converge para o melhor preditor linear, β_* . Pode-se também derivar a taxa desta convergência, isto é, o quanto rápido ela ocorre; veja por exemplo [Györfi and Krzyzak \(2002\)](#).

2.1.2 Regressão Linear no R

Se dados é um data frame com o banco de dados, o estimador de mínimos quadrados pode ser calculado fazendo

```
ajuste = lm(nomeResposta ~ nomeExplicativa1+nomeExplicativa2,
            data = dados)
```

Automaticamente o R inclui um intercepto. Para ajustar a regressão com todas as covariáveis no banco de dados, é possível usar o atalho

```
ajuste = lm(nomeResposta ~ . , data = dados)
```

Para predizer novas observações, pode-se usar a função predict:

```
valoresPrevistos= predict(ajuste, newdata = novosDados)
```

Notamos que novosDados deve ser um dataframe com o mesmo formato (incluindo nomes das variáveis) que dados.

2.2 Resumo

Vimos neste capítulo que uma regressão linear assume que a função de regressão pode ser escrita como uma expansão da forma $r(\mathbf{x}) = \boldsymbol{\beta}^t \mathbf{x} = \beta_0 + \beta_1 x_1 + \dots + \beta_d x_d$. O vetor \mathbf{x} pode representar as covariáveis originais ou transformações dessas. Vimos o método dos mínimos quadrados, uma forma de se estimar o vetor $\boldsymbol{\beta}$, e mostramos que, mesmo quando a verdadeira regressão não é linear, esse estimador converge para $\boldsymbol{\beta}_*$, o melhor preditor linear de y com base em \mathbf{x} .

Infelizmente, quando há muitas covariáveis, o método dos mínimos quadrados não leva a bons resultados devido ao super-ajuste e à variância extremamente alta (Seção 1.4). Quando $d > n$, o estimador não está nem bem definido, pois a matriz $\mathbb{X}^t \mathbb{X}$ utilizada para estimar $\boldsymbol{\beta}$ (Eq. 2.3) não é invertível. No próximo capítulo mostramos algumas abordagens para contornar esse problema.

Capítulo 3

Métodos Paramétricos em Dimensões Altas

Quando há muitas covariáveis (i.e., d é grande), o estimador de mínimos quadrados em geral possui performance ruim devido ao super-ajuste: há muitos parâmetros a serem estimados e, portanto, a função de regressão estimada em geral possui baixo poder preditivo. Em outras palavras, a variância do estimador resultante é alta pois muitos parâmetros devem ser estimados¹. Neste capítulo investigamos alguns métodos que podem ser usados para contornar este problema.

3.1 Busca pelo Melhor Subconjunto de Covariáveis

Uma solução para este problema consiste em retirar algumas das variáveis da regressão de modo a diminuir a variância da função de predição estimada \hat{g} . Em outras palavras, esta solução consiste em aumentar o viés do estimador de $r(\mathbf{x})$ e, em troca, esperar que isso resulte em uma diminuição substancial de sua variância. Note que um estimador com base em apenas um subconjunto das variáveis explicativas tem ainda mais motivos para ter uma performance boa: são comuns os problemas em que esperamos que algumas das covariáveis não influenciem (ou influenciem pouco) a resposta Y .

Uma maneira de retirar algumas variáveis da regressão de modo fundamentado é buscar a estimativa dada por

$$\widehat{\boldsymbol{\beta}}_{L_0} = \arg \min_{\boldsymbol{\beta} \in \mathbb{R}^d} \sum_{k=1}^n \left(y_k - \beta_0 - \sum_{i=1}^d \beta_i x_{k,i} \right)^2 + \lambda \sum_{i=1}^d \mathbb{I}(\beta_i \neq 0). \quad (3.1)$$

A penalização $\sum_{i=1}^d \mathbb{I}(\beta_i \neq 0)$ induz modelos com poucas covariáveis, em particular se λ é alto. Por exemplo, quando $\lambda \rightarrow \infty$, a solução de 3.1 é $\widehat{\boldsymbol{\beta}}_{L_0} \equiv \mathbf{0}$, ou seja, um vetor de

¹ Se $d > n$, o método de mínimos quadrados nem pode ser implementado, uma vez que $\mathbb{X}^t \mathbb{X}$ não é invertível!

zeros (i.e., a penalização alta descarta todas as covariáveis do modelo). No outro extremo, quando $\lambda = 0$, temos que $\widehat{\beta}_{L_0}$ nada mais é que o estimador de mínimos quadrados (i.e., sem penalização nenhuma variável é descartada). Quando $\lambda = \frac{2}{n}\widehat{\sigma}^2$, encontrar a solução da Equação 3.1 nada mais é que uma busca entre os 2^d modelos na classe

$$\begin{aligned} \mathbb{G} = & \{g(\mathbf{x}) = \widehat{\beta}_0, \\ & g(\mathbf{x}) = \widehat{\beta}_0 + \widehat{\beta}_1 x_1, \\ & g(\mathbf{x}) = \widehat{\beta}_0 + \widehat{\beta}_2 x_2, \\ & \dots \\ & g(\mathbf{x}) = \widehat{\beta}_0 + \widehat{\beta}_d x_d, \\ & g(\mathbf{x}) = \widehat{\beta}_0 + \widehat{\beta}_1 x_1 + \widehat{\beta}_2 x_2, \\ & g(\mathbf{x}) = \widehat{\beta}_0 + \widehat{\beta}_1 x_1 + \widehat{\beta}_3 x_3, \\ & \dots \\ & g(\mathbf{x}) = \widehat{\beta}_0 + \widehat{\beta}_1 x_1 + \widehat{\beta}_2 x_2 + \dots + \widehat{\beta}_d x_d\}, \end{aligned}$$

em que se utiliza o critério AIC para determinar o melhor modelo. Em outras palavras, quando $\lambda = \frac{2}{n}\widehat{\sigma}^2$, encontrar a solução da Equação 3.1 é equivalente a ajustar cada um dos 2^d modelos via mínimos quadrados e estimar o risco $R(g)$ para cada um deles via AIC (Seção 1.4.1.1). O modelo escolhido é aquele com menor risco estimado.

De um ponto de vista teórico, estimar β com base na Equação 3.1 resolve o problema, i.e., conseguimos descartar várias das variáveis e, assim, obter um estimador com variância (e erro preditivo) menor. Em contrapartida, há um aumento no viés do estimador, já que nem todas as variáveis são usadas. Como explicado na Seção 1.4.2, quando o aumento no viés é pequeno se comparado à diminuição na variância, o modelo resultante tem poder preditivo maior.

Contudo, do ponto de vista prático, resolver a Equação 3.1 é computacionalmente difícil quando há muitas covariáveis, uma vez que há 2^d modelos para serem ajustados. Veja na Figura 3.1 como o número de modelos cresce à medida que d aumenta.

```
library(ggplot2)

n_modelos <- function(p) {
  x           <- numeric()
  x[1:p]     <- 1:p
  return(sum(sapply(x, choose, n = p))) }

d = seq(1, 20, by = 1)
```

```
modelos <- data.frame(d = d, mod = sapply(d, n_modelos) )

ggplot(modelos, aes(x = d, y = mod)) +
  geom_line(colour = "blue", size = 1) +
  geom_point(colour = "blue", size = 2) +
  ylab("Número de modelos")
```

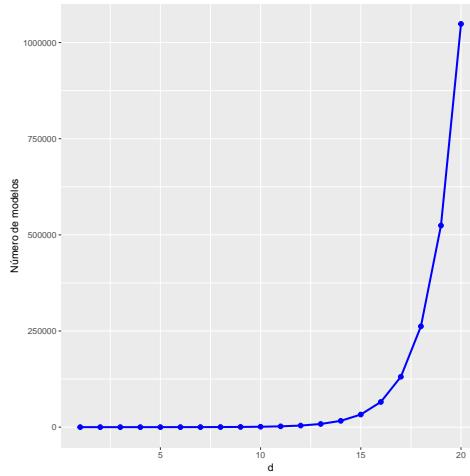


Figura 3.1: Número de subconjuntos de d covariáveis como função de d .

Um modo de contornar esse problema é usar alguma heurística para encontrar um modelo com erro baixo. Isto é, ao invés de resolver a Equação 3.1, busca-se por uma aproximação desta solução, como descrito na próxima seção.

3.2 Regressão Stepwise

Encontrar o mínimo da Equação 3.1 é computacionalmente difícil, pois envolve uma busca de um entre 2^d modelos. Uma série de algoritmos que consistem de *heurísticas* para encontrar esta solução são os chamados de *stepwise regression* (James et al., 2013). Um exemplo de tal heurística é o *forward stepwise*, um algoritmo sequencial no qual a cada passo apenas uma variável é adicionada:

1. Para $j = 1, \dots, d$, ajuste a regressão de Y na j -ésima variável X_j . Seja $\widehat{R}(g_j)$ o risco estimado desta função (usando AIC ou validação cruzada). Defina $\widehat{j} = \arg \min_j \widehat{R}(g_j)$ e $S = \{\widehat{j}\}$.
2. Para cada $j \in S^c$, ajuste a regressão de $Y = \beta_j X_j + \sum_{s \in S} \beta_s X_s + \epsilon$, e seja $\widehat{R}(g_j)$ o risco estimado desta função (usando AIC ou validação cruzada). Defina $\widehat{j} = \arg \min_{j \in S^c} \widehat{R}(g_j)$ e atualize $S \leftarrow S \cup \widehat{j}$
3. Repita o passo anterior até que todas as variáveis estejam em S ou até que não seja possível mais ajustar a regressão
4. Selecione o modelo com menor risco estimado.

Note que, no *forward stepwise*, ao invés de buscarmos um entre 2^d modelos, somente é necessário investigar $1 + d(d + 1)/2$ modelos. Uma segunda maneira de encontrar um estimador linear da regressão com bom risco quando há muitas covariáveis é o *lasso*, que descrevemos a seguir.

3.3 Lasso

O lasso, desenvolvido por Tibshirani (1996), tem como finalidade encontrar um estimador de uma regressão linear que possua risco menor que o dos mínimos quadrados. O lasso possui duas grandes vantagens com relação a heurísticas *stepwise*: sua solução é mais rápida e possui mais garantias teóricas. Ele também busca soluções esparsas, isto é, soluções em que as estimativas de vários dos coeficientes β_j são zero. Assim como no caso da busca pelo melhor subconjunto de covariáveis visto acima, isso é feito minimizando-se o erro quadrático médio adicionado a um fator de penalização. Contudo, a penalização utilizada pelo lasso não é $\sum_{i=1}^d \mathbb{I}(\beta_i \neq 0)$ como na Equação 3.1.

Note que $\sum_{i=1}^d \mathbb{I}(\beta_i \neq 0)$ é uma medida de *esparsidade* do vetor β . A ideia do lasso é trocar esta medida por $\sum_{j=1}^d |\beta_j|$, que também mede esparsidade: vetores com entradas próximas a zero possuem um valor de $\sum_{j=1}^d |\beta_j|$ menor que vetores cujas entradas são grandes. Compare a segunda e terceira colunas da Tabela 3.1.

Tabela 3.1: Diferentes medidas de esparsidade/penalização. Todas elas conseguem capturar se um vetor tem entradas pequenas.

β	$\sum_{j=1}^d \mathbb{I}(\beta_j \neq 0)$	$\sum_{j=1}^d \beta_j $	Penalidade $\sum_{j=1}^d \beta_j^2$
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)	0	0	0
(1, 5, 6, -1, -4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)	5	17	79
(1, 5, 6, -1, -4, 1, 4, 1, 1, 5, 8, 12, 1)	13	50	332

Note também que modelos com coeficientes pequenos também são pouco complexos, pois, para qualquer valor das covariáveis, dão previsões parecidas. Assim, $\sum_{j=1}^d |\beta_j|$ é uma outra forma de medir a complexidade de um modelo.

Desta forma, no lasso, buscamos por

$$\hat{\beta}_{L_1, \lambda} = \arg \min_{\beta} \sum_{k=1}^n \left(y_k - \beta_0 - \sum_{j=1}^d \beta_j x_{k,j} \right)^2 + \lambda \sum_{j=1}^d |\beta_j|, \quad (3.2)$$

em que L_1 indica o fato de que estamos medindo a esparsidade de um vetor β usando sua norma em L_1 , $\|\beta\|_{L_1} = \sum_{j=1}^d |\beta_j|$.

Cada valor do parâmetro λ (*tuning parameter*) leva a um conjunto de coeficientes estimados $\hat{\beta}_{L_1, \lambda}$ diferentes. Assim como ocorre quando usamos a penalização da Equação 3.1, quando $\lambda = 0$, minimizar a Eq. 3.2 é equivalente a minimizar a Eq. 2.2, i.e., o lasso torna-se idêntico ao estimador de mínimos quadrados, com todos coeficientes diferentes de zero. Por outro lado, quando λ é grande,

$$\sum_{k=1}^n \left(y_k - \beta_0 - \sum_{j=1}^d \beta_j x_{k,j} \right)^2 + \lambda \sum_{j=1}^d |\beta_j| \approx \lambda \sum_{j=1}^d |\beta_j|,$$

de modo que o estimador dos coeficientes é tal que $\hat{\beta}_1 = 0, \dots, \hat{\beta}_d = 0$. Assim, para λ grande, o estimador dado pelo lasso tem variância zero, mas um viés muito alto. Desta maneira, adicionar a penalização $\lambda \sum_{j=1}^d |\beta_j|$ também induz estimadores com variância menor que aqueles dados pelo método de mínimos quadrados.

A escolha de λ é em geral feita pela validação cruzada. Isto é, estimamos $R(g_\lambda)$ para cada λ de interesse², e então selecionamos λ que leva ao melhor modelo selecionado; veja a Figura 3.2.

² A cada λ corresponde um $\hat{\beta}_{L_1, \lambda}$, que por sua vez corresponde uma função de previsão $g(\mathbf{x})$.

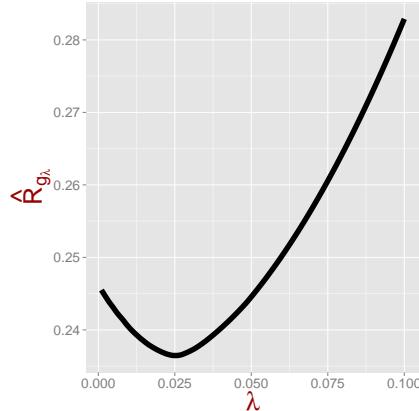


Figura 3.2: Processo de escolha de λ via validação cruzada. O risco para cada valor de λ é estimado usando-se validação cruzada.

Duas características positivas do lasso são:

- É extremamente rápido achar $\hat{\beta}_{L_1, \lambda}$ para todos os λ simultaneamente. Diversos algoritmos foram desenvolvidos nos últimos anos para fazer tal tarefa, sendo o LARS um dos primeiros desses. Veja mais detalhes sobre isso em [Friedman et al. \(2010\)](#).
- A solução induzida por 3.2 possui muitos zeros (i.e., o vetor $\hat{\beta}_{L_1, \lambda}$ é esparsão). Assim, o modelo resultante é de fácil interpretação. Note que o fato de $\hat{\beta}_{L_1, \lambda}$ ser esparsão não é trivial (ao contrário do que ocorre com a penalização $\sum_{j=1}^d \mathbb{I}(\beta_j \neq 0)$). Para mais detalhes sobre esse aspecto veja, e.g., [Hastie et al. 2009b](#).

3.3.1 Garantias Teóricas

Mesmo quando a regressão não é de fato linear, o estimador dado pelo lasso converge para o oráculo esparsa. Mais especificamente, temos o seguinte teorema:

Teorema 3.1 ([Greenshtein and Ritov, 2004](#)) *Seja*

$$\beta_* = \arg \min_{\beta} \mathbb{E}[(Y - \beta^t \mathbf{X})^2] \text{ sujeito a } \|\beta\|_1 \leq L$$

o melhor preditor linear esparsa. Se $(\mathbf{X}_1, Y_1), \dots, (\mathbf{X}_n, Y_n)$ são i.i.d.'s e $|Y|, |\mathbf{X}_1|, \dots, |\mathbf{X}_n| \leq B$ para algum $B > 0$, então

$$\widehat{\boldsymbol{\beta}} = \arg \min_{\boldsymbol{\beta}} \frac{1}{n} \sum_{i=1}^n (Y_i - \boldsymbol{\beta}^t \mathbf{X}_i)^2 \text{ sujeito a } \|\boldsymbol{\beta}\|_1 \leq L$$

(veja na Eq. 3.5 que este é justamente o estimador do lasso) é tal que, com probabilidade ao menos $1 - \delta$,

$$R(g_{\widehat{\boldsymbol{\beta}}}) - R(g_{\boldsymbol{\beta}_*}) \leq \sqrt{\frac{16(L+1)^4 B^2}{n} \log\left(\frac{\sqrt{2d}}{\sqrt{\delta}}\right)}.$$

Note que quanto menor o valor de L no Teorema 3.1, mais próximo o risco do estimador do lasso fica do risco do oráculo. Ou seja, mais fácil é se recuperar o melhor $\boldsymbol{\beta}$. Por outro lado, quanto menor o valor de L , pior é o oráculo.

3.4 Regressão Ridge

Uma alternativa ao lasso (que na realidade é anterior a ele) é a regressão ridge (Hoerl and Kennard, 1970). A ideia é novamente buscar o modelo que minimize a soma do erro quadrático médio de g com uma medida de sua esparsidade. Na regressão ridge isto é feito encontrando-se o estimador dado por

$$\widehat{\boldsymbol{\beta}}_{L_2, \lambda} = \arg \min_{\boldsymbol{\beta}} \sum_{k=1}^n \left(y_k - \beta_0 - \sum_{j=1}^d \beta_j x_{k,j} \right)^2 + \lambda \sum_{j=1}^d \beta_j^2, \quad (3.3)$$

em que L_2 indica o fato de que estamos medindo a esparsidade de um vetor $\boldsymbol{\beta}$ usando sua norma em L_2 , $\|\boldsymbol{\beta}\|_{L_2}^2 = \sum_{j=1}^d \beta_j^2$. Ao contrário do lasso, a regressão ridge possui solução analítica, dada por

$$\widehat{\boldsymbol{\beta}}_{L_2, \lambda} = (\widehat{\beta}_0, \widehat{\beta}_1, \dots, \widehat{\beta}_d) = (\mathbb{X}^t \mathbb{X} + \lambda \mathbb{I})^{-1} \mathbb{X}^t \mathbb{Y}. \quad (3.4)$$

Compare esta solução com a Equação 2.3. Apesar de não introduzir soluções com zeros como o lasso, a regressão ridge também diminui a variância dos estimadores da regressão pois *encolhe (shrinks)* os coeficientes β estimados pela regressão linear. Por exemplo, no caso em que as covariáveis originais são ortogonais (i.e., $\mathbb{X}^t \mathbb{X} = \mathbb{I}$), temos $\widehat{\boldsymbol{\beta}}_{L_2, \lambda} = \widehat{\boldsymbol{\beta}} / (1 + \lambda)$, em que $\widehat{\boldsymbol{\beta}}$ é o estimador de mínimos quadrados da Equação 2.3. Assim,

$$\mathbb{V} [\widehat{\beta}_{i, L_2, \lambda}] = \frac{\mathbb{V} [\widehat{\beta}_i]}{(1 + \lambda)^2}.$$

Evidentemente, como no lasso, apesar da variância da ridge regression ser menor, seu viés é maior. Assim, λ deve ser escolhido de modo a controlar o balanço viés-variância. Novamente, isso pode ser feito via validação cruzada.

Na Seção 4.6.3, estudamos uma extensão não linear da regressão ridge, a kernel ridge regression. Nela também mostramos como calcular facilmente a solução para todos os λ 's simultaneamente.

3.5 Formulação Alternativa

O lasso possui uma formulação alternativa. Pode-se mostrar que, para cada $\lambda \geq 0$, existe $B \geq 0$ a solução de 3.2 é dada por

$$\arg \min_{\beta} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_1 - \dots - \beta_d x_d)^2 \text{ sujeito a } \sum_{j=1}^d |\beta_j| \leq B. \quad (3.5)$$

Analogamente, as soluções das Equações 3.1 e 3.3 (AIC e Regressão Ridge, respectivamente) podem ser reexpressas como

$$\arg \min_{\beta} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_1 - \dots - \beta_d x_d)^2 \text{ sujeito a } \sum_{j=1}^d \mathbb{I}(\beta_j \neq 0) \leq B_1$$

e

$$\arg \min_{\beta} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_1 - \dots - \beta_d x_d)^2 \text{ sujeito a } \sum_{j=1}^d (\beta_j)^2 \leq B_2,$$

para algum B_1 e B_2 , respectivamente. Isso evidencia mais uma vez que, em todas as abordagens, a penalização favorece coeficientes “pequenos” quando comparados à solução de mínimos quadrados. Contudo, enquanto que o lasso e a penalização por AIC fazem seleção de variáveis, o mesmo não ocorre com a regressão ridge.

3.6 Interpretação Bayesiana

Tanto a regressão ridge quanto o lasso admitem uma interpretação sob o ponto de vista Bayesiano: os estimadores de β em ambos os procedimentos podem ser escritos como sendo a moda da distribuição a posteriori para β para uma dada distribuição a priori e verossimilhança. Mais especificamente, se assumimos que

$$Y_i | \mathbf{x}_i, \boldsymbol{\beta} \sim N(\boldsymbol{\beta}^t \mathbf{x}_i; \sigma^2 I_d),$$

com Y_1, \dots, Y_n independentes e σ^2 conhecido, então:

- Se $\boldsymbol{\beta} \sim N(\mathbf{0}, \sigma_\beta^2 I_d)$, a moda da distribuição a posteriori de $\boldsymbol{\beta}$ dados $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$ é justamente o estimador de $\boldsymbol{\beta}$ dado pela regressão ridge (Eq. 3.3) com $\lambda = \sigma^2 / \sigma_\beta^2$;
- Se β_1, \dots, β_d i.i.d. $\sim \text{Laplace}(\mathbf{0}, \tau_\beta)^3$, a moda da distribuição a posteriori de $\boldsymbol{\beta}$ dados $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$ é justamente o estimador de $\boldsymbol{\beta}$ dado pelo lasso (Eq. 3.2) com $\lambda = \sigma^2 \tau_\beta$.

Note que, sob a abordagem Bayesiana, o *tuning parameter* λ é definido pela distribuição a priori. Quanto menor a dispersão da priori, maior o valor de λ e, consequentemente, mais próximo de zero estarão as estimativas a posteriori. Assim, a abordagem Bayesiana evita o *overfitting*, por mais que esse não seja necessariamente seu objetivo primordial. O interessante é que isso ocorre justamente porque a priori centrada em $\mathbf{0}$ reflete a crença de que os parâmetros relativos à maioria das covariáveis, em geral, devem ser pequenos.

Isto leva a uma interpretação filosófica interessante: escolher λ por validação cruzada equivale a escolher a distribuição a priori que leva a um maior poder preditivo. Em outras palavras, usamos a maquinaria Bayesiana apenas para motivar estimadores com uma performance (preditiva) melhor que a dos mínimos quadrados.

Veja mais sobre a relação entre estimadores de Bayes e os estimadores descritos neste capítulo em, por exemplo, [Bishop \(2006\)](#) e [Park and Casella \(2008\)](#).

3.7 Regressão Ridge e Lasso no R

No R, ambos os métodos podem ser implementados via a biblioteca “glmnet”. Se x é a matriz com as covariáveis e y é a matriz com os valores da variável resposta, para ajustar o lasso basta fazer:

```
ajuste = cv.glmnet(x, y, alpha=1) # ajusta o modelo
                                    # calcula coeficientes
                                    # para diferentes lambdas,
                                    # a escolha do grid é automática
                                    # mas pode ser mudada
plot(ajuste) # plota lambda vs risco estimado
lambdaOtimo = validacaoCruzada$lambda.min # retorna
                                                # melhor lambda
```

³ Aqui, τ_β é um parâmetro de precisão.

```
coefficients(ajuste, s = lambdaOtimo) # melhor lambda
predict(ajuste, newx=xNovo, s = lambdaOtimo) # prediz Y para
# cada linha de xNovo
# usando melhor lambda
```

Note que a validação cruzada já é feita automaticamente pela função `cv.glmnet`.

A regressão ridge pode ser ajustada usando-se essas mesmas funções, mas usamos `alpha=0` ao invés de `alpha=1`.

3.8 Exemplos

Exemplo 3.1 Geramos $n = 500$ observações i.i.d. segundo

$$Y_k = 3X_{k,1} - 2X_{k,2} + X_{k,3} - 3X_{k,4} + X_{k,5} + \sum_{i=6}^{20} 0X_{k,i} + \epsilon_k,$$

com $\epsilon_k \sim N(0, 0.5^2)$ e $X_{k,i} \sim N(0, 1)$, $i = 1, \dots, 20$ independentes.

A Tabela 3.2 mostra os resultados encontrados. Pode-se observar que o método dos mínimos quadrados com todas as variáveis foi rápido de ser ajustado, mas têm poder preditivo muito baixo (o risco estimado é alto). Por outro lado, todos os métodos de seleção de variáveis vistos neste texto possuem poder preditivo muito maior para este exemplo. A heurística do forward stepwise fornece os mesmos resultados que a da busca pelo mínimo da Equação 3.1 (critério AIC), com a vantagem de levar um tempo substancialmente menor. Por outro lado, o lasso forneceu um risco ainda menor em um intervalo de tempo ainda mais curto; em particular as variáveis selecionadas foram as que realmente influenciam Y . A regressão ridge foi substancialmente melhor que o método de mínimos quadrados com todas as covariáveis, mas levemente pior que os demais métodos. Contudo, teve tempo de execução menor que o forward stepwise.

Tabela 3.2: Resultados dos métodos de seleção de variáveis no exemplo da Seção 3.8. *: Busca pelo melhor subconjunto.

Método	Variáveis Selecionadas	Tempo de ajuste	Risco Estimado
Mínimos Quadrados	Todas	0.002 segundos	14.63 (0.02)
Melhor AIC*	$x_1, x_2, x_3, x_4, x_5, x_{10}, x_{12}, x_{13}, x_{19}, x_{20}$	1 hora e 20 minutos	0.30 (0.02)
Forward Stepwise	$x_1, x_2, x_3, x_4, x_5, x_{10}, x_{12}, x_{13}, x_{19}, x_{20}$	0.46 segundos	0.30 (0.02)
Ridge	Todas	0.19 segundos	0.33 (0.03)
Lasso	x_1, x_2, x_3, x_4, x_5	0.08 segundos	0.25 (0.02)

□

Exemplo 3.2 (Câncer de Próstata) *Esses dados são provenientes de um estudo publicado em Stamey et al. (1989) e foram descritos por James et al. (2013). Nesse trabalho estudaram a correlação entre o nível de antígeno específico de próstata (PSA) e algumas medidas clínicas. Foram observados dados de 97 homens que estavam para fazer prostatectomia radical. O objetivo era prever o logaritmo da PSA (lpsa) com base nas demais variáveis.*

Para a ilustração das técnicas de mínimos quadrados, Ridge e Lasso, utilizamos a separação do banco em treinamento e validação definida por TIBS. Foram considerados dados de 67 indivíduos para treinamento e os 30 restantes para validação.

```
library(dplyr)
library(glmnet)
library(plotmo)

set.seed(1)
dados = read.table("data/prostate.data")

# treinamento
tr = dados$train

dados[,1:8] = apply(dados[,1:8], 2, scale, center = TRUE,
                     scale = TRUE)
dados       = dados[,1:9]

# conjunto treinamento
xtr = as.matrix(dados[tr,1:8])
ytr = dados$lpsa[tr]

# conjunto validação
xval = as.matrix(dados[!tr,1:8])
yval = dados$lpsa[!tr]
```

A seguir iremos ajustar o modelo de mínimos quadrados, ridge e lasso.

```
# Mínimos Quadrados
ajuste_mq = lm(lpsa ~ ., data = dados[tr,])
```

```

predito_mq = predict.lm(ajuste_mq,
                       newdata = as.data.frame(dados[!tr, 1:8]))

# Regressão Ridge
vc_ridge      = cv.glmnet(xtr, ytr, alpha = 0)
ajuste_ridge  = glmnet(xtr, ytr, alpha = 0)
predito_ridge = predict(ajuste_ridge, s = vc_ridge$lambda.1se,
                       newx = xval)

# Regressão LASSO
vc_lasso      = cv.glmnet(xtr, ytr, alpha = 1)
ajuste_lasso  = glmnet(xtr, ytr, alpha = 1)
predito_lasso = predict(ajuste_lasso, s = vc_lasso$lambda.1se,
                       newx = xval)

```

Para a regressão Ridge e Lasso, é interessante utilizar o gráfico dos coeficientes em função de λ ou $\log(\lambda)$. As Figuras 3.3 e 3.4 ilustram o comportamento dos coeficientes estimados de acordo com λ para as duas técnicas.

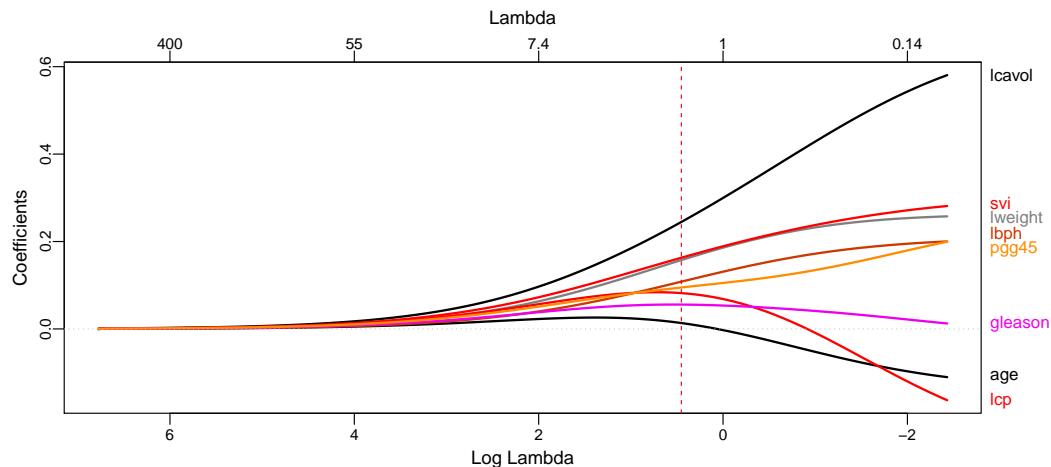


Figura 3.3: Coeficientes da regressão Ridge estimados em função de λ ou $\log(\lambda)$. A reta vertical indica o valor escolhido via validação cruzada.

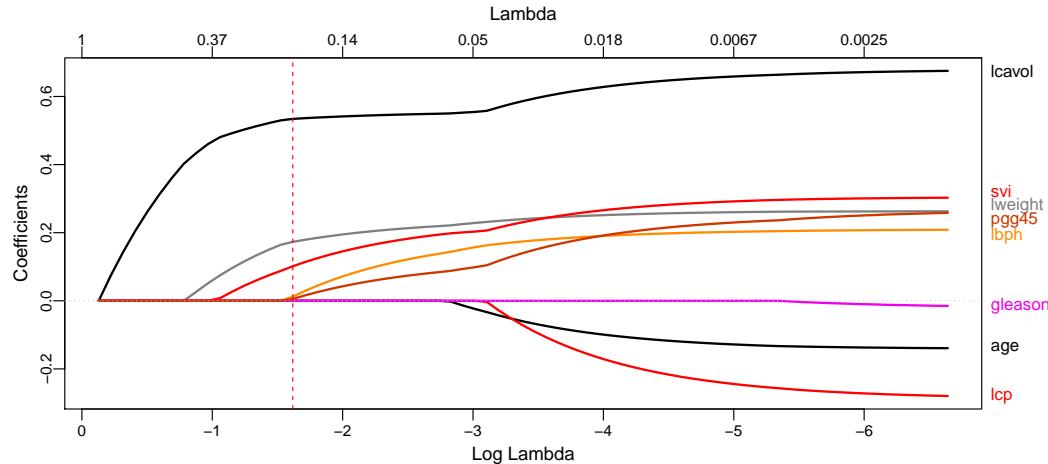


Figura 3.4: Coeficientes da regressão Lasso estimados em função de λ ou $\log(\lambda)$. A reta vertical indica o valor escolhido via validação cruzada.

A Tabela 3.3 apresenta os coeficientes estimados com a utilização das três técnicas. Note a redução dos coeficientes obtidos por mínimos quadrados com estimação dos modelos Ridge e Lasso. Ainda, note que excluímos três variáveis (age, lcp e gleason) com a utilização do modelo Lasso.

Tabela 3.3: Coeficientes estimados para os dados da próstata considerando mínimos quadrados, ridge e lasso.

	Mínimos Quadrados	Ridge	Lasso
(Intercept)	2.465	2.456	2.468
lcavol	0.680	0.245	0.534
lweight	0.263	0.158	0.173
age	-0.141	0.014	-
lbph	0.210	0.108	0.012
svi	0.305	0.163	0.101
lcp	-0.288	0.082	-
gleason	-0.021	0.056	-
pgg45	0.267	0.095	0.006

A Tabela 3.4 apresenta o desempenho preditivo dos modelos obtidos com os dados de treinamento quando aplicados aos dados de validação dos 30 indivíduos. Assim, notamos que a regressão Lasso apresentou o melhor desempenho preditivo e o menor erro padrão. A regressão Ridge apresentou um desempenho intermediário entre a performance obtida com Lasso e mínimos quadrados. Já o desempenho obtido com o modelo de mínimos quadrados apresentou a pior performance preditiva e maior variabilidade.

Tabela 3.4: Desempenho preditivo em relação ao conjunto de validação dos modelos de mínimos quadrados, ridge e lasso.

	EQM	EP
Mínimos Quadrados	0.5213	0.1787
Ridge	0.5492	0.2001
Lasso	0.4731	0.1620

□

Exemplo 3.3 (Amazon Fine Food Reviews) Nesse exemplo mostramos como fazer previsões de notas dadas em resenhas da Amazon com base no conteúdo (texto) dela. Para isso, selecionaremos um subconjunto do banco de dados *Amazon Fine Food Reviews*⁴. Esse banco conta com aproximadamente meio milhão de resenhas. Para ilustração das técnicas apresentadas anteriormente, separamos 70.000 observações para treinamento dos modelos e 20.000 para validação.

```
library(data.table)
library(tm)
library(glmnet)

dados = fread("data/Reviews.csv", header = TRUE)

# seleciona 70.000 observações
set.seed(1)
selecao = sample(nrow(dados), 70000)
dados = dados[selecao,]

# indica observações de treinamento
tr = sample.int(70000, 50000, replace = F)
```

⁴ (<https://www.kaggle.com/snap/amazon-fine-food-reviews>).

```

corp = VCorpus(VectorSource(dados$Text))

dtm = DocumentTermMatrix(corp,
    control = list(tolower = TRUE,
                    stemming = FALSE,
                    removeNumbers = TRUE,
                    removePunctuation = TRUE,
                    removeStripwhitespace = TRUE,
                    weighting = weightTf,
                    bounds=list(global=c(50, Inf)))))

dtmMatrix = sparseMatrix(i = dtm$i, j = dtm$j, x = dtm$v,
    dimnames = list(NULL, dtm$dimnames[[2]]),
    dims = c(dtm$nrow, dtm$ncol))

dim(dtmMatrix)
## [1] 70000 4537

```

Assim, consideraremos nesse exemplo um total de 4.537 palavras. É importante notar que não é necessário converter a matriz de dados para o formato usual com todos os elementos preenchidos. Ela pode ser utilizada de forma esparsa (veja a Seção A.3 para mais detalhes) e, assim, economizar uma relativa quantidade de memória. Note que, com a utilização do formato esparso, é possível trabalhar com o conjunto completo de meio milhão de resenhas.

A seguir iremos considerar o modelo de mínimos quadrados, ridge e lasso.

```

# Mínimos Quadrados
ajuste_mq = glmnet(dtmMatrix[tr,], dados$Score[tr], alpha = 0,
                    lambda = 0)
predito_mq = predict(ajuste_mq, newx = dtmMatrix[-tr,])

# Ridge
vc_ridge      = cv.glmnet(dtmMatrix[tr,], dados$Score[tr],
                           alpha = 0)
ajuste_ridge  = glmnet(dtmMatrix[tr,], dados$Score[tr],
                           alpha = 0)

```

```

predito_ridge = predict(ajuste_ridge, s = vc_ridge$lambda.1se,
                        newx = dtmMatrix[-tr,])

# LASSO
vc_lasso      = cv.glmnet(dtmMatrix[tr,], dados$Score[tr],
                           alpha = 1)
ajuste_lasso = glmnet(dtmMatrix[tr,], dados$Score[tr],
                           alpha = 1)
predito_lasso = predict(ajuste_lasso, s = vc_lasso$lambda.1se,
                        newx = dtmMatrix[-tr,])

```

Comparando o erro quadrático médio e o respectivo erro padrão, notamos que a regressão LASSO apresentou o melhor desempenho preditivo. A regressão Ridge apresentou um desempenho mais próximo ao desempenho obtido com a regressão LASSO. Já a regressão com mínimos quadrados apresentou um desempenho inferior às duas técnicas citadas anteriormente.

Tabela 3.5: Desempenho preditivo em relação ao conjunto de validação dos modelos de mínimos quadrados, ridge e lasso.

	EQM teste	EP teste
Mínimos Quadrados	1.08	0.01
Ridge	1.02	0.01
Lasso	1.01	0.01

Uma forma de visualizar quais palavras apresentam maiores pesos para a previsão é com a utilização de gráficos de barras. Nesses gráficos consideramos os valores positivos e negativos dos coeficientes estimados ordenados. Para esse exemplo selecionamos os 20 maiores de cada para a regressão Ridge e Lasso.

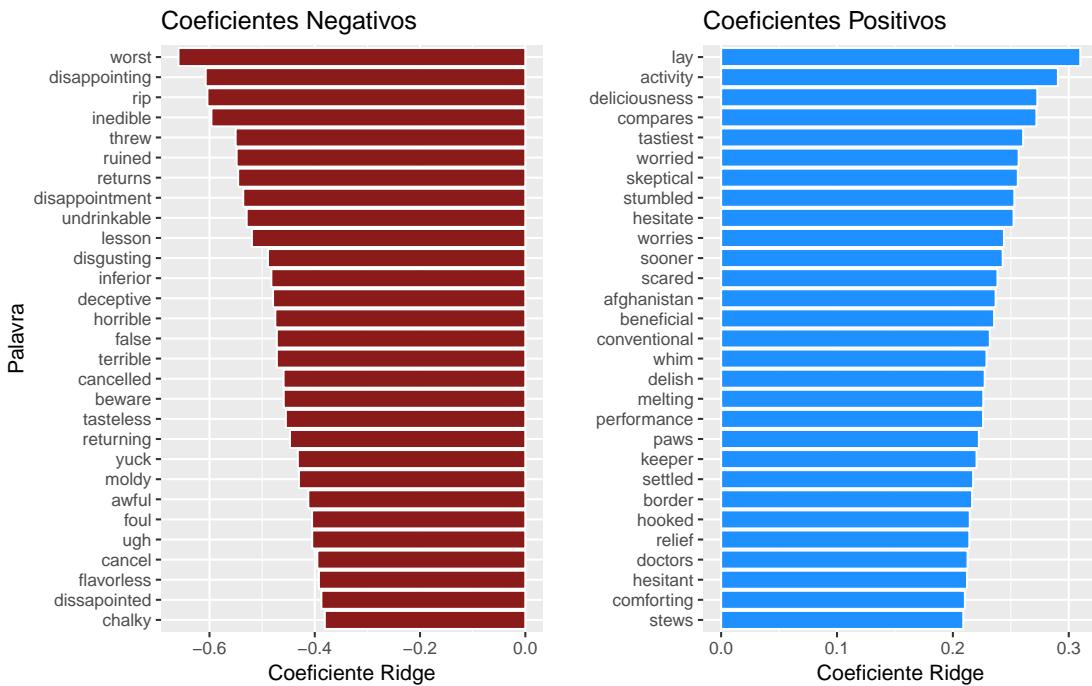


Figura 3.5: Coeficientes com os 20 maiores valores da regressão ridge.

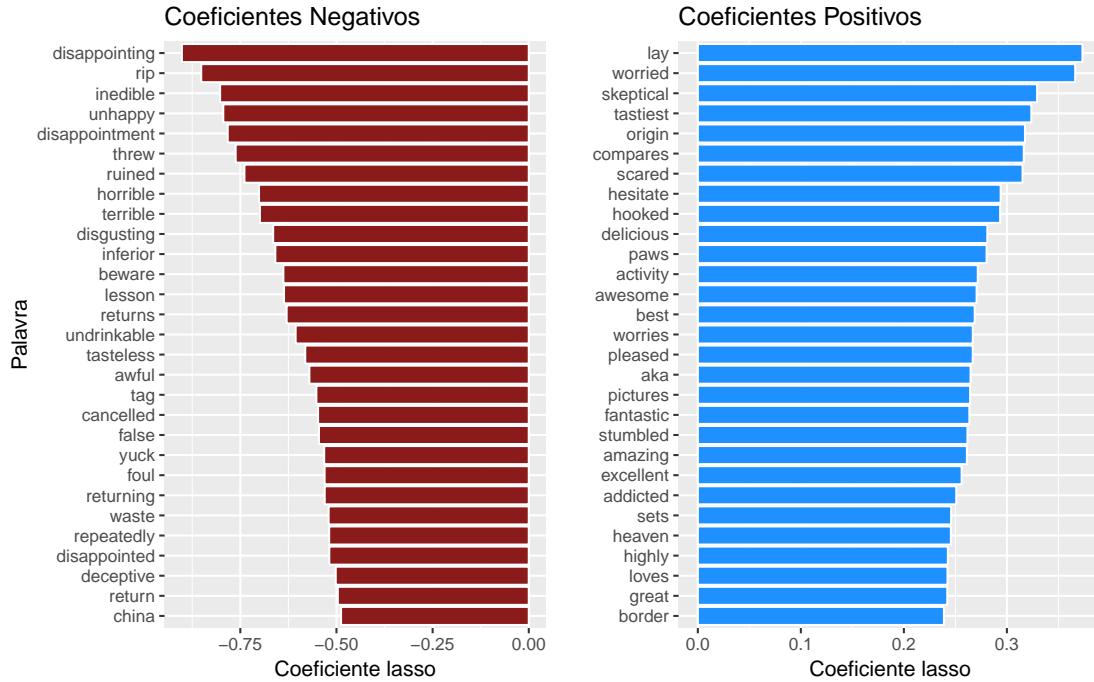


Figura 3.6: Coeficientes com os 20 maiores valores da regressão Lasso.

□

3.9 Resumo

Neste capítulo vimos que, mesmo quando há muitas covariáveis, muitas vezes é possível criar bons estimadores de $r(\mathbf{x})$ com base em uma regressão linear. A chave para isso é usar o fato de que, frequentemente, muitas variáveis tem pouca influência na resposta. Assim, adicionando um termo de penalização à soma de quadrados (Eq. 3.1), criamos estimadores dos coeficientes β que encorajam *esparsidade*, isto é, muitos zeros na solução. Assim, criamos estimadores com uma variância menor. A esperança é que, se o aumento no viés por retirar essas variáveis não é muito grande, o risco também diminui e, assim conseguimos um estimador melhor para $r(\mathbf{x})$ que aquele dado pelo método dos mínimos quadrados, como mostra a Figura 3.7.

Vimos três formas de se fazer a penalização:

- Penalidade L_0 : $\sum_{i=1}^d \mathbb{I}(\beta_i \neq 0)$. Computacionalmente, achar o mínimo consome um tempo computacional extremamente alto pois deve-se buscar o melhor entre os 2^d

subconjuntos de covariáveis. Assim, é necessário usar uma heurística como *stepwise* para aproximar este valor.

- Penalidade $L_1: \sum_{i=1}^d |\beta_i|$. Trata-se do *lasso*. Esta penalização é rápida de ser implementada e induz esparsidade em β . Frequentemente ela tem uma performance muito boa.
- Penalidade $L_2: \sum_{i=1}^d \beta_i^2$. Trata-se da *regressão ridge*. Esta penalização não induz zeros em β , mas reduz a variância do estimador da regressão pois “encolhe” o estimador de mínimos quadrados.

Métodos paramétricos muitas vezes impõem muitas limitações para $r(\mathbf{x})$: por exemplo, nem sempre o melhor estimador linear é um bom estimador para $r(\mathbf{x})$. No próximo capítulo iremos introduzir métodos não paramétricos para $r(\mathbf{x})$, que são modelos mais flexíveis que métodos paramétricos. Em linhas gerais, métodos paramétricos diminuem o viés de métodos paramétricos (em troca de um aumento na variância), como indica a Figura 3.7.



Figura 3.7: Relação entre métodos paramétricos com penalização (esquerda), sem penalização (centro) e métodos não-paramétricos (direita).

Capítulo 4

Métodos Não Paramétricos

Para amostras pequenas, métodos paramétricos costumam levar a bons resultados: impondo-se um modelo com poucos parâmetros, é possível criar estimadores com baixa variância mesmo se n é pequeno. Contudo, quando temos n grande, podemos muitas vezes aumentar o número de parâmetros do modelo, de modo que tenhamos um viés muito menor à custa, claro, de uma variância um pouco maior. É exatamente aí que métodos não paramétricos ganham importância: informalmente, um modelo não paramétrico é um modelo que tem *infinitos parâmetros*. Neste capítulo exploramos alguns destes modelos.

4.1 Séries Ortogonais

Métodos baseados em séries ortogonais são bastante antigos (Chentsov, 1962) e baseiam-se em uma ideia bastante simples: na expansão da função de regressão em termos de uma base ortogonal. Para facilitar a discussão, vamos assumir por hora que $\mathbf{x} \in [0, 1]$ e $f(\mathbf{x}) = 1$.

O ponto inicial é escolher $(\psi_j(\mathbf{x}))_{j \in \mathbb{N}}$, uma base ortonormal¹ para o conjunto de funções

$$L^2([0, 1]) := \left\{ f : [0, 1] \longrightarrow \mathbb{R} : \int_0^1 f^2(\mathbf{x}) d\mathbf{x} < \infty \right\}.$$

Um exemplo de $(\psi_j(\mathbf{x}))_{j \in \mathbb{N}}$ é a *base de Fourier*:

$$\psi_0(\mathbf{x}) = 1; \quad \psi_{2j-1}(\mathbf{x}) = \sqrt{2} \sin(2\pi j \mathbf{x}), \quad j \in \mathbb{N}; \quad \psi_{2j}(\mathbf{x}) = \sqrt{2} \cos(2\pi j \mathbf{x}), \quad j \in \mathbb{N}$$

¹ Isto é, $\int_0^1 \psi_i(\mathbf{x}) \psi_j(\mathbf{x}) d\mathbf{x} = \mathbb{I}(i = j)$.

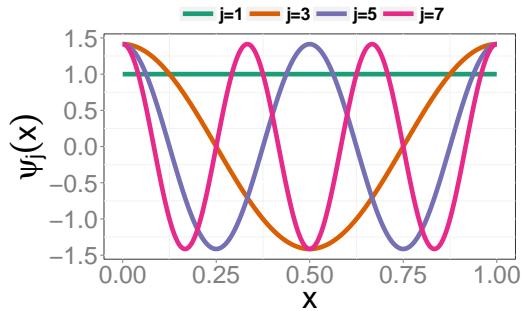


Figura 4.1: Alguns elementos da base de Fourier. Os primeiros termos são mais suaves que os termos de ordem mais alta. Qualquer função integrável em L^2 pode ser escrita como combinação linear dos elementos da base de Fourier.

A ideia central de métodos baseados em séries é que, se a função de regressão $r(\mathbf{x})$ pertence a $L^2([0, 1])$, então $r(\mathbf{x})$ pode ser representada como

$$r(\mathbf{x}) = \sum_{j \in \mathbb{N}} \beta_j \psi_j(\mathbf{x}),$$

em que, devido à ortonormalidade da base $(\psi_j(\mathbf{x}))_{j \in \mathbb{N}}$, temos que os coeficientes da expansão admitem uma forma muito simples:

$$\beta_j = \int r(\mathbf{x}) \psi_j(\mathbf{x}) d\mathbf{x} = \int \mathbb{E}[Y|\mathbf{x}] \psi_j(\mathbf{x}) f(\mathbf{x}) d\mathbf{x} = \int \mathbb{E}[Y \psi_j(\mathbf{x})|\mathbf{x}] f(\mathbf{x}) d\mathbf{x} = \mathbb{E}[Y \psi_j(\mathbf{X})].$$

Um estimador não viésado para os *coeficientes da expansão* β_j é dado por

$$\hat{\beta}_j = \frac{1}{n} \sum_{i=1}^n Y_i \psi_j(\mathbf{X}_i).$$

Assim, o estimador baseado em séries tem a seguinte forma:

$$\hat{r}(\mathbf{x}) = \sum_{j=0}^J \hat{\beta}_j \psi_j(\mathbf{x}). \quad (4.1)$$

O parâmetro J é um *tuning parameter* que controla o balanço viés-variância. Valores alto de J levam a um viés baixo, mas variância alta.

Observação 4.1 Se $r(\mathbf{x})$ pertence a $L^2([0, 1])$, sabemos que $\beta_j \rightarrow 0$ quando $j \rightarrow \infty$, o que justifica o truncamento da Equação 4.1.

□

Note que o número de parâmetros, $(\beta_j)_{j \in \mathbb{N}}$, é infinito, por isso o método de séries é dito ser um método não paramétrico.

Se $d > 1$, é mais complexo construir $\{\psi_j(\mathbf{x})\}_j$. Tipicamente, isso é feito usando-se d *produtos tensoriais* para cada uma das coordenadas de \mathbf{x} . Por exemplo, se $d = 2$, a base dada por produtos tensoriais é

$$\{\psi_{i,j}(\mathbf{x}) = \psi_i(x_1)\psi_j(x_2) : i, j \in \mathbb{N}\},$$

em que $\mathbf{x} = (x_1, x_2)$, e $\{\psi_i(\mathbf{x}_1)\}_i$ e $\{\psi_j(\mathbf{x}_2)\}_j$ são bases para $\mathfrak{L}^2(\mathbb{R})$. Esta é uma base para $\mathfrak{L}^2(\mathbb{R}^2)$ (Wasserman, 2006).

Note que a abordagem de séries ortogonais é essencialmente a mesma que aquela da regressão linear com um número variável de covariáveis, com a exceção de que o estimador de cada coeficiente β_i é mais rapidamente calculado devido à ortogonalidade das funções de base. Com a finalidade de deixar essa conexão mais clara, considere o seguinte exemplo. Pela expansão de Taylor, sabemos que qualquer função analítica (Krantz and Parks, 2002) pode ser expressa como

$$r(x) = \sum_{i \geq 0} \beta_i x^i.$$

Essa é justamente a família de modelos usada no Exemplo 1.6. Assim, o modelo ajustado naquele exemplo foi um modelo não paramétrico, apesar de que os coeficientes da expansão foram ajustados via regressão linear. O que torna o modelo não paramétrico é justamente o fato de que o número de coeficientes usados, J (que naquele exemplo é denotado por p) é um *tuning parameter* que pode assumir qualquer valor em $\{0, 1, \dots\}$. Assim, há um número infinito de parâmetros². Finalmente, notamos que é possível tornar a base de polinômios ortonormal via o processo de ortogonalização de Gram-Schmidt. A base resultante é chamada de base de Hermite (Efromovich, 1999).

Para outras abordagens de estimação com séries ortogonais, veja Efromovich (1999), Beran (2000) e Wasserman (2006). Em particular, tais referências discutem o uso de outras bases além da base de Fourier, como por exemplo Wavelets. Veja também a Seção 4.12 para um pouco de teoria sobre estimadores baseados em séries.

O método de séries pode ser implementado usando-se uma base polinomial via função `lm`, descrita no Capítulo 2:

```
ajuste = lm(y ~ poly(as.matrix(covariaveis), degree=5))
```

A função `poly` automaticamente calcula todos os polinômios (neste caso com grau até 5) em \mathbb{R}^d .

² Note que quando escolhemos $J = J_0$, estamos estimando $\beta_j = 0, \forall j > J_0$.

4.2 Splines

Vamos assumir novamente que $d = 1$. Assim como métodos baseados em séries ortogonais, em splines aproximamos $r(x)$ como uma combinação linear $\sum_{i=1}^I \beta_i f_i(x)$. Contudo, ao invés de usar uma base ortonormal, utiliza-se uma base para um espaço de *splines*.

Para descrevermos o que é um spline, fixe $t_1 < \dots < t_p$. Chamamos tais pontos de *nós*. Um spline de k -ésima ordem é uma função polinomial por partes de grau k que é contínua e tem derivadas de ordem $1, \dots, k-1$ contínuas em seus nós. Como um spline é bem mais suave que uma função que é polinomial por partes (mas sem as restrições de continuidade), ele consegue aproximar de forma mais eficiente funções contínuas³.

Há diversas formas de se definir uma base para tal espaço. Uma delas é através das seguintes funções:

$$\begin{aligned} f_1(x) &= 1; \quad f_2(x) = x; \dots; \quad f_{k+1}(x) = x^k, \\ f_{k+1+j}(x) &= (x - t_j)_+^k, \quad j = 1, \dots, p \end{aligned}$$

Ou seja, com essa base, $I = k + 1 + p$. Há, contudo, muitas outras formas de definir uma base para este espaço. Existem também muitas variações de splines, como natural splines ou smoothing splines (veja, por exemplo, [Hastie et al. 2009a](#) e a Seção 4.6.3.1).

Fixados os nós, pode-se então utilizar o método de mínimos quadrados para estimar os valores dos coeficientes β_1, \dots, β_I .

No R, splines podem ser usados via a biblioteca `splines`:

```
fit = lm(y~bs(x, degree=3), data = dados)
pred = predict(fit, newdata = list(x=x.grid), se = T)
```

Este código ajusta B -splines. Utilizando-se `ns`, pode-se ajustar natural splines da mesma maneira. Para smoothing splines, pode-se usar

```
fit = smooth.spline(x, y, df=3)
```

4.3 Método dos k Vizinhos Mais Próximos

O método dos k vizinhos mais próximos (em inglês, *k -nearest neighbours*, KNN) ([Benedetti, 1977](#), [Stone, 1977](#)) é um dos mais populares na comunidade de aprendizado de

³ Por exemplo, se r é uma função duas vezes diferenciável em $[0, 1]$, então existe um spline cúbico f com nós em $t_1 < \dots < t_p$ tal que $\sup_{x \in [0, 1]} |r(x) - f(x)| \leq O\left(\frac{1}{p}\sqrt{\int_0^1 r''(x)^2 dx}\right)$ ([De Boor, 1978](#)).

máquina. Ele tem como base estimar a função de regressão $r(\mathbf{x})$ para uma dada configuração das covariáveis \mathbf{x} com base nas respostas Y dos k -vizinhos mais próximos a \mathbf{x} . Formalmente, definimos

$$\hat{r}(\mathbf{x}) = \frac{1}{k} \sum_{i \in \mathcal{N}_{\mathbf{x}}} y_i, \quad (4.2)$$

em que $\mathcal{N}_{\mathbf{x}}$ é o conjunto das k observações mais próximas de \mathbf{x} , i.e.,

$$\mathcal{N}_{\mathbf{x}} = \{i \in \{1, \dots, n\} : d(\mathbf{x}_i, \mathbf{x}) \leq d_{\mathbf{x}}^k\}$$

e $d_{\mathbf{x}}^k$ é a distância do k -ésimo vizinho mais próximo de \mathbf{x} a \mathbf{x} . Em palavras, a função de regressão avaliada em \mathbf{x} é estimada utilizando-se uma *média local* das respostas dos k vizinhos mais próximos a \mathbf{x} no espaço das covariáveis.

O *tuning parameter* k novamente pode ser escolhido via validação cruzada. Um valor alto de k leva a um modelo muito simples (uma constante quando $k \rightarrow \infty$) e, assim um viés alto, mas uma variância baixa. Por sua vez, um valor baixo para k leva a um estimador com variância alta, mas viés baixo. Veja a Figura 4.2 para uma ilustração.

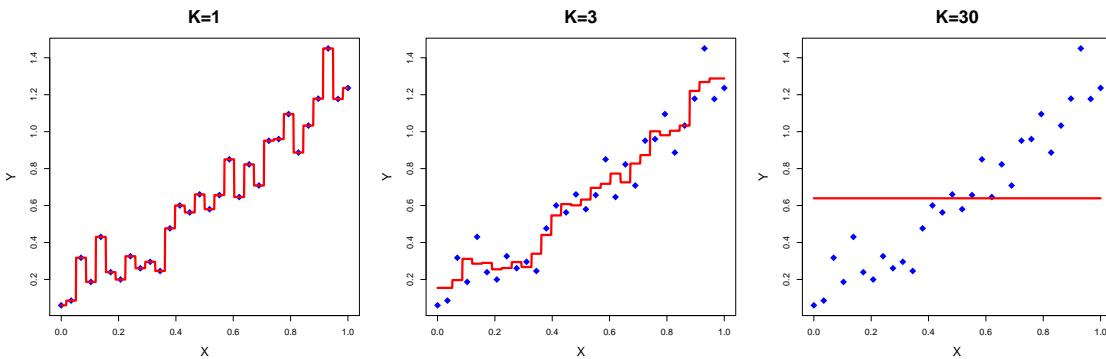


Figura 4.2: Influência na escolha de k no estimador dos k vizinhos mais próximos.

O método dos vizinhos mais próximos pode ser implementando usando-se o pacote FNN. A função `knn.regr` faz a predição para novas observações com covariáveis `xNovo` (uma observação por linha) utilizando um conjunto de treinamento com covariáveis `xTreinamento`, respostas `yTreinamento` e k vizinhos através do seguinte comando:

```
library(FNN)
ajuste = knn.reg(train=xTreinamento, test=xNovo, y=yTreinamento,
                    k=k)
predVal=ajuste$pred
```

4.4 Nadaraya-Watson

O método de Nadaraya-Watson, criado por [Nadaraya \(1964\)](#) e [Watson \(1964\)](#) é uma variação do método dos k -vizinhos mais próximos bastante difundida na comunidade de estatística. Este método baseia-se em estimar a função de regressão em um dado ponto \mathbf{x} utilizando-se uma média ponderada das observações do conjunto de treinamento:

$$\hat{r}(\mathbf{x}) = \sum_{i=1}^n w_i(\mathbf{x})y_i,$$

em que $w_i(\mathbf{x})$ é um peso atribuído à i -ésima observação, e que mede o quanto similar \mathbf{x}_i é a \mathbf{x} . Mais especificamente, $w_i(\mathbf{x})$ tem a forma

$$w_i(\mathbf{x}) = \frac{K(\mathbf{x}, \mathbf{x}_i)}{\sum_{j=1}^n K(\mathbf{x}, \mathbf{x}_j)},$$

em que $K(\mathbf{x}, \mathbf{x}_i)$ é um *kernel* usado para medir a similaridade entre as observações. Algumas escolhas populares para $K(\mathbf{x}, \mathbf{x}_i)$ são:

- $K(\mathbf{x}, \mathbf{x}_i) = \mathbb{I}(d(\mathbf{x}, \mathbf{x}_i) \leq h)$ (kernel uniforme)
- $K(\mathbf{x}, \mathbf{x}_i) = (\sqrt{2\pi h^2})^{-1} \exp\left\{-\frac{d^2(\mathbf{x}, \mathbf{x}_i)}{2h^2}\right\}$ (kernel gaussiano)
- $K(\mathbf{x}, \mathbf{x}_i) = (1 - d(\mathbf{x}, \mathbf{x}_i)/h)\mathbb{I}(d(\mathbf{x}, \mathbf{x}_i) \leq h)$ (kernel triangular)
- $K(\mathbf{x}, \mathbf{x}_i) = (1 - d^2(\mathbf{x}, \mathbf{x}_i)/h^2)\mathbb{I}(d(\mathbf{x}, \mathbf{x}_i) \leq h)$ (kernel de Epanechnikov)

Enquanto que o kernel uniforme atribui o mesmo peso para cada observação a uma distância menor que h de \mathbf{x} e peso zero para observações a uma distância maior que h (um *tuning parameter*), os kernels triangulares e de Epanechnikov atribuem pesos diferentes de acordo com a distância até \mathbf{x} : observações mais próximas recebem peso maior. O kernel gaussiano, por sua vez, atribui peso positivo para *todas* as observações do conjunto de treinamento (novamente, observações mais próximas recebem peso maior). Outros exemplos de kernel podem ser encontrados em http://en.wikipedia.org/wiki/Kernel_%28statistics%29.

Note que um valor alto do *tuning parameter* h leva um estimador da função de regressão com variância baixa e viés alto, já que o mesmo peso é atribuído para cada amostra \mathbf{x}_i neste caso. Por sua vez, h baixo leva a uma estimador com variância alta, mas viés baixo. Veja a Figura 4.3 para um exemplo com o kernel uniforme.

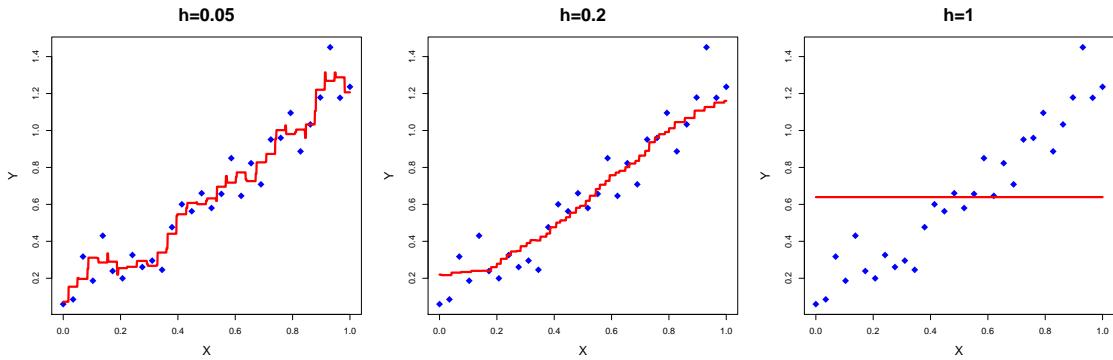


Figura 4.3: Influência na escolha de h no estimador de Nadaraya-Watson. Aqui o kernel utilizado foi o kernel uniforme.

Na prática, muitas vezes é observado que a escolha do kernel não influencia muito os resultados; a escolha dos *tuning parameters* associados a ele sim. Veja um exemplo de comparação de dois kernels na Figura 4.4.

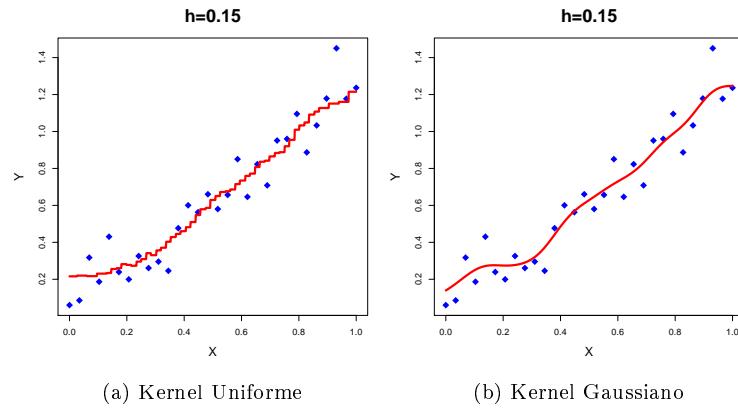


Figura 4.4: Influência na escolha do kernel no estimador de Nadaraya-Watson.

Pode-se verificar que o estimador de Nadaraya Watson em um dado \mathbf{x} fixo é justamente o ponto β_0 que minimiza

$$\sum_{i=1}^n w_i(\mathbf{x}) (Y_i - \beta_0)^2,$$

isto é,

$$\hat{r}(\mathbf{x}) := \hat{\beta}_0 = \arg \min_{\beta_0} \sum_{i=1}^n w_i(\mathbf{x}) (Y_i - \beta_0)^2, \quad (4.3)$$

Assim, tal estimador pode ser entendido como um estimador de mínimos quadrados *ponderado*, onde consideramos uma função de regressão que contém somente o intercepto, $\hat{r}(\mathbf{x}) = \hat{\beta}_0$. Note que \mathbf{x} está fixo; para cada \mathbf{x} temos um valor de $\hat{\beta}_0$ diferente, i.e., uma outra regressão linear. Esta motivação para o estimador de Nadaraya-Watson leva a uma extensão de tal método, conhecida como regressão polinomial local, que descrevemos na Seção 4.5.

Uma forma de se ajustar o estimador de Nadaraya-Watson é usando o pacote `locfit`. Pode-se mudar o kernel utilizado via o argumento `kern`.

```
ajuste=locfit(nomeResposta ~ nomeExplicativa1+nomeExplicativa2,
               alpha=c(0, 0.3), deg=0,
               data=dados)
valoresPrevistos= predict(ajuste, newdata=novosDados)
```

4.5 Regressão Polinomial Local

A regressão polinomial local (Stone, 1977, Cleveland, 1979, Fan, 1993, Fan and Gijbels, 1996) é uma extensão do estimador de Nadaraya-Watson. Suponha por simplicidade que há apenas $d = 1$ covariável. A ideia chave é que, ao invés de buscar um estimador baseado no método de mínimos quadrados ponderados com apenas um intercepto como na Equação 4.3, utilizamos um polinômio de grau p :

$$\hat{r}(\mathbf{x}) := \hat{\beta}_0 + \sum_{j=1}^p \hat{\beta}_j \mathbf{x}^j,$$

em que $\hat{\beta}_0, \dots, \hat{\beta}_p$ são dados por

$$\arg \min_{\beta_0, \dots, \beta_p} \sum_{i=1}^n w_i(\mathbf{x}) \left(Y_i - \beta_0 - \sum_{j=1}^p \beta_j \mathbf{x}_i^j \right)^2.$$

Aqui, novamente, $w_i(\mathbf{x}) = \frac{K(\mathbf{x}, \mathbf{x}_i)}{\sum_{j=1}^n K(\mathbf{x}, \mathbf{x}_j)}$. Note que, para cada \mathbf{x} de interesse, as estimativas dos coeficientes β_0, \dots, β_p é outra, ou seja este estimador consiste em usar o método de mínimos quadrados *localmente*. A solução para tal problema é dada por

$$(\hat{\beta}_0 \ \dots \ \hat{\beta}_p) = (B^t \Omega B)^{-1} B^t \Omega y,$$

em que B é uma matriz n por $p+1$ cuja i -ésima linha é dada por $(1 \ \mathbf{x}_i \ \dots \ \mathbf{x}_i^p)$, e Ω é uma matriz diagonal n por n cujo i -ésimo elemento da diagonal é $w_i(\mathbf{x})$.

A grande vantagem de se utilizar polinômios ao invés de apenas um intercepto é que isso induz estimadores com vieses menores (em particular, o viés em regiões próximas ao limite do domínio dos dados; veja, por exemplo, [Wasserman 2006](#)). Por outro lado, a variância de tais estimadores é maior, de modo que é importante selecionar o grau do polinômio a ser utilizado adequadamente.

Para regressão polinomial local com mais de uma covariável (i.e., $d > 1$) remetemos o leitor a [Fan and Gijbels \(1996\)](#).

Polinômios locais também podem ser ajustados no R usando o pacote `locfit`, como o estimador de Nadaraya-Watson ([Seção 4.4](#)). Para isso, basta modificar o valor do argumento `deg`, que representa o grau do polinômio a ser utilizado. Note que 0 corresponde ao estimador de Nadaraya-Watson.

4.6 Penalização em RKHSs

Métodos de estimação da função de regressão com base em penalização em RKHSs (*Reproducing Kernel Hilbert Spaces*, [Aronszajn 1950](#)) ([Kimeldorf and Wahba, 1970](#), [Hastie et al., 2009a](#), [Nosedal-Sánchez et al., 2012](#)) são uma família de métodos bastante geral.

Fundamentalmente, a ideia é chave é definir uma função objetivo para quantificar a qualidade de uma função de predição e então buscar pela função que tem o melhor ajuste segundo esse critério dentro de um espaço \mathcal{H} . Assim, busca-se pela solução de

$$\arg \min_{g \in \mathcal{H}} \sum_{k=1}^n L(g, (\mathbf{x}_k, y_k)) + \mathcal{P}(g),$$

em que L é uma função de perda arbitrária e \mathcal{P} é uma medida de complexidade de g e \mathcal{H} é um espaço de funções. Para um espaço \mathcal{H} arbitrário, a solução para esse problema é difícil, uma vez que se trata de um problema de otimização sobre um espaço de funções. RKHSs permitem descrever uma grande família de espaços \mathcal{H} (chamadas de Reproducing Kernel Hilbert Spaces) de modo que a solução do problema de otimização seja relativamente simples de ser implementado.

A fim de motivar esta família, considere novamente o problema de minimização proposto na regressão ridge (Seção 3.4):

$$\hat{\beta} = \arg \min_{\beta} \sum_{k=1}^n (y_k - \beta_0 - \beta_1 x_{k,1} - \dots - \beta_d x_{k,d})^2 + \lambda \sum_{i=1}^d \beta_i^2. \quad (4.4)$$

Denotando a i -ésima variável de \mathbf{x} por $\phi_i(\mathbf{x})$ (usando a convenção de que $\phi_0(x) \equiv x_0 \equiv 1$), vemos que tal problema pode ser reformulado como

$$\hat{r}(\mathbf{x}) = \arg \min_{g \in \mathcal{H}} \sum_{k=1}^n (y_k - g(\mathbf{x}_k))^2 + \lambda \|g\|_{\mathcal{H}}^2 \quad (4.5)$$

em que

$$\mathcal{H} := \left\{ g \in L^2(\mathcal{X}) : \text{existem } (c_i)_{i=0}^d \text{ com } \sum_{i=0}^d c_i^2 < \infty \text{ tais que } g(\mathbf{x}) = \sum_{i=0}^d c_i \phi_i(\mathbf{x}) \right\} \quad (4.6)$$

e

$$\|g\|_{\mathcal{H}}^2 := \sum_{i=0}^d c_i^2.$$

Enquanto que 4.4 é um problema de otimização sobre vetores de \mathbb{R}^{d+1} , 4.5 é um problema de otimização sobre *funções*. Note que enquanto o termo $\sum_{k=1}^n (y_k - g(\mathbf{x}_k))^2$ mede a bondade de ajuste de g , o termo $\|g\|_{\mathcal{H}}^2$ mede a *suavidade* de g : se $\|g\|_{\mathcal{H}}^2$ é baixo, temos g suave, caso contrário g oscila muito. Em outras palavras, $\|g\|_{\mathcal{H}}^2$ alto implica que uma mudança pequena em \mathbf{x} acarreta em uma mudança grande em $g(\mathbf{x})$.

A ideia por trás de métodos de regressão com base em penalização em *Reproducing Kernel Hilbert Spaces* (RKHS) é generalizar esta abordagem. Ao invés de usarmos apenas este espaço específico \mathcal{H} , consideraremos espaços mais gerais \mathcal{H} em que podemos resolver a Equação 4.5 facilmente. Tais espaços são justamente os *Reproducing Kernel Hilbert Spaces*, que descrevemos a seguir. Isto permite criar uma classe mais ampla de estimadores que tem possibilidade de ter boa performance em uma grande variedade de situações. Mais especificamente, buscaremos resolver o problema

$$\arg \min_{g \in \mathcal{H}} \sum_{k=1}^n L(g, (\mathbf{x}_k, y_k)) + \lambda \|g\|_{\mathcal{H}}^2,$$

em que \mathcal{H} é um RKHS e L é uma função adequada para o problema em questão. Veremos que a norma $\|g\|_{\mathcal{H}}$ reflete a suavidade das funções em \mathcal{H} , e que cada espaço contém uma noção de suavidade diferente. Assim, dado um novo problema, um usuário pode desenhar sua função de perda L e escolher um espaço que julgar adequado para criar sua função de predição g . Veremos que *smoothing splines*, *support vector regression* e *kernel ridge regression* são casos particulares desta abordagem, para escolhas particulares de L e \mathcal{H} .

4.6.1 Penalização em Reproducing Kernel Hilbert Spaces (RKHS)

Nesta seção veremos que um RKHS \mathcal{H} é essencialmente um subespaço de $L^2(\mathcal{X})$ que contém funções *suaves*. Em particular, a norma $\|g\|_{\mathcal{H}}^2$ é uma medida de suavidade da função g . Uma quantidade fundamental na definição de um RKHS é um *Kernel de Mercer*⁴, que definimos no que segue.

Definição 4.1 Seja $K(\mathbf{x}, \mathbf{y})$ uma função cujo domínio é $\mathcal{X} \times \mathcal{X}$ ⁵ que é:

- **Simétrica:** $K(\mathbf{x}, \mathbf{y}) = K(\mathbf{y}, \mathbf{x})$ para todo $\mathbf{x}, \mathbf{y} \in \mathcal{X}$
- **Positiva Semi-Definida:** a matriz $[K(\mathbf{x}_i, \mathbf{x}_j)]_{i,j=1}^n$ é positiva semi-definida para todo $n \in \mathbb{N}$ e para todo $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathcal{X}$.

Dizemos que K é um *Kernel de Mercer*.

Exemplo 4.1 Alguns kernels de Mercer comuns são:

- $K(\mathbf{x}_i, \mathbf{x}_l) = (1 + \langle \mathbf{x}_i, \mathbf{x}_l \rangle)^d$ (kernel polinomial)
- $K(\mathbf{x}_i, \mathbf{x}_l) = e^{-\frac{d^2(\mathbf{x}_i, \mathbf{x}_l)}{2h^2}}$ (kernel gaussiano)

Veja mais exemplos em [Scholkopf and Smola \(2002\)](#).

□

No contexto de regressão, $K(\mathbf{x}, \mathbf{y})$ representa uma maneira de se medir a similaridade entre dois vetores de covariáveis \mathbf{x} e \mathbf{y} .

A seguinte decomposição de um kernel é essencial para a construção de RKHSs:

⁴ Não confundir com um kernel suavizador como aquele usado no estimador de Nadaraya-Watson.

⁵ Para métodos de penalização em RKHS, o espaço amostral \mathcal{X} pode ser um espaço qualquer, i.e., não precisa ser necessariamente \mathbb{R}^d .

Teorema 4.1 [Teorema de Mercer] Todo Kernel de Mercer K pode ser decomposto como

$$K(\mathbf{x}, \mathbf{y}) = \sum_{i \geq 0} \gamma_i \phi_i(\mathbf{x}) \phi_i(\mathbf{y}),$$

em que $\sum_{i \geq 0} \gamma_i^2 < \infty$ e ϕ_0, ϕ_1, \dots é um conjunto de funções.

Um caso particular é o dado no início da Seção 4.6: o kernel implícito na regressão ridge: $K(\mathbf{x}, \mathbf{y}) = \langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=0}^d x_i y_i$, o produto interno das covariáveis.

Um RKHS é definido da seguinte maneira:

Definição 4.2 Seja K um kernel, e sejam ϕ_i e γ_i , $i \geq 0$, como no Teorema 4.1. Considere o espaço de funções

$$\mathcal{H}_K = \left\{ g \in L^2(\mathcal{X}) : \text{existem } (c_i)_{i \geq 0} \text{ com } \sum_{i \geq 1} \frac{c_i^2}{\gamma_i} < \infty \text{ tais que } g(\mathbf{x}) = \sum_{i \geq 1} c_i \phi_i(\mathbf{x}) \right\}$$

Dizemos que \mathcal{H}_K é o Reproducing Kernel Hilbert Space (RKHS) associado ao kernel K , em que a norma de uma função $g(\mathbf{x}) = \sum_{i \geq 0} c_i \phi_i(\mathbf{x})$ é definida por $\|g\|_{\mathcal{H}_K}^2 := \sum_{i \geq 0} c_i^2 / \gamma_i$.

A norma $\|g\|_{\mathcal{H}_K}$ captura a suavidade de uma função g . Isso ocorre pois (i) a condição $\sum_{i \geq 0} \gamma_i^2 < \infty$ do Teorema de Mercer implica que $\gamma_i \rightarrow 0$ e (ii) tipicamente as funções ϕ_i 's ficam menos suaves à medida que $i \rightarrow \infty$. Assim, $\|g\|_{\mathcal{H}_K}^2 := \sum_{i \geq 0} c_i^2 / \gamma_i$ é pequeno quando c_i é extremamente pequeno para i grande. De fato, para c_i^2 / γ_i ser baixo quando i é grande, c_i deve ser extremamente pequeno, pois $\gamma_i \approx 0$.

Embora a decomposição do Teorema 4.1 em geral não é única, cada kernel K define um único RKHS:

Teorema 4.2 A um kernel K corresponde um único RKHS \mathcal{H}_K .

4.6.2 Solução

O problema de estimação de uma função de regressão via RKHSs consiste em encontrar a solução para uma generalização da Equação 4.5. Mais especificamente, buscamos encontrar

$$\arg \min_{g \in \mathcal{H}_K} \sum_{k=1}^n L(g, (\mathbf{x}_k, y_k)) + \lambda \|g\|_{\mathcal{H}_K}^2, \quad (4.7)$$

em que \mathcal{H}_K é um RKHS arbitrário e $L(g, (\mathbf{x}, y))$ é uma função de perda arbitrária, ambos definidos pelo usuário do método. λ é um *tuning parameter* que determina o balanço entre viés e variância como no lasso. λ grande leva a funções mais suaves (pois $\|g\|_{\mathcal{H}_K}^2$ é menor), enquanto que λ pequeno leva a funções que se ajustam melhor ao conjunto de treinamento.

O seguinte teorema, frequentemente atribuído a [Kimeldorf and Wahba \(1971\)](#), é de fundamental importância para resolução do problema de minimização da Equação 4.7:

Teorema 4.3 [Teorema da Representação] *Seja K um kernel de Mercer correspondente ao RKHS \mathcal{H}_K . Considere um conjunto de treinamento $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$ e uma função de perda arbitrária L . Então a solução de*

$$\arg \min_{g \in \mathcal{H}_K} \sum_{k=1}^n L(g, (\mathbf{x}_k, y_k)) + \lambda \|g\|_{\mathcal{H}_K}^2 \quad (4.8)$$

existe, é única e tem a forma

$$g(\mathbf{x}) = \sum_{k=1}^n \alpha_k K(\mathbf{x}_k, \mathbf{x})$$

Este resultado converte o problema de busca pelo mínimo na classe \mathcal{H}_K em um problema de otimização de um número finito de coeficientes $\alpha_1, \dots, \alpha_n$. Esse novo problema de otimização pode ser resolvido usando técnicas usuais de cálculo ou cálculo numérico.

Notamos ainda que, usando o Teorema de Mercer (Teorema 4.1), a fórmula para a norma de g do Teorema 4.3 pode ser simplificada. Temos que

$$\sum_{k=1}^n \alpha_k K(\mathbf{x}_k, \mathbf{x}) = \sum_{k=1}^n \alpha_k \sum_{i \geq 0} \gamma_i \phi_i(\mathbf{x}_k) \phi_i(\mathbf{x}) = \sum_{i \geq 0} \gamma_i \left(\sum_{k=1}^n \alpha_k \phi_i(\mathbf{x}_k) \right) \phi_i(\mathbf{x})$$

Assim, para $g(\mathbf{x}) = \sum_{k=1}^n \alpha_k K(\mathbf{x}_k, \mathbf{x})$, temos

$$\begin{aligned} \|g\|_{\mathcal{H}_K}^2 &= \sum_{i \geq 0} \frac{(\gamma_i (\sum_{k=1}^n \alpha_k \phi_i(\mathbf{x}_k)))^2}{\gamma_i} = \sum_{i \geq 0} \gamma_i \left(\sum_{1 \leq j, k \leq n} \alpha_j \alpha_k \phi_i(\mathbf{x}_j) \phi_i(\mathbf{x}_k) \right) \\ &= \sum_{1 \leq j, k \leq n} \alpha_j \alpha_k \left(\sum_{i \geq 0} \gamma_i \phi_i(\mathbf{x}_j) \phi_i(\mathbf{x}_k) \right) = \sum_{1 \leq j, k \leq n} \alpha_j \alpha_k K(\mathbf{x}_j, \mathbf{x}_k) \end{aligned} \quad (4.9)$$

Tal decomposição facilita o cálculo da solução apresentada pelo Teorema 4.3: a solução para o problema de otimização sobre funções da Equação 4.8 é dada pela solução do seguinte problema de otimização sobre vetores:

$$\arg \min_{\alpha_1, \dots, \alpha_n} \sum_{k=1}^n L \left(\sum_{i=1}^n \alpha_i K(\mathbf{x}_i, \mathbf{x}), (\mathbf{x}_k, y_k) \right) + \lambda \sum_{1 \leq j, k \leq n} \alpha_j \alpha_k K(\mathbf{x}_j, \mathbf{x}_k).$$

Na sequência, apresentamos alguns exemplos do cálculo desta solução em problemas particulares.

4.6.3 Exemplo 1: Kernel Ridge Regression

Quando a função de perda da Equação 4.8 é a perda quadrática, $L(g, (\mathbf{x}_i, y_i)) = (y_i - g(\mathbf{x}_i))^2$, o estimador obtido minimizando-se a Equação 4.7 é chamado de *kernel ridge regression*. De fato, se \mathcal{H}_K é o espaço descrito na Equação 4.6 (i.e., $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$), o problema da Equação 4.7 equivale ao problema da regressão ridge. Neste caso, o Teorema da Representação mostra que a solução de

$$\arg \min_{g \in \mathcal{H}_K} \sum_{j=1}^n (y_j - g(\mathbf{x}_j))^2 + \lambda \|g\|_{\mathcal{H}_K}^2$$

é dada por $\hat{g}(\mathbf{x}) = \sum_{k=1}^n \hat{\alpha}_k K(\mathbf{x}_k, \mathbf{x})$, em que $\hat{\alpha}_k$'s são obtidos via

$$\arg \min_{\alpha_1, \dots, \alpha_n} \sum_{j=1}^n \left(y_j - \sum_{k=1}^n \alpha_k K(\mathbf{x}_k, \mathbf{x}_j) \right)^2 + \lambda \sum_{1 \leq j, k \leq n} \alpha_j \alpha_k K(\mathbf{x}_j, \mathbf{x}_k),$$

em que usamos a Equação 4.9 para reescrever a penalidade $\|g\|_{\mathcal{H}_K}^2$. Matricialmente, este problema pode ser reescrito como

$$\arg \min_{\boldsymbol{\alpha}} (\mathbf{y} - \mathbb{K}\boldsymbol{\alpha})^t (\mathbf{y} - \mathbb{K}\boldsymbol{\alpha}) + \lambda \boldsymbol{\alpha}^t \mathbb{K} \boldsymbol{\alpha}, \quad (4.10)$$

em que $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_n)^t$, $\mathbf{y} = (y_1, \dots, y_n)^t$ e

$$\mathbb{K} = \begin{bmatrix} \langle \mathbf{x}_1, \mathbf{x}_1 \rangle & \langle \mathbf{x}_1, \mathbf{x}_2 \rangle & \cdots & \langle \mathbf{x}_1, \mathbf{x}_n \rangle \\ \langle \mathbf{x}_2, \mathbf{x}_1 \rangle & \langle \mathbf{x}_2, \mathbf{x}_2 \rangle & \cdots & \langle \mathbf{x}_2, \mathbf{x}_n \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \mathbf{x}_n, \mathbf{x}_1 \rangle & \langle \mathbf{x}_n, \mathbf{x}_2 \rangle & \cdots & \langle \mathbf{x}_n, \mathbf{x}_n \rangle \end{bmatrix}$$

A solução para a Equação 4.10 é dada por

$$\hat{\boldsymbol{\alpha}} = (\mathbb{K} + \lambda \mathbb{I})^{-1} \mathbf{y}, \quad (4.11)$$

de modo que o estimador dado pela *kernel ridge regression* é

$$\hat{g}(\mathbf{x}) = \hat{\boldsymbol{\alpha}}^t \mathbf{k} = \mathbf{y}^t (\mathbb{K} + \lambda \mathbb{I})^{-1} \mathbf{k}, \quad (4.12)$$

com $\mathbf{k} = (K(\mathbf{x}_1, \mathbf{x}), \dots, K(\mathbf{x}_n, \mathbf{x}))$.

4.6.3.1 Smoothing Splines

Quando $\mathbf{x} \in [0, 1]$, um caso particular da *kernel ridge regression* é o de *smoothing splines*, em que se busca pela função g que minimiza

$$\sum_{k=1}^n (y_k - g(\mathbf{x}_k))^2 + \lambda \int_0^1 \|g''(\mathbf{x})\|^2 d\mathbf{x}. \quad (4.13)$$

Note que $\int_0^1 \|g''(\mathbf{x})\|^2 d\mathbf{x}$ também é uma forma de se medir o quanto suave g é (veja mais sobre isso na Seção 4.12); na realidade essa quantidade corresponde à normal em um RKHS específico, veja Wahba (1990), Nosedal-Sánchez et al. (2012), Pearce and Wand (2006)⁶.

Pode-se mostrar que a solução para esse problema corresponde a uma expansão de $g(\mathbf{x})$ em uma base formada por splines naturais cúbicos com nós no conjunto de treinamento (Wahba, 1990). Contudo, ao contrário da metodologia apresentada na Seção 4.2, aqui os coeficientes da expansão são penalizados de modo a se obter soluções mais suaves. É justamente por isso que esse método é chamado de *smoothing splines*.

4.6.3.2 O Truque do Kernel*

O estimador do *kernel ridge regression* também pode ser motivado sem a evocação de *Reproducing Kernel Hilbert Spaces*. Para tanto, começamos reescrevendo o estimador dado pela *regressão ridge*. Lembre-se que, como visto na Seção 3.4, a regressão ridge busca por

$$\hat{\boldsymbol{\beta}} = \arg \min_{\boldsymbol{\beta}} \sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^d \beta_j x_{i,j} \right)^2 + \lambda \sum_{j=1}^d \beta_j^2,$$

e tem como solução

$$\hat{\boldsymbol{\beta}} = (\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_d) = (\mathbb{X}^t \mathbb{X} + \lambda \mathbb{I})^{-1} \mathbb{X}^t Y.$$

⁶ Tecnicamente, neste contexto é necessário uma versão um pouco mais geral do Teorema da Representação, veja por exemplo Wahba (1990), Nosedal-Sánchez et al. (2012).

Com um pouco de álgebra, pode-se mostrar que $\hat{\beta}$ pode ser escrito como $\mathbb{X}^t(\mathbb{X}\mathbb{X}^t + \lambda\mathbb{I})^{-1}Y$, de modo que o estimador da regressão é dado por

$$\hat{g}(\mathbf{x}) = Y^t(\mathbb{X}\mathbb{X}^t + \lambda\mathbb{I})^{-1}\mathbb{X}\mathbf{x} = Y^t(\mathbb{K} + \lambda\mathbb{I})^{-1}\mathbf{k}$$

em que

$$\mathbf{k} = (\langle \mathbf{x}_1, \mathbf{x} \rangle, \dots, \langle \mathbf{x}_n, \mathbf{x} \rangle)^t$$

e

$$\mathbb{K} = \begin{bmatrix} \langle \mathbf{x}_1, \mathbf{x}_1 \rangle & \langle \mathbf{x}_1, \mathbf{x}_2 \rangle & \cdots & \langle \mathbf{x}_1, \mathbf{x}_n \rangle \\ \langle \mathbf{x}_2, \mathbf{x}_1 \rangle & \langle \mathbf{x}_2, \mathbf{x}_2 \rangle & \cdots & \langle \mathbf{x}_2, \mathbf{x}_n \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \mathbf{x}_n, \mathbf{x}_1 \rangle & \langle \mathbf{x}_n, \mathbf{x}_2 \rangle & \cdots & \langle \mathbf{x}_n, \mathbf{x}_n \rangle \end{bmatrix}$$

Assim, para calcular $\hat{g}(\mathbf{x})$, é suficiente sabermos os produtos internos entre todos os pares de observação, i.e., $\langle \mathbf{x}_i, \mathbf{x}_l \rangle = \sum_{k=1}^d \mathbf{x}_{i,k} \mathbf{x}_{l,k}$. Este fato é fundamental para o que será descrito a seguir.

Suponha, agora, que estejamos interessados em uma *transformação das covariáveis originais*, da mesma maneira que discutido na Seção 2.1. Por exemplo, suponhamos que temos duas covariáveis, (x_1, x_2) , e que queremos estimar uma regressão que não é linear no espaço original, mas sim que tem a forma

$$r(\mathbf{x}) = \beta_1 x_1 + \beta_2 x_1^2 + \beta_3 x_2 + \beta_4 x_2^2 + \beta_5 x_1 x_2. \quad (4.14)$$

Para isto, podemos usar o estimador da regressão ridge com novas covariáveis $\mathbf{w} = (1, \sqrt{2}x_1, x_1^2, \sqrt{2}x_2, x_2^2, \sqrt{2}x_1 x_2)$. Como mostrado anteriormente, para calcular este estimador, somente é necessário que saibamos os produtos internos das observações com relação às *novas covariáveis*, i.e.,

$$\langle \mathbf{w}_i, \mathbf{w}_l \rangle = 1 + 2x_{i,1}*x_{l,1} + x_{i,1}^2*x_{l,1}^2 + 2x_{i,2}*x_{l,2} + x_{i,2}^2*x_{l,2}^2 + 2x_{i,1}x_{i,2}*x_{l,1}x_{l,2}. \quad (4.15)$$

Mais precisamente, o estimador é dado por

$$\hat{r}(\mathbf{x}) = Y^t(\mathbb{K} + \lambda\mathbb{I})^{-1}\mathbf{k} \quad (4.16)$$

em que

$$\mathbf{k} = (K(\mathbf{x}_1, \mathbf{x}), \dots, K(\mathbf{x}_n, \mathbf{x}))$$

$$\mathbb{K} = \begin{bmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & K(\mathbf{x}_1, \mathbf{x}_2) & \cdots & K(\mathbf{x}_1, \mathbf{x}_n) \\ K(\mathbf{x}_2, \mathbf{x}_1) & K(\mathbf{x}_2, \mathbf{x}_2) & \cdots & K(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ K(\mathbf{x}_n, \mathbf{x}_1) & K(\mathbf{x}_n, \mathbf{x}_2) & \cdots & K(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix}$$

e $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{w}_i, \mathbf{w}_j \rangle$. Note que com \mathbf{x} podemos calcular \mathbf{w} , por isso usamos a notação $K(\mathbf{x}_i, \mathbf{x}_j)$ para representar o produto interno no espaço transformado. O motivo de usarmos a letra K ficará claro na sequência.

O truque, chamado de *truque do kernel*, consiste em notar que, para algumas transformações $\mathbf{w}(\mathbf{x})$, é possível calcular o produto interno $\langle \mathbf{w}_i, \mathbf{w}_l \rangle$ facilmente, em particular não é necessário calcular as novas covariáveis em cada amostra. Isto ocorre na transformação da ilustração: note que o produto interno da Equação 4.15 pode ser desenvolvido como

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{w}_i, \mathbf{w}_j \rangle = (1 + x_{i,1} * x_{l,1} + x_{i,2} * x_{l,2})^2 = (1 + \langle \mathbf{x}_i, \mathbf{x}_j \rangle)^2.$$

Assim, para calcular $\langle \mathbf{w}_i, \mathbf{w}_j \rangle$, precisamos apenas saber calcular $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$, i.e., não é necessário que se calcule em nenhum momento \mathbf{w}_i . Ou seja, é possível calcular um estimador do tipo da Equação 4.14 sem nunca calcular x_1^2, x_1x_2 etc! Evidentemente, neste exemplo isto pode não ser muita vantagem, mas imagine se temos $d = 10^6$ covariáveis e estamos interessados em todos os produtos dois a dois. Calcular a transformação neste caso não só seria lento, mas também poderia levar a uma matriz de observações que dificilmente conseguiria ser armazenada na memória de um computador (seria uma matriz com $(10^{12})/2$ colunas).

Na prática, para se usar o truque do kernel, começamos especificando diretamente um kernel $K(\mathbf{x}_i, \mathbf{x}_j)$ que corresponde a um produto interno em um espaço transformado. Qualquer kernel de Mercer pode ser utilizado, pois o Teorema de Mercer (Teorema 4.1) garante que tais kernels correspondem a um produto interno no espaço (potencialmente infinito) dado por $(\sqrt{\gamma}_1 \phi_1(\mathbf{x}), \sqrt{\gamma}_2 \phi_2(\mathbf{x}), \dots)$. Uma vez que o kernel está escolhido, podemos calcular o estimador da regressão ridge para este kernel.

Note que este truque permite que trabalhemos com a regressão ridge em espaços com infinitas covariáveis, o que não seria possível com a formulação original da regressão ridge. Isto ocorre justamente porque nunca é necessário calcular essas novas covariáveis, mas apenas produtos internos neste espaço. Este é o caso, por exemplo, do kernel gaussiano: a⁷ transformação correspondente a ele é de fato infinita.

Para terminar esta seção, veja que o estimador da Equação 4.16 – ainda que seja motivado por considerações bastante diferentes – é o mesmo que o estimador discutido anteriormente (Equação 4.12).

⁷ Na realidade existem infinitas transformações que levam ao mesmo produto interno.

Observação 4.2 Note que o kernel linear $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$ corresponde à regressão ridge nas covariáveis originais. Neste caso, usar a Equação 4.16 ao invés de 3.4 para resolver o problema não é uma vantagem: se $n > d$, ela leva a um tempo computacional maior para implementação, uma vez que é necessário inverter uma matriz $n \times n$ ao invés de uma matriz $d \times d$ como na formulação original. A grande vantagem desta abordagem é permitir transformações não lineares possivelmente infinitas.

□

4.6.3.3 Implementação eficiente da Kernel Ridge Regression

Note que, para implementar a *Kernel Ridge Regression*, é necessário calcular

$$(\mathbb{K} + \lambda \mathbb{I})^{-1}$$

na Equação 4.11. Uma implementação ingênuo do *kernel ridge regression* é computacionalmente lenta, pois é necessário calcular a inversa de $\mathbb{K} + \lambda \mathbb{I}$ para cada λ de interesse. Uma maneira de torná-la mais rápida é usar a seguinte ideia. Usando a decomposição SVD, podemos escrever

$$\mathbb{K} = UDU^t,$$

em que D é diagonal. Ademais, temos que $UU^t = \mathbb{I}$. Assim,

$$(\mathbb{K} + \lambda \mathbb{I})^{-1} = (UDU^t + \lambda \mathbb{I})^{-1} = (U(D + \lambda \mathbb{I})U^t)^{-1} = (U^{-1})^t(D + \lambda \mathbb{I})^{-1}U^{-1}.$$

Logo, para cada λ , precisamos apenas calcular a inversa da matriz diagonal, o que é extremamente rápido e fácil de se fazer.

4.6.4 Exemplo 2: Support Vector Regression Machines

Para criar estimadores da função de regressão, Support Vector Regression Machines ([Smola and Vapnik, 1997](#)) utilizam uma função de perda diferente da quadrática. Mais especificamente, seja $\epsilon > 0$ um número fixo e K um kernel de Mercer. A função de predição g dada pela Support Vector Regression (SVR) é aquela função que minimiza

$$\arg \min_{g \in \mathcal{H}_K} \sum_{k=1}^n L(g, (\mathbf{x}_k, y_k)) + \lambda \|g\|_{\mathcal{H}_K}^2, \quad (4.17)$$

em que $L(g, (\mathbf{x}_k, y_k)) = (|y_k - g(\mathbf{x}_k)| - \epsilon)_+$ (Pontil, 2003). Segundo esta função de perda, a distância entre a predição $g(\mathbf{x}_k)$ e a observação y_k ser menor que ϵ é suficiente para que não haja nenhum custo no erro. Por outro lado, se a distância é maior que ϵ , a perda é dada por $|y_k - g(\mathbf{x}_k)| - \epsilon$. Por este motivo, esta perda é chamada de ϵ -insensível.

Pode-se utilizar o Teorema da Representação (Teorema 4.3) para simplificar o problema dado pela Equação 4.17 de forma a torná-lo um problema finito-dimensional. Contudo, ao contrário do que ocorre com a Kernel Ridge Regression (Seção 4.6.3), a solução para tal problema não é analítica, e portanto requer o uso de métodos numéricos de otimização.

Para ajustar a support vector regression machines no R, pode-se utilizar o pacote e1071.

```
ajuste = svm(x=covariaveisTreinamento, y=respostaTreinamento,
             type="eps-regression",
             kernel="radial",
             gamma = gamma)
predVal=predict(ajuste, newdata=novosDados)
```

O valor de ϵ pode ser definido pelo usuário do método e é em geral escolhido por validação cruzada.

4.7 Modelos Aditivos

Um problema da maioria dos métodos não paramétricos é sua dificuldade de interpretabilidade quando comparado com métodos paramétricos. Modelos aditivos (*additive models*, Hastie and Tibshirani 1986) são uma forma de encontrar um balanço entre interpretabilidade e flexibilidade. Eles consistem em uma generalização de uma regressão linear de modo a permitir uma relação não linear entre cada covariável x_j e a resposta y . A ideia básica é trocar o termo $\beta_j x_j$ na equação da regressão linear (Eq. 2.1) por $r_j(x_j)$, em que $r_j(x_j)$ são funções *univariadas* suaves. Assim, um modelo aditivo pode ser escrito como

$$r(\mathbf{x}) = \alpha + \sum_{j=1}^d r_j(x_j).$$

Note que este modelo não é identificável, já que podemos adicionar qualquer constante α e retirar a mesma constante de qualquer r_j , obtendo assim a mesma solução. Uma forma de evitar isso é restringir que nossos estimadores sejam tais que $\hat{\alpha} = \bar{Y}$, e forçar $\sum_{i=1}^n \hat{r}_j(X_{i,j}) = 0$. Este modelo é chamado de aditivo pois supõe-se que a função de regressão pode ser decomposta em termos de uma soma de funções suaves. Assim, este

método faz mais suposições que, por exemplo, o estimador de Nadaraya-Watson. Sua grande vantagem é que ele possui interpretação mais simples que estimadores mais gerais, como mostramos nos exemplos.

Modelos aditivos em geral são ajustados utilizando-se o algoritmo backfitting:

1. Inicialize $\hat{\alpha} = \bar{Y}$, e crie estimativas iniciais $\hat{r}_1, \dots, \hat{r}_d$.

2. Faça, até obter convergência:

Para $j = 1, \dots, d$

a. Calcule $\tilde{Y}_i = Y_i - \hat{\alpha} - \sum_{k \neq j} \hat{r}_k(x_{i,k})$

b. Estimar a regressão de $(\tilde{Y}_i)_i$ em $(x_{j,i})_i$, \hat{r}_j . Aqui podemos usar qualquer método de regressão univariada, paramétrico ou não paramétrico.

c. Definir $\hat{r}_j(\mathbf{x}) = \hat{r}_j(\mathbf{x}) - \frac{1}{n} \sum_{i=1}^n \hat{r}_j(x_{i,j})$

A seguir mostramos um exemplo das vantagens deste método.

Exemplo 4.2 Neste exemplo ajustamos um modelo aditivo para dados do Programa das Nações Unidas para o Desenvolvimento sobre o IDH de municípios do Brasil em 1991, 2000 e 2010⁸. Cada amostra aqui representa dados sobre um município em um dado ano. A variável resposta utilizada é o IDH de cada município, enquanto que as variáveis explicativas são:

- O percentual da população de 11 a 13 anos de idade frequentando os anos finais do fundamental ou que já concluiu o fundamental no município,
- A renda per capita média do município,
- O ano em que os dados foram coletados.

Note que a última variável mencionada é categórica, mas mesmo assim é possível utilizar método aditivos para estes casos. Para as variáveis contínuas, optou-se por utilizar smoothing splines com 4 graus de liberdade.

⁸ <http://www.atlasbrasil.org.br/>.

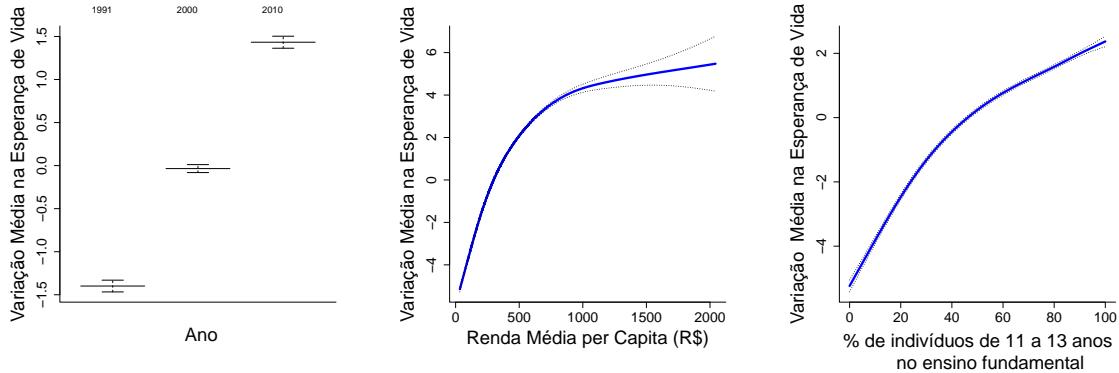


Figura 4.5: Ajuste dado pelo modelo aditivo para os dados de IDH municipais no Brasil (Exemplo 4.2).

A Figura 4.5 ilustra as três funções estimadas $r_j(x_j)$, juntamente com erros padrão sobre elas. Note que este modelo é muito mais interpretativo que os estudados anteriormente; em particular com esse resultado é possível identificar não linearidades nas relações entre as variáveis explicativas e a variável resposta. Por outro lado, modelos aditivos são mais restritivos que os outros métodos estudados neste capítulo, e portanto podem possuir poder preditivo mais baixo dependendo da natureza de $r(\mathbf{x})$ e do tamanho amostral.

□

Para implementar modelos aditivos, pode-se usar o pacote `gam`.

```
library(gam)
ajuste = gam(nomeResposta ~ s(nomeExplicativa1, 4) +
              s(nomeExplicativa2, 4), data=dados)
```

A função `s()` é parte do pacote `gam` e é usada para indicar que estamos usando smoothing splines com 4 graus de liberdade. Predições podem ser feitas com a função `predict`.

4.8 Árvores de Regressão

Árvores de regressão consistem em uma metodologia não paramétrica que leva a resultados extremamente interpretáveis. Uma árvore é construída por particionamentos

recursivos no espaço das covariáveis. Cada particionamento recebe o nome de nó e cada resultado final recebe o nome de folha, veja a Figura 4.6.

A utilização da árvore para prever uma nova observação é feita do seguinte modo: começando pelo topo, verificamos se a condição descrita no topo (primeiro nó) é satisfeita. Caso seja, seguimos a esquerda. Caso contrário, seguimos a direita. Assim prosseguimos até atingir uma folha. No caso ilustrativo da figura 4.6, se a condição 1 for satisfeita, a predição é dada por F1. Caso não seja satisfeita, seguimos a direita e assim, encontramos outra condição. Caso a mesma seja satisfeita, a observação é prevista como F2 e, caso contrário, é prevista como F3.

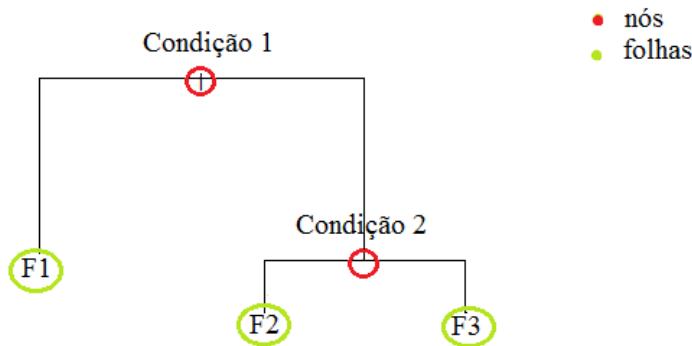


Figura 4.6: Exemplo de estrutura de uma árvore.

A Figura 4.7 ilustra um exemplo concreto de uma árvore de regressão, construída com o objetivo de prever o salário de um indivíduo dadas covariáveis como idade e anos de estudo. Note como é fácil usar tal objeto para entender a relação entre as variáveis explicativas e a variável resposta, ao contrário do que ocorre com a maior parte dos métodos não paramétricos.

Formalmente, uma árvore cria uma partição do espaço das covariáveis em regiões distintas e disjuntas: R_1, \dots, R_j . A predição para a resposta Y de uma observação com covariáveis \mathbf{x} que estão em R_k é então dada por:

$$g(\mathbf{x}) = \frac{1}{|\{i : \mathbf{x}_i \in R_k\}|} \sum_{i:\mathbf{x}_i \in R_k} y_i, \quad (4.18)$$

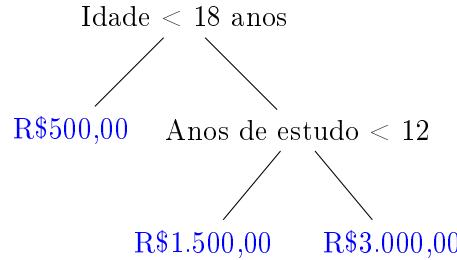


Figura 4.7: Exemplo fictício de árvore da regressão do salário de um indivíduo dadas as covariáveis idade e anos de estudo.

isto é, para prever o valor da resposta de \mathbf{x} , observamos a região a qual a observação \mathbf{x} pertence e, então, calculamos a média dos valores da variável resposta das amostras do conjunto treinamento pertencentes àquela mesma região.

A criação da estrutura de uma árvore de regressão é feita através de duas grandes etapas: (I) a criação de uma árvore completa e complexa e (II) a poda de tal árvore, com a finalidade de evitar o super ajuste.

No passo I, busca-se criar uma árvore que leve a partições “puras”, i.e., partições nas quais os valores de Y nas observações do conjunto de treinamento em cada uma das folhas sejam homogêneos. Para tanto, avalia-se o quanto razoável uma dada árvore T é através de seu erro quadrático médio,

$$\mathcal{P}(T) = \sum_R \sum_{\mathbf{x}_k \in R} \frac{(y_k - \hat{y}_R)^2}{n},$$

em que \hat{y}_R é o valor predito para a resposta de uma observação pertencente à região R . Infelizmente, encontrar T que minimize $\mathcal{P}(T)$ é computacionalmente inviável. Assim, utiliza-se uma heurística para encontrar uma árvore com erro quadrático médio baixo. Esta consiste na criação de divisões binárias recursivas, como mostrado na Figura 4.8. Inicialmente, o algoritmo particiona o espaço de covariáveis em duas regiões distintas (Figura 4.8 (a)). Para escolher tal partição, busca-se, dentre todas as covariáveis x_i e cortes t_1 , aquela combinação que leva a uma partição (R_1, R_2) com previsões de menor erro quadrático:

$$\sum_{i:\mathbf{x}_i \in R_1}^n (y_i - \hat{y}_{R_1})^2 + \sum_{i:\mathbf{x}_i \in R_2}^n (y_i - \hat{y}_{R_2})^2, \quad (4.19)$$

em que \hat{y}_{R_k} é a previsão fornecida para a região R_k (veja Eq. 4.18). Assim, define-se

$$R_1 = \{\mathbf{x} : x_{i^*} < t_1^*\} \text{ e } R_2 = \{\mathbf{x} : x_{i^*} \geq t_1^*\},$$

em que x_{i^*} é a variável e o corte escolhido, respectivamente.

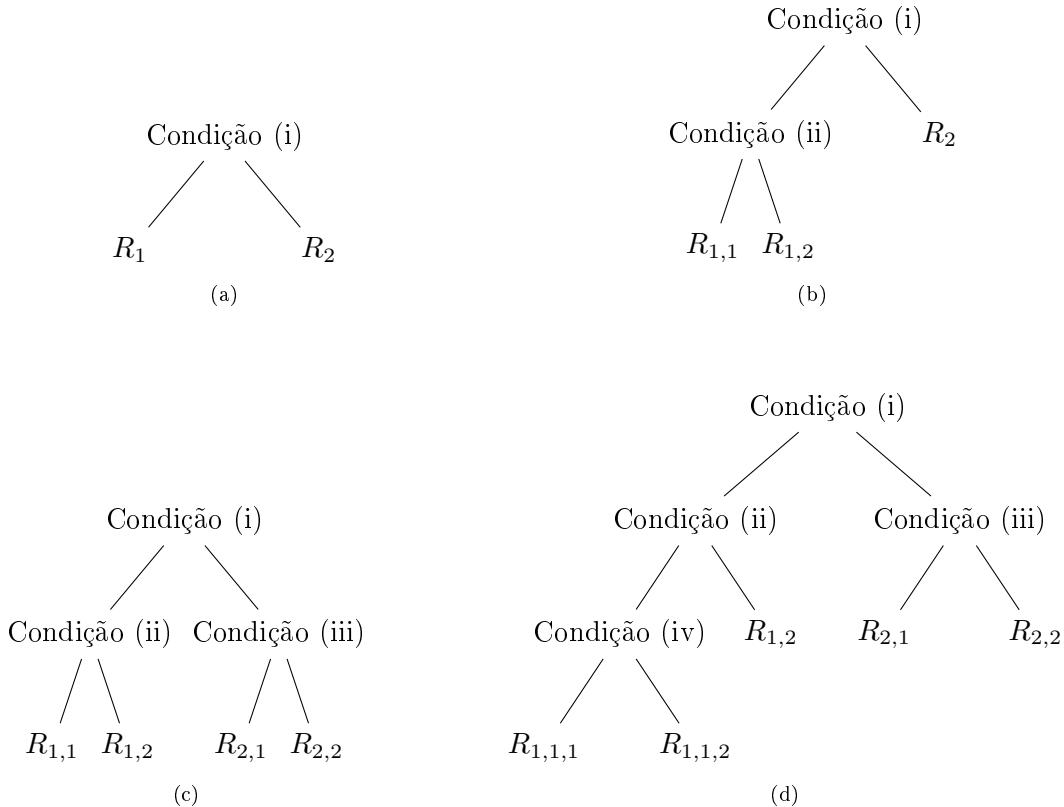


Figura 4.8: Processo de crescimento de uma árvore de regressão.

Uma vez estabelecidas tais regiões, o nó inicial da árvore é então fixo, e busca-se partitionar R_1 ou R_2 em regiões menores (Figura 4.8 (b)). Para escolher a nova divisão, a mesma estratégia é utilizada: busca-se, dentre todas as covariáveis x_i e cortes t_2 , aquela combinação que leva a uma partição com menor erro quadrático. Note que agora também é necessário escolher qual região deve ser partitionada: R_1 ou R_2 . Assuma que R_1 foi a região escolhida, juntamente com a covariável x_{j^*} e o corte t_2^* . Chamemos a partição de R_1 de $\{R_{1,1}, R_{1,2}\}$, como mostra a Figura 4.8 (b). Assim,

$$R_{1,1} = \{\mathbf{x} : x_{i^*} < t_1^*, x_{j^*} < t_2^*\}, \quad R_{1,2} = \{\mathbf{x} : x_{i^*} < t_1^*, x_{j^*} \geq t_2^*\} \text{ e } R_2 = \{\mathbf{x} : x_{i^*} \geq t_1^*\}.$$

O procedimento continua recursivamente (veja Figuras 4.8 (c) e (d)), até que cheguemos a uma árvore com poucas observações em cada uma das folhas (por exemplo, o processo para quando todas as folhas têm menos de 5 observações).

A árvore criada utilizando-se este processo produz ótimos resultados para o conjunto de treinamento, mas é muito provável que ocorra o super-ajuste. Isso gera uma performance preditiva ruim em novas observações. Assim, prossegue-se para o passo (II), que é chamado de poda. Tal etapa visa tornar a árvore de regressão menor e menos complexa, de modo a diminuir a variância deste estimador. Nessa etapa do processo, cada nó é retirado, um por vez, e observa-se como o erro de predição varia no conjunto de validação. Com base nisso, decide-se quais nós permanecerão na árvore.

Para ajustar uma árvore no R, pode-se utilizar o pacote `rpart`.

```
library(rpart)
library(rpart.plot)
data("mtcars")

# Ajustar a árvore:
fit <- rpart(mpg ~ ., method="anova", data=mtcars)
# poda:
melhorCp=fit$cptable[which.min(fit$cptable[, "xerror"]),"CP"]
  # cp é uma medida de complexidade da árvore, essencialmente
  # proporcional ao número de folhas presentes. Este código
  # escolhe o melhor cp via validação cruzada.
pfit<- prune(fit,
             cp=melhorCp)
# plotar árvore podada
rpart.plot(pfit, type = 4, extra = 1)
```

A Figura 4.9 ilustra a árvore gerada pelo código acima no R. O tamanho de cada ramo na árvore gerada é proporcional à diminuição do erro quadrático médio que ocorreu quando a respectiva partição foi criada. Assim, ramos grandes indicam uma importância maior da covariável na predição da variável resposta.

Notamos também que árvores podem facilmente lidar com covariáveis discretas. Por exemplo, se X_1 representa a cor de uma flor, pode-se utilizar critérios de divisão como $X_1 \in \{\text{verde, vermelho}\}$ nos nós da árvore gerada.

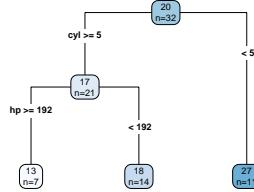


Figura 4.9: Árvore de regressão já podada.

4.9 Bagging e Florestas Aleatórias

Infelizmente, apesar de serem extremamente interpretáveis, árvores em geral costumam ter um poder preditivo muito baixo quando comparados aos demais estimadores. Uma forma de contornar isso é através da combinação da predição fornecida por diversas árvores para se fazer predições.

Para motivar esta abordagem, imagine que, em um contexto de regressão, temos duas funções de predição Y , $g_1(\mathbf{x})$ e $g_2(\mathbf{x})$. Os riscos destas (condicionais em \mathbf{x}) são dados, respectivamente, por

$$\mathbb{E} [(Y - g_1(\mathbf{x}))^2 | \mathbf{x}] \quad \text{e} \quad \mathbb{E} [(Y - g_2(\mathbf{x}))^2 | \mathbf{x}] .$$

Considere agora o estimador combinado $g(\mathbf{x}) = (g_1(\mathbf{x}) + g_2(\mathbf{x}))/2$. Pelo resultado da Seção 1.4.2, temos que

$$\begin{aligned} \mathbb{E} [(Y - g(\mathbf{x}))^2 | \mathbf{x}] &= \\ &= \mathbb{V}[Y|\mathbf{x}] + \frac{1}{4} (\mathbb{V}[g_1(\mathbf{x}) + g_2(\mathbf{x})|\mathbf{x}]) + \left(\mathbb{E}[Y|\mathbf{x}] - \frac{\mathbb{E}[g_1(\mathbf{x})|\mathbf{x}] + \mathbb{E}[g_2(\mathbf{x})|\mathbf{x}]}{2} \right)^2 \end{aligned}$$

Assim, se $g_1(\mathbf{x})$ e $g_2(\mathbf{x})$ são não correlacionados ($\text{Cor}(g_1(\mathbf{x}), g_2(\mathbf{x})|\mathbf{x}) = 0$), não viesados ($\mathbb{E}[g_1(\mathbf{x})|\mathbf{x}] = \mathbb{E}[g_2(\mathbf{x})|\mathbf{x}] = r(\mathbf{x})$) e têm mesma variância ($\mathbb{V}[g_1(\mathbf{x})|\mathbf{x}] = \mathbb{V}[g_2(\mathbf{x})|\mathbf{x}]$),

$$\mathbb{E} [(Y - g(\mathbf{x}))^2 | \mathbf{x}] = \mathbb{V}[Y|\mathbf{x}] + \frac{1}{2} \mathbb{V}[g_i(\mathbf{x})|\mathbf{x}] \leq \mathbb{E} [(Y - g_i(\mathbf{x}))^2 | \mathbf{x}] , \quad (4.20)$$

$i = 1, 2$. Assim, é melhor se utilizar o estimador combinado $g(\mathbf{x})$ do que usar $g_1(\mathbf{x})$ ou $g_2(\mathbf{x})$ separadamente. Embora neste exemplo utilizamos apenas dois estimadores da função de regressão, ele continua valendo quando se combina um número B qualquer de estimadores.

Tanto o método de *bagging* quanto florestas aleatórias utilizam essa ideia para melhorar previsões dadas por árvores. Ambas as abordagens consistem em criar B árvores distintas e combinar seus resultados para melhorar o poder preditivo de cada árvore individual. Para criar as B árvores distintas, o *bagging* utiliza B amostras bootstrap da amostra original⁹. Para cada uma destas amostras, cria-se então uma árvore utilizando as técnicas descritas na Seção 4.8. Note, contudo, que assumimos na derivação da Equação 4.20 que os estimadores são não viesados. Assim, para criar árvores próximas de não-viesadas, não podemos usar as árvores criadas, i.e., utilizamos apenas o passo (I) descrito naquela seção.

Seja $g_b(\mathbf{x})$ a função de previsão obtida segundo a b -ésima árvore. A função de previsão dada pelo *bagging* é dada por

$$g(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^B g_b(\mathbf{x}).$$

Apesar do método *bagging* retornar preditores que não são tão fáceis de se interpretar quanto árvores, ele permite a criação de uma medida de importância para cada covariável: a média de quanto ela foi importante em cada árvore. Por exemplo, pode-se utilizar como medida de importância o quanto, em média, cada covariável contribuiu para a diminuição do erro quadrático médio. A Figura 4.10 ilustra a importância de cada covariável segundo dois critérios para o conjunto CO2 do R. Tal estimador foi ajustado utilizando-se o pacote `randomForest` do R.

```
library(randomForest)
data(CO2)
ajuste = randomForest(x=CO2[, ! colnames(CO2) %in% c("uptake")],
                      y=CO2[, "uptake"],
                      mtry=ncol(CO2) - 1,
                      importance = TRUE)
varImpPlot(ajuste)
```

⁹ Uma amostra bootstrap é uma amostra com reposição de mesmo tamanho da amostra original.

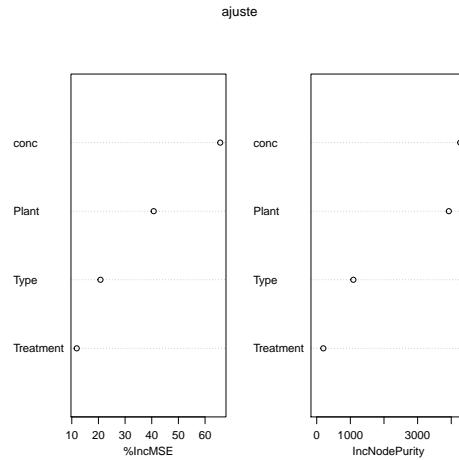


Figura 4.10: Importância da cada covariável do conjunto de dados CO2 na predição da variável uptake segundo o bagging.

4.9.1 Florestas Aleatórias

Além da suposição de que os estimadores são não viésados, na Equação 4.20 também assumimos que eles são não correlacionados. Em geral, mesmo utilizando amostras bootstrap para construir cada uma das árvores, as árvores construídas tendem a dar previsões muito parecidas. Assim, as funções $g_b(\mathbf{x})$ resultantes em geral tem correlação muito alta. Florestas tentam diminuir esta correlação. A ideia central é modificar o método de criação das árvores para que estas se tornem diferentes umas das outras. Mais especificamente, ao invés de escolher qual das d covariáveis será utilizada em cada um dos nós da árvore, em cada passo só é permitido que seja escolhida uma dentre as $m < d$ covariáveis. Estas m covariáveis são escolhidas aleatoriamente dentre as covariáveis originais e, a cada nó criado, um novo subconjunto de covariáveis é sorteado.

O valor de m pode ser escolhido por validação cruzada, mas em geral $m \approx \sqrt{d}$ leva a uma boa performance. Note que, quando $m = d$, o método de florestas aleatórias se reduz ao *bagging*.

Florestas aleatórias podem ser ajustadas utilizando-se o pacote `randomForest`:

```
library(randomForest)
data(CO2)
m=sqrt(ncol(CO2))
ajuste = randomForest(x=CO2[, ! colnames(CO2) %in% c("uptake")],
```

```

y=CO2[, "uptake"],
mtry=m,
importance = TRUE)
varImpPlot(ajuste)

```

4.10 Boosting

Assim como florestas aleatórias e o *bagging*, o *boosting* também consiste na agregação de diferentes estimadores da função de regressão. Contudo, tal combinação é feita de forma bastante diferente.

No *boosting*, o estimador $g(\mathbf{x})$ é construído incrementalmente. Inicialmente, atribuímos o valor de $g(\mathbf{x}) \equiv 0$. Este estimador possui alto viés, mas baixa variância (zero). A cada passo, atualizaremos o valor de g de modo a diminuir o viés e aumentar a variância da nova função. Isto é feito adicionando-se a g uma função que prevê os resíduos $Y_i - g(\mathbf{x}_i)$. Por exemplo, pode-se utilizar uma árvore de regressão para isso. É importante que esta tenha profundidade pequena de modo a evitar o super-ajuste. Além disso, ao invés de simplesmente adicionar essa função por completo, adicionamos ela multiplicada por λ , um fator entre 0 e 1 que tem por finalidade evitar o super-ajuste. Formalmente, o *boosting* consiste no seguinte algoritmo:

1. Definimos $g(\mathbf{x}) \equiv 0$ e $r_i = y_i \forall i$.
2. Para $b = 1, \dots, B$:
 - a. Ajustamos uma árvore com d folhas para $(\mathbf{x}_1, r_1), \dots, (\mathbf{x}_n, r_n)$. Seja $g^b(\mathbf{x})$ sua respectiva função de predição.
 - b. Atualizamos g e os resíduos: $g(\mathbf{x}) \leftarrow g(\mathbf{x}) + \lambda g^b(\mathbf{x})$ e $r_i \leftarrow Y_i - g(\mathbf{x})$.
3. Retornamos o modelo final $g(\mathbf{x})$

Note que os *tuning parameters* do *boosting* são B , d e λ . Tipicamente λ é pequeno (e.g., 0.001), $B \approx 1000$ e d é da ordem de 2 ou 4.

O *boosting* não necessariamente é feito com árvores, mas em geral são utilizados estimadores “fracos” (e.g., uma regressão com poucos parâmetros).

No R, o *boosting* pode ser implementando com o código a seguir. A Figura 4.11 ilustra como o risco estimado varia conforme o valor de B (número de iterações) utilizado.

```
library(bst)
ajuste = cv.bst(x=CO2[, ! colnames(CO2) %in% c("uptake")],
                 y=CO2[, "uptake"],
                 learner="tree",
                 control.tree = list(maxdepth = 3),
                 ctrl = bst_control(mstop = 10000, nu = 0.02),
                 K=2, se=FALSE)
```

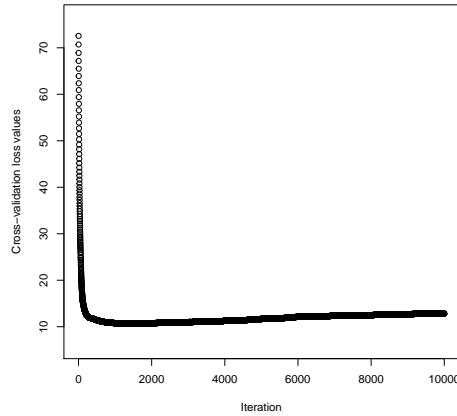


Figura 4.11: Diminuição do risco estimado por validação cruzada para a função de regressão segundo o número de iterações do algoritmo boosting.

```
#predito=predict(ajuste,xTest)
```

4.11 Redes Neurais Artificiais

Redes neurais artificiais são um conceito bastante antigo em inteligência artificial ([McCulloch and Pitts, 1943](#), [Rosenblatt, 1958](#)). Matematicamente, no contexto de regressão, trata-se apenas de um estimador não linear de $r(\mathbf{x})$, que pode ser representado por uma estrutura como a da Figura 4.12. Os nós do lado esquerdo da figura representam as *entradas* da rede, i.e., cada uma das covariáveis do banco de dados (neste caso, há três delas). Os nós da segunda camada são os chamados de nós da camada oculta da rede. Cada nó

nessa camada representa uma *transformação* dos nós das variáveis da camada anterior. Esta transformação é representada com mais detalhes na Figura 4.13: ela consiste em fazer uma combinação linear das entradas e então aplicar uma função f (chamada de *função de ativação*) já pré-definida ao resultado desta soma. Assim, para o caso da rede da Figura 4.12, se $\mathbf{x} = (x_1, x_2, x_3)$ é o vetor de entrada, então a saída de um dado neurônio j da camada oculta é dada por $x_j^1 := f(\beta_{0,j}^0 + \sum_{i=1}^3 \beta_{i,j}^0 x_i^0)$, em que $x_i^0 = x_i$ para $i = 1, 2, 3$. O índice superescrito nesta equação denota a camada da rede. Calculados os valores de x_j^1 para todo neurônio j da camada oculta, pode-se então calcular a saída do modelo. Esta é dada por $x_1^2 := f(\beta_{0,1}^1 + \sum_{i=1}^3 \beta_{i,1}^1 x_i^1)$.

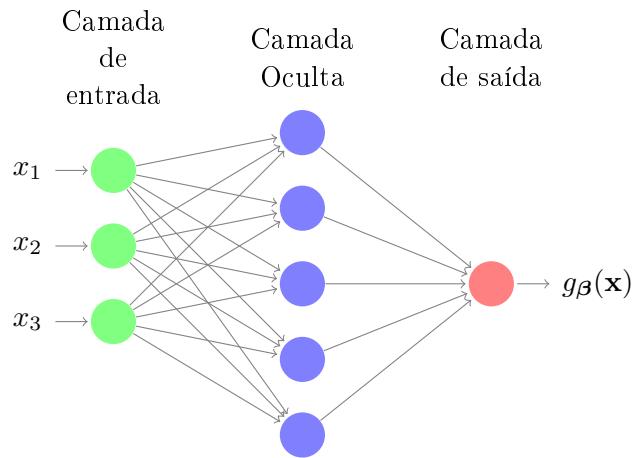


Figura 4.12: Exemplo de rede neural com uma camada oculta.

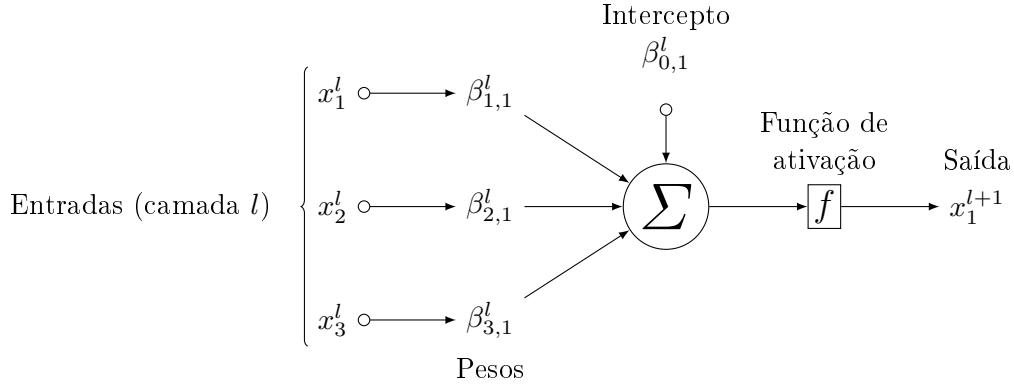


Figura 4.13: Exemplo do processamento que ocorre dentro de um neurônio situado na camada $l+1$. A saída x_1^{l+1} é dada por $f(\beta_{0,1}^l + \sum_{i=1}^3 \beta_{i,1}^l x_i^l)$

De forma geral, uma rede neural pode ter várias camadas ocultas. Considere uma rede com H camadas ocultas, cada uma com n_h neurônios ($h = 1, \dots, H$), seja $\beta_{i,j}^l$ o peso atribuído à conexão entre a entrada i na camada l e a saída j na camada $l+1$, $l = 0, \dots, H$. Aqui $h = 0$ denota a camada de entrada, e $H+1$ a camada de saída. O estimador de uma rede neural artificial para a função de regressão tem a forma

$$g(\mathbf{x}) = x_1^{H+1} = f(\beta_{0,1}^H + \sum_{i=1}^{n_H} \beta_{i,1}^H x_i^H) \quad (4.21)$$

em que, para todo $l = 1, \dots, H$ e $j = 1, \dots, n_l$,

$$x_j^{l+1} = f(\beta_{0,j}^l + \sum_{i=1}^{n_l} \beta_{i,j}^l x_i^l).$$

Note que se $f(z) = z$ e não há nenhuma camada oculta, uma rede neural nada mais é que uma regressão linear.

Uma rede neural pode ser representada via notação matricial. Para $l = 0, \dots, H$, seja

$$\mathbb{H}_l = \begin{pmatrix} \beta_{0,1}^l & \beta_{1,1}^l & \cdots & \beta_{n_l,1}^l \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{0,n_{l+1}}^l & \beta_{1,n_{l+1}}^l & \cdots & \beta_{n_l,n_{l+1}}^l \end{pmatrix}$$

Então o estimador da Equação 4.21 pode ser escrito como a seguinte composição de funções:

$$g(\mathbf{x}) = f(\mathbb{H}_H \dots \tilde{f}(\mathbb{H}_1 \tilde{f}(\mathbb{H}_0 \tilde{\mathbf{x}})) \dots),$$

em que $\tilde{\mathbf{x}} = (1, \mathbf{x})$ e $\tilde{f}(\mathbf{y}) = (1, f(\mathbf{y}))$. De forma mais geral, cada função f aplicada em cada camada pode ser diferente. Além disso, a saída da rede não precisa ser um número real.

4.11.1 Estimação: Backpropagation

TO DO: atualizar notação

Para estimar os coeficientes β_i^l na Equação 4.21, deve-se especificar uma função objetivo a ser minimizada. No caso de regressão linear, esta é em geral o erro quadrático médio:

$$EQM(g_{\beta}) = \frac{1}{n} \sum_{k=1}^n (g_{\beta}(\mathbf{x}_k) - y_k)^2,$$

em que a notação g_{β} é usada para enfatizar a dependência de g em seus parâmetros β . Pode-se também introduzir uma penalização a esse objetivo.

O vetor β que minimiza $EQM(g_{\beta})$ não possui solução analítica. Contudo, pode-se utilizar métodos numéricos para aproximá-lo. Um método que tem se destacado no contexto de redes neurais é o *backpropagation*, que nada mais é que um método de gradiente descendente executado em uma ordem específica: primeiro atualizam-se os coeficientes da última camada, depois se atualizam os coeficientes da camada anterior a essa, e assim por diante (por isso o nome backpropagation).

Mais especificamente: seja

$$\beta^l = (\beta_0^l, \dots, \beta_{n_l}^l).$$

Então, cada passo do backpropagation consiste em fazer, para $l = H, \dots, 0$, a seguinte atualização:

$$\beta_j^l \leftarrow \beta_j^l - \eta \frac{\partial EQM(g_{\beta})}{\partial \beta_j^l}, \quad j = 1, \dots, n_l$$

O truque é que $\frac{\partial EQM(g_{\beta})}{\partial \beta_j^l}$ depende apenas dos valores de β_j^i para $i \geq l$ e é simples de ser calculado. Por exemplo, para uma função de ativação $f(z) = z$ na camada de saída e $j > 0$,

$$\begin{aligned}\frac{\partial EQM(g_{\beta})}{\partial \beta_j^H} &= \frac{\partial}{\partial \beta_j^H} \left[\frac{1}{n} \sum_{k=1}^n (\beta_0^H + \sum_{i=1}^{n_H} \beta_i^H x_{k,i}^H - y_k)^2 \right] = \\ &= \frac{1}{n} \sum_{k=1}^n (\beta_0^H + \sum_{i=1}^{n_H} \beta_i^H x_{k,i}^H - y_k) \beta_j^H\end{aligned}$$

Para mais detalhes sobre o backpropagation veja, por exemplo, [Rojas \(2013\)](#). No R, redes neurais podem ser utilizadas utilizando-se o pacote `neuralnet`.

```
library(neuralnet)
fit=neuralnet(as.formula("y~X1+X2"), data=dados, hidden=c(2, 3),
              linear.output=TRUE)
```

Aqui, `hidden` indica quantos neurônios há em cada camada oculta (no exemplo, 2 na primeira e 3 na segunda); `linear.output` indica que $f(x) = x$. Caso deseje-se utilizar outra função f , pode-se usar o argumento `act.fct`. Predições podem ser feitas via a função `compute`, que também calcula a saída de cada um dos neurônios. Uma implementação mais completa de redes neurais pode ser encontrada no pacote `keras`.

4.11.2 Deep Learning

4.12 Um Pouco de Teoria

Quando um método não paramétrico funciona? Uma maneira de quantificar a qualidade de um método é avaliando sua (taxa de) convergência. Em particular, pode-se verificar sob quais circunstâncias um dado estimador converge. Investigaremos dois métodos: os k -vizinhos mais próximos (Seção 4.12.1) e séries ortogonais (Seção 4.12.2). Para estudar a taxa de convergência de um estimador não-paramétrico, é necessário supor que a função de regressão $r(\mathbf{x})$ real é *suave*; caso contrário ela pode ser extremamente difícil de ser estimada. Existem várias definições de suavidade ([Tsybakov, 2009](#)); utilizarmos uma noção diferente para cada uma das análises que seguem.

4.12.1 k -vizinhos Mais Próximos

Para analisar o método KNN, utilizarmos o conceito de funções Lipschitz para capturar a suavidade da função de regressão. Mais especificamente, iremos assumir que r é L -Lipschitz, isto é, assumiremos que existe L tal que, para todo $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$,

$$|r(\mathbf{x}) - r(\mathbf{x}')| \leq L\|\mathbf{x} - \mathbf{x}'\|_2.$$

Iniciamos com o seguinte resultado, que decorre do Lema 6.4 e do Teorema 6.2 de [Györfi and Krzyzak \(2002\)](#).

Lema 4.1 *Se \mathbf{X} é limitado e $d \geq 3$, então*

$$\mathbb{E} \left[\left(\frac{1}{k} \sum_{i \in \mathcal{N}_{\mathbf{x}}} \|\mathbf{X}_i - \mathbf{X}\|_2 \right)^2 \right] \leq C \left(\frac{k}{n} \right)^{2/d}.$$

Essencialmente, o Lema 4.1 indica como a distância média entre as k observações mais próximas a \mathbf{X} é limitada pelo tamanho amostral.

Teorema 4.4 *Seja $\hat{r}_k(\mathbf{X})$ o estimador da Equação 4.2. Sob as suposições do Lema 4.1 e se $\sup_{\mathbf{x}} \mathbb{E}[Y|\mathbf{x}] := \sigma^2 \leq \infty$ e r é L -Lipschitz, então*

$$\mathbb{E}[(\hat{r}_k(\mathbf{X}) - r(\mathbf{X}))^2] \leq K \left(\frac{k}{n} \right)^{2/d} + \frac{\sigma^2}{k},$$

em que K não depende de k nem n . Assim, se tomamos $k \asymp n^{\frac{2}{2+d}}$, obtemos

$$\mathbb{E}[(\hat{r}_k(\mathbf{X}) - r(\mathbf{X}))^2] \leq K' n^{-\frac{2}{2+d}}$$

Demonstração. Seja $\tilde{\mathbf{x}} = (\mathbf{x}_1, \dots, \mathbf{x}_n, \mathbf{x})$. Pela decomposição viés-variância (Seção 1.4.2), temos que

$$\begin{aligned} \mathbb{E}[(\hat{r}_k(\mathbf{x}) - r(\mathbf{x}))^2 | \tilde{\mathbf{x}}] &= (\mathbb{E}[\hat{r}_k(\mathbf{x}) | \tilde{\mathbf{x}}] - r(\mathbf{x}))^2 + \mathbb{E}[(\hat{r}_k(\mathbf{x}) - \mathbb{E}[(\hat{r}_k(\mathbf{x}) | \tilde{\mathbf{x}})])^2 | \tilde{\mathbf{x}}] \\ &= \left(\mathbb{E} \left[\frac{1}{k} \sum_{i \in \mathcal{N}_{\mathbf{x}}} Y_i | \tilde{\mathbf{x}} \right] - r(\mathbf{x}) \right)^2 + \mathbb{V}[\hat{r}_k(\mathbf{x}) | \tilde{\mathbf{x}}] \\ &\leq \left(\frac{1}{k} \sum_{i \in \mathcal{N}_{\mathbf{x}}} r(\mathbf{x}_i) - r(\mathbf{x}) \right)^2 + \frac{\sigma^2}{k} \\ &\leq \left(\frac{L}{k} \sum_{i \in \mathcal{N}_{\mathbf{x}}} \|\mathbf{x}_i - \mathbf{x}\|_2 \right)^2 + \frac{\sigma^2}{k}, \end{aligned}$$

em que a última desigualdade decorre do fato de r ser L -Lipschitz. Assim, pela lei da esperança total e do Lema 4.1, segue que

$$\begin{aligned}
\mathbb{E}[(\hat{r}_k(\mathbf{X}) - r(\mathbf{X}))^2] &= \mathbb{E}[\mathbb{E}[(\hat{r}_k(\mathbf{X}) - r(\mathbf{X}))^2 | \tilde{\mathbf{X}}]] \\
&\leq \mathbb{E}\left[\left(\frac{L}{k} \sum_{i \in \mathcal{N}_{\mathbf{X}}} \|\mathbf{X}_i - \mathbf{X}\|_2\right)^2\right] + \frac{\sigma^2}{k} \\
&\leq C' \left(\frac{k}{n}\right)^{2/d} + \frac{\sigma^2}{k}
\end{aligned}$$

□

Note que o Teorema 4.4 indica que, quando k é grande, o termo associado ao viés do estimador dado pelo KNN (i.e., $C'(k/n)^{2/d}$) é alto, enquanto que o termo associado à sua variância (i.e., σ^2/k) é baixo. Para se ter um bom equilíbrio entre esses termos, deve-se tomar $k \asymp n^{\frac{2}{2+d}}$. Discutiremos essa taxa de convergência mais a fundo na Seção 5.1; em particular argumentaremos que o limitante superior mostrado no Teorema 4.4 não pode ser melhorado, i.e., não se pode encontrar um limitante superior menor que o encontrado.

4.12.2 Séries Ortogonais

Nesta subseção iremos fornecer taxas de convergência para o método de séries ortogonais. Nos restringiremos ao caso em que $\mathbf{x} \in [0, 1]$ e tem densidade uniforme neste intervalo. Para estudar a convergência do estimador, novamente iremos também supor que a função de regressão $r(\mathbf{x})$ real é suave. Para esta seção, contudo, utilizaremos o conceito de derivadas para definir suavidade, uma vez que estas medem mudanças bruscas de $r(\mathbf{x})$ como função de \mathbf{x} . Diremos que $r(\mathbf{x})$ é suave se $r(\mathbf{x}) \in L^2([0, 1])$ e

$$\int_0^1 (D^m r(\mathbf{x}))^2 d\mathbf{x} \leq K_1.$$

Quanto maior m , mais suave r é. Este conceito de suavidade está ligado a espaços de Sobolev, veja [Wasserman \(2006\)](#).

Um resultado fundamental mostra que, se $r(\mathbf{x})$ é suave, usar apenas os primeiros termos da expansão $\sum_{j \in \mathbb{N}} \beta_j \psi_j(\mathbf{x})$ é suficiente para aproximar $r(\mathbf{x})$ bem. Isso está formalmente descrito no lema a seguir.

Lema 4.2 *Se $r(\mathbf{x}) \in L^2([0, 1])$ é tal que $\int_0^1 (D^m r(\mathbf{x}))^2 d\mathbf{x} \leq K_1$, então*

$$\int_0^1 \left(\sum_{j=0}^J \beta_j \psi_j(\mathbf{x}) - r(\mathbf{x}) \right)^2 d\mathbf{x} = \sum_{j \geq J+1} \beta_j^2 \leq \frac{K_1}{J^{2m}},$$

em que β_j 's são os coeficientes de expansão de $r(\mathbf{x})$ na base de Fourier $(\psi_j)_{j \geq 0}$.

Note que quanto maior m (a suavidade de r), menor o erro em se usar J termos para aproximar $r(\mathbf{x})$. É exatamente por isso que o método de séries ortogonais funciona bem: quando r é suave, um truncamento do tipo usado na Equação 4.1 leva a um viés baixo.

O resultado a seguir combina o viés obtido no Lemma 4.2 com a variância do estimador, de modo a obter a taxa de convergência do método de séries ortogonais.

Teorema 4.5 *Seja $\hat{r}_J(\mathbf{X})$ o estimador da Equação 4.1 com base na série de Fourier. Se $r(\mathbf{x}) \in L^2([0, 1])$ é tal que $\int_0^1 (D^m r(\mathbf{x}))^2 d\mathbf{x} \leq K_1$ e $\mathbb{V}[Y] < \infty$, tem-se que o risco deste estimador satisfaz*

$$\mathbb{E}[(\hat{r}_J(\mathbf{X}) - r(\mathbf{X}))^2] \leq K_1 \frac{1}{J^{2m}} + K_2 \frac{J}{n},$$

em que K_2 não depende de J nem n . Assim, se tomamos $J = Cn^{\frac{1}{2m+1}}$, obtemos

$$\mathbb{E}[(\hat{r}_J(\mathbf{X}) - r(\mathbf{X}))^2] \leq Kn^{-\frac{2m}{2m+1}}$$

Demonstração. Pela decomposição viés-variância (Seção 1.4.2) e pela lei da esperança total, temos que

$$\mathbb{E}[(\hat{r}_J(\mathbf{X}) - r(\mathbf{X}))^2] = \mathbb{E}\left\{\mathbb{E}\left[(r(\mathbf{X}) - \mathbb{E}[\hat{r}_J(\mathbf{X})|\mathbf{X}])^2 | \mathbf{X}\right]\right\} + \mathbb{E}\{\mathbb{V}[\hat{r}_J(\mathbf{X})|\mathbf{X}]\} \quad (4.22)$$

Notando que $\mathbb{E}[\hat{\beta}_j] = \beta_j$, temos que $\mathbb{E}[\hat{r}_J(\mathbf{x})] = \sum_{j=0}^J \beta_j \psi_j(\mathbf{x})$ e, assim, o termo de viés pode ser escrito como

$$\mathbb{E}\left\{\mathbb{E}\left[(r(\mathbf{X}) - \mathbb{E}[\hat{r}_J(\mathbf{X})|\mathbf{X}])^2 | \mathbf{X}\right]\right\} = \int_0^1 \left(\sum_{j>J} \beta_j \psi_j(\mathbf{x})\right)^2 dx \leq K_1 \frac{1}{J^{2m}}, \quad (4.23)$$

em que a última desigualdade segue do Lema 4.2.

Para desenvolver o termo da variância, note que, para todo $\mathbf{x} \in \mathbb{R}$,

$$\begin{aligned} \mathbb{V}\left[\sum_{j=0}^J \hat{\beta}_j \psi_j(\mathbf{x})\right] &= \mathbb{E}\left[\left(\sum_{j=0}^J \hat{\beta}_j \psi_j(\mathbf{x}) - \sum_{j=0}^J \beta_j \psi_j(\mathbf{x})\right)^2\right] = \mathbb{E}\left[\left(\sum_{j=0}^J (\hat{\beta}_j - \beta_j) \psi_j(\mathbf{x})\right)^2\right] \\ &= \sum_{i,j < J} \mathbb{E}[(\hat{\beta}_i - \beta_i)(\hat{\beta}_j - \beta_j)] \psi_i(\mathbf{x}) \psi_j(\mathbf{x}) \end{aligned}$$

Assim, usando novamente a ortogonalidade da base, o termo de variância pode ser decomposto como

$$\begin{aligned}\mathbb{E}\{\mathbb{E}[\mathbb{V}[\hat{r}_J(\mathbf{X})]|\mathbf{X}]\} &= \int_0^1 \sum_{i,j < J} \mathbb{E}[(\hat{\beta}_i - \beta_i)(\hat{\beta}_j - \beta_j)] \psi_i(\mathbf{x}) \psi_j(\mathbf{x}) dx \\ &= \sum_{j=0}^J \mathbb{E}[(\hat{\beta}_j - \beta_j)^2] = \sum_{j=0}^J \mathbb{V}[\hat{\beta}_j] = \sum_{j=0}^J \frac{1}{n} \mathbb{V}[Y \psi_j(\mathbf{X})] \leq K_2 \frac{J}{n},\end{aligned}\quad (4.24)$$

em que K_2 é uma constante que não depende de j e J , e a última desigualdade segue do fato que $|\psi_j(\mathbf{X})| < \sqrt{2}$ e que $\mathbb{V}[Y]$ é finita.

Assim, combinando as Equações 4.23 e 4.24 via Eq. 4.22, temos que

$$\mathbb{E}[(\hat{r}_J(\mathbf{X}) - r(\mathbf{X}))^2] \leq K_1 \frac{1}{J^{2m}} + K_2 \frac{J}{n}$$

□

O teorema anterior mostra que $n^{-\frac{2m}{2m+1}}$ é (um limitante superior para) a taxa de convergência do estimador baseado em séries ortogonais quando assumimos que $r(\mathbf{x})$ possui m derivadas integráveis e $\mathbf{x} \in \mathbb{R}$. Pode-se mostrar que tal limitante não poderia ser melhorado (veja a Seção 5.1). Além disso, quando $\mathbf{x} \in \mathbb{R}^d$, esta taxa é dada por $n^{-\frac{2m}{2m+d}}$.¹⁰ Pode-se mostrar que esta taxa é ótima sob estas suposições, isto é, nenhum estimador pode ter taxas melhores que esta. Mais precisamente, $n^{-\frac{2m}{2m+d}}$ é dito ser a taxa minimax dentro desta classe (Tsybakov, 2009). Discutiremos isso mais a fundo na Seção 5.1.

4.13 Exemplos

4.13.1 Esperança de Vida e PIB per Capita

Retomamos aqui o exemplo visto nos Exemplos 1.1, 1.5 e 1.6. Os resultados estão na Tabela 4.1 e na Figura 4.14. Como o conjunto de dados deste exemplo é relativamente pequeno, ao invés de data splitting utilizamos leave-one-out cross validation para ajustar os *tuning parameters*.

¹⁰ Para isso, assumimos, por exemplo, que todas as derivadas parciais de ordem m de r são L -Lipschitz (Györfi and Krzyzak, 2002).

Tabela 4.1: Resultados dos métodos não paramétricos no exemplo da Seção 4.13.1.

Método	Séries	KNN	NW	Polinômio grau 1	Polinômio grau 2
Risco Estimado	31.15	31.45	32.25	36.70	39.64

Método	B Splines	Natural Splines	Smoothing Splines	SVR
Risco Estimado	31.15	30.42	31.25	32.08

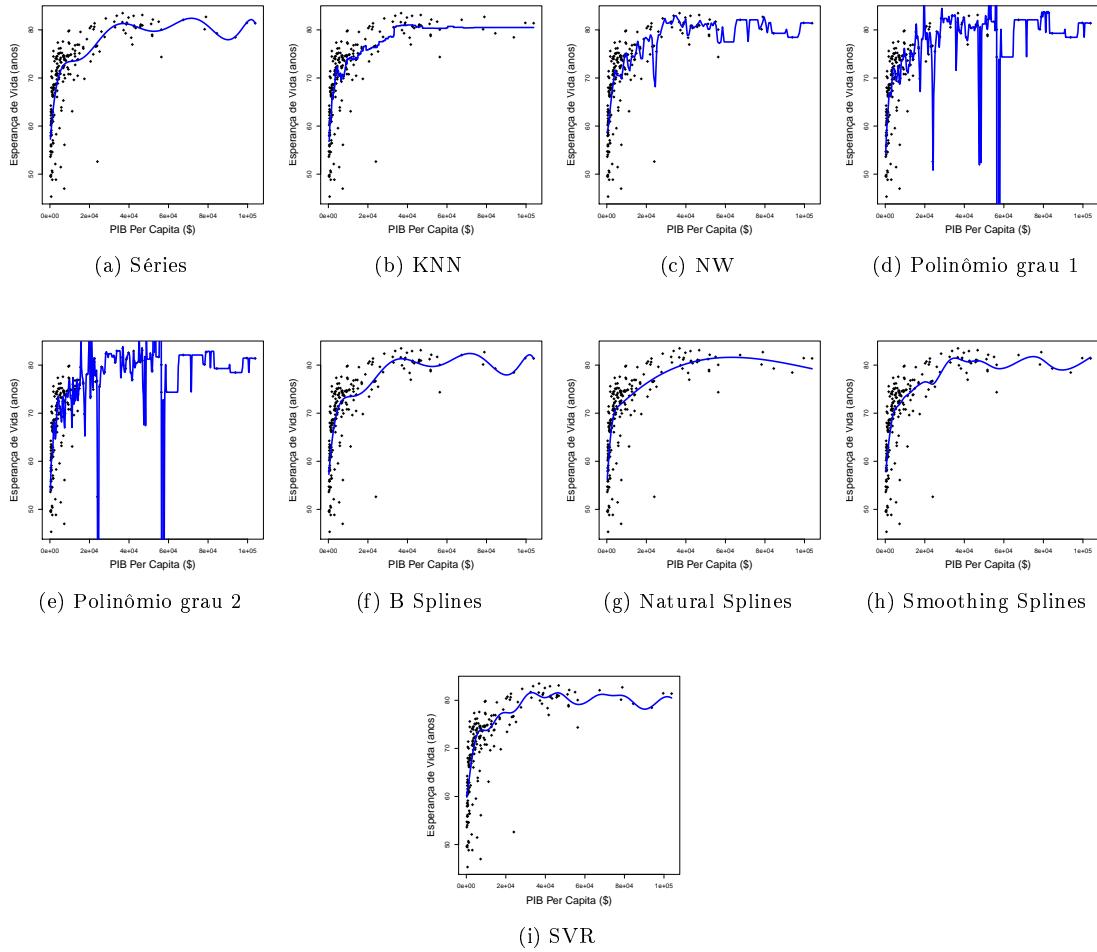


Figura 4.14: Predições dos modelos ajustados em um conjunto de teste para o exemplo da Seção 4.13.1.

4.13.2 Exemplo em Duas Dimensões

Simulamos aqui 600 observações de

$$Y_i = X_{i,1}^2 + \cos(\pi X_{i,2}) + 0.1X_{i,1} * X_{i,2} + \epsilon_i,$$

com $X_{i,1}$ e $X_{i,2} \sim N(0, 1)$, e $\epsilon_i \sim N(0, 0.1^2)$. Comparamos os estimadores descritos neste capítulo, com todos os parâmetros escolhidos via validação cruzada. O modelo aditivo foi composto por smoothing splines com graus de liberdade escolhido via validação cruzada. Os resultados obtidos encontram-se na Tabela 4.2, e na Figura 4.15. Para este conjunto de dados, polinômios locais de grau maior que 1 apresentaram resultados muito superiores ao tradicional Nadaraya-Watson. Modelos aditivos também apresentaram resultados satisfatórios quando comparados aos demais.

Tabela 4.2: Resultados dos métodos não paramétricos no exemplo da Seção 4.13.2.

Método	Séries	KNN	NW	SVR	Árvores	Florestas
Risco Estimado	0.03 (0.001)	0.08 (0.02)	0.06 (0.01)	0.03 (0.01)	0.39 (0.07)	0.07 (0.02)

Método	Polinômio grau 1	Polinômio grau 2	Polinômio grau 3	Modelo Aditivo
Risco Estimado	0.06 (0.01)	0.01 (0.001)	0.03 (0.01)	0.02 (0.001)

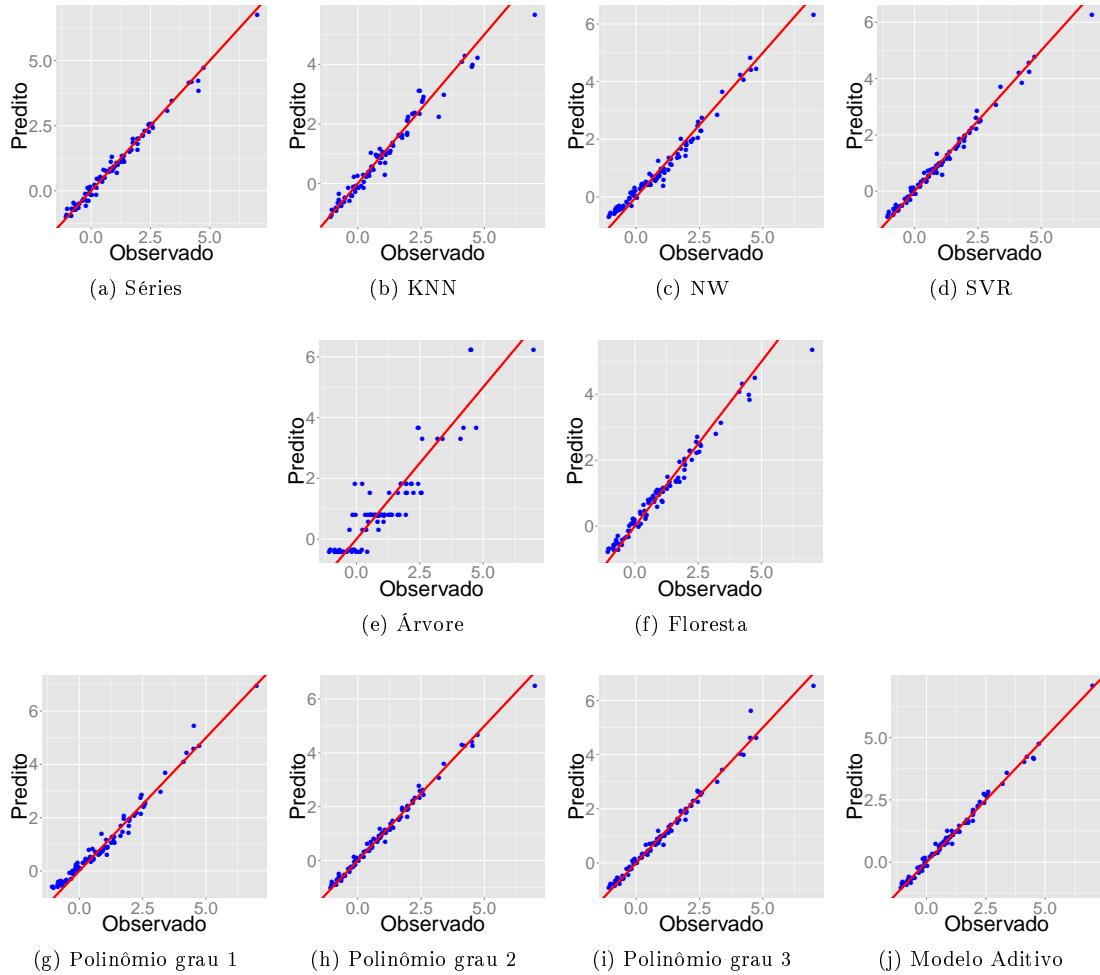


Figura 4.15: Predições dos modelos ajustados em um conjunto de teste para o exemplo da Seção 4.13.2.

4.14 Resumo

Neste capítulo estudamos algumas classes de métodos não paramétricos que têm motivações muito distintas. Vimos que a vantagem de tais métodos é sua flexibilidade quando comparada com estimadores paramétricos: não é necessário assumir uma forma paramétrica para $r(\mathbf{x})$ para garantir a consistência de um estimador não paramétrico $\hat{r}(\mathbf{x})$;

necessita-se apenas que $r(\mathbf{x})$ seja suave (o que pode ser medido de diferentes maneiras). Esta flexibilidade, contudo, vem com o custo de requerer amostras maiores para se obter taxas de convergências razoáveis: enquanto que métodos paramétricos tipicamente possuem taxas de convergência n^{-1} quando suas suposições são válidas, a taxa usual de métodos não paramétricos é $n^{-\frac{\alpha}{\alpha+d}}$ (em que α é a suavidade de r), que é muito mais lenta. Este problema é ainda mais notório quando o número de covariáveis d é grande, assunto que será abordado no próximo capítulo.

Capítulo 5

Métodos Não Paramétricos em Dimensões Altas

Como vimos no capítulo anterior, métodos não paramétricos, apesar de mais flexíveis, exigem tamanhos de amostras maiores quando comparados com métodos paramétricos, caso contrário possuem taxas de convergência muito lentas. Neste capítulo estudamos como tais métodos se comportam em dimensões altas. Veremos, em particular, que só poderemos fazer estimação não paramétrica satisfatoriamente se fizermos suposições que simplificam o problema. Em particular, duas suposições usuais que frequentemente valem na prática são esparsidade e redundância nas covariáveis.

5.1 Taxas de convergência e a maldição da dimensionalidade

A *maldição da dimensionalidade* (Bellman, 1961) é um problema enfrentado por estimadores em dimensões altas. Essencialmente, o problema é que, sem restrições adicionais, uma função de regressão fica cada vez mais difícil de se estimar conforme a dimensão do espaço das covariáveis cresce.

Mais precisamente: considere que temos d covariáveis e um tamanho amostral n . Argumentamos no Capítulo 4 que o risco de um estimador não-paramétrico \hat{r}^* frequentemente satisfaz

$$\mathbb{E} [(\hat{r}^*(\mathbf{X}) - r(\mathbf{X}))^2] \leq \frac{K}{n^{\alpha/(\alpha+d)}}$$

para todo $r \in S$, em que K é uma constante e S é um conjunto de regressões suaves (por exemplo, com derivadas parciais de segunda ordem L-Lipschitz; o parâmetro α está ligado à noção de suavidade usada). Assim, vale que

$$\sup_{r \in S} \mathbb{E} [(\hat{r}^*(\mathbf{X}) - r(\mathbf{X}))^2] \leq \frac{K}{n^{\alpha/(\alpha+d)}}. \quad (5.1)$$

Pode-se também mostrar que

$$\inf_{\hat{r}} \sup_{r \in S} \mathbb{E} [(\hat{r}(\mathbf{X}) - r(\mathbf{X}))^2] = \frac{K}{n^{4/(4+d)}},$$

i.e., $\frac{K}{n^{\alpha/(\alpha+d)}}$ é a taxa minimax para esse problema. Este resultado, juntamente com a Equação 5.1, implica que

1. $\sup_{r \in S} \mathbb{E} [(\hat{r}^*(\mathbf{X}) - r(\mathbf{X}))^2] = \frac{K}{n^{\alpha/(\alpha+d)}}$ (i.e., a desigualdade na Equação 5.1 é na realidade uma igualdade)
2. Se um estimador \hat{r}^* satisfaz a Equação 5.1 ele é ótimo em um sentido minimax, isto é, o risco mais alto que ele atinge para funções na classe S é o menor (pior) risco que poderia ser obtido por qualquer estimador. Isto é, $\sup_{r \in S} \mathbb{E} [(\hat{r}^*(\mathbf{X}) - r(\mathbf{X}))^2] \leq \sup_{r \in S} \mathbb{E} [(\hat{r}(\mathbf{X}) - r(\mathbf{X}))^2]$ para qualquer estimador \hat{r} .

Segue da Equação 5.1 que, para conseguirmos obter um risco de no máximo δ , precisamos de uma amostra de tamanho ao menos

$$n \approx \left(\frac{K}{\delta} \right)^{(\alpha+d)/\alpha}.$$

Este crescimento exponencial indica que, mesmo em dimensões moderadas, estimar satisfatoriamente uma função de regressão requer uma amostra grande. Veja a Figura 5.1, que ilustra como o risco varia como função do tamanho amostral n para diferentes dimensões d e para $\alpha = 4$.

Suponha que desejamos estimar a função de regressão em um dado ponto \mathbf{x} . Em termos intuitivos, a maldição da dimensionalidade ocorre pois, quando d é grande, a probabilidade que exista *algum* elemento na amostra com covariáveis \mathbf{X}_i é extremamente baixa mesmo para tamanhos de amostra moderados. Em outras palavras, *a vizinhança de \mathbf{x} com alta probabilidade é vazia*. Logo, podemos aprender muito pouco sobre $r(\mathbf{x})$. Veja mais detalhes em [Wasserman \(2006\)](#).

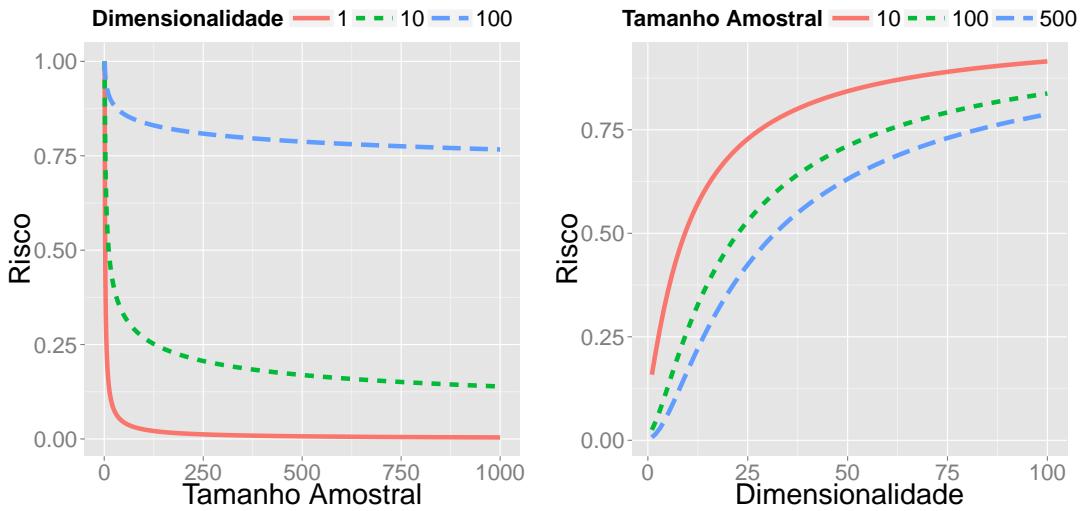


Figura 5.1: A maldição da dimensionalidade: quanto maior o número de covariáveis d , mais difícil é estimar r . Aqui tomamos $\alpha = 4$.

Assim, estimar uma função de regressão em problemas com dimensionalidade alta é um problema notoriamente difícil. O fenômeno da maldição da dimensionalidade indica que, sem suposições extras, é impossível resolver esse problema satisfatoriamente para tamanho de amostras realísticos. Aqui exploraremos duas dessas suposições: alta esparsidade e alta redundância entre as covariáveis.

5.1.1 Esparsidade

A suposição de esparsidade diz que a função de regressão $r(\mathbf{x})$ depende apenas de alguns x_i 's. Isto é,

$$r(\mathbf{x}) = r((x_i)_{i \in S}),$$

em que $S \subset \{1, \dots, d\}$. Assim, apesar de haver várias covariáveis, na realidade poucas delas são relevantes para o problema. Assim, estimar $r(\mathbf{x})$ pode não ser tão difícil.

5.1.2 Redundância

A suposição de redundância diz, intuitivamente, que as covariáveis \mathbf{x} são altamente redundantes. Um exemplo trivial é o de que $x_1 = \dots = x_d$, i.e., todas as covariáveis são iguais. Mais realisticamente, considere que x_1 é a altura de um indivíduo, x_2 é seu peso e x_3 é seu índice de massa corpórea. Com quaisquer duas dentre essas três variáveis, conseguimos recuperar a terceira, de modo que essas variáveis podem trivialmente ser reduzidas para apenas duas.

Outro caso em que redundância costuma ser uma suposição adequada é o de imagens. Neste contexto, cada covariável x_i é o pixel de uma figura (veja Seção A.1 para uma explicação do que são pixels). Espera-se, em geral, que pixels adjacentes tenham valores próximos, de modo que há, assim, muita redundância nas covariáveis associadas a uma imagem.

Existem diversas formas de formalizar a noção de redundância; uma muito usada é a de que \mathbf{X} pertence a uma subvariedade de \mathbb{R}^d com dimensionalidade intrínseca baixa ([Aswani et al., 2011](#)).

A seguir veremos alguns métodos não paramétricos que são adequados para problemas com alta dimensionalidade.

5.2 k Vizinhos Mais Próximos e Regressão Linear Local

Tanto o estimador dos k vizinhos mais próximos (Seção 4.3) quanto a regressão linear local (Seção 4.5 para o caso $p = 1$) se adaptam automaticamente à dimensionalidade intrínseca das observações quando estas vivem em uma subvariedade de \mathbb{R}^d . Veja, por exemplo, [Kpotufe \(2011\)](#) e [Bickel and Li \(2007\)](#). Assim, quando se utiliza um destes estimadores, nenhuma adaptação é necessária para evitar a maldição da dimensionalidade quando x_i 's são altamente redundantes. Mais precisamente, a taxa de convergência de ambos os estimadores é $\frac{K}{n^{4/(4+u)}}$, em que u é a dimensão intrínseca da subvariedade em que \mathbf{X} vive. Essa taxa é muito mais rápida que a vista anteriormente, $\frac{K}{n^{4/(4+d)}}$, especialmente se $u \ll d$.

5.3 Support Vector Regression

Support Vector Regression (Seção 4.6.4) com o kernel Gaussiano também têm bom desempenho sob a hipótese de redundância. Veja, por exemplo, [Eberts and Steinwart \(2011\)](#) e [Steinwart and Christmann \(2008\)](#).

5.4 Séries

A abordagem de séries ortogonais abordada na Seção 4.1 também pode ser adaptada para o cenário de dimensionalidade alta. Nos focamos aqui na abordagem de séries espetrais.

5.4.1 Bases Espectrais

Nesta seção estudamos uma extensão do método de séries ortogonais estudado na Seção 4.1 que é mais adequada para dados com dimensionalidade alta. O estimador também consiste em uma expansão em termos de uma base ortogonal. Contudo, ao contrário do que foi estudado na Seção 4.1, aqui a base utilizada é construída com base nos dados. Em outras palavras, ao invés de usar, por exemplo, uma base de Fourier, usamos uma base $\{\psi_j\}_{j \in \mathbb{N}}$ construída com base na amostra $\mathbf{x}_1, \dots, \mathbf{x}_n$.

Mais especificamente, para se construir uma base espectral, começamos definindo um kernel de Mercer $K(\mathbf{x}, \mathbf{y})$ (veja Seção 4.6.1) Seja $P(\mathbf{x})$ a distribuição do vetor de covariáveis \mathbf{X} . Considere o seguinte operador (Shi et al., 2009):

$$\begin{aligned} \mathbf{K} : \mathcal{L}^2(\mathcal{X}, P) &\longrightarrow \mathcal{L}^2(\mathcal{X}, P) \\ \mathbf{K}(g)(\mathbf{x}) &= \int_{\mathcal{X}} K(\mathbf{x}, \mathbf{y}) g(\mathbf{y}) dP(\mathbf{y}) \end{aligned} \quad (5.2)$$

O operador \mathbf{K} tem uma quantidade enumerável de autofunções $\{\psi_j\}_{j \in \mathbb{N}}$ com respectivos autovalores $\lambda_1 \geq \lambda_2 \geq \dots \geq 0$ (Minh et al., 2006).

As autofunções $\{\psi_j\}_{j \in \mathbb{N}}$ constituem uma *base espectral*. Elas formam uma base orthonormal de $\mathcal{L}^2(\mathcal{X}, P)$ (Minh, 2010).

Há duas principais razões pelas quais bases espetrais são candidatas ideais para aproximar funções suaves de \mathbf{x} em dimensões altas:

1. As autofunções $\{\psi_j\}_{j \in \mathbb{N}}$ são *adaptadas à dimensão intrínseca dos dados*. Mais especificamente, quando o domínio \mathcal{X} é uma subvariedade de \mathbb{R}^d , a base se comporta como uma base de Fourier adaptada à geometria intrínseca do dados, em que os primeiros termos são mais suaves que os termos de ordem mais alta¹. Como um exemplo, na Figura 5.2 mostramos as autofunções do operador da Equação 5.2 quando o domínio dos dados é uma subvariedade (uma espiral) de \mathbb{R}^2 . Compare esta figura com a Figura 4.1; a base se comporta como uma base de Fourier na direção da espiral. Segue que

¹ Isto ocorre porque a distância euclidiana é localmente a mesma que a distância geodésica; veja, e.g., Shi et al. 2009 para uma derivação.

se $r(\mathbf{x})$ é suave com relação a esse domínio, então só precisamos de algumas poucas funções para aproximá-la bem. Esta adaptação leva a taxas de convergência que dependem apenas da dimensão intrínseca dos dados, ao invés da potencialmente maior dimensão ambiente.

2. Ao contrário das bases ortogonais tradicionais, as autofunções são ortogonais *com relação a* $P(\mathbf{x})$, a distribuição marginal dos dados, e não com relação à medida de Lebesgue (Bengio et al., 2004). Isto é,

$$\int_{\mathcal{X}} \psi_i(\mathbf{x}) \psi_j(\mathbf{x}) dP(\mathbf{x}) = \delta_{i,j}.$$

Isto leva a estimadores dos coeficientes de expansão que são mais rapidamente calculados. Além disso, não há necessidade de usar produtos tensoriais em dimensões altas, que são extremamente ineficientes para serem calculados.

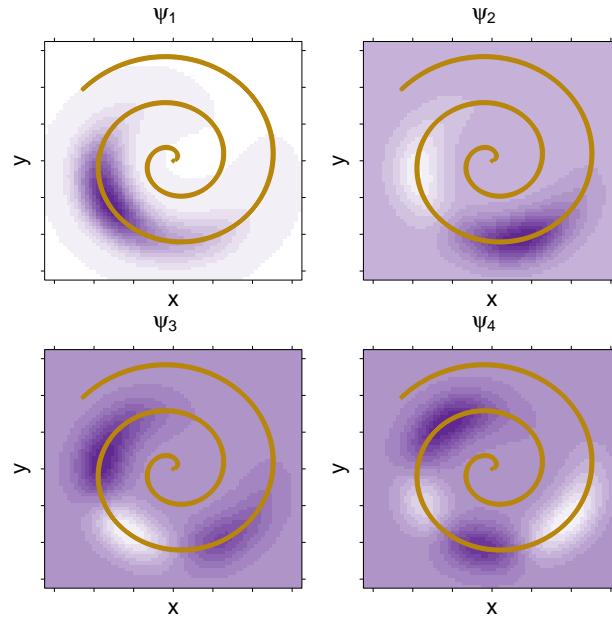


Figura 5.2: Curvas de nível das primeiras quatro autofunções do operador do kernel Gau- siano quando o domínio das covariáveis $\mathbf{x} = (x_1, x_2)$ é em uma espiral. As autofunções formam uma base de Fourier adaptada à geometria dos dados, e são apropriadas para aproximar funções suaves de \mathbf{x} neste domínio. Compare esta figura com a Figura 4.1.

Como $P(\mathbf{x})$ é desconhecido, é necessário estimar $\{\psi_j\}_j$. Isso pode ser feito primeiramente calculando-se a matriz de Gram

$$\mathbf{G} = \begin{bmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & K(\mathbf{x}_1, \mathbf{x}_2) & \cdots & K(\mathbf{x}_1, \mathbf{x}_n) \\ K(\mathbf{x}_2, \mathbf{x}_1) & K(\mathbf{x}_2, \mathbf{x}_2) & \cdots & K(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ K(\mathbf{x}_n, \mathbf{x}_1) & K(\mathbf{x}_n, \mathbf{x}_2) & \cdots & K(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix} \quad (5.3)$$

Seja

$$\tilde{\psi}_j := (\tilde{\psi}_j(\mathbf{x}_1), \dots, \tilde{\psi}_j(\mathbf{x}_n))$$

o j -ésimo autovetor da matrix 5.3, e seja \hat{l}_j seu respectivo autovalor, em que ordenamos os autovetores segundo ordem decrescente de autovalores, e os normalizamos de maneira que $\sum_{k=1}^n \tilde{\psi}_j^2(\mathbf{x}_k) = 1$. Um estimador consistente de ψ_j é

$$\hat{\psi}_j(\mathbf{x}) = \frac{\sqrt{n}}{\hat{l}_j} \sum_{k=1}^n \tilde{\psi}_j(\mathbf{x}_k) K(\mathbf{x}, \mathbf{x}_k). \quad (5.4)$$

Este estimador é a extensão de Nyström do autovetor $\tilde{\psi}_j$ para valores fora da amostra \mathbf{x} ([Bengio et al., 2004](#), [Drineas and Mahoney, 2005](#)).

5.4.2 O Estimador

Seja K um kernel fixo, e seja $\{\psi_j\}_{j \in \mathbb{N}}$ a base ortonormal relacionado ao operador da Equação (5.2). A expansão da função de regressão r nessa base é dada por

em que

$$\beta_j = \int_{\mathcal{X}} \psi_j(\mathbf{x}) r(\mathbf{x}) dP(\mathbf{x}) = \int_{\mathcal{X}} \psi_j(\mathbf{x}) \mathbb{E}[Z|\mathbf{x}] dP(\mathbf{x}) = \mathbb{E}[Z\psi_j(\mathbf{X})]$$

Note que a ortogonalidade da base espectral com relação a $P(\mathbf{x})$ é a chave para que β_j seja simplesmente $\mathbb{E}[Z\psi_j(\mathbf{X})]$.

Assim, o estimador baseado em séries espetrais ([Lee and Izbicki, 2016](#)) é dado por

$$\hat{r}(\mathbf{x}) = \sum_{j=1}^J \hat{\beta}_j \hat{\psi}_j(\mathbf{x}), \quad (5.5)$$

em que $\widehat{\psi}_j$'s são estimados como descrito anteriormente e

$$\widehat{\beta}_j = \frac{1}{n} \sum_{k=1}^n z_k \widehat{\psi}_j(\mathbf{x}_k).$$

O corte J pode ser escolhido por validação cruzada.

5.5 Florestas Aleatórias

5.6 SpAM - Modelos Aditivos Esparsos

Os modelos aditivos apresentados na Seção 4.7 são uma tentativa de diminuir a influência da maldição da dimensionalidade, uma vez que eles assumem que $r(\mathbf{x})$ pode ser decomposta como uma soma linear de funções suaves de cada uma das covariáveis. Os modelos aditivos esparsos (SpAM - Sparse Additive Models; Ravikumar et al. 2009) vão um passo além: eles combinam modelos aditivos com o lasso (Seção 3.3) de modo a se obter um estimador não paramétrico que vença a maldição da dimensionalidade em alguns problemas.

Mais especificamente, em um modelo aditivo esparso, busca-se um representação da função de regressão da forma

$$r(\mathbf{x}) = \sum_{j=1}^d \beta_j g_j(x_j),$$

em que impõe-se que $\sum_{j=1}^d |\beta_j| \leq L$. Isto é, como em modelos aditivos, o SpAM assume que $r(\mathbf{x})$ pode ser decomposta como uma soma linear de funções suaves de cada uma das covariáveis, $g_j(x_j)$, mas, ao mesmo tempo, requer que algumas destas funções sejam zero. Modelos aditivos esparsos são ajustados utilizando-se uma combinação do backfitting (Seção 4.7) com um método de solução do lasso. Assim como no lasso, o valor de L pode ser escolhido via validação cruzada. Veja detalhes técnicos em Ravikumar et al. (2009).

No R, o método SpAM pode ser implementado com o pacote SAM. O ajuste e predições podem ser feitos via

```
library(SAM)
ajuste = samQL(X=xTreinamento, y=yTreinamento)
predicoes=predict(ajuste, xTest)
```

Exemplo 5.1 Aqui exemplificamos modelos aditivos esparsos para o problema de prever a qualidade de um vinho (medida através de uma nota dada por um *expert*)

segundo suas característica (pH, quantidade de álcool entre outras, veja mais detalhes em <https://archive.ics.uci.edu/ml/datasets/Wine+Quality>). As funções suaves g_j utilizadas foram splines com 3 funções de base.

A Figura 5.3 mostra os resultados de tal ajuste. Assim como o caso de modelos aditivos, pode-se observar que o SpAM leva a ajustes altamente interpretáveis.

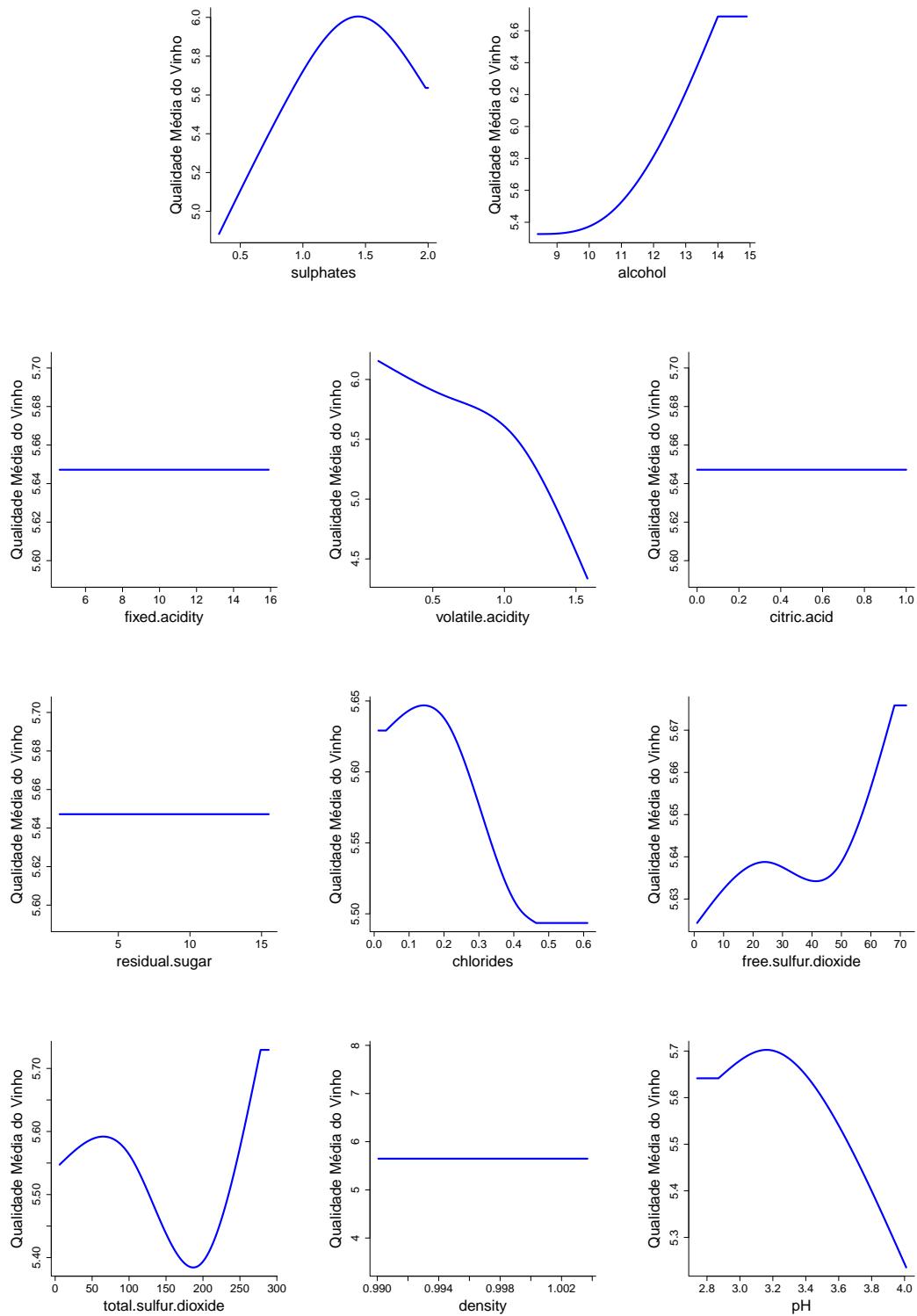


Figura 5.3: Ajuste dado pelo modelo esparso aditivo para os dados de vinho (Exemplo 5.1).

□

5.7 Exemplos

5.7.1 Isomap face data

Trabalhamos aqui com os dados descritos no Exemplo 1.3, que tem por objetivo estimar a direção horizontal para a qual um indivíduo está olhando com base em sua imagem. Cada imagem aqui é representada por uma matriz 64×64 , i.e., há $d = 4096$ covariáveis. Temos $n = 698$ observações. Note que, como $n < d$, não é possível implementar o método dos mínimos quadrados para esses dados. A Tabela 5.1 e a Figura 5.4 mostram os resultados dos ajustes desses modelos. Neste problema, os métodos paramétricos usados levam a bons resultados, mas as previsões obtidas pelos métodos não paramétricos têm melhor qualidade. Em particular, o método baseado em séries espectrais foi o que teve melhor desempenho.

Tabela 5.1: Riscos estimados e erros-padrão de alguns estimadores para o exemplo da Seção 5.7.1.

Método	Séries Espectrais	KNN	NW	Lasso	Ridge
Risco Estimado	2.70 (0.70)	10.09 (1.50)	11.91 (1.91)	27.69 (6.43)	56.76 (13.48)

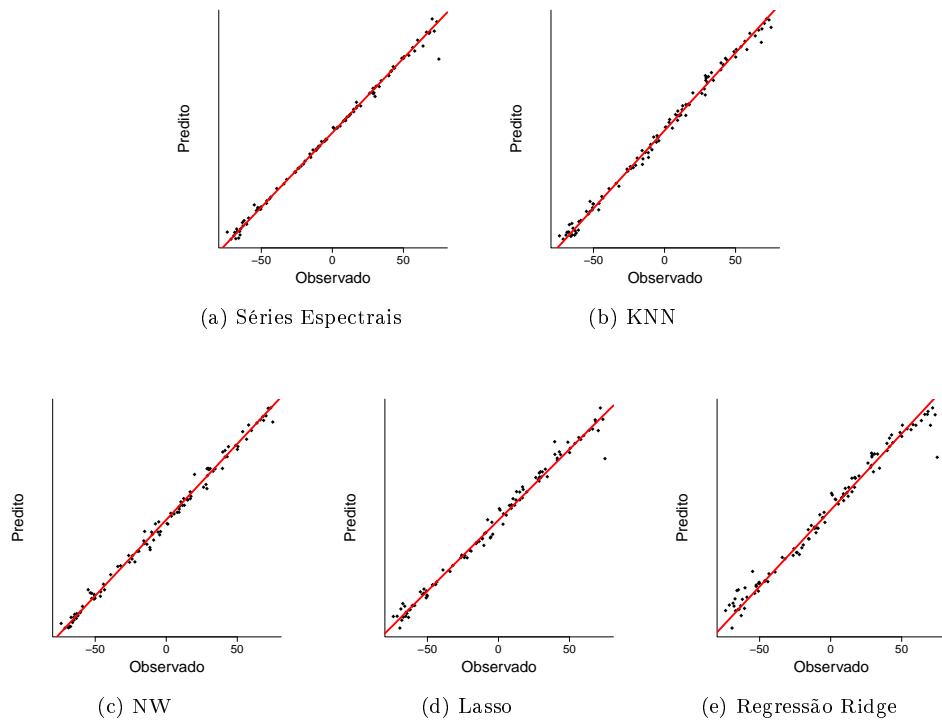


Figura 5.4: Predições dos modelos ajustados em um conjunto de teste para o exemplo da Seção 5.7.1.

5.8 Resumo

Neste capítulo, vimos que é necessário fazer suposições adicionais para que seja possível estimar uma função de regressão de modo não paramétrico quando há muitas covariáveis. As duas suposições investigadas foram “irrelevâncias de algumas covariáveis” e “redundância das covariáveis”. Vimos que vários dos métodos estudados no Capítulo 4 automaticamente se adaptam para essas situações, no sentido de que as taxas de convergência destes modelos são muito mais rápidas se essas suposições de fato são realistas. Vimos também alguns novos métodos especialmente desenhados para lidar bem com elas.

Capítulo 6

Outros Aspectos de Regressão

6.1 Interpretabilidade

Ainda que o enfoque deste livro seja criar bons algoritmos de predição, métodos que permitem maior interpretabilidade costumam ser mais utilizados, uma vez que trazem mais segurança para os usuários destes modelos, além de trazer *insights* adicionais sobre o problema abordado. Por exemplo, entendendo que variáveis são importantes para se obter um bom poder preditivo em um dado problema pode indicar que o classificador obtido irá ter bom poder de generalização para novos dados com características possivelmente diferentes.

Vimos que árvores de predição, modelos aditivos e modelos lineares esparsos são relativamente fáceis de serem interpretados, enquanto que support vector regression e métodos baseados no truque do kernel tipicamente são mais difíceis. Existem algumas formas de tentar interpretar modelos de predição que são caixas pretas. Aqui apresentamos o LIME ([Ribeiro et al., 2016](#)).

A ideia chave do LIME é tentar responder à pergunta: por que o algoritmo me forneceu a predição $g(\mathbf{x}^*)$ para a nova amostra \mathbf{x}^* ? A resposta para tal pergunta vem na forma de uma lista de quais variáveis foram importantes para explicar essa observação. Para tanto, o LIME aproxima a solução $g(\mathbf{x}^*)$ *localmente* (i.e., em uma vizinhança de \mathbf{x}^*) via uma regressão linear ajustada por lasso. Isso é feito pois, mesmo se g é não linear, localmente uma aproximação linear pode ser razoável. O procedimento do LIME para fazer essa aproximação consiste nos seguintes passos:

- Gerar as covariáveis de novas observações $\mathbf{x}_1^*, \dots, \mathbf{x}_B^*$ perturbando \mathbf{x}^* (e.g., adicionando ruído em cada dimensão de \mathbf{x}^*)
- Ajustar o lasso para o conjunto $(\mathbf{x}_1^*, g(\mathbf{x}_1^*)), \dots, (\mathbf{x}_B^*, g(\mathbf{x}_B^*))$ O ajuste pode ser feito dando pesos para cada observação de acordo com a similaridade: $w_i = K(\mathbf{x}_i^*, \mathbf{x}^*)$. O valor da penalização λ do lasso é escolhido de forma que se tenha m covariáveis

escolhidas (m é escolhido pelo usuário e em geral é um número pequeno para facilitar a interpretação).

Retorna-se então quais foram as m variáveis escolhidas.

6.2 Individual Sequence Predictions

([Cesa-Bianchi and Lugosi, 2006](#)) Objetivo: predizer uma sequência $y_1, y_2, \dots, y_t, \dots$ de elementos que pertencem a um conjunto \mathcal{Y} . As previsões para cada um destes valores serão denotadas por $\hat{p}_1, \hat{p}_2, \dots, \hat{p}_t, \dots$, respectivamente. Elas pertencem a um espaço \mathcal{D} . As previsões serão feitas de maneira sequencial. Para criá-las, contaremos com um conjunto de *experts*. Mais precisamente, no tempo t , temos acesso a um conjunto de previsões $f_{1,t}, \dots, f_{k,t}$, dadas por k experts. Nossa objetivo é combinar estas previsões de como a criar nossa previsão para y_t , \hat{p}_t . Após \hat{p}_t ser calculado, o valor de y_t , e devemos então criar nossa previsão para y_{t+1} com base nas novas previsões dos experts, $f_{1,t+1}, \dots, f_{k,t+1}$. Para avaliar a performance de uma previsão \hat{p}_t , assumiremos que uma função de perda $l(\hat{p}_t, y_t)$ está definida.

Uma quantidade que possui papel fundamental na teoria de [Cesa-Bianchi and Lugosi](#) é o *arrependimento com relação a um dado expert*. Para um expert j , este é definido, após n observações, como sendo

$$R_{j,n} = \sum_{t=1}^n [l(\hat{p}_t, y_t) - l(f_{j,t}, y_t)].$$

Isto é $R_{j,n}$ avaliar o quanto boas nossas previsões (i.e., $\hat{p}_1, \dots, \hat{p}_n$) foram com relação às previsões do expert j (i.e., $f_{j,1}, \dots, f_{j,n}$) para as n primeiras observações.

Uma maneira de se construir as previsões $\hat{p}_1, \dots, \hat{p}_n$ é através de médias ponderadas entre as previsões fornecidas pelos k experts:

$$\hat{p}_t = \frac{\sum_{j=1}^k w_{j,t} f_{j,t}}{\sum_{j=1}^k w_{j,t}}, \quad (6.1)$$

onde $w_{j,t} \geq 0$ é o peso dado ao expert j no instante t . Note que o denominador em 6.1 é usado para normalizar os pesos de modo que eles somem um.

- (Peso Polinomial) $w_{j,t} = (R_{j,t-1})_+^{p-1}$, onde p deve ser escolhido¹. Uma escolha razoável motivada por considerações teóricas é $p \approx 2 \log_e(k)$.

¹ Aqui, $(x)_+$ vale x quando $x > 0$, e 0 caso contrário.

- (Peso Exponencial) $w_{j,t} = \exp(\eta R_{j,t-1})$, onde η deve ser escolhido. Duas escolhas razoáveis motivadas por considerações teóricas são $\eta = \sqrt{2 \log_e(k)/n}$ e $\eta = \sqrt{8 \log_e(k)/n}$.

Logarithmic Loss

Sequential Probability Assignment

Linear Classification - zero one loss

6.3 Estimação de Densidades

Apesar do problema de regressão desempenharem papel central nos dias de hoje, outras tarefas também são importantes (inclusive muitas vezes são utilizadas na construção de estimadores de regressão e de classificadores).

6.4 Estimação de Densidades Condicionais

Nem sempre a função de regressão é suficiente para resumir a incerteza sobre Y quando se conhece \mathbf{x} . Uma alternativa é estimar não só a esperança condicional $\mathbb{E}[Y|\mathbf{x}]$, mas sim toda a densidade condicional $f(y|\mathbf{x})$. Alguns métodos não paramétricos de estimação de tal quantidade em problemas com baixa dimensionalidade podem ser encontrados em Rosenblatt (1969), Hyndman et al. (1996), Fan et al. (1996), Sugiyama et al. (2010), Hall et al. (2004). Para métodos que funcionam em alta dimensionalidade, veja Izbicki and Lee (2016) e Izbicki and Lee (2017).

Parte II

Classificação

Capítulo 7

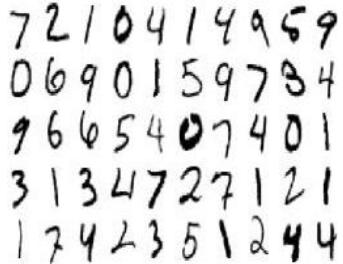
Introdução

Neste capítulo, estudamos outro problema central em aprendizado de máquina, o de classificação. Trata-se de um problema similar ao de predição em regressão: observamos uma amostra com observações independentes $(\mathbf{X}_1, Y_1), \dots, (\mathbf{X}_n, Y_n) \sim (\mathbf{X}, Y)$, e nosso objetivo é construir uma função $g(\mathbf{x})$ que possa ser usada para fazer bem a predição de novas observações $(\mathbf{X}_{n+1}, Y_{n+1}), \dots, (\mathbf{X}_{n+m}, Y_{n+m})$, i.e., queremos que

$$g(\mathbf{x}_{n+1}) \approx y_{n+1}, \dots, g(\mathbf{x}_{n+m}) \approx y_{n+m}.$$

A diferença de um problema de classificação para um problema de regressão é que no primeiro a variável resposta Y não é uma variável quantitativa, mas sim uma variável qualitativa.

Um exemplo tradicional de classificação é o problema de prever se um paciente tem uma certa doença com base em variáveis clínicas \mathbf{x} . Outro exemplo é o da classificação automática de dígitos escritos à mão com base em suas imagens, veja Figura 7.1 para alguns exemplos.



Handwritten digits from 0 to 9, arranged vertically:

- 7 2 1 0 4 1 4 9 5 9
- 0 6 9 0 1 5 9 7 3 4
- 9 6 6 5 4 0 7 4 0 1
- 3 1 3 4 7 2 7 1 2 1
- 1 7 4 2 3 5 1 2 4 4

Figura 7.1: Exemplos de dígitos escritos à mão. O objetivo de um problema de classificação é criar uma função $g(\mathbf{x})$ que consiga automaticamente predizer que dígito corresponde a cada imagem com alta acurácia.

Neste capítulo, investigaremos como a avaliação da performance de uma dada função de predição $g(\mathbf{x})$ (i.e., um *classificador*) difere com relação ao estudo sobre regressão. No Capítulo 8, estudaremos então alguns métodos de criação de classificadores com bom poder preditivo.

7.1 Função de Risco

Vamos assumir que Y possui valores em um conjunto \mathcal{C} (e.g., $\mathcal{C} = \{\text{spam, não spam}\}$). Primeiramente, notamos que a função de risco $R(g) = \mathbb{E}[(Y - g(\mathbf{X}))^2]$ estudada para o caso de regressão (Y quantitativo) não faz sentido para classificação (o que é $Y - g(\mathbf{X})$?). Ao invés dela, é comum se utilizar

$$R(g) := \mathbb{E}[\mathbb{I}(Y \neq g(\mathbf{X}))] = \mathbb{P}(Y \neq g(\mathbf{X})),$$

ou seja, o risco de g é agora a probabilidade de erro em um nova observação (\mathbf{X}, Y) . Em outras palavras, uma função de perda mais adequada para o contexto de classificação é $L(g, (\mathbf{X}, Y)) = \mathbb{I}(Y \neq g(\mathbf{X}))$, a chamada função de perda 0-1.

Vimos que em regressão a função que minimiza $\mathbb{E}[(Y - g(\mathbf{X}))^2]$ é dada pela função de regressão $r(\mathbf{x}) = \mathbb{E}[Y | \mathbf{X} = \mathbf{x}]$ (Capítulo 2). Existe um análogo para classificação: a melhor função de classificação g segundo a função de risco $R(g)$ é dada por

$$g(\mathbf{x}) = \arg \max_{d \in \mathcal{C}} \mathbb{P}(Y = d | \mathbf{x}),$$

i.e., deve-se classificar \mathbf{x} como sendo daquela classe com maior probabilidade *a posteriori*. Tal classificador é conhecido como classificador de Bayes (não confundir com teorema de

Bayes) e, como no caso de regressão, ele é desconhecido, pois $\mathbb{P}(Y = d|\mathbf{x})$ é desconhecida. O Teorema a seguir formaliza esta caracterização.

Teorema 7.1 *Suponha que definimos o risco de uma função de predição $g : \mathbb{R}^d \rightarrow \mathbb{R}$ via perda 0-1: $R(g) = \mathbb{P}(Y \neq g(\mathbf{X}))$, em que (\mathbf{X}, Y) é uma nova observação que não foi usada para estimar g . Então a função g que minimiza $R(g)$ é dada por*

$$g(\mathbf{x}) = \arg \max_{d \in \mathcal{C}} \mathbb{P}(Y = d|\mathbf{x})$$

Demonstração. Para simplificar a demonstração do teorema, vamos assumir que Y assume só dois valores, digamos, c_1 e c_2 (i.e., temos um problema *binário*, e.g, spam/não spam). Temos que

$$\begin{aligned} R(g) &:= \mathbb{E}[\mathbb{I}(Y \neq g(\mathbf{X}))] = \mathbb{P}(Y \neq g(\mathbf{X})) = \int_{\mathbb{R}^d} \mathbb{P}(Y \neq g(\mathbf{X})|\mathbf{x}) f(\mathbf{x}) d\mathbf{x} \\ &= \int_{\mathbb{R}^d} [\mathbb{I}(g(\mathbf{x}) = c_2) \mathbb{P}(Y = c_1|\mathbf{x}) + \mathbb{I}(g(\mathbf{x}) = c_1) \mathbb{P}(Y = c_2|\mathbf{x})] f(\mathbf{x}) d\mathbf{x}. \end{aligned}$$

Assim, para um dado \mathbf{x} , devemos escolher $g(\mathbf{x}) = c_1$ quando

$$\mathbb{P}(Y = c_1|\mathbf{x}) \geq \mathbb{P}(Y = c_2|\mathbf{x}),$$

caso contrário devemos escolher $g(\mathbf{x}) = c_2$.

□

Note que, para o caso binário, o classificador de Bayes pode ser reescrito como

$$g(\mathbf{x}) = c_1 \iff \mathbb{P}(Y = c_1|\mathbf{x}) \geq \frac{1}{2}.$$

Observação 7.1 *É comum na literatura denotar os elementos de \mathcal{C} por 0, 1, 2, etc. Em particular, para o caso binário, é comum usar as classes 0 e 1. Note que tal escolha é válida, mas arbitrária (i.e., não se deve entender que há uma ordenação entre esses elementos).*

□

7.2 Estimação do Risco e Seleção de Modelos

Da mesma maneira que em regressão (Seção 1.4.1), pode-se estimar o risco de um método de classificação utilizando-se data splitting ou validação cruzada. Consideraremos a primeira dessas abordagens, lembrando que a divisão deve ser feita **aleatoriamente**:

$$\overbrace{(\mathbf{X}_1, Y_1), (\mathbf{X}_2, Y_2), \dots, (\mathbf{X}_s, Y_s)}^{\text{Treinamento (e.g., 70\%)}}, \overbrace{(\mathbf{X}_{s+1}, Y_{s+1}), \dots, (\mathbf{X}_n, Y_n)}^{\text{Validação (e.g., 30\%)}}$$

Como em regressão, usamos o conjunto de treinamento para estimar g , mas usamos o conjunto de validação *apenas para estimar $R(g)$* via

$$R(g) \approx \frac{1}{n-s} \sum_{i=s+1}^n \mathbb{I}(Y_i \neq g(X_i)) := \hat{R}(g), \quad (7.1)$$

isto é, avaliamos a proporção de erros no conjunto de validação.

Desta maneira, uma forma de selecionar um modelo g dentro de uma classe de modelos \mathbb{G} consiste em usar validação cruzada para estimar $R(g)$ para cada $g \in \mathbb{G}$ e, então, escolher g com o menor risco estimado. O teorema a seguir mostra que este procedimento de fato retorna com probabilidade alta o melhor modelo em \mathbb{G} .

Teorema 7.2 *Seja $\mathbb{G} = \{g_1, \dots, g_N\}$ uma classe de classificadores estimados com base em um conjunto de treinamento, e seja $\hat{R}(g)$ o erro estimado de g com base no conjunto de validação (Equação 7.1). Seja g^* o modelo que minimiza o risco real $R(g)$ dentre $g \in \mathbb{G}$ e seja \hat{g} o modelo que minimiza o risco estimado $\hat{R}(g)$ dentre $g \in \mathbb{G}$. Então, com probabilidade de no máximo ϵ ($\epsilon > 0$),*

$$|\hat{R}(\hat{g}) - R(g^*)| > 2\sqrt{\frac{1}{2(n-s)} \log \frac{2N}{\epsilon}}.$$

Demonastração. Pela desigualdade de Hoeffding, para todo $\delta > 0$ e todo $g \in \mathbb{G}$, tem-se que $\mathbb{P}(|\hat{R}(g) - R(g)| > \delta) \leq 2e^{-2(n-s)\delta^2}$. Assim,

$$\begin{aligned} \mathbb{P}(\max_{g \in \mathbb{G}} |\hat{R}(g) - R(g)| > \delta) &= \mathbb{P}\left(\bigcup_{g \in \mathbb{G}} |\hat{R}(g) - R(g)| > \delta\right) \\ &\leq \sum_{g \in \mathbb{G}} \mathbb{P}(|\hat{R}(g) - R(g)| > \delta) \leq N2e^{-2(n-s)\delta^2} := \epsilon, \end{aligned}$$

de modo que $\delta = \sqrt{\frac{1}{2(n-s)} \log \frac{2N}{\epsilon}}$. Assim, com probabilidade ao menos $1 - \epsilon$, $|\hat{R}(\hat{g}) - R(g)| \leq \delta$ para todo $g \in \mathbb{G}$ e, assim,

$$R(\hat{g}) \leq \hat{R}(\hat{g}) + \delta \leq \hat{R}(g^*) + \delta \leq R(g^*) + 2\delta.$$

□

7.3 Balanço entre Viés e Variância

Assim como em regressão (Seção 1.4.2), em classificação também se enfrenta o paradigma do balanço entre viés e variância. Contudo, os detalhes são um pouco diferentes. Suponha que \mathbb{G} seja uma classe de classificadores (e.g., todos os classificadores baseados em uma regressão logística; veja Seção 8.1.2), seja $R_{or} = \inf_{g \in \mathbb{G}} R(g)$ o melhor risco (o risco do oráculo) que pode ser obtido usando-se classificadores de \mathbb{G} e seja R_* o risco do classificador de Bayes (Teorema 7.1). Então, para todo $g \in \mathbb{G}$,

$$R(g) - R_* = T_1 + T_2,$$

em que $T_1 = R(g) - R_{or}$ é o análogo da variância (em geral é alto se \mathbb{G} possui muitos elementos – por exemplo, se há muitos parâmetros na regressão logística) e $T_2 = R_{or} - R_*$ é o análogo do viés ao quadrado (em geral é baixo se \mathbb{G} possui muitos elementos). Assim, \mathbb{G} não pode ser nem muito grande, nem muito pequena.

7.4 Outras medidas de performance

Nem sempre a função de risco $R(g) := \mathbb{E}[\mathbb{I}(Y \neq g(\mathbf{X}))] = \mathbb{P}(Y \neq g(\mathbf{X}))$ traz toda informação sobre o quanto razoável g é. Por exemplo, suponha que Y indica se uma pessoa tem uma certa doença rara, e que, portanto, em uma amostra i.i.d., há poucos pacientes com $Y = 1$. O classificador trivial $g(\mathbf{x}) \equiv 0$ terá risco baixo, pois $\mathbb{P}(Y \neq 0)$ é pequena, mas sua performance deixa a desejar.

Na prática, para evitar esse tipo de situação, é comum considerar matrizes de confusão; veja a Tabela 7.1.

Tabela 7.1: Exemplo de matriz de confusão.

		Valor verdadeiro	
Valor Predito			Y=1
	Y=0	VN	
Y=1	FP	FN	VP

Aqui, V indica verdadeiro, F é falso, P é positivo e N é negativo. Com base nessa tabela, pode-se definir várias medidas, como por exemplo:

- Sensibilidade: $S=VP/(VP+FN)$ (dos pacientes doentes, quantos foram corretamente identificados?)
- Especificidade: $E=VN/(VN+FP)$ (dos pacientes não doentes, quantos foram corretamente identificados?)
- Valor preditivo positivo: $VPP=VP/(VP+FP)$ (dos pacientes classificados como doentes, quantos foram corretamente identificados?)
- Valor preditivo negativo: $VPN=VN/(VN+FN)$ (dos pacientes classificados como não doentes, quantos foram corretamente identificados?)
- Estatística F: $F=2/(1/S+1/VPP)$
- Estatística G: $G=\sqrt{VPP \cdot VPN}$

Note que, para o classificador trivial $g(\mathbf{x}) \equiv 0$, temos sensibilidade zero e especificidade um. Assim, apesar da especificidade ser alta, a sensibilidade é muito baixa. Isso indica que o classificador pode na realidade ser ruim.

Assim, na prática é recomendável olhar para outras medidas além do risco estimado. Isso se torna particularmente importante na presença de dados desbalanceados (i.e., caso a frequência de uma classe é muito diferente das demais). Voltaremos para esse tópico na Seção 9.1.

Observação 7.2 É importante calcular os valores de VP, FN, VN e FP usando-se uma amostra de teste ou validação para evitar o *overfitting*.

□

Observação 7.3 A sensibilidade estima $\mathbb{P}(g(\mathbf{X}) = 1 | Y = 1)$, e a especificidade estima $\mathbb{P}(g(\mathbf{X}) = 0 | Y = 0)$.

□

Capítulo 8

Métodos de classificação

Neste capítulo apresentamos diversos métodos que visam fornecer classificadores com bom poder preditivo.

8.1 Classificadores Plug-in

O Teorema 7.1 sugere uma abordagem simples para resolver um problema de predição:

1. Estimamos $\mathbb{P}(Y = c|\mathbf{x})$ para cada categoria $c \in \mathcal{C}$.
2. Tomamos então

$$g(\mathbf{x}) = \arg \max_{c \in \mathcal{C}} \widehat{\mathbb{P}}(Y = c|\mathbf{x})$$

Esta abordagem é conhecida como classificador *plug-in*, pois pluga-se o estimador da probabilidade condicional na fórmula do g ótimo. Assim, sob esta abordagem, criar um classificador se resume a estimar $\mathbb{P}(Y = c|\mathbf{x})$. Nas seguintes seções veremos algumas formas de estimar tais probabilidades.

Observação 8.1 Na Seção 9.1 veremos que, muitas vezes, usar outros cortes além de $1/2$ pode levar a melhores resultados.

□

8.1.1 Métodos de regressão

Note que, para todo $c \in \mathcal{C}$,

$$\mathbb{P}(Y = c|\mathbf{x}) = \mathbb{E}[\mathbb{I}(Y = c)|\mathbf{x}].$$

Assim, pode-se usar qualquer modelo de regressão (Parte I deste livro) para estimar $\mathbb{P}(Y = c|\mathbf{x})$, basta estimar a função de regressão $\mathbb{E}[Z|\mathbf{x}]$, em que $Z = \mathbb{I}(Y = c)$. Por exemplo, pode-se estimar tal probabilidade via regressão linear, i.e., assumindo-se que

$$\mathbb{P}(Y = c|\mathbf{x}) = \mathbb{E}[Z|\mathbf{x}] = \beta_0^{(c)} + \beta_1^{(c)}x_1 + \dots + \beta_p^{(c)}x_p.$$

Pode-se, por exemplo, usar o método de mínimos quadrados para estimar tais coeficientes, ou então o *lasso* ou outras abordagens já discutidas. Ainda que as estimativas de $\mathbb{P}(Y = c|\mathbf{x})$ possam ser menores que zero, e maiores que um, essas podem ser usadas para definir o classificador

$$g(\mathbf{x}) = \arg \max_{c \in \mathcal{C}} \widehat{\mathbb{P}}(Y = c|\mathbf{x}).$$

Embora classificadores criados desta maneira frequentemente apresentam bons resultados, há pessoas que se sentem desconfortáveis em obter estimativas para uma probabilidade maiores que um ou menores que zero. Diversos métodos evitam isso; na sequência veremos alguns destes.

8.1.2 Regressão logística

Vamos assumir por ora que Y é binário, i.e., $|\mathcal{C}| = 2$. Denotando $\mathcal{C} = \{0, 1\}$, a regressão logística assume que

$$\mathbb{P}(Y = 1|\mathbf{x}) = \frac{e^{\beta_0 + \sum_{i=1}^d \beta_i x_i}}{1 + e^{\beta_0 + \sum_{i=1}^d \beta_i x_i}}.$$

Note que, como no caso de regressão, não estamos assumindo que esta relação é válida, mas apenas esperamos que ela nos leve a um bom classificador.

Para estimar os coeficientes de uma regressão logística, pode-se usar o método de máxima verossimilhança. Neste caso, a função de verossimilhança dada uma amostra i.i.d. $(\mathbf{X}_1, Y_1), \dots, (\mathbf{X}_n, Y_n)$, condicional nas covariáveis, é dada por

$$L(y; (\mathbf{x}, \boldsymbol{\beta})) = \prod_{k=1}^n (\mathbb{P}(Y_k = 1|\mathbf{x}_k, \boldsymbol{\beta}))^{y_k} (1 - \mathbb{P}(Y_k = 1|\mathbf{x}_k, \boldsymbol{\beta}))^{1-y_k} \\ \prod_{k=1}^n \left(\frac{e^{\beta_0 + \sum_{i=1}^d \beta_i x_i}}{1 + e^{\beta_0 + \sum_{i=1}^d \beta_i x_i}} \right)^{y_k} \left(\frac{1}{1 + e^{\beta_0 + \sum_{i=1}^d \beta_i x_i}} \right)^{1-y_k}$$

Para encontrarmos as estimativas dos coeficientes β , maximizamos $L(y; (\mathbf{x}, \boldsymbol{\beta}))$. Ao contrário do estimador de mínimos quadrados de uma regressão linear (Eq. 2.2), é necessário usar algoritmos numéricos para maximizar a verossimilhança induzida pela regressão

logística e, assim, chegar nas estimativas para os coeficientes β . No R, pode-se utilizar a função `glm`:

```
glm.fit = glm(formula, data = dados, family = binomial)
```

Assim como no caso da regressão linear, também é possível utilizar penalização para estimar tais coeficientes para reduzir a variância do estimador e, assim, se obter possivelmente melhor poder preditivo. Para mais detalhes veja [Hastie et al. \(2009a\)](#). Em particular, o pacote `glmnet` do R permite que tais modelos sejam facilmente ajustados.

Quando $|\mathcal{C}| > 2$, i.e., quando há várias categorias, podemos estimar, para cada $c \in \mathcal{C}$, $\mathbb{P}(Y = c|\mathbf{x})$ usando uma regressão logística diferente. Para tanto, basta estimar $\mathbb{P}(Z = 1|\mathbf{x})$, em que $Z = \mathbb{I}(Y = c)$. Assim, ajustamos $|\mathcal{C}|$ regressões logísticas, e então utilizamos o classificador

$$g(\mathbf{x}) = \arg \max_{c \in \mathcal{C}} \widehat{\mathbb{P}}(Y = c|\mathbf{x}).$$

Existem outros métodos que garantem que a soma das probabilidades estimadas seja um, veja por exemplo [Friedman et al. \(2010\)](#).

8.1.3 Bayes Ingênuo

Uma outra abordagem para estimar $\mathbb{P}(Y = c|\mathbf{x})$ consiste em usar o Teorema de Bayes. Assumindo que \mathbf{X} seja um vetor de covariáveis contínuas, temos que

$$\mathbb{P}(Y = c|\mathbf{x}) = \frac{f(\mathbf{x}|Y = c)\mathbb{P}(Y = c)}{\sum_{s \in \mathcal{C}} f(\mathbf{x}|Y = s)\mathbb{P}(Y = s)}$$

Assim, pode-se obter uma estimativa de $\mathbb{P}(Y = c|\mathbf{x})$ estimando-se as probabilidades marginais $\mathbb{P}(Y = s)$ e as densidades condicionais $f(\mathbf{x}|Y = s)$ para cada $s \in \mathcal{C}$.

O termo $\mathbb{P}(Y = s)$ pode ser facilmente estimado usando-se as proporções amostrais de cada classe. Contudo, para estimar $f(\mathbf{x}|Y = s)$, é necessário assumir algum modelo para as covariáveis. O método Bayes ingênuo (*naive Bayes* em inglês) assume que, para todo $s \in \mathcal{C}$, $f(\mathbf{x}|Y = s)$ pode ser fatorada como

$$f(\mathbf{x}|Y = s) = f((x_1, \dots, x_d)|Y = s) = \prod_{j=1}^d f(x_j|Y = s),$$

i.e., assume que as componentes de \mathbf{x} são independentes condicionalmente à classe Y . Apesar de tal suposição não ser razoável em muitos problemas, ela é muito conveniente, e frequentemente leva a bons classificadores.

Podemos então estimar cada $f(x_j|Y = s)$ assumindo, por exemplo, que

$$X_j|Y=s \sim N(\mu_{j,s}, \sigma_{j,s}^2), \quad j = 1, \dots, d.$$

Em outras palavras, assumimos que cada componente do vetor \mathbf{X} tem distribuição normal, com parâmetros que dependem da classe e da componente em questão. Os parâmetros deste modelo podem ser facilmente estimados usando-se o estimador de máxima verossimilhança:

$$\widehat{\mu}_{j,s} = \frac{1}{|\mathcal{C}_s|} \sum_{k \in \mathcal{C}_s} X_{j,k} \quad \widehat{\sigma}_{j,s}^2 = \frac{1}{|\mathcal{C}_s|} \sum_{k \in \mathcal{C}_s} (X_{j,k} - \widehat{\mu}_{j,s})^2$$

em que $\mathcal{C}_s = \{j : Y_j = s\}$ é o conjunto de todas observações de treinamento da classe s .

Assim, o estimador para a densidade condicional $f(\mathbf{x}|Y = c)$ é dado por

$$\widehat{f}(\mathbf{x}|Y = c) = \prod_{k=1}^d \widehat{f}(x_k|Y = c) = \prod_{k=1}^d \frac{1}{\sqrt{2\pi\widehat{\sigma}_{k,c}^2}} e^{-\left(\frac{(x_k - \widehat{\mu}_{k,c})^2}{2\widehat{\sigma}_{k,c}^2}\right)}$$

Evidentemente, outras distribuições podem ser usadas além da distribuição normal, e outros métodos de estimação podem ser usados além do método de máxima verossimilhança. Pode-se inclusive utilizar métodos não-paramétricos para estimar cada uma das densidades condicionais.

Se \mathbf{X} tem componentes discretas, o Teorema de Bayes afirma que

$$\mathbb{P}(Y = c|x) = \frac{\mathbb{P}(\mathbf{X} = \mathbf{x}|Y = c)\mathbb{P}(Y = c)}{\sum_{s \in \mathcal{C}} \mathbb{P}(\mathbf{X} = \mathbf{x}|Y = s)\mathbb{P}(Y = s)}$$

Neste caso, pode-se por exemplo assumir que

$$X_j|Y=c \sim \text{Multinomial}(1, \theta_{j,c}),$$

em que $\theta_{j,c} \in \mathbb{R}^q$ é um vetor com q dimensões, o número de categorias que X_j assume. Novamente, os parâmetros de tal distribuição podem ser estimados via o método da máxima verossimilhança.

Observação 8.2 Na implementação do Bayes ingênuo para dados contínuos, calcular produtos como $\prod_{k=1}^d \widehat{f}(x_k|Y = c)$ é numericamente desafiador, pois cada termo é, em geral, muito próximo de zero e, assim, o produto é aproximado por 0 no computador. Uma forma de solucionar esse problema é trabalhar com logaritmos. Para isso, note que o classificador plugin com probabilidades estimadas pelo método Bayes ingênuo pode ser escrito como

$$\begin{aligned}
g(\mathbf{x}) &= \arg \max_{c \in \mathcal{C}} \widehat{\mathbb{P}}(Y = c | \mathbf{x}) = \arg \max_{c \in \mathcal{C}} \widehat{f}(\mathbf{x} | Y = c) \widehat{\mathbb{P}}(Y = c) \\
&= \arg \max_{c \in \mathcal{C}} \left(\prod_{k=1}^d \widehat{f}(x_k | Y = c) \right) \widehat{\mathbb{P}}(Y = c) \\
&= \arg \max_{c \in \mathcal{C}} \sum_{k=1}^d \log \left(\widehat{f}(x_k | Y = c) \right) + \log \widehat{\mathbb{P}}(Y = c),
\end{aligned}$$

uma vez que o logaritmo é uma função monotônica crescente. Assim, não é necessário calcular $\widehat{f}(\mathbf{x} | Y = c)$ diretamente.

□

Para dados discretos, pode-se ajustar o método Bayes ingênuo utilizando-se a função `naiveBayes` no pacote `e1071` do R.

Exemplo 8.1 (Detecção de SPAMs) Neste exemplo, consideramos o conjunto de dados Spambase, disponível do repositório do UCI¹. Este conjunto contém informações relativas a $n = 4.601$ emails. Nestes emails foram medidas a frequência relativa (i.e., qual a proporção de vezes que essa palavra aparece em cada email) de 57 palavras, como `internet`, `free`, `credit`, `money`, `data`, `technology` e `direct`, entre outras. Também foi verificado se cada email era ou não era SPAM (variável resposta). A Tabela 8.1 apresenta o erro preditivo estimado em um conjunto de teste de tamanho 600 para três classificadores.

Tabela 8.1: Riscos estimados e erros-padrão de alguns estimadores para o exemplo 8.1.

Método	Ressonâo linear	Ressonâo logística	Naive Bayes
Risco Estimado	0.111 (0.02)	0.086 (0.02)	0.188 (0.02)

□

8.1.4 Análise Discriminante

Enquanto que o método de Bayes ingênuo assume que a distribuição condicional das covariáveis pode ser fatorada como $f(\mathbf{x} | Y = c) = f(x_1, \dots, x_d | Y = c) = \prod_{j=1}^d f(x_j | Y = c)$, outras suposições podem ser feitas de modo a estimar tal quantidade. Na análise discriminante, supõe-se que o vetor \mathbf{X} , condicional em $Y = c$, possui distribuição normal

¹ <https://archive.ics.uci.edu/ml/datasets/Spambase>.

multivariada. Existem duas formas de análise discriminante mais comuns. Estas serão estudadas nas seções que seguem.

8.1.4.1 Análise Discriminante Linear

Na análise discriminante linear, assume-se que cada distribuição condicional $\mathbf{X}|Y = c$ segue uma distribuição normal multivariada. Essas podem ter médias diferentes, contudo todas têm a mesma matriz de variância-covariância. Assim, assume-se que

$$\mathbf{X} = (X_1, \dots, X_d)|Y = c \sim \text{Normal}(\mu_c, \Sigma),$$

i.e.,

$$f(\mathbf{x}|Y = c) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} e^{-(\mathbf{x} - \mu_c)' \Sigma^{-1} (\mathbf{x} - \mu_c)}.$$

Pode-se estimar os parâmetros desta distribuição utilizando-se o método da máxima verossimilhança. Com isso, obtém-se

$$\widehat{\mu}_c = \frac{1}{|\mathcal{C}_c|} \sum_{k \in \mathcal{C}_c} \mathbf{X}_k; \quad \widehat{\Sigma} = \frac{1}{n} \sum_{c \in \mathcal{C}} \sum_{k \in \mathcal{C}_c} (\mathbf{x}_k - \widehat{\mu}_c)(\mathbf{x}_k - \widehat{\mu}_c)'$$

em que $\mathcal{C}_c = \{j = 1, \dots, n : Y_j = c\}$

Como no método Bayes ingênuo, pode-se então usar as estimativas de $\widehat{f}(\mathbf{x}|Y = c)$ para construir o classificador plugin

$$g(\mathbf{x}) = \arg \max_{c \in \mathcal{C}} \widehat{\mathbb{P}}(Y = c|\mathbf{x}) = \arg \max_{c \in \mathcal{C}} \widehat{f}(\mathbf{x}|Y = c) \widehat{\mathbb{P}}(Y = c).$$

Para o caso binário, tem-se que a regra de classificação é $g(\mathbf{x}) = 1$ se, e só se,

$$\begin{aligned} \frac{\widehat{\mathbb{P}}(Y = 1|\mathbf{x})}{\widehat{\mathbb{P}}(Y = 0|\mathbf{x})} \geq K &\iff \frac{\widehat{f}(\mathbf{x}|Y = 1)\widehat{\mathbb{P}}(Y = 1)}{\widehat{f}(\mathbf{x}|Y = 0)\widehat{\mathbb{P}}(Y = 0)} \geq K \iff \\ \log \widehat{f}(\mathbf{x}|Y = 1) - \log \widehat{f}(\mathbf{x}|Y = 0) &\geq \log K + \log \widehat{\mathbb{P}}(Y = 0) - \log \widehat{\mathbb{P}}(Y = 1) \\ \iff -(\mathbf{x} - \widehat{\mu}_1)' \widehat{\Sigma}^{-1} (\mathbf{x} - \widehat{\mu}_1) + (\mathbf{x} - \widehat{\mu}_0)' \widehat{\Sigma}^{-1} (\mathbf{x} - \widehat{\mu}_0) &\geq K' \\ \iff +2\mathbf{x}' \widehat{\Sigma}^{-1} \widehat{\mu}_1 - \widehat{\mu}_1' \widehat{\Sigma}^{-1} \widehat{\mu}_1 - 2\mathbf{x}' \widehat{\Sigma}^{-1} \widehat{\mu}_0 + \widehat{\mu}_0' \widehat{\Sigma}^{-1} \widehat{\mu}_0 &\geq K' \\ \iff a\mathbf{x}' &\geq K'', \end{aligned}$$

em que K , K' e K'' são constantes².

Assim, a região do espaço amostral em que a regra de decisão consiste em $g(\mathbf{x}) = 1$ é um hiperplano de \mathbb{R}^d , por isso o método recebe o nome de *análise discriminante linear*. Veja uma ilustração desta região no Exemplo 8.2.

No R, a análise discriminante linear pode ser ajustada via o pacote “MASS”:

```
library(MASS)
lda.fit = lda(x=xTreino, grouping=yTreino)
lda.pred = predict(lda.fit, newdata=xNovo)
lda.pred$posterior # contém as estimativas de  $P(Y=c|x)$ 
```

Note que, assim como nos outros métodos preditivos vistos, não acreditamos necessariamente na suposição de normalidade, mas ela é apenas usada para obter um classificador com poder preditivo potencialmente bom.

8.1.4.2 Análise Discriminante Quadrática

A suposição feita pela análise discriminante quadrática também é de normalidade multivariada. Contudo, cada distribuição condicional pode ter sua própria matriz de variância-covariância. Assim, assume-se que

$$\mathbf{X} = (X_1, \dots, X_d) | Y = c \sim \text{Normal}(\mu_c, \Sigma_c),$$

i.e.,

$$f(\mathbf{x}|Y = c) = \frac{1}{\sqrt{(2\pi)^d |\Sigma_c|}} e^{-(\mathbf{x}-\mu_c)' \Sigma_c^{-1} (\mathbf{x}-\mu_c)}$$

Novamente, pode-se estimar esses parâmetros pelo método da máxima verossimilhança:

$$\widehat{\mu}_c = \frac{1}{|\mathcal{C}_c|} \sum_{k \in \mathcal{C}_c} \mathbf{X}_k; \quad \widehat{\Sigma}_c = \frac{1}{|\mathcal{C}_c|} \sum_{k \in \mathcal{C}_c} (\mathbf{x}_k - \widehat{\mu}_c)(\mathbf{x}_k - \widehat{\mu}_c)'$$

em que $\mathcal{C}_c = \{j = 1, \dots, n : Y_j = c\}$

Como na análise discriminante linear, pode-se então usar as estimativas de $\widehat{f}(\mathbf{x}|Y = c)$ para construir o classificador plugin

$$g(\mathbf{x}) = \arg \max_{c \in \mathcal{C}} \widehat{\mathbb{P}}(Y = c | \mathbf{x}) = \arg \max_{c \in \mathcal{C}} \widehat{f}(\mathbf{x}|Y = c) \widehat{\mathbb{P}}(Y = c).$$

Para o caso binário, tem-se que a regra de classificação é $g(\mathbf{x}) = 1$ se, e só se,

² Até aqui estamos tomando $K = 1$, mas na Seção 9.1 veremos que tal corte pode ser diferente.

$$\begin{aligned} \frac{\widehat{\mathbb{P}}(Y=1|x)}{\widehat{\mathbb{P}}(Y=0|x)} \geq K &\iff \frac{\widehat{f}(\mathbf{x}|Y=1)\widehat{\mathbb{P}}(Y=1)}{\widehat{f}(\mathbf{x}|Y=0)\widehat{\mathbb{P}}(Y=0)} \geq K \iff \\ \log \widehat{f}(\mathbf{x}|Y=1) - \log \widehat{f}(\mathbf{x}|Y=0) &\geq \log K + \log \widehat{\mathbb{P}}(Y=0) - \log \widehat{\mathbb{P}}(Y=1) \\ \iff -(\mathbf{x} - \widehat{\boldsymbol{\mu}}_1)' \widehat{\boldsymbol{\Sigma}}_1^{-1} (\mathbf{x} - \widehat{\boldsymbol{\mu}}_1) + (\mathbf{x} - \widehat{\boldsymbol{\mu}}_0)' \widehat{\boldsymbol{\Sigma}}_0^{-1} (\mathbf{x} - \widehat{\boldsymbol{\mu}}_0) &\geq K' \end{aligned}$$

Assim, a região do espaço amostral \mathbb{R}^d em que a regra de decisão consiste em $g(\mathbf{x}) = 1$ é dada por uma equação quadrática, por isso o método recebe o nome de análise discriminante quadrática. Veja uma ilustração desta região no Exemplo 8.2.

No R, a análise discriminante quadrática pode ser ajustada via o pacote “MASS”:

```
library(MASS)
qda.fit = qda(x=xTreino, grouping=yTreino)
qda.pred = predict(qda.fit, newdata=xNovo)
qda.pred$posterior # contém as estimativas de  $P(Y=c|x)$ 
```

Exemplo 8.2 Neste exemplo simulado, mostramos as formas da região de decisão para a análise discriminante linear e quadrática. Veja a Figura 8.1.

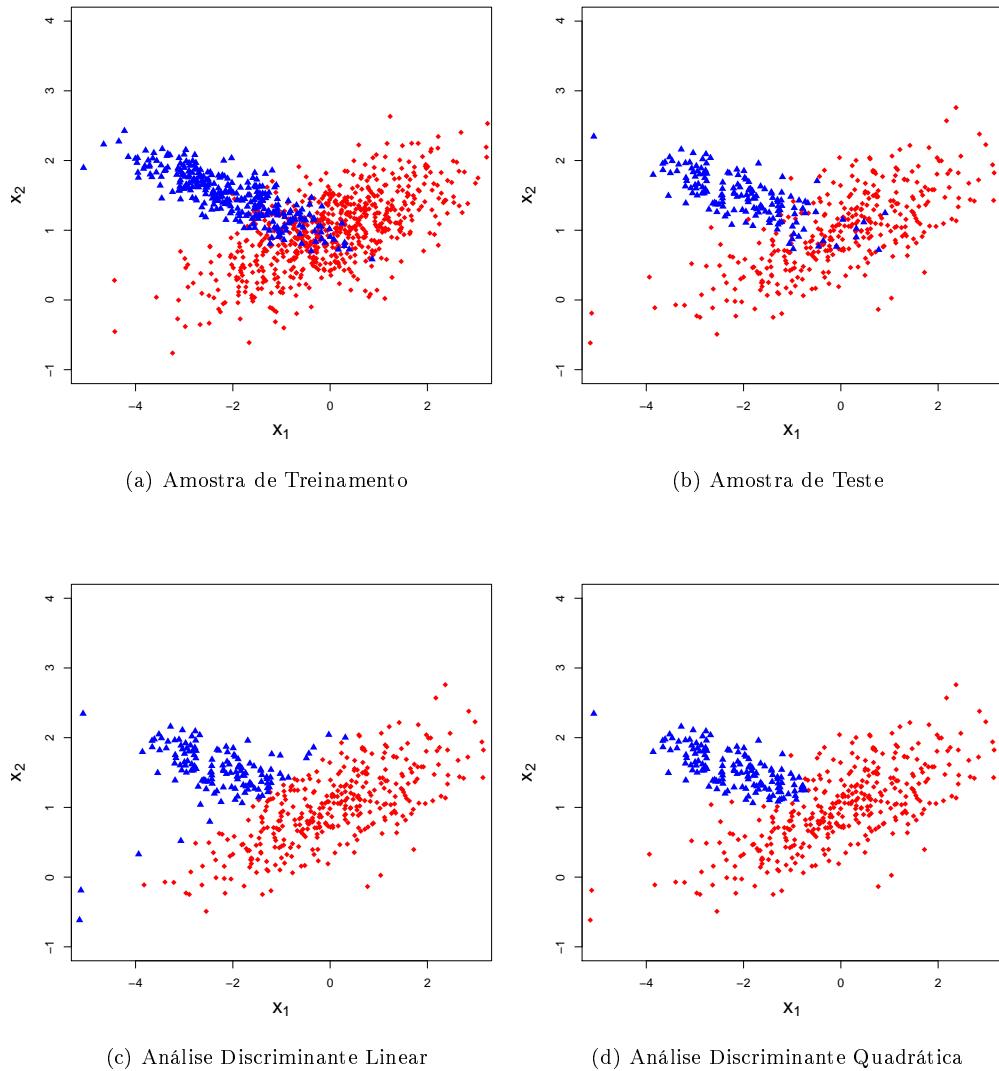


Figura 8.1: Acima: conjunto de treinamento e teste. Abaixo: ajuste das análises discriminantes linear (esquerda) e quadrática (direita).

□

8.2 Support Vector Machines (SVM)

Support Vector Machines é uma metodologia de classificação proposta por [Cortes and Vapnik \(1995\)](#) que leva a resultados excelentes em muitas aplicações. Esta técnica conta com uma motivação muito diferente daquela associada aos classificadores vistos até aqui. No SVM, em nenhum momento estimamos as probabilidades $\mathbb{P}(Y = c|\mathbf{x})$; o resultado desta técnica é apenas as classes estimadas de novas observações. Vamos assumir nesta seção que Y assume valores em $\mathcal{C} = \{-1, 1\}$.

Considere uma função linear

$$f(\mathbf{x}) := \beta_0 + \beta_1 x_1 + \dots + \beta_d x_d.$$

O classificador $g(\mathbf{x})$ dado pelo SVM tem a seguinte forma:

$$\begin{cases} \text{Se } f(\mathbf{x}) < 0, \quad g(\mathbf{x}) = -1 \\ \text{Se } f(\mathbf{x}) \geq 0, \quad g(\mathbf{x}) = 1 \end{cases} \quad (8.1)$$

Para descrever como construir $f(\mathbf{x})$, suponha inicialmente que existe um hiperplano que separa perfeitamente todas as observações do conjunto de treinamento segundo a classe que pertencem (veja um exemplo na Figura 8.2), ou seja, existe $f(\mathbf{x})$ linear tal que $f(\mathbf{x}_i) < 0$ se, e só se, $y_i = -1$. Note que temos então que, para todo $i = 1, \dots, n$,

$$y_i(\beta_0 + \beta_1 x_{i,1} + \dots + \beta_d x_{i,d}) = y_i f(\mathbf{x}_i) > 0. \quad (8.2)$$

Quando existem muitos hiperplanos que separam os dados (ou seja, quando há várias funções f tais que a Eq. 8.2 está satisfeita), o SVM busca por aquele que tem maiores margens, i.e., aquele que fica "mais distante" de todos os pontos observados. Para descrever matematicamente tal tarefa, lembre-se que $|f(\mathbf{x})|$ é (a menos de um fator multiplicativo) a distância do ponto \mathbf{x} à reta dada por $f(\mathbf{x})$. Assim, se $|f(\mathbf{x})|$ é muito alto, \mathbf{x} está muito longe do plano separador. Em particular, se f separa os dados perfeitamente bem, $|f(\mathbf{x})| = y_i f(\mathbf{x}_i)$ (devido à Eq. 8.2). Logo, o SVM busca f tal que $y_i f(\mathbf{x}_i)$ seja grande para todo \mathbf{x}_i . Assim, no caso de haver um hiperplano que separa perfeitamente bem os dados, o SVM busca, portanto, o hiperplano com coeficientes $\boldsymbol{\beta}$ tais que

$$(\boldsymbol{\beta}, M) = \arg \max_{\boldsymbol{\beta}, M} M$$

sujeito às restrições

- (1) $\sum_{i=1}^d \beta_i^2 = 1$ e
- (2) para todo $i = 1, \dots, n$, $y_i f_{\boldsymbol{\beta}}(\mathbf{x}_i) \geq M$.

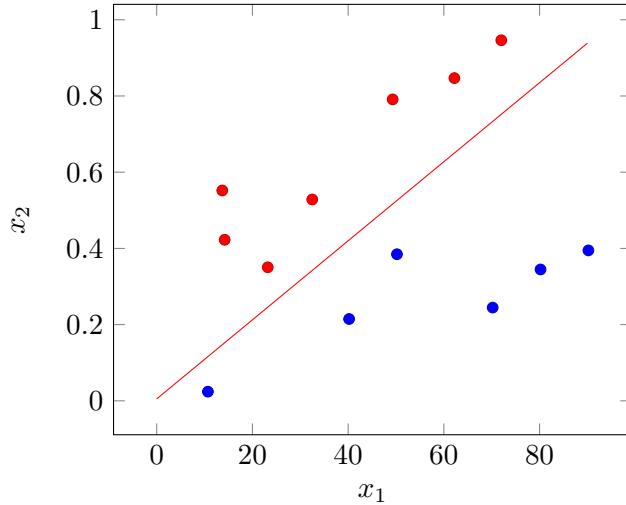


Figura 8.2: Exemplo de conjunto de dados no qual existe um hiperplano que os separa perfeitamente bem.

Note que a restrição (1) visa apenas garantir a comparabilidade dos diferentes hiperplanos.

Além de assumir que existe um hiperplano que separa perfeitamente bem os dados, esta formulação infelizmente também é muito sensível a pequenas mudanças nos dados. Assim, a formulação usada pelo SVM na realidade se propõe a resolver uma generalização do programa descrito acima, de modo que seja permitido que alguns dos pontos estejam do lado “errado” das margens (e eventualmente do hiperplano). Matematicamente, o SVM busca pela solução de

$$\arg \max_{\beta, M} M$$

sujeito às restrições

- (1) $\sum_{i=1}^d \beta_i^2 = 1$ e
- (2) para todo $i = 1, \dots, n$, $y_i f_\beta(\mathbf{x}_i) \geq M(1 - \epsilon_i)$, em que $\epsilon_i > 0$ e $\sum_{i=1}^n \epsilon_i \leq C$.

Note que ϵ_i pode ser maior que um, de modo que é permitido que $y_i f_\beta(\mathbf{x}_i)$ seja negativo, ou seja, que a i -ésima amostra caia do lado errado do hiperplano. O quanto longe ela cai, contudo, é limitado por C , uma vez que ϵ_i não pode ser maior que C . Assim, C é um tuning parameter: quanto maior é seu valor, mais se permite que observações caiam do lado “errado” das margens.

Utilizando-se o truque do kernel (Seção 4.6.3.2), pode-se buscar por divisões mais complexas que hiperplanos. A ideia central é que a solução ótima $f(\mathbf{x})$ do SVM pode ser

reescrita como

$$f(\mathbf{x}) = \beta_0 + \beta_1 x_1 + \dots + \beta_d x_d = \beta_0 + \sum_{k=1}^n \alpha_k \langle \mathbf{x}, \mathbf{x}_k \rangle, \quad (8.3)$$

em que $\langle \mathbf{x}, \mathbf{x}_k \rangle = \sum_{i=1}^d x_i x_{k,i}$. Assim, para calcular f (i.e., os coeficientes α_k), tudo o que precisamos é do produto interno entre todas as observações. Pode-se também mostrar que, para calcular α_k , também necessita-se apenas dos produtos internos entre as observações. Podemos, portanto, trocar $\langle \mathbf{x}, \mathbf{x}_k \rangle$ por um kernel genérico $K(\mathbf{x}, \mathbf{x}_k)$ e, assim, utilizar o truque do kernel.

Support Vector Machines também estão ligados a Reproducing Kernel Hilbert Spaces (Seção 4.6). Pode-se mostrar que, dado um kernel de Mercer K , o classificador g dado pelo SVM é justamente aquele que minimiza

$$\arg \min_{g \in \mathcal{H}_K} \sum_{k=1}^n L(y_k, g(\mathbf{x}_k)) + \lambda \|g\|_{\mathcal{H}_K}^2,$$

em que $L(y_k, g(\mathbf{x}_k)) = (1 - y_k g(\mathbf{x}_k))_+$ (Pontil, 2003). Assim, pode-se utilizar o Teorema da Representação (Teorema 4.3) para encontrar os valores dos coeficientes α_k da Equação 8.3. Note que a perda L é zero quando $y g(\mathbf{x}) > 1$. No caso linear, essa restrição indica que (\mathbf{x}, y) está do lado correto do hiperplano definido por g e \mathbf{x} está a uma distância de ao menos $1/\|g\|_K$ do hiperplano. Ou seja, a perda é zero quando o exemplo é classificado corretamente de forma fácil. Note também que, no caso separável, essa função objetivo indica (como esperado) que o SVM procura, entre todos os hiperplanos separados, aquele com maior margem (i.e., $1/\|g\|_K$).

No R, o ajuste do SVM pode ser feito via o pacote e1071.

```
library(e1071)

# Kernel Linear
tune.out=tune(svm,train.x=xTrein,
              train.y=yTrein,
              kernel="linear",scale=TRUE,
              ranges=list(cost=c(0.001,0.01,0.1,1,5,10,100)))
plot(tune.out)
melhorModelo = tune.out$best.model
ypred=predict(melhorModelo,xTeste)

# Kernel Gaussiano
```

```
tune.out=tune(svm,train.x=xTrein,
              train.y=yTrein,
              kernel="radial",scale=TRUE,
              ranges=list(cost=c(5,10,20),
                          gamma = c(0.001,0.01,0.05)))
plot(tune.out)
melhorModelo = tune.out$best.model
ypred=predict(melhorModelo,xTest)
```

8.3 Árvores de Classificação

Árvores de classificação são o análogo de árvores de regressão (Seção 4.8), mas com a finalidade de se fazer classificação. Sua construção se dá de forma análoga à descrita naquela seção. Em particular, o processo de criar uma árvore grande e depois podá-la também ocorre. Há, contudo, duas grandes diferenças. Primeiramente, a predição para a resposta Y de uma observação com covariáveis \mathbf{x} que estão em uma região R_k não é mais dada pela média amostral das observações do conjunto de treinamento que pertencem à essa região (Equação 4.18), mas sim pela moda destas:

$$g(\mathbf{x}) = \text{moda}\{y_i : \mathbf{x}_i \in R_k\}. \quad (8.4)$$

Além disso, o critério utilizado para buscar a melhor partição em cada etapa do processo (I) também é diferente, uma vez que o erro quadrático (e.g, Equação 4.19) já não faz mais sentido. No lugar deste, um critério muito utilizado é buscar minimizar o índice de Gini, dado por

$$\sum_R \sum_{c \in \mathcal{C}} \hat{p}_{R,c} (1 - \hat{p}_{R,c})$$

Aqui, R representa uma das regiões induzidas pela árvore, e $\hat{p}_{R,c}$ é a proporção de observações classificadas como sendo da categoria c entre as que caem na região R . Note que este índice é mínimo quando todas as proporções $\hat{p}_{R,c}$ são zero ou um, indicando assim uma árvore “pura” (i.e., cada folha contém somente observações de uma única classe).

Exemplo 8.3 *Para ilustrar uma aplicação do índice de Gini, considere a situação em que se deseja predizer se um hospital possui tomógrafo. Para isso contamos com informações sobre a presença de especialidade em cardiologia e se o hospital apresenta mais de 100 leitos. Os dados são apresentados na Tabela 8.2.*

Tabela 8.2: Dados fictícios do Exemplo 8.3.

Cardiologia	Acima de 100 leitos	Possui tomógrafo
Sim	Sim	Sim
Sim	Não	Sim
Sim	Sim	Não
Sim	Não	Não
Não	Sim	Sim
Não	Sim	Sim
Não	Não	Não
Não	Não	Não
Não	Sim	Sim
Não	Sim	Sim

Assim, para determinar qual será a melhor partição (cardiologia ou acima de 100 leitos), utilizaremos o índice de Gini. Lembre-se que queremos determinar a partição que apresenta a maior "pureza", ou seja, o menor valor do índice. Dessa forma, teremos:

- $Gini(\text{Cardiologia}) = \frac{2}{4} \times \frac{2}{4} + \frac{4}{6} \times \frac{2}{6} = 0.472$
- $Gini(\text{Leitos}) = \frac{5}{6} \times \frac{1}{6} + \frac{1}{4} \times \frac{3}{4} = 0.326$

Logo, a partição que apresenta a maior "pureza" é dada pela partição da variável leitos.

□

Observação 8.3 À primeira vista, pode-se estranhar o fato do índice de Gini ser utilizado ao invés de, por exemplo, a proporção de erros no conjunto de validação, que é uma estimativa do risco. O motivo desta escolha é que o índice de Gini é mais sensível a mudanças nas proporções de cada categoria nos nós.

□

Para a etapa da poda, em geral utiliza-se a proporção de erros no conjunto de validação como estimativa do risco.

Assim como em árvores de regressão, podemos fazer árvores de classificação no R utilizando o pacote `rpart`.

```
set.seed(0)
library(rpart)
```

```
# Ajustar a árvore:  
fit <- rpart(Kyphosis ~ Age + Number + Start,  
               method="class", data=kyphosis)  
  
# poda:  
melhorCp=fit$cptable[which.min(fit$cptable[, "xerror"]),"CP"]  
# cp é uma medida de complexidade da árvore, essencialmente  
# proporcional ao número de folhas presentes. Este código  
# escolhe o melhor cp via validação cruzada.  
pfit <- prune(fit,  
               cp=melhorCp)  
  
# plotar árvore podada  
plot(pfit)  
text(pfit, use.n=FALSE, all=FALSE, cex=1.5)
```

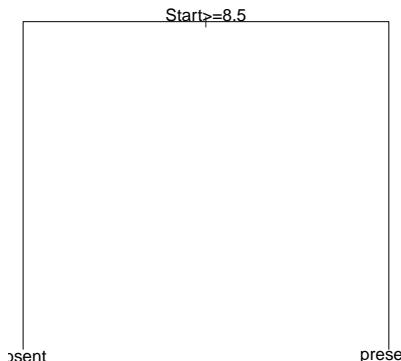


Figura 8.3: Árvore de classificação já podada.

A Figura 8.3 ilustra a árvore gerada pelo código acima no R.

8.4 Bagging e Florestas Aleatórias

Tanto o bagging quanto florestas aleatórias, vistos no contexto de regressão na Seção 4.9, podem ser trivialmente adaptados para classificação. Para tanto, a agregação das diferentes árvores de classificação é feita através da função

$$g(\mathbf{x}) = \text{moda}_b g^b(\mathbf{x}),$$

isto é, uma observação com covariáveis \mathbf{x} é classificada por cada árvore construída, e a predição é então dada pela categoria predita mais frequente.

8.5 Boosting

Assim como em regressão (Seção 4.10), métodos de boosting também podem ser utilizados para agrregar classificadores *fracos* de forma a construir um classificador mais poderoso. Aqui descreveremos uma forma de boosting específica, o *adaboost* (Freund and Schapire, 1995). Para tanto, assuma que, como no SVM, $y_i \in \{-1, 1\}$.

1. Inicialize os pesos $w_1 = \dots = w_n = \frac{1}{n}$
2. Para $b = 1, \dots, B$:
 - a. Ajuste um classificador $g_b(\mathbf{x})$ para a amostra de treinamento usando os pesos w_1, \dots, w_n
 - b. Calcule o erro $er_b = \frac{\sum_{i=1}^n w_i \mathbb{I}(y_i \neq g_b(\mathbf{x}_i))}{\sum_{i=1}^n w_i}$
 - c. Calcule $\alpha_b = \log((1 - er_b)/er_b)$
 - d. Atualize $w_i \leftarrow w_i \exp(\alpha_b \mathbb{I}(y_i \neq g_b(\mathbf{x}_i))), i = 1, \dots, n$
3. Retornamos o modelo final $g(\mathbf{x}) = \text{sinal}\left(\sum_{b=1}^B \alpha_b g_b(\mathbf{x})\right)$

Em geral, o passo 2a) deste método é executado utilizando-se um classificador bastante simples, como por exemplo uma árvore com 4 folhas.

Note que a atualização dos pesos dada por $w_i \leftarrow w_i \exp(\alpha_b \mathbb{I}(y_i \neq g_b(\mathbf{x}_i)))$ tende a fazer com que observações classificadas erroneamente por g_b recebam peso maior no ajuste de g_{b+1} . Além disso, α_b mede o quanto bom o classificador g_b é: quanto menor o valor de er_b , maior o valor de α_b . Em particular, se $er_b > 50\%$, então $\alpha_b < 0$, ou seja, as predições de g_b são invertidas no classificador final. Tal inversão se dá pois o classificador $-g_b$ tem erro menor que 50% nesta situação.

add: conexão com Exponential Loss

Para se ajustar o adaboost no R, pode-se utilizar o pacote `gbm`.

```

library(gbm)

resposta=CO2[, "uptake"]>20
ajuste = gbm.fit(x=CO2[, ! colnames(CO2) %in% c("uptake")],
                  y=resposta,
                  n.trees=100,
                  interaction.depth = 1,
                  distribution="adaboost")
predito = predict(ajuste, newdata = xTest, n.trees=ajuste$n.trees)

```

8.6 Método dos k Vizinhos Mais Próximos

O método do KNN (Seção 4.3) pode ser trivialmente adaptado para o contexto de classificação. Para tanto, pode-se definir o classificador

$$g(\mathbf{x}) = \text{moda}_{i \in \mathcal{N}_{\mathbf{x}}} y_i,$$

em que $\mathcal{N}_{\mathbf{x}}$ é o conjunto das k observações mais próximas de \mathbf{x} , como definido na Seção 4.3. Ou seja, busca-se a classe mais frequentemente observada entre as observações mais próximas ao vetor de covariáveis \mathbf{x} de interesse.

No R, tal método pode ser utilizado via

```

library(FNN)
ajuste = knn(train=xTreinamento, test=xNovo, cl=yTreinamento,
              k=k)
predVal=ajuste$pred

```

8.7 Redes Neurais Artificiais

Redes Neurais Artificiais (Seção 4.11) também podem ser utilizadas no contexto de classificação. A estrutura da rede é essencialmente a mesma, contudo em geral há algumas diferenças:

- Ao invés de buscar apenas uma função de predição g_{β} , estima-se $|\mathcal{C}|$ funções, $g_{\beta,1}, \dots, g_{\beta,|\mathcal{C}|}$, uma para cada categoria que a variável resposta pode assumir, veja

a Figura 8.4. A i -ésima dessas funções representa a probabilidade de Y assumir a categoria i , $i = 1, \dots, |\mathcal{C}|$.

- Utiliza-se uma função de ativação f diferente da linear na camada de saída. Pode-se usar, por exemplo, a função logística $f(y) = e^y/(1 + e^y)$.
- A função objetivo a ser minimizada também não é mais o EQM. Pode-se utilizar, por exemplo, a entropia cruzada, dada por

$$CE(g_{\beta,1}, \dots, g_{\beta,|\mathcal{C}|}) = -\frac{1}{n} \sum_{k=1}^n \sum_{c \in \mathcal{C}} \mathbb{I}(y_k = c) \log(g_{\beta;c}(\mathbf{x}_k))$$

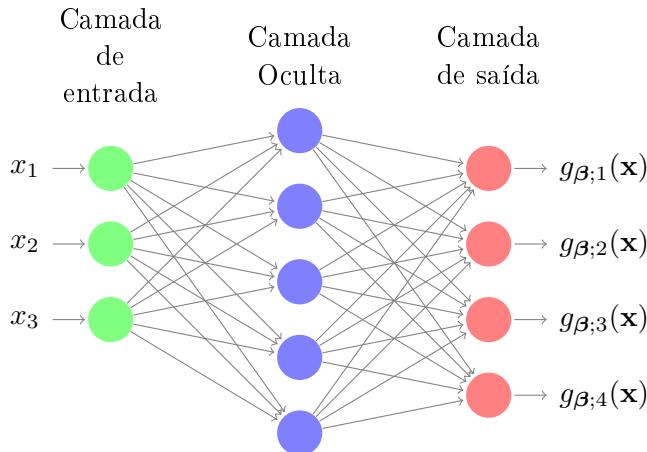


Figura 8.4: Exemplo de rede neural de classificação com uma camada oculta no caso em que Y assume quatro possíveis valores.

No R, redes neurais para classificação podem ser usadas via

```
library(neuralnet)
fit=neuralnet(as.formula("y1+y2~X1+X2+X3"), data=dados,
              hidden=c(2,3), err.fct = "ce")
```

Para se obter a matriz de variáveis *dummy* y_1, y_2, \dots a partir de y pode-se utilizar o comando `y=model.matrix(~dados$y-1)`.

8.8 Exemplos

Exemplo 8.4 (Amazon Fine Food Reviews) A seguir iremos trabalhar novamente com o conjunto de dados Amazon Fine Food Reviews (apresentado no Exemplo 3.3). No entanto, nesse momento, trabalharemos com as notas categorizadas em máxima (escore igual a 5) vs demais (escore de 0 a 4).

Nesse exemplo utilizaremos 40.000 observações de treinamento e 10.000 de validação. Iremos utilizar regressão logística com mínimos quadrados, ridge e lasso. Também utilizaremos árvore e floresta para a classificação.

```
library(dplyr)
library(glmnet)
library(data.table)
library(tm)
library(rpart)
library(rpart.plot)
library(xgboost)
library(ggplot2)
library(randomForest)
```

A seguir indicamos uma forma de leitura dos dados e organização para análise.

```
dados <- fread("data/Reviews.csv", header = TRUE)

set.seed(1)
# seleciona 50.000 observações
selecao <- sample(nrow(dados), 50000)
dados <- dados[selecao,]

# indica observações de treinamento
tr <- sample.int(50000, 40000, replace = F)

# categoriza escore
dados$Score <- ifelse(dados$Score <= 4, 0, 1)

corp <- VCorpus(VectorSource(dados$Text))

dtm <- DocumentTermMatrix(corp,
```

```
control = list(tolower = TRUE,  
stemming = FALSE,  
removeNumbers = TRUE,  
removePunctuation = TRUE,  
removeStripwhitespace = TRUE,  
weighting = weightTf,  
bounds=list(global=c(50, Inf))))
```

Note que nesse caso estamos trabalhando com um total de 3.683 palavras.

```
dim(dtmMatrix)
```

Após a leitura dos dados, iremos ajustar os modelos de regressão logística com mínimos quadrados, ridge e lasso. Iremos atribuir a classe de escore 5 aos indivíduos do conjunto de validação que apresentarem probabilidade dessa classe maior ou igual a proporção de escore 5 no conjunto de treinamento. Note que, para essas técnicas, estamos utilizando os dados no formato esparsos.

```

newx = dtmMatrix[-tr,],
type = "response") >=
mean(dados$Score[tr]), 1, 0)

# LASSO
vc_lasso      = cv.glmnet(dtmMatrix[tr,], dados$Score[tr],
                           family=c("binomial"), alpha = 1)
ajuste_lasso  = glmnet(dtmMatrix[tr,], dados$Score[tr],
                           family=c("binomial"), alpha = 1)
predito_lasso = ifelse(predict(ajuste_lasso,
                               s = vc_lasso$lambda.1se,
                               newx = dtmMatrix[-tr,],
                               type = "response") >=
mean(dados$Score[tr]), 1, 0)

```

Para o ajuste da árvore de classificação, utilizaremos o pacote `rpart`. Já para o gráfico da árvore utilizaremos o pacote `rpart.plot`. Ainda, faremos a poda com base na medida CP via validação cruzada.

```

# ajuste da arvore
arvore = rpart(Score ~., method = "class",
                data = dtmMatrixArvore[tr,])

# poda
melhorCp = arvore$cptable[which.min(arvore$cptable[, "xerror"]),
                           "CP"]
poda     = prune(arvore, cp = melhorCp)

# predito árvore
predito_arvore = predict(poda,
                           dtmMatrixArvore[-tr, -1],
                           type="class")

# arvore
rpart.plot(poda, type = 4, extra = 102)

```

A Figura 8.5 ilustra a árvore podada gerada pelo código acima. Nessa figura o primeiro número na forma geométrica indica a classe a qual o indivíduo é classificado em cada

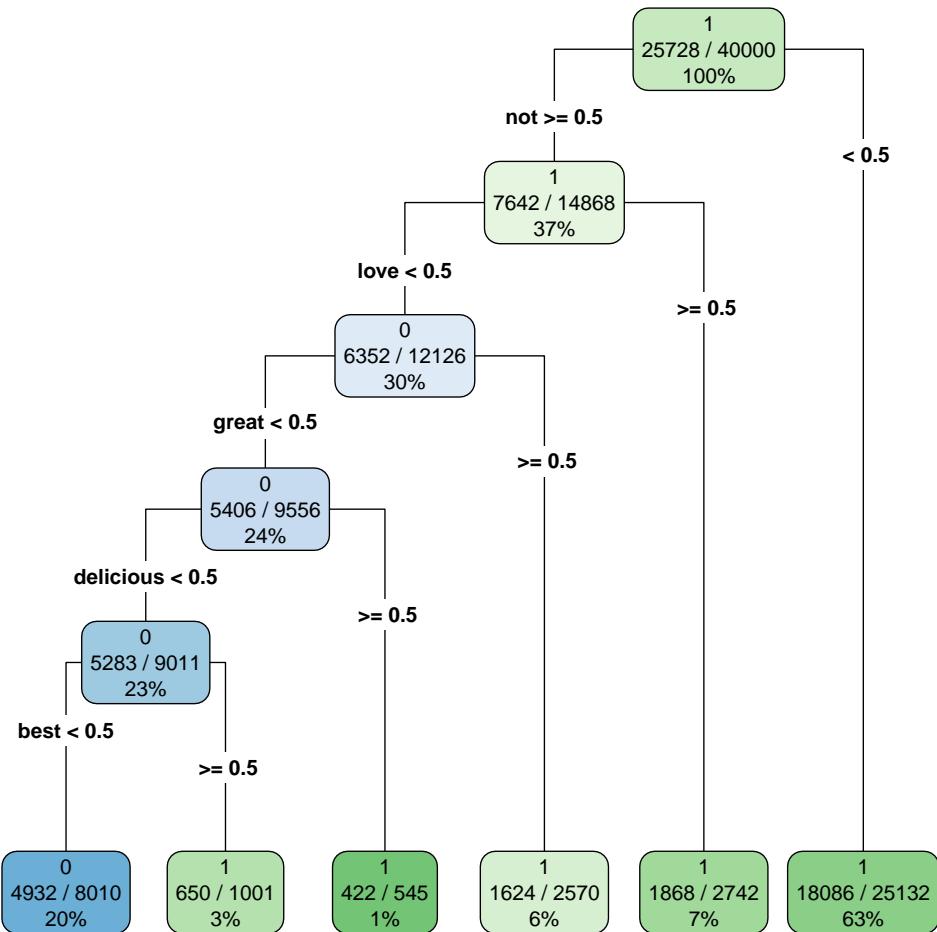


Figura 8.5: Árvore de classificação já podada para os dados da Amazon Fine Food Reviews.

nó/folha, o número de classificações corretas em cada nó com o número de classificações em cada nó e, por fim, a porcentagem de observações em cada nó relativa ao total.

```
# floresta aleatória
floresta = randomForest(Score ~., method = "class",
                         data = dtmMatrixArvore[tr,], ntree = 100)

# predito floresta
predito_floresta = predict(floresta,
                           dtmMatrixArvore[-tr, -1])

bstCV = xgb.cv(data = dtmMatrix[tr,],
                label = dados$Score[tr],
                nthread = 7, nround = 2000,
                nfold = 4, metrics = "error",
                objective = "binary:logistic",
                lambda = .01, verbose = FALSE)

bst = xgboost(data = dtmMatrix[tr,],
              label = dados$Score[tr], nthread = 7,
              nround = which.min(bstCV$evaluation_log$test_error_mean),
              objective = "binary:logistic", verbose = FALSE,
              lambda = 0.01)

predicoes = predict(bst, dtmMatrix[-tr,])
predito_bst = ifelse(predicoes > mean(dados$Score[tr]), 1, 0)
```

Considerando esses dados e as técnicas utilizadas, é possível notar que mínimos quadrados, Ridge e Lasso apresentam uma performance muito semelhante com uma leve vantagem para Lasso. A árvore apresentou o pior desempenho geral e de especificidade entre todas as técnicas, embora tenha apresentado a segunda melhor sensibilidade. A floresta apresenta um desempenho geral muito próximo ao das três técnicas citadas anteriormente. No entanto, apresenta uma sensibilidade melhor e uma baixa especificidade. Já BST apresentou o melhor dos resultados com respeito à porcentagem total de acerto.

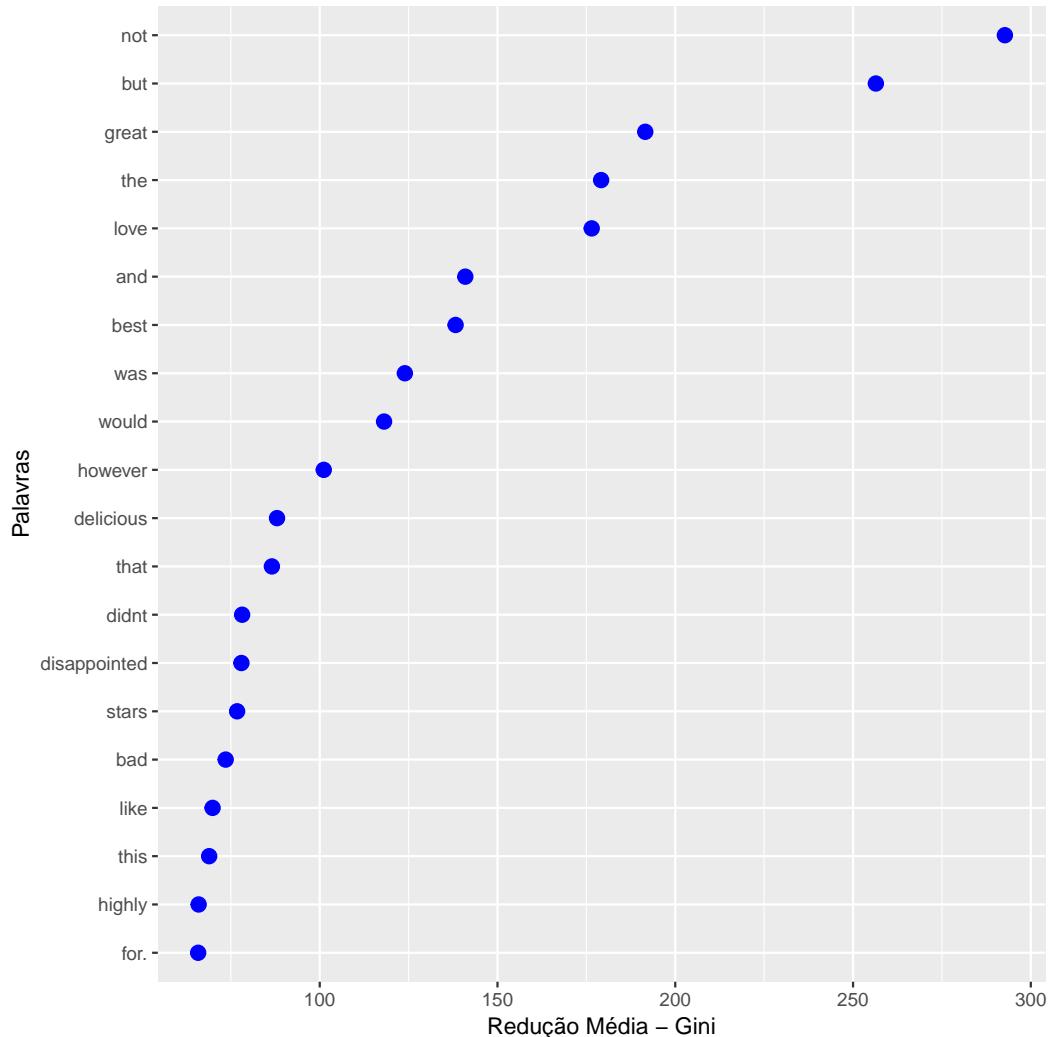


Figura 8.6: Importância obtida com floresta aleatória para os dados da Amazon Fine Food Reviews.

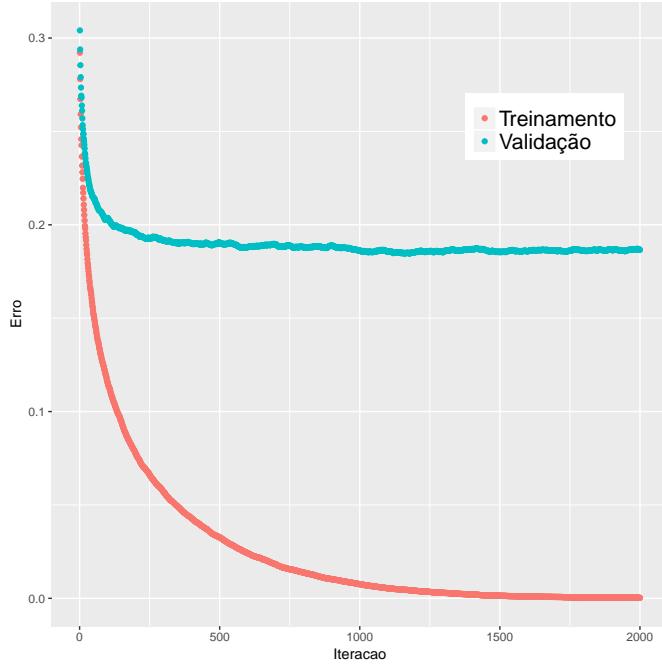


Figura 8.7: Desempenho preditivo considerando o conjunto de treinamento e validação para xgboost.

Tabela 8.3: Desempenho preditivo considerando Mínimos Quadrados, Ridge, Lasso e Árvore.

	% Acerto Total	Sensibilidade	Especificidade
MQ	78.4	80.1	75.4
Ridge	79.4	80.9	76.7
Lasso	79.4	80.5	77.5
Árvore	68.9	88.4	34.2
Floresta	80.2	92.8	57.8
BST	80.7	83.9	75.1

□

Capítulo 9

Outros Aspectos de Classificação

9.1 Assimetria na Função de Perda e Conjuntos de Dados Desbalanceados

Nem sempre a função de perda $\mathbb{I}(g(\mathbf{X}) \neq Y)$ é razoável, pois ela atribui perda 1 para todo erro cometido. Contudo, em muitas situações, erros de diferentes tipos podem ter custos diferentes. Por exemplo, suponha que temos um conjunto de dados de pacientes doentes ($Y = 1$) e saudáveis $Y = 0$. Se essa doença é muito rara ($\mathbb{P}(Y = 0) \approx 0$), então o conjunto será muito desbalanceado. Além disso, o classificador $g(\mathbf{X}) \equiv 0$ terá risco muito pequeno, uma vez que $R(g) = \mathbb{P}(Y = 1)$. Contudo, esse classificador não é satisfatório para esse exemplo: nenhum dos pacientes doentes será corretamente diagnosticado. Entretanto, em geral é muito mais prejudicial classificar um paciente doente como saudável que classificar um paciente saudável como doente. Assim, o risco $R(g) = \mathbb{P}(g(\mathbf{X}) \neq Y)$ não é adequado para esse problema. Uma consequência direta desse fato é que o classificador plugin (Seção 8.1) também não é adequado.

Existem diversas formas de contornar este problema. Uma abordagem comum consiste em buscar cortes diferentes de $1/2$, i.e., buscam-se por regras do tipo

$$g(\mathbf{x}) = \mathbb{I}(\widehat{\mathbb{P}}(Y = 1|\mathbf{x}) \geq K)$$

para diferentes cortes K . Uma forma de escolher K é utilizando a curva ROC, que mostra como a sensibilidade varia com a especificidade para diferentes valores de K . Para evitar que a sensibilidade e especificidade sejam subestimadas, tal gráfico deve ser feito com o conjunto de validação ou teste, nunca com o de treinamento!

A Figura 9.1 ilustra a curva ROC da análise discriminante linear para o conjunto de dados de spams do Exemplo 8.1.

```

library(MASS)
set.seed(401)

data(spam)
index=sample(1:nrow(spam), nrow(spam))
nTrain=round(0.7*nrow(spam))
whichTrain=index[1:nTrain]
ajusteLinear=lda(x=spam[whichTrain,-ncol(spam)],
                  grouping = spam[whichTrain, "spam"])

library(ROCR)
predicted=predict(ajusteLinear, spam[-whichTrain,-ncol(spam)])
predLinear = prediction(predicted$posterior[,2],
                        spam[-whichTrain, "spam"])
perfLinear = performance(predLinear, measure = "tpr",
                         x.measure = "fpr")
plot(perfLinear, col=2, lwd=4, xlab="1-Especificidade",
      ylab="Sensibilidade", cex.lab=1.6)
abline(a=0,b=1, lwd=3)

```

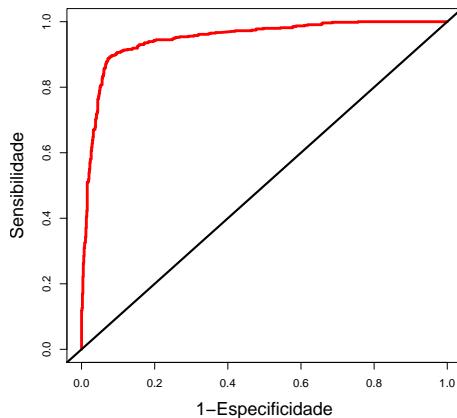


Figura 9.1: Exemplo de curva ROC.

É comum escolher K que maximize o valor de “Sensibilidade+Especificidade”, embora esta não é a única abordagem existente.

Alternativamente, pode-se definir o risco com base em outra outra função de perda que leva em conta que erros diferentes podem ser penalizados de forma distinta. Por exemplo, pode-se tomar

$$\begin{aligned} R(g) &= \mathbb{E}[(\pi_1 \mathbb{I}(Y \neq g(\mathbf{X}) \text{ e } Y = 0)) + (\pi_0 \mathbb{I}(Y \neq g(\mathbf{X}) \text{ e } Y = 1))] = \\ &= \pi_1 \mathbb{P}(Y \neq g(\mathbf{X}) \text{ e } Y = 0) + \pi_0 \mathbb{P}(Y \neq g(\mathbf{X}) \text{ e } Y = 1), \end{aligned}$$

em que π_0 é a probabilidade de uma observação pertencer à classe $Y = 0$ e π_1 é a probabilidade de uma observação pertencer à classe $Y = 1$. Assim, tal risco dá maior importância do erro de uma observação da classe 1 ser classificada como pertencente à classe 0 e menor importância ao erro de uma observação da classe 0 ser classificada como pertencente à classe 1. A função $g(\mathbf{x})$ que minimiza $R(g)$ é dada por $g(\mathbf{x}) = \mathbb{I}(\mathbb{P}(Y = 1|\mathbf{x}) > \pi_1)$. De fato, a decisão ótima é $g(\mathbf{x}) = 1$ se, e somente se,

$$\pi_0 \mathbb{P}(Y = 0|\mathbf{x}) \geq \pi_1 \mathbb{P}(Y = 1|\mathbf{x}) \iff \mathbb{P}(Y = 1|\mathbf{x}) \geq \pi_1.$$

Isso motiva o uso do classificador plugin $\mathbb{I}(\widehat{\mathbb{P}}(Y = 1|\mathbf{x}) \geq \widehat{\mathbb{P}}(Y = 0))$ em que $\widehat{\mathbb{P}}(Y = 1)$ é a proporção amostral da classe de interesse. Ou seja, ao invés de se usar $1/2$ como corte, utiliza-se $\widehat{\mathbb{P}}(Y = 1)$.

Para classificadores que não são baseados na estimativa de $\mathbb{P}(Y = 1|\mathbf{x})$, outras abordagens são utilizadas como, por exemplo, a atribuição de pesos maiores a observações da categoria menos frequente. Tais pesos são então usados de modo que a contribuição de cada observação na criação do classificador seja proporcional ao peso por ela recebida. A forma como estes pesos são utilizados depende do método de classificação em questão.

9.2 Dataset Shift e Viés de Seleção

Em uma parcela significativa de problemas de predição, a suposição básica de que as observações onde um classificador será aplicado são identicamente distribuídas às observações rotuladas usadas para construir g não é razoável. Tal situação é chamada de *dataset shift* (Sugiyama et al., 2017).

Muitas vezes, o *dataset shift* ocorre devido a um *viés de seleção*, que faz com que os dados rotulados tenham características diferentes daqueles em que os classificadores serão aplicados. Por exemplo, em pesquisas cosmológicas que visam estimar a distância de uma galáxia até a Terra (Exemplo 1.2), é muito mais fácil observar a distância real de galáxias que são luminosas (Freeman et al., 2017, Izbicki et al., 2017) do que galáxias pouco luminosas. Isso faz com que o conjunto rotulado tenha mais galáxias luminosas que o conjunto em que os métodos de predição serão aplicados.

Dataset shift também podem ocorrer quando se utiliza dados de um certo domínio (e.g., resenhas da Amazon) para criar um classificador que será aplicado em outro domínio (e.g., textos do Twitter).

Para que seja possível aprender algo com o conjunto rotulado, é necessário fazer algumas suposição sobre como esse conjunto se relaciona ao conjunto em que o classificador será aplicado. Nesta seção investigaremos duas suposições: *covariate shift* (Seção 9.2.1) e *prior shift* (Seção 9.2.2). Em ambas as seções, assumiremos que o conjunto rotulado é dado por observações i.i.d.'s $(\mathbf{X}_1^L, Y_1^L), \dots, (\mathbf{X}_{n_L}^L, Y_{n_L}^L)$, enquanto que o conjunto em que a função de predição será aplicada é dado por observações i.i.d.'s $(\mathbf{X}_1^U, Y_1^U), \dots, (\mathbf{X}_{n_U}^U, Y_{n_U}^U)$. Note que os valores de $Y_1^U, \dots, Y_{n_U}^U$ não são observados.

9.2.1 Covariate Shift

A suposição de *covariate shift* consiste em assumir que $Y^L | \mathbf{X}^L \sim Y^U | \mathbf{X}^U$, de modo que a distribuição conjunta de (\mathbf{X}^L, Y^L) difere de (\mathbf{X}^U, Y^U) apenas segundo a distribuição marginal das covariáveis. Outra caracterização de tal suposição é a de que o processo de escolha de que amostras serão rotuladas depende apenas das covariáveis medidas (veja [Izbicki et al. 2017](#) para maior detalhamento).

À primeira vista, pode parecer que o *covariate shift* não é um problema para problemas de predição: como a distribuição de $Y|\mathbf{x}$ é a mesma em ambos os conjuntos de dados, então $\mathbb{P}(Y^L = 1|\mathbf{x}^L) = \mathbb{P}(Y^U = 1|\mathbf{x}^U)$. Assim, a quantidade $\mathbb{P}(Y^U = 1|\mathbf{x}^U)$, que é essencial para criar classificadores *plug-in*, é a mesma no conjunto rotulado e no conjunto de interesse. Note, contudo, que isso não implica que um estimador razoável de $\mathbb{P}(Y^L = 1|\mathbf{x}^L)$ segundo o conjunto rotulado tenha bom desempenho no conjunto não-rotulado: frequentemente $\mathbb{P}(Y^L = 1|\mathbf{x}^L)$ estará bem estimado somente em regiões em que há bastantes dados rotulados, e essas regiões não necessariamente coincidem com regiões em que há bastantes dados não-rotulados; veja a Figura 9.2 para um exemplo no contexto de regressão. Levando em consideração apenas o conjunto rotulado, a reta tracejada parece ter bom desempenho, contudo isso não ocorre para o conjunto não-rotulado.

Sob uma perspectiva estatística, o problema ocorre porque a função de risco usada para construir o método de predição depende da distribuição de \mathbf{X} . Assim, um estimador que tem bom desempenho com respeito a $f_L(\mathbf{x})$ não necessariamente tem bom desempenho com respeito a $f_U(\mathbf{x})$.

Em cenários com *covariate shift*, enfrentamos dois problemas:

- (i) como comparar métodos de predição
- (ii) como construir bons métodos de predição.

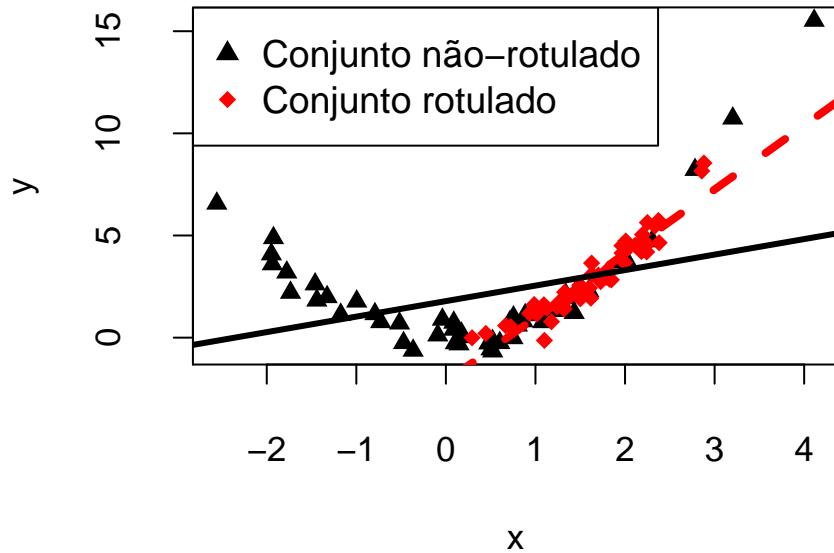


Figura 9.2: Ilustração do covariate shift em uma regressão linear: a linha tracejada é razoável para prever novos dados do conjunto rotulado, mas ela está longe de ser ótima para o conjunto não-rotulado.

Uma solução para ambos os problemas é atribuir pesos a cada observação rotulada de modo que a distribuição ponderada do conjunto rotulado seja próxima à distribuição do conjunto não-rotulado.

Mais especificamente, a ideia chave para resolver (i) é notar que uma função de risco razoável é o risco *sob a distribuição do conjunto não-rotulado*, $\mathbb{E}[L(g, (\mathbf{X}^U, Y^U))]$, em que L é uma função de perda (veja a Observação 1.3). Sob a suposição de *covariate shift*, este se relaciona com a distribuição do conjunto rotulado da seguinte maneira:

$$\begin{aligned}
 R(g) &:= \mathbb{E}[L(g, (\mathbf{X}^U, Y^U))] = \int L(g, (\mathbf{x}, y)) f_U(y|\mathbf{x}) f_U(\mathbf{x}) d\mathbf{x} dy = \\
 &= \int L(g, (\mathbf{x}, y)) f_L(y|\mathbf{x}) \frac{f_U(\mathbf{x})}{f_L(\mathbf{x})} f_L(\mathbf{x}) d\mathbf{x} dy \\
 &= \mathbb{E}[L(g, (\mathbf{X}^L, Y^L)) \beta(\mathbf{X}^L)],
 \end{aligned} \tag{9.1}$$

em que

$$\beta(\mathbf{x}) = \frac{f_U(\mathbf{x})}{f_L(\mathbf{x})}.$$

A função $\beta(\mathbf{x})$ é responsável por ponderar cada observação \mathbf{x} de acordo com sua frequência no conjunto não-rotulado. Na prática, essa função deve ser estimada. Uma forma de se fazer isso é estimando a densidade de \mathbf{x} no conjunto rotulado e no conjunto não-rotulado separadamente (pode-se usar, por exemplo, o estimador de Nadaraya-Watson).

Um estimador alternativo para $\beta(\mathbf{x})$ consiste em criar o conjunto dados

$$(\mathbf{X}_1^L, S_1), \dots, (\mathbf{X}_{n_L}^L, S_{n_L}), (\mathbf{X}_1^U, S_{n_L+1}), \dots, (\mathbf{X}_{n_U}^U, S_{n_L+n_U}),$$

em que $S_i = \mathbb{I}(1 \leq i \leq n_L)$ é um indicador da amostra ser rotulada. Com base nessa amostra, estima-se então $\mathbb{P}(S = 1|\mathbf{x})$, a probabilidade de uma unidade amostral ser proveniente da amostra rotulada (usando, por exemplo, uma regressão logística). Como

$$\beta(\mathbf{x}) = \frac{\mathbb{P}(S = 1)}{\mathbb{P}(S = 0)} \frac{\mathbb{P}(S = 0|\mathbf{x})}{\mathbb{P}(S = 1|\mathbf{x})},$$

utiliza-se então como estimador de β

$$\hat{\beta}(\mathbf{x}) := \frac{\hat{\mathbb{P}}(S = 0)}{\hat{\mathbb{P}}(S = 1)} \frac{\hat{\mathbb{P}}(S = 1|\mathbf{x})}{\hat{\mathbb{P}}(S = 0|\mathbf{x})}.$$

Seja $\hat{\beta}$ uma estimativa de β . A Equação 9.1 indica que uma forma de se estimar o risco de um método de predição g no conjunto não-rotulado é utilizar

$$\hat{R}(g) = \frac{1}{\tilde{n}_L} \sum_{k=1}^{\tilde{n}_L} L(g, (\tilde{\mathbf{X}}_k^L, \tilde{Y}_k^L)) \hat{\beta}(\tilde{\mathbf{X}}_k^L),$$

em que $(\tilde{\mathbf{X}}_1^L, \tilde{Y}_1^L), \dots, (\tilde{\mathbf{X}}_{\tilde{n}_L}^L, \tilde{Y}_{\tilde{n}_L}^L)$ é uma amostra de validação do conjunto de rotulado. Isto é, com o uso dos pesos, pode-se estimar o risco de interessado utilizando-se apenas a amostra rotulada. \hat{R} pode então ser utilizado para se comparar diversos modelos. Note, contudo, que se β é muito alto para alguns pontos, o estimador do risco terá desempenho ruim, uma vez que consistirá efetivamente em avaliar a função de perda em apenas algumas observações. Em particular, $\beta(\mathbf{x}) = 0$ em regiões do espaço amostral em que há apenas observações não-rotuladas, de modo que a suposição de *covariate-shift* não é útil nesse caso.

Os pesos $\hat{\beta}$ também podem ser utilizados para responder (ii), i.e., para criar funções de predição com bom poder preditivo no conjunto não-rotulado. Isso pode ser feito utilizando versões ponderadas dos métodos de predição estudados nesse livro. Por exemplo, a função

`glmnet` do R permite que pesos sejam passados como um dos argumentos para que o lasso seja ajustado.

Note que a solução apresentada aqui para o problema de viés de seleção sob *covariate shift* pode ser utilizada tanto para problemas de classificação quanto para problemas de regressão, bastando utilizar uma função de perda L adequada para cada problema.

9.2.2 Prior Shift

A suposição de *covariate shift* consiste em supor que $\mathbf{X}^L|Y^L \sim \mathbf{X}^U|Y^U$, de modo que a distribuição conjunta de (\mathbf{X}^L, Y^L) difere de (\mathbf{X}^U, Y^U) apenas segundo a distribuição marginal da variável resposta. Como $\mathbb{P}(Y^U = 1) \neq \mathbb{P}(Y^L = 1)$, então $\mathbb{P}(Y^U = 1|\mathbf{x}^U) \neq \mathbb{P}(Y^L = 1|\mathbf{x}^L)$, de modo que alguma correção deve ser feita para estimar $\mathbb{P}(Y^U = 1|\mathbf{x}^U)$. Pode-se mostrar que, sob a suposição de *prior shift* ([Saerens et al., 2002](#)),

$$\mathbb{P}(Y^U = 1|\mathbf{x}^U) = \frac{\frac{\mathbb{P}(Y^U=1)}{\mathbb{P}(Y^L=1)}\mathbb{P}(Y^L = 1|\mathbf{x}^L)}{\sum_{i=0}^1 \frac{\mathbb{P}(Y^U=i)}{\mathbb{P}(Y^L=i)}\mathbb{P}(Y^L = i|\mathbf{x}^L)}.$$

Assim, para se estimar $\mathbb{P}(Y^U = 1|\mathbf{x}^U)$ basta estimar:

- $\mathbb{P}(Y^L = 1|\mathbf{x}^L)$, que pode ser feito usando as técnicas apresentadas na Parte II deste livro,
- $\mathbb{P}(Y^L = 1)$, que pode ser feito usando a proporção de amostras com rótulo $Y = 1$ na amostra rotulada,
- $\mathbb{P}(Y^U = 1)$.

Em muitos problemas, uma estimativa razoável de $\mathbb{P}(Y^U = 1)$ é conhecida de antemão. Por exemplo, considere um estudo caso-controle para criar um classificador de que um paciente tem determinada doença. Neste caso, a amostra rotulada é criada de modo que o número de pacientes doentes (e não doentes) investigados seja um número pré-fixado anteriormente. Assim, a proporção de doentes na amostra é diferente da proporção de doentes na população, $\mathbb{P}(Y^U = 1)$. Contudo, $\mathbb{P}(Y^U = 1)$ é frequentemente conhecido devido a estudos prévios.

Quando $\mathbb{P}(Y^U = 1)$ não é conhecida, é necessário estimá-la. Um método bastante popular para estimar essa quantidade foi desenvolvido por , e consiste em notar que, seja $g(\mathbf{x})$ é um classificador, então ([Vaz et al., 2018](#))

$$\mathbb{P}(Y^U = 1) = \frac{\mathbb{P}(g(\mathbf{X}^U) = 1) - \mathbb{P}(g(\mathbf{X}^L) = 1|Y^L = 0)}{\mathbb{P}(g(\mathbf{X}^L) = 1|Y^L = 1) - \mathbb{P}(g(\mathbf{X}^L) = 1|Y^L = 0)}.$$

Todos os termos desta equação podem ser estimados usando o conjunto de dados disponível.

9.3 Combinando classificadores

Combinar diversos classificadores construídos usando o mesmo conjunto de treinamento pode levar a um classificador com poder preditivo ainda maior. Bagging, florestas aleatórias e boosting (Seções 8.4 e 8.5) são algumas formas de se fazer isso. Como vimos, enquanto que bagging e florestas aleatórias foram criados para combinar funções de predição aproximadamente não viesadas (mais especificamente, árvores sem poda); boosting combina classificadores fracos. Existem, contudo, outras formas de se combinar classificadores; nesta seção exploramos uma delas, o *blending* (muitas vezes chamado de *stacked ensembling*).

A ideia deste método é criar um *meta-classificador*, treinado usando como covariáveis as previsões dos modelos a serem combinados. Mais especificamente, divide-se o conjunto de treinamento em duas partes: treinamento (e.g., 90%) e ensemble set (e.g., 10%). Sejam g_1, \dots, g_B as funções de predição treinadas utilizando-se o conjunto de treinamento. Tais funções podem ser obtidas usando-se quaisquer métodos (e.g., regressão logística, SVM, florestas aleatórias etc). Denotando o ensemble set por $(\tilde{\mathbf{x}}_1, \tilde{y}_1), \dots, (\tilde{\mathbf{x}}_{\tilde{n}}, \tilde{y}_{\tilde{n}})$, ajustamos então um classificador para o conjunto $(\tilde{\mathbf{w}}_1, \tilde{y}_1), \dots, (\tilde{\mathbf{w}}_{\tilde{n}}, \tilde{y}_{\tilde{n}})$, em que $\tilde{\mathbf{w}}_i = (g_1(\tilde{\mathbf{X}}_i), \dots, g_B(\tilde{\mathbf{X}}_i))$.

Para exemplificar tal método, considere o conjunto de dados `spam`. Ajustamos aqui os seguintes métodos: análise discriminante linear, regressão logística com penalização L1, florestas aleatórias e SVM.

```
library(MASS)
set.seed(401)

data(spam)
index=sample(1:nrow(spam), nrow(spam))
nTrain=round(0.5*nrow(spam))
nEnsemble=round(0.2*nrow(spam))
whichTrain=index[1:nTrain]
whichEnsemble=index[(nTrain+1):(nTrain+nEnsemble)]
whichTest=index[-c(1):(nTrain+nEnsemble)]]

# LDA
```

```

ajusteLinear=lda(x=spam[whichTrain,-ncol(spam)],
                  grouping = spam[whichTrain, "spam"])
predLinear=predict(ajusteLinear,
                    spam[, -ncol(spam)])$posterior[, 2]

# Logistica
formula=paste("spam~", paste(colnames(spam)[-ncol(spam)],
                                collapse = "+"))
ajusteLogistico=cv.glmnet(x=as.matrix(spam[whichTrain,
                                              -ncol(spam)]),
                           y = as.factor(
                               spam[whichTrain, "spam"]),
                           family="binomial",
                           alpha=1)
lambdaOtimo = ajusteLogistico$lambda.min
predLogistico=predict(ajusteLogistico,
                      newx=as.matrix(spam[, -ncol(spam)]),
                      s = lambdaOtimo,type="response")

# Floresta
floresta=randomForest::randomForest(x=spam[whichTrain,
                                              -ncol(spam)],
                                      y = as.factor(
                                          spam[whichTrain, "spam"]))
predFloresta=predict(floresta,spam[, -ncol(spam)],type="prob")[,2]

# SVM
ajusteSVM=e1071::tune(e1071::svm,
                       train.x=spam[whichTrain,-ncol(spam)],
                       train.y=as.factor(spam[whichTrain, "spam"]),
                       kernel="radial",scale=TRUE,
                       ranges=list(cost=c(5,10,20),
                                   gamma = c(0.001,0.01,0.05)))
melhorSVM = ajusteSVM$best.model
predSVM=predict(melhorSVM,spam[, -ncol(spam)])

# Ensemble

```

```

data.ensemble=data.frame(response=spam[whichEnsemble,ncol(spam)],
                         predLinear=predLinear[whichEnsemble],
                         predLogistico=predLogistico[whichEnsemble],
                         predFloresta=predFloresta[whichEnsemble],
                         predSVM=predSVM[whichEnsemble])
modelo.ensemble.logistic=glm("response~predLinear+
                                predLogistico+predFloresta+predSVM",
                                data=data.ensemble,family=binomial())
log=cbind(modelo.ensemble.logistic$coefficients)
names(log)=names(coef(modelo.ensemble.logistic))
colnames(log)=c("Coeficientes")
knitr::kable(round(log,3))

```

	Coeficientes
(Intercept)	-4.370
predLinear	-0.228
predLogistico	0.981
predFloresta	8.069
predSVM1	0.551

```

data.test=data.frame(response=spam[whichTest,ncol(spam)],
                      predLinear=predLinear[whichTest],
                      predLogistico=predLogistico[whichTest],
                      predFloresta=predFloresta[whichTest],
                      predSVM=predSVM[whichTest])

predictedTest.logistic=predict(modelo.ensemble.logistic,
                                 newdata = data.test,
                                 type="response")

# Discriminante:
mean((predLinear[whichTest]>mean(spam$spam))!=
      spam[whichTest,]$spam)

## [1] 0.09847936

# Logistica:
mean((predLogistico[whichTest]>mean(spam$spam))!=
      spam[whichTest,]$spam)

```

```

## [1] 0.08037654

# Floresta:
mean((predFloresta[whichTest]>mean(spam$spam) ) !=  

      spam[whichTest,]$spam)

## [1] 0.06517017

# SVM:
mean((predSVM[whichTest]) !=  

      spam[whichTest,]$spam)

## [1] 0.07675597

# Ensemble:
mean(predictedTest.logistic>mean(spam$spam) ) !=  

      spam[whichTest,]$spam)

## [1] 0.06010138

```

Neste caso, observamos que o ensemble (feito a partir de uma regressão logística das previsões dos demais) foi capaz de melhorar o desempenho dos demais classificadores. Além disso, vê-se que quase todo o peso do modelo foi dado ao modelo de florestas aleatórias, o que explica porque o desempenho deste é semelhante ao do modelo combinado (enquanto o primeiro tem risco de aproximadamente 6,5%, o segundo tem risco de aproximadamente 6,0%).

9.4 Teoria do Aprendizado Estatístico

Seja $\mathbf{Z}_i = (\mathbf{X}_i, Y_i)$, $i = 1, \dots, n$ vetores aleatórios i.i.d. e $f : \mathbb{R}^{d+1} \rightarrow \mathbb{R}$ uma função. Defina

$$P(f) := \mathbb{E}[f(\mathbf{Z})]$$

e

$$P_n(f) = \frac{1}{n} \sum_{i=1}^n f(\mathbf{Z}_i)$$

e seja

$$\hat{f} = \arg \min_{f \in \mathcal{F}} P_n(f)$$

e

$$f^* = \arg \min_{f \in \mathcal{F}} P(f),$$

em que \mathcal{F} é um conjunto de funções de interesse. O principal objetivo da teoria do aprendizado estatístico é dar garantias sobre o quanto longe $P(\hat{f})$ está de $P(f^*)$. Embora a teoria do aprendizado estatístico seja bastante geral, nos focaremos inicialmente no problema de classificação:

Exemplo 9.1 No contexto de classificação, podemos tomar $f(\mathbf{Z}) = \mathbb{I}(Y \neq g(\mathbf{X}))$, de modo que $P(f) = R(g)$ (i.e., é o risco de classificação) e $P_n(f) = \hat{R}_{tr}(g) := \frac{1}{n} \sum_{i=1}^n \mathbb{I}(Y_i \neq g(\mathbf{X}_i))$ (i.e., é a proporção de erros que g faz no conjunto de treinamento). Uma forma de se escolher um classificador é definir

$$\hat{g} = \arg \min_{g \in \mathcal{G}} \hat{R}_{tr}(g),$$

em que \mathcal{G} é uma classe de classificadores (e.g., o conjunto de todos os classificadores lineares). Essa abordagem é chamada de minimização do erro empírico. Assim, a teoria do aprendizado estatístico fornece garantias sobre tal procedimento. Mais especificamente, essa teoria fornece garantias sobre o quanto longe o risco de \hat{g} está do risco do oráculo g^* , definido por

$$g^* = \arg \min_{g \in \mathcal{G}} R(g),$$

ou seja, o melhor classificador dentro de \mathcal{G} (como visto na Seção 7.3). Isso decorre do fato de que $\min_{g \in \mathcal{G}} \hat{R}_{tr}(g) = \min_{f \in \mathcal{F}} \hat{P}_n(f)$ e $\min_{g \in \mathcal{G}} R(g) = \min_{f \in \mathcal{F}} P(f)$, onde $\mathcal{F} = \{f : f(\mathbf{Z}) = \mathbb{I}(Y \neq g(\mathbf{X})) \text{ para algum } g \in \mathcal{G}\}$,

A estratégia da teoria do aprendizado estatístico para limitar o quanto longe o risco de \hat{g} está do risco do oráculo g^* é limitar o quanto longe $R(g)$ está de $\hat{R}_{tr}(g)$ para todo $g \in \mathcal{G}$ simultaneamente. Uma forma de se obter tal limitante é através do Teorema VC:

Teorema 9.1 (Teorema VC) Se \mathcal{F} é um conjunto de funções binárias, então, para todo $\epsilon > \sqrt{2/n}$,

$$\mathbb{P}(\sup_{f \in \mathcal{F}} |P_n(f) - P(f)| > \epsilon) \leq 4s(\mathcal{F}, 2n)e^{-n\epsilon^2/8}, \quad (9.2)$$

em que $s(\mathcal{F}, 2n)$ é o shattering number da classe \mathcal{F} para uma amostra de tamanho n . Assim, com probabilidade ao menos $1 - \delta$,

$$\sup_{f \in \mathcal{F}} |P_n(f) - P(f)| \leq \sqrt{\frac{8}{n} \log \left(\frac{4s(\mathcal{F}, 2n)}{\delta} \right)}.$$

Antes de detalhar o que é o *shattering number* $s(\mathcal{F}, 2n)$, vamos discutir o motivo de tal teorema de fato nos ajudar a atingir nosso objetivo. Seja $\hat{f} = \arg \min_{f \in \mathcal{F}} P_n(f)$ e $f^* = \arg \min_{f \in \mathcal{F}} P(f)$. O fato chave da teoria VC é que o Teorema 9.1 implica que, com probabilidade ao menos $1 - 4s(\mathcal{F}, 2n)e^{-2n\epsilon^2/8}$,

$$\begin{aligned} P(\hat{f}) - P(f^*) &= P(\hat{f}) - P_n(\hat{f}) + P_n(\hat{f}) - P_n(f^*) + P_n(f^*) - P(f^*) \\ &\leq P(\hat{f}) - P_n(\hat{f}) + P_n(f^*) - P(f^*) \\ &\leq 2\epsilon, \end{aligned}$$

onde a primeira desigualdade segue do fato de que, por definição, $P_n(\hat{f}) - P_n(f^*) \leq 0$, e a última desigualdade segue da Equação 9.2. Em palavras, se n é grande e $s(\mathcal{F}, 2n)$ cresce a uma taxa menor que $e^{-n\epsilon^2/8}$ (em breve veremos quando esta última condição ocorre), então com probabilidade alta $P(\hat{f}) \approx P(f^*)$. No contexto de classificação isso implica que, com probabilidade alta, o risco do estimador encontrado via minimização de risco empírico estará próximo do risco do oráculo.

Pode-se também mostrar que, com probabilidade ao menos $1 - \delta$, $P(\hat{f}) - P(f^*) \leq 2\sqrt{\frac{8}{n} \log\left(\frac{4s(\mathcal{F}, 2n)}{\delta}\right)}$

Note que, em particular, o Teorema VC garante que, se

$$\begin{aligned} 4s(\mathcal{F}, 2n)e^{-2n\epsilon^2/8} &\xrightarrow[n \rightarrow \infty]{} 0, \\ \text{então} \\ P(\hat{f}) &\xrightarrow[n \rightarrow \infty]{\mathbb{P}} P(f^*), \end{aligned}$$

isto é, $P(\hat{f})$ converge em probabilidade para $P(f^*)$.

Shattering number. Note que, se \mathcal{F} contém apenas funções binárias, então $(f(\mathbf{z}_1), \dots, f(\mathbf{z}_n)) \in \{0, 1\}^n$ para todo $\mathbf{z}_1, \dots, \mathbf{z}_n$ e $f \in \mathcal{F}$. Seja $\mathcal{F}(\mathbf{z}_1, \dots, \mathbf{z}_n) = \{(f(\mathbf{z}_1), \dots, f(\mathbf{z}_n)) : f \in \mathcal{F}\}$ o conjunto de todos os valores que o vetor $(f(\mathbf{z}_1), \dots, f(\mathbf{z}_n))$ pode assumir para $\mathbf{z}_1, \dots, \mathbf{z}_n$ fixos. Se \mathcal{F} é uma classe rica de funções, esperamos que $\mathcal{F}(\mathbf{z}_1, \dots, \mathbf{z}_n)$ contenha um grande número de vetores. Com base nessa ideia, uma forma de quantificar a complexidade de uma classe \mathcal{F} de funções binárias é o número máximo de elementos que $\mathcal{F}(\mathbf{z}_1, \dots, \mathbf{z}_n)$ pode ter levando em conta todos os possíveis valores que $\mathbf{z}_1, \dots, \mathbf{z}_n$ podem assumir. Tal quantidade é chamada de *shattering number*. Mais especificamente, tal número é dado por

$$s(\mathcal{F}, n) := \sup_{\mathbf{z}_1, \dots, \mathbf{z}_n} |\mathcal{F}(\mathbf{z}_1, \dots, \mathbf{z}_n)|.$$

Note que, necessariamente, $s(\mathcal{F}, n) \leq 2^n$. Quando $|\mathcal{F}(\mathbf{z}_1, \dots, \mathbf{z}_n)| = 2^n$, dizemos que os dados $\mathbf{z}_1, \dots, \mathbf{z}_n$ são destruídos (shattered) por \mathcal{F} .

Exemplo 9.2 Seja $\mathcal{F} = \{f_t : t \in \mathbb{R}\}$, em que $f_t(z) = \mathbb{I}(z > t)$, $z \in \mathbb{R}$. Para quaisquer $z_1 < \dots < z_n$, temos que

$$\mathcal{F}(z_1, \dots, z_n) = \{(0, \dots, 0, 0), (0, \dots, 0, 1), (0, \dots, 1, 1), \dots, (1, \dots, 1, 1)\}.$$

Logo $|\mathcal{F}(z_1, \dots, z_n)| = n + 1$. Em geral, $|\mathcal{F}(z_1, \dots, z_n)| \leq n + 1$, de modo que, neste caso, $s(\mathcal{F}, n) = n + 1$.

Uma condição necessária e suficiente para a consistência (com relação ao oráculo) do procedimento de minimização empírica do risco é que

$$\log s(\mathcal{F}, n)/n \longrightarrow 0.$$

Assim, se \mathcal{F} é muito complexa (e.g., $s(\mathcal{F}, n) = 2^n$) não há consistência.

Dimensão VC. Outro conceito importante na Teoria VC é o de dimensão VC, que também é uma forma de medir a complexidade de um espaço de funções binárias. Tal dimensão é dada por:

$$VC(\mathcal{F}) = \sup\{n : s(\mathcal{F}, n) = 2^n\}.$$

Em palavras, $VC(\mathcal{F})$ é o maior tamanho amostral n para o qual existe uma amostra $\mathbf{z}_1, \dots, \mathbf{z}_n$ que é destruída por \mathcal{F} . Se \mathcal{F} contém uma grande variedade de funções, é razoável esperar que sua dimensão VC seja grande.

Exemplo 9.3 No Exemplo 9.2, $VC(\mathcal{F}) = 1$, pois $s(\mathcal{F}, 1) = 2 = 2^1$, mas $s(\mathcal{F}, n) = n + 1 < 2^n$ para $n > 1$.

Em geral, a dimensão VC de espaços usuais no contexto de classificação já foram calculados. Veja REFERENCIA para alguns exemplos.

A dimensão VC também está relacionada com o *shattering number* via o Lema de Sauer:

Teorema 9.2 (Lema de Sauer) Se $d := VC(\mathcal{F}) < \infty$, então

$$s(\mathcal{F}, n) \leq \left(\frac{en}{d}\right)^d$$

para todo $n > d$. Além disso, se a dimensão VC é infinita, então $s(\mathcal{F}, n) = 2^n$ para todo n .

Tal lema é útil pois permite uma forma alternativa para o limitante do Teorema VC. Mais especificamente, se a dimensão VC d de uma classe \mathcal{F} é finita, então o Teorema VC juntamente com o Lema de Sauer implicam que, se $n > d$, então com probabilidade ao menos $1 - \delta$

$$\sup_{f \in \mathcal{F}} |P_n(f) - P(f)| \leq \sqrt{\frac{8}{n} \left(\log\left(\frac{4}{\delta}\right) + d \log\left(\frac{ne}{d}\right) \right)}.$$

Segue que se a dimensão VC de \mathcal{F} é finita, então $P(\hat{f})$ converge em probabilidade para $P(f^*)$. Além disso, se a dimensão VC de \mathcal{F} é infinita, então $P(\hat{f})$ não converge em probabilidade para $P(f^*)$. Segue que a dimensão VC caracteriza completamente a consistência do método de minimização empírica do risco em uma classe de funções.

Exemplo 9.4 (Aplicação para a escolha de tuning parameters via data splitting)

9.4.1 Prova do teorema VC

Inicialmente limitaremos o quanto longe $P_n(f)$ está de $P(f)$ para uma função f fixa. Uma forma de fazer isso se dá com a utilização da desigualdade de Hoeffding:

Teorema 9.3 (Desigualdade de Hoeffding) Se W_1, \dots, W_n são variáveis aleatórias independentes e W_i tem suporte em (a_i, b_i) então, para todo $t > 0$,

$$\mathbb{P}(|\bar{W}_n - \mu| > \epsilon) \leq 2e^{-2n^2\epsilon^2/\sum_{i=1}^n (b_i - a_i)^2},$$

em que $\bar{W}_n = n^{-1} \sum_{i=1}^n W_i$ e $\mu = \mathbb{E}[\bar{W}_n]$.

De tal desigualdade, segue imediatamente que

Corolário 9.1 Para toda função binária f ,

$$\mathbb{P}(|P_n(f) - P(f)| > \epsilon) \leq 2e^{-2n\epsilon^2}$$

e, portanto, com probabilidade ao menos $1 - \delta$,

$$|P_n(f) - P(f)| \leq \sqrt{\frac{1}{2n} \log(2/\delta)}.$$

Exemplo 9.5 No contexto de classificação, o Corolário 9.1 implica que, para todo classificador binário g ,

$$\mathbb{P}(|\hat{R}_{tr}(g) - R(g)| > \epsilon) \leq 2e^{-2n\epsilon^2}$$

e, portanto, com probabilidade ao menos $1 - \delta$,

$$|\hat{R}_{tr}(g) - R(g)| \leq \sqrt{\frac{1}{2n} \log(2/\delta)}.$$

Lema 9.1 (Lema da simetrização) Para todo $\epsilon > \sqrt{2/n}$,

$$\mathbb{P} \left(\sup_{f \in \mathcal{F}} |P_n(f) - P(f)| > \epsilon \right) \leq 2\mathbb{P} \left(\sup_{f \in \mathcal{F}} |P_n(f) - P'_n(f)| > \epsilon/2 \right),$$

em que $P'_n(f) = \frac{1}{n} \sum_{i=1}^n f(\mathbf{Z}'_i)$, com $\mathbf{Z}'_1, \dots, \mathbf{Z}'_n$ uma amostra de vetores aleatórios iids a $\mathbf{Z}_1, \dots, \mathbf{Z}_n$.

Demonstração. Seja $\hat{f} = \arg \sup_{f \in \mathcal{F}} |P_n(f) - P(f)|$. Note que, pela desigualdade triangular,

$$\epsilon < |P_n(\hat{f}) - P(\hat{f})| = |P_n(\hat{f}) - P'_n(\hat{f}) + P'_n(\hat{f}) - P(\hat{f})| \leq |P_n(\hat{f}) - P'_n(\hat{f})| + |P'_n(\hat{f}) - P(\hat{f})|.$$

Assim,

$$|P_n(\hat{f}) - P(\hat{f})| > \epsilon \text{ e } |P'_n(\hat{f}) - P(\hat{f})| \leq \epsilon/2 \Rightarrow |P_n(\hat{f}) - P'_n(\hat{f})| > \epsilon/2.$$

Logo,

$$\mathbb{I}(|P_n(\hat{f}) - P(\hat{f})| > \epsilon) \mathbb{I}(|P'_n(\hat{f}) - P(\hat{f})| \leq \epsilon/2) \leq \mathbb{I}(|P_n(\hat{f}) - P'_n(\hat{f})| > \epsilon/2).$$

Tomando a esperança sobre $\mathbf{Z}'_1, \dots, \mathbf{Z}'_n$, concluímos que

$$\mathbb{I}(|P_n(\hat{f}) - P(\hat{f})| > \epsilon) \mathbb{P}'(|P'_n(\hat{f}) - P(\hat{f})| \leq \epsilon/2) \leq \mathbb{P}'(|P_n(\hat{f}) - P'_n(\hat{f})| > \epsilon/2). \quad (9.3)$$

Mas, pela desigualdade de Chebyshev,

$$\mathbb{P}'(|P(\hat{f}) - P'_n(\hat{f})| > \epsilon/2) \leq \frac{4\mathbb{V}'[\hat{f}]}{n\epsilon^2} \leq \frac{1}{n\epsilon^2} \leq \frac{1}{2}.$$

Logo,

$$\mathbb{P}'(|P(\hat{f}) - P'_n(\hat{f})| \leq \epsilon/2) \geq \frac{1}{2}.$$

Deste fato e da Equação 9.3, concluímos que

$$\mathbb{I}(|P_n(\hat{f}) - P(\hat{f})| > \epsilon) \leq 2\mathbb{P}'(|P_n(\hat{f}) - P'_n(\hat{f})| > \epsilon/2).$$

Tomando a esperança sobre $\mathbf{Z}_1, \dots, \mathbf{Z}_n$, concluímos que

$$\mathbb{P}(|P_n(\hat{f}) - P(\hat{f})| > \epsilon) \leq 2\mathbb{P}(|P_n(\hat{f}) - P'_n(\hat{f})| > \epsilon/2)$$

. Assim,

$$\mathbb{P} \left(\sup_{f \in \mathcal{F}} |P_n(f) - P(f)| > \epsilon \right) \leq 2\mathbb{P} \left(\sup_{f \in \mathcal{F}} |P_n(f) - P'_n(f)| > \epsilon/2 \right)$$

Demonstração. [Prova do Teorema VC] Seja

$$V = \mathcal{F}(\mathbf{z}_1, \dots, \mathbf{z}_n, \mathbf{z}'_1, \dots, \mathbf{z}'_n).$$

Para $v \in V$, seja

$$\mathcal{P}_n(v) - \mathcal{P}'_n(v) = \frac{1}{n} \left(\sum_{i=1}^n v_i - \sum_{i=n+1}^{2n} v_i \right).$$

Segue do Lema 9.1 que

$$\begin{aligned} \mathbb{P}\left(\sup_{f \in \mathcal{F}} |P_n(f) - P(f)| > \epsilon\right) &\leq 2\mathbb{P}\left(\sup_{f \in \mathcal{F}} |P_n(f) - P'_n(f)| > \epsilon/2\right) \\ &= 2\mathbb{P}\left(\sup_{v \in \mathcal{F}(\mathbf{Z}_1, \dots, \mathbf{Z}_n, \mathbf{Z}'_1, \dots, \mathbf{Z}'_n)} |\mathcal{P}_n(v) - \mathcal{P}'_n(v)| > \epsilon/2\right) \\ &\leq 2s(\mathcal{F}, 2n)\mathbb{P}(|\mathcal{P}_n(v) - \mathcal{P}'_n(v)| > \epsilon/2) \\ &\leq 4s(\mathcal{F}, 2n)e^{-n\epsilon^2/8}, \end{aligned}$$

em que a penúltima desigualdade segue da desigualdade da união e do fato que $\mathcal{F}_{\mathbf{Z}_1, \dots, \mathbf{Z}_n, \mathbf{Z}'_1, \dots, \mathbf{Z}'_n}$ possui no máximo $s(\mathcal{F}, 2n)$ elementos e a última desigualdade segue da desigualdade de Hoeffding.

Parte III

Aprendizado não supervisionado

Enquanto que em regressão e classificação disponhamos de covariáveis \mathbf{x} e respostas y , em aprendizado não supervisionado dispomos apenas de uma amostra com covariáveis $\mathbf{x}_1, \dots, \mathbf{x}_n$. Nossa objetivo é aprender algo sobre a estrutura destas covariáveis.

Capítulo 10

Redução de Dimensionalidade

Métodos de redução de dimensionalidade buscam encontrar transformações das covariáveis originais que capturam boa parte das informações presentes nelas, de modo a retirar redundâncias nas covariáveis e diminuir o número destas. Existem diversas razões para isso ser feito, como por exemplo facilitar a visualização dos dados e promover a criação de funções de predição com maior poder preditivo. Veja, por exemplo, a Figura 10.1, que apresenta um diagrama de dispersão de duas variáveis criadas a partir dos pixels das imagens de dígitos escritos a mão (como na Figura 7.1) utilizando a técnica de componentes principais. Cada ponto é representado pelo dígito correspondente. Note que 9's e 7's estão bastante sobrepostos, indicando uma maior similaridade entre si.

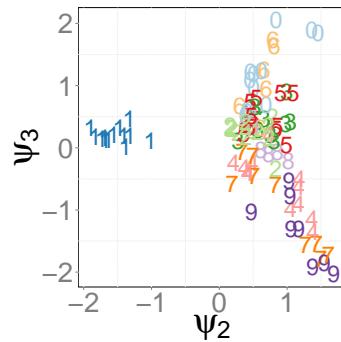


Figura 10.1: Diagrama de dispersão de duas variáveis criadas a partir dos pixels das imagens de dígitos escritos a mão (veja a Figura 7.1). Cada ponto é representado pelo dígito correspondente.

Assim, técnicas de redução de dimensionalidade visam criar um número “pequeno” de variáveis Z_1, Z_2, \dots a partir de X_1, X_2, \dots que resumam bem as informações presentes nelas. Neste capítulo estudamos diversas formas de se fazer tal redução.

10.1 Componentes Principais (PCA)

Por simplicidade, nesta seção iremos assumir que as covariáveis estão normalizadas para ter média zero.

Na técnica de componentes principais, a informação de uma variável Z_i é medida através de sua variabilidade. Mais especificamente, ela é medida via sua variância. Além disso, a redundância entre duas variáveis Z_i e Z_j é medida através de sua correlação. Finalmente, esta técnica se restringe a buscar por variáveis que sejam *combinações lineares* das covariáveis originais.

Formalmente, o primeiro componente principal de \mathbf{X} é a variável Z_1 que

1. é combinação linear das variáveis \mathbf{X} , isto é, pode ser escrito como

$$Z_1 = \phi_{11}X_1 + \dots + \phi_{d1}X_d \quad (10.1)$$

2. tem maior variância.

Assim, Z_1 é a variável que pode ser representada pela Equação 10.1, em que os coeficientes $\phi_{11}, \dots, \phi_{d1}$ são tais que Z_1 resultante possui a maior variabilidade possível. Adiciona-se também a restrição de que $\sum_{k=1}^d \phi_{k1}^2 = 1$, caso contrário os coeficientes ϕ_{k1} teriam valor absoluto tão grande quanto possível.

Da mesma maneira, o segundo componente principal de \mathbf{X} é a variável Z_2 que

1. é combinação linear das variáveis \mathbf{X} , isto é, pode ser escrito como

$$Z_2 = \phi_{12}X_1 + \dots + \phi_{d2}X_d$$

com a restrição: $\sum_{k=1}^d \phi_{k2}^2 = 1$

2. tem maior variância.
3. tem correlação zero com Z_1 .

Em outras palavras, o segundo componente também deve ser função linear das variáveis, deve ter a maior variância possível (para trazer o máximo de informação) e ter correlação zero com o componente já calculado (de modo que não haja informação redundante entre ambos).

De modo genérico, o i -ésimo componente principal de \mathbf{X} , $i > 1$, é a variável Z_i que

1. é combinação linear das variáveis \mathbf{X} , isto é, pode ser escrito como

$$Z_i = \phi_{1i}X_1 + \dots + \phi_{di}X_d$$

com a restrição: $\sum_{k=1}^d \phi_{ki}^2 = 1$

2. tem maior variância.
3. tem correlação zero com Z_1, \dots, Z_{i-1} .

Felizmente a solução para tal problema é bastante simples. Ela consiste em primeiramente encontrar autovetores de $C = \mathbf{X}^t\mathbf{X}$, que é matriz de variância-covariância das covariáveis, pois estamos assumindo que elas têm média zero. Seja U a matriz $d \times d$ em que a i -ésima coluna contém o i -ésimo autovetor de C . U é justamente a matriz de *cargas*, i.e., o seu elemento i, j é dado pelo coeficiente ótimo ϕ_{ij} . Assim, $Z = \mathbf{X}U$ é a matriz $n \times d$ com as d novas covariáveis para cada uma das n observações.

A Figura 10.2 apresenta uma ilustração dos dois primeiros componentes principais para um conjunto de dados artificial. Neste caso, o primeiro componente, z_1 , representa a posição de cada ponto com relação ao eixo marcado de vermelho, que é o eixo de maior variabilidade dos dados. Note que, sabendo o valor desta posição (ex: $z_1 = -1$), pode-se ter uma noção do valor tanto de x_1 quanto de x_2 . Assim, de fato o primeiro componente captura o que é mais importante sobre os dados. O segundo componente, z_2 é a posição de cada ponto com relação ao eixo azul. Tal informação complementa aquela dada pelo componente z_1 : com base em z_1 e z_2 é possível reconstruir x_1 e x_2 . Além disso, os eixos são ortogonais entre si. Isso sempre ocorre devido ao fato que se exige que a correlação entre Z_1 e Z_2 seja zero.

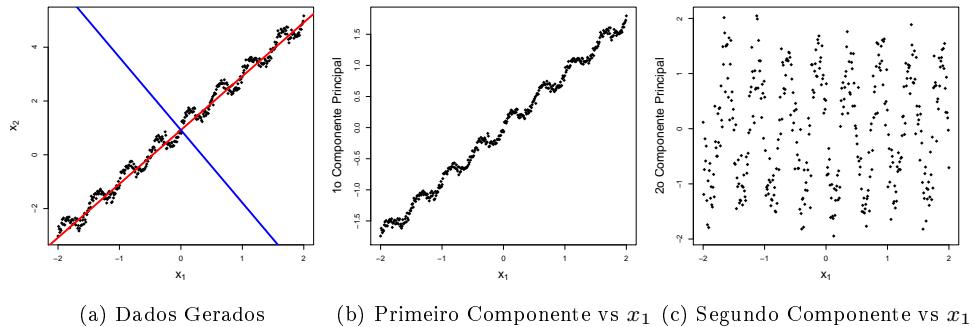


Figura 10.2: Componentes principais encontrados para um conjunto de dados artificial.

10.1.1 Interpretação alternativa: escalonamento multidimensional

Uma outra forma de motivar o uso dos componentes principais é dada pelo escalonamento multidimensional. Suponha que deseja-se encontrar as transformações *lineares* $\mathbf{Z} = (Z_1, \dots, Z_m)$, $m < d$ de $\mathbf{X} = (X_1, \dots, X_d)$ tal que

$$\sum_{i,j} (||\mathbf{x}_i - \mathbf{x}_j||^2 - ||\mathbf{z}_i - \mathbf{z}_j||^2)$$

tenha o menor valor possível. Ou seja, deseja-se buscar a transformação linear das variáveis originais tal que as distâncias euclidianas entre os pares de vetores originais, $||\mathbf{x}_i - \mathbf{x}_j||$ sejam o mais próximas possíveis das distâncias euclidianas entre os pares de novos vetores, $||\mathbf{z}_i - \mathbf{z}_j||$.

A solução para tal problema consiste justamente em tomar \mathbf{Z} como sendo o vetor dos m primeiros componentes principais de \mathbf{x} . □

Exemplo 10.1 Considere o conjunto de dados Zip Code¹, que contém a imagem de dígitos escritos à mão. A Figura 10.3 ilustra o diagrama de dispersão dos dois primeiros componentes principais, assim como a imagem de um dígito outlier (correspondente ao ponto circulado na primeira figura) e um dígito inlier.

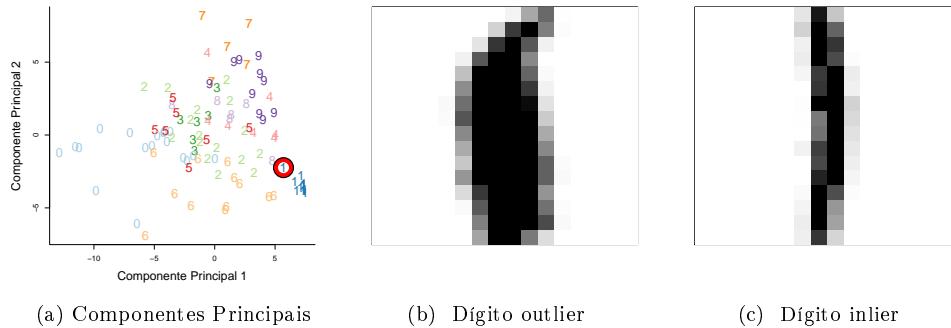


Figura 10.3: Dois primeiros componentes principais encontrados para o conjunto de dados de dígitos escritos à mão, juntamente com a imagem de um dígito outlier (correspondente ao ponto circulado na primeira figura) e um dígito inlier. □

¹ <http://statweb.stanford.edu/~tibs/ElemStatLearn/>.

10.1.2 Aplicação: compressão de imagens

Seja X a matriz $n \times d$ com os dados observados. Em PCA, fazemos a decomposição SVD da matriz de covariâncias:

$$X^t X = U \Sigma U^t,$$

em que U é uma matriz $d \times d$ cujas colunas contém os autovetores da matriz $X^t X$ (a chamada matriz de cargas) e Σ é uma matriz diagonal com seus autovalores. Ordena-se os autovetores de U de modo que seus respectivos autovalores sejam descrescentes.

A matriz $Z = XU$ (que é $n \times d$) contém os valores das d novas covariáveis avaliadas em cada uma das n observações. Além disso, os dados X podem ser reconstruídos calculando-se ZU^t . Note que a i -ésima linha de ZU^t é dada por

$$\sum_{k=1}^d z_{i,k} \mathbf{u}_k,$$

em que $z_{i,k} = \mathbf{u}_k^t \mathbf{x}_i$ é o valor da k -ésima nova variável para a i -ésima amostra e \mathbf{u}_k é a k -ésima coluna de U . É possível mostrar que

$$\sum_{k=1}^p z_{i,k} \mathbf{u}_k \approx \sum_{k=1}^d z_{i,k} \mathbf{u}_k$$

para $p < d$, contanto que os autovalores da matriz de covariâncias decaiam rapidamente para zero.

Isso sugere aproximar \mathbf{x}_i truncando $\sum_{k=1}^p c_{i,k} \mathbf{u}_k$ em um valor pequeno de p . A vantagem de tal aproximação é que ela reduz o espaço necessário para armazenar \mathbf{x}_i . Só é necessário guardar o valor das p novas covariáveis $(z_{i,1}, \dots, z_{i,p})$ e os pesos p primeiros pesos $\mathbf{u}_1, \dots, \mathbf{u}_p$. Matricialmente,

$$X \approx Z_p U_p^t,$$

em que Z_p contém as p primeiras colunas de Z e U_p contém as p primeiras colunas de U .

Nessa seção mostraremos como usar PCA para comprimir a seguinte imagem:



Figura 10.4: Figura que será comprimida via PCA

A ideia chave para fazer isso é tratar a imagem como sendo a coleção de três matrizes, uma para cada canal de cor (red/green/blue). Primeiramente vamos fazer a leitura da imagem no R.

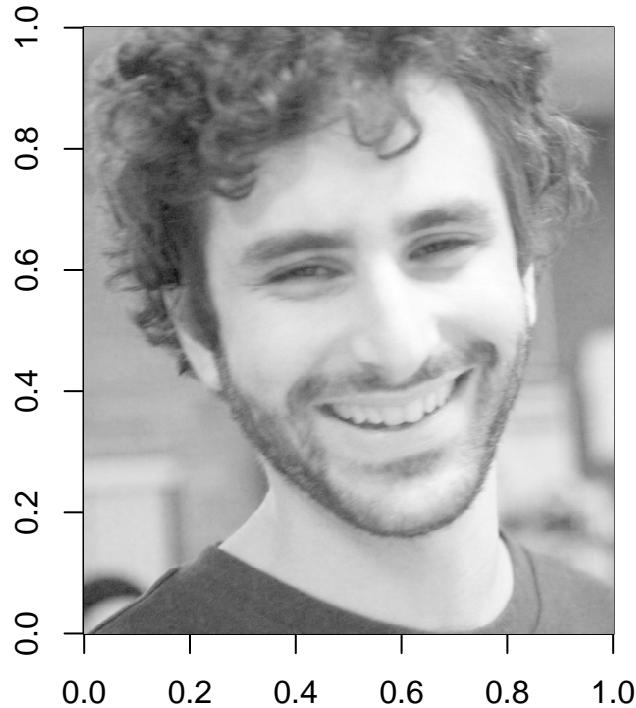
```
library(jpeg)
imagem <- readJPEG("Figures/izbicki.jpg", native = FALSE)

red    <- imagem[, , 1]
green <- imagem[, , 2]
blue   <- imagem[, , 3]

cat("Canal vermelho")

## Canal vermelho

rotate <- function(x) t(apply(x, 2, rev))
image(rotate(red), col=grey.colors(300, 0, 1))
```



Iremos agora fazer a decomposição SVD da matriz de cada canal separadamente.

```
r.svd <- eigen(t(red) %*% red)
g.svd <- eigen(t(green) %*% green)
b.svd <- eigen(t(blue) %*% blue)
```

Com a finalidade de visualizar as imagens, utilizaremos a seguinte função:

```
plot_jpeg = function(figure, add=FALSE, main="")
{
  figure[figure<0] <- 0
  figure[figure>1] <- 1
  res = dim(figure)[2:1] # get the resolution, [x, y]
```

```

cat(main)
if (!add) # initialize an empty plot area if add==FALSE
  plot(1,1,xlim=c(1,res[1]),ylim=c(1,res[2]),asp=1,
        type='n',xaxs='i',yaxs='i',
        xaxt='n',yaxt='n',xlab='',ylab='',bty='n')
  rasterImage(figure,1,1,res[1],res[2])
}

```

Finalmente, iremos aplicar PCA para comprimir as imagens. Com a finalidade de comparação, iremos utilizar diferentes níveis de compressão.

```

ncolunas <- c(3, 5, 25, 100, 200, 300) # níveis
# de compressão
for(i in 1:length(ncolunas)) {

  r.projecao <- red%*%r.svd$vectors[,1:ncolunas[i]]
  # (novas variáveis)
  r.approx <- r.projecao%*%t(r.svd$vectors[,1:ncolunas[i]])
  # (reconstrução)

  g.projecao <- green%*%g.svd$vectors[,1:ncolunas[i]]
  g.approx <- g.projecao%*%t(g.svd$vectors[,1:ncolunas[i]])

  b.projecao <- blue%*%b.svd$vectors[,1:ncolunas[i]]
  b.approx <- b.projecao%*%t(b.svd$vectors[,1:ncolunas[i]])

  imagemFinal <- array(
    NA,
    dim = c(nrow(red), ncol(red), 3))
  imagemFinal[, , 1] <- r.approx
  imagemFinal[, , 2] <- g.approx
  imagemFinal[, , 3] <- b.approx
  plot_jpeg(imagemFinal, main=
              paste0("Número de componentes =",
                     ncolunas[i], "\n",
                     "Economia de espaço = ",
                     paste0(round(100-(prod(dim(r.projecao))+
                     prod(dim(r.svd$vectors[,1:ncolunas[i]]))
                     prod(dim(red))*100,

```

```
    2) , " % " ) ) )  
}
```



(a) 3 componentes; 98,9% de compressão
(b) 5 componentes; 98,3% de compressão
(c) 25 componentes; 91,46% de compressão



(d) 100 componentes; 65,8% de compressão
(e) 200 componentes; 31,2% de compressão
(f) 300 componentes; -2,5% de compressão

Figura 10.5: Imagens recuperadas a partir dos componentes principais

10.2 Kernel PCA (KPCA)

Infelizmente, se restringir a transformações lineares das variáveis originais pode ser um fator muito limitante. Considere a Figura 10.6. Intuitivamente, o primeiro componente

principal deveria ser capaz de distinguir os grupos vermelho e preto. O primeiro componente, contudo, não é capaz de fazer isso, como mostrado na figura. Contudo, Kernel PCA, a técnica descrita nesta seção, é capaz de fazê-lo.

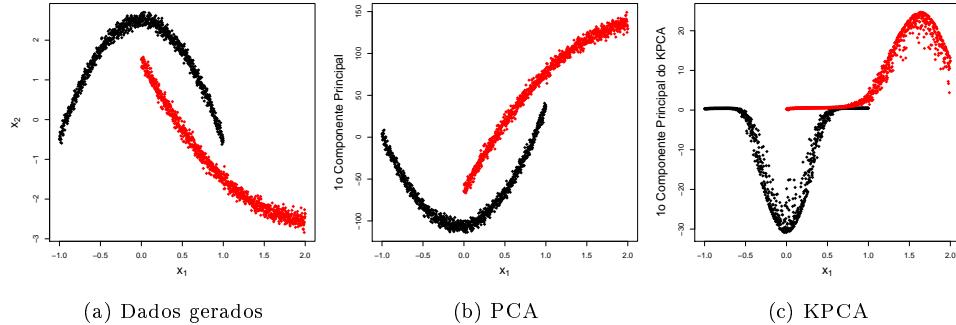


Figura 10.6: Primeiro componentes principais e primeiro componente do KPCA encontrados para um conjunto de dados artificial.

A ideia do Kernel PCA é utilizar o truque do kernel (já utilizado na Seção 4.6.3.2 no contexto de regressão) de forma a considerar transformações não lineares das covariáveis.

Lembre-se que a solução do PCA consiste em encontrar autovetores de $C = \mathbf{X}^t \mathbf{X}$ (matriz de covariâncias). O fato fundamental para usar o truque do kernel é que Z também pode ser calculada via o cálculo dos autovetores de

$$K = \mathbf{X} \mathbf{X}^t,$$

a matriz com os produtos internos entre cada par de observações. Note que esta matriz é diferente de C . Mais especificamente, tem-se que

$$Z = U_2^t,$$

em que U_2 é a matriz de autovetores de K . Assim, para calcular as componentes principais, basta saber os produtos internos entre as observações.

O truque do kernel na análise de componentes principais consiste em utilizar outros produtos internos (kernels) ao invés de $\langle \mathbf{x}_i, \mathbf{x}_j \rangle = \sum_{k=1}^d x_{i,k} x_{j,k}$. Ou seja, ao invés de calcular a autodecomposição de $K = \mathbf{X} \mathbf{X}^t$, usamos outros kernels,

$$K = \begin{bmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & K(\mathbf{x}_1, \mathbf{x}_2) & \cdots & K(\mathbf{x}_1, \mathbf{x}_n) \\ K(\mathbf{x}_2, \mathbf{x}_1) & K(\mathbf{x}_2, \mathbf{x}_2) & \cdots & K(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ K(\mathbf{x}_n, \mathbf{x}_1) & K(\mathbf{x}_n, \mathbf{x}_2) & \cdots & K(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix}$$

em que $K(\mathbf{x}_i, \mathbf{x}_j)$ corresponde ao produto interno de alguma transformação das covariáveis \mathbf{x} . Lembre-se que o truque consiste no fato de que a transformação não precisa ser calculada analiticamente (Seção 4.6.3.2).

No R, o KPCA (e, em particular, PCA) pode ser ajustado utilizando o pacote `kernlab`.

```
library(kernlab)
# kernel polinomial:
kpc=kpca(dados,kernel="polydot",
           kpar=list(degree=2),features=2)
# note que degree=1 corresponde a PCA usual

# ou kernel gaussiano: (sigma \'e um sobre a variância)
kpc=kpca(dados,kernel="rbfdot",
           kpar=list(sigma=1),features=2)

variaveisEmNovosDados <- predict(kpc,novosDados)
```

Observação 10.1 KPCA também pode ser motivada via escalonamento multidimensional: fazer KPCA equivale a fazer escalonamento multidimensional com base na distância $d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{2(1 - K(\mathbf{x}_i, \mathbf{x}_j))}$.

□

10.3 Principal Curves

10.4 t-SNE

10.5 Projeções Aleatórias

Uma estratégia alternativa para reduzir a dimensionalidade dos dados mas que frequentemente apresenta bons resultados é o método de projeções aleatórias. Nela, cada

componente z_i consiste em uma projeção linear das covariáveis originais com coeficientes escolhidos aleatoriamente.

Mais especificamente, seja S uma matriz com m linhas e d colunas cujas entradas são amostras i.i.d. de normais com média zero e variância um. Seja $s_{i,k}$ a entrada (i, j) desta matriz. Definimos a k -ésima projeção aleatória de \mathbf{x} como

$$z_i = \sum_{k=1}^d \frac{s_{i,k}}{\sqrt{m}} x_k.$$

Seja $\mathbf{z}_i = (z_{i,1}, \dots, z_{i,m})$ o vetor composto pelas novas variáveis. O teorema a seguir mostra que, com probabilidade alta, tal transformação preserva distâncias, i.e., $\|\mathbf{x}_i - \mathbf{x}_j\| \approx \|\mathbf{z}_i - \mathbf{z}_j\|$ para todos índices i e j .

Teorema 10.1 (Johnson-Lindenstrauss) Fixe $\epsilon > 0$ e seja $m \geq 32 \log n / \epsilon^2$. Então, com probabilidade ao menos $1 - e^{-m\epsilon^2/16}$, vale que

$$(1 - \epsilon) \|\mathbf{x}_i - \mathbf{x}_j\|^2 \leq \|\mathbf{z}_i - \mathbf{z}_j\|^2 \leq (1 + \epsilon) \|\mathbf{x}_i - \mathbf{x}_j\|^2$$

para todos $i, j = 1, \dots, n$.

Exemplo 10.2 Considere novamente o conjunto de dados Zip Code (Exemplo 10.1). A Figura 10.7 ilustra as duas (esquerda) e três (direita) primeiras projeções aleatórias calculadas. Note como imagens referentes aos mesmos dígitos tendem a se agrupar.

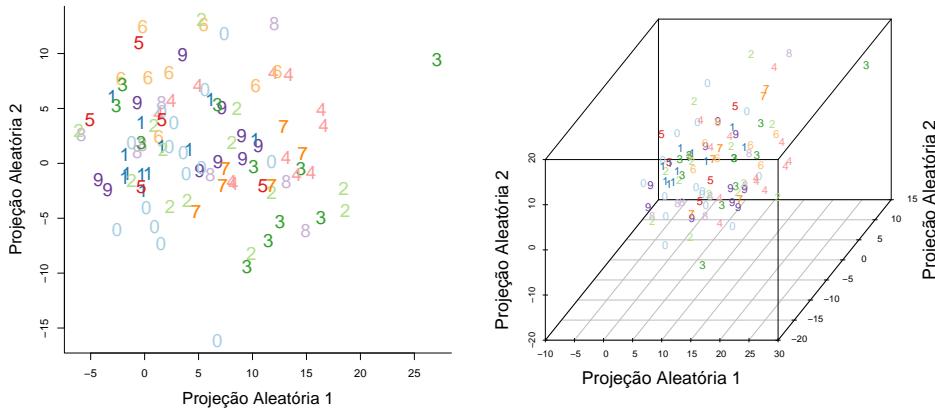


Figura 10.7: Duas (esquerda) e três (direita) primeiras projeções aleatórias calculadas no conjunto de dados de dígitos escritos à mão.

□

Além das aplicações de técnicas de redução de dimensionalidade já discutidas, projeções aleatórias também podem ser utilizadas como forma de deixar diversos algoritmos mais rápidos. Isso ocorre pois este método preserva distâncias (e produtos internos) e é muito rápido de ser calculado. Assim, métodos regressão e classificação que exigem apenas o cálculo do produto interno para serem implementados (como a regressão ridge e support vector machines) podem ser aproximados sem grandes perdas de performance estatística.

10.6 Autoencoders

10.7 Quantos componentes utilizar?

Uma pergunta fundamental é como escolher o número de componentes a se utilizar. Da mesma maneira, como devemos escolher tuning parameters associados ao kernel no caso de KPCA, ou mesmo como escolher qual kernel usar? Ao contrário do que ocorre com métodos de predição – em que busca-se escolher parâmetros que levam a um risco estimado baixo – em aprendizado não supervisionado esta é uma questão muito mais delicada e ainda não bem compreendida. Isso ocorre pois não há uma função de risco clara associada à tarefa de redução de dimensionalidade. Assim, dependendo da aplicação, diferentes critérios podem ser utilizados. Por exemplo, se estamos usando esses componentes como primeiro passo para classificação, podemos escolhê-los por validação cruzada. Se queremos visualizar os dados, podemos testar várias configurações, pois elas podem trazer diferentes insights.

Capítulo 11

Análise de Agrupamento

Métodos de análise de agrupamento (análise de cluster) têm por finalidade dividir a amostra em grupos de indivíduos. Essa divisão é feita de modo que os grupos sejam diferentes uns dos outros, mas indivíduos pertencentes ao mesmo grupo sejam parecidos entre si. Tais métodos podem ser utilizados com diversas finalidades. Por exemplo, uma loja de departamento pode utilizar análise de agrupamento para segmentar seus clientes. Assim, pode decidir como deve agir com cada grupo. Por exemplo, ela pode decidir mandar cartas com certas propagandas para um grupo, mas não para outro. Outro exemplo são as resenhas de usuários mostradas em sites de compras online quando um indivíduo busca por um produto. Em geral a empresa escolhe automaticamente duas ou três resenhas bastante distintas, mas que ao mesmo tempo são representativas das milhares de resenhas feitas sobre esse produto.

Formalmente, o objetivo de um método de clustering é criar uma partição C_1, \dots, C_K dos elementos amostrais $\{1, \dots, n\}$. Isto é, devemos ter ao mesmo tempo

$$C_1 \bigcup C_2 \bigcup \dots \bigcup C_K = \{1, 2, \dots, n\}$$

e

$$C_i \bigcap C_j = \emptyset \quad \forall i \neq j$$

Neste capítulo apresentamos diversos métodos de análise de agrupamento, que usam diferentes heurísticas para criar tais partições.

Um conceito essencial a diversos métodos de agrupamento é como medir dissimilaridade (ou similaridade) entre dois indivíduos com covariáveis \mathbf{x}_i e \mathbf{x}_j . Existem várias medidas possíveis. Um exemplo é a distância Euclidiana

$$d^2(\mathbf{x}_i, \mathbf{x}_j) = \sum_{k=1}^p (x_{i,k} - x_{j,k})^2.$$

Outra possibilidade é a distância de Mahalanobis. Para variáveis discretas, é comum criar variáveis “dummies”.

11.1 *K*-Médias

O método das *K*-Médias assume que a medida de dissimilaridade usada é a distância Euclíadiana. Para utilizá-lo, é necessário especificar de antemão o valor de *K*, quantos clusters se deseja. Neste algoritmo, buscar o melhor clustering é entendido como buscar pela partição C_1, \dots, C_K da nossa amostra tal que

$$\sum_{k=1}^K \frac{1}{|C_k|} \sum_{i,j \in C_k} d^2(\mathbf{x}_i, \mathbf{x}_j)$$

seja o menor possível. Esta quantidade é a soma de quadrados dentro de cada cluster.

O método para encontrar a resposta para esse problema é um algoritmo iterativo chamado Algoritmo de Lloyd. Não é garantido que a melhor solução seja de fato encontrada, apenas que mínimos locais sejam encontrados. Mesmo assim, ele frequentemente leva a bons resultados. O algoritmo consiste nos seguintes passos:

1. Escolha aleatoriamente k centroides c_1, \dots, c_k .

Itere até obter convergência:

2. **Atribuição:** Defina o cluster C_j ($j = 1, \dots, k$) como sendo

$$C_j = \{x_i : \arg \min_r d(x_i, c_r) = r\}$$

3. **Atualização:** Calcule os novos centroides usando os grupos que foram criados:

$$c_j \leftarrow \frac{1}{|C_j|} \sum_{j:x_j \in C_j} x_j$$

O algoritmo depende de escolhas iniciais e portanto o resultado pode ser um mínimo local dependendo da inicialização feita, veja o exemplo da Figura 11.1. Uma melhoria simples que pode ser utilizada é o chamado k-médias++. Tal algoritmo consiste em alterar a escolha dos pontos iniciais.

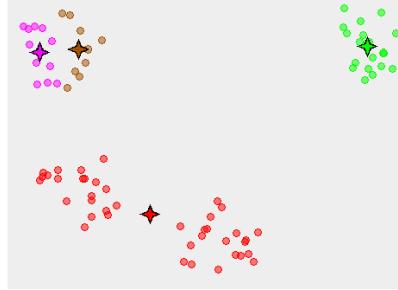


Figura 11.1: Exemplo em que o K -médias ficou preso em um mínimo local. Fonte: Wikipedia Commons.

1. Escolha c_1 aleatoriamente entre $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ e defina $C = \{c_1\}$.
2. Para $j = 2, \dots, k$:
 - a. Calcule $D(\mathbf{x}_i) = \min_{c \in C} \|\mathbf{x}_i - c\|$ para cada \mathbf{x}_i
 - b. Escolha uma amostra \mathbf{x}_i aleatoriamente entre todas as amostras observadas com probabilidade
$$p_i = \frac{D^2(\mathbf{x}_i)}{\sum_{j=1}^n D^2(\mathbf{x}_j)}$$
- c. Defina c_j como sendo o ponto escolhido. Atualize

$$C \leftarrow C \cup \{c_j\}$$

O kmédias++ prossegue então como no kmeans, mas desta vez com a escolha de pontos iniciais C fornecidas pelo algoritmo acima.

No R, K -médias pode ser aplicado com a função `kmeans`. Já o K -médias++ pode ser utilizado via a função `kmeanspp` do pacote `LICORS`.

11.2 Métodos Hierárquicos

Um problema do método K -médias é que K deve ser especificado de antemão. Métodos hierárquicos são uma alternativa ao K -médias que evitam isso. Tais métodos consistem na aplicação do seguinte algoritmo:

1. Atribua cada observação a um cluster diferente. Calcule cada uma das $\binom{n}{2}$ distâncias entre esses clusters.
2. Para $i = n, n-1, \dots, 2$:

- a. Procure entre todos os pares formados por dois dos i clusters aqueles mais parecidos. Junte esses dois clusters em um só. A dissimilaridade entre esses dois clusters indica a altura do dendrograma em que a junção será feita.
- b. Calcule cada uma das distâncias entre os novos $i - 1$ clusters.

A Figura 11.3 ilustra esse procedimento passo a passo para um conjunto de dados fictício.

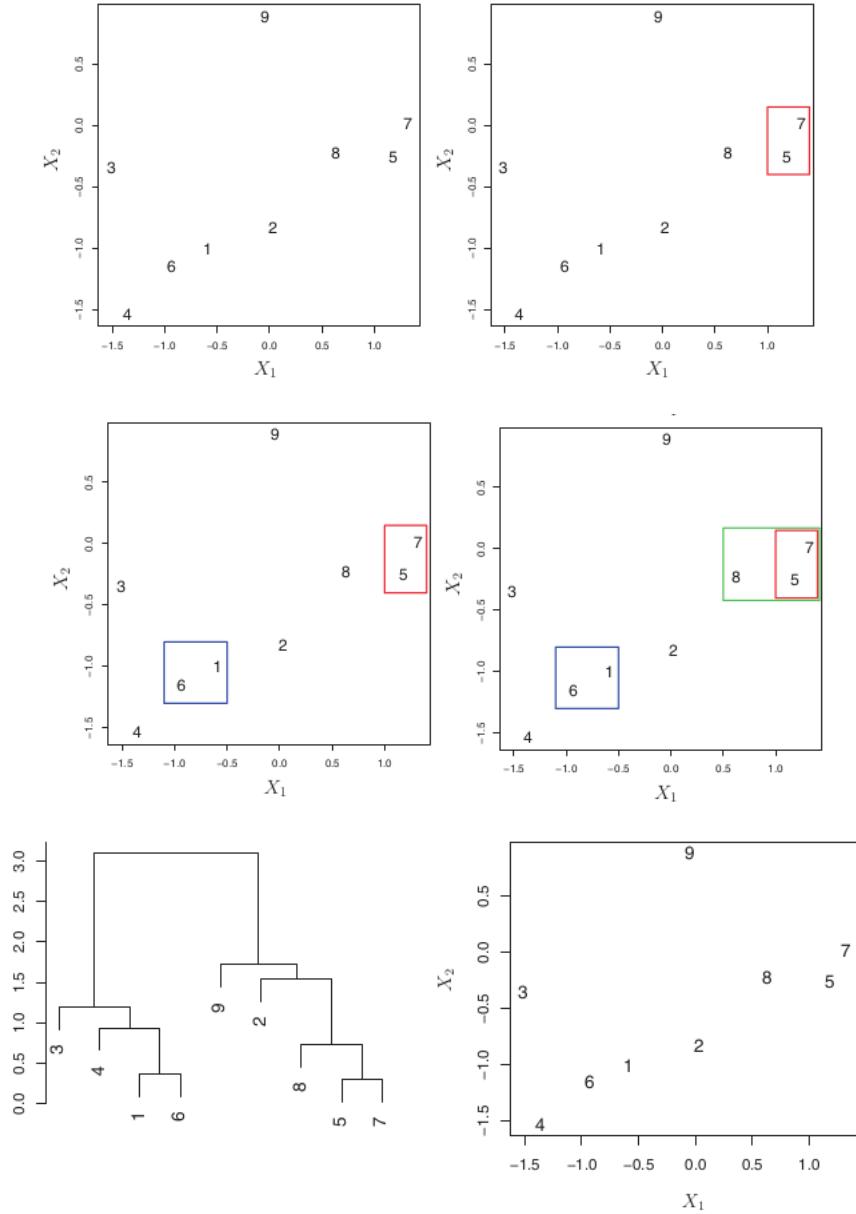


Figura 11.2: Clustering hierárquico.

Há várias formas de se definir a distância entre dois clusters. Estas são as chamadas *linkages*. Algumas formas são:

- **Complete:** A maior das distâncias entre todos os pares de observações pertencentes aos dois clusters.
- **Single:** A menor das distâncias entre todos os pares de observações pertencentes aos dois clusters.
- **Average:** A média das distâncias entre todos os pares de observações pertencentes aos dois clusters.
- **Centroid:** A distância entre os centroides dos dois clusters.

No R, clustering hierárquico pode ser ajustado com a função `hclust`.

11.3 Análise de Agrupamento Espectral

Nem sempre utilizar distâncias Euclidianas entre as variáveis originais produz bons resultados. Veja, por exemplo, a Figura 11.3. Uma forma de contornar isso é utilizar a análise espectral.

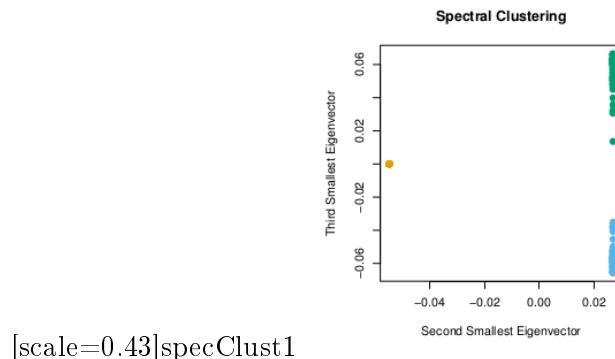


Figura 11.3: Clustering hierárquico. (Fonte: Elements of Statistical Learning) - Atualizar

Existem diversas variações deste método. Uma delas consiste em primeiramente utilizar algum método de redução de dimensionalidade não linear (como Kernel PCA, Seção 10.2) e então aplicar alguma das técnicas de clustering já vistas nas novas covariáveis.

11.4 Clustering com base em modas

[https://en.wikipedia.org/wiki/Silhouette_\(clustering\)](https://en.wikipedia.org/wiki/Silhouette_(clustering))

Parte IV

Sistemas de Recomendação

Capítulo 12

Regras de Associação

Imagine que temos um banco de dados em que cada linha representa a ida de uma pessoa a um supermercado e cada coluna representa se ela comprou ou não determinado produto. O objetivo de regras de associação (*market basket*) é descobrir regras do tipo:

- “Quem compra leite em geral também compra pão”,
- “Quem compra cerveja e refrigerante em geral também compra carne”.

que ocorram com alta frequência no banco de dados. Com base nestas regras, o mercado pode, por exemplo, decidir onde colocar produtos de forma a maximizar suas vendas. Outro exemplo do uso de regras de associação se dá no contexto de sites de vendas. Usando tais regras, esses sites podem decidir que produtos oferecer a um usuário com base em quais produtos ele já comprou.

Ainda que esta tarefa abordada por regras de associação pareça simples (a princípio basta investigar diversas tabelas de contingência), a chave do problema é como fazer isso eficientemente, uma vez que tais bancos de dados em geral possuem muitas covariáveis e, desta forma, o número de tabelas a serem investigadas é extremamente grande.

Nesta seção, vamos assumir que cada variável X_i do banco é binária. A fim de tornar o problema de regras de associação factível, nos restrinjimos a regras que envolvam subconjuntos S das d variáveis (ex: $S = \{2, 4, 10\}$) tais que

$$\mathbb{P}\left(\bigcap_{i \in S} X_i = 1\right) \quad (12.1)$$

é alto. Ou seja, nos restrinjimos a regras que envolvem combinações de produtos que são frequentemente comprados.

A probabilidade da Eq. 12.1 é chamada de suporte do conjunto de itens S e é estimada utilizando-se $\frac{1}{n} \sum_{k=1}^n \mathbb{I}(X_{k,i} = 1 \forall i \in S)$. Busca-se, portanto, entre todos os subconjuntos de itens S , aqueles que tenham suporte maior ou igual a um corte predefinido t .

Para tornar a busca por esses conjuntos eficiente, pode-se utilizar o algoritmo *Apriori*. Para tanto, tal algoritmo explora, entre outros, o fato de que se $S_0 \subseteq S_1$, então o suporte de S_1 é necessariamente menor ou igual ao suporte de S_0 . Assim, limita-se o número de subconjuntos a serem investigados.

Após encontrar todos os subconjuntos S com suporte alto, o algoritmo *Apriori* busca por regras do tipo

$$A \Rightarrow B,$$

em que A é chamado de antecedente, e B de consequente. Dado uma regra deste tipo, definem-se duas estatísticas que são úteis para descobrir regras de associação interessantes:

- Confiança: Uma estimativa de $P(B|A)$ (entre os usuários que compraram A , quantos compraram B ?)
- Lift/Levantamento: Uma estimativa de $\frac{P(B|A)}{P(B)}$ (O quanto o usuário ter comprado A aumenta a probabilidade dele comprar B)

Assim, buscamos, entre todas as regras que tem suporte maior que t prefixado, aquelas que tem levantamento ou confiança alta. Esta busca é factível, pois em geral há poucas regras com suporte alto.

Observação 12.1 Em geral utilizam-se matrizes esparsas para representar os dados binários utilizados, pois em geral há poucos uns nas matrizes envolvidas, e a representação esparsa então leva a uma grande economia de memória. Veja mais detalhes na Seção A.3.

□

No R, regras de associação podem ser implementadas utilizando-se o pacote arules.

```
library(arules)
library(arulesViz)

# Carregar o conjunto
data(Groceries)

# Encontrar Regras com suporte ao menos 0.001 e confiança 0.8:
# obs: maxlen=3: no máximo 3 variáveis do lado esquerdo
regras=apriori(Groceries,
                parameter = list(supp=0.005, conf=0.5,maxlen=3))

## Apriori
##
## Parameter specification:
```

```

##   confidence minval smax arem  aval originalSupport maxtime support minlen
##       0.5      0.1     1 none FALSE                      TRUE      5  0.005     1
##   maxlen target   ext
##       3   rules FALSE
##
## Algorithmic control:
##   filter tree heap memopt load sort verbose
##       0.1 TRUE TRUE FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 49
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [120 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 done [0.00s].
## writing ... [99 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
options(digits=2) # algarismos significativos

# Mostrar 5 regras com maior suporte
inspect(regras[1:5])

##   lhs                                rhs          support
## [1] {baking powder}                  => {whole milk} 0.0093
## [2] {other vegetables,oil}          => {whole milk} 0.0051
## [3] {root vegetables,onions}        => {other vegetables} 0.0057
## [4] {onions,whole milk}             => {other vegetables} 0.0066
## [5] {other vegetables,hygiene articles} => {whole milk} 0.0052
##   confidence lift count
## [1] 0.52      2.0  91
## [2] 0.51      2.0  50
## [3] 0.60      3.1  56
## [4] 0.55      2.8  65
## [5] 0.54      2.1  51

# Reordenar por confiança:
regras=sort(regras, by="confidence", decreasing=TRUE)
inspect(regras[1:5])

```

```

##      lhs                      rhs          support  confidence lift
## [1] {butter,whipped/sour cream} => {whole milk} 0.0067    0.66    2.6
## [2] {pip fruit,whipped/sour cream} => {whole milk} 0.0060    0.65    2.5
## [3] {butter,yogurt}                  => {whole milk} 0.0094    0.64    2.5
## [4] {root vegetables,butter}        => {whole milk} 0.0082    0.64    2.5
## [5] {tropical fruit,curd}          => {whole milk} 0.0065    0.63    2.5
##      count
## [1] 66
## [2] 59
## [3] 92
## [4] 81
## [5] 64

# Reordenar por lift:
regras=sort(regras, by="lift", decreasing=TRUE)
inspect(regras[1:5])

##      lhs                      rhs          support  confidence lift  count
## [1] {tropical fruit,
##       curd}                  => {yogurt}      0.0053    0.51    3.7   52
## [2] {pip fruit,
##       whipped/sour cream} => {other vegetables} 0.0056    0.60    3.1   55
## [3] {root vegetables,
##       onions}                 => {other vegetables} 0.0057    0.60    3.1   56
## [4] {citrus fruit,
##       root vegetables}     => {other vegetables} 0.0104    0.59    3.0   102
## [5] {tropical fruit,
##       root vegetables}     => {other vegetables} 0.0123    0.58    3.0   121

# buscar regras que envolvam cerveja no lado direito (rhs):
regras=apriori(data=Groceries,
                parameter=list(supp=0.001,
                               conf = 0.08),
                appearance =
                  list(default="lhs",
                        rhs="bottled beer"),
                control = list(verbose=F))
regras=sort(regras, by="lift", decreasing=TRUE)
inspect(regras[1:10])

##      lhs                      rhs          support  confidence lift  count

```

```

## [1] {liquor,
##       red/brush wine} => {bottled beer} 0.0019      0.90 11.2   19
## [2] {soda,
##       liquor}          => {bottled beer} 0.0012      0.57  7.1   12
## [3] {liquor}          => {bottled beer} 0.0047      0.42  5.2   46
## [4] {herbs,
##       bottled water}  => {bottled beer} 0.0012      0.40  5.0   12
## [5] {whole milk,
##       soups}           => {bottled beer} 0.0011      0.38  4.7   11
## [6] {soda,
##       red/brush wine} => {bottled beer} 0.0016      0.36  4.4   16
## [7] {other vegetables,
##       red/brush wine} => {bottled beer} 0.0015      0.31  3.8   15
## [8] {other vegetables,
##       domestic eggs,
##       bottled water}   => {bottled beer} 0.0012      0.30  3.7   12
## [9] {oil,
##       bottled water}   => {bottled beer} 0.0012      0.29  3.6   12
## [10] {tea}             => {bottled beer} 0.0011     0.29  3.6   11

# buscar regras que envolvam cerveja no lado esquerdo (lhs):
regras=apriori(data=Groceries,
                 parameter=list(supp=0.001,conf = 0.15,minlen=2),
                 appearance = list(default="rhs",
                                    lhs="bottled beer"),
                 control = list(verbose=F))
inspect(regras)

##      lhs                  rhs          support  confidence lift count
## [1] {bottled beer} => {bottled water} 0.016      0.20      1.77 155
## [2] {bottled beer} => {soda}          0.017      0.21      1.21 167
## [3] {bottled beer} => {rolls/buns}  0.014      0.17      0.92 134
## [4] {bottled beer} => {other vegetables} 0.016      0.20      1.04 159
## [5] {bottled beer} => {whole milk}    0.020      0.25      0.99 201

```

Para visualizar interativamente as regras, pode-se utilizar

```

regras=apriori(Groceries,
                parameter = list(supp = 0.001, conf = 0.8))
plot(regras[1:20],method="graph",interactive=TRUE,shading=NA)

```


Capítulo 13

Sistemas de Recomendação

Vimos, no Capítulo 12, que uma maneira de indicar um produto a um usuário com base em uma lista de itens que cada indivíduo comprou de uma empresa é através de regras de associação. Neste capítulo, vamos estudar algumas técnicas de recomendação que funcionam em contextos mais gerais. Vamos assumir que cada usuário pode atribuir uma nota (*avaliação*) para cada produto. Por exemplo, no Netflix, podemos avaliar cada filme como 1, 2, 3, 4 ou 5 estrelas.

De um ponto de vista formal, considere que temos um conjunto de usuários

$$\mathcal{U} = \{u_1, \dots, u_m\}$$

e um conjunto de produtos (*itens*)

$$\mathcal{I} = \{i_1, \dots, i_n\}.$$

Seja $R_{j,k}$ a avaliação dada pelo usuário j ao produto k , como mostrado na Tabela 13.1.

Tabela 13.1: Estrutura dos dados envolvidos em um problema sistemas de recomendação.

Usuário/Produto	i_1	i_2	\dots	i_n
u_1	$R_{1,1}$	$R_{1,2}$	\dots	$R_{1,n}$
u_2	$R_{2,1}$	$R_{2,2}$	\dots	$R_{2,n}$
\vdots	\vdots	\vdots	\vdots	\vdots
u_m	$R_{m,1}$	$R_{m,2}$	\dots	$R_{m,n}$

Tipicamente, várias das entradas desta matriz são desconhecidas, pois vários usuários nunca atribuíram notas para vários dos produtos. O problema central desta seção consiste em como imputar essas notas do modo a descobrir, para cada usuário, produtos não avaliados por ele que provavelmente o agradarão.

Existem dois principais grupos de sistemas de recomendação:

- **Sistemas baseados no conteúdo.** Estes sistemas baseiam-se em características dos produtos que o usuário gosta (ex: diretores dos filmes, gêneros etc) e, com base neles, buscam descobrir outros produtos com as mesmas características.
- **Sistemas baseados em filtros colaborativos.** Estes sistemas fornecem recomendações com base apenas na matriz de notas. Utilizando essa matriz, tais métodos buscam por padrões (ex: usuários parecidos com o usuário de interesse). O pressuposto básico por trás de filtros colaborativos é que usuários que concordaram no passado irão concordar no futuro.

Sistemas baseados no conteúdo em geral são obtidos através de métodos de regressão já estudados anteriormente. Assim, nos focaremos aqui em filtros colaborativos.

Observação 13.1 (Normalização das notas) *É comum se observar um viés nas notas. Alguns usuários tendem a dar notas muito altas para todos os produtos; outros tendem a dar notas muito baixas. Para tentar levar isso em conta, é comum renormalizar as notas de cada usuário antes de aplicar os métodos descritos neste capítulo. Uma forma de se fazer isso é trabalhar com as avaliações dadas por*

$$\tilde{R}_{j,k} = R_{j,k} - \bar{R}_j,$$

em que \bar{R}_j é a média das notas dadas pelo usuário j .

□

Nas Seções 13.1-13.3 apresentamos alguns filtros colaborativos comumente usados, enquanto que na Seção 13.4 discutimos como avaliar a performance de tais métodos e fazer seleção de modelos.

13.1 Filtro colaborativo baseado no usuário

Filtros colaborativos baseados no usuário funcionam de forma parecida com o método KNN (Seções 4.3 e 8.6). Inicialmente, define-se uma medida de similaridade entre os usuários, que avalia a similaridade entre as notas por eles atribuídas. Uma medida usual é a correlação de Pearson:

$$\text{sim}(u_a, u_b) = \text{cor}(R_{u_a, \cdot}^*, R_{u_b, \cdot}^*),$$

em que $R_{u_a, \cdot}^*$ é o vetor de notas atribuídas pelo usuário u_a aos produtos avaliados por ambos u_a e u_b .

Seja $k \geq 1$ fixo. Para estimar $R_{j,l}$, a nota que o usuário j daria para o produto l caso o avaliasse, buscam-se os k usuários mais parecidos a j (i.e., os usuários com maior similaridade com j) que avaliaram o produto l . Calcula-se então as médias das notas dadas por esses k usuários ao produto l

$$\hat{R}_{j,l} = \frac{1}{k} \sum_{s \in \mathcal{V}_j} R_{s,l},$$

em que \mathcal{V}_j é o conjunto dos k usuários mais similares ao usuário j .

O valor de k pode ser escolhido utilizando as técnicas abordadas na Seção 13.4.

13.2 Filtro colaborativo baseado no produto

Filtros colaborativos baseados no produto são muito parecidos com filtros colaborativos baseados no usuário (Seção 13.1). A diferença básica entre ambos é que, para se estimar $R_{j,l}$, a nota que o usuário j daria para o produto l caso o avaliasse, buscam-se agora os k produtos mais parecidos ao item l , e não os k usuários mais parecidos com j .

Para medir a similaridade entre as notas atribuídas por cada um dos usuários ao produto l com as notas atribuídas por cada um dos usuários aos outros produtos, pode-se novamente usar a correlação de Pearson, desta vez definida como:

$$\text{sim}(i_a, i_b) = \text{cor}(R_{\cdot, i_a}^*, R_{\cdot, i_b}^*),$$

em que R_{\cdot, i_a}^* é o vetor de notas atribuídas para o produto i_a por cada um dos usuários que avaliaram i_a e i_b .

A predição para $R_{j,l}$ é então dada pelas médias das notas dos k produtos mais parecidos a l :

$$\hat{R}_{j,l} = \frac{1}{k} \sum_{s \in \mathcal{V}_l} R_{j,s},$$

em que \mathcal{V}_l é o conjunto dos k itens mais próximos ao produto l .

O valor de k pode ser escolhido utilizando as técnicas abordadas na Seção 13.4.

13.3 FunkSVD

13.4 Seleção de Modelos

Para comparar modelos, utilizamos uma variação de data splitting (Seção 1.4.1). Essa técnica permite também escolher parâmetros dos modelos usados (ex: k no filtro colaborativo). Primeiramente, separamos os usuários em dois conjuntos: treinamento e validação. Fingimos então que desconhecemos algumas das notas dadas por alguns dos usuários do conjunto de validação e verificamos a capacidade dos algoritmos usados de preverem essas notas.

Seja \mathcal{K} o conjunto de todos os pares usuário/produto para o qual a nota real $R_{i,j}$ é conhecida, mas não foi usada para treinar o modelo. Diversas medidas da capacidade preditiva podem ser usadas, dentre as quais

- EQM: $\frac{\sum_{(i,j) \in \mathcal{K}} (R_{i,j} - \hat{R}_{i,j})^2}{|\mathcal{K}|}$
- RMSE: \sqrt{EQM}
- MAE: $\frac{\sum_{(i,j) \in \mathcal{K}} |R_{i,j} - \hat{R}_{i,j}|}{|\mathcal{K}|}$

Assim, estas medidas avaliam se as notas estão bem preditas e podem ser utilizadas para fazer seleção de modelos. Contudo, podemos ter interesse apenas em medir se o método está fornecendo boas recomendações, e não se as notas individuais estão preditas de forma satisfatória. Para tanto, outras medidas podem ser utilizadas. Assuma que recomendamos para um dado usuário os N produtos com maior valor predito para a avaliação. Assuma também que definimos o que é uma avaliação boa e o que é uma avaliação ruim (e.g., uma avaliação boa é uma nota maior que 3).

Com base nas N recomendações feitas para cada usuário do conjunto de validação (i.e., os N produtos com maior nota predita), podemos montar a matriz de confusão apresentada na Tabela 13.2.

Tabela 13.2: Matriz de confusão para produtos recomendados.

		Valor Predito	
Valor verdadeiro	Ruim	Bom	
	Ruim	a	b
Bom	c	d	

Com base nesta tabela, diversas medidas podem ser utilizadas para fazer seleção de modelos. Entre elas destacamos:

- Acurácia: $\frac{a+d}{a+b+c+d}$
- Precisão: $\frac{d}{b+d}$
- Lembrança: $\frac{d}{c+d}$

Outras medidas como sensibilidade, especificidade já definidas na Seção 7.4 no contexto de classificação podem também ser utilizadas.

No R, o pacote `recommenderlab` permite a implementação e comparação de sistemas de recomendação.

```
library("recommenderlab", verbose = FALSE)

# dados de filmes
data(MovieLens)

# listar algoritmos que serão usados,
# dando um nome para cada um deles
metodos <- list(
  "item-based CF 1" = list(name = "IBCF", param = list(k = 5)),
  "user-based CF 1" = list(name = "UBCF", param = list(nn = 5)),
  "item-based CF 2" = list(name = "IBCF", param = list(k = 10)),
  "user-based CF 2" = list(name = "UBCF", param = list(nn = 10)) )

# definir o esquema do data splitting
esquema <- evaluationScheme(MovieLens,
                             method = "split",
                             train = 0.8,
                             given = 15)
# given: quantas avaliações são consideradas dadas
# de cada usuário do conjunto de teste

# parte 1: avaliar o quanto bom são as notas atribuídas
results = evaluate(esquema, metodos, type = "ratings")

## IBCF run fold/sample [model time/prediction time]
##    1 [10.305sec/0.195sec]
## UBCF run fold/sample [model time/prediction time]
```

```

##    1 [0.006sec/0.775sec]
## IBCF run fold/sample [model time/prediction time]
##    1 [10.637sec/0.113sec]
## UBCF run fold/sample [model time/prediction time]
##    1 [0.041sec/0.828sec]

results$item-based CF 1`@results

## [[1]]
## An object of class "confusionMatrix"
## Slot "cm":
##       RMSE      MSE      MAE
## res 1.543033 2.380952 1.174603
##
## Slot "model":
## NULL

results$item-based CF 2`@results

## [[1]]
## An object of class "confusionMatrix"
## Slot "cm":
##       RMSE      MSE      MAE
## res 1.492228 2.226744 1.110465
##
## Slot "model":
## NULL

# parte 2: avaliar o quao bom sao as recomendacoes top-N
esquema <- evaluationScheme(MovieLense,
                           method="split",
                           train=0.8,
                           given=10,
                           goodRating=3)
# goodRating: a partir de que nota consideramos que temos uma boa predicao?

results=evaluate(esquema, metodos, n=c(1, 5, 10))

## IBCF run fold/sample [model time/prediction time]
##    1 [10.417sec/0.048sec]
## UBCF run fold/sample [model time/prediction time]

```

```
##    1 [0.006sec/1.036sec]
## IBCF run fold/sample [model time/prediction time]
##    1 [9.774sec/0.067sec]
## UBCF run fold/sample [model time/prediction time]
##    1 [0.006sec/1.368sec]

# n: quantos produtos estamos recomendando

results$item-based CF 1`@results

## [[1]]
## An object of class "confusionMatrix"
## Slot "cm":
##           TP        FP        FN        TN  precision      recall
## 1  0.01587302 0.978836 75.70370 1577.302 0.01595745 6.810746e-05
## 5  0.07407407 4.783069 75.64550 1573.497 0.01489362 5.131112e-04
## 10 0.16931217 8.793651 75.55026 1569.487 0.01764184 1.328814e-03
##           TPR        FPR
## 1  6.810746e-05 0.0006205803
## 5  5.131112e-04 0.0030376748
## 10 1.328814e-03 0.0055831050
##
## Slot "model":
## NULL

plot(results, legend = "right")
```

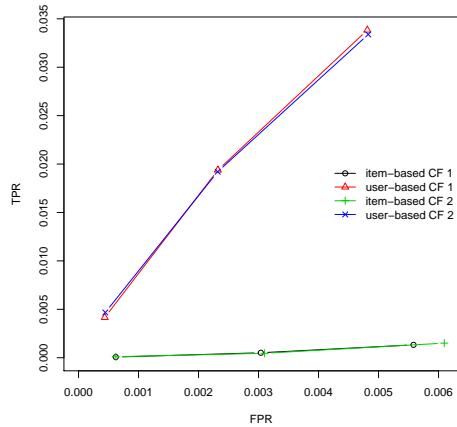


Figura 13.1: Valor verdadeiro positivo vs valor verdadeiro negativo para diferentes métodos e $N=1,5$ e 10.

Parte V

Apêndice

Apêndice A

Apêndice

A.1 Imagens

Este apêndice descreve brevemente como manipular imagens que tem formato do tipo *raster* (ex: JPEG, PNG, ...). Um formato é dito do tipo *raster* quando ele representa uma dada imagem usando uma ou mais matrizes que contém informações sobre os pixels da figura. Para entender esse conceito, vamos começar com uma ideia simples: considere a matriz binária

$$M = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

A ideia chave é que podemos associar a essa matriz a imagem



Figura A.1: Imagem correspondente à matriz M .

Aqui, uma entrada "1" na matriz simboliza que o pixel correspondente a tal elemento na imagem tem a cor preto, enquanto que o valor de 0 simboliza um pixel de cor branca. Usando essa ideia, podemos criar imagens como

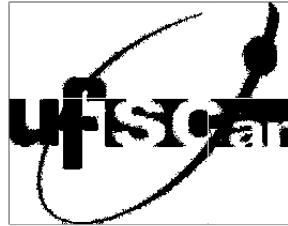


Figura A.2: Imagem feita utilizando-se apenas uma matriz binária.

Note que quanto mais pixels em uma imagem (i.e., quanto mais entradas há em uma matriz), maior a resolução da imagem.

Podemos ir um passo além nesta ideia. Ao invés de usar apenas 0 (branco) e 1 (preto), podemos usar qualquer número entre 0 e 1 para denotar uma intensidade de cinza. Com isso, podemos fazer imagens como



Figura A.3: Imagem feita utilizando-se uma matriz cujas entradas representam o tom de cinza a ser utilizado.

Utilizando-se a mesma ideia, podemos também construir imagens coloridas. Lembre-se que usando cores primárias podemos compor qualquer cor. Usando essa ideia, podemos representar uma imagem com três matrizes simultaneamente:

- A primeira indica quanto azul vamos ter em cada pixel (cada elemento é um número entre 0 e 1)

- A segunda indica quanto amarelo vamos ter em cada pixel (cada elemento é um número entre 0 e 1)
- A terceira indica quanto vermelho vamos ter em cada pixel (cada elemento é um número entre 0 e 1)

Com isso, podemos fazer imagens como



Figura A.4: Imagem feita utilizando-se três matrizes cujas entradas representam a “quantidade” de cada cor primária utilizada em cada pixel. Veja versão online para a imagem colorida.

Esse é o princípio usado por exemplo no formato JPEG. Mas, ao invés de usar cores primárias, são usados os RGB channels (vermelho, verde e azul). Cada matriz é chamada de um *canal*.

Note que utilizar o valor 0 para branco e 1 preto é apenas uma convenção. Formatos diferentes usam convenções diferentes (por exemplo, alguns formatos atribuem 0 a branco e 256 a preto). Em vários dos formatos também é feita uma compressão da imagem de modo que ela ocupe menos espaço. Neste caso, leitores destes formatos devem saber também como descomprimi-las (o que muitas vezes leva a uma pequena perda de resolução).

A.1.1 Lendo Imagens no R

```
## Exemplo artificial:
m=matrix(c(1,1,0,1,1,0,0,0,0),3,3)
image(m[,3:1],col = c("white","black"))

## Exemplo do simbolo da ufscar:
```

```

library(jpeg)
imagem=readJPEG("1024px-UFSCar.jpg") # ler a imagem
dim(imagem)
#[1] 746 1024    3 # 3 matrizes com 746*1024 pixels
image(t(imagem[746:1,,3]), col = grey.colors(1000, start = 0, end = 1))
# imagem em tons de cinza (utilizando apenas a terceira matriz)
rasterImage(imagem, 0, 0, 1, 1) # imagem colorida

```

A.2 Textos

Primeiramente, introduziremos algumas definições úteis para mineração de texto:

- *Termo*: uma sequência de letras de um determinado alfabeto, isto é, para nós, termo e palavra são elementos similares;
- *Documento*: um conjunto de termos ordenados, por exemplo, um texto;
- *Coleção de documentos*: um conjunto formado por diferentes documentos;
- *Dicionário*: um conjunto formado pelos diferentes termos presentes em uma coleção de documentos.

A representação Modelo Espaço Vetorial (Vector Space Model - VSM) para documentos foi introduzida por [Salton et al. \(1975\)](#). A ideia principal é representar um documento como um vetor, em particular uma *bag of words* em que a ordem dos elementos não importa ([Srivastava and Sahami, 2009](#)). Considere uma coleção de documentos \mathcal{C} e seja \mathcal{S} um conjunto contendo todos os diferentes termos de \mathcal{C} , isto é, \mathcal{S} é um dicionário sobre \mathcal{C} . Definimos:

- d_i o i -ésimo documento em \mathcal{C} ;
- $n = \#(\mathcal{C})$;
- t_j o j -ésimo termo em \mathcal{S} ;
- $s = \#(\mathcal{S})$;
- $f : \mathcal{C} \times \mathcal{S} \rightarrow \mathbb{R}$ uma função que relaciona o i -ésimo termo com o j -ésimo documento.

Basicamente, queremos encontrar uma função f , tal que, a representação

$$\begin{aligned}\phi : \mathcal{C} &\longrightarrow \mathbb{R}^s \\ d_i &\longrightarrow \phi(d_i) = (f(d_i, t_1), f(d_i, t_2), \dots, f(d_i, t_s))\end{aligned}$$

represente bem o documento, isto é, preserve bem suas informações. Duas formas simples de definir a representação são apresentadas a seguir:

- $f(d_i, t_j) = \mathbb{1}_{d_i}(t_j)$, dessa forma, $\phi(d_i) = (\mathbb{1}_{d_i}(t_1), \mathbb{1}_{d_i}(t_2), \dots, \mathbb{1}_{d_i}(t_s))$, em que $\mathbb{1}_{d_i}(t_j)$ é a função indicadora de t_j no conjunto d_i . Em palavras, esta representação indica quais termos estão em um determinado documento.
- $f(d_i, t_j) = F_{ji}$, assim, $\phi(d_i) = (F_{1i}, F_{2i}, \dots, F_{si})$, em que F_{ji} é a frequência absoluta do j -ésimo termo no i -ésimo documento. Aqui, nossa *bag of words* conta o número de vezes em que um termo aparece em um determinado documento.

A partir destas funções, podemos construir uma matriz *documento-termo* \mathbf{D} , em que a i -ésima linha de \mathbf{D} é a representação $\phi(d_i)$. Isto é

$$\mathbf{D} = \begin{pmatrix} f(t_1, d_1) & \cdots & f(t_s, d_1) \\ \vdots & \ddots & \vdots \\ f(t_1, d_n) & \cdots & f(t_s, d_n) \end{pmatrix}.$$

A matriz \mathbf{D} pode ser usada como a matriz desenho (i.e., cada linha é uma amostra, e cada coluna uma covariável) para aplicação dos métodos estudados neste livro.

A.3 Representação de Matrizes Esparsas

Diversos dos métodos estudados neste livro requerem o armazenamento de matrizes. Infelizmente, armazenar tais matrizes em um computador pode ser uma tarefa desafiadora se essa tem uma dimensão muito grande, uma vez que é necessária muita memória para que isso seja feito. Contudo, algumas matrizes possuem algumas características que facilitam tal armazenamento. Nesta seção estudamos uma destas propriedades: *esparsidade*.

Uma matriz é dita ser esparsa quando ela tem quase todas as suas entradas com o valor zero, veja por exemplo a matriz da Equação A.1. Tais matrizes aparecem em muitas aplicações: por exemplo, no método bag-of-words (Seção A.2), esperamos que cada texto em geral *não* contenha a maior parte das palavras. Assim, quase todas entradas da matriz documento-termo serão zero.

$$\begin{bmatrix} 3 & 6 & 0 & 0 & \dots & 0 & 3 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 5 & 0 & 0 & \dots & 0 & 0 \end{bmatrix}_{3 \times 1000} \quad (\text{A.1})$$

Uma ideia simples para representar a informação presente nesta matriz de modo a reduzir a memória necessária é armazenar somente as posições (número da linha e coluna)

de cada elemento diferente de zero, juntamente com o valor observado. Assim, a matriz da Equação A.1 pode ser representada como

$$\begin{aligned} & (1, 1, 3) \\ & (1, 2, 6) \\ & (1, 1000, 3) \\ & (3, 2, 5) \end{aligned}$$

Assim, ao invés de termos que armazenar $3 \times 1000 = 3000$ números, precisamos armazenar apenas $3 \times 4 = 12$ valores.

A.3.1 Word2vec

Referências

- N. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68(3):337–404, 1950. [57](#)
- Anil Aswani, Peter Bickel, and Claire Tomlin. Regression on manifolds: Estimation of the exterior derivative. *The Annals of Statistics*, pages 48–81, 2011. [96](#)
- R. E. Bellman. *Adaptive control processes - A guided tour*. Princeton University Press, 1961. [93](#)
- J. K. Benedetti. On the nonparametric estimation of regression functions. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 248–253, 1977. [52](#)
- Y. Bengio, O. Delalleau, N. Le Roux, J. F. Paiement, P. Vincent, and M. Ouimet. Learning eigenfunctions links Spectral Embedding and Kernel PCA. *Neural Computation*, 16(10):2197–2219, 2004. [98](#), [99](#)
- R. Beran. REACT scatterplot smoothers: Superefficiency through basis economy. *Journal of the American Statistical Association*, 95(449):155–171, 2000. [51](#)
- P. J. Bickel and B. Li. Local polynomial regression on unknown manifolds. In *IMS Lecture Notes-Monograph Series, Complex Datasets and Inverse Problems*, volume 54, pages 177–186. Institute of Mathematical Statistics, 2007. [96](#)
- C. M. Bishop. *Pattern recognition and machine learning*, volume 1. Springer New York, 2006. [37](#)
- L. Breiman. Statistical modeling: The two cultures. *Statistical Science*, 16(3):199–231, 2001. [8](#)
- N. Cesa-Bianchi and G. Lugosi. *Prediction, learning, and games*. Cambridge University Press, 2006. [106](#)

- N. Chentsov. Evaluation of an unknown distribution density from observations. *DOKLADY*, 147(1):1559–1562, 1962. [49](#)
- W. S. Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the American statistical association*, 74(368):829–836, 1979. [56](#)
- C. Cortes and V. Vapnik. Support vector machine. *Machine learning*, 20(3):273–297, 1995. [128](#)
- C. De Boor. *A practical guide to splines*, volume 27. Springer-Verlag New York, 1978. [52](#)
- P. Drineas and M. W. Mahoney. On the Nyström Method for approximating a Gram matrix for improved kernel-based learning. *Journal of Machine Learning Research*, 6: 2153–2175, 2005. [99](#)
- M. Eberts and I. Steinwart. Optimal learning rates for least squares svms using gaussian kernels. In *Advances in neural information processing systems*, pages 1539–1547, 2011. [96](#)
- S. Efromovich. *Nonparametric Curve Estimation: Methods, Theory and Application*. Springer, 1999. [51](#)
- J. Fan. Local linear regression smoothers and their minimax efficiencies. *Annals of Statistics*, 21:196–216, 1993. [56](#)
- J. Fan and I. Gijbels. *Local polynomial modelling and its applications*, volume 66. Monographs on statistics and applied probability 66, CRC Press, 1996. [56](#), [57](#)
- J. Fan, Q. Yao, and H. Tong. Estimation of conditional densities and sensitivity measures in nonlinear dynamical systems. *Biometrika*, 83(1):189–206, 1996. [107](#)
- P. E. Freeman, R. Izbicki, and A. B. Lee. A unified framework for constructing, tuning and assessing photometric redshift density estimates in a selection bias setting. *Monthly Notices of the Royal Astronomical Society*, 468(4):4556–4565, 2017. [147](#)
- Y. Freund and R. E. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995. [134](#)
- J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010. [34](#), [121](#)
- C. F. Gauss. *Theoria motus corporum coelestium in sectionibus conicis solem ambientium*. 1809. [3](#)
- E. Greenshtein and Y. Ritov. Persistence in high-dimensional linear predictor selection and the virtue of overparametrization. *Bernoulli*, 10(6):971–988, 2004. [34](#)
- L. Györfi and A. Krzyzak. *A distribution-free theory of nonparametric regression*. Springer, 2002. [27](#), [83](#), [86](#)
- P. Hall, J. S. Racine, and Q. Li. Cross-validation and the estimation of conditional probability densities. *Journal of the American Statistical Association*, 99:1015–1026, 2004. [107](#)

- T. Hastie and R. Tibshirani. Generalized additive models. *Statistical science*, pages 297–310, 1986. [67](#)
- T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, second edition, 2009a. [52](#), [57](#), [121](#)
- T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning*, volume 2. Springer, 2009b. [34](#)
- A. E. Hoerl and R. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12:55–67, 1970. [35](#)
- R. J. Hyndman, D. M. Bashtannyk, and G. K. Grunwald. Estimating and visualizing conditional densities. *Journal of Computational & Graphical Statistics*, 5:315–336, 1996. [107](#)
- R. Izbicki and A. B. Lee. Nonparametric conditional density estimation in a high-dimensional regression setting. *Journal of Computational and Graphical Statistics*, 25(4):1297–1316, 2016. [107](#)
- R. Izbicki and A. B. Lee. Converting high-dimensional regression to high-dimensional conditional density estimation. *Electron. J. Statist.*, 11(2):2800–2831, 2017. doi: 10.1214/17-EJS1302. URL <http://dx.doi.org/10.1214/17-EJS1302>. [107](#)
- R. Izbicki, A. B. Lee, and P. E. Freeman. Photo- z estimation: An example of nonparametric conditional density estimation under selection bias. *The Annals of Applied Statistics*, 11(2):698–724, 2017. [147](#), [148](#)
- G. James, D. Witten, T. Hastie, and R. Tibshirani. *An introduction to statistical learning*. Springer, 2013. [31](#), [39](#)
- G. Kimeldorf and G. Wahba. Some results on tchebycheffian spline functions. *Journal of Mathematical Analysis and Applications*, 33(1):82–95, 1971. [61](#)
- G. S. Kimeldorf and G. Wahba. A correspondence between bayesian estimation on stochastic processes and smoothing by splines. *The Annals of Mathematical Statistics*, pages 495–502, 1970. [57](#)
- S. Kpotufe. k-NN regression adapts to local intrinsic dimension. In *Advances in Neural Information Processing Systems 24*, pages 729–737. The MIT Press, 2011. [96](#)
- S. G. Krantz and H. R. Parks. *A primer of real analytic functions*. Springer, 2002. [51](#)
- A. B. Lee and R. Izbicki. A spectral series approach to high-dimensional nonparametric regression. *Electronic Journal of Statistics*, 10(1):423–463, 2016. [99](#)
- A. M. Legendre. *Nouvelles méthodes pour la détermination des orbites des comètes*. F. Didot, 1805. [3](#)
- W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943. [78](#)
- H. Q. Minh. Some properties of Gaussian Reproducing Kernel Hilbert Spaces and their implications for function approximation and learning theory. *Constructive Approximation*, 32(2):307–338, 2010. [97](#)

- H. Q. Minh, P. Niyogi, and Y. Yao. Mercer's theorem, feature maps, and smoothing. In *19th Annual Conference on Learning Theory*, 2006. 97
- E. A. Nadaraya. On estimating regression. *Theory of Probability & Its Applications*, 9(1):141–142, 1964. 54
- J. Neter, M. H. Kutner, C. J. Nachtsheim, and W. Wasserman. *Applied linear statistical models*, volume 4. Irwin Chicago, 1996. 26
- A. Nosedal-Sanchez, C. B. Storlie, T. C.M. Lee, and R. Christensen. Reproducing kernel hilbert spaces for penalized regression: A tutorial. *The American Statistician*, 66(1):50–60, 2012. 57, 63
- T. Park and G. Casella. The bayesian lasso. *Journal of the American Statistical Association*, 103(482):681–686, 2008. 37
- N. D. Pearce and M. P. Wand. Penalized splines and reproducing kernel methods. *The american statistician*, 60(3), 2006. 63
- M. Pontil. Learning with reproducing kernel hilbert spaces: a guide tour. *Bulletin of the Italian Artificial Intelligence Association, AI* IA Notizie*, 2003. 67, 130
- P. Ravikumar, J. Lafferty, H. Liu, and L. Wasserman. Sparse additive models. *Journal of the Royal Statistical Society, Series B*, 71(5):1009–1030, 2009. 100
- M. T. Ribeiro, S. Singh, and C. Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM, 2016. 105
- R. Rojas. *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013. 82
- F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958. 78
- M. Rosenblatt. Conditional probability density and regression estimators. In P.R. Krishnaiah, editor, *Multivariate Analysis II*. 1969. 107
- M. Saerens, P. Latinne, and C. Decaestecker. Adjusting the outputs of a classifier to new a priori probabilities: a simple procedure. *Neural computation*, 14(1):21–41, 2002. 151
- G. Salton, A. Wong, and C. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975. 208
- B. Scholkopf and A. Smola. Learning with kernels. *MIT Press*, pages 110–146, 2002. 59
- T. Shi, M. Belkin, and B. Yu. Data spectroscopy: eigenspace of convolution operators and clustering. *The Annals of Statistics*, 37, 6B:3960–3984, 2009. 97
- A. Smola and V. Vapnik. Support vector regression machines. *Advances in neural information processing systems*, 9:155–161, 1997. 66
- A. N. Srivastava and M. Sahami. *Text mining: Classification, clustering, and applications*. CRC Press, 2009. 208

- T. A. Stamey, J. N. Kabalin, J. E. McNeal, I. M. Johnstone, F. Freiha, E. A. Redwine, and N. Yang. Prostate specific antigen in the diagnosis and treatment of adenocarcinoma of the prostate. ii. radical prostatectomy treated patients. *The Journal of urology*, 141(5):1076–1083, 1989. [39](#)
- I. Steinwart and A. Christmann. *Support Vector Machines*. Springer, 2008. [96](#)
- C. J. Stone. Consistent nonparametric regression. *The annals of statistics*, pages 595–620, 1977. [52](#), [56](#)
- M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 111–147, 1974. [13](#)
- M. Sugiyama, I. Takeuchi, T. Suzuki, T. Kanamori, H. Hachiya, and D. Okanohara. Conditional density estimation via least-squares density ratio estimation. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 781–788, 2010. [107](#)
- M. Sugiyama, N. D. Lawrence, and A. Schwaighofer. *Dataset shift in machine learning*. The MIT Press, 2017. [147](#)
- J. B. Tenenbaum, V. De Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000. [6](#)
- R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B, Methodological*, 58:267–288, 1996. [32](#)
- A. B. Tsybakov. *Introduction to nonparametric estimation*. Springer Series in Statistics, 2009. [82](#), [86](#)
- A. F. Vaz, R. Izbicki, and R. B. Stern. Quantification under prior probability shift: the ratio estimator and its extensions. *arXiv preprint arXiv:1807.03929*, 2018. [151](#)
- G. Wahba. *Spline Models for Observational Data*. SIAM, 1990. [63](#)
- L. Wasserman. *All of Nonparametric Statistics*. Springer-Verlag New York, Inc., 2006. [14](#), [23](#), [51](#), [57](#), [84](#), [94](#)
- G. S. Watson. Smooth regression analysis. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 359–372, 1964. [54](#)

Índice Remissivo

AIC, 29	Perda 0-1, 114
Características (features), 3	Prior shift, 151
Conjunto de teste, 12	Rótulo (label), 3
Conjunto de treinamento, 12	Regressão Ridge, 35
Conjunto de validação, 12	Regressão linear, 25
convergência, 82	Risco, 9
Covariate shift, 148	Risco condicional, 10
Data splitting, 12	Risco esperado, 10
Dataset shift, 147	Risco quadrático, 9
Função de perda, 9	Seleção de variáveis, 36
Função de risco, 114	Stepwise, 31
Lasso, 32	Sub-ajuste, 11
Lipschitz, 82	Super-ajuste, 11
Método de mínimos quadrados, 25	Tuning parameters, 23
Oráculo, 26	Underfitting, 11
Overfitting, 11	Validação cruzada, 12
Penalização, 20, 29	Variáveis explicativas, 3
	Variável resposta, 3
	Viés de seleção, 147