

Anotações sobre Machine Learning

Igor Ruys Cartucho

SUMÁRIO

Capítulo 1:	Introdução	5
Capítulo 2:	Estimação de Parâmetros.....	10
2.1	Maximum Likelihood Estimation (MLE)	10
2.2	Maximum A Posteriori (MAP) Estimation	14
Capítulo 3:	<i>K-Nearest Neighbors</i>	16
3.1	Implementações do Método.....	16
3.1.1	Força bruta	16
3.1.2	K-D Tree	18
3.1.3	<i>Ball Tree</i>	21
3.1.4	Variações do KNN	23
3.2	Maldição da Dimensionalidade	24
3.3	Métricas de Distância	26
Capítulo 4:	Regressão Linear	29
4.1	Diferentes Interpretações da Regressão Linear	30
4.1.1	Ponto de Vista de Otimização	30
4.1.2	Ponto de Vista de Álgebra Linear	31
4.1.3	Ponto de Vista Probabilístico	33
4.2	Regressão Polinomial	34
4.3	Forma Geral da Regressão Linear.....	35
Capítulo 5:	Regressão Logística	36
5.1	Mínimos Quadrados Para Classificação	37
5.2	Modelando a Probabilidade A Priori.....	40
5.2.1	Caso $K = 2$	40
5.2.2	Caso $K > 2$	41
5.2.3	Aplicação para distribuições conhecidas.....	41
5.3	Construção da Regressão Logística	44
5.4	Conclusões.....	47
Capítulo 6:	Support Vector Machine	49
6.1	Caso Não Linearmente Separável	53
6.2	Máquina de Vetor de Suporte.....	56
6.2.1	SVM como Método de Penalização	57
6.2.2	Classificação Multiclasse	58
6.3	SVM para Regressão.....	59

6.4	Aspectos Práticos	61
Capítulo 7:	Árvore de Decisão	63
7.1	Árvore de Classificação	63
Capítulo 8:	Boosting.....	69
8.1	AdaBoost	69
8.1.1	Descrição do Algoritmo	69
8.1.2	Discussão sobre a <i>loss function</i>	73
8.2	Boosting Trees.....	75
8.3	<i>Gradient Boosting</i>	76
8.3.1	Exemplo com a <i>loss erro quadrático</i>	77
8.3.2	Derivação Matemática	77
8.4	XGBoost.....	78
Capítulo 9:	Rede Neural.....	80
9.1	Estrutura da Rede Neural.....	80
9.2	Algoritmo de Backpropagation	83
9.3	Escolhendo a Função Custo e as Não-Linearidades	87
9.3.1	Observando a Camada de Saída.....	87
9.3.2	Observando as Camadas Escondidas	88
9.4	Regularização	89
Capítulo 10:	Feature Selection.....	90
Capítulo 11:	Principal Component Analysis.....	92
11.1	Formulação da Máxima Variância	93
11.2	Detalhes Importantes.....	94
11.3	Relação com a Decomposição SVD	95
Capítulo 12:	Avaliação de Desempenho e Seleção de Modelo	96
12.1	Métricas para Classificação Binária.....	96
12.1.1	Acurácia.....	96
12.1.2	True Positive Rate (TPR) ou Sensibilidade.....	96
12.1.3	False Positive Rate (FPR)	96
12.1.4	Precisão	97
12.1.5	<i>Recall</i>	97
12.1.6	<i>F1-score</i>	97
12.1.7	<i>F-score</i>	98
12.1.8	<i>Receiver Operating Characteristic (ROC)</i>	98
12.2	Métricas para Classificação Multiclasse.....	99
12.2.1	Acurácia.....	99

12.2.2	Acurácia Balanceada	100
12.2.3	Métricas Macro	100
12.2.4	Métricas Macro Ponderadas (<i>weighted</i>).....	101
12.2.5	Métricas Micro	101
12.3	Métricas para Regressão	101
12.3.1	<i>Mean Squared Error</i> (MSE).....	102
12.3.2	<i>Root Mean Squared Error</i> (RMSE)	102
12.3.3	<i>Mean Absolute Error</i> (MAE)	102
12.3.4	<i>Mean Absolute Percentage Error</i> (MAPE)	102
12.3.5	Coeficiente de Correlação de Pearson (PCC)	103
12.3.6	Coeficiente de Determinação (<i>R</i>²)	103
12.4	Métodos de Validação.....	104
Capítulo 13:	105
Capítulo 14:	Apêndices	106
Capítulo 15:	Referências	122

Capítulo 1: INTRODUÇÃO

- Tipos de aprendizado (supervisionado, não-supervisionado, semi-supervisionado, federado, por reforço).
- Teoria da decisão.
- Tipos de modelos (paramétrico x não-paramétrico; generativo x discriminativo)
- Estimação de uma função (minimização do valor esperado de uma certa *loss*) [1].
- Função *loss* x função custo.
- Lista de principais *losses*? Ou fica para um apêndice?
- *No free lunch theorem*

1.1 DILEMA VIÉS-VARIÂNCIA

A avaliação de viés e variância de modelos pode ser confuso, pois em estatística, esses termos são usados com frequência, principalmente para descrever variáveis aleatórias. No contexto de avaliação de modelos preditivos, esses termos são usados para descrever o comportamento de um modelo no que diz respeito a sua sensibilidade ao conjunto de treinamento utilizado para construí-lo. Modelos com alta variância e baixo viés, tendem a sofrer mudanças mais drásticas até para pequenas mudanças no conjunto de treinamento. Por outro lado, um modelo com alto viés e baixa variância são tendem a ser menos adaptáveis, o que faz com que mudanças no conjunto de treinamento não gerem modelos muito diferentes.

A variância e o viés estão bastante relacionados com a complexidade do modelo. Quanto maior a complexidade, mais adaptável é o modelo e mais ele se ajusta ao conjunto de treinamento, chegando ao ponto de, para modelos muito complexos, se adaptar até às menores estruturas dos dados, como flutuações aleatórias, o que costuma ser indesejável. Assim, quanto mais complexo um modelo, maior sua variância. Por outro lado, modelos menos complexos tendem a ser construídos com hipóteses e suposições muito fortes acerca do fenômeno sendo modelado, o que o torna menos flexível e menos adaptável ao conjunto de treinamento. Modelos desse tipo tendem a ter alto viés e baixa variância.

Essa relação entre a complexidade e a variância e o viés podem ser mais bem entendidas com ajuda de um exemplo. Considere que desejamos criar um modelo preditivo $\hat{f}(X)$ para descrever um fenômeno, representado pelas variáveis aleatórias X e Y de entrada e *target*, respectivamente, que apresentam distribuição conjunta $p_{X,Y}(x, y)$. Considere também que o modelo preditivo é construído a partir de um conjunto de treinamento $\mathcal{T} = \{(x_n, y_n)\}_{n=1}^N$, formado a partir de N amostragens conjuntas de (X, Y) . Nesse problema específico, podemos considerar que $Y = f(X) + \epsilon$, em que $f(X) = \sin(2\pi X)$ e ϵ representam flutuações aleatórias gaussianas. O modelo preditivo é dado por uma regressão linear usando 24 funções base Gaussianas, com regularização *ridge* controlada pelo coeficiente λ para ajustar a complexidade do modelo.

Amostragem a partir de $p_{X,Y}(x, y)$

Para entender intuitivamente como é feita uma amostragem a partir de uma distribuição conjunta, considere o seguinte exemplo. Em determinado bairro, o preço das casas, representado por Y , é dependente do tamanho delas, representado por X . Nesse exemplo,

$X \sim \mathcal{N}(100, 20^2)$ (sendo as unidades em m^2) e $Y = 2000X + \epsilon$ reais, em que $\epsilon \sim \mathcal{N}(0, 5000^2)$ (sendo as unidades em reais) representa uma flutuação aleatória. Seja, $g(\cdot; \mu, \sigma^2)$ a p.d.f. de uma distribuição $\mathcal{N}(\mu, \sigma^2)$. Com isso, temos:

- $p_X(x) = g(x; 100, 20^2)$;
- $p_Y(y) = g(y; 200\,000, 40\,000^2 + 5000^2)$;
- $p_{Y|X}(y|x) = 2000x + g(y; 0, 5000^2) = g(y; 2000x, 5000^2)$;
- $p_{X,Y}(x, y) = p_{Y|X}(y|x)p_X(x) = g(y; 2000x, 5000^2) \times g(x; 100\,m^2, 20^2)$.

A Figura 1-1 mostra alguns valores amostrados das variáveis X e Y , junto com a curva de nível da função de densidade conjunta delas.

Quando falamos em amostrar valores de X e Y em conjunto, estamos falando em amostrar de uma distribuição com densidade $p_{X,Y}(x, y)$. Obviamente, na prática, não conhecemos previamente $p_{X,Y}(x, y)$. Logo, coletamos manualmente os valores de x_n e y_n , registrando o tamanho de uma casa desse bairro e o seu preço.

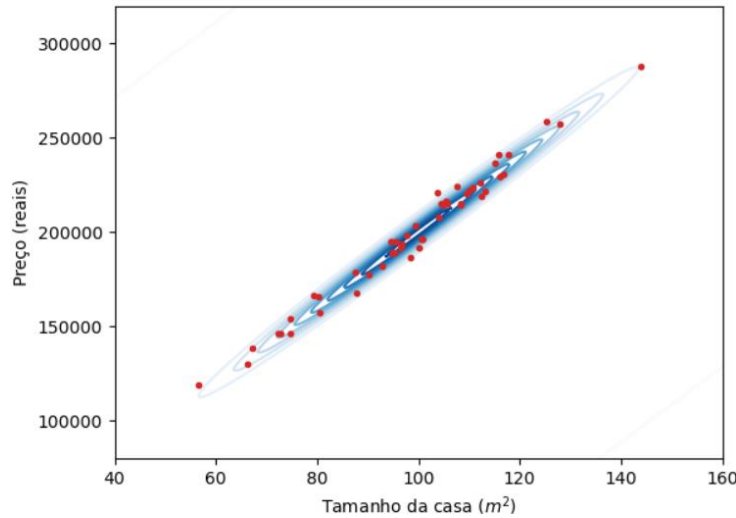


Figura 1-1: Algumas amostras (em vermelho) de (X, Y) do exemplo do preço das casas. As curvas de nível da p.d.f. conjunta $p_{X,Y}(x, y)$ também são mostradas (em azul).

Dadas essas premissas, geramos 100 conjuntos de treinamento diferentes amostrando pontos de $p_{X,Y}(x, y)$, com $N = 25$. Com isso, geramos 100 modelos diferentes $\hat{f}_{\mathcal{T}_i}(X)$, cada um gerado a partir de um conjunto de treinamento \mathcal{T}_i diferente. Fazemos exatamente esse mesmo procedimento para 3 valores de λ diferentes. Os resultados são mostrados na Figura 1-2. Primeiramente, podemos perceber que, para o modelo menos complexo (maior valor de λ), os modelos não se adaptam muito bem à forma da senoide dada por f , além de serem todos muito próximos uns dos outros. Modelos desse tipo apresentam viés grande e baixa variância. Por outro lado, à medida que aumentamos a complexidade do modelo (diminuindo o valor de λ), percebemos uma grande variabilidade das curvas de regressão, o que indica que mudanças no conjunto de treinamento têm alto impacto na forma do modelo. Modelos desse tipo apresentam alta variância e baixo viés.

Outro ponto interessante de se reparar é que, no lado direito, em que é mostrada a curva de média dos modelos, podemos ver que, quanto mais complexos são os modelos, melhor se torna a curva de média. Esse é o princípio que está na base dos métodos de *ensemble*, que serão detalhados em capítulos posteriores.

No entanto, em geral, tanto modelos com alto viés, quanto modelos com alta variância tendem a ser indesejáveis, pois ambos tendem a aumentar o erro do modelo. Para entender isso, precisamos primeiramente definir alguns conceitos.

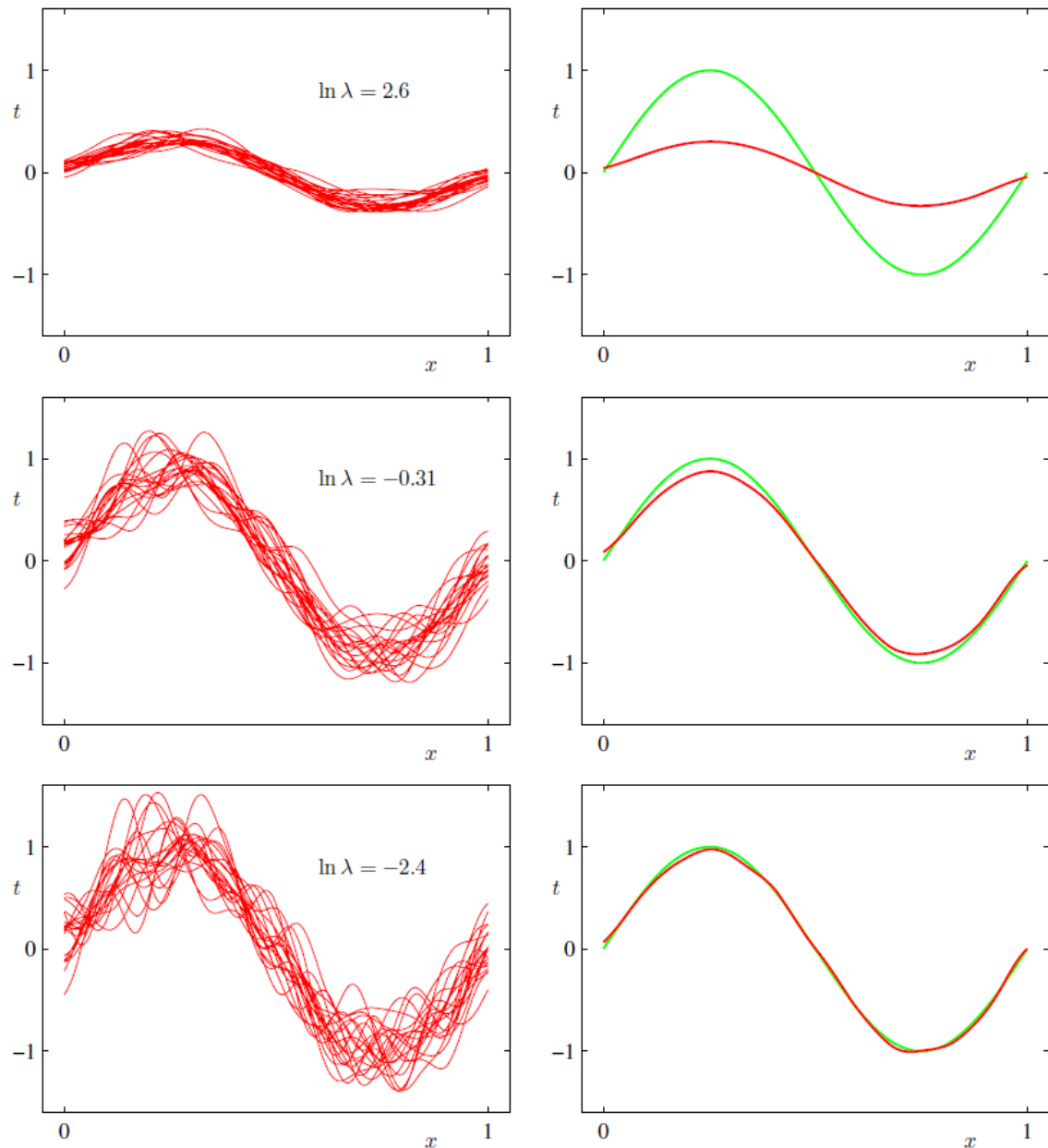


Figura 1-2: Ilustração da relação entre a complexidade do modelo e o viés e a variância. São usados 100 conjuntos de treinamento para cada valor de λ , cada conjunto com tamanho $N = 25$. Os gráficos do lado esquerdo mostram os 100 modelos treinados, cada linha representando um valor de λ diferente. Já os gráficos do lado direito, mostram a função de referência $f(X)$ em verde e, em vermelho, a média dos modelos para o valor de λ dessa linha. Essa imagem foi retirada de [2].

O primeiro deles é o chamado de **erro de teste** ou **erro de generalização**, que é o erro de predição obtido por meio de dados de teste independentes, que seguem a distribuição conjunta $p_{X,Y}(x, y)$. O erro de generalização é dado por

$$\text{Err}_{\mathcal{T}} = \mathbb{E} \left[L \left(Y, \hat{f}(X) \right) | \mathcal{T} \right],$$

em que L é uma função *loss* que quantifica o erro entre Y e $\hat{f}(X)$.

O que essa expressão está dizendo é que, dado um conjunto de treinamento \mathcal{T} fixo utilizado para construir \hat{f} , é calculado o valor esperado de $L(Y, \hat{f}(X))$ sobre as diferentes observações (x_n, y_n) das variáveis aleatórias X e Y , que representam a amostra de teste. Observar que, como estamos calculando o valor esperado, estamos considerando as infinitas combinações de (x_n, y_n) para os dados de teste, uma vez que o cálculo de \mathbb{E} envolve uma integração (possivelmente uma soma) ao longo de todas as realizações possíveis de X e Y .

Ao construirmos o estimador \hat{f} usando diferentes conjuntos de teste, obtemos também diferentes valores de $\text{Err}_{\mathcal{T}}$, pois o estimador será diferente para cada \mathcal{T} escolhido. Logo, se $\mathcal{T}_1 \neq \mathcal{T}_2$, em geral, $\text{Err}_{\mathcal{T}_1} \neq \text{Err}_{\mathcal{T}_2}$. Para termos uma noção do erro médio sobre diferentes conjuntos de teste, calculamos o valor esperado do erro de generalização com relação a \mathcal{T} , o que dá origem ao **erro de teste esperado**:

$$\text{Err} = \mathbb{E}[\text{Err}_{\mathcal{T}}] = \mathbb{E}_{\mathcal{T}} \left[\mathbb{E} \left[L(Y, \hat{f}(X)) | \mathcal{T} \right] \right] = \mathbb{E} \left[L(Y, \hat{f}(X)) \right],$$

em que o subscrito \mathcal{T} em $\mathbb{E}_{\mathcal{T}}$ foi utilizado apenas para deixar explícito que o valor esperado mais externo nessa expressão é calculado sobre os diferentes conjuntos de teste.

Além dessas métricas de erro apresentadas, também existe o **erro de treinamento**, calculado a partir da média amostral da amostra de treinamento:

$$\overline{\text{err}} = \frac{1}{N} \sum_{n=1}^N L(y_n, \hat{f}(x_n)).$$

Além disso, também podemos calcular o erro de treinamento esperado $\mathbb{E}[\overline{\text{err}}]$, calculando o valor esperado sobre os diferentes conjuntos de treinamento.

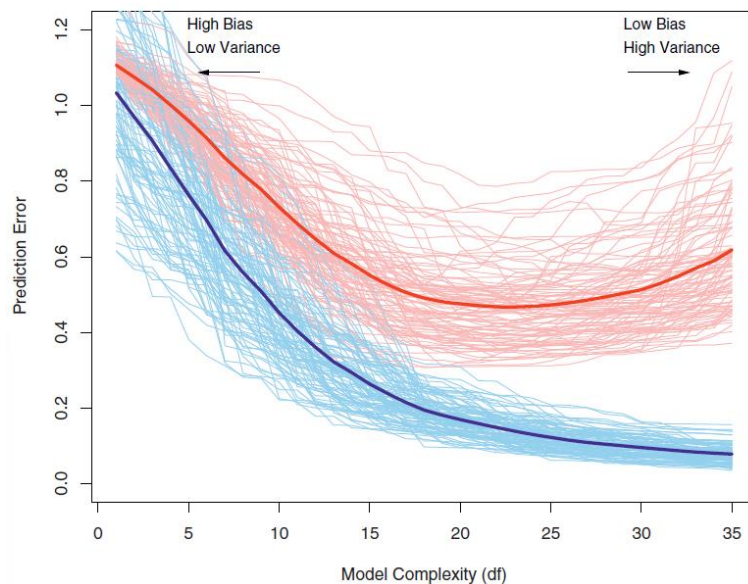


Figura 1-3: Comportamento dos erros de teste e treinamento em função da complexidade do modelo. As curvas de cor azul-claro representam o erro de treinamento $\overline{\text{err}}$ e as curvas de cor vermelho-claro representam o erro de teste $\text{Err}_{\mathcal{T}}$, para 100 diferentes conjuntos de treinamento \mathcal{T} (para cada cor), todos de tamanho 50. Já as curvas azul-escuro e vermelho-escuro representam Err e $\mathbb{E}[\overline{\text{err}}]$, respectivamente. Essa imagem foi retirada de [3].

A Erro! Fonte de referência não encontrada. ilustra como se comportam esses erros de acordo com a complexidade do modelo. Com isso, conseguimos perceber que, à medida que o modelo fica mais complexo e, consequentemente, com maior variância, seu erro de

treinamento diminui, mas seu erro de teste aumenta. Além disso, à medida que o modelo fica menos complexo e, conseqüentemente, com maior viés, seu erro de treinamento aumenta, assim como seu erro de teste. Portanto, podemos perceber que existe uma faixa de complexidade ideal para o modelo, em que ele não apresenta muito viés nem muita variância e seu erro de teste é menor.

Capítulo 2: ESTIMAÇÃO DE PARÂMETROS

Para entendermos os métodos desta seção, precisamos lembrar do Teorema de Bayes:

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)},$$

em que

$P(\theta)$: **Probabilidade a priori** de a distribuição que gerou os dados D tenham θ como parâmetro.

$P(D)$: Probabilidade de os dados D terem sido gerados. O símbolo D resume o acontecimento de um conjunto de eventos. Se estivermos considerando um experimento em que uma moeda é lançada N vezes, teremos uma variável aleatória que assume valor 0 (coroa) e 1 (cara). D poderia representar, por exemplo, o conjunto de 3 lançamentos, ou seja, $(X_1 = 1) \cap (X_2 = 1) \cap (X_3 = 0)$. Assim, $P(D)$ representaria a probabilidade de acontecer essa sequência de resultados.

$P(\theta|D)$: **Probabilidade a posteriori**, ou seja, é a probabilidade de que seja θ o parâmetro da distribuição que gera os dados, sabendo que os dados D foram gerados a partir dessa distribuição.

$P(D|\theta)$: **Função de verossimilhança** (*likelihood*), que é a probabilidade de os dados D terem sido gerados por uma distribuição com parâmetro θ dado. Normalmente, em problemas de estimação, temos os dados e tentamos determinar θ . Dependendo do contexto, ela também costuma ser definida como [2, p. 122]

$$\mathcal{L}_N(\theta) = \prod_{n=1}^N P(X_i; \theta),$$

em que N é o número de amostra contidas nos dados. Lembrando que, $\prod_{n=1}^N P(X_i; \theta) = P(\cap_{n=1}^N X_i; \theta) = P(D; \theta)$. Logo, é só uma questão de como interpretar θ : como uma informação a priori ($P(D|\theta)$) ou como um parâmetro da lei de probabilidade ($P(D; \theta)$).

Em geral, quando falamos de *machine learning*, as probabilidades a priori e a posteriori e a verossimilhança são sempre representadas por essas probabilidades mostradas acima. No entanto, em geral, em qualquer outro problema de estatística, as probabilidades a priori e a posteriori vão depender do ponto de vista. Em dado exemplo, podemos considerar $P(A)$ como sendo a probabilidade a priori e $P(A|B)$ a probabilidade posteriori, mas se invertermos o ponto de vista, podemos também considerar que a $P(B)$ e a $P(B|A)$ são as probabilidades a priori e a posteriori, respectivamente.

2.1 MAXIMUM LIKELIHOOD ESTIMATION (MLE)

Estimamos os parâmetros somente com base nas evidências (dados). Para isso, maximizamos a função de verossimilhança da distribuição, isto é,

$$\theta_{MLE} = \arg \max_{\theta} P(D|\theta).$$

Conhecendo-se a expressão de $P(D|\theta)$, torna-se fácil resolver esse problema. Isso nem sempre é possível, pois nem sempre conhecemos a distribuição geradora dos dados $X_i \in D$. No caso de lançamentos de moedas, como veremos no exemplo abaixo, podemos supor que os dados tem distribuição de Bernoulli com parâmetro p . Mesmo nos casos em que não conhecemos a distribuição dos dados, ainda é possível realizar hipóteses e assumir que eles seguem alguma distribuição conhecida (Bernoulli, Gaussiana...).

Lembrar que, usualmente, quando nos referimos à expressão das distribuições usamos uma notação como $f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]$, sem deixar explícitos os parâmetros. No entanto, é só uma questão de notação e poderíamos escrever $f(x; \mu, \sigma^2)$, $f(x|\mu, \sigma^2)$ ou $\mathcal{L}(\mu, \sigma^2)$. Cada notação interpreta os parâmetros de uma maneira diferente, como já foi discutido no início do capítulo.

É importante perceber que toda probabilidade pode ser interpretada como uma probabilidade condicional, por mais que a informação a posteriori não esteja explicitamente indicada na expressão matemática. Assim, de maneira resumida, quando escrevíamos apenas $P(x)$ nas aulas de probabilidade e estatística, era só uma maneira resumida de escrever $P(x|K)$, em que K é algum conhecimento a posteriori (os parâmetros da distribuição, por exemplo, que estavam implícitos). Em [3, p. 59], é feita uma ótima discussão sobre isso.

Para finalizar, é necessário prestar atenção que, quando utilizamos a notação $P(D)$, estamos assumindo que não temos nenhuma outra informação a priori, nem mesmo a distribuição dos dados e, por isso, não podemos utilizar as expressões conhecidas das funções de distribuição de probabilidade.

Exemplo 1

Consideramos uma moeda que foi lançada 5 vezes, dando como resultado $D = \{C, K, C, C, C\}$. Queremos determinar a distribuição de probabilidades que representa os resultados dessa moeda. Podemos dizer que é uma distribuição de Bernoulli com parâmetro p (probabilidade de sucesso, que será representado pelo resultado “Cara”).

Usando o critério de MLE, desejamos que a *likelihood* $P(D|p)$, seja a maior possível, ou seja, a probabilidade de que os dados D tenham vindo de uma distribuição de Bernoulli com parâmetro p dado. Se considerarmos que os lançamentos são independentes, temos

$$P(D|p) = P(C|p)P(K|p)P(C|p)P(C|p)P(C|p)$$

Como, $P(K|p) = 1 - p$ e $P(C|p) = p$, temos

$$P(D|p) = p^4(1 - p) = -p^5 + p^4$$

Para maximizar a expressão acima, fazemos

$$\frac{\partial P(D|p_{MLE})}{\partial p_{MLE}} = 0$$

$$-5p_{MLE}^4 + 4p_{MLE}^3 = 0$$

$$p_{MLE}^3(-5p_{MLE} + 4) = 0$$

A única solução coerente para esse problema é a solução para

$$-5p_{MLE} + 4 = 0$$

$$p_{MLE} = 4/5 = 0,8.$$

Com isso, a partir das evidências, chegamos à conclusão de que a moeda é viciada. No entanto, podemos argumentar que os dados não são representativos e que precisaríamos de mais dados para chegar a um resultado mais acurado. De toda forma, o método nos dá um mecanismo para estimar a probabilidade p de sair “Cara”.

Se tivéssemos um número genérico n de “Caras” e m de “Coroas”, com $N = n + m$, teríamos uma expressão do tipo

$$P(D|p) = p^n(1 - p)^m.$$

Essa expressão é muito difícil de derivar. Um possível artifício que podemos utilizar é aplicar o logaritmo, o que nos daria

$$\ell_N(p) = \log P(D|p) = n \log p + m \log(1 - p),$$

ficamos com uma soma em vez de um produto, que é muito mais simples de derivar.

À primeira vista isso pode parecer estranho, mas, se repararmos, o ponto de mínimo de $\ell_N(p)$ também é o ponto de mínimo de $P(D|p)$. O termo $\ell_N(p)$ é chamado de **log likelihood**.

Assim, prosseguimos fazendo

$$\frac{\partial}{\partial p} \ell_N(p) = 0$$

$$\frac{n}{p} - \frac{m}{1 - p} = 0$$

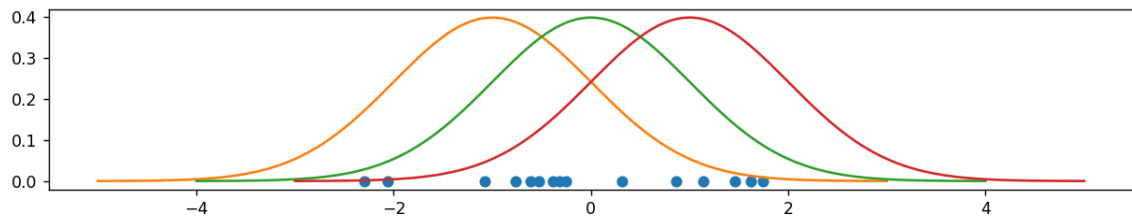
$$\frac{n - np - mp}{p(1 - p)} = 0$$

$$p_{MLE} = \frac{n}{N}, \text{ para } p \in (0,1),$$

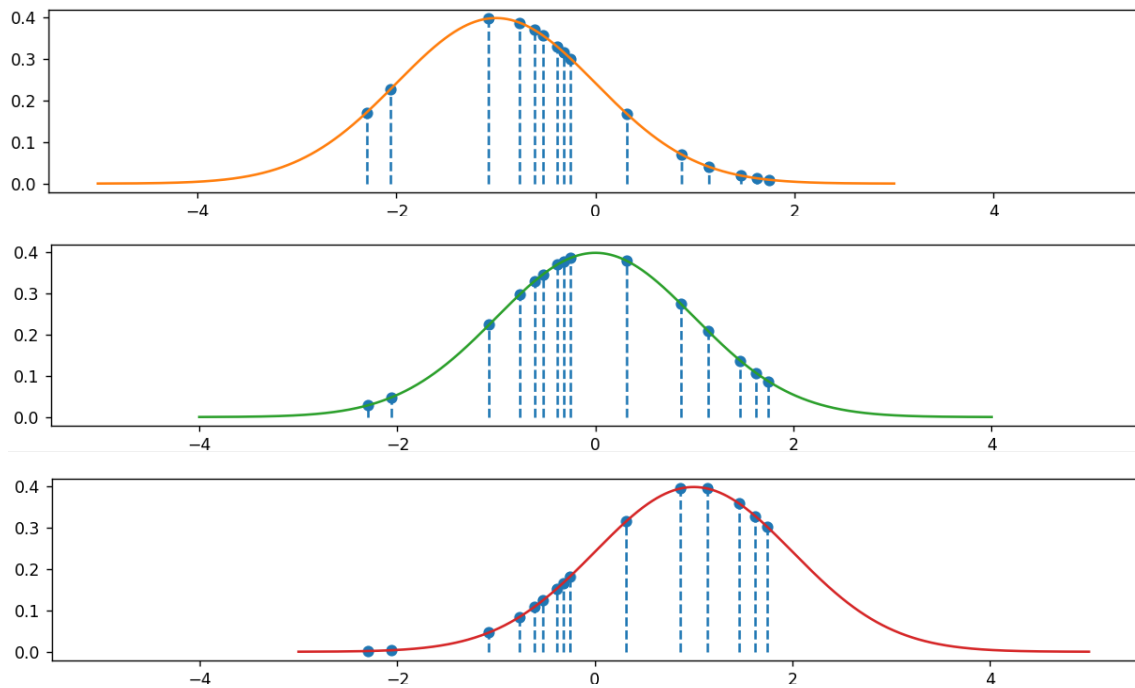
que é justamente a proporção de “Caras” no experimento.

Exemplo 2

Vamos considerar que agora temos alguns dados D e sabemos que eles seguem uma distribuição Gaussiana, mas com parâmetros desconhecidos μ e σ . Nosso trabalho, novamente, é encontrar o valor desses parâmetros ao tentar maximizar a *likelihood* $f(D|\theta)$, com $\theta = (\mu, \sigma)$ (aqui estamos usando f no lugar de P por se tratar de uma distribuição contínua e não mais de uma distribuição discreta). Intuitivamente, estamos tentando encontrar a curva representativa da distribuição Gaussiana que se encaixa melhor nos dados. Na imagem abaixo, observamos os dados em azul e três exemplos de gaussianas que poderiam ter gerado esses dados.



Abaixo, as três Gaussianas são representadas em gráficos separados e os pontos são projetados sobre as curvas para facilitar a análise.



A partir dessa análise visual, podemos ver que as Gaussianas em laranja e verde são as que mais parecem ter gerado os dados em azul.

Intuitivamente, é isso que fazemos: variamos os parâmetros até encontrar a curva que compreende os dados de maneira mais “coerente”, ou seja, encontramos a curva mais provável de ter gerado esses dados. Isso é o equivalente a encontrar os parâmetros que dão o maior valor da *likelihood* $f(D|\theta)$.

Como temos N dados supostamente gerados independentemente, a *likelihood* pode ser escrita como

$$f(D|\theta) = \prod_{n=1}^N f(x_n|\theta).$$

Assim, queremos que esse produto seja o maior possível. Falando de maneira simplificada e visual, queremos encontrar θ tal que os pontos estejam arranjados em sua maioria mais próximos do pico da gaussiana do que das caudas, pois assim, o produto será o maior possível.

Partindo para a análise matemática, teremos um sistema de duas equações: uma considerando a derivada com relação a μ e outra com relação a σ , como mostrado abaixo:

$$\begin{cases} \frac{\partial f(D|\boldsymbol{\theta})}{\partial \mu} = 0 \\ \frac{\partial f(D|\boldsymbol{\theta})}{\partial \sigma} = 0 \end{cases}$$

O produtório de $f(D|\boldsymbol{\theta})$ torna as derivadas um pouco inconvenientes. Por isso, adotamos um procedimento similar ao do exemplo anterior e tentamos minimizar a *log likelihood*.

Com isso, teremos

$$\begin{cases} \frac{\partial \log f(D|\boldsymbol{\theta})}{\partial \mu} = 0 \\ \frac{\partial \log f(D|\boldsymbol{\theta})}{\partial \sigma} = 0 \end{cases}$$

Por sorte, a primeira equação desse sistema é independente de σ , o que facilita as contas.

Lembrando que, $f(x|\boldsymbol{\theta}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]$, a primeira equação fica

$$\frac{\partial \log f(D|\boldsymbol{\theta})}{\partial \mu} = \frac{\partial}{\partial \mu} \left[-\sum_{n=1}^N \log \frac{1}{\sqrt{2\pi\sigma^2}} \frac{(x_n - \mu)^2}{2\sigma^2} \right] = 0$$

$$\frac{\partial}{\partial \mu} \left[\sum_{n=1}^N (x_n - \mu)^2 \right] = 0$$

$$-2 \sum_{n=1}^N (x_n - \mu) = 2N\mu - 2 \sum_{n=1}^N x_n = 0$$

$$\mu_{MLE} = \frac{1}{N} \sum_{n=1}^N x_n$$

Ao desenvolver a segunda equação, encontramos

$$\sigma_{MLE} = \frac{1}{N} \sum_{n=1}^N (x_n - \mu_{MLE})^2$$

Portanto, para encontrar o valor dos parâmetros da distribuição gaussiana mais provável de ter gerado os dados, pelo critério MLE, basta substituir os valores de x na equação acima.

2.2 MAXIMUM A POSTERIORI (MAP) ESTIMATION

Estimamos os parâmetros com base nas evidências (dados), mas também em conhecimento a posteriori. Com isso, o que fazemos é calcular

$$\boldsymbol{\theta}_{MAP} = \arg \max_{\boldsymbol{\theta}} P(\boldsymbol{\theta}|D).$$

Diferentemente do caso MLE, não temos uma expressão direta para $P(\boldsymbol{\theta}|D)$, pois, normalmente, não conhecemos uma expressão para a distribuição de probabilidade do

parâmetro θ . Por isso, nos utilizamos do teorema de Bayes e reescrevemos essa equação como

$$\theta_{MAP} = \arg \max_{\theta} \frac{P(D|\theta)P(\theta)}{P(D)} = \arg \max_{\theta} P(D|\theta)P(\theta).$$

Capítulo 3: K-NEAREST NEIGHBORS

O *K-nearest neighbors* (KNN) é um método que pode ser utilizado para classificação e regressão, baseado no princípio de que a saída do método depende dos K vizinhos mais próximos do ponto para o qual se deseja realizar uma predição. Por isso ele é considerado um método baseado em distância. Existem diferentes métricas que podem ser utilizadas para determinar a distância entre dois pontos, como será apresentado ainda neste capítulo, porém a mais comum é a distância Euclidiana.

3.1 IMPLEMENTAÇÕES DO MÉTODO

Nesta seção, são apresentados diferentes algoritmos para resolver o problema de encontrar os K vizinhos mais próximos de um determinado ponto de teste. É importante mencionar que, apesar de serem algoritmos diferentes, eles sempre terão como resultado os mesmos conjuntos de K vizinhos. A diferença entre eles está apenas na forma como é feita a busca por esses K vizinhos mais próximos e, conseqüentemente, na complexidade de tempo deles.

3.1.1 Força bruta

É a implementação mais comum de ser vista do KNN. Para entender seu funcionamento, comecemos pelo problema de classificação. Para um dado valor de K e um certo ponto \mathbf{z} que se deseja classificar, o algoritmo consiste em comparar a distância $D(\mathbf{x}_i, \mathbf{z})$ (podendo ser a distância Euclidiana, por exemplo, como veremos adiante) entre o ponto \mathbf{z} e cada um dos pontos \mathbf{x}_i , que possuem classes conhecidas. Em seguida, separamos os K pontos \mathbf{x}_i mais próximos de $D(\mathbf{x}_i, \mathbf{z})$ e fazemos a contagem de quantos desses pontos pertencem a cada classe, como uma espécie de votação. A classe mais votada (com mais pontos) é a classe utilizada para classificar o ponto \mathbf{x}_i . Essa maneira de classificar o ponto \mathbf{z} é chamada de **hard labeling**, em oposição ao *soft labeling*, que será apresentado adiante. Em caso de empate entre classes, normalmente, a decisão é feita arbitrariamente. Particularmente, no scikit-learn, o classificador KNeighborsClassifier resolve esse empate escolhendo a classe que aparece primeiro no conjunto de dados [4]. Para evitar esse tipo de problema, é recomendável que K não seja um múltiplo do número de classes [5, p. 316].

Outra maneira de classificar o ponto \mathbf{z} , é por meio do **soft labeling** [6], que, em vez de atribuir de a classe majoritária ao ponto \mathbf{z} , associa probabilidades a cada classe. Dessa forma, para cada classe C_j , é calculada a probabilidade de \mathbf{z} pertencer a essa classe:

$$\mathbb{P}(C_j|\mathbf{z}) = \frac{k_j}{K},$$

em que k_j é o número de vizinhos pertencentes à classe C_j .

É interessante notar que, não é necessário um treinamento propriamente dito para utilizar o KNN. Isso porque esse método não aprende parâmetros para classificar novas instâncias. Basta dispor de um conjunto de dados anotados e com isso, já é possível classificar novas instâncias. Assim, para funcionar, ele precisa memorizar todas as instâncias de “treinamento”. Por isso, podemos dizer que o KNN é um método que não apresenta um modelo [7, p. 463]. Outros termos também são usados para caracterizar esse tipo de “aprendizado” do KNN, como *instance based learning* e *non-generalizing learning* [4].

Apesar de, até o momento, só termos tratado o caso da classificação, o KNN também pode ser utilizado para realizar regressões. O funcionamento é muito similar, com a diferença que, em vez de realizar uma classificação com base na classe majoritária dos vizinhos, atribuímos um valor contínuo a \mathbf{z} , calculado a partir da média do *target* dos K vizinhos, ou seja,

$$t_z = \frac{1}{K} \sum_{i=1}^K t_{x_i}.$$

Na Figura 3-1 podemos ver um exemplo de classificação e na Figura 3-2 podemos ver um exemplo de regressão usando o método KNN.

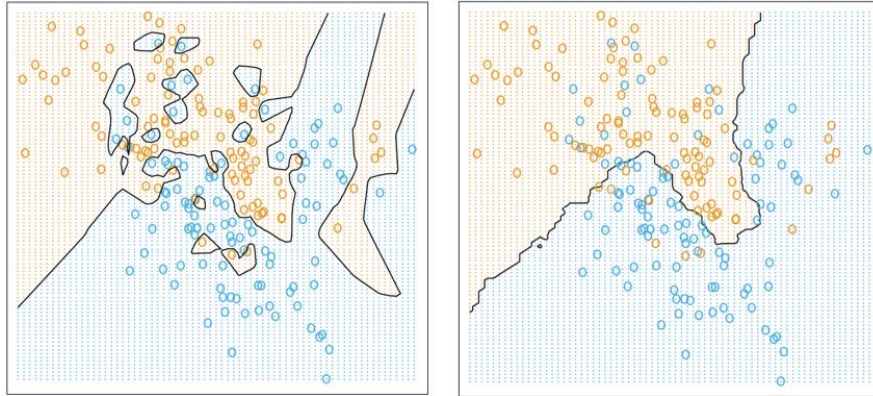


Figura 3-1: Fronteiras de decisão encontrada usando KNN para $K = 1$ (esquerda) e $K = 15$ (direita).

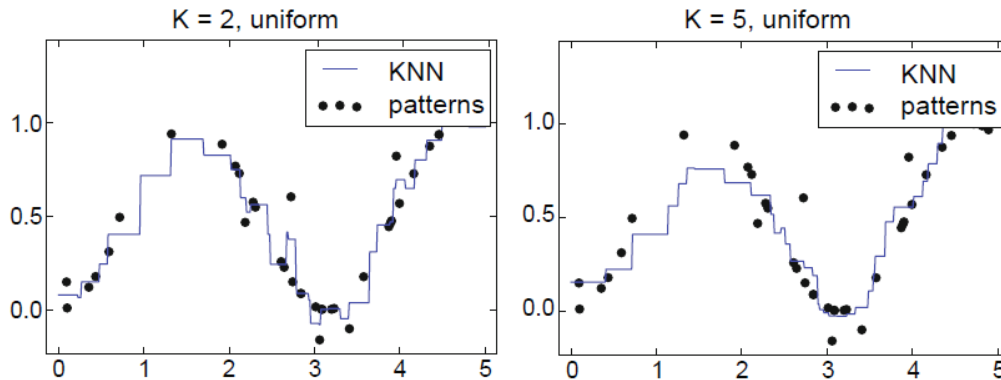


Figura 3-2: Regressão realizado com KNN para dois valores de K .

Do ponto de vista de complexidade, como esse algoritmo não exige treinamento, só faz sentido analisar a complexidade de tempo da inferência diretamente. Nesse caso, consideremos que, para um vizinho, ou seja, $K = 1$, deveremos iterar ao longo dos N pontos de dimensão d e calcular suas respectivas distâncias para um ponto \mathbf{z} . Cada cálculo de distância (considerando a distância Euclidiana) envolve resolver uma expressão do

tipo $\sqrt{\sum_{j=1}^d (x_{(j)} - z_{(j)})^2}$. Logo, nesse caso, teremos uma complexidade $O(N \times d)$. No entanto, para um número K de vizinhos, precisamos repetir todo esse processo K vezes, calculando a cada iteração o k -ésimo vizinho mais próximo. Nesse caso, a complexidade total do algoritmo é de $O(K \times N \times d)$. Portanto, vemos que para espaços de dimensão elevada ou para grandes quantidades de amostra, esse algoritmo pode ser ineficiente.

Com relação à escolha do valor de K , o valor ótimo para esse hiperparâmetro varia dependendo do problema. No entanto, quanto menor o valor de K , maior será a tendência de o método capturar padrões mais sutis do conjunto de dados e possivelmente ruído caso K seja muito pequeno, o que pode levar o método a sofrer *overfitting* durante a predição de novos dados. No caso limite em que $K = 1$, o método tem 100% de acurácia no conjunto de dados inicial, mas pode apresentar erros elevados para novos dados.

Por outro lado, quanto maior o valor de K menos sensível é o método a detalhes mais finos da distribuição dos dados iniciais. Se o valor de K for consideravelmente grande, o método tem maior chance de sofrer *underfitting* e também acabamos por desviar da suposição elementar do método, de que entradas (pontos) semelhantes/próximas possuem saídas semelhantes. Isso porque ao aumentar excessivamente o valor de K , acabamos levando em consideração muitos outros pontos que muitas vezes não estarão perto do ponto de interesse, o que é equivalente a ter uma vizinhança muito grande. No limite em que $K = N$, a solução do problema é trivial, pois qualquer ponto terá todos os outros pontos do conjunto inicial como vizinhos. O problema, então, se resume a sempre classificar (no caso de um problema de classificação) novos pontos como sendo da classe majoritária, o que eleva o erro de classificação. Na Figura 3-1 e na Figura 3-2 é possível ver como se comporta o KNN para diferentes valores de K .

Assim, podemos dizer que:

- ao aumentar o valor de K , diminuimos a complexidade do método, aumentamos seu viés e, para valores excessivamente elevados de K , aumentamos a chance de *underfitting*;
- ao diminuir o valor de K , aumentamos a complexidade do método, aumentamos sua variância e, para valores excessivamente baixos de K , aumentamos a chance de *overfitting*.

3.1.2 K-D Tree

Outra maneira de se encontrar os K vizinhos mais próximos de um ponto z se dá por meio do algoritmo K-D tree [8]. Ele surge como uma alternativa ao algoritmo de força bruta, que precisa checar a distância do ponto z para cada um dos pontos x_i , o que pode ser muito custoso computacionalmente, principalmente se o espaço tiver um número grande de dimensões ou grande quantidade de dados. O K-D tree particiona o espaço previamente de maneira que, no geral, não é necessário visitar todos os pontos para encontrar o vizinho mais próximo de um novo ponto (porém, no pior dos casos, pode acontecer de ser necessário visitar todos os pontos) [9]. Uma observação interessante é que, o nome K-D tree vem originalmente de K-dimensional tree. Logo, esse “K” indica a dimensão dos dados com os quais estamos trabalhando.

Primeiramente, é necessário construir uma árvore binária em que cada nó representa um ponto do conjunto inicial de dados x_i . Para entender como é feita a decisão de qual ponto representará qual nó da árvore, comecemos a montar a árvore pela raiz. Depois, escolhemos arbitrariamente uma das dimensões do espaço e encontramos a mediana dos pontos referente a essa dimensão. O ponto que representa a mediana nessa dimensão é o ponto escolhido para representar o nó raiz. Por exemplo, se tivermos pontos $x_1 = (1 \ 1 \ 1)$, $x_2 = (2 \ 1 \ 1)$ e $x_3 = (3 \ 1 \ 1)$, ao escolhermos o eixo referente à primeira dimensão, veremos que o ponto que representa a mediana (com valor 2) nessa dimensão é x_2 e, por isso, ele seria o ponto representante do nó raiz. Ao escolher o ponto do nó raiz, segmentamos o espaço em

duas regiões, por meio de um hiperplano ortogonal ao eixo da dimensão escolhida e passando pelo ponto escolhido. A Figura 3-3 ilustra essa primeira etapa para o caso de duas dimensões.

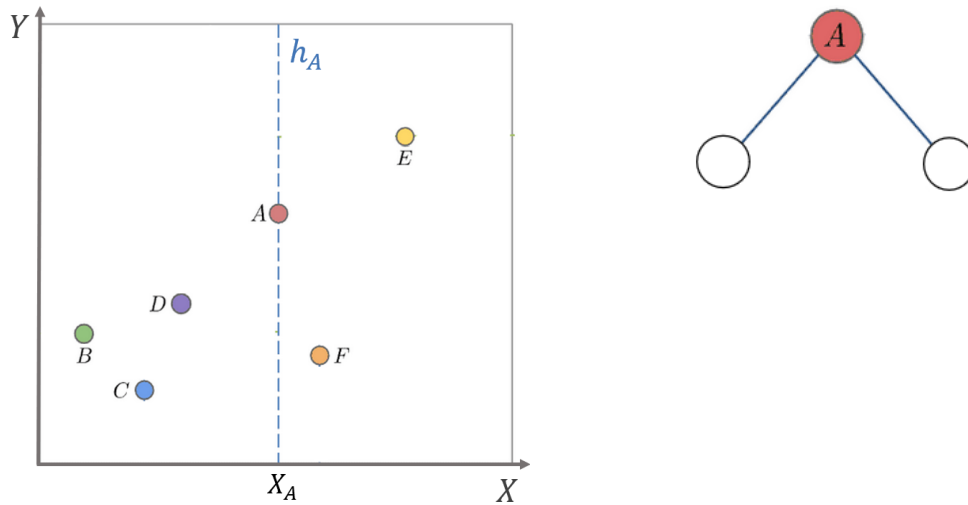


Figura 3-3: Primeira etapa da construção da K-D tree.

Após essa primeira etapa, o nó raiz é dividido em dois nós filhos, mas ainda não sabemos a quais pontos eles estão associados. Então, repetimos o procedimento descrito anteriormente, só que dessa vez, trocamos a dimensão que iremos considerar. No caso da Figura 3-3, olhamos primeiro para o eixo X , então agora olharemos para o eixo Y . Claramente, na porção do espaço à esquerda do hiperplano h_A , o ponto que representa a mediana em Y é o B . Do lado direito, a escolha é arbitrária, então escolhemos o ponto E . Com isso, dividimos cada uma das regiões que tínhamos anteriormente em duas, como mostra a Figura 3-4.

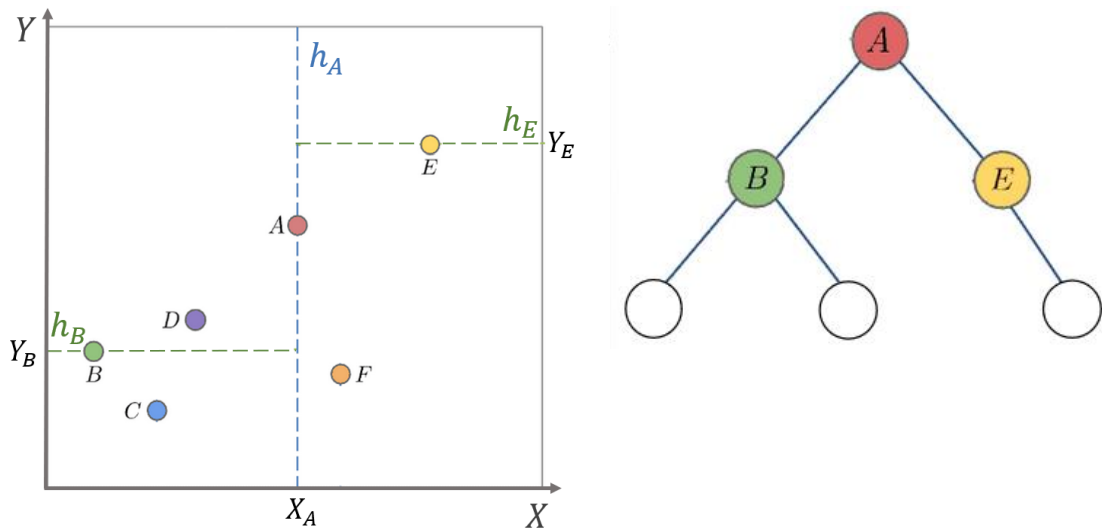


Figura 3-4: Segunda etapa da construção da K-D tree.

Esse procedimento é repetido analogamente para os demais nós da árvore, sempre alternando os eixos considerados e traçando de uma vez todos os hiperplanos correspondentes a nós de uma mesma profundidade da árvore. O resultado da terceira e última etapa do exemplo em duas dimensões é mostrado na Figura 3-5.

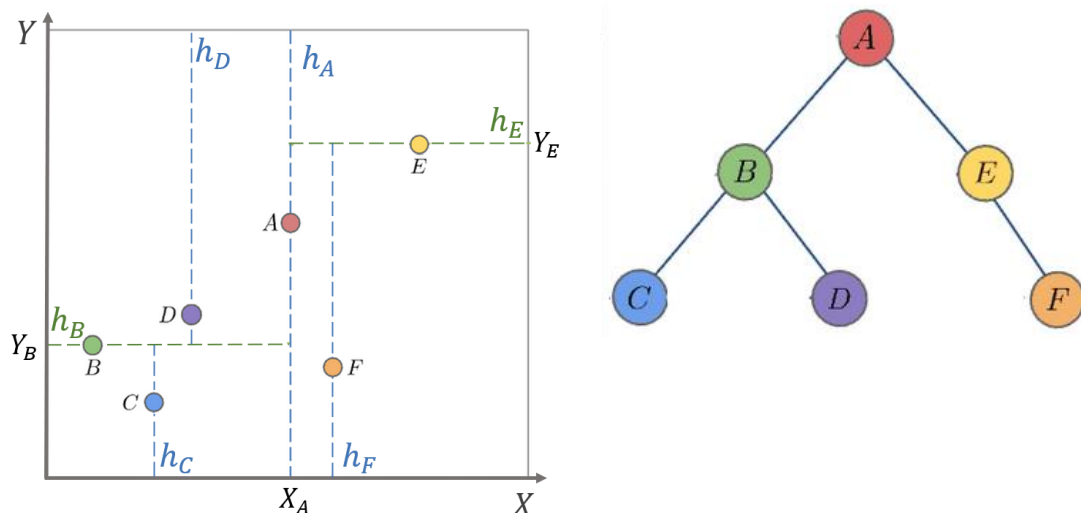


Figura 3-5: Última etapa da construção da K-D tree.

Construída a árvore, podemos utilizá-la para encontrar o vizinho mais próximo de um novo ponto z . Para isso, começando na raiz, percorremos a árvore descendo para o nó esquerdo sempre que o ponto estiver à esquerda do hiperplano vertical ou abaixo do hiperplano horizontal e descendo para o nó da direita caso contrário, até chegar a uma folha. Por exemplo, considere o ponto z , da Figura 3-6. Ao percorremos a árvore, fazemos o seguinte caminho: da raiz para B, pois z está à esquerda de h_A ; de B para C, pois z está abaixo de h_B . No entanto, o ponto C não é o vizinho mais próximo. Repare que, mesmo z estando na partição abaixo de h_B e à esquerda de h_A , podem existir pontos próximos desses hiperplanos que estão mais próximos. De fato, é o que acontece. Reparar que, apesar de o ponto D estar localizado em outra partição, ele está mais próximo de z do que C. Para resolver esse problema, o algoritmo exige que voltemos em outros nós da árvore para conferir se não existe outro vizinho mais próximo.

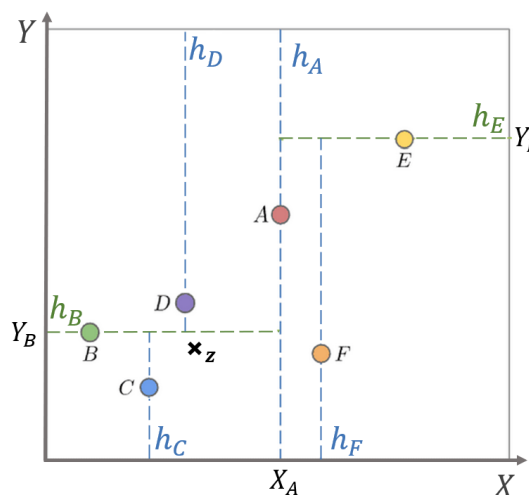


Figura 3-6: Procura pelo vizinho mais próximo de z .

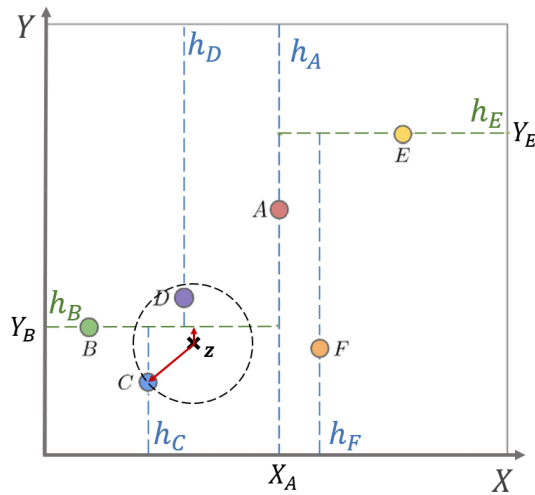


Figura 3-7: Checagem de interseção com outros hiperplanos de separação

Primeiramente, após a primeira descida da raiz até um nó folha, armazenamos o valor da distância do ponto z até o ponto C, ou seja, $D(C, z)$, que é o ponto mais próximo no momento. Depois, subimos um nó na árvore, indo para o nó B, como $D(B, z) > D(C, z)$, mantemos C como vizinho mais próximo. Como $D(C, z) > D(z, h_B)$ (Figura 3-7), é possível que haja pontos acima desse hiperplano que deveríamos considerar. Por isso, é necessário descer para a

subárvore da direita, onde está o nó D. Como $D(C, \mathbf{z}) > D(D, \mathbf{z})$, então o novo vizinho mais próximo é D.

Para decidir se vale a pena continuar percorrendo a árvore e subir para o nó raiz, verificamos a distância de \mathbf{z} para h_A . Como $(\mathbf{z}, h_A) > D(\mathbf{z}, D)$, é impossível que haja pontos à direita de h_A que estejam mais próximos de \mathbf{z} do que D. Então, encerramos o algoritmo e o vizinho mais próximo de \mathbf{z} é o ponto D. Com isso, vemos que foi possível encontrar o vizinho mais próximo sem necessariamente ter que calcular a distância de \mathbf{z} para todos os pontos do conjunto de dados.

É interessante notar que, em algumas implementações desse algoritmo, é possível estabelecer uma profundidade máxima que a árvore pode chegar ou o número máximo de pontos que se pode ter em uma folha. Nesse caso, ao chegar ao nó folha, o algoritmo precisa comparar a distância do ponto \mathbf{z} para todos os pontos abrangidos por essa folha e encontrar o vizinho mais próximo dentro dessa sub-região. Essa forma alternativa de executar o K-D *tree* surge casos em que a árvore é muito grande, o que pode ocupar muito espaço na memória se o algoritmo tiver que considerar um ponto por folha [10].

Ao analisarmos a complexidade desse algoritmo, considerando um caso genérico em que temos K vizinhos, notamos que, para construir a árvore, temos uma complexidade $O(K \times N \times \log N)$ na média [11], sendo N o número total de pontos (dependendo do algoritmo utilizado para calcular a mediana, a complexidade pode aumentar). Por outro lado, dado que a árvore já foi construída, a complexidade para encontrar o vizinho mais próximo de um ponto \mathbf{z} é $O(K \times \log N)$ em média. Com isso, vemos que a implementação por K-D *tree*, na média, tem uma complexidade menor do que por força bruta.

Comparado com o método de força bruta, a procura de vizinhos mais próximos pela K-D *tree* é mais rápido, durante a inferência, na maior parte dos casos – a complexidade é de $O(K \times N)$ para a força bruta e $O(K \times \log N)$ para a K-D *tree*.

Outro ponto interessante a ser observado é que, para espaços de dimensão muito elevada, o algoritmo K-D *tree* perde eficiência e

3.1.3 Ball Tree

O *ball tree* [12] é outro algoritmo de árvore que pode ser utilizado na tarefa de encontrar o vizinho mais próximo. Ele é uma alternativa ao K-D *tree* quando o número de dimensões do problema é muito grande. Os dois métodos se baseiam no mesmo princípio de segmentar o espaço dos dados e separá-los em subgrupos, representados por cada nó da árvore, porém, em vez de criar partições com fronteiras ortogonais aos eixos, o *ball tree* divide o espaço criando regiões circulares.

Para entender a construção dessa árvore, considere o conjunto de dados mostrado na Figura 3-8. Primeiramente, calculamos a média \mathbf{m}_0 dos pontos do conjunto de dados e encontramos o ponto mais distante dessa média, que está a uma distância D_0 de \mathbf{m}_0 . Em seguida, traçamos um círculo com centro em \mathbf{m}_0 e raio D_0 (Figura 3-9).

Após isso, selecionamos aleatoriamente um ponto \mathbf{x}_0 do conjunto de dados e encontramos o ponto \mathbf{x}_1 mais distante dele. Após isso, encontramos o ponto \mathbf{x}_2 mais distante de \mathbf{x}_1 e traçamos um segmento de reta auxiliar entre eles (Figura 3-10). Em seguida, realizamos a projeção de todos os pontos sobre esse segmento auxiliar e calculamos a mediana dessas projeções (Figura 3-11). Separamos os dados em dois grupos: dados cujas projeções estão

antes da mediana (grupo G_a) e depois dela (grupo G_d). Para cada um desses grupos, calculamos a média dos pontos (Figura 3-12). Em cada grupo, encontramos o ponto mais distante da média e traçamos um círculo centrado na média e com raio igual à distância entre a média e o ponto mais distante (Figura 3-13). Esse procedimento é repetido, gerando novos círculos até que cada um contenha apenas um ponto.

A árvore resultante é tal que cada nó representa um desses círculos, guardando a localização do centro e seu raio. Assim, o nó raiz representa o primeiro círculo, que engloba todos os pontos, os dois nós filhos representam os dois círculos gerados a partir do primeiro e assim por diante até chegar a uma folha, que representa um círculo contendo um único ponto.

Construída a árvore, ela pode ser utilizada para encontrar o vizinho mais próximo de um novo ponto \mathbf{z} . O processo é análogo ao processo de procurar o vizinho mais próximo da *K-D tree*. Começamos na raiz e, a cada nó, precisamos decidir se descemos para o nó filho da direita ou da esquerda. Se o centro do círculo do filho esquerdo for mais próximo do que o do direito, seguimos para o filho esquerdo. Caso contrário, seguimos para o filho direito. Ao chegar a um nó folha, temos um candidato a vizinho mais próximo, que será o ponto representado por essa folha. No entanto, assim como acontece na *K-D tree*, não é possível ter certeza de que esse ponto da folha é o vizinho mais próximo, pois podem existir pontos mais próximos fora do círculo atual que não foram considerados. Para isso ser possível, é necessário que a distância de \mathbf{z} para qualquer outro círculo (de qualquer nó e de qualquer profundidade) seja menor do que a distância de \mathbf{z} para o ponto mais próximo no nó folha. Matematicamente, há a possibilidade de existir um vizinho mais próximo em um outro círculo C_2 se $D(\mathbf{z}, \mathbf{x}_1) > D(\mathbf{z}, \mathbf{c}_2) - r_2$, em que \mathbf{c}_2 e r_2 são o centro e o raio de C_2 , respectivamente, e \mathbf{x}_1 o vizinho mais próximo até o momento. $D(\mathbf{z}, \mathbf{c}_2) - r_2$ representa a distância de \mathbf{z} até a margem de C_2 .

Com relação à complexidade de tempo, a construção da árvore tem uma complexidade $O(d \times N \times \log(N))$ e o processo de encontrar os K vizinhos mais próximos, dado que a árvore já está construída, tem complexidade $O(K \times \log(N))$ [13]. Apesar de apresentar a mesma complexidade de tempo que o algoritmo *K-D tree*, para espaços de maior dimensão, o *ball tree* tende a ser mais rápido que o *K-D tree*, devido a um problema relacionado à maldição da dimensionalidade, conforme será explicado adiante [14].

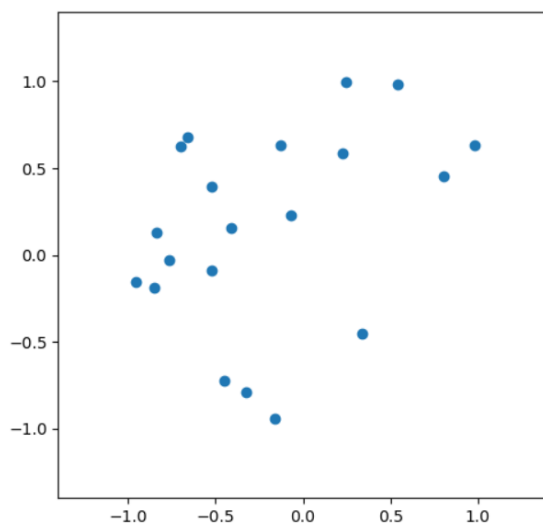


Figura 3-8: Conjunto de dados de exemplo para construção da ball tree.

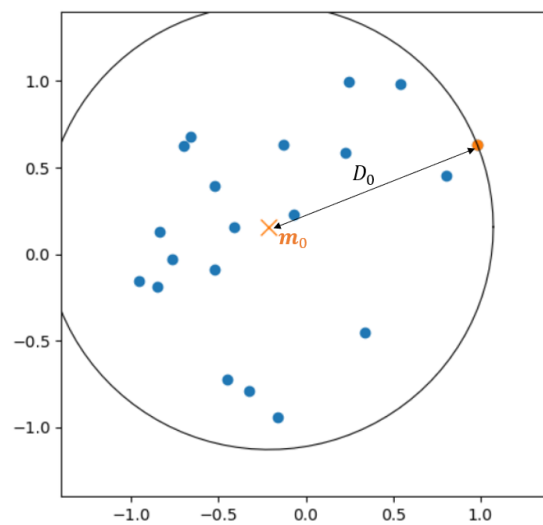


Figura 3-9: Primeiro círculo da árvore, representando o nó raiz.

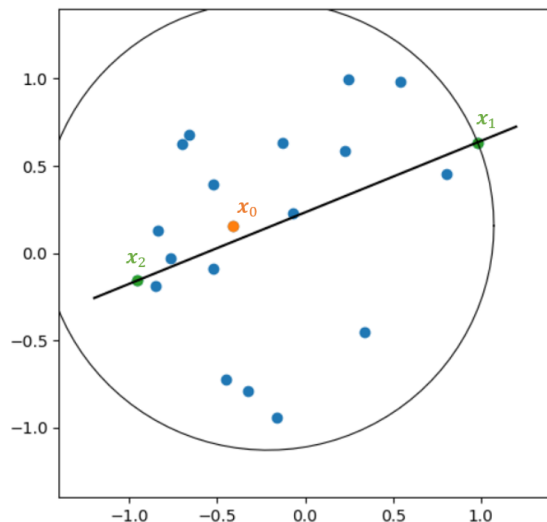


Figura 3-10: Segmento de reta auxiliar entre pontos mais distantes.

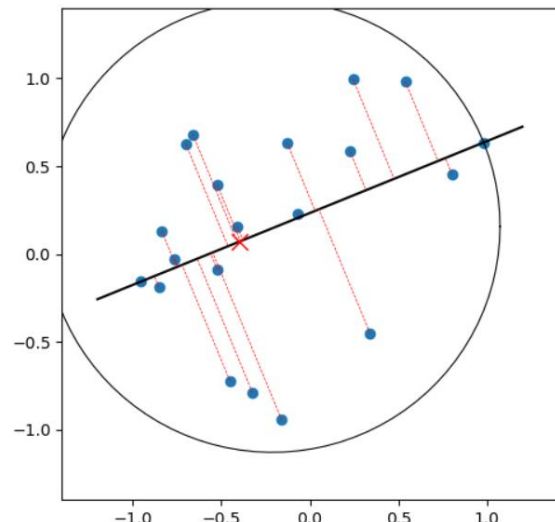


Figura 3-11: Mediana das projeções representada por um X vermelho.

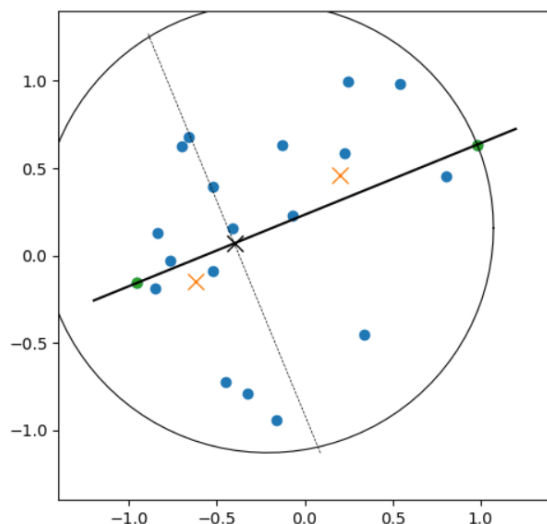


Figura 3-12: Médias dos grupos G_a e G_d representadas em amarelo.

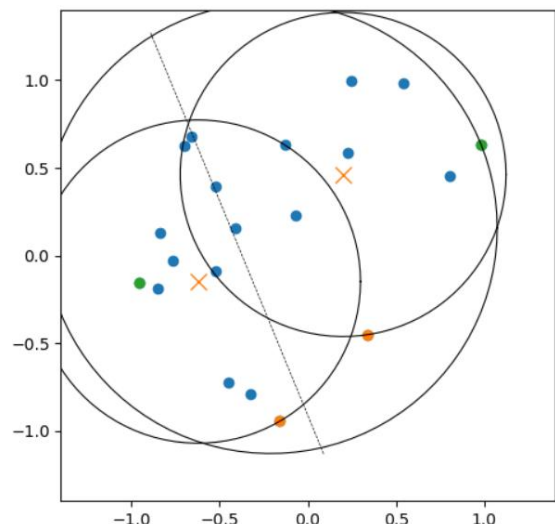


Figura 3-13: Círculos centrados na média, passando pelo ponto mais distante de cada grupo.

3.1.4 Variações do KNN

Uma variação ao método tradicional do KNN consiste em ponderar a importância dos vizinhos considerados de acordo com suas respectivas distâncias para um determinado ponto de teste z . Normalmente, nesse caso, são atribuídos pesos proporcionais ao inverso da distância de cada ponto até z . Dessa forma, é possível fazer com que a contribuição de pontos mais próximos no resultado da predição seja maior do que a de pontos mais distantes.

Outro método com filosofia similar ao do KNN é o **Radius Nearest Neighbors** [4], que em vez de encontrar um número fixo de vizinhos de um determinado ponto de teste z , encontra todos os vizinhos dentro de uma hipersfera centrada em z e de raio fixo. Quando o problema conta com dados de densidade muito desuniforme, é preferível utilizar esse método do que o tradicional KNN. Isso porque nesse tipo de problema, nas áreas mais densas do conjunto de dados, os K vizinhos mais próximos de z tendem a estar a uma distância menor e constante dele, mas em regiões mais esparsas desse mesmo conjunto de dados, os K vizinhos mais próximos podem incluir pontos muito distantes de z , que são pouco semelhantes a ele. Ao

fixar um valor de raio em volta de \mathbf{z} , garantimos que só haverá vizinhos até uma determinada distância.

3.2 MALDIÇÃO DA DIMENSIONALIDADE

Os algoritmos de KNN são baseados na suposição de que pontos similares possuem *labels* similares, o que faz com que eles sejam especialmente afetados pela maldição da dimensionalidade. Isso acontece porque, para um mesmo número N de amostras sorteadas aleatoriamente uniformemente, a distância entre essas amostras tende a aumentar à medida que o número de dimensões cresce.

Para entender como isso afeta os algoritmos de KNN, imaginemos que os dados estão restritos a um espaço representado por um hipercubo unitário, isto é, $[0,1]^d$ e que estamos considerando os $K = 10$ vizinhos mais próximos de um certo ponto de teste. Consideremos também que ℓ é o tamanho da aresta do menor hipercubo que engloba todos os K vizinhos mais próximos do ponto de teste (Figura 3-14). A aresta desse hipercubo pode ser escrita como $\ell \approx \left(\frac{K}{N}\right)^{1/d}$ [15]. Utilizando essa equação, notamos que, à medida que aumentamos d , a aresta ℓ também aumenta (Tabela 3-1). No entanto, o espaço é limitado a um hipercubo de aresta 1 e, conseqüentemente, volume igual a 1, ou seja, à medida que o espaço aumenta em dimensão, maior é a proporção do espaço necessária para o hipercubo de aresta ℓ englobar os K vizinhos.

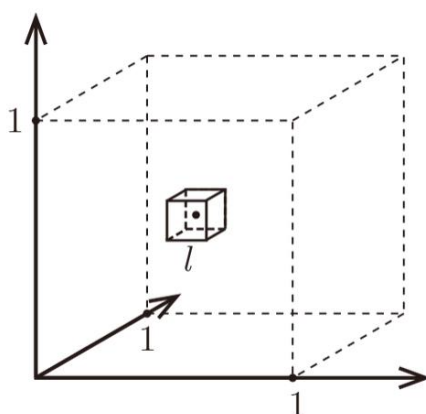


Figura 3-14: Hipercubo de aresta 1 representando o espaço $[0,1]^d$ e o hipercubo de aresta ℓ .

d	ℓ
2	0,1
3	0,215
10	0,63
100	0,955
1000	0,9954

Tabela 3-1: Variação de ℓ à medida que aumentamos a dimensão d do espaço.

Outro gráfico que ilustra esse fenômeno é o mostrado na Figura 3-15. Com isso vemos que, por mais que o espaço seja restrito a um hipercubo unitário, as distâncias dentro dele crescem à medida que acrescentamos uma nova dimensão. É importante lembrar que a maior distância dentro de um hipercubo unitário é dada por \sqrt{d} .

Por isso, como para valores muito elevados de d há uma tendência de não haver pontos muito próximos, hipótese de que pontos próximos possuem a mesma *label* perde o sentido. No entanto, mesmo que o espaço tenha dimensão muito elevada, caso os dados estejam concentrados em um subespaço de menor dimensão, ainda pode ser possível aplicar algoritmos de KNN. Um exemplo que ilustra essa situação é mostrado na Figura 3-16.

Uma solução para esse problema da esparsidade dos dados em dimensões maiores poderia ser resolvido ao aumentar o número N de amostras. Para saber a quantidade de amostras ideal, poderíamos usar como critério o tamanho do menor hipercubo que engloba os K vizinhos e estabelecer $\ell = 0,1$. Com isso, como $\ell \approx \left(\frac{K}{N}\right)^{1/d}$, poderíamos escrever $N = \frac{K}{\ell^d} = 10^d K$. Logo, vemos que o número de amostras necessário cresce exponencialmente, o que inviabiliza a coleta de novas amostras para valores elevados de d .

Esse problema relacionado à maldição da dimensionalidade não é exclusivo de um algoritmo de KNN, a implementação por força bruta, K-D *tree* e *ball tree* estão sujeitos a ele. No entanto, a K-D *tree* ainda enfrenta um problema adicional relacionado à perda de eficiência no que se refere ao aumento da dimensão do espaço.

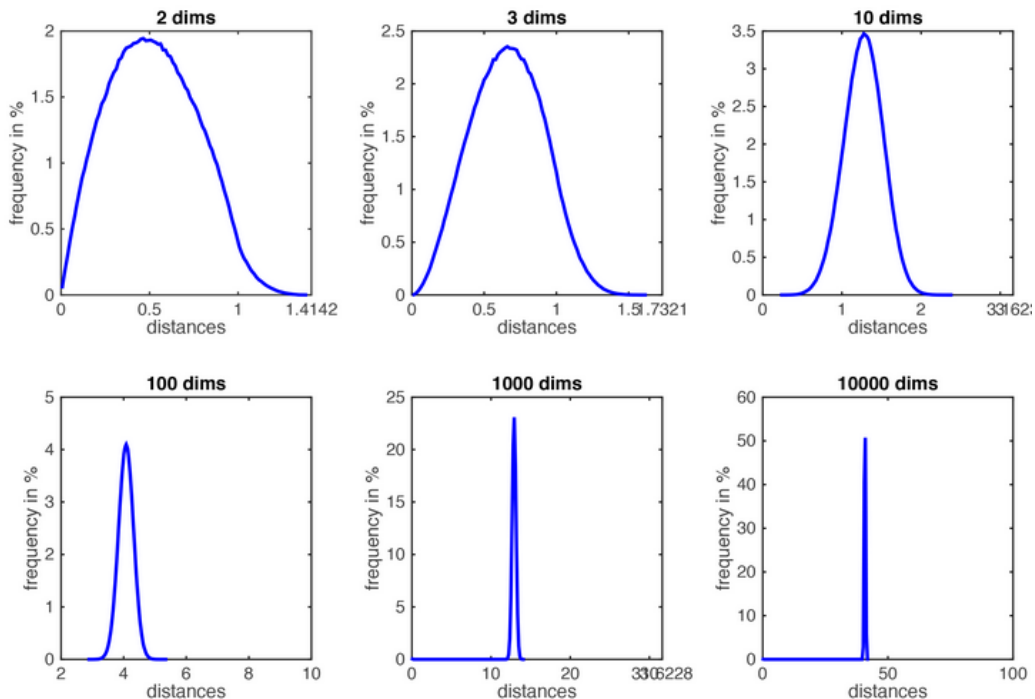


Figura 3-15: Histogramas das distâncias entre pares de pontos para espaços de diferentes dimensões.

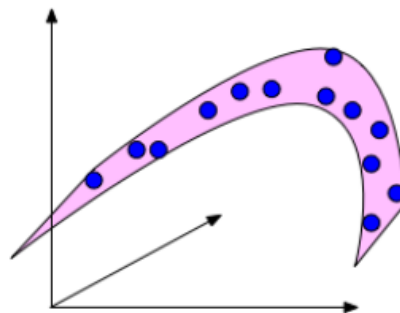


Figura 3-16: Subespaço de dimensão 2 em um espaço de 3 dimensões.

Como visto anteriormente, o algoritmo *K-D tree* se baseia em criar hiperplanos de separação ortogonais aos eixos para segmentar o espaço em partições. À medida que o algoritmo é executado, é necessário conferir se a distância entre o ponto mais próximo dentro de uma partição D_{nn} é maior do que a distância do ponto teste até um dos hiperplanos D_h . Sendo maior, é necessário fazer mais cálculos de distância, procurando por pontos mais próximos em outras partições. Considerando ainda a situação em que o espaço é um hipercubo unitário, conforme aumentamos d , a maior distância possível dentro desse hipercubo (\sqrt{d}) aumenta, porém, as distâncias ao longo dos eixos serão sempre limitadas ao intervalo $[0, 1]$, já que as arestas do hipercubo são unitárias. Com isso, à medida que d aumenta, as distâncias entre os pontos e os hiperplanos se mantêm, mas as distâncias entre pares de ponto aumentam (Figura 3-17), o que tende a aumentar o número de casos em que $D_h < D_{nn}$. Isso aumenta o número de cálculos de distância necessários, o que torna o algoritmo *K-D tree* mais lento.

O algoritmo *ball tree*, por outro lado, não sofre com esse problema, pois as partições criadas são circulares e não são dependentes dos eixos, apenas das distâncias entre os pontos. Por isso, em espaços com número muito elevado de dimensões, o algoritmo *ball tree* apresenta uma melhor performance comparado o *K-D tree*.

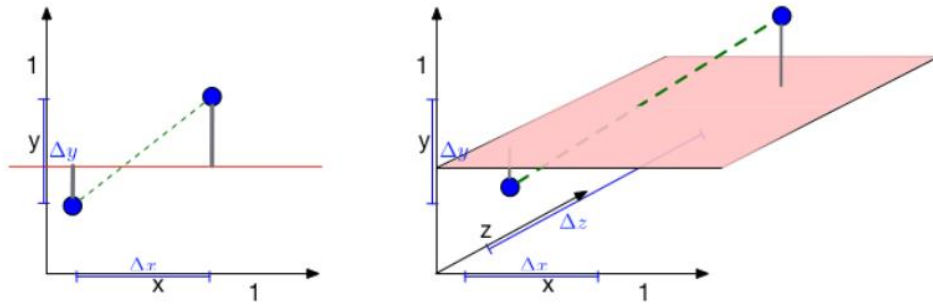


Figura 3-17: Efeito do aumento de dimensão nas distâncias de um espaço.

3.3 MÉTRICAS DE DISTÂNCIA

Algoritmos de KNN são baseados nas distâncias entre pontos. Normalmente, quando falamos em distância, pensamos primeiramente na distância Euclidiana, que é a que faz mais parte do nosso dia a dia. No entanto, existem outras métricas que podem ser utilizadas para medir distância, que podem ser mais adequadas do que a Euclidiana dependendo da situação.

Uma métrica bastante conhecida é a **métrica de Minkowski**. Para um espaço de dimensão q , podemos escrevê-la como

$$\|x - y\|_p = \left(\sum_{i=1}^q |x_i - y_i|^p \right)^{1/p}.$$

Como podemos ver, ela possui uma equação geral, que varia conforme mudamos o valor de p . Para alguns valores específicos de p , essa equação pode receber um nome especial como mostrado abaixo:

Distância de Manhattan ($p = 1$)	$\ x - y\ _1 = \sum_{i=1}^q x_i - y_i $
------------------------------------	--

Distância Euclidiana ($p = 2$)

$$\|x - y\|_2 = \sqrt{\sum_{i=1}^q (x_i - y_i)^2}$$

A Figura 3-18 compara as distâncias de Manhattan e Euclidiana.

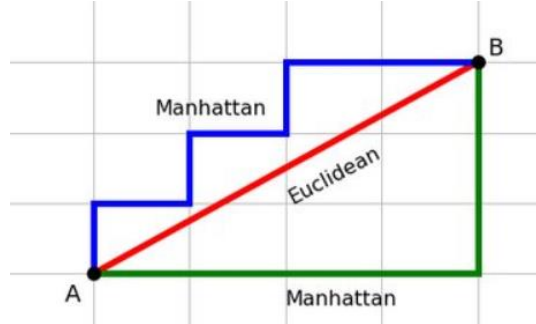


Figura 3-18: Comparação entre as distâncias de Manhattan e Euclidiana.

Como podemos ver, para o cálculo da distância de Manhattan, desconsideramos a possibilidade de realizar caminhos na diagonal entre A e B, apenas caminhos paralelos aos eixos. De uma maneira mais prática, considerando as arestas de cada quadrado/retângulo de *grid* como um passo, a distância de Manhattan é equivalente ao número mínimo de passos para ir de um ponto a outro. Ela encontra maior utilidade quando tratamos de problemas em que faz sentido considerarmos a possibilidade de deslocamento apenas em direções paralelas aos eixos. Um exemplo, é quando o problema envolve *features* representando dimensões espaciais de uma cidade, em que não se pode atravessar um quarteirão.

Outra métrica que pode ser utilizada é a distância de Hamming, que é dada por

$$D_H(x, y) = \sum_{i=1}^p (x_i + y_i \bmod 2),$$

com x e y sendo vetores binários. Em outras palavras, a distância de Hamming calcula o número total de valores diferentes entre duas sequências de bits. Por exemplo, se $x = (1 \ 0 \ 1)$ e $y = (1 \ 1 \ 1)$, então, $D_H = 2$, pois o primeiro e o último bits de x e y são diferentes. Pode fazer sentido utilizar essa métrica quando temos *features* binárias. Nesse caso, teríamos um espaço de *features* similar ao mostrado em.

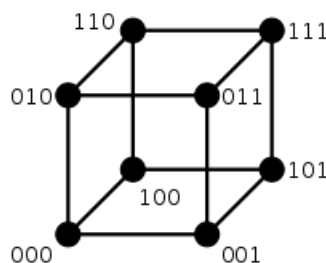


Figura 3-19: Espaço de Hamming mostrando coordenadas binárias.

Por fim, outra métrica conhecida é a distância de Chebyshev. Ela é dada por $\max_i |x_i - y_i|$, em que $x = (x_1 \ x_2 \ \dots)$ e $y = (y_1 \ y_2 \ \dots)$ são dois pontos. De maneira intuitiva, se considerarmos um jogo de xadrez podemos dizer que todas as casas que o rei pode acessar em

um turno estão a uma distância de Chebyshev de uma unidade. Além disso, ainda nesse exemplo, a distância de Chebyshev do rei para qualquer casa do tabuleiro é igual ao número de turnos necessários para essa peça chegar à tal casa Figura 3-20.


	a	b	c	d	e	f	g	h	
8	5	4	3	2	2	2	2	2	8
7	5	4	3	2	1	1	1	2	7
6	5	4	3	2	1		1	2	6
5	5	4	3	2	1	1	1	2	5
4	5	4	3	2	2	2	2	2	4
3	5	4	3	3	3	3	3	3	3
2	5	4	4	4	4	4	4	4	2
1	5	5	5	5	5	5	5	5	1
	a	b	c	d	e	f	g	h	

Figura 3-20: Distância de Chebyshev de cada casa de um tabuleiro de xadrez para o rei.

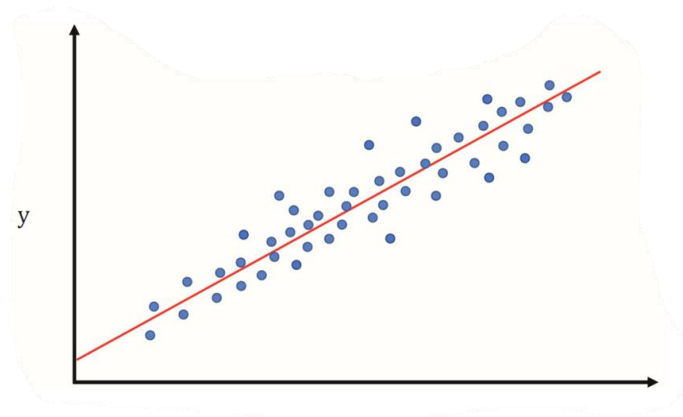
Capítulo 4: REGRESSÃO LINEAR

O problema de regressão linear consiste em criar um modelo que consiga relacionar variáveis $\{x_m\}_{m=1}^M$ com uma saída y de referência por meio de uma equação linear da seguinte forma:

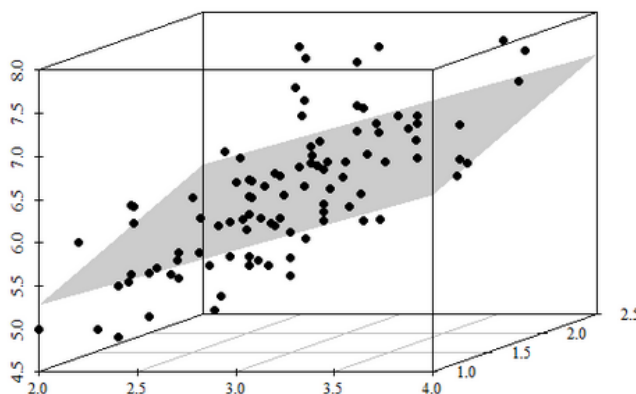
$$y = \theta_0 + \sum_{m=1}^M \theta_m x_m = \boldsymbol{\theta}^T \mathbf{x},$$

em que $\boldsymbol{\theta} = [\theta_M \ \theta_{M-1} \ \dots \ \theta_0]^T$, $\mathbf{x} = [x_M \ x_{M-1} \ \dots \ x_1]^T$ e $\mathbf{x} = [x_M \ x_{M-1} \ \dots \ x_1]^T$. O termo θ_0 recebe é conhecido como *bias* ou *intercept*.

No caso em que temos apenas uma variável x , o problema consiste em encontrar a reta descrita pela equação $y = \boldsymbol{\theta}^T \mathbf{x} = \theta_1 x_1 + \theta_0$, que aproximar se ajustar aos pontos (x, t) , em que t é o valor de referência da variável x . Assim, o objetivo é que, com essa equação, consigamos fazer com que y da melhor maneira possível os valores de referência t , como ilustra a figura abaixo.



Quando temos um problema em que há duas variáveis de entrada, ou seja, $\mathbf{x} \in \mathbb{R}^2$, o problema é parecido, mas, para representá-lo visualmente, precisamos de um espaço de três dimensões, como mostrado na figura abaixo.



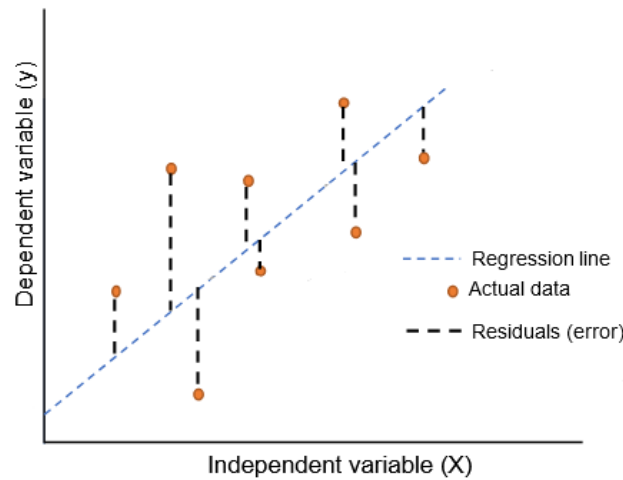
Com isso, podemos perceber que o problema da regressão linear consiste em encontrar o *hiperplano* que melhor descreve a relação entre as variáveis x_m e a saída y . O caso da reta e do plano mostrados acima são apenas casos particulares do hiperplano em que os vetores \mathbf{x} pertencem ao \mathbb{R}^1 e ao \mathbb{R}^2 , respectivamente.

Agora, como determinar os valores de θ que definem esse hiperplano? A seguir, são mostradas diferentes interpretações desse problema que levam a mesma solução.

4.1 DIFERENTES INTERPRETAÇÕES DA REGRESSÃO LINEAR

4.1.1 Ponto de Vista de Otimização

Uma maneira de resolver esse problema é encontrar o θ tal que a diferença entre os valores preditos y_n e o valores de referência t_n sejam os menores possíveis, levando em consideração todos os pares $(x_n, t_n)_{n=1}^N$ existentes no conjunto de dados. Essa diferença é comumente chamada de **erro** ou **resíduo**. No caso em que $x_n \in \mathbb{R}$ temos algo similar ao mostrado na imagem abaixo:



Assim, queremos que a soma de todos os resíduos em módulo seja a menor possível, ou seja, queremos algo como $\min_{\theta} \sum_{n=1}^N |y_n - t_n|$, em que $y_n = f(\theta)$. No entanto, se levarmos em consideração que para utilizar métodos tradicionais de minimização teríamos que derivar essa expressão, percebemos que talvez a **loss** $|y_n - t_n|$ não seja a melhor escolha.

Uma alternativa seria utilizar como **loss** o erro quadrático $(y_n - t_n)^2$ que é facilmente derivável e não altera o objetivo final, pois minimizar a soma dos erros quadráticos é equivalente a minimizar o módulo dos resíduos (o ponto de ótimo será o mesmo).

Com isso, adotamos como **função custo** a ser minimizada a função descrita abaixo:

$$J(\theta) = \frac{1}{2} \sum_{n=1}^N (y_n - t_n)^2.$$

O termo $\frac{1}{2}$, que ainda não tinha sido considerado, surge apenas para simplificar futuras derivações, em que o 2 advindo do expoente cancela com o termo $\frac{1}{2}$. Novamente, isso não altera o resultado final, pois

$$\theta = \arg \min_{\theta} \sum_{n=1}^N (y_n - t_n)^2 = \arg \min_{\theta} \frac{1}{2} \sum_{n=1}^N (y_n - t_n)^2.$$

Para encontrar valor de θ^* que minimiza a função custo, derivamos e igualamos a zero:

$$\frac{\partial J(\theta)}{\partial \theta} = \nabla J(\theta) = \sum_{n=1}^N \frac{\partial}{\partial \theta} \left(\frac{\theta^T x_n}{\widehat{y}_n} - t_n \right) = \sum_{n=1}^N x_n^T (\theta^T x_n - t_n) = \mathbf{0}$$

$$\sum_{n=1}^N x_n^T \theta^T x_n = \sum_{n=1}^N x_n^T t_n$$

Com isso, vemos que o θ ótimo é aquele que satisfaz a equação acima.

Na prática, uma forma de se resolver esse problema é utilizar o algoritmo do gradiente descendente, em que escolhemos um θ inicial aleatório para ser nosso ponto de partida e, a cada iteração, movemos esse ponto na direção de máxima decida, ou seja, a cada iteração, fazemos

$$\theta \leftarrow \theta - \alpha \nabla J(\theta),$$

em que α é a taxa de aprendizado e controla o tamanho do passo que será dado na direção de maior descida.

Apesar de podermos aplicar o método do gradiente descendente, há um outro método muito mais direto e que resolve o problema da regressão linear em uma iteração apenas, como veremos a seguir.

Observação: Loss function x Cost function

Loss function

Função utilizada para medir o erro entre o valor de referência t e o valor predito y . Nesta seção, foram vistos dois exemplos: o erro absoluto $|y - t|$ e o erro quadrático $(y - t)^2$.

Cost function

Função $J(\theta)$ utilizada para medir o custo total por se adotar $\theta = \theta'$ como parâmetro, considerando todas as N observações do conjunto de dados D . Muitas vezes, algumas pessoas usam de maneira intercambiável os termos *loss function* e *cost function*, apesar de serem diferentes.

Exemplo:

$$\overbrace{\frac{1}{2} \sum_{n=1}^N (y_n - t_n)^2}^{\text{cost}}$$

loss

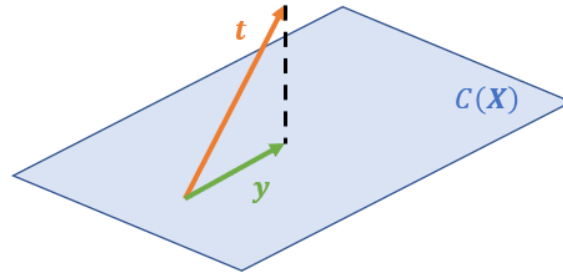
4.1.2 Ponto de Vista de Álgebra Linear

O nosso problema de relacionar variáveis x_m a uma saída y pode ser representado de maneira vetorial por meio da seguinte equação

$$y = X\theta,$$

em que $y = [y_1 \quad \cdots \quad y_N]^T$ e $X = \begin{bmatrix} x_{1M} & \cdots & x_{11} & 1 \\ x_{2M} & \cdots & x_{21} & 1 \\ \vdots & \ddots & \ddots & \vdots \\ x_{NM} & \cdots & x_{N1} & 1 \end{bmatrix}$.

Idealmente, desejaríamos que $X\theta = t$, em que $t = [t_1 \quad \cdots \quad t_N]^T$, porém, normalmente, essa equação não tem solução, pois $t \notin C(X)$, ou seja, t não pertence ao espaço coluna de X . Uma possível solução para esse problema é encontrar um vetor $y \in C(X)$ que está mais próximo de t do que qualquer outro vetor, ou seja, queremos $\theta^* = \arg \min_{\theta} \|X\theta - t\|$. Evidentemente, esse vetor $y = X\theta$ precisa ser necessariamente a projeção de t em $C(X)$, como ilustra a figura abaixo.



Com auxílio dessa figura, reparamos que $y - t = X\theta^* - t$ é ortogonal a $C(X)$, isto é, $X\theta^* - t \in C(X)^\perp$. Como consequência disso, temos que o produto interno de $X\theta^* - t$ por qualquer vetor da coluna de X é nulo, ou seja, $X^T(X\theta^* - t) = 0$. Logo, chegamos à equação

$$X^T X \theta^* = X^T t,$$

que possui uma solução, diferentemente de $X\theta = t$. Se $X^T X$ for invertível (que normalmente é, a não ser que $X\theta = t$ admita várias soluções, o que não é o caso no nosso problema), podemos escrever

$$\theta^* = (X^T X)^{-1} X^T t.$$

Assim, conseguimos resolver o problema da regressão linear com uma única equação em um único passo.

Bônus:

Se reparamos, o que desejamos é $\theta^* = \arg \min_{\theta} \|X\theta - t\|$, que é equivalente a

$$\theta^* = \arg \min_{\theta} \|X\theta - t\|^2 = \arg \min_{\theta} \sum_{n=1}^N (y_n - t_n)^2 = \arg \min_{\theta} \frac{1}{2} \sum_{n=1}^N (y_n - t_n)^2 = \arg \min_{\theta} J(\theta),$$

em que $J(\theta)$ é a função custo encontrada na Seção 4.1.1.

Além disso, essa equação poderia ter sido resolvida mais facilmente na forma matricial, fazendo

$$\nabla_{\theta} \|X\theta - t\|^2 = \nabla_{\theta} [(X\theta - t)^T (X\theta - t)] = (X\theta - t)^T X + (X\theta - t)^T X = 0$$

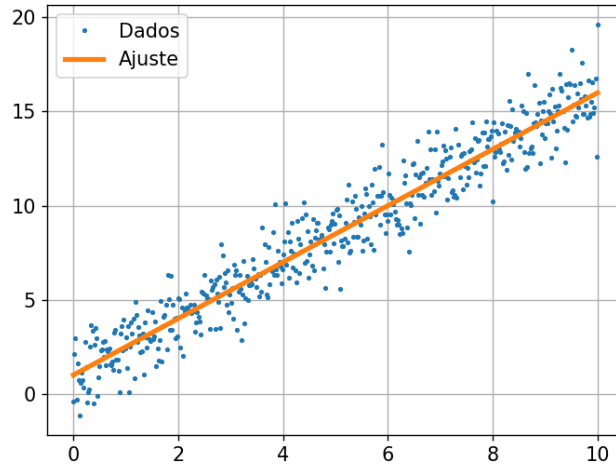
$$2(X\theta - t)^T X = 0$$

$$X^T (X\theta - t) = 0$$

$$\boldsymbol{\theta}^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{t}.$$

4.1.3 Ponto de Vista Probabilístico

Também é possível chegar ao mesmo resultado adotando uma abordagem probabilística. Para isso, assumimos que a equação que define a geração dos pontos é dada por $t = \boldsymbol{\theta}^T \mathbf{x} + \eta$, ou seja, é um hiperplano definido por $y = \boldsymbol{\theta}^T \mathbf{x}$ mais uma perturbação η . Na imagem abaixo, a reta em laranja representa o hiperplano $y = \boldsymbol{\theta}^T \mathbf{x}$ e os pontos em azuis são gerados por $t = \boldsymbol{\theta}^T \mathbf{x} + \eta$. O que gostaríamos de obter é a curva laranja, que é totalmente definida pelo parâmetro $\boldsymbol{\theta}$, porém, só conhecemos os dados $D = (\mathbf{x}_n, t)_{n=1}^N$, não conhecemos os valores de y_n correspondentes a \mathbf{x}_n .



Uma possível suposição é de que $\eta \sim \mathcal{N}(0, \sigma^2)$, o que faz com que $t \sim \mathcal{N}(y(\boldsymbol{\theta}), \sigma^2)$.

Assim, podemos aplicar a estimação de máxima verossimilhança (MLE) para determinar o parâmetro $\boldsymbol{\theta}$.

Para começar, sabemos que

$$p(t | y(\boldsymbol{\theta}), \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left[-\frac{(x - y(\boldsymbol{\theta}, \mathbf{x}))^2}{2\sigma^2} \right].$$

Se considerarmos todos os pares (\mathbf{x}_n, t_n) de D e considerarmos que eles são gerados de maneira independente, teremos

$$\mathcal{L}_N(\boldsymbol{\theta}) = p(\mathbf{t} | \mathbf{y}(\boldsymbol{\theta}), \sigma^2) = \prod_{n=1}^N p(t | y(\boldsymbol{\theta}, \mathbf{x}_n), \sigma^2)$$

Seguindo um procedimento similar ao visto na Seção 2.1, aplicamos a função log (base e) para encontrar a log verossimilhança:

$$\ell_N(\boldsymbol{\theta}) = \sum_{n=1}^N \log p(t | y(\boldsymbol{\theta}, \mathbf{x}_n), \sigma^2) = N \log \left(\frac{1}{\sqrt{2\pi\sigma^2}} \right) - \sum_{n=1}^N \frac{(x_n - y(\boldsymbol{\theta}, \mathbf{x}_n))^2}{2\sigma^2}$$

Com isso, passamos para a etapa de maximizar a log verossimilhança, o que é equivalente a minimizar o negativo da log verossimilhança. Normalmente, é preferível esse ponto de vista de “minimização” em vez de “maximização”, pois, eventualmente, podemos considerar o

negativo da log verossimilhança como uma função custo, que normalmente é uma função que visamos normalizar:

$$J(\boldsymbol{\theta}) = -\ell_N(\boldsymbol{\theta}).$$

Portanto, ao minimizarmos essa função custo obtemos

$$\begin{aligned}\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \min_{\boldsymbol{\theta}} -N \log \left(\frac{1}{\sqrt{2\pi\sigma^2}} \right) + \sum_{n=1}^N \frac{(x_n - y(\boldsymbol{\theta}, \mathbf{x}_n))^2}{2\sigma^2} \\ &= \min_{\boldsymbol{\theta}} \frac{1}{2} \sum_{n=1}^N (x_n - \boldsymbol{\theta}^T \mathbf{x}_n)^2,\end{aligned}$$

que é justamente à conclusão que tínhamos chegado na Seção 4.1.1 e que é equivalente a minimizar a função custo

$$J(\boldsymbol{\theta}) = \frac{1}{2} \sum_{n=1}^N (x_n - \boldsymbol{\theta}^T \mathbf{x}_n)^2.$$

4.2 REGRESSÃO POLINOMIAL

É uma modalidade da regressão linear em que temos uma variável de entrada x e a substituímos por variáveis $\phi_m(x) = x^m$ que são potências de x . Em geral, funções $\phi_m(x)$ substituem as variáveis antigas são chamadas de **funções de base**. Além disso, de forma geral, podemos chamar essas novas variáveis formadas pelas funções base de **atributos** ou **features**. Dependendo da interpretação, também podemos chamar as variáveis x_m de *features*, mas geralmente, o que temos nesse caso, na verdade, são funções base identidade na forma $\phi_m(x) = x_m$.

Normalmente, escolhemos essa abordagem quando não conseguimos modelar os dados com uma equação na forma

$$y = \theta_0 + \boldsymbol{\theta}^T \mathbf{x},$$

e precisamos recorrer a uma modelagem não linear da relação entre y e x ;

Assim, utilizamos um modelo na forma

$$y = \boldsymbol{\theta}^T \boldsymbol{\phi} = \sum_{m=0}^M \theta_m \phi_m = \sum_{m=0}^M \theta_m x^m,$$

em que M é o grau do polinômio e $\boldsymbol{\phi} = [x^M \quad x^{M-1} \quad \dots \quad 1]^T$.

Por mais que existam termos de ordem maior do que 1 e que a relação entre y e x seja não linear, ainda temos uma regressão linear, pois a relação entre y e os termos $\boldsymbol{\theta}$ é linear. Se considerarmos todos os dados, independentemente da relação que cada *feature* mantém entre elas, ainda temos uma equação linear $\mathbf{v}(\mathbf{w}) = \mathbf{A}\mathbf{w} + \mathbf{b}$, em que \mathbf{A} corresponde à matriz

de dados $\boldsymbol{\Phi} = \begin{bmatrix} \phi_{1M} & \dots & \phi_{11} & 1 \\ \phi_{2M} & \dots & \phi_{21} & 1 \\ \vdots & \ddots & \ddots & \vdots \\ \phi_{NM} & \dots & \phi_{N1} & 1 \end{bmatrix}$, a variável independente \mathbf{w} corresponde a $\boldsymbol{\theta}$ e $\mathbf{b} = \mathbf{0}$.

Por mais que essa seja a explicação oficial do porquê de essa regressão ainda ser linear, ainda acho mais lógico ela ser linear pelo fato de y ser linear com relação às *features* ϕ_m e podermos escrever uma equação linear $y = \theta^T \phi$. Fazendo uma comparação, nas seções anteriores, tínhamos um vetor de variáveis de entrada $\mathbf{x} = [x_M \ x_{M-1} \ \dots \ x_1]^T$ e uma equação similar: $y = \theta^T \mathbf{x}$. A princípio, não sabemos se as variáveis x_m são independentes entre si ou se elas têm algum tipo de relação entre elas. É possível que $x_2 = x_1^2$ (x_1 sendo a largura de um terreno quadrado e x^2 sendo a área desse terreno, por exemplo), que é o caso em que teríamos algo que lembra uma regressão polinomial e, mesmo assim, ainda temos uma regressão linear.

4.3 FORMA GERAL DA REGRESSÃO LINEAR

Percebemos ao longo deste capítulo que a regressão linear é um problema cujo objetivo é encontrar uma equação que relacione de maneira linear uma variável de saída y com variáveis $\phi_m(x)$ (que podem ser dependentes entre si ou não), por meio de uma equação na forma

$$y = \theta^T \phi,$$

sendo que o parâmetro ótimo é dado por

$$\theta^* = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{t}.$$

O caso da regressão polinomial é apenas um caso particular, em que forçamos a dependência entre as variáveis ϕ_m . No início do capítulo tínhamos outro caso particular, só que não forçamos nenhuma relação entre as variáveis de entrada que nos foram dadas. Tínhamos $\phi_m = x_m$.

Capítulo 5: REGRESSÃO LOGÍSTICA

O modelo de regressão logística, apesar do nome, não é um modelo de regressão, mas sim, de classificação. Ele faz parte de um conjunto de **modelos discriminativos**, que associam a cada classe do problema uma **função discriminante** $\delta_k(\mathbf{x})$ [7, p. 101], de forma que, associamos o exemplo \mathbf{x} à classe k se $\delta_k(\mathbf{x}) > \delta_i(\mathbf{x}) \forall k \neq i$. Mais especificamente, na regressão logística, essa função discriminante é a probabilidade a posteriori $P(\omega_i|\mathbf{x})$. Alguns exemplos de modelos discriminantes são: regressão logística, SVM, árvore de decisão.

Em oposição, existem os **modelos generativos**, que modelam a probabilidade a priori $P(\omega_i)$ e a verossimilhança $p(\mathbf{x}|\omega_i)$ para só depois encontrar a probabilidade a posteriori $P(\omega_i|\mathbf{x})$ por meio do teorema de Bayes.

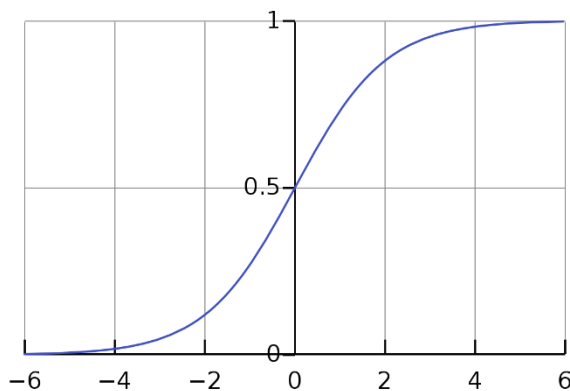
É interessante ressaltar que nos dois grupos de modelos, busca-se determinar $P(\omega_i|\mathbf{x})$ justamente porque, pela teoria de decisão, um determinado exemplo \mathbf{x} deverá ser classificado como classe ω_i se $P(\omega_i|\mathbf{x}) > P(\omega_j|\mathbf{x}) \forall i \neq j$, ou seja, atribuímos \mathbf{x} à classe cuja probabilidade a posteriori for a maior dentre todas as classes.

No modelo da regressão logística, a probabilidade a posteriori é modelada por

$$P(\omega_i|\Phi) = \sigma(\theta^T \Phi),$$

em que $\Phi = [\phi_0 \ \phi_1 \ \dots \ \phi_M]^T$, $\phi_m = \phi_m(x_m)$ é uma função base de x e $\sigma(\cdot)$ é a função sigmoide logística, dada por

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$



Com isso, atribuímos o exemplo \mathbf{x} , correspondente às *features* Φ , à classe ω_i com maior $P(\omega_i|\Phi)$.

Para chegar a esse modelo, existe todo um desenvolvimento que será abordado a seguir.

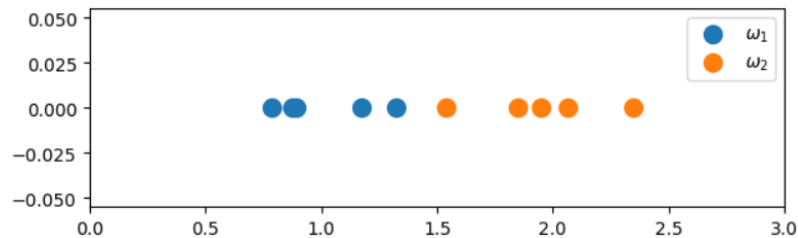
Antes de começarmos, é importante ressaltar que, como o problema é de classificação, o *target* t não assume mais valores decimais. Caso o problema tenha um número K de classes igual a 2, consideramos $t \in \{0,1\}$, em que cada valor de t indica uma classe diferente. Caso $K > 2$, adotamos uma codificação *one-hot* para t , em que temos um vetor de tamanho K cujas entradas são todas nulas, exceto na posição i , que vale 1, indicando que o exemplo em questão pertence à classe ω_i . Um ponto pertencente à classe ω_3 , por exemplo, teria um vetor de *target* da seguinte forma:

$$\mathbf{t} = [0 \ 0 \ 1]^T.$$

Passemos, agora, ao desenvolvimento que leva à criação do modelo da regressão logística.

5.1 MÍNIMOS QUADRADOS PARA CLASSIFICAÇÃO

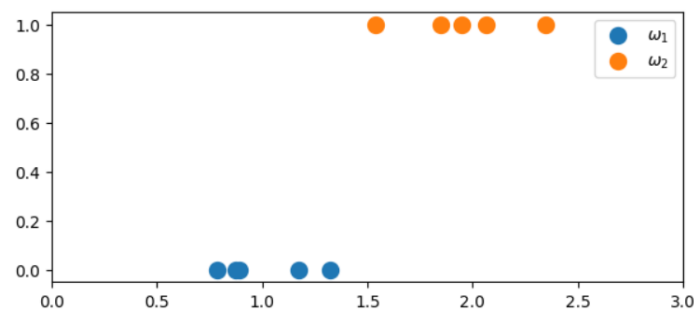
Consideremos um problema de classificação em que temos conjuntos de dados que pertencem a duas classes distintas, como ilustra a imagem abaixo:



Como criar um modelo matemático para, dados novos pontos, conseguir classificá-los em classe ω_1 ou ω_2 ? Se conhecêssemos bem as distribuições que geraram esses pontos, poderíamos propor um valor de x a partir do qual passaríamos a classificar os pontos como classe ω_2 . Por exemplo, se soubéssemos que $X_1 \sim \mathcal{N}(1, 0,1)$ e $X_2 \sim \mathcal{N}(2, 0,1)$ são as variáveis aleatórias que geram essas duas classes de pontos, poderíamos propor uma fronteira de decisão passando em $x = 1,5$, na metade do caminho entre as médias das duas Gaussianas.

No entanto, em quase todos os casos, não temos essa informação e precisamos encontrar um modelo para classificar os pontos mesmo sem saber as funções de distribuição geradoras.

Uma possibilidade mais simples seria utilizar o método dos mínimos quadrados. Mas como fazer isso se temos pontos alinhados em uma única dimensão, com mesmo valor de y ? Podemos codificar a informação presente nas cores em número, por exemplo, fazendo $y = 0$ para a classe ω_1 (azul) e $y = 1$ para a classe ω_2 (laranja), o que nos daria a seguinte representação:



Com isso, fica um pouco menos obscura a ideia de utilizar mínimos quadrados para encontrar uma função $y(x) = \theta^T x$ que se ajuste a esses dados. O ideal seria que essa função $y(x)$ assumisse o valor zero quando x pertencesse a ω_1 e assumisse o valor 1 caso contrário. No entanto, só conseguiríamos encontrar uma reta $y(x) = \theta^T x$ desse tipo somente se todos os pontos azuis estivessem concentrados em um mesmo valor x_1 e todos os pontos laranjas estivessem concentrados em um mesmo valor de x_2 . Claramente isso não acontece e o que temos, na realidade, é um modelo em que y pode assumir qualquer valor real, como mostrado na Figura 5-1.

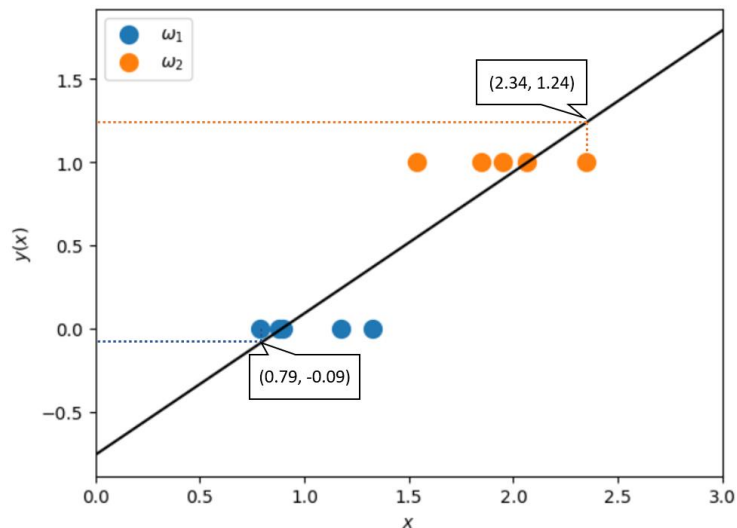


Figura 5-1: Ajuste de uma reta aos pontos das classes azul e laranja.

Uma solução para isso é adotarmos a regra de que, para $y < 0,5$, associamos x a ω_1 e associamos a ω_2 caso contrário. Com essa regra bastante intuitiva, separamos perfeitamente os pontos da imagem acima.

E no caso em que temos 3 classes, por exemplo, como mostra a Figura 5-2?

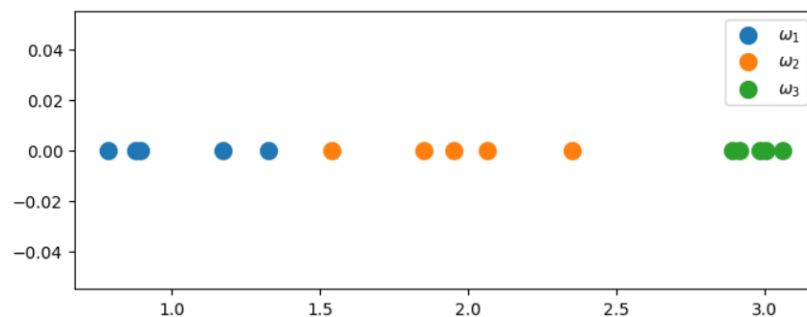


Figura 5-2: Problema multiclasse.

Nesse caso, existem diferentes abordagens. Em [16, p. 183], Bishop menciona algumas delas:

- Um contra um, em que se adota uma função discriminante na forma $y_{ij}(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x}$ para cada par de classes (ω_i, ω_j) . Se $y_{ij}(\mathbf{x}) > 0$, classificamos como ω_i e, caso contrário, classificamos como ω_j . Essa abordagem gera uma região de indecisão, formada por pontos que são atribuídos a mais de uma classe, ou seja, quando para o mesmo ponto \mathbf{x} , existe mais de um $y_{ij}(\mathbf{x}) > 0$ (Figura 5-3).
Observar que as fronteiras de decisão da figura são curvas de nível do hiperplano $y_{ij}(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x}$, e são encontradas fazendo $y_{ij}(\mathbf{x}) = 0$, que é o limiar entre uma classe e outra. Assim, em um exemplo em que tivéssemos $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, $y_{12}(x_1, x_2) = x_2 \theta_2 + x_1 \theta_1 + \theta_0$ é a nossa função discriminante, que é um plano no espaço tridimensional (pode ser reescrito como $x_2 \theta_2 + x_1 \theta_1 + \theta_0 + y_{12} = 0$ para facilitar a visualização) e $x_2 \theta_2 + x_1 \theta_1 + \theta_0 = 0$ é uma reta (que pode ser reescrita como $x_2 = x_1 \frac{\theta_1}{\theta_2} + \frac{\theta_0}{\theta_2}$ para facilitar a visualização), contida no mesmo plano onde se encontram os pontos \mathbf{x} .
- Um contra o resto, em que temos funções discriminante $y_i(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x}$ e, se $y_i(\mathbf{x}) > 0$, dizemos que \mathbf{x} pertence à classe ω_i . Caso contrário, dizemos que \mathbf{x} não é da classe ω_i .

Essa abordagem gera uma região de indecisão, formada por pontos que não foram classificados como nenhuma classe, ou seja, pontos em que $y_k(\mathbf{x}) < 0 \forall k$ (Figura 5-4). Aqui, as fronteiras de decisão da imagem também são curvas de nível da função discriminante.

- Abordagem em que temos uma função $y_i(\mathbf{x}) = \theta^T \mathbf{x}$ para cada classe e atribuímos \mathbf{x} à classe ω_k que tiver maior valor de $y_k(\mathbf{x})$. Nesse caso, não são criadas regiões de indecisão, pois, excetuando as interseções entre os hiperplanos discriminantes, sempre haverá $y_k(\mathbf{x}) > y_j(\mathbf{x}) \forall j \neq k$.

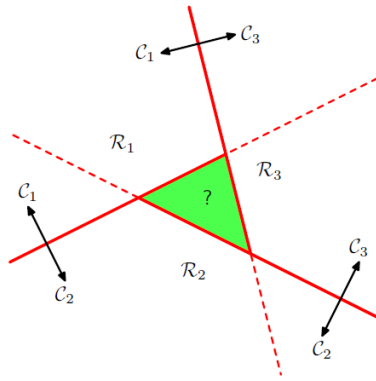


Figura 5-3: Um contra um.

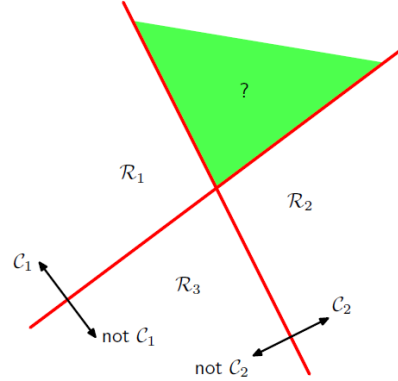


Figura 5-4: Um contra o resto.

Dessa forma, a abordagem que parece mais razoável para nosso problema é a terceira, já que não são criadas regiões de indecisão. Nela, poderíamos utilizar o método dos mínimos quadrados para encontrar as funções discriminantes $y(\mathbf{x})$. No entanto, as fronteiras de decisão são mais difíceis de encontrar, quando comparamos com os dois outros métodos, principalmente no caso multiclasse, pois elas são definidas pelas regiões em que cada curva tem seu valor de $y(\mathbf{x})$ maior do que o de todas as outras classes.

Uma propriedade interessante dessa abordagem é que $\sum_{k=1}^K y_k(\mathbf{x}) = 1$ [16, p. 185], o que é muito similar ao que temos em medidas de probabilidade. Apesar disso, $y_k(\mathbf{x})$ não pode ser considerado uma medida de probabilidade, pois $y_k(\mathbf{x})$ não está restrito apenas ao intervalo $[0, 1]$.

Portanto, aplicando a terceira abordagem ao problema multiclasse ilustrado na Figura 5-2, obtemos uma reta para cada classe, como ilustrado na Figura 5-5. Observamos que, para modelar cada $y_k(\mathbf{x})$, precisamos a cada instante considerar a classe ω_k com $t = 1$ e todas as outras com $t = 0$.

Percebemos que, para $K > 2$, o modelo dos mínimos quadrados para classificação começa a apresentar alguns problemas. A classe ω_2 , por exemplo, apresenta uma curva que atribui valores baixos e mais ou menos próximos de $y_k(\mathbf{x})$ para todos os pontos, mesmo que eles pertençam a outras classes. Isso faz com que os pontos de ω_2 sejam classificados incorretamente como pertencentes às outras classes, cujos $y_k(\mathbf{x})$ são maiores que $y_2(\mathbf{x})$.

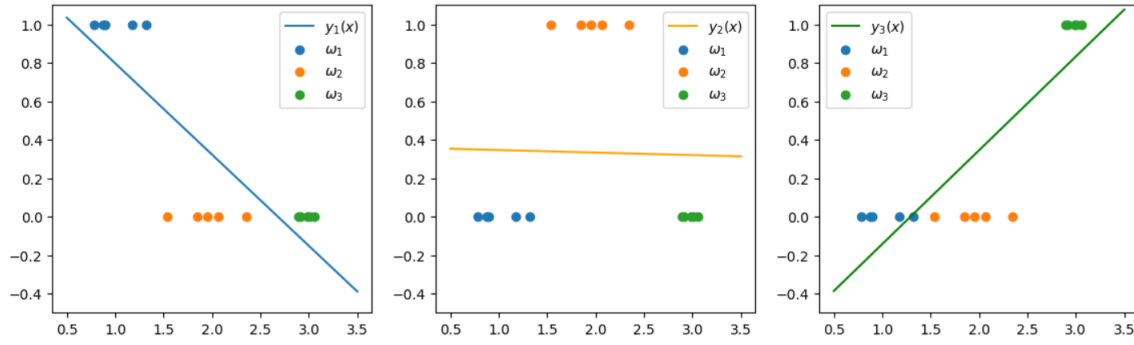


Figura 5-5: Abordagem em que se tem uma regressão linear para cada classe (terceira abordagem).

Por fim, podemos resumir esse problema em uma única equação vetorial:

$$\mathbf{y}(\mathbf{x}) = \mathbf{T}^T (\mathbf{X}^\dagger)^T \mathbf{x},$$

$$\text{em que } \mathbf{y}(\mathbf{x}) = [y_1(\mathbf{x}) \quad \dots \quad y_K(\mathbf{x})]^T, \mathbf{T} = \begin{bmatrix} - & \mathbf{t}_1^T & - \\ & \vdots & \\ - & \mathbf{t}_N^T & - \end{bmatrix}_{N \times K}, \mathbf{X}^\dagger = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \text{ e}$$

$$\mathbf{X} = \begin{bmatrix} - & \mathbf{x}_1^T & - \\ & \vdots & \\ - & \mathbf{x}_N^T & - \end{bmatrix}_{N \times K}. \text{ Lembrando que } \mathbf{t}_k \text{ é o vetor de } target \text{ na codificação } one-hot \text{ e } \mathbf{x} \text{ é o}$$

vetor \mathbf{x} aumentado, ou seja, $\mathbf{x} = [1 \quad \mathbf{x}^T]^T$.

5.2 MODELANDO A PROBABILIDADE A PRIORI

Na seção anterior, vimos que o modelo dos mínimos quadrados pode ser utilizado em problema de classificação. Por outro lado, vimos também que ele não é muito adequado por alguns fatores: não transmitir uma noção de probabilidade (valor não estão restritos a intervalos $[0, 1]$), dificuldades para classificação com $K > 2$.

Por isso, nesta seção, estudamos uma forma de escrever uma equação para probabilidade a posteriori que possa auxiliar na construção de um modelo para classificação.

5.2.1 Caso $K = 2$

Podemos utilizar o Teorema de Bayes para modelar a probabilidade a posteriori da seguinte maneira:

$$P(\omega_1|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_1)P(\omega_1)}{p(\mathbf{x}|\omega_1)P(\omega_1) + p(\mathbf{x}|\omega_2)P(\omega_2)}$$

Com algumas manipulações, podemos reescrever essa equação da seguinte maneira:

$$P(\omega_1|\mathbf{x}) = \frac{1}{1 + \frac{p(\mathbf{x}|\omega_2)P(\omega_2)}{p(\mathbf{x}|\omega_1)P(\omega_1)}} = \frac{1}{1 + e^{-\ln \frac{p(\mathbf{x}|\omega_1)P(\omega_1)}{p(\mathbf{x}|\omega_2)P(\omega_2)}}} = \frac{1}{1 + e^{-\ln \frac{P(\omega_1|\mathbf{x})}{P(\omega_2|\mathbf{x})}}},$$

que é equivalente a

$$P(\omega_1|\mathbf{x}) = \sigma\left(\ln \frac{P(\omega_1|\mathbf{x})}{P(\omega_2|\mathbf{x})}\right) = \sigma\left(\ln \frac{P(\omega_1|\mathbf{x})}{1 - P(\omega_1|\mathbf{x})}\right).$$

Por curiosidade, percebemos que $P(\omega_1|\mathbf{x})$ pode ser escrita com uma composição de funções dela mesma, ou seja,

$$P(\omega_1|\mathbf{x}) = \sigma \circ f(P(\omega_1|\mathbf{x})).$$

Ora, se $x = f \circ g(x)$, então $f = g^{-1}$. Assim, notamos que a inversa da função sigmoide é dada por

$$f(x) = \ln\left(\frac{x}{1-x}\right),$$

que é conhecida como **função logit**.

5.2.2 Caso $K > 2$

Para mais de duas classes, podemos escrever a probabilidade a posteriori da classe ω_i como

$$P(\omega_i|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_i)P(\omega_i)}{\sum_{k=1}^K p(\mathbf{x}|\omega_k)P(\omega_k)} = \sigma\left([\ln(p(\mathbf{x}|\omega_k)P(\omega_k))]_{k=1}^K\right)_i,$$

em que $[\ln(p(\mathbf{x}|\omega_k)P(\omega_k))]_{k=1}^K = [\ln(p(\mathbf{x}|\omega_1)P(\omega_1)) \quad \cdots \quad \ln(p(\mathbf{x}|\omega_K)P(\omega_K))]^T$.

A função

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{k=1}^K e^{z_k}},$$

com $\mathbf{z} = [z_1 \quad \cdots \quad z_K]^T$, é conhecida como **softmax**. Observar que é necessário dar como entrada tanto \mathbf{z} quanto o índice i . Ela é o caso geral da função sigmoide.

5.2.3 Aplicação para distribuições conhecidas

Se formos aplicar as equações desenvolvidas acima em casos em que conhecemos a *likelihood*, como no caso da distribuição normal, exponencial, discretas binárias, encontramos probabilidades a priori com o seguinte formato

$$K = 2:$$

$$P(\omega_k|\mathbf{x}) = \sigma(\boldsymbol{\theta}_k^T \mathbf{x})$$

$$K > 2:$$

$$P(\omega_i|\mathbf{x}) = \sigma\left([\boldsymbol{\theta}_k^T \mathbf{x}]_{k=1}^K\right)_i$$

A expressão para $\boldsymbol{\theta}_k$ varia de acordo com a distribuição [16, pp. 198-203].

Com isso, percebemos que, para várias distribuições bastante comuns, temos uma probabilidade a posteriori que é dada pela *softmax* (o que, por generalização, inclui a

sigmoide) de uma função linear. Isso faz com que as fronteiras de decisão entre essas classes sejam hiperplanos.

Tomemos por exemplo o caso em que sabemos que os dados pertencem às classes ω_1 e ω_2 e que são gerados por distribuições gaussianas com parâmetros específicos para cada classe. Isso quer dizer que conhecemos as *likelihoods* $p(\mathbf{x}|\omega_1)$ e $p(\mathbf{x}|\omega_2)$, com $\mathbf{x} \in \mathbb{R}^2$.

Para o caso em que as duas distribuições apresentam a mesma matriz de covariância Σ temos probabilidade a posteriori como mostrado acima: $P(\omega_k|\mathbf{x}) = \sigma(\boldsymbol{\theta}_k^T \mathbf{x})$. Para encontrar a fronteira de decisão, segundo a teoria de decisão, precisamos determinar:

$$P(\omega_1|\mathbf{x}) = P(\omega_2|\mathbf{x})$$

$$\sigma(\boldsymbol{\theta}_1^T \mathbf{x}) = \sigma(\boldsymbol{\theta}_2^T \mathbf{x})$$

$$\boldsymbol{\theta}_1^T \mathbf{x} = \boldsymbol{\theta}_2^T \mathbf{x}$$

$$(\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2)^T \mathbf{x} = 0$$

$$(\theta_{1_2} - \theta_{2_2})x_2 + (\theta_{1_1} - \theta_{2_1})x_1 + (\theta_{1_0} - \theta_{2_0}) = 0$$

que é a equação de uma reta no \mathbb{R}^2 e de um plano no \mathbb{R}^3 .

Um erro que poderia ser cometido é achar que o plano que correspondente a essa reta do \mathbb{R}^2 no \mathbb{R}^3 é a função $f(\mathbf{x}) = (\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2)^T \mathbf{x}$. No entanto, não podemos afirmar isso, pois a equação dessa reta não surgiu a partir de uma curva de nível de uma função $z = f(\mathbf{x})$, fazendo $f(\mathbf{x}) = 0$, mas sim, da equação $P(\omega_1|\mathbf{x}) = P(\omega_2|\mathbf{x})$.

Então, não é possível fazer o caminho inverso $(\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2)^T \mathbf{x} = 0 \rightarrow z = f(\mathbf{x}) = (\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2)^T \mathbf{x}$, pois não conhecemos o comportamento da variável z em outros valores diferentes de $z = 0$.

Por que não poderia ser $z = f(\mathbf{x}) = \frac{1}{3}(\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2)^T \mathbf{x}$, ou escrito de maneira equivalente, $(\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2)^T \mathbf{x} - 3z = 0$, que é igual a equação mostrada acima em $z = 0$?

No \mathbb{R}^3 , temos sempre 3 variáveis: x_1 , x_2 e z . Nossa equação tem a forma $ax_1 + bx_2 + d = 0$. O z também está representado nessa equação, mas de maneira implícita: $ax_1 + bx_2 + 0z + d = 0$. Isso nos indica que, x_1 e x_2 só possuem dependência entre eles e o valor de z não influencia nos valores que essas duas variáveis podem assumir. Então, temos um plano paralelo ao eixo z , pois, independentemente do valor de z escolhido, a relação entre x_1 e x_2 é sempre a mesma: $x_1 = \frac{d}{a} - \frac{b}{a}x_2$.

Para encerrar essa discussão, isso é só uma questão de ponto de vista. Se estamos trabalhando com representações 2D desse problema, representaremos a fronteira de decisão como uma reta no \mathbb{R}^2 . Se estamos trabalhando com representações em 3D, representaremos nossa fronteira de decisão como um plano no \mathbb{R}^3 .

A Figura 5-7 e a Figura 5-7 ilustram as curvas desse exemplo de diferentes pontos de vista. Na curva da probabilidade a posteriori, pode parecer que ela representa uma função de densidade de probabilidade (contínua) de \mathbf{x} . Para que isso fosse verdade, precisaríamos que

$$\int_{\mathbf{x} \in \mathbb{R}^2} P(\omega_1|\mathbf{x}) d\mathbf{x} = 1,$$

o que é um absurdo, já que $\lim_{(x_1, x_2) \rightarrow (\infty, \infty)} \sigma(\theta_1^T \mathbf{x}) = 1$ e, por isso, $\int_{\mathbf{x} \in \mathbb{R}^2} P(\omega_1 | \mathbf{x}) d\mathbf{x} = \int_{\mathbf{x} \in \mathbb{R}^2} \sigma(\theta_1^T \mathbf{x}) d\mathbf{x}$ não converge. Lembrando que, como $\theta_1^T \mathbf{x}$ é linear, $\sigma(\theta_1^T \mathbf{x})$ tem a mesma forma da sigmoide, porém possivelmente deslocada e distorcida ao longo do eixo x , no caso unidimensional, ou ao longo do plano formado por pelas componentes de \mathbf{x} , no caso multidimensional.

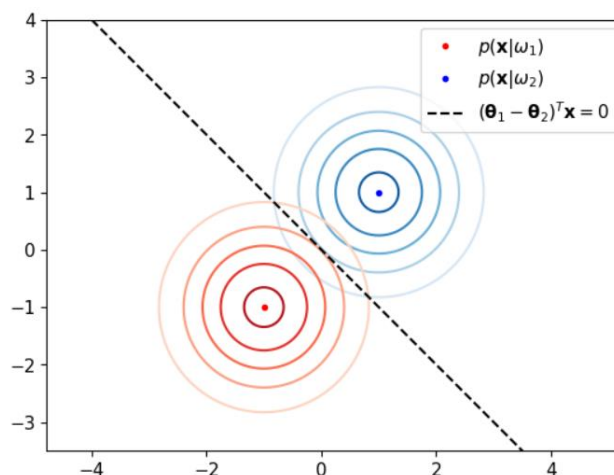


Figura 5-6: Curvas de nível das likelihoods.

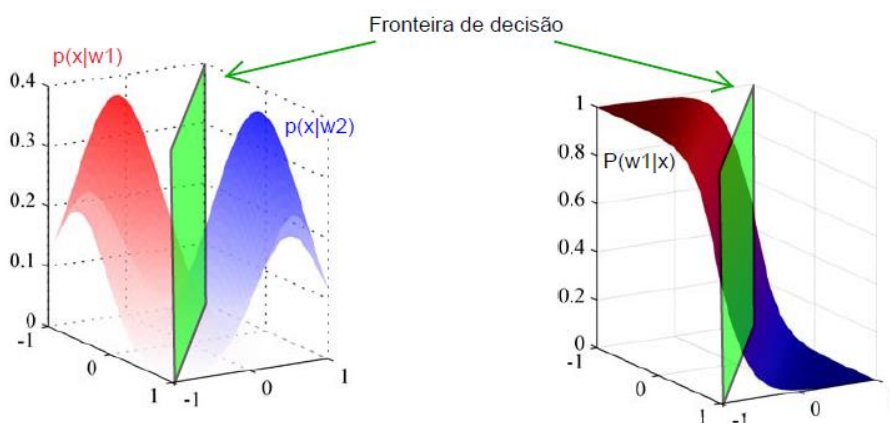


Figura 5-7: Representação das likelihoods (à esquerda) e da função de probabilidade a posteriori (à direita). Nessas figuras, está representada a fronteira de decisão $(\theta_1 - \theta_2)^T \mathbf{x} = 0$, só que aqui, representada no \mathbb{R}^3 .

Em outras palavras, a aba vermelha do gráfico à direita na Figura 5-7 se estende até o infinito. Isso faz sentido, já que, quanto mais além do plano da fronteira de decisão maior é a chance do ponto \mathbf{x} pertencer à classe ω_1 . Logo, esse gráfico não representa o gráfico de uma função de distribuição de probabilidade.

No entanto, isso não quer dizer $P(\omega | \mathbf{x})$ não seja uma p.d.f. Ela é, mas não em função de \mathbf{x} e sim de ω . Mais especificamente, ela é uma função de massa, pois ela é discreta. Repare que $\sum_{k=1}^K P(\omega_k | \mathbf{x}) = 1$.

Uma última observação, é que podemos perceber perfeitamente o formato 3D da sigmoide na figura. Vemos também que a região em que a probabilidade de ser a classe ω_1 é 0,5 (chances iguais de ser ω_1 e ω_2) é justamente uma reta ao longo da qual passa o plano representando a fronteira de decisão.

Para finalizar este exemplo, é importante lembrar que a expressão de θ depende dos parâmetros das duas Gaussianas, ou seja, Σ , μ_1 e μ_2 . Se por acaso só soubermos que as duas classes geram dados segundo uma distribuição Gaussiana, mas não conhecermos o valor desses parâmetros (o que é comum na realidade), podemos estimá-los por meio da estimação de máxima verossimilhança utilizando os dados x . Com isso, determinamos os valores numéricos de θ caso os parâmetros não sejam conhecidos.

5.3 CONSTRUÇÃO DA REGRESSÃO LOGÍSTICA

Na seção anterior, vimos que, para alguns casos de *likelihood*, podemos escrever a probabilidade a posteriori como a função sigmoide/softmax de uma função linear. Assim, conhecendo a *likelihood* dos dados gerados por cada classe, conseguimos escrever uma equação para $P(\omega_k|x)$ com auxílio do teorema de Bayes.

No entanto, na grande maioria das vezes, os dados não seguem exatamente as distribuições especiais da seção anterior e, normalmente, não conhecemos a distribuição deles. Com isso, seríamos obrigados a estimar suas distribuições de probabilidade, o que é uma abordagem completamente válida e é a marca dos modelos generativos, mas também pode gerar resultados ruins se a estimação não for adequada.

Uma alternativa seria, em vez de desenvolvermos um método em que precisemos estimar a verossimilhança para só assim calcularmos a posteriori, desenvolver um modelo discriminativo. É daí que surge a ideia da regressão logística.

O modelo da regressão logística consiste em generalizar a probabilidade a posteriori por meio da expressão

$$P(\omega_i|\Phi) = \sigma \left([\theta_k^T \Phi]_{k=1}^K \right)_i,$$

o que parece bastante natural, visto que várias distribuições têm expressões analíticas exatas para a posteriori dessa mesma forma.

Assim, mesmo que não conheçamos a distribuição dos dados, aplicamos a mesma expressão que vimos na seção anterior. A diferença é que antes tínhamos uma expressão exata para $P(\omega_k|x)$ e, agora, temos algo estimado. Se antes tínhamos expressões analíticas exatas para θ calculadas a partir da expressão da verossimilhança conhecida/estimada e com auxílio do teorema de Bayes, agora, calcularemos a expressão da probabilidade a posteriori com auxílio da estimação de máxima verossimilhança sem conhecer a *likelihood*.

Observar que, aqui, substituímos o vetor x de entradas por um vetor genérico Φ de *features*, formado por funções base não lineares $\phi_m(x_m)$. Essas funções são importantes para separar dados que não podem ser separados linearmente, com fronteiras de decisão lineares. A Figura 5-8 ilustra um exemplo em que foram utilizadas funções base gaussianas com centros marcados nas cruzes verdes. Com isso percebemos que com a escolha de funções base corretas, conseguimos separar os dados (no espaço original das entradas) usando uma fronteira de decisão circular. Observar também que no espaço das *features*, a fronteira de decisão é linear, da forma como desenvolvemos ao longo desta seção.

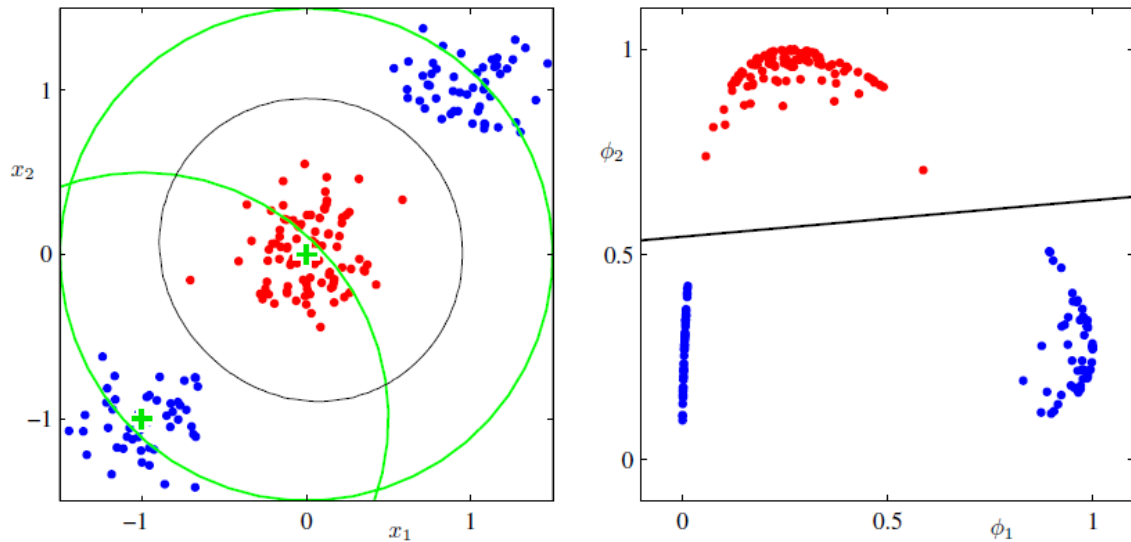


Figura 5-8: À esquerda, dados no espaço original das entradas x . À direita, a transformação desses dados para o espaço das features com funções base ϕ .

Dada a expressão da regressão logística, precisamos encontrar o parâmetro θ . Para isso, nos basearemos no princípio da estimação de máxima verossimilhança.

Precisaremos ficar atentos para não nos confundirmos. No início desta seção, foi sugerido usar a MLE para calcular os parâmetros a distribuição dos dados quando supúnhamos que eles eram gerados por uma distribuição específica conhecida (gaussiana). Nesse caso, tentávamos maximizar $p(x|\Sigma_i, \mu_i)$ para cada classe ω_i , pois, no final, por meio do teorema de Bayes, conseguíamos escrever a expressão de θ em função desses parâmetros (Σ_i, μ_i) específicos de cada classe.

Aqui, estimaremos o parâmetro θ diretamente por meio da MLE, sem tentar escrevê-lo em função de outros parâmetros, maximizando $P(t|\theta)$. É importante lembrar que $P(t|\phi, \theta) \equiv P(t|\theta)$, mas omitimos o ϕ para simplificar a notação. Se considerarmos que as amostras são independentes, teremos a seguinte expressão para a verossimilhança:

$$P(t|\theta) = \prod_{n=1}^N \log P(t_n|\theta).$$

No caso de classificação binária, teremos duas possibilidades:

- $P(t_n|\theta) = P(\omega_1|\phi_n, \theta);$
- $P(t_n|\theta) = P(\omega_2|\phi_n, \theta).$

Deixamos explícito nas expressões acima que ϕ_n é dado porque senão não ficaria claro que essa probabilidade está associada apenas a esse exemplo de índice n . Além disso, reparar que manter o ϕ_n como uma informação dada é apenas uma notação. Poderíamos ter uma variável aleatória que codificasse isso. Por exemplo, poderíamos ter uma variável aleatória Ω_n que representa o evento “classe a qual pertence o exemplo ϕ_n ” e poderíamos expressar as probabilidades acima como $P(\Omega_n = \omega_i; \theta)$. Nesse último caso, também mostramos que não precisamos deixar o θ como uma informação dada, mais sim, como um parâmetro, o que é marcado explicitamente pelo ponto e vírgula.

Por mais que possa parecer confuso, a expressão que estabelecemos inicialmente para a posteriori, $P(\omega_i|\Phi) = \sigma(\theta^T \Phi_n)$ (no caso binário), é equivalente a $P(\omega_i|\Phi_n, \theta)$. Isso porque na expressão $P(\omega_i|\Phi)$, temos o termo θ explicitamente. Logo, ele tem que ser um parâmetro dado, por mais que não apareça explicitamente na notação $P(\omega_i|\Phi)$. Lembrando que, dizer que θ é dado, não significa dizer que conhecemos os valores de suas componentes, pois ele pode ser “dado” e ainda assim ser uma variável desconhecida. O problema está na nomenclatura “dado”, que nesse caso, é o equivalente a dizer que estamos calculando a probabilidade do exemplo Φ_n pertencer à classe ω_i condicionado a θ . Assim, fica mais claro que não se trata de probabilidades em que se tem valores conhecidos dados, mas sim, de probabilidades condicionadas a uma variável. Portanto, se temos $\sigma(\theta^T \Phi_n)$, obviamente essa expressão está condicionada a θ , por mais que o vetor de parâmetros não se encontre explícito na expressão da probabilidade.

Depois dessa discussão, vamos voltar ao nosso problema de encontrar o valor de θ para o caso da classificação binária. Lembremos que, no caso binário, temos que $t_n \in \{0,1\}$. Nosso objetivo é maximizar com respeito a θ :

$$P(\mathbf{t}|\theta) = \prod_{n=1}^N P(\omega_1|\Phi_n)^{t_n} [1 - P(\omega_1|\Phi_n)]^{1-t_n} = \prod_{n=1}^N s_n^{t_n} (1 - s_n)^{1-t_n},$$

em que $s_n = P(\omega_1|\Phi_n) = \sigma(\theta^T \Phi_n)$.

Por simplicidade, como de costume, minimizamos o negativo da log verossimilhança:

$$\log P(\mathbf{t}|\theta) = - \sum_{n=1}^N t_n \log s_n + (1 - t_n) \log(1 - s_n) \quad (5.1)$$

A expressão dentro desse somatório é uma *loss* conhecida como **entropia cruzada binária** e é amplamente utilizada em problemas de classificação. Para o caso em que temos K classes e o ponto x sendo avaliado pertence a uma classe ω_j com $j \in [1, K]$, a expressão da entropia cruzada pode ser escrita como

$$H_c(x) = - \sum_{k=1}^K p_k \log q_k = - \sum_{k=1}^K \mathbb{1}_{\{x \in \omega_k\}} \log q_k = - \log q_j$$

em que q_k é a probabilidade de o ponto x pertencer à classe k e p_k é a probabilidade de o ponto x pertencer à classe k sabendo qual a classe a qual ele realmente pertence, ou seja, $q_k = \mathbb{1}_{\{x \in \omega_k\}}$. Lembrando que essa expressão (considerando p_k e q_k da forma como definimos) é um caso específico do problema de classificação e foi derivada a partir da eq. (5.1). Na expressão geral da entropia cruzada (para problemas que não são necessariamente relacionados a *machine learning* ou a classificação) q_k e p_k são funções de densidade de probabilidade quaisquer (consultar 0).

Reparar que $H_c(x)$ terá apenas uma parcela do somatório não nula.

A função custo genérica para um problema de classificação com K classes teria a seguinte forma:

$$J(\boldsymbol{\theta}) = \sum_{n=1}^N H_c(\mathbf{x}_n) = - \sum_{n=1}^N \sum_{k=1}^K p_{k(\mathbf{x}_n)} \log q_{k(\mathbf{x}_n)}$$

Ela poderia ter sido derivada da mesma maneira como fizemos para o caso binário.

Reparar que, apesar de termos derivado a expressão da entropia cruzada à medida que derivávamos a expressão da regressão logística, elas não são dependentes uma da outra. A entropia cruzada não surgiu do modelo de regressão logística. Ela poderia ter sido derivada de qualquer outro problema de classificação.

Tentando ver esse problema de uma forma mais intuitiva, o que estamos fazendo é minimizar a entropia cruzada, que quantifica a dissimilaridade entre as distribuições p_k e q_k (lembrando que minimizar a entropia cruzada é equivalente a minimizar a divergência KL, conforme 0). Então, como $q_k(\mathbf{x}) = \mathbb{1}_{\{\mathbf{x} \in \omega_k\}}$, queremos fazer com que a probabilidade p_k se aproxime o máximo possível de $\mathbb{1}_{\{\mathbf{x} \in \omega_k\}}$.

Por fim, para encontrar os parâmetros $\boldsymbol{\theta}$ aplicamos o método *Iterative Reweighted Least Squares (IRLS)*, que é baseado no método do Newton e consiste em atualizar o valor de $\boldsymbol{\theta}$ a cada iteração da seguinte maneira:

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \mathbf{H}^{-1} \nabla J(\boldsymbol{\theta}),$$

em que $\mathbf{H} = \nabla \nabla J(\boldsymbol{\theta})$ [16, p. 207].

5.4 CONCLUSÕES

Ao final de todo o desenvolvimento para se chegar ao modelo da regressão logística, percebemos que existe uma relação entre ele e o modelo dos mínimos quadrados para classificação.

No método dos mínimos quadrados para classificação, um dos pontos negativos era o fato de não termos funções $y_i(\mathbf{x})$ que possam ser interpretadas como probabilidades, pois seus valores não se limitavam ao intervalo $[0,1]$. Na regressão logística, é como se pegássemos o modelo dos mínimos quadrados que tínhamos desenvolvido e o empacotássemos na função logística sigmoide, obrigando, assim, que os valores fiquem limitados ao intervalo $[0,1]$.

Obviamente, não é possível encontrar $\boldsymbol{\theta}$ com a equação dos mínimos quadrados $\boldsymbol{\theta}^* = \mathbf{T}^T (\mathbf{X}^\dagger)^T$ e substituir esse valor em $\sigma(\boldsymbol{\theta}^T \boldsymbol{\Phi})$, pois, só foi possível chegar a essa equação porque não havia inicialmente a não linearidade da função sigmoide.

Por fim, é interessante reparar que, mesmo para os casos em que a distribuição dos dados pertence a uma daquelas distribuições especiais descritas na Seção 5.2.3, ainda é preferível usar o modelo da regressão logística em vez de usar as expressões analíticas para cada um desses casos particulares. Isso porque, na regressão logística, precisamos estimar consideravelmente menos parâmetros comparado com os métodos que se baseiam no teorema de Bayes. Por exemplo, se, num problema de classificação binária, tivermos vetores de *features* de dimensão M e nossos dados seguirem uma distribuição Gaussiana, então, na regressão logística, teremos apenas M parâmetros para estimar. Já utilizando o modelo

baseado no teorema de Bayes teríamos que estimar $2M$ parâmetros para as médias $\boldsymbol{\mu}$ e $M(M + 1)/2$ parâmetros para a matriz de covariância $\boldsymbol{\Sigma}$ [16, p. 205] [5, p. 319].

Capítulo 6: SUPPORT VECTOR MACHINE

Um conceito importante para entender o modelo SVM é o **hiperplano de separação ótimo**, que separa duas classes linearmente separáveis e maximiza a distância entre os pontos mais próximos de cada classe [7]. Esse conceito é ilustrado na Figura 6-1. A distância do hiperplano de separação para o ponto mais próximo de cada classe é chamada de **margem** e é representado pela letra M .

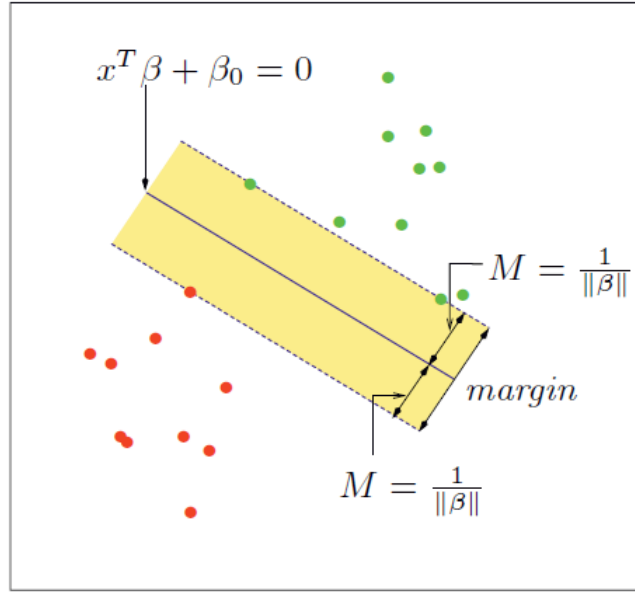


Figura 6-1: Exemplo de classificador de vetor suporte.

Um hiperplano de separação pode ser definido como $L = \{x : f(x) = w_0 + \mathbf{w}^T x = 0\}$, como mostra a Figura 6-2. Ele consegue separar duas classes de dados: os dados que estão acima do hiperplano, para os quais $f(x) > 0$, e os dados que estão abaixo do hiperplano, para os quais $f(x) < 0$. Assim, podemos usar o sinal de $f(x)$ para classificar os dados.

Uma métrica importante de se conhecer é a distância de um ponto x qualquer para esse hiperplano. Primeiramente, façamos algumas considerações.

Se $x_1, x_2 \in L$, então $\mathbf{w}^T(x_1 - x_2) = 0$. Como $(x_1 - x_2) \parallel L$, concluímos que $\mathbf{w} \perp L$.

Seja $x_0 \in L$. A distância de um ponto x para L é dada por

$$r = \frac{\mathbf{w}^T}{\|\mathbf{w}\|} (x - x_0) = \frac{1}{\|\mathbf{w}\|} (\mathbf{w}^T x - \mathbf{w}^T x_0)$$

Como $w_0 + \mathbf{w}^T x_0 = 0 \Rightarrow -\mathbf{w}^T x_0 = w_0$, temos que

$$\begin{aligned} r &= \frac{1}{\|\mathbf{w}\|} (\mathbf{w}^T x + w_0) \\ r &= \frac{f(x)}{\|\mathbf{w}\|} \end{aligned} \tag{6.1}$$

Observar que r pode ser positivo ou negativo, dependendo do sinal de $f(x)$.

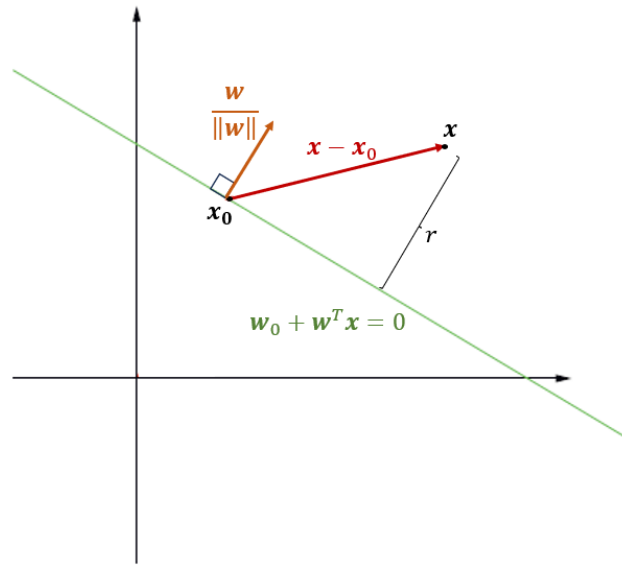


Figura 6-2: Distância de um hiperplano para um ponto.

Tendo uma expressão para a distância entre um hiperplano e um ponto qualquer, podemos passar ao problema do hiperplano de separação ótimo.

Consideremos o conjunto de dados (x_i, y_i) com $i = 1, \dots, N$ e $y_i \in \{-1, 1\}$. Consideremos também que é possível usar um hiperplano de separação $L = \{x : f(x) = 0\}$ para separar os dados, de forma que $f(x) > 0$ quando $y_i = 1$ e $f(x) < 0$ quando $y_i = -1$. Assim, podemos resumir o problema de encontrar o hiperplano de separação ótimo ao problema de maximizar a margem M , para a qual ainda não temos uma expressão definida, mas com a restrição de que todos os pontos devem estar acima dessa margem. No entanto, só estamos interessados nos pontos que foram corretamente classificados, para os quais $y_i f(x_i) > 0$ e cuja distância absoluta para L é dada por $y_i \frac{f(x_i)}{\|w\|} = \frac{y_i(w_0 + w^T x_i)}{\|w\|}$.

Matematicamente, escrevemos esse problema como

$$\begin{aligned} \max_{w, w_0} \quad & M \\ \text{sujeito a} \quad & \frac{y_i(w_0 + w^T x_i)}{\|w\|} \geq M, \quad i = 1, \dots, N \end{aligned}$$

Como $M > 0$, todos os pontos incorretamente classificados são automaticamente deixados de fora.

Observar que multiplicar w e w_0 (os únicos termos que temos a liberdade de variar nesse problema de otimização) por um fator α , ou seja, $w \leftarrow \alpha w$ e $w_0 \leftarrow \alpha w_0$, não altera em nada o problema acima:

$$\frac{y_i(\alpha w_0 + \alpha w^T x_i)}{\|\alpha w\|} = \frac{y_i(w_0 + w^T x_i)}{\|w\|} \geq M$$

Assim, se $w = w'$ e $w_0 = w'_0$ maximizam o problema acima, então $w = \alpha w'$ e $w_0 = \alpha w'_0$ também maximizam. Temos essa liberdade para variar essas variáveis porque temos uma única restrição, mas duas variáveis.

Com isso, podemos escolher um α tal que $\alpha\|\mathbf{w}\| = \frac{1}{M}$. Portanto, fazemos $w_0 \leftarrow \alpha w'_0$ e $\mathbf{w} \leftarrow \alpha \mathbf{w}'$, com $\alpha\|\mathbf{w}'\| = \frac{1}{M}$ e encontramos:

$$y_i(\alpha w'_0 + \alpha \mathbf{w}'^T \mathbf{x}_i) \geq 1$$

Por simplicidade, podemos definir $w''_0 = \alpha w'_0$ e $\mathbf{w}'' = \alpha \mathbf{w}'$. Além disso, como $M = \frac{1}{\alpha\|\mathbf{w}'\|} = \frac{1}{\|\mathbf{w}''\|}$, o problema de otimização pode ser escrito como:

$$\begin{aligned} \max_{\mathbf{w}, w_0} \quad & \frac{1}{\|\mathbf{w}''\|} \\ \text{sujeito a} \quad & y_i(w''_0 + \mathbf{w}''^T \mathbf{x}_i) \geq 1, \quad i = 1, \dots, N \end{aligned}$$

Para manter a notação anterior, $w''_0 \leftarrow w_0$ e $\mathbf{w}'' \leftarrow \mathbf{w}$. Além disso, maximizar $\frac{1}{\|\mathbf{w}\|}$ é equivalente a minimizar $\|\mathbf{w}\|$, que é equivalente a minimizar $\frac{1}{2}\|\mathbf{w}\|^2$. Portanto, o problema de otimização se resume a

$$\begin{aligned} \min_{\mathbf{w}, w_0} \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{sujeito a} \quad & y_i(w_0 + \mathbf{w}^T \mathbf{x}_i) \geq 1, \quad i = 1, \dots, N. \end{aligned} \quad (6.2)$$

A escolha de minimizar $\frac{1}{2}\|\mathbf{w}\|^2$ em vez de $\|\mathbf{w}\|$ ficará mais evidente adiante.

A função de Lagrange $L(\mathbf{w}, w_0, \boldsymbol{\lambda})$ e a função de Lagrange dual $g(\boldsymbol{\lambda})$ desse problema são, respectivamente:

$$\begin{aligned} L(\mathbf{w}, w_0, \boldsymbol{\lambda}) &= \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^N \lambda_i [-y_i(w_0 + \mathbf{w}^T \mathbf{x}_i) + 1] \\ g(\boldsymbol{\lambda}) &= \inf_{\mathbf{w}, w_0} L(\mathbf{w}, w_0, \boldsymbol{\lambda}) \end{aligned}$$

Para encontrar $\inf_{\mathbf{w}, w_0} L(\mathbf{w}, w_0, \boldsymbol{\lambda})$, fazemos

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w}^T - \sum_{i=1}^N \lambda_i y_i \mathbf{x}_i^T = 0 \Rightarrow \mathbf{w} = \sum_{i=1}^N \lambda_i y_i \mathbf{x}_i \quad (6.3)$$

$$\frac{\partial L}{\partial w_0} = 0 \Rightarrow \sum_{i=1}^N \lambda_i y_i = 0 \quad (6.4)$$

Lembrando que $\|\mathbf{w}\|^2 = \mathbf{w}^T \mathbf{w}$. Assim vemos que, anteriormente, foi escolhido minimizar $\frac{1}{2}\|\mathbf{w}\|^2$ porque o quadrado torna a derivação mais simples.

O passo anterior só é possível quando há dualidade forte, ou seja, quando o gap de dualidade é zero e temos $\inf_{\mathbf{w}, w_0} L(\mathbf{w}, w_0, \boldsymbol{\lambda}) = L(\mathbf{w}^*, w_0^*, \boldsymbol{\lambda})$, em que \mathbf{w}^* e w_0^* são os parâmetros ótimos, que minimizam a função objetivo. Como o problema primal é convexo (função objetivo quadrática e restrição linear), basta que as condições de KKT sejam satisfeitas para que possamos afirmar que há dualidade forte. As condições de KKT são as seguintes:

1. $\mathbf{w} = \sum_{i=1}^N \lambda_i y_i \mathbf{x}_i$

2. $\sum_{i=1}^N \lambda_i y_i = 0$
3. $y_i(\mathbf{w}''_0 + \mathbf{w}''^T \mathbf{x}_i) - 1 \leq 0$
4. $\lambda_i [y_i(\mathbf{w}_0 + \mathbf{w}^T \mathbf{x}_i) - 1] = 0$
5. $\lambda_i \geq 0$

Se garantirmos que todas elas sejam satisfeitas, o que é trivial, podemos seguir assumindo dualidade forte.

A expressão da função de Lagrange dual fica:

$$\begin{aligned}
 g(\boldsymbol{\lambda}) &= \frac{1}{2} \|\mathbf{w}\|^2 - w_0 \overbrace{\sum_{i=1}^N \lambda_i y_i}^{\text{eq. (6.4)}} - \sum_{i=1}^N \lambda_i y_i \overbrace{\widehat{\mathbf{w}}^T \mathbf{x}_i}^{\text{eq. (6.3)}} + \sum_{i=1}^N \lambda_i \\
 g(\boldsymbol{\lambda}) &= \frac{1}{2} \left(\sum_{i=1}^N \lambda_i y_i \mathbf{x}_i^T \right) \left(\sum_{i=1}^N \lambda_i y_i \mathbf{x}_i \right) - \sum_{i=1}^N \sum_{j=1}^N \lambda_i \lambda_j y_i y_j \mathbf{x}_j^T \mathbf{x}_i + \sum_{i=1}^N \lambda_i \\
 g(\boldsymbol{\lambda}) &= \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \lambda_i \lambda_j y_i y_j \mathbf{x}_j^T \mathbf{x}_i - \sum_{i=1}^N \sum_{j=1}^N \lambda_i \lambda_j y_i y_j \mathbf{x}_j^T \mathbf{x}_i + \sum_{i=1}^N \lambda_i \\
 g(\boldsymbol{\lambda}) &= -\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \lambda_i \lambda_j y_i y_j \mathbf{x}_j^T \mathbf{x}_i + \sum_{i=1}^N \lambda_i \tag{6.5}
 \end{aligned}$$

Para $\boldsymbol{\lambda} \geq 0$, em que \geq é o símbolo de maior ou igual elemento a elemento.

Com isso, o problema de otimização se resume a maximizar a função de Lagrange dual acima (comumente resolvida pelo computador) respeitando as condições de KKT.

Tendo encontrados os valores ótimos de \mathbf{w} e w_0 , o hiperplano de separação ótimo produz uma função $\hat{f}(\mathbf{x}) = w_0 + \mathbf{w}\mathbf{x}$, que pode ser usada para classificar novos dados. Se $\hat{f}(\mathbf{x})$ for positiva, temos uma classe e, se for negativa, temos outra.

É interessante observar uma das condições KKT, dada pela *complementary slackness*, mostrada a seguir:

$$\lambda_i [y_i(w_0 + \mathbf{w}^T \mathbf{x}_i) - 1] = \lambda_i [y_i f(\mathbf{x}_i) - 1] = 0$$

Para essa condição ser respeitada, precisamos ter $\lambda_i = 0$ ou $y_i f(\mathbf{x}_i) = 1$. Isso nos deixa com dois cenários interessantes:

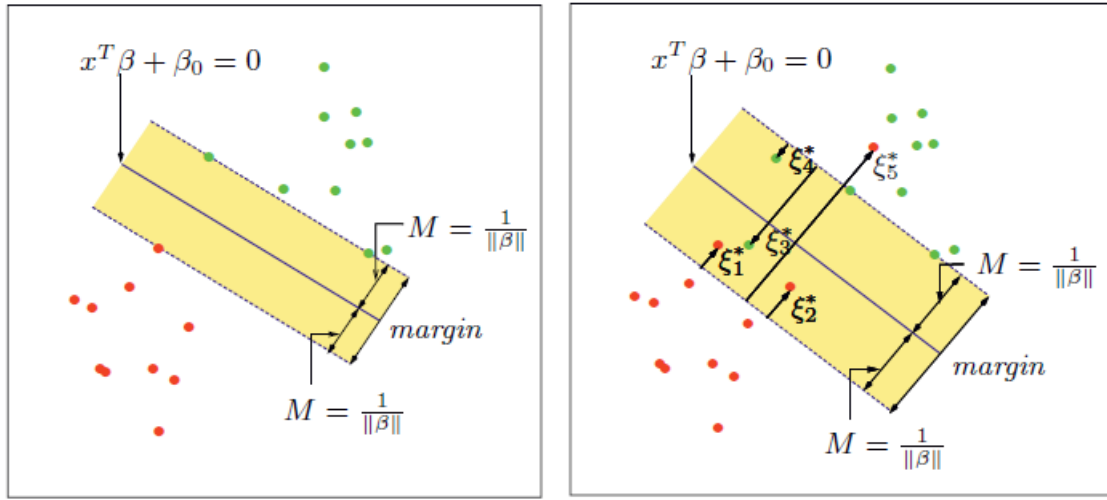
- $\lambda_i > 0$, o que implica $y_i f(\mathbf{x}_i) = 1$, isto é, \mathbf{x}_i está exatamente na margem (reparar que, em (6.2), $y_i f(\mathbf{x}_i) \geq 1$, havendo igualdade somente quando \mathbf{x}_i está na margem).
- $y_i f(\mathbf{x}_i) > 1$, isto é, \mathbf{x}_i não está na margem e $\lambda_i = 0$, o que zera o somatório da eq. (6.5) fazendo com que esse ponto não contribua para a solução do problema.

Portanto, reparamos que apenas os pontos que estão na margem contribuem para a determinação do plano de separação ótimo, já que \mathbf{w} é uma combinação linear desses pontos (eq. (6.3)). Esses pontos são chamados de **vetores de suporte**.

6.1 CASO NÃO LINEARMENTE SEPARÁVEL

Quando temos dados que não são linearmente separáveis, ou seja, que estão sobrepostos, o desenvolvimento acima não é válido. Por isso, refaremos todo esse desenvolvimento, mas de maneira mais geral.

No caso em questão, não conseguimos separar completamente todos os pontos com um hiperplano. É necessário aceitar que, em cada lado do hiperplano, haverá pontos das duas classes, em geral (observar a Figura 6-3). Além disso, haverá ainda pontos que estão do lado certo, mas que estão dentro da margem, ou seja, estão a uma distância do hiperplano menor que M . Essa abordagem, muitas vezes, é chamada de *soft margin classification*, em oposição ao método apresentado anteriormente, que não permitia que pontos estivessem dentro da margem ou do lado errado do hiperplano, chamado muitas vezes de *hard margin classification* [17].



(a) Caso linearmente separável.

(b) Caso não linearmente separável.

Figura 6-3: Possíveis casos para o classificador de vetor de suporte.

Para considerar esses casos, usaremos variáveis $\xi_i, i = 1, \dots, N$, para indicar uma noção de distância que um ponto x_i está da margem da classe a qual ela pertence, na direção contrária ao lado ao qual ela pertence (observando a Figura 6-3b isso fica mais claro). Assim, escreveremos a restrição do nosso problema como

$$y_i(x_i^T \mathbf{w} + w_0) \geq M(1 - \xi_i), \quad (6.6)$$

em que $\xi_i \geq 0$ e $\sum_{i=1}^N \xi_i \leq \text{constante}$. Isso nos deixa com alguns casos diferentes:

- $\xi_i < 1$, indica que o ponto está do lado correto do hiperplano, mas dentro da margem;
- $\xi_i = 0$, indica que o ponto está do lado certo do hiperplano e está fora da margem, independentemente de sua distância para a margem;
- $\xi_i \geq 1$ indica que o ponto está do lado errado do hiperplano (ou exatamente em cima do hiperplano, no caso em que $\xi_i = 1$).

Com isso, vemos que ξ_i indica a distância de um ponto para a margem certa de maneira relativa. Talvez uma forma mais natural de escrever essa restrição fosse $y_i(x_i^T \mathbf{w} + w_0) \geq M - \xi_i$, pois, nesse caso, ξ_i está de fato indicando a distância absoluta para a margem. No entanto, essa forma de escrever a restrição não facilita o resto do desenvolvimento e, por isso, a restrição adotada será na forma da eq. (6.6).

Logo, seguindo a lógica adotada no problema linearmente separável, o problema pode ser escrito como

$$\begin{aligned} & \text{minimizar}_{\mathbf{w}, w_0} \quad \frac{1}{2} \|\mathbf{w}\|^2 \\ & \text{sujeito a} \quad y_i(\mathbf{x}_i^T \mathbf{w} + w_0) \geq 1 - \xi_i \\ & \quad \quad \quad \xi_i \geq 0 \\ & \quad \quad \quad \sum \xi_i \leq \text{constante} \end{aligned}$$

Uma forma mais conveniente de reescrever esse problema é a seguinte:

$$\begin{aligned} & \text{minimizar}_{\mathbf{w}, w_0} \quad \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i \\ & \text{sujeito a} \quad y_i(\mathbf{x}_i^T \mathbf{w} + w_0) \geq 1 - \xi_i \\ & \quad \quad \quad \xi_i \geq 0 \end{aligned} \tag{6.7}$$

Dessa forma, temos um parâmetro “custo” C na função objetivo que assume um papel similar ao da constante que limita $\sum \xi_i$. Quando $C = \infty$, temos o caso separável.

Como o problema de otimização é convexo, podemos seguir passos análogos aos que foram adotados anteriormente para resolver esse problema. A função de Lagrange é dada por:

$$L(\mathbf{w}, w_0, \boldsymbol{\xi}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i + \sum_{i=1}^N \lambda_i [1 - y_i(\mathbf{x}_i^T \mathbf{w} + w_0) - \xi_i] - \sum_{i=1}^N \mu_i \xi_i.$$

Como a função de Lagrange dual é $g(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \inf_{\mathbf{w}, w_0, C} L(\mathbf{w}, w_0, \boldsymbol{\xi}, \boldsymbol{\lambda}, \boldsymbol{\mu})$, derivamos e igualamos a zero:

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{w}} &= \mathbf{w} - \sum_{i=1}^N \lambda_i y_i \mathbf{x}_i = 0 \Rightarrow \mathbf{w} = \sum_{i=1}^N \lambda_i y_i \mathbf{x}_i \\ \frac{\partial L}{\partial w_0} &= - \sum_{i=1}^N \lambda_i y_i w_0 = 0 \Rightarrow \sum_{i=1}^N \lambda_i y_i = 0 \end{aligned} \tag{6.8}$$

$$\frac{\partial L}{\partial \xi_i} = C - \lambda_i - \mu_i = 0 \Rightarrow \lambda_i = C - \mu_i \text{ ou } \mu_i = C - \lambda_i \tag{6.9}$$

Usando as equações acima, encontramos a seguinte expressão para função de Lagrange dual:

$$\begin{aligned} g(\boldsymbol{\lambda}, \boldsymbol{\mu}) &= \frac{1}{2} \left(\sum_{i=1}^N \lambda_i y_i \mathbf{x}_i^T \right) \left(\sum_{i=1}^N \lambda_i y_i \mathbf{x}_i \right) + C \sum_{i=1}^N \xi_i + \sum_{i=1}^N \lambda_i - \sum_{i=1}^N \lambda_i y_i \mathbf{x}_i^T \sum_{j=1}^N \lambda_j y_j \mathbf{x}_j \\ &\quad - \sum_{i=1}^N \xi_i (C - \mu_i) - \sum_{i=1}^N (C - \lambda_i) \xi_i \\ g(\boldsymbol{\lambda}, \boldsymbol{\mu}) &= \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \lambda_i \lambda_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^N \lambda_i - \sum_{i=1}^N \sum_{j=1}^N \lambda_i \lambda_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \\ g(\boldsymbol{\lambda}, \boldsymbol{\mu}) &= \sum_{i=1}^N \lambda_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \lambda_i \lambda_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \end{aligned}$$

Lembrando que é preciso considerar as condições de KKT para podermos resolver esse problema, que são dadas por:

1. Restrição 1: $y_i(x_i^T \mathbf{w} + w_0) \geq 1 - \xi_i$ ou $y_i(x_i^T \mathbf{w} + w_0) - (1 - \xi_i) \geq 0$ (6.10)

2. Restrição 2: $\xi_i \geq 0$ (6.11)

3. Condição do Lagrangiano 1: $\lambda_i \geq 0$ (6.12)

4. Condição do Lagrangiano 2: $\mu_i \geq 0$ (6.13)

5. Complementary slackness 1: $\lambda_i[y_i(x_i^T \mathbf{w} + w_0) - (1 - \xi_i)] = 0$ (6.14)

6. Complementary slackness 2: $\mu_i \xi_i = 0$ (6.15)

7. Gradiente nulo 1: $\mathbf{w} = \sum_{i=1}^N \lambda_i y_i \mathbf{x}_i$ (6.16)

8. Gradiente nulo 2: $\sum_{i=1}^N \lambda_i y_i = 0$ (6.17)

9. Gradiente nulo 3: $C - \lambda_i - \mu_i = 0 \Rightarrow \lambda_i = C - \mu_i$ ou $\mu_i = C - \lambda_i$ (6.18)

A condição dada pela eq. (6.14), pode ser escrita como $\lambda_i[y_i f(\mathbf{x}_i) - (1 - \xi_i)] = 0$. A partir dessa equação e condição dada pela eq.(6.16), vemos que, assim como no caso linearmente separável, \mathbf{w} é uma combinação linear de apenas alguns pontos \mathbf{x}_i , que são chamados de **vetores de suporte**. Nesse caso, os únicos pontos que não são vetores de suporte são aqueles que estão do lado certo do hiperplano e fora da margem (sem estar em cima da borda da margem). Isso pode ser percebido ao observarmos as diferentes possibilidades para \mathbf{x}_i :

- Quando $\xi_i = 0$ (pontos do lado correto do hiperplano em cima da borda da margem ou fora da margem), temos uma situação parecida com o caso linearmente separável:
 - Quando $y_i f(\mathbf{x}_i) > 1$, ou seja, \mathbf{x}_i está do lado certo do hiperplano, mas fora da margem, $\lambda_i = 0$ e, portanto, esse ponto não é contabilizado na eq. (6.16).
 - Quando $\lambda_i > 0$, temos $y_i f(\mathbf{x}_i) = 1$, ou seja, o ponto está exatamente em cima da borda da margem. Como $\lambda_i > 0$, o ponto é contabilizado.
 - Lembrar que uma das condições de KKT é que $y_i f(\mathbf{x}_i) \geq 1 - \xi_i = 1$, então, não haverá o caso $y_i f(\mathbf{x}_i) < 1$.

É interessante reparar que, nesse caso, pela eq. (6.18), $0 \leq \lambda_i \leq C$.

- Quando $\xi_i > 0$ (pontos do lado correto do hiperplano dentro da margem ou pontos do lado incorreto do hiperplano), $y_i f(\mathbf{x}_i) = 1 - \xi_i$. Logo, $\lambda_i \geq 0$ e todos os pontos são contabilizados na eq. (6.16).

Isso acontece porque $y_i f(\mathbf{x}_i) = r \|\mathbf{w}\| = \frac{r}{M}$ representa a distância normalizada de um ponto \mathbf{x}_i para o hiperplano, sendo que ela será positiva quando o ponto estiver do lado certo do hiperplano e negativa caso contrário. Quando $\xi_i = 0$, isso não é necessariamente verdade, pois, esse caso inclui tanto os pontos que estão em cima da borda da margem, em que de fato $y_i f(\mathbf{x}_i) = 1 - \xi_i = 1$, mas também inclui os pontos corretamente classificados que estão fora da margem, em que $y_i f(\mathbf{x}_i) > 1 - \xi_i = 1$. Dois exemplos ajudam a entender isso:

- Se $0 < \xi_i < 1$, o ponto está do lado correto do hiperplano, mas dentro da margem, e $1 - \xi_i > 0$. Se, por exemplo, $\xi_i = 0,4$, sua distância normalizada da margem é dada por $\xi_i = 0,4$ e sua distância normalizada para o hiperplano é dada por $\frac{r}{M} = y_i f(\mathbf{x}_i) = 1 - \xi_i = 0,6$.
- Se $\xi_i \geq 1$, o ponto está do lado errado do hiperplano e $1 - \xi_i \leq 0$. Se, por exemplo, $\xi_i = 3$, sua distância normalizada da margem é dada por $\xi_i = 3$ e sua distância normalizada para o hiperplano é $\frac{r}{M} = y_i f(\mathbf{x}_i) = -(\xi_i - 1) =$

$1 - \xi_i = -2$. Lembrando que o sinal de menos em $-(\xi_i - 1)$ é usado porque $y_i f(\mathbf{x}_i)$ é negativo quando o ponto está do lado errado do hiperplano.

É interessante reparar que, nesse caso, pelas eqs. (6.15) e (6.18), $\lambda_i = C$.

Depois de \mathbf{w} e w_0 terem sido determinados, para classificar novos pontos, basta analisar o sinal de $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$.

C é um parâmetro de *tunning*. Quanto maior o valor de C , menores deverão ser os valores de ξ_i para poder minimizar o problema (6.7). Logo, menor será a margem. De forma análoga, quanto menor for C , maiores serão os valores de ξ_i . A Figura 6-4 ilustra essas situações.

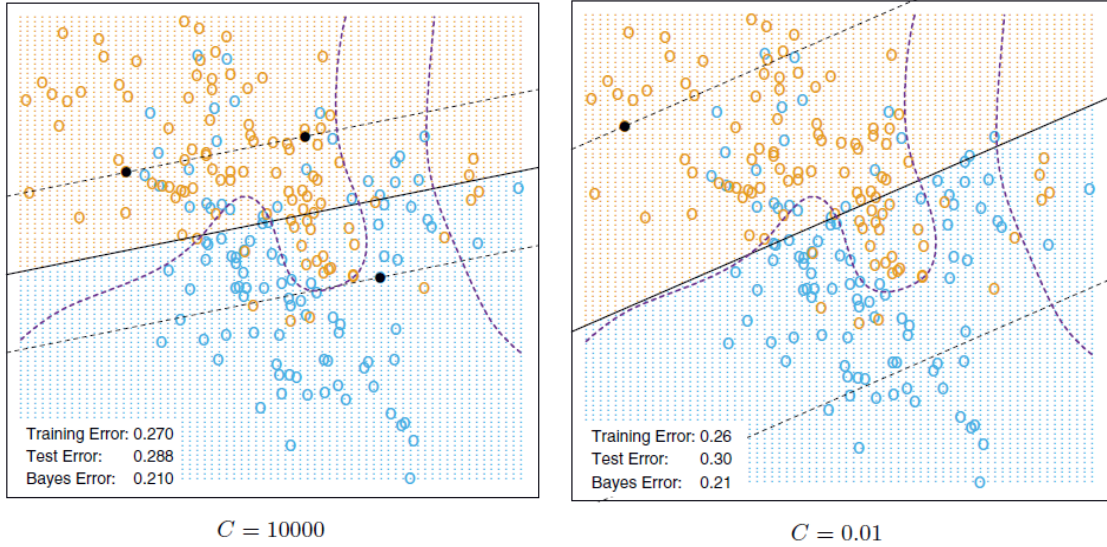


Figura 6-4: Margens do classificador de vetor de suporte para diferentes valores de C .

6.2 MÁQUINA DE VETOR DE SUPORTE

Como vimos acima, o classificador de vetor de suporte encontra fronteiras lineares de decisão no espaço das entradas \mathbf{x} . No entanto, é possível tornar esse procedimento mais flexível aumentando esse espaço usando funções base, criando assim, um espaço de *features*. Quando aplicamos o método dessa maneira, encontramos uma fronteira de decisão linear no espaço das *features*, porém não-linear no espaço das entradas.

Essa utilização do método do classificador de vetor de suporte com a possibilidade de trabalhar em um outro espaço de dimensão maior (possivelmente infinito) dá origem ao modelo *support vector machine* (SVM).

O procedimento para encontrar a fronteira de decisão desse modelo é, basicamente, o mesmo que foi desenvolvido na Seção 6.1, bastando apenas substituir as entradas \mathbf{x} pelas suas funções correspondentes $\phi(\mathbf{x})$. Então, com base na eq. (6.16), podemos escrever

$$f(\mathbf{x}) = \phi(\mathbf{x})^T \mathbf{w} + w_0 = \sum_{i=1}^N \lambda_i y_i \langle \phi(\mathbf{x}), \phi(\mathbf{x}_i) \rangle_{\mathbb{H}} + w_0. \quad (6.19)$$

É importante dizer que $\phi(\mathbf{x})$ é uma aplicação definida como

$$\begin{aligned}\phi: \mathbb{R}^d &\rightarrow \mathbb{H} \\ \mathbf{x} &\mapsto \phi(\mathbf{x})\end{aligned}$$

em quem d é a dimensão dos dados e \mathbb{H} é um *reproducing kernel Hilbert Space* (RKHS) de funções reais. $\phi(\mathbf{x}) = [\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \dots]^T$ é chamada de *feature map* e pode ter dimensão finita ou infinita. Para mais detalhes sobre o RKHS, consultar [1].

Observar que, na eq. (6.19), o produto interno que existia originalmente na eq. (6.16), $\mathbf{x}^T \mathbf{x}_i = \langle \mathbf{x}^T \mathbf{x}_i \rangle$, produto interno do espaço Euclidiano, foi substituído pelo produto interno dos *features maps* $\langle \phi(\mathbf{x}), \phi(\mathbf{x}_i) \rangle_{\mathbb{H}}$. O subscrito \mathbb{H} serve para indicar que o produto interno entre essas duas aplicações não é mais um produto interno do espaço Euclidiano, pois essa aplicação pertence ao espaço de Hilbert, que é um espaço de produto interno $\langle \cdot, \cdot \rangle_{\mathbb{H}}$.

Todo RKHS apresenta uma função kernel $\kappa(\cdot, \cdot)$ que o define. Além disso, nesse espaço, temos que $\kappa(\mathbf{x}_1, \mathbf{x}_2) = \langle \phi(\mathbf{x}_1), \phi(\mathbf{x}_2) \rangle$. Logo, podemos reescrever a eq. (6.19) como

$$f(\mathbf{x}) = \phi(\mathbf{x})^T \mathbf{w} + w_0 = \sum_{i=1}^N \lambda_i y_i \kappa(\mathbf{x}, \mathbf{x}_i) + w_0. \quad (6.20)$$

Com isso, vemos que não é necessário conhecer as *features* $\phi_i(\mathbf{x})$ definidas por $\phi(\mathbf{x})$ para determinar a fronteira de decisão. Basta conhecer a função kernel, que normalmente, é uma escolha de *design* do modelo SVM. Escolhas populares para o kernel são:

- Kernel polinomial: $\kappa(\mathbf{x}, \mathbf{y}) = (1 + \langle \mathbf{x}, \mathbf{y} \rangle)^d$
- Kernel Gaussiano: $\kappa(\mathbf{x}, \mathbf{y}) = e^{-\gamma \|\mathbf{x} - \mathbf{y}\|_2^2}$
- Sigmoid kernel: $\kappa(\mathbf{x}, \mathbf{y}) = \tanh(\gamma \langle \mathbf{x}, \mathbf{y} \rangle + c)$

em que $\langle \cdot, \cdot \rangle$ é o produto interno do espaço Euclidiano, pois \mathbf{x} e \mathbf{y} representam entradas.

6.2.1 SVM como Método de Penalização

O problema de minimização do modelo SVM é similar ao do classificador de vetor de suporte, definido na eq. (6.7), porém substituindo \mathbf{x} por $\phi(\mathbf{x})$ quando necessário. Assim, omitindo as restrições, o problema de minimização consiste em

$$\min_{\mathbf{w}, w_0} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i.$$

Podemos reescrever esse problema da seguinte forma:

$$\begin{aligned}& \min_{\mathbf{w}, w_0} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N [1 - y_i f(\mathbf{x})] \mathbb{1}_{\{A\}}(\mathbf{x}) \\& \equiv \min_{\mathbf{w}, w_0} \frac{1}{2C} \|\mathbf{w}\|^2 + \sum_{i=1}^N [1 - y_i f(\mathbf{x})] \mathbb{1}_{\{A\}}(\mathbf{x}) \\& \equiv \min_{\mathbf{w}, w_0} \overbrace{\frac{\lambda}{2} \|\mathbf{w}\|^2}^{\text{penalidade}} + \sum_{i=1}^N \overbrace{[1 - y_i f(\mathbf{x})] \mathbb{1}_{\{A\}}(\mathbf{x})}^{\text{loss}}\end{aligned}$$

em que $\lambda = 1/C$, $A = \{\mathbf{x} : 1 - y_i f(\mathbf{x})\}$ e $f(\mathbf{x}) = \phi(\mathbf{x})^T \mathbf{w} + w_0$.

Com isso, vemos que o problema de minimização do modelo SVM tem a forma *loss* + penalidade, típica de um problema de otimização com regularização.

A função de *loss* $\ell(y, f) = [1 - y_i f(\mathbf{x})] \mathbb{1}_{\{A\}}(\mathbf{x})$ é chamada de *hinge loss* (*loss* articulação), devido ao seu formato, como mostrado na Figura 6-5. Nessa imagem, outras funções de *loss* são mostradas para efeito de comparação. Para mais detalhes, consultar [7].

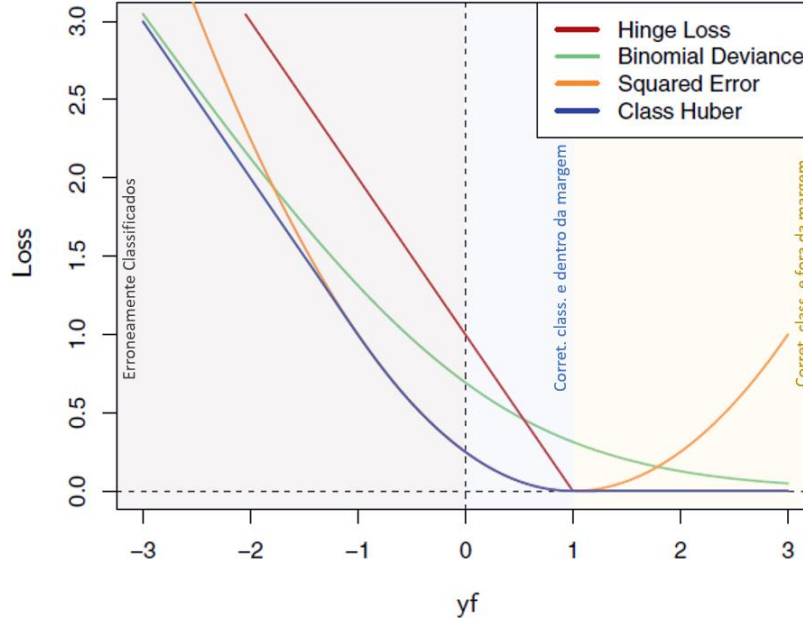


Figura 6-5: Comparação de diferentes funções de *loss*.

Com isso, vemos que a *hinge loss* atribui uma perda nula para os termos para os quais $y_i f(\mathbf{x}_i) \geq 1$, ou seja, para todos os pontos que tenham sido classificados corretamente e que estejam fora da margem. Além disso, atribui uma perda linear para os pontos dentro da margem ou classificados erroneamente.

Um detalhe importante para se atentar é que, só encontramos a expressão da *hinge loss* na modelagem do SVM por causa de algumas suposições feitas, como $M = \frac{1}{\|\mathbf{w}\|}$ e $y_i f(\mathbf{x}_i) \geq M(1 - \xi_i)$. Se outras escolhas tivessem sido feitas, obteríamos outra *loss*.

6.2.2 Classificação Multiclasse

O mais comum é adotar a abordagem *one-vs-the-rest*. Para corrigir o problema das regiões de ambiguidade, diferentes técnicas podem ser adotadas, como mostrado em [16, pp. 338-339].

Notar que, no modelo da regressão logística, esse problema era facilmente resolvido ao atribuir uma função discriminante $\delta_k(\mathbf{x}) = \mathbb{P}(\mathbf{x} \in \omega_k | \mathbf{x})$ para cada uma das K funções e estabelecer a regra de que $\mathbf{x} \in \omega_i$ se $\delta_i(\mathbf{x}) > \delta_j(\mathbf{x}) \forall i \neq j$. Lembrando que a variável aleatória nesse caso é ω_k , que é o conjunto dos pontos pertencentes à classe k . Nesse caso, não existirá regiões de ambiguidade.

No entanto, no SVM, isso não é possível, pois não temos uma função discriminante que representa a probabilidade a posteriori de o dado pertencer à classe k .

6.3 SVM PARA REGRESSÃO

Nas seções anteriores, tínhamos dados que pertenciam a duas classes diferentes e utilizávamos um hiperplano $f(x) = 0$ para separá-los, considerando que existe uma margem ao redor desse hiperplano.

Para o caso da regressão, é necessário adaptar o método, já que o nosso problema não é mais de separação dos dados. A adaptação que pode ser feita é considerar que, agora, $f(x) = \mathbf{x}^T \mathbf{w} + w_0$ representará uma curva de regressão que tentará aproximar a função geradora dos dados $\mathcal{G}(x) = y$. Para isso, consideraremos que os dados podem estar exatamente em cima da curva $f(x)$ ou dentro de uma margem de espessura ϵ ao redor da curva, como mostra a Figura 6-6.

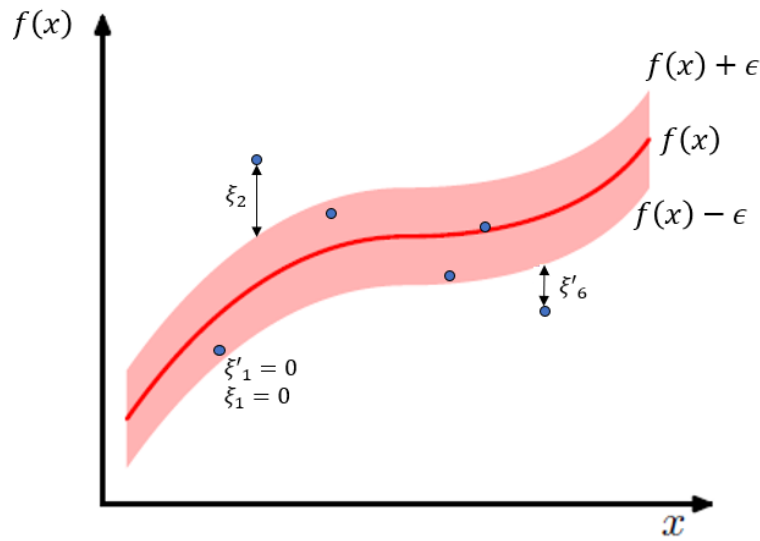


Figura 6-6: Modelo de SVM para regressão.

Podemos escrever esse problema no formato $\sum \text{loss} + \text{regularização}$, sujeito à condição de que os pontos devem estar dentro do “tubo- ϵ ” que envolve $f(x)$. Com isso, o método acima se resume a

$$\begin{aligned} \min_{\mathbf{w}, w_0} \quad & C \sum_{i=1}^N V_{\epsilon}[y_i - f(x_i)] + \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{sujeito a} \quad & f(x_i) + \epsilon \geq y_i \\ & f(x_i) - \epsilon \leq y_i \end{aligned}$$

em que V_{ϵ} é a função de erro ϵ -insensível, dada por

$$V_{\epsilon}(r) = \begin{cases} 0, & |r| < \epsilon \\ |r| - \epsilon, & \text{c. c.} \end{cases}$$

ou seja, desprezamos todo erro $|r|$ abaixo de ϵ .

No entanto, nem sempre vamos conseguir fazer com que todos os pontos se encontrem dentro do tubo- ϵ . Por isso, assim como no caso da classificação, introduzimos variáveis ξ_i para

permitir a contabilização de pontos que não estejam dentro do tubo- ϵ . No entanto, na regressão, precisaremos de variáveis $\xi_i \geq 0$ para os pontos acima da curva $f(\mathbf{x})$ e variáveis $\xi'_i \geq 0$ para pontos abaixo da curva $f(\mathbf{x})$. Essas variáveis são definidas como $\xi_i = \{y_i - [f(\mathbf{x}_i) + \epsilon]\} \mathbb{1}_{\{y_i > f(\mathbf{x}_i) + \epsilon\}}(\mathbf{x}_i)$ e $\xi'_i = \{[f(\mathbf{x}_i) - \epsilon] - y_i\} \mathbb{1}_{\{y_i < f(\mathbf{x}_i) - \epsilon\}}(\mathbf{x}_i)$.

Incluindo essas variáveis no problema, temos um novo problema de otimização, dado por

$$\begin{aligned} \min_{\mathbf{w}, w_0} \quad & C \sum_{i=1}^N (\xi_i + \xi'_i) + \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{sujeito a} \quad & f(\mathbf{x}_i) + \epsilon + \xi_i \geq y_i \\ & f(\mathbf{x}_i) - \epsilon - \xi'_i \leq y_i \\ & \xi_i \geq 0 \\ & \xi'_i \geq 0 \end{aligned}$$

Se repararmos, temos algo muito parecido com o que tínhamos no problema (6.7). Com isso, vemos que é preferível ter ξ_i e ξ'_i menores possíveis, pois ξ_i e ξ'_i grandes levam a um custo maior.

Com isso, o Lagrangiano pode ser escrito como

$$\begin{aligned} L(\mathbf{w}, w_0, \xi, \xi') = & C \sum_{i=1}^N (\xi_i + \xi'_i) + \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^N \lambda_i [f(\mathbf{x}_i) + \epsilon - y_i] - \sum_{i=1}^N \lambda'_i [-f(\mathbf{x}_i) + \epsilon + y_i] \\ & - \sum_{i=1}^N (\mu_i \xi_i + \mu'_i \xi'_i) \end{aligned}$$

em que $\lambda_i \geq 0$, $\lambda'_i \geq 0$, $\mu_i \geq 0$ e $\mu'_i \geq 0$ são os multiplicadores de Lagrange.

A função dual de Lagrange é dada por

$$\begin{aligned} g(\lambda, \lambda') = & \inf_{\mathbf{w}, w_0} L(\mathbf{w}, w_0, \xi, \xi') = L(\mathbf{w}^*, w_0^*, \xi^*, \xi'^*) \\ = & -\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N (\lambda_i - \lambda'_i)(\lambda_j - \lambda'_j) \langle \mathbf{x}_i, \mathbf{x}_j \rangle - \epsilon \sum_{i=1}^N (\lambda_i + \lambda'_i) + \sum_{i=1}^N (\lambda_i + \lambda'_i) y_i \end{aligned}$$

com

$$\mathbf{w}^* = \sum_{i=1}^N (\lambda'_i - \lambda_i) \mathbf{x}_i.$$

Para terminar de resolver o problema de otimização, basta maximizar $g(\lambda, \lambda')$ sujeito às condições de KKT. Duas dessas condições, advindas da propriedade *complementary slackness*, são interessantes para serem analisadas, como mostradas abaixo:

$$\lambda_i [\epsilon + f(\mathbf{x}_i) + \xi_i - y_i] = 0 \quad (6.21)$$

$$\lambda'_i [\epsilon - f(\mathbf{x}_i) + y_i + \xi'_i] = 0. \quad (6.22)$$

A partir da eq. (6.21), vemos que $\lambda_i \neq 0$ somente se $[\epsilon + f(\mathbf{x}_i) + \xi_i - y_i] = A = 0$, ou seja, se o ponto estiver fora do tubo- ϵ ou exatamente em sua borda. Isso porque

- se o ponto está fora do tubo- ϵ , então $y_i = f(x_i) + \epsilon + \xi_i$ e $A = 0$;
- se o ponto está na borda do tubo- ϵ , então $\xi_i = 0$, $y_i = f(x_i) + \epsilon$ e $A = 0$.

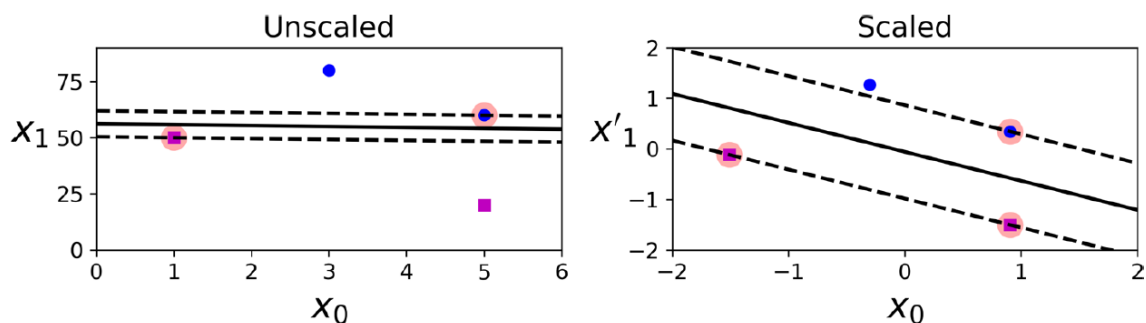
No caso em que o ponto está dentro da margem $A = \epsilon + f(x_i) + \xi_i - y_i = \epsilon + f(x_i) - y_i > 0$, pois $y_i < \epsilon + f(x_i)$.

O raciocínio é análogo para λ'_i na eq. (6.22). Com isso, vemos que os únicos pontos x_i que contribuem para w^* são aqueles que estão fora do tubo- ϵ ou exatamente em cima de sua borda.

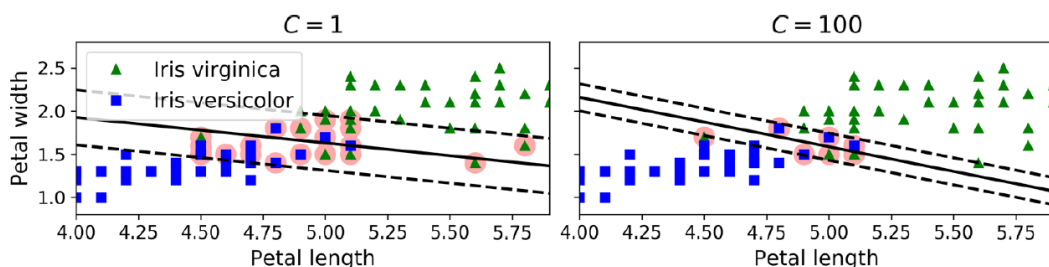
Assim, como foi feito no caso da classificação, todo esse desenvolvimento pode ser generalizado para o caso em que estamos trabalhando no espaço das *features*. Basta substituir as entradas x_i pelas *features* ϕ_i e o produto interno Euclidiano $\langle x_i, x'_i \rangle$ pela função kernel $\kappa(x_i, x'_i)$.

6.4 ASPECTOS PRÁTICOS

- O modelo SVM é mais adequado para *datasets* de tamanho pequeno/médio, pois ele não escala bem com o número de dados, tendo complexidade entre $\mathcal{O}(m^2 \times n)$ e $\mathcal{O}(m^3 \times n)$ para a classe SVC do scikit-learn, em que m é o número de exemplos e n o número de *features* [17]. Para kernel linear, dar preferência para a classe LinearSVC do scikit-learn, pois o algoritmo tem complexidade $\mathcal{O}(m \times n)$.
- É sensível à escala.



- Baixos valores de C aumentam a regularização do modelo, deixando a margem menos estreita.



- Poderíamos aplicar um classificador de vetor de suporte *soft* para separar dados não-linearmente separáveis. No entanto, precisaríamos criar novas *features* a partir das variáveis de entrada, de forma a encontrar uma fronteira de decisão adequada (e não linear). Criar *features* polinomiais geralmente é uma boa opção. Para polinômios de grau pequeno, isso pode até ser viável, mas para polinômios de grau elevado, isso se torna inviável. A partir disso, podemos ver a vantagem do SVM sobre métodos lineares

que utilizam transformações para gerar novas *features*: não é preciso criar novas *features*, apenas escolher a função *kernel* adequada.

- As funções *kernel* mais comumente usadas são a linear e a *Gaussian Radial Basis Function* (RBF). Como regra de ouro para escolher a função *kernel* na prática, pode-se começar com a linear (dando preferência para o LinearSVC, em vez do SVC do scikit-learn, pela eficiência computacional), especialmente se o conjunto de dados for muito grande. Caso o conjunto de dados não seja tão grande, pode-se testar em seguida a *Gaussian RBF*.

Capítulo 7: ÁRVORE DE DECISÃO

A árvore de decisão é um modelo de aprendizado supervisionado que funciona dividindo o espaço dos atributos em regiões cuboides, cujas arestas são alinhadas com os eixos, e treinando um modelo simples (como uma constante) a cada uma dessas regiões. Existem diferentes métodos de construção de uma árvore de decisão, como o C4.5, C5.0 e o CART (*Classification and Regression Tree*), mas o mais popular é o último [16] e, por isso, é nele que o resto do capítulo se baseia.

O modelo da árvore de decisão pode ser aplicado tanto para problemas de classificação (nesse caso, ela recebe o nome de árvore de classificação) quanto de regressão (nesse caso, ela recebe o nome de árvore de regressão).

Uma das vantagens da árvore de decisão é que ela pode ser representada na forma de uma árvore binária, como mostra a Figura 7-1, o que torna sua visualização e interpretação mais simples. O primeiro nó é chamado de **raiz**, os nós das extremidades da árvore, de onde não partem outros nós, são chamados de **folhas** e os nós restantes são chamados de **nós internos**. Da raiz e dos nós internos sempre partem duas ramificações, que dão origem ao nó filho esquerdo e direito. A **profundidade** de um nó é a distância entre esse nó e a raiz da árvore, ou seja, é o número de passos que se dá para chegar na raiz a partir do nó em questão.

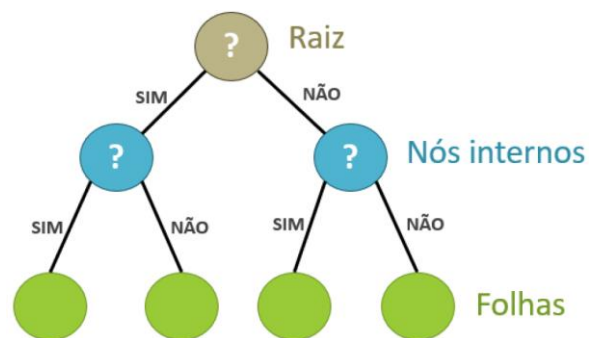


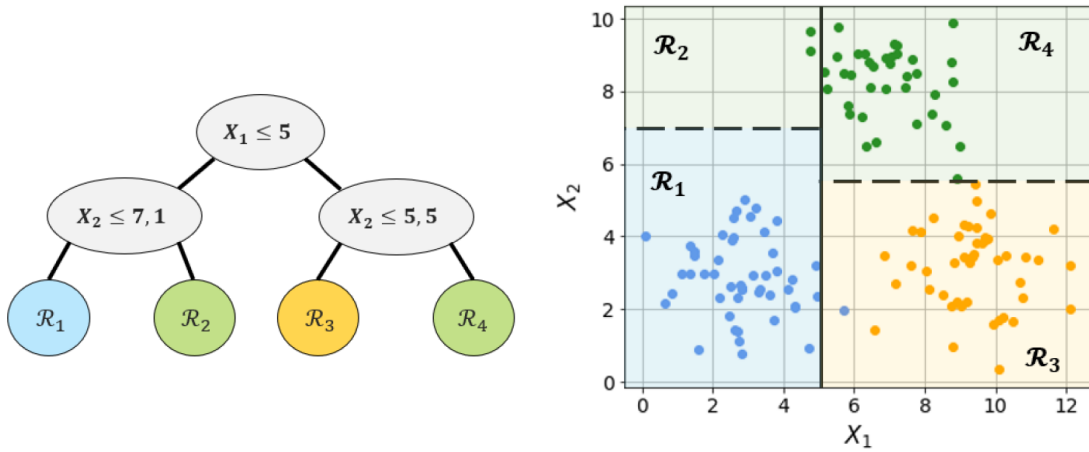
Figura 7-1: Diagrama de uma árvore de decisão.

A cada um dos nós internos e à raiz é associada uma condição do tipo $X_j < t$, em que X_j é um atributo e t uma constante. Considere certa instância que tem x_i como entrada. À medida que essa instância percorre a árvore, começando na raiz, toda vez que ela satisfizer a condição de um nó, ele deve descer um nível tomando o caminho da esquerda, caso contrário, ele deve tomar o caminho da direita. O percurso acaba quando se chega a uma folha que apresenta o resultado da classificação/regressão.

7.1 ÁRVORE DE CLASSIFICAÇÃO

Nas árvores de classificação, a cada folha, associamos uma classe do problema. A Figura 7-2a ilustra o exemplo de uma árvore desse tipo que foi construída com base em um conjunto de dados de treinamento com atributos X_1 e X_2 e que apresenta 3 classes (representadas pelas cores azul, verde e amarelo). Com auxílio da Figura 7-2b, é possível perceber que, a cada bifurcação dessa árvore, o espaço dos atributos é dividido em uma região $X_j < s$ e outra $X_j > s$, associadas aos nós filhos esquerdo e direito, respectivamente. A classificação que um dado exemplo x_i recebe ao chegar a um nó folha de índice m depende, portanto, da região \mathcal{R}_m que

esse nó representa. Assim, se um certo exemplo estiver localizado na região \mathcal{R}_3 , ele será classificado como amarelo.



(a) Representação da árvore de decisão.

(b) Dados divididos por região.

Figura 7-2: Exemplo de uma árvore de decisão aplicada a um conjunto de dados de dois atributos e três classes.

Considere um conjunto de dados de treinamento com N exemplos, com entradas x_i e saídas y_i , em que $i \in \{1, 2, \dots, N\}$. Para a construção de uma árvore de decisão com base nesses dados, o algoritmo parte do nó raiz. Como, inicialmente, a árvore só apresenta um único nó, ou seja, não apresenta bifurcações, o espaço dos atributos ainda não foi segmentado e o nó raiz representa todo o espaço dos atributos, que contém todos os dados de treinamento.

Primeiramente, o algoritmo precisa encontrar o par (X_j, s) que divide os dados em duas regiões da melhor forma possível. Para isso, é necessário definir uma **medida de impureza** e encontrar os valores de X_j e s que minimizam a soma ponderada das impurezas das duas regiões resultantes da divisão do nó raiz. Após ter encontrado o melhor par (X_j, s) , o nó raiz é dividido em duas regiões, uma para cada nó filho, e esse processo é repetido para cada novo nó. Considerando uma árvore T qualquer (ainda em processo de construção), um nó terminal m (associado à região \mathcal{R}_m), os eventuais futuros nós filhos esquerdo m_E e direito m_D desse nó m e uma métrica de impureza $Q_m(T)$ (isso ficará mais claro adiante), podemos escrever matematicamente o processo de encontrar o melhor par (X_j, s) como

$$(X_j, s) = \arg \min_{X_j, s} \frac{N_{m_E} Q_{m_E}(T) + N_{m_D} Q_{m_D}(T)}{N_m}, \quad (7.1)$$

em que N_{m_E} e N_{m_D} correspondem ao número de instâncias de treinamento pertencentes às regiões \mathcal{R}_{m_E} e \mathcal{R}_{m_D} após uma eventual divisão do nó pai m e $N_m = N_{m_E} + N_{m_D}$ é o total de instâncias na região \mathcal{R}_m do nó pai. Portanto, vemos que a busca pelo melhor par (X_j, s) consiste em minimizar a impureza média ponderada da divisão do nó pai (lembrando que, como N_m não depende de X_j nem de s , ele pode ser desconsiderado da minimização).

É importante mencionar que, encontrar o particionamento que minimize a impureza de toda a árvore é praticamente impossível computacionalmente. Por isso, esse processo é feito de maneira gulosa, ou seja, os nós vão sendo divididos da raiz até as folhas e, dado que um nó já foi dividido, o par (X_j, s) desse nó não pode ser reconsiderado. Além disso, para cada variável de particionamento X_j , o valor de s é encontrado ao avaliar os diferentes valores dessa variável no conjunto de treinamento.

Por fim, depois de a árvore de classificação já ter sido construída, é necessário atribuir uma classe a cada folha. Essa atribuição é feita levando em consideração a proporção \hat{p}_{mk} de instâncias de treinamento pertencente à classe k presentes na região \mathcal{R}_m associada à folha m . A categoria atribuída à folha é aquela que está em maior proporção, ou seja, a categoria da folha m é dada por $k(m) = \arg \max_k \hat{p}_{mk}$. Na Figura 7-2b, por exemplo, a folha associada à região \mathcal{R}_3 contém 48 exemplos, sendo 47 amarelos (97,9 %), e 1 azul (2,1 %). Portanto, essa folha é associada à categoria amarela, pois é a categoria com maior frequência nesse nó.

Retornando à eq. (7.1), diferentes métricas $Q_m(T)$ podem ser utilizadas. Algumas delas são listadas abaixo:

Índice de Gini

$$Q_m(T) = \sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = 1 - \sum_{k=1}^K \hat{p}_{mk}^2$$

A igualdade acima vem do fato que $\sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_{k=1}^K \hat{p}_{mk} (1 - \hat{p}_{mk}) = \sum_{k=1}^K \hat{p}_{mk} - \sum_{k=1}^K \hat{p}_{mk}^2 = 1 - \sum_{k=1}^K \hat{p}_{mk}^2$.

Entropia

$$Q_m(T) = - \sum_{k=1}^K \hat{p}_{mk} \log(\hat{p}_{mk})$$

Alguns autores chamam essa métrica de entropia cruzada ou *deviance*. Em [18], encontra-se uma discussão interessante sobre o assunto.

Taxa de erro de classificação

$$Q_m(T) = \frac{1}{N_m} \sum_{n|x_n \in \mathcal{R}_m} \mathbb{1}_{\{t \neq k(m)\}}(t_n) = 1 - \hat{p}_{mk(m)}$$

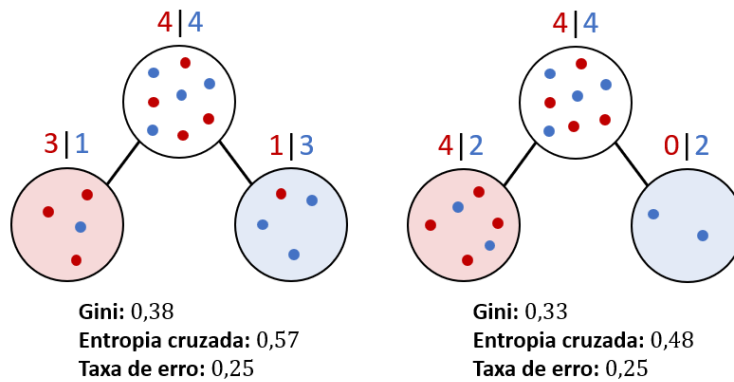


Figura 7-3: Valores da média ponderada da impureza (como na eq. (7.1)) nos nós filhos para as três métricas de impureza apresentada.

Quando todas as instâncias de um nó pertencem a uma mesma classe, esse nó é considerado **puro**. Nesse caso, as três métricas de impureza apresentadas são nulas. Por outro lado, para outros casos, essas métricas têm comportamentos diferentes. O índice de Gini e a entropia

cruzada tendem a apresentar valores menores quanto mais puros forem os nós, o que não é necessariamente verdade para a taxa de erro. Isso pode ser visto a partir do exemplo mostrado na Figura 7-3, em que são mostradas duas possibilidades de divisão de um certo nó. Nos dois casos, a taxa de erro é igual, porém, vemos que as médias ponderadas que usam o índice de Gini e a entropia cruzada são menores no segundo caso, em que a divisão gera nós mais puros. Portanto, como divisões que geram nós mais puros tendem a ser preferíveis [7], o índice de Gini e a entropia cruzada são mais comumente usadas na eq. (7.1) para encontrar os pares (X_j, s) . Além disso, essas duas métricas são diferenciáveis (como fica evidente pela Figura 7-4, que compara as três métricas de impureza apresentadas), o que facilita o processo de otimização numérica. Apesar disso, a taxa de erro de classificação é a métrica mais utilizada em um processo que será visto a diante neste capítulo, chamado de poda por custo-complexidade (*cost-complexity pruning*).

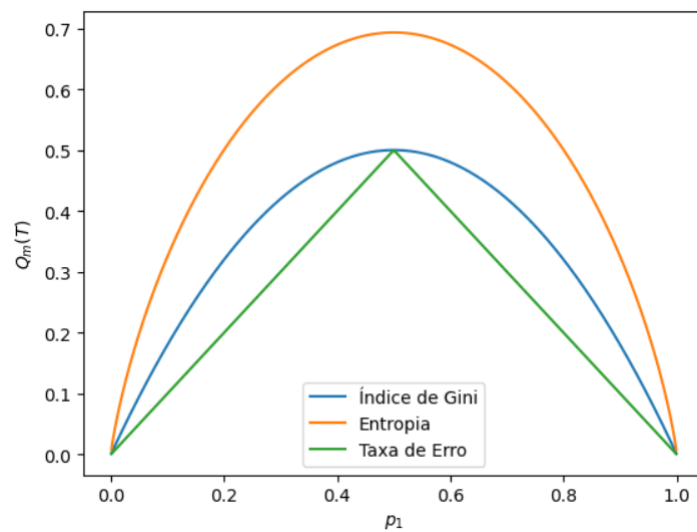


Figura 7-4: Comparação entre diferentes métricas de impureza no caso em que temos duas classes e proporções p_1 e p_2 para as classes 1 e 2.

7.2 ÁRVORE DE REGRESSÃO

Em uma árvore de regressão, a forma como o modelo realiza a predição segue uma lógica similar a da árvore de classificação. Na última, a cada nó folha, era atribuída uma constante k representando uma das classes do problema. No caso da regressão, atribuímos uma constante c_m que representará o valor da predição para qualquer ponto pertencente à região R_m do nó folha m . Para realizar a escolha do valor de c_m , podemos utilizar como critério a minimização do MSE dentro do nó em questão. Ao realizarmos a minimização dessa expressão com respeito a c_m , vemos que o valor ótimo para essa constante deve ser a média amostral dos valores de target, ou seja,

$$\hat{c}_m = \frac{1}{N_m} \sum_{n|x_n \in R_m} t_n.$$

O treinamento de uma árvore de regressão também segue uma lógica muito parecida com a da árvore de classificação, porém, como a natureza do problema é diferente é necessário fazer

outra escolha para $Q_m(T)$. Como a saída do modelo não é mais categórica, não podemos utilizar as métricas de impureza apresentadas anteriormente. Uma opção muito comum é utilizar o MSE, como mostrado abaixo:

$$Q_m(T) = \frac{1}{N_m} \sum_{n|x_n \in \mathcal{R}_m} [t_n - f(x_n)]^2,$$

em que $f(x_n)$ é o valor predito pela árvore para a entrada x_n .

Com a exceção dessas diferenças, todo o funcionamento da árvore de regressão é similar ao da de classificação. À medida que vamos construindo a árvore, também utilizamos a eq. (7.1). Além disso, também podemos utilizar as mesmas estratégias de poda para controlar a complexidade da árvore.

A Figura 7-3 ilustra exemplos de resultados de uma regressão usando uma árvore de decisão para um problema de apenas uma variável. Verificar que, quanto maior o valor do hiperparâmetro max_depth , mais ajustada aos dados é a curva de regressão.

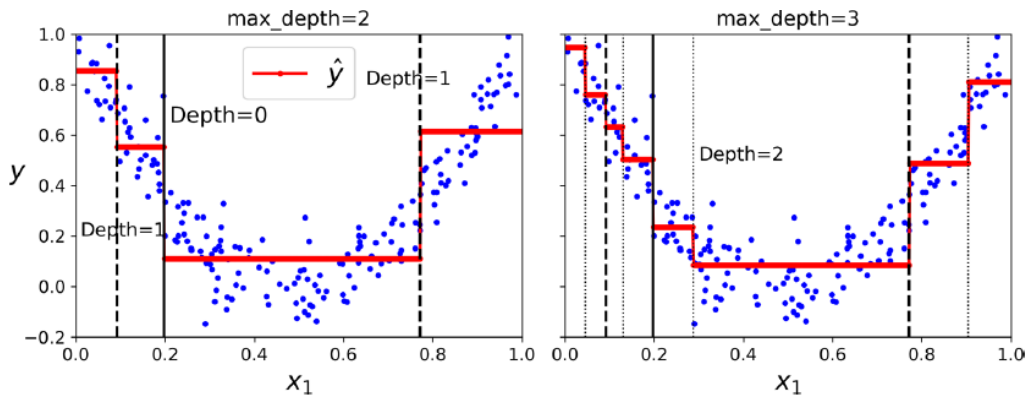


Figura 7-5: Dois exemplos de regressão realizadas nos dados de treinamento com uma árvore de decisão usando dois valores diferentes para o hiperparâmetro “profundidade máxima” (max_depth). A imagem foi retirada de [17].

7.3 REGULARIZAÇÃO

Uma questão sobre o treinamento de uma árvore de decisão que ainda não foi abordada é: quando sabemos que é hora de parar de crescer a árvore? Deixamo-la crescer até que haja apenas uma instância por nó? Até poderíamos fazer isso, porém, isso fariam com que essa árvore sofresse de *overfitting*. Uma solução para evitar esse problema é realizar a poda da árvore usando uma função custo que leva em consideração a complexidade da árvore. Esse método é usualmente chamado de *cost-complexity pruning*. Para aplica-lo, deixamos a árvore T_0 crescer até atingir um determinado número mínimo de instâncias por nó. Após isso, procuramos subárvores $T_{sub} \subset T_0$ que minimizem a função custo

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha |T|,$$

em que $|T|$ é o número de folhas da árvore T .

Com isso, garantimos um *trade-off* entre a pureza dos nós terminais da árvore (primeira parcela) e a complexidade dela (segunda parcela). Quanto maior o valor de α , maior é o efeito da regularização e menor será o tamanho da árvore T_{sub} . Quando temos $\alpha = 0$, não há

regularização. Uma maneira de procurar a melhor subárvore é utilizar o método weakest link pruning (poda do elo mais fraco) [7, p. 308].

Além da poda, existem outros artifícios que podem ser usados para controlar a complexidade de uma árvore de decisão. Os mais comuns consistem em interromper o crescimento da árvore seguindo algum critério de parada. Alguns desses critérios são listados abaixo [16]:

- profundidade máxima da árvore;
- número mínimo de instâncias que deve haver em um nó folha após a divisão;
- número mínimo de instâncias necessárias para dividir um nó folha;
- sendo $\Delta := Q_m(T) - \left(\frac{N_{mE} Q_{mE}(T) + N_{mD} Q_{mD}(T)}{N_m} \right)$ a diminuição de impureza ao se dividir o nó m , se Δ for menor do que um determinado limiar, não dividimos mais o nó;

Capítulo 8: BOOSTING

O *Boosting* é um esquema iterativo, em que a cada iteração um *base learner* (também chamado de *weak learner* [16]) é treinado utilizando um conjunto de treinamento diferente. A geração desses conjuntos é feita ao atribuir pesos às amostras de treinamento, que são mudados a cada iteração. O resultado é um modelo de *boosting* cuja saída é a média ponderada de todos os modelos base (*base learners*) [5].

A Figura 8-1 mostra como é feito o processo de predição em um modelo genérico de *boosting* para uma entrada x . Observar que a saída de cada bloco de *weak learner* é $\phi_k(x; \theta_k)$, que é função que representa o modelo fraco, sendo θ_k o parâmetro associado ao *weak learner* k . Também é possível observar que a saída de cada *weak learner* é ponderada por um peso a_k . A saída do modelo de *boosting*, portanto, é dada por $F(x) = a_1\phi_1(x; \theta_1) + a_2\phi_2(x; \theta_2) + a_3\phi_3(x; \theta_3)$.

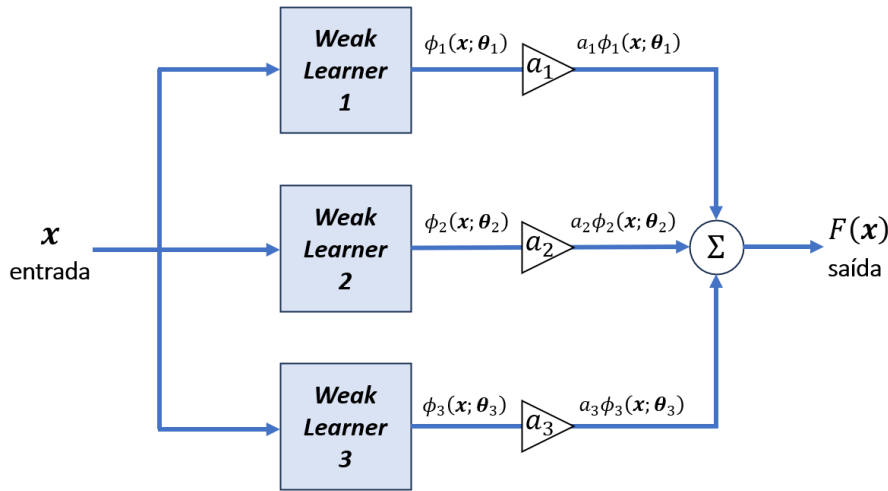


Figura 8-1: Modelo de *boosting* genérico com 3 *weak learners* sendo usado na inferência.

A seguir, são mostrados alguns dos principais modelos de *boosting*.

8.1 ADABOOST

8.1.1 Descrição do Algoritmo

Para definir o AdaBoost (*adaptive boosting*), focamos na tarefa de classificação binária. O objetivo é construir um classificador do tipo

$$f(x) = \text{sgn}\{F(x)\}, \quad (8.1)$$

em que x é uma amostra genérica de um conjunto de dados $\mathcal{D} = \{(x_n, t_n)\}_{n=1}^N$, com $t_n \in \{-1, 1\}$, e $F(x)$ representa o modelo final de *boosting*, dado por

$$F(x) := \sum_{k=1}^K a_k \phi(x; \theta_k),$$

em que a_k são coeficientes que permitem atribuir importância diferente a modelos diferentes e $\phi(x; \theta_k) \in \{-1, 1\}$ representa o modelo base (binário) da iteração k , caracterizado em

termos de um conjunto de parâmetros θ_k . A cada iteração, esses parâmetros são otimizados de maneira gulosa, ou seja, para cada iteração k , encontramos os valores ótimos dos parâmetros θ_k considerando que os parâmetros $\theta_i, i < k$, de outras iterações são fixos.

Para começarmos a detalhar a construção desse modelo, em uma certa iteração i , podemos escrever a soma parcial dos termos, de maneira iterativa, como

$$F_i(\cdot) = \sum_{k=1}^i a_k \phi(\cdot; \theta_k),$$

ou, de maneira recursiva, como

$$F_i(\cdot) = F_{i-1}(\cdot) + a_i \phi(\cdot; \theta_i), \quad i = 1, 2, \dots, K.$$

Em $i = 1$, temos $F_i(\cdot) = a_i \phi(\cdot; \theta_i)$.

Como o algoritmo é executado de maneira gulosa, os termos das iterações anteriores, como F_{i-1} , são considerados conhecidos.

Para encontrar os valores de a_i e θ_i , é realizada uma otimização empregando a **loss exponencial**, dada por $\mathcal{L}(t, F(x)) = e^{-tF(x)}$, que é justamente um dos elementos que caracteriza o AdaBoost.

Se repararmos na Figura 8-2, tanto nos casos em que a classificação foi correta ($tF(x) > 0$), quanto nos casos em que ela foi incorreta ($tF(x) < 0$), a função *loss* tem valor positivo, diferentemente da *loss* 0-1, que vale 1 quando a classificação é incorreta e 0 quando for correta. Mesmo assim, na *loss* exponencial, as classificações erradas levam a uma *loss* maior do que as certas. Além disso, a *loss* exponencial é diferenciável, enquanto a *loss* 0-1 não é.

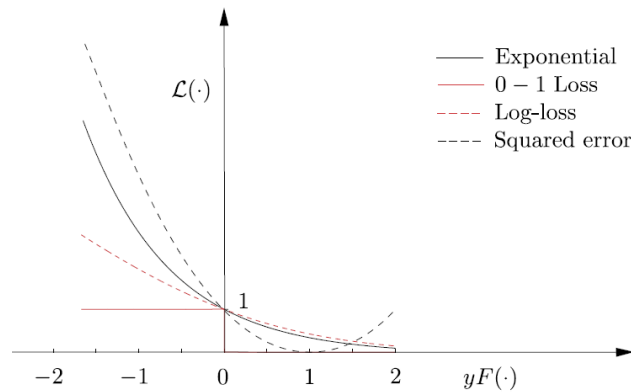


Figura 8-2: Comparação de diferentes funções de *loss* (retirado de [5]).

O problema de minimizar a função custo com a *loss* exponencial é dado por

$$(a_i, \theta_i) = \arg \min_{a, \theta} \sum_{n=1}^N e^{-t_n(F_{i-1}(x_n) + a\phi(x_n; \theta))} \quad (8.2)$$

$$(a_i, \theta_i) = \arg \min_{a, \theta} \sum_{n=1}^N w_n^{(i)} e^{-t_n a \phi(x_n; \theta)} \quad (8.3)$$

em que

$$w_n^{(i)} := \begin{cases} \frac{e^{-t_n F_{i-1}(x_n)}}{Z_{i-1}}, & i > 1 \\ \frac{1}{N}, & i = 1 \end{cases}, \quad (8.4)$$

com $Z_{i-1} = \sum_{n=1}^N e^{-t_n F_{i-1}(x_n)}$ sendo um fator de normalização para garantir que $w_n^{(i)} \in [0, 1]$ e

$$\sum_{n=1}^N w_n = 1. \quad (8.5)$$

Reparar que a inclusão do fator de normalização não altera o resultado da eq. (8.2), pois, como Z_{i-1} é constante para um dado i ,

$$\arg \min_{a, \theta} \sum_{n=1}^N e^{-t_n (F_{i-1}(x_n) + a\phi(x_n; \theta))} = \arg \min_{a, \theta} \frac{1}{Z_{i-1}} \sum_{n=1}^N e^{-t_n (F_{i-1}(x_n) + a\phi(x_n; \theta))}.$$

Como $w_n^{(i)}$ só depende da amostra de índice n , ele pode ser considerado o **peso associado à amostra n** . Além disso, como na iteração $i = 1$ não temos um modelo anterior $F_{i-1}(x_n)$, inicializamos os valores de $w_n^{(i)}$ em $1/N$, pois assim, garantimos que, inicialmente, todas as amostras tenham o mesmo peso e que a eq. (8.5) é válida.

Realizamos a otimização da eq. (8.3), primeiramente, com relação à θ , mantendo a constante:

$$\begin{aligned} \theta_i &= \arg \min_{\theta} \sum_{n=1}^N w_n^{(i)} e^{-t_n a \phi(x_n; \theta)} \\ &= \arg \min_{\theta} P_i, \end{aligned} \quad (8.6)$$

em que

$$P_i := \sum_{n=1}^N w_n^{(i)} \mathbb{1}_{\{\leq 0\}}(-t_n \phi(x_n; \theta)). \quad (8.7)$$

Como a é constante e $t_n \phi(x_n; \theta) \in \{-1, 1\}$, então:

- quando $t_n \phi(x_n; \theta) = 1$, temos
 - $e^{-t_n a \phi(x_n; \theta)} = e^{-a}$;
 - $\mathbb{1}_{\{\leq 0\}}(-t_n \phi(x_n; \theta)) = 0$;
- quando $t_n \phi(x_n; \theta) = -1$, temos
 - $e^{-t_n a \phi(x_n; \theta)} = e^a$;
 - $\mathbb{1}_{\{\leq 0\}}(-t_n \phi(x_n; \theta)) = 1$.

Logo, fica claro que, de fato, o valor de θ que minimiza a eq. (8.6) também minimiza a eq. (8.7). Com isso, vemos que, apenas os pontos classificados incorretamente contribuem para o cálculo de θ_i . Portanto,

$$P_i = \sum_{t_n \phi(x_n; \theta) = -1} w_n^{(i)}. \quad (8.8)$$

Reparar que $P_i \in [0, 1]$, pois, os pesos $w_n^{(i)}$ são normalizados.

Passamos para a otimização de a :

$$a_i = \arg \min_a \sum_{n=1}^N w_n^{(i)} e^{-t_n a \phi(x_n; \theta)} = \arg \min_a \sum_{t_n \phi(x_n; \theta) = -1} w_n^{(i)} e^a + \sum_{t_n \phi(x_n; \theta) = 1} w_n^{(i)} e^{-a}$$

A partir das equações (8.5) e (8.8), podemos escrever

$$a_i = \arg \min_a P_i e^a + (1 - P_i) e^{-a}.$$

Ao derivarmos $P_i e^a + (1 - P_i) e^{-a}$ com relação a a e igualar a zero, obtemos

$$a_i = \frac{1}{2} \ln \left(\frac{1 - P_i}{P_i} \right) \quad (8.9)$$

Notar que, quando $P_i < 0,5$, $a_i > 0$, o que é esperado na prática.

Conhecendo os valores de θ_i e a_i , podemos determinar $F_i(x)$ e passar para a próxima iteração.

Com isso, podemos fazer algumas observações importantes:

1. Percebemos que os pesos $w_n^{(i)}$ assumem valores maiores nas iterações seguintes para as amostras classificadas incorretamente e valores menores caso contrário, pois, $w_n^{(i+1)} = \frac{e^{-t_n F_i(x_n)}}{Z_i} = \frac{e^{-t_n [F_{i-1}(x_n) + a_i \phi(x_n; \theta_i)]}}{Z_i} = \frac{z_{i-1} w_n^{(i)} e^{-a_i \phi(x_n; \theta_i) t_n}}{Z_i} \equiv w_n^{(i)} e^{-a_i \phi(x_n; \theta_i) t_n}$, já que as constantes z_i e z_{i-1} desaparecem dentro do argmin da eq. (8.3).
2. Vemos também que o termo P_i representa a taxa de erro ponderada do modelo base da iteração i (o que é mais facilmente visto pela eq. (8.7)). Quando P_i aumenta, a_i diminui. Logo, modelos que têm menor taxa de erro ponderada recebem maior peso a_i durante a predição conjunta dos diferentes *base learners*.

A Figura 8-3 resume o processo de treinamento e predição do algoritmo AdaBoost.

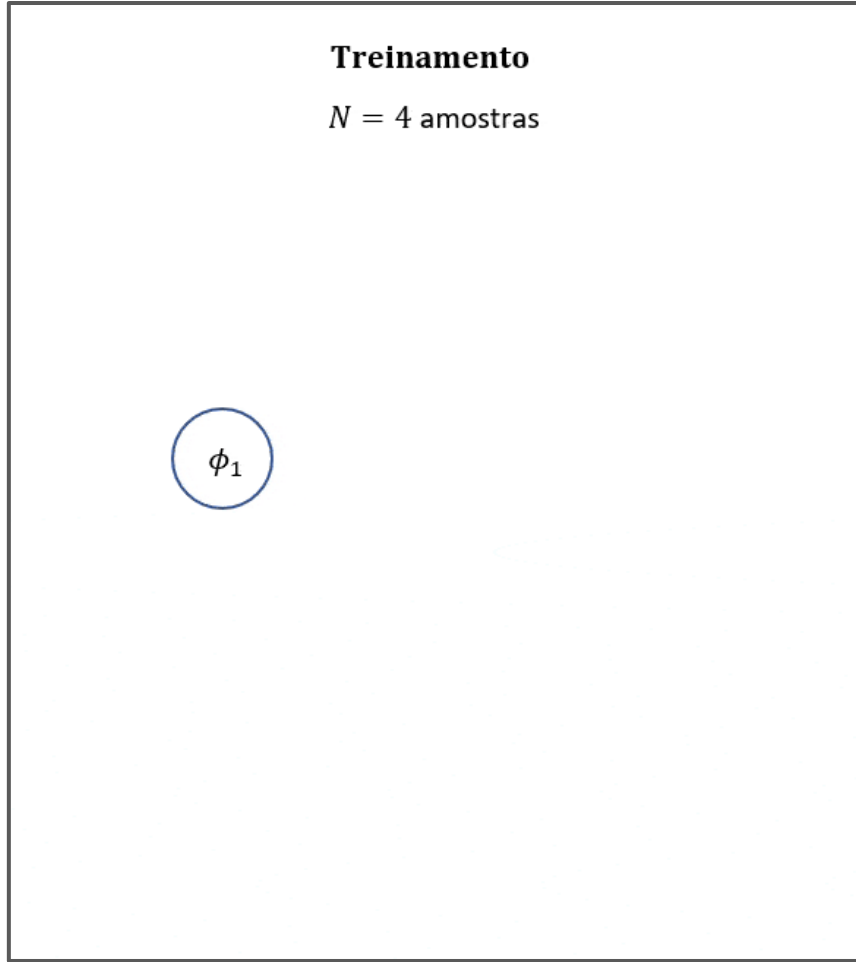


Figura 8-3: Diagrama ilustrativo do AdaBoost. Cada círculo representa um weak learner.

8.1.2 Discussão sobre a *loss function*

A escolha da *loss* exponencial para o modelo *AdaBoost*, descrito na Subseção 8.1.1, e a forma como a função de classificação $f(\mathbf{x})$ da eq. (8.1) é definida não são arbitrárias. Para entender isso, olhemos para o valor esperado da *loss* exponencial com relação à variável aleatória T que representa a classe de um determinado ponto \mathbf{x} :

$$\mathbb{E}_T[e^{-TF(\mathbf{x})}] = \mathbb{P}(T = 1|\mathbf{x})e^{-F(\mathbf{x})} + \mathbb{P}(T = -1|\mathbf{x})e^{F(\mathbf{x})}. \quad (8.10)$$

A ideia transmitida por esse cálculo é que, se considerarmos os diferentes pontos do conjunto de dados, com uma distribuição T qualquer de classes, é esperado que o valor da função *loss* assumo o valor $\mathbb{E}_T[e^{-TF(\mathbf{x})}]$. Com isso, se quisermos saber qual deve a forma da função $F(\mathbf{x})$ que minimiza esse valor devemos calcular:

$$\begin{aligned} F_*(\mathbf{x}) &= \arg \min_{F(\mathbf{x})} \mathbb{E}[e^{-TF(\mathbf{x})}] = \frac{1}{2} \ln \frac{\mathbb{P}(T = 1|\mathbf{x})}{\mathbb{P}(T = -1|\mathbf{x})} \\ F_*(\mathbf{x}) &= \frac{1}{2} \text{logit}(\mathbb{P}(T = 1|\mathbf{x})) \end{aligned} \quad (8.11)$$

Aqui, foi omitido o parâmetro θ que caracteriza a função $F(\mathbf{x})$, apenas por simplicidade. Com isso, vemos que, ao utilizar a *loss* exponencial, o valor esperado dessa *loss* é minimizado quando a função $F(\mathbf{x})$ é proporcional a uma função logit. Isso quer dizer que, quando

$\mathbb{P}(T = 1|\mathbf{x}) > \mathbb{P}(T = -1|\mathbf{x})$, $F(\mathbf{x}) > 0$ e, caso contrário, $F(\mathbf{x}) < 0$ (Figura 8-4). Portanto, ao adotar a função de classificação $f(\mathbf{x}) = \text{sgn}(F(\mathbf{x}))$, estamos classificando o ponto \mathbf{x} com base na classe que tem maior probabilidade a posteriori, o que está de acordo com a teoria da decisão.

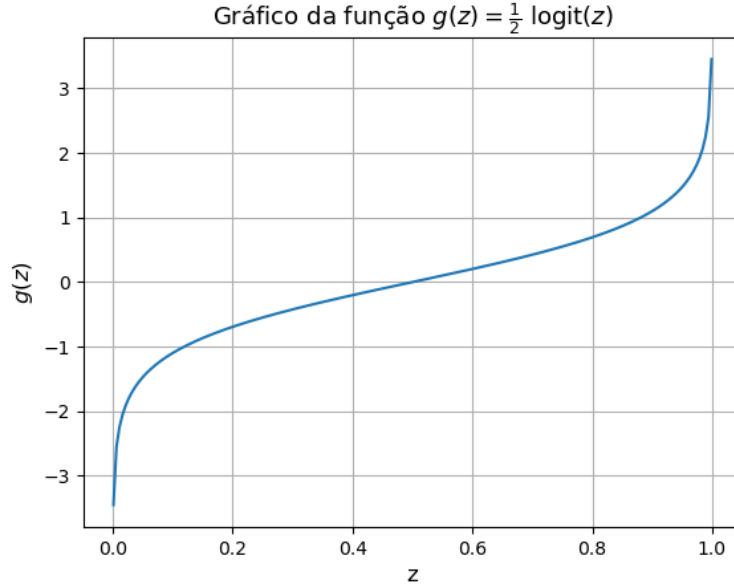


Figura 8-4: Curva da função logit.

Como isso se relaciona com o desenvolvimento da Subseção 8.1.1? Repare que, a função custo minimizada na eq. (8.2), é justamente a média amostral \bar{L} da *loss* exponencial, que é um estimador natural para ela:

$$\arg \min_{a, \theta} \sum_{n=1}^N e^{-t_n F(\mathbf{x}_n; \theta)} = \arg \min_{a, \theta} \frac{1}{N} \sum_{n=1}^N e^{-t_n F(\mathbf{x}_n; \theta)} = \arg \min_{a, \theta} \bar{L}.$$

Logo, o desenvolvimento feito na Subseção 8.1.1 está de acordo com a teoria estatística da decisão, pois estamos garantindo que os valores de a e de θ encontrados são tais que $F(\mathbf{x}; \theta)$ é dado pela equação (8.11), o que garante que $f(\mathbf{x})$ será sempre igual à classe com maior probabilidade a posteriori.

No entanto, como visto anteriormente, a *loss* exponencial penaliza demais pontos errados, principalmente aqueles cuja margem $m_{\mathbf{x}} := |yF(\mathbf{x})|$ está muito distante da superfície de decisão, localizada em $x F(\mathbf{x}) = 0$. Por isso, *outliers* tendem a receber um peso muito maior quando comparados com outros pontos, o que impacta o processo de otimização. Assim, na presença de *outliers* a *loss* exponencial não é a mais adequada.

Por outro lado, nesses casos, podemos utilizar como alternativa a **log-loss**, definida como

$$L_{\log} := -[Y' \log(\hat{p}_{\theta}) + (1 - Y') \log(1 - \hat{p}_{\theta})], \quad (8.12)$$

em que $Y' = \frac{t+1}{2}$ e $\hat{p}_{\theta} = \mathbb{P}(T = 1|\mathbf{x}; \theta)$. Essa *loss* tem seu valor mínimo quando $\hat{p}_{\theta} = \mathbb{P}(T = 1|\mathbf{x})$, ou seja, quando a distribuição de probabilidade predita (aproximada) é igual à verdadeira. Lembrando que a probabilidade $\mathbb{P}(T = 1|\mathbf{x})$ é teórica e não estamos nos preocupando com sua distribuição (se é *point mass* ou não, por exemplo).

Se definirmos

$$\hat{p}_\theta = \frac{1}{1 + e^{-2F(\mathbf{x})}} = \sigma(2F(\mathbf{x})), \quad (8.13)$$

o valor de \hat{p}_θ que minimiza a *loss* da eq. (8.12) é aquele tal que $F(\mathbf{x}) = \frac{1}{2} \text{logit}(\mathbb{P}(T = 1|\mathbf{x}))$, pois, ao substituir essa expressão de $F(\mathbf{x})$ na eq. (8.13), encontramos $\hat{p}_\theta = \sigma(\text{logit}(\mathbb{P}(T = 1|\mathbf{x}))) = \mathbb{P}(T = 1|\mathbf{x})$. Com isso, percebemos que a escolha de \hat{p}_θ da eq. (8.13) é feita de forma que o valor de $F(\mathbf{x})$ que minimiza a eq. (8.12) seja o mesmo que minimiza a eq. (8.10), ou seja,

$$\arg \min_{F(\mathbf{x})} L_{\log} = \arg \min_{F(\mathbf{x})} \mathbb{E}[e^{-TF(\mathbf{x})}].$$

Com isso, mesmo utilizando a *log-loss* ainda estaremos seguindo a teoria estatística da decisão ao adotarmos $f(\mathbf{x})$ como regra de decisão.

Usando a eq. (8.13), podemos reescrever a *loss* da eq. (8.12) em sua forma mais usual:

$$L_{\log} = \mathcal{L}(t, F(\mathbf{x})) = \ln(1 + e^{-2tF(\mathbf{x})}).$$

Como pode ser visto na Figura 8-2, essa *loss* apresenta valores mais balanceados para diferentes valores de $tF(\mathbf{x})$, diminuindo a atribuição de pesos excessivamente grandes para *outliers*. Um detalhe importante é que, ao se utilizar essa função, a otimização fica muito complicada e, por isso, é recomendado utilizar métodos como gradiente descendente ou baseados no método de Newton [1]. Também é importante notar que, ao utilizar a *log-loss*, não podemos mais chamar o algoritmo de AdaBoost, já que o que o caracteriza é a utilização da *loss* exponencial.

8.2 BOOSTING TREES

As *boosting trees* são modelos de *boosting* que utilizam árvores como *weak learner*. Assim, diferentemente de antes, agora temos definido o modelo do *weak learner*, o que nos permite definir matematicamente o termo $\phi(\mathbf{x}; \theta)$. Como estamos falando de árvores (*tree*), utilizaremos no lugar do símbolo ϕ o símbolo T . Com isso, podemos escrever um *weak learner* da *boosting tree* da seguinte forma:

$$T(\mathbf{x}; \Theta) = \sum_{j=1}^J \gamma_j \mathbb{1}_{\{\mathbf{x} \in R_j\}}(\mathbf{x}),$$

em que: J é o número de folhas na árvore; R_j é a região do espaço de *features* associada à folha j ; γ_j é o valor de predição associado à região R_j ; Θ é a matriz de parâmetros $(\gamma_j, R_j)_{j=1}^J$.

Com isso, podemos escrever a saída de um modelo *boosted tree* como

$$F(\mathbf{x}) = \sum_{k=1}^K T(\mathbf{x}; \Theta_k),$$

em que K é o número de árvores do modelo e Θ_k é a matriz de parâmetros da árvore de índice k . Reparar que é dada a mesma importância para todos os *weak learners*, ou seja, temos $a_k = 1, \forall k$.

Para calcular os parâmetros desse modelo, fazemos

$$\Theta_k = \arg \min_{\Theta} \sum_{n=1}^N L(t_n, F_{k-1}(x_n) + T(x_n; \Theta)). \quad (8.14)$$

Essa otimização pode ser dividida em duas etapas: calcular as regiões R_j ; calcular os valores \hat{t}_j .

Para uma única árvore de decisão, dada a região R_j , encontrar γ_j não é complexo. No caso da regressão, basta calcular a média dos *targets* dos pontos contidos nessa região e, no caso da classificação, basta associar γ_j à classe majoritária. No entanto, o cálculo de R_j é um pouco mais complexo e, normalmente, envolve aplicar um algoritmo guloso, como visto na seção sobre árvores de decisão.

Quando saímos do caso em que temos uma única árvore e vamos para o caso do *boosting tree*, composto por K árvores, o processo de otimização torna-se mais complexo, principalmente no que diz respeito a encontrar as regiões R_j [7, p. 357]. Para algumas escolhas de *loss* (erro quadrático, por exemplo), esse problema de otimização até pode ser simplificado, mas, para outras (Huber *loss*, por exemplo), que tornam o algoritmo mais robusto, a otimização torna-se muito custosa computacionalmente.

8.3 GRADIENT BOOSTING

Gradient boosting [18], também conhecido como GBM (*Gradient Boosting Model*) é uma técnica utilizada para aprimorar modelos de *boosting* tendo árvores de decisão como *weak learners*. Os modelos de *boosting* baseados em árvores tendem a apresentar melhor desempenho [citação], porém, o treinamento deles da maneira tradicional torna-se muito complexo. A partir disso, surge a técnica de *gradiente boosting*.

Essa técnica pode ser resumida pelo seguinte algoritmo:

Algoritmo Gradient Boosting

1. Inicializar $F_0(x) = \arg \min_{\gamma} \sum_{n=1}^N L(\gamma, x_n, t_n)$
2. Para $k = 1$ a K :
 - a. Para $n = 1$ a N :
Calcular $r_{nk} = - \left[\frac{\partial L(F, x_n, t_n)}{\partial F} \right]_{F=F_{k-1}}$
 - b. Treinar uma árvore $T(x; \Theta_k)$ usando como conjunto de treinamento os pares $(x_n, r_{nk})_{n=1}^N$. Matematicamente, isso equivale a resolver a eq. (8.14).
 - c. Atualizar $F_k(x) = F_{k-1} + T(x, \Theta_k)$
3. Retornar $F(x) := F_M(x) = F_0 + \sum_{k=1}^K T(x, \Theta_k)$.

De maneira resumida, o algoritmo *gradiente boosting* consiste em, a partir de um valor inicial, dado por $F_0(x) = \arg \min_{\gamma} \sum_{i=1}^N L(t_i, \gamma)$, encontrar as K árvores (*weak learner*) que são treinadas para serem aproximadoras dos valores da *loss* a cada iteração. Com isso, temos um algoritmo que, a cada iteração, avalia a *loss* do modelo e o corrige incrementalmente com o valor do gradiente dessa *loss*. No fim, teremos o valor inicial F_0 corrigido por diversas parcelas δ_k de gradiente da *loss*, ou seja, algo como $F_0 + \delta_1 + \delta_2 + \dots + \delta_K$.

8.3.1 Exemplo com a *loss erro quadrático*

Considerando o caso em que a *loss* é o erro quadrático, teríamos $F_0(\mathbf{x}) = \arg \min_{\gamma} \sum_{i=1}^N (t_n - \gamma)^2 = \frac{1}{N} \sum_{i=1}^N t_n = \bar{t}$, ou seja, a média amostral t_n (lembrando que essa minimização é facilmente resolvida ao fazer $\frac{\partial}{\partial \gamma} [\sum_{i=1}^N (t_n - \gamma)^2] = 0$). Como $F_0(\mathbf{x}) = \bar{t}$ é constante, podemos interpretá-lo como sendo uma árvore de apenas um nó, que dá como saída o escalar \bar{t} independentemente da entrada.

No passo 2, na primeira iteração, em que $k = 1$, calculamos os valores de gradiente da *loss*, avaliada em $F = F_0$, ou seja, $r_{n1} = t_n - F|_{F=F_0} = t_n - F_0 = t_n - \bar{t}$. Isso é equivalente ao erro de predição do modelo F_0 .

Depois de calcular r_{n1} para todas as N instâncias, treinamos uma árvore $T(\mathbf{x}; \Theta_1)$, utilizando como entradas \mathbf{x}_n e como *target* r_{n1} . Por fim, adicionamos essa árvore ao modelo de *boosting*, fazendo $F_1(\mathbf{x}) = \bar{t} + T(\mathbf{x}; \Theta_1)$.

Repetimos esses procedimentos para cada k . No final, teremos um modelo de *boosting* dado por $F(\mathbf{x}) = \bar{t} + \sum_{k=1}^K T(\mathbf{x}, \Theta_k)$. Como podemos ver, o modelo resultante corresponde à média dos *targets* mais árvores que representam pequenas correções que, para cada \mathbf{x}_n , levam \bar{t} o mais próximo possível de t_n , por meio de pequenas correções δ_k , ou seja, $t_n \approx \bar{t} + \delta_1 + \delta_2 + \dots + \delta_K$. A cada δ_k somado, o valor de t_n se aproxima mais de \bar{t} .

8.3.2 Derivação Matemática

O objetivo do GBM é encontrar uma função $F^* \in \mathbb{H}$, representando o modelo, que minimize o custo, representado pelo funcional $C: \text{lin}(\mathcal{F}) \rightarrow \mathbb{R}$, $f \mapsto C(f)$, em que \mathcal{F} é uma classe de funções que caracteriza a hipótese base (representando algum modelo como uma árvore de decisão, por exemplo) e $\text{lin}(\mathcal{F})$ é o conjunto de todas as combinações lineares das funções de \mathcal{F} . Para chegar a esse objetivo, podemos adotar um procedimento análogo ao do *steepest descent*, em que um determinado parâmetro é atualizado iterativamente da seguinte maneira:

$$\theta \leftarrow \theta + \mu v,$$

em que $v = -\nabla C(\theta)$. Nessa formulação, estamos incrementando o parâmetro θ a cada iteração (inicializado aleatoriamente), somando a ele um vetor μv na direção da maior descida de valor de $C(\theta)$, que é uma função de θ .

No caso do GBM, desejamos fazer algo similar, porém no lugar de θ temos uma função F e no lugar da função de custo $C(\theta)$ temos o funcional de custo $C(F)$. Então, desejamos fazer a cada iteração a seguinte atualização

$$F \leftarrow F + \mu f,$$

com $f \in \mathcal{F}$. Nesse caso, como f está restrita a \mathcal{F} , em geral, não é possível encontrar $f = -\nabla C(F)$. Por isso, procuramos f mais similar possível a $-\nabla C(F)$. Podemos usar como métrica de similaridade o produto interno, ou seja, procuramos f tal que $\langle -\nabla C(F), f \rangle$ seja o maior possível. Além de o produto interno ser uma escolha comum para medir similaridade entre funções, podemos também ver essa escolha pelo ponto de vista de derivada direcional do funcional $C(F)$ na “direção” da função f [19], dada por

$$\langle \nabla C(F), f \rangle = \lim_{\epsilon \rightarrow 0} \frac{C(F + \epsilon f) - C(F)}{\epsilon}.$$

Reparar que essa formulação da derivada direcional para um funcional é análoga a da derivada direcional de uma função em que temos $D_v f(x) = \lim_{\epsilon \rightarrow 0} \frac{f(x+\epsilon v) - f(x)}{\epsilon \|v\|} = \frac{1}{\|v\|} \langle \nabla f, v \rangle$.

Considerando uma variação $\Delta\epsilon$ suficiente pequena, podemos escrever

$$C(F + \Delta\epsilon f) \approx C(F) + \Delta\epsilon \langle \nabla C(F), f \rangle.$$

Com isso, vemos que a maior redução no custo $C(F)$ acontece ao minimizarmos $\langle \nabla C(F), f \rangle$, que é equivalente a maximizar $-\langle \nabla C(F), f \rangle = \langle -\nabla C(F), f \rangle$.

Esse desenvolvimento dá origem ao algoritmo AnyBoost [20], que é uma generalização do GBM para qualquer modelo de *weak learner*. Se voltarmos no algoritmo descrito no início desta seção, veremos que ele segue a lógica descrita no desenvolvimento matemático acima:

- no passo 1, definimos uma função arbitrária F_0 ;
- no passo 2, maximizamos $\langle -\nabla C(F), f \rangle$ ao encontrar uma função $f := T(\mathbf{x}; \Theta_k)$ mais similar possível à função custo

$$C(D) = \frac{1}{|D|} \sum_{(x_n, t_n) \in D} L(F, x_n, t_n), \quad (8.15)$$

utilizando $D = \{(x_n, t_n)\}_{n=1}^N$ como conjunto de treinamento para treinar a árvore $T(\mathbf{x}; \Theta_k)$;

- passo 3, atualizamos o modelo $F_k \leftarrow F_{k-1} + T(\mathbf{x}; \Theta_k)$, de maneira a reduzir o custo.

É importante reparar que, no passo 2, apesar de não parecer, a expressão de r_{nk} é o gradiente do custo da eq. (8.15). O que gera essa confusão é que, para realizar o treinamento da árvore, é mais conveniente utilizar uma amostra por vez na expressão da função custo, ou seja, a cada iteração temos $D = \{x_n, t_n\}$. Isso permite que tenhamos N pares de instâncias de treinamento $(x_n, C(D_n))$ para treinar a árvore. Portanto, no fim das contas, considerando esse cenário, a cada iteração, temos uma função custo $C(D) = L(F, x_n, t_n)$, já que $|D| = 1$.

8.4 XGBOOST

[Em construção]

Parecido com o *gradient boosting*, mas apresenta algumas modificações incluindo otimizações de algumas etapas e inclusão de novas etapas.

Algumas das principais diferenças relacionadas a otimização são:

- Permite paralelização no processo de construção das árvores.
- Critério de poda das árvores é profundidade máxima, em vez do valor da *loss*.
- Uso de *cache awareness* para fazer uso mais eficiente do *hardware*.

Diferenças nas etapas do algoritmo:

- Regularização LASSO e Ridge.
- *Sparsity awareness*, que permite lidar de maneira mais eficiente com dados esparsos e *missing values*.
- Mudança na forma como o *split point* das árvores é encontrado (*quantile sketch algorithm*).

- *Built-in cross-validation.*

Capítulo 9: REDE NEURAL

Nas seções sobre regressão linear e regressão logística, vimos que os modelos tinham a forma

$$y(\mathbf{x}, \boldsymbol{\theta}) = f(\boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x})) = f\left(\sum_{j=1}^M \theta_j \phi_j(\mathbf{x})\right),$$

com $\boldsymbol{\theta} = [\theta_M \ \theta_{M-1} \ \dots \ \theta_0]^T$ e $\mathbf{x} = [x_M \ x_{M-1} \ \dots \ x_0]^T$. Na regressão linear, tínhamos $f(x) = x$ e, na regressão logística, tínhamos uma função não-linear (mais especificamente uma sigmoide).

Na rede neural, teremos um modelo muito parecido, mas as funções base ϕ_j , agora, também dependerão de parâmetros θ ajustáveis e terão a forma abaixo:

$$\begin{aligned} y_k(\mathbf{x}, \boldsymbol{\theta}) &= f\left(\sum_{j=1}^M \theta_{kj}^{(L)} \phi_j(\mathbf{x}, \boldsymbol{\theta}_j)\right) = f\left(\sum_{j=1}^M \theta_{kj}^{(L)} h\left(\sum_{i=1}^M \theta_{ji}^{(L-1)} \phi_i(\mathbf{x}, \boldsymbol{\theta}_i)\right)\right) \\ &= f\left(\sum_{j=1}^M \theta_{kj}^{(L)} h\left(\sum_{i=1}^M \theta_{ji}^{(L-1)} g(\dots)\right)\right), \end{aligned}$$

em que f , h e g são funções não lineares. Como a rede neural pode ter K saídas, usamos o índice k para indicar a qual saída estamos nos referindo. Além disso, percebemos que a função base ϕ_j é uma composição de várias outras funções que dependem dos parâmetros ajustáveis. Utilizamos um sobrescrito nos parâmetros para indicar a qual das L **camadas** da rede neural eles estão vinculados.

Assim, vemos que a rede neural é uma versão mais complexa de uma regressão logística (no caso da classificação), em que a função base depende dos parâmetros ajustáveis e de uma estrutura recursiva de funções não lineares. Isso nos permite criar fronteiras de decisão não lineares (no espaço das entradas \mathbf{x}) muito mais complexas do que as da regressão logística, graças à existência de uma função base ϕ_j bastante adaptável.

9.1 ESTRUTURA DA REDE NEURAL

O elemento básico de uma rede neural é chamado de **neurônio**, como pode ser visto na Figura 9-1.

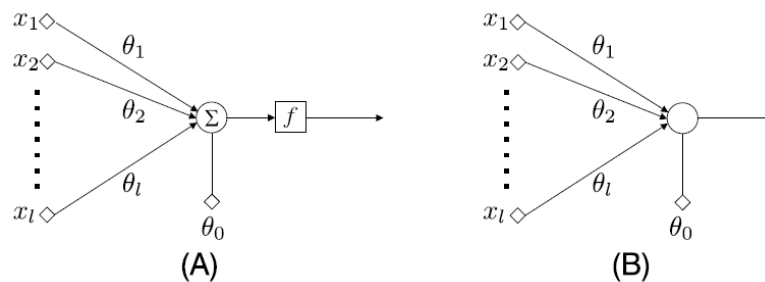


Figura 9-1: Ilustração do neurônio de uma rede neural.

Nessa imagem, podemos ver os elementos já mencionados anteriormente, nomeadamente as entradas x_i , os pesos θ_i e a função não linear f , que recebe o nome especial de **função de ativação**. Além disso, o termo θ_0 , que não está associado a nenhuma entrada recebe o nome de **bias** (ou *threshold*). A parte B dessa figura mostra que, usualmente, agrupamos o bloco de combinação linear com o da função de ativação para formar o que é chamado de **nó**. Muitas vezes, o termo nó será utilizado para se referir ao neurônio inteiro.

Juntando vários neurônios, podemos formar o que é chamado de **feed-forward network** ou **fully connected network** (FCN), como é mostrado na Figura 9-2. O nome *feed-forward network* se dá pelo fato de a informação ser unidirecional, indo das entradas x_i até camada de saída. Já o nome *fully connected network* enfatiza o fato de que cada neurônio desse tipo de rede está conectado a todos os neurônios da camada anterior.

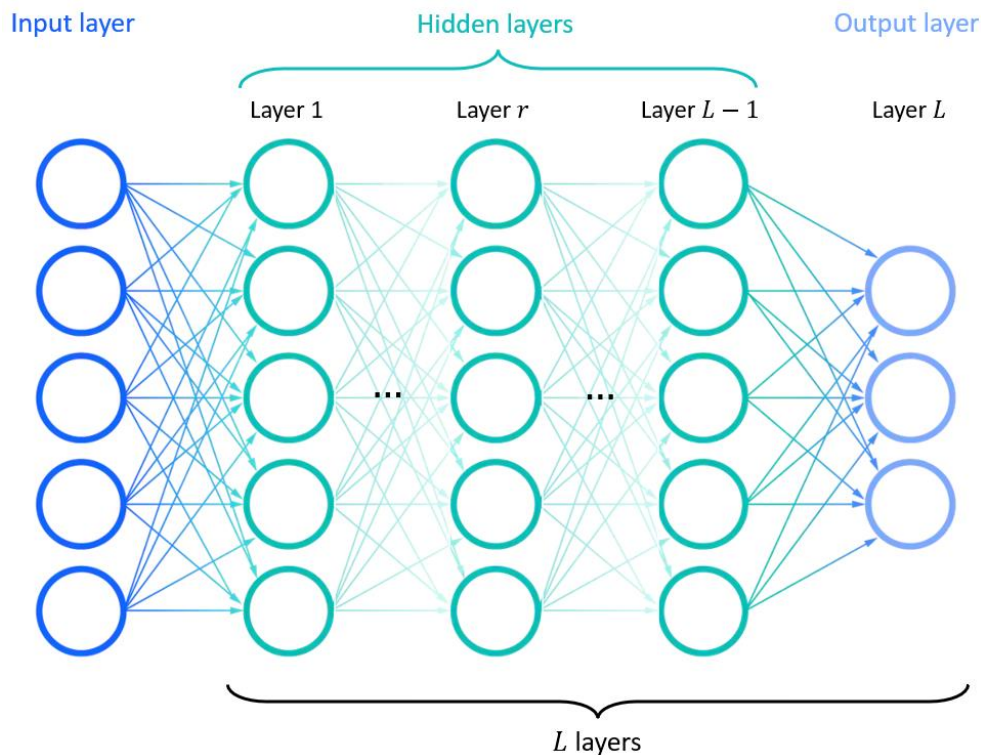


Figura 9-2: Rede neural totalmente conectada com L camadas.

Atentando-se para a questão de nomenclatura, damos os nomes de **neurônio de saída** e **neurônios escondidos** aos neurônios pertencentes às camadas de saída e camadas escondidas, respectivamente. Os nós pertencentes à camada de entrada não são neurônios, ou seja, não realizam nenhum tipo de processamento. Por causa disso, dizemos que a rede da Figura 9-2 é uma rede de L camadas, excluindo a camada de entrada da contagem. Redes com até 3 camadas (até duas camadas escondidas) são chamadas de **rasas** e redes com mais camadas são chamadas **profundas**.

Muitas pessoas também chamam essa rede de **multilayer perceptron** (MLP), por causa da origem desse modelo, que era baseado em um modelo de neurônio chamado de perceptron.

Em cada neurônio da rede neural, estamos realizando a seguinte operação:

$$y_j^r = f(z_j^r) = f(\theta_j^{rT} \mathbf{y}^{r-1}).$$

O sobrescrito r indica a qual das L camadas o neurônio pertence.

O subscrito j identifica o número do neurônio nessa camada, considerando um total de k_r neurônios.

Assim, a expressão diz que a saída y_j^r do neurônio j da camada r é igual ao produto do vetor \mathbf{y}^{r-1} com as saídas dos neurônios da camada anterior pelo vetor de parâmetros $\boldsymbol{\theta}_j^r$, vinculado ao neurônio em questão, passado por uma função não linear $f(\cdot)$.

O vetor das saídas de todos os neurônios de uma camada r é dado por

$$\mathbf{y}^r = \begin{bmatrix} 1 \\ f(\mathbf{z}^r) \end{bmatrix} = \begin{bmatrix} 1 \\ f(\boldsymbol{\Theta}^r \mathbf{y}^{r-1}) \end{bmatrix},$$

em que $\boldsymbol{\Theta}^r = [\boldsymbol{\theta}_1^r \quad \boldsymbol{\theta}_2^r \quad \dots \quad \boldsymbol{\theta}_{k_r}^r]^T$ e k_r é o número de neurônios na camada r .

Lembrando que o elemento 1 em \mathbf{y}^r leva em consideração o *bias* θ_{j0}^r , contido no vetor $\boldsymbol{\theta}_j^r$.

A Figura 9-3 mostra um diagrama que resume essas equações para uma camada r .

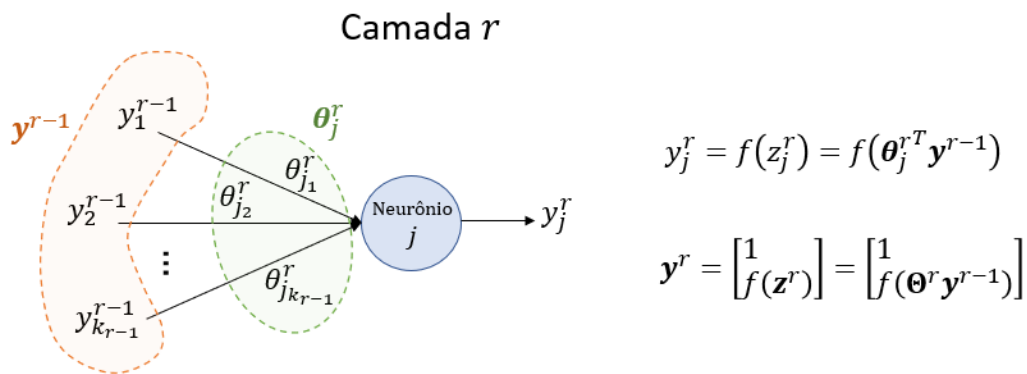


Figura 9-3: Equações da FCN.

Para a função não linear f , é comum utilizarmos funções como a sigmoide ou a tangente hiperbólica, por alguns motivos específicos, mas principalmente por elas serem diferenciáveis. Seus gráficos são mostrados na Figura 9-4.

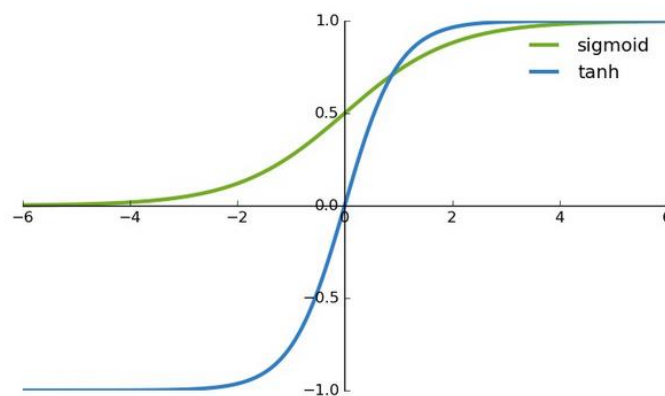


Figura 9-4: Funções sigmoide e tangente hiperbólica.

9.2 ALGORITMO DE BACKPROPAGATION

É o algoritmo utilizado pelas redes neurais para otimizar os parâmetros θ . Assim como em outros modelos de *machine learning*, nosso objetivo é otimizar uma função objetivo/custo com o seguinte formato:

$$J(\theta) = \sum_{n=1}^N \mathcal{L}(y_n, f_{\theta}(x_n)),$$

em que N é o número total de amostras de treinamento (x_n, y_n) , $n \in [1, N]$ e $\mathcal{L}(\cdot, \cdot)$ é a função *loss*.

Uma das grandes dificuldades de otimizar uma função como essa está na estrutura multicamadas da rede neural, em que cada neurônio possui um vetor de parâmetros θ único associado a cada um deles e as saídas de cada neurônio depende das saídas dos neurônios mais **rasos** (pertencentes a camadas de índice menor). Assim, temos uma estrutura em que a saída de um neurônio é uma composição de várias funções não lineares.

Outra dificuldade é que, devido à natureza altamente não linear da rede neural, a função custo tende a apresentar diversos mínimos locais, o que dificulta encontrarmos a solução ótima, principalmente quando aplicamos técnicas de otimização como o gradiente descendente. Muitas vezes, o mínimo local é tão fundo que se aproxima do valor de $J(\theta)$ ótimo. Nesses casos, podemos adotar esse mínimo local como solução, se o resultado final for satisfatório.

Feitas essas considerações, entraremos nos detalhes do algoritmo de *backpropagation*. Primeiramente, precisamos escolher uma função de perda. Nesse momento, adotaremos a função de erro quadrático para assumir esse papel. Assim, a função custo fica da seguinte forma:

$$J(\theta) = \sum_{n=1}^N J_n(\theta) \tag{9.1}$$

com

$$J_n(\theta) = \frac{1}{2} \|\hat{y}_n - y_n\|^2 = \frac{1}{2} \sum_{k=1}^{k_L} (\hat{y}_{n_k} - y_{n_k})^2.$$

Com isso, estamos avaliando o erro quadrático entre o valor estimado da saída \hat{y}_{n_k} no neurônio k para a amostra de índice n e o valor real y_{n_k} que deveria estar na saída, sendo o índice k o indicador da posição no vetor y_n .

O próximo passo é escolher um algoritmo de otimização para nos auxiliar a encontrar os melhores valores de θ . Normalmente, são utilizados algoritmos baseados no cálculo do gradiente, devido à sua simplicidade. O método do gradiente descendente é um exemplo e será empregado na derivação do algoritmo do *backpropagation* por simplicidade, apesar de existirem variações desse método que são até mais utilizadas na prática (Adagrad, Adam...) [5, pp. 924-934]. De toda forma, no método do gradiente descendente básico, o valor de θ é atualizado a cada iteração da seguinte maneira:

$$\theta_j^r \leftarrow \theta_j^r - \mu \frac{\partial J(\theta)}{\partial \theta_j^r}. \quad (9.2)$$

Assim, vemos que é necessário determinar o termo da derivada. Antes de desenvolver a expressão dessa derivada, lembremos que

$$\hat{y}_{n_k} = f(z_{n_k}^L) = f(\theta_k^L y_n^L).$$

Observação: não confundir $y_{n_j}^r$, com sobrescrito, que é a saída do neurônio j da camada r , com y_{n_j} , sem sobrescrito, que é o valor do *target* na posição j correspondente à entrada x_n .

Como $J(\theta)$ é uma soma de $J_n(\theta)$, basta encontramos uma expressão para $\frac{\partial J(\theta)}{\partial \theta_j^r}$. É interessante notar também que, apesar de estarmos explicitando J_n como dependente de θ , também poderíamos escrever $J_n = J_n(z_{n_1}^r(\theta_1^r), \dots, z_{n_{k_L}}^r(\theta_{n_{k_L}}^r))$, já que J_n depende de $z_{n_j}^r$, que, por sua vez, depende de θ_j^r .

Com isso, usando a regra da cadeia, podemos escrever

$$\frac{\partial J_n(\theta)}{\partial \theta_j^r} = \frac{\partial J_n}{\partial z_{n_j}^r} \frac{\partial z_{n_j}^r}{\partial \theta_j^r} = \frac{\partial J_n}{\partial z_{n_j}^r} \frac{\partial (\theta_j^{rT} y_n^{r-1})}{\partial \theta_j^r} = \frac{\partial J_n(\theta)}{\partial z_{n_j}^r} y_n^{r-1}.$$

Devido ao fato de o termo $\frac{\partial J_n(\theta)}{\partial z_j^r}$ aparecer com frequência no desenvolvimento do algoritmo de *backpropagation*, utilizamos a seguinte definição:

$$\delta_{nj}^r := \frac{\partial J_n(\theta)}{\partial z_j^r}$$

Portanto, nosso objetivo é encontrar

$$\frac{\partial J_n(\theta)}{\partial \theta_j^r} = \delta_{nj}^r y_n^{r-1} \quad (9.3)$$

Dessa forma, partindo da eq. (9.2) podemos encontrar a expressão que resume a atualização dos pesos da rede neural:

$$\begin{aligned} \theta_j^r &\leftarrow \theta_j^r - \mu \frac{\partial J(\theta)}{\partial \theta_j^r} \\ \theta_j^r &\leftarrow \theta_j^r - \mu \sum_{n=1}^N \frac{\partial J_n(\theta)}{\partial \theta_j^r} \\ \theta_j^r &\leftarrow \theta_j^r - \mu \sum_{n=1}^N \delta_{nj}^r y_n^{r-1} \end{aligned} \quad (9.4)$$

O valor y_n^{r-1} é observável, então não precisamos calcular esse termo. Basta determinarmos o valor de δ_{nj}^r para cada camada r e para cada neurônio j nessa camada. Para a camada $r = L$,

determinar δ_{nj}^r é simples, pois $J_n(\theta) = \frac{1}{2} \sum_{k=1}^{k_L} (f(z_j^r) - y_{n_k})^2$, o que faz com que a derivada com relação a z_j^r seja simples. Estamos calculando a derivada da função custo, que está sendo avaliada na última camada, com relação a z_j^r de um neurônio que também está na última camada. Logo, não temos composições de funções por causa da passagem de uma camada à outra.

Por outro lado, quando avaliamos δ_{nj}^r para neurônios que não estão na última camada, os cálculos se tornam um pouco mais complicados por causa da composição de funções. Por isso, abaixo, vamos derivar expressões gerais para δ_{nj}^r para os dois possíveis casos de r .

Detalhes dos cálculos do algoritmo de *backpropagation*

$r = L$:

No caso mais simples, em que o neurônio está na última camada, temos

$$\delta_{nj}^r = \frac{1}{2} \frac{\partial}{\partial z_j^r} \sum_{k=1}^{k_L} \left[\overbrace{f(z_{n_k}^r)}^{\hat{y}_{n_k}} - y_{n_k} \right]^2 = \frac{1}{2} \frac{\partial}{\partial z_j^r} [f(z_{n_j}^r) - y_{n_j}]^2 = (\hat{y}_{n_j} - y_{n_j}) f'(z_{n_j}^r)$$

$$\delta_{nj}^r = e_{nj} f'(z_{n_j}^r),$$

com $e_{nj} = \hat{y}_{n_j} - y_{n_j}$ sendo o erro da rede neural no neurônio j da camada de saída.

$r < L$:

Quando analisamos δ_{nj}^r para as camadas anteriores à camada de saída, verificamos que existe uma distância entre $J_n(\theta)$, que está sendo avaliado na saída, e a variável $z_{n_j}^r$, com relação a qual estamos derivando e que está associada à camada r . Logo, é natural que surjam regras da cadeia.

Da mesma maneira como foi feito anteriormente, podemos explicitar a dependência de J_n com relação às variáveis da rede neural como

$$J_n = J_n(z_{n_1}^r, \dots, z_{n_{k_L}}^r) = J_n(z_{n_1}^r(z_{n_1}^{r-1}, \dots, z_{n_{k_{r-1}}}^{r-1}), \dots, z_{n_{k_L}}^r(z_{n_1}^{r-1}, \dots, z_{n_{k_{r-1}}}^{r-1})).$$

Com isso, podemos escrever,

$$\begin{aligned} \delta_{nj}^{r-1} &= \frac{\partial J_n}{\partial z_{n_1}^r} \frac{\partial z_{n_1}^r}{\partial z_{n_j}^{r-1}} + \dots + \frac{\partial J_n}{\partial z_{n_{k_r}}^r} \frac{\partial z_{n_{k_r}}^r}{\partial z_{n_j}^{r-1}} = \sum_{k=1}^{k_r} \frac{\partial J_n}{\partial z_{n_k}^r} \frac{\partial z_{n_k}^r}{\partial z_{n_j}^{r-1}} = \sum_{k=1}^{k_r} \delta_{nk}^r \frac{\partial z_{n_k}^r}{\partial z_{n_j}^{r-1}} \\ &= \sum_{k=1}^{k_r} \delta_{nk}^r \frac{\partial (\theta_{nk}^T \mathbf{y}_n^{r-1})}{\partial z_{n_j}^{r-1}} = \sum_{k=1}^{k_r} \delta_{nk}^r \frac{\partial (\theta_k^{rT} f(\mathbf{z}_n^{r-1}))}{\partial z_{n_j}^{r-1}} = \left(\sum_{k=1}^{k_r} \delta_{nk}^r \theta_{kj}^r \right) f'(z_{n_j}^{r-1}) \end{aligned}$$

$$\delta_{nj}^{r-1} = e_n^r f'(z_{n_j}^{r-1}), \quad (9.5)$$

com $e_n^r = \sum_{k=1}^{k_r} \delta_{nk}^r \theta_{kj}^r$. Esse termo é só para criar uma uniformidade na notação.

Lembrando que poderíamos ter escrito uma expressão para δ_{nj}^r explicitamente, mas daria no mesmo. Só teríamos que substituir o termo δ_{nj}^r na equação acima por δ_{nj}^{r+1} . Da forma como está a equação acima é mais conveniente.

Depois de todo esse desenvolvimento, vemos que para determinar δ_{nj}^r , dependemos do valor de δ_{nj}^{r+1} , da camada seguinte (uma profundidade abaixo). Por isso, para encontrarmos todos os valores de δ_{nj}^r , precisamos fazer os cálculos de trás para frente (*backwards*), começando na camada de saída indo em direção à primeira camada.

O algoritmo de *backpropagation* consiste em

1. Inicializar os pesos θ aleatoriamente;
2. Iterar variando n de 1 a N :
 - a. Realizar o *forward propagation* (calcular as saídas da rede para a entrada x_n);
 - b. Realizar o *backpropagation*, determinando os valores de δ_{nj}^r para todos os valores de j , começando na camada de saída $r = L$ e indo em direção à primeira camada.
3. Atualizar os valores de θ_j^r a partir da eq. (9.4);
4. Se o critério de parada não for satisfeito, retornar ao passo 2.

O algoritmo acima pode ser executado várias vezes no mesmo conjunto de treinamento, repetindo os pares (x_n, y_n) . Toda vez que completamos um ciclo passando por todas as amostras de treinamento (x_n, y_n) , dizemos que uma *época* foi completada.

Alguns critérios de parada que podem ser adotados são: valor da função custo baixo o suficiente; os valores dos pesos de uma iteração para outra mudam muito pouco; número máximo de épocas foi atingido.

O algoritmo de *backpropagation* descrito acima utiliza uma abordagem do tipo *batch*, em que todas as N amostras de treinamento são utilizadas de uma vez para o cálculo da função custo (observar que o somatório vai de 1 até N). Outra abordagem possível é a *stochastic gradient descent* (SGD). Nesse caso, usamos um método de otimização similar ao gradiente descendente, mas, para o cálculo da função custo, consideramos uma amostra por vez. Assim, em vez de calcular os valores δ_{nj}^r para todos os valores de n para somente no final atualizar os pesos θ_j^r com a eq. (9.1), atualizamos os pesos em cada iteração de n , de forma que, em uma época, atualizamos N vezes os pesos.

Finalmente, existe uma abordagem intermediária, chamada de *minibatch*, em que não atualizamos os pesos a cada iteração de n , mas também não deixamos para atualizá-los depois de iterarmos ao longo das N amostras. Atualizamos-nos de K em K amostras de treinamento, com $K < N$, usando uma função custo da forma

$$J(\theta) = \sum_{n=1}^K J_n(\theta),$$

em que K é o tamanho do *minibatch*.

9.3 ESCOLHENDO A FUNÇÃO CUSTO E AS NÃO-LINEARIDADES

9.3.1 Observando a Camada de Saída

Ao observarmos os neurônios da saída, podemos ter uma noção dos efeitos da combinação de determinadas funções não-lineares com determinadas funções custo. Por simplicidade, vamos considerar uma rede com um único neurônio na camada de saída. Consideraremos os casos da função custo de erro quadrático e de entropia cruzada.

9.3.1.1 Função Custo de Erro Quadrático

Nesse caso, temos

$$J = \frac{1}{2} \sum_{n=1}^N (t_n - y_n^L)^2 = \frac{1}{2} \sum_{n=1}^N (t_n - f(\mathbf{z}_n))^2 = \frac{1}{2} \sum_{n=1}^N \left(t_n - f(\boldsymbol{\theta}^L \mathbf{y}_n^{L-1}) \right)^2,$$

em que N é o total de exemplos de treinamento, t_n é o valor do *target* e y_n^L é a saída do neurônio da última camada.

Se adotarmos a função sigmoide como não-linearidade, teremos

$$\frac{\partial J}{\partial \boldsymbol{\theta}^L} = \frac{\partial J}{\partial y_n^L} \frac{\partial y_n^L}{\partial \mathbf{z}_n} \frac{\partial \mathbf{z}_n}{\partial \boldsymbol{\theta}^L} = \left[-\frac{1}{2} \sum_{n=1}^N 2y_n^L - 2t_n \right] [\sigma'(\mathbf{z}_n)] [\mathbf{y}_n^{L-1}] = - \sum_{n=1}^N (y_n^L - t_n) \sigma'(\mathbf{z}_n) \mathbf{y}_n^{L-1}.$$

Reparar que isso é exatamente o que encontraríamos se aplicássemos a eq. (9.3) diretamente.

Com isso, vemos que o gradiente da função custo depende do erro $(y_n^L - t_n)$, o que é justo, pois, à medida que o erro diminui, o gradiente também diminui, assim como o tamanho do passo do gradiente descendente. Por outro lado, a dependência de $\sigma'(\mathbf{z}_n)$ é incômoda, pois, se \mathbf{z}_n for muito distante do valor zero, $\sigma'(\mathbf{z}_n)$ tende a zero, assim como o valor do gradiente.

Portanto, vemos que quando usamos a função sigmoide como não-linearidade do neurônio de saída, a função custo de erro quadrático não é muito adequada.

9.3.1.2 Função Custo de Entropia Cruzada

Nesse caso, temos

$$J = \sum_{n=1}^N H(t_n, y_n) = - \sum_{n=1}^N \sum_{k=1}^{k_L} t_{nk} \log y_{nk}^L = - \sum_{n=1}^N \sum_{k=1}^{k_L} t_{nk} \log \sigma(\mathbf{z}_{nk}).$$

Desenvolvendo a expressão para o gradiente da função custo, obtemos

$$\frac{\partial J}{\partial \boldsymbol{\theta}_j^L} = \sum_{n=1}^N y_{nj} (t_{nj} - 1) \mathbf{y}_n^{L-1}.$$

Verificamos que, diferentemente do caso da função de custo de erro quadrático, o gradiente não depende da derivada da função não-linear, eliminando o problema do gradiente que “desaparece” quando \mathbf{z}_n for muito distante do valor zero. Portanto, a função custo de entropia cruzada é mais adequada quando utilizamos a função sigmoide como não-linearidade.

Em geral, quando queremos garantir que as saídas da rede neural sejam probabilidades e somem 1, utilizamos a função softmax na saída. Pode ser mostrado que, utilizar a softmax em

vez da função sigmoide gera um gradiente que também é independente da derivada da função não-linear.

9.3.2 Observando as Camadas Escondidas

Para as camadas anteriores à camada de saída, a expressão que rege o algoritmo de *backpropagation* é a eq. (9.5):

$$\delta_{nj}^{r-1} = e_n^r f' \left(z_{nj}^{r-1} \right) = \sum_{k=1}^{k_r} \delta_{nk}^r \theta_{kj}^r f' \left(z_{nj}^{r-1} \right).$$

Como podemos perceber, temos uma lógica recursiva, em que o δ da camada anterior depende do δ da camada seguinte. Podemos deixar isso mais explícito ao substituímos o termo δ_{nk}^r na equação acima da seguinte forma:

$$\delta_{nj}^{r-1} = \sum_{k=1}^{k_r} \left[\sum_{k=1}^{k_{r+1}} \delta_{nk}^{r+1} \theta_{kj}^{r+1} f' \left(z_{nj}^r \right) \right] \theta_{kj}^r f' \left(z_{nj}^{r-1} \right). \quad (9.6)$$

Com isso, percebemos que existe um produto dos pesos θ e das derivadas da função não-linear. À medida que continuamos substituindo δ recursivamente, mais termos referentes às outras camadas vão sendo adicionados a esse produto.

A derivada da função não-linear pode assumir valores menores do que 1. Com isso, dependendo do caso, pode acontecer de o gradiente da função custo com respeito aos parâmetros das camadas mais superficiais assumam valores muito pequenos e sofram desvanecimento à medida que caminhamos para a parte mais rasa da rede. Esse fenômeno é conhecido como **desvanecimento do gradiente**. O maior problema associado a esse fenômeno é que ele torna o aprendizado mais lento.

Outro fenômeno associado a esse produto recursivo é o **gradiente explosivo**, que acontece quando o valor do gradiente assume valores mais elevados à medida que caminhamos na direção das camadas mais rasas. Isso afeta o aprendizado, pois faz com que a atualização dos parâmetros dê um salto muito elevado, podendo desviar os parâmetros do valor correto. Além disso, o gradiente explosivo pode levar a problema relacionados à overflow das variáveis que armazenam os parâmetros.

Por fim, em redes neurais com muitas camadas, os parâmetros de cada camada podem assumir valores de escalas muito diferentes, o que faz com que a velocidade do aprendizado de cada uma delas seja muito diferente, causando instabilidade.

Uma maneira de lidar com esse problema é utilizar função de ativação ReLU (Rectified Linear Unit) nos neurônios das camadas escondidas. Ela é dada por

$$f(z) = \max\{0, z\} \quad (9.7)$$

Seu gráfico é mostrado na Figura 9-5.

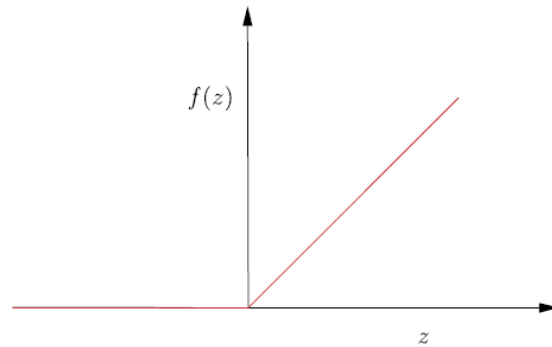


Figura 9-5: Gráfico da ReLU.

Ela ajuda a resolver os problemas do gradiente desvanecente e explosivo e, consequentemente, acelerando o treinamento porque sua derivada é sempre uma constante. Para $z > 0$, a derivada é um e os termos $f'(\cdot)$ na (9.6) desaparecem. Para $z = 0$, podemos escolher o valor 0 ou 1 para a derivada.

Como é possível perceber, para $z < 0$, a derivada se anula e o treinamento tende a ficar mais lento e a empacar. Por essa razão, ao inicializar os pesos da rede, no início do treinamento, é importante escolher pequenos valores positivos para os pesos.

9.4 REGULARIZAÇÃO

Como as redes neurais apresentam uma quantidade muito grande de parâmetros, elas ficam sujeitas ao problema de *overfitting*. Uma forma prática de lidar com esse problema é utilizar regularização. As principais técnicas de regularização que podem ser aplicadas a redes neurais são mostradas abaixo:

- **Decaimento de peso:** consiste em alterar a função custo adicionando um termo de regularização, como por exemplo

$$J'(\theta) = J(\theta) + \lambda \|\theta\|^2.$$

- **Early stopping:** no início do treinamento, há uma tendência de as curvas de erro de treinamento e de teste decaírem. A partir de determinado momento, a curva de erro de teste deixa de decair e começa a subir. A técnica de *early stopping* consiste em interromper o treinamento assim que a curva de erro de teste começa a subir. Esse é um dos métodos de regularização mais utilizados e pode ser combinado com outros métodos.
- **Dropout:** consiste em remover da rede neural, durante o treinamento, uma certa porção de nós. Quando um nó é removido, todas as suas conexões com os nós da camada anterior e da camada seguinte também são removidas. A cada iteração, cada nó tem probabilidade p de ser removido. Quando um nó é removido em uma iteração, os parâmetros associados a ele não são atualizados. Na iteração seguinte, todos os nós removidos retornam com os seus pesos conservados da iteração anterior a que eles foram removidos e é feito um novo sorteio para decidir quais são os nós que serão removidos novamente. Depois de a rede neural estar treinada e é utilizada para inferências, os pesos de cada neurônio são multiplicados por p . Esse é um dos métodos de regularização mais utilizados e pode ser combinado com outros métodos.

Capítulo 10: FEATURE SELECTION

Feature selection é uma técnica de redução de dimensionalidade que consiste em selecionar algumas *features* do conjunto total sem modificá-las ou criar novas *features*.

Um procedimento que tem um nome parecido e pode criar confusão é *feature extraction*, que também faz parte do conjunto de técnicas de redução de dimensionalidade. No entanto, nesse caso, realizamos a projeção de todas as *features* em um novo espaço com dimensão menor. Não selecionamos um subconjunto das *features*, como é o caso da *feature selection*.

As principais vantagens da *feature selection* (e da *feature extraction* também) são [21]:

- Aumento de eficiência computacional, pois são utilizadas menos variáveis durante o aprendizado do modelo.
- Modelo com maior capacidade de generalização, já que, mantendo-se a quantidade de dados constante, diminuir o número de *features* diminui o risco de *overfitting*.

O último ponto está ligado à maldição da dimensionalidade, que diz que, à medida que a dimensão do problema aumenta, os dados tornam-se mais esparsos. Uma forma de visualizar isso é imaginando o exemplo em que temos dois pontos e o posicionamos o mais distante possível dentro de um hipercubo de aresta a . Quando o número de dimensões é $d = 2$, a maior distância é $a\sqrt{2}$. Quando $d = 3$, a maior distância é $a\sqrt{3}$.

As técnicas de *feature selection* podem ser **supervisionadas**, que é o caso em que utilizamos os *targets* para selecionar o subconjunto de *features*, ou **não-supervisionados**, que é o caso em que não são utilizados os *targets*.

Além disso, as técnicas de *features selection*, sejam elas supervisionadas ou não-supervisionadas, podem ser divididas em:

- **Wrapper methods:** o critério utilizado para avaliar as *features* selecionadas é baseado no desempenho do modelo de *machine learning*. Assim, primeiramente, selecionamos um grupo de *features* e as utilizamos para treinar o modelo. Utilizamos os resultados do teste desse modelo para comparar essa seleção com as próximas seleções. A principal desvantagem desse método é a demora relacionada aos diversos treinamentos e testes do modelo que deverão ser executadas. Por isso, essa abordagem é raramente usada na prática.
- **Filter methods:** são métodos independentes de algoritmos de aprendizado. Eles avaliam a importância de cada *feature* com base apenas nas características dos dados. Alguns critérios que podem ser utilizados são correlação entre *features*, informação mútua.
- **Embedded (ou intrinsic) methods:** são métodos em que o processo de seleção das *features* já é executada automaticamente dentro do algoritmo de aprendizado, como é o caso das árvores de decisão.

Capítulo 11: PRINCIPAL COMPONENT ANALYSIS

O PCA é uma técnica comumente utilizada em aplicações como redução de dimensionalidade, compressão com perda de dados, extração de *features* e visualização de dados [16].

Essa técnica funciona encontrando a projeção linear que minimiza a distância quadrática média entre os pontos e suas projeções. Ao minimizar essa distância quadrática média também estamos maximizando a variância das projeções dos pontos no hiperplano de projeção. Isso pode ser visto de maneira intuitiva à medida que escolhemos diferentes hiperplanos de projeção, como mostrado na Figura 11-1 e na Figura 11-2.

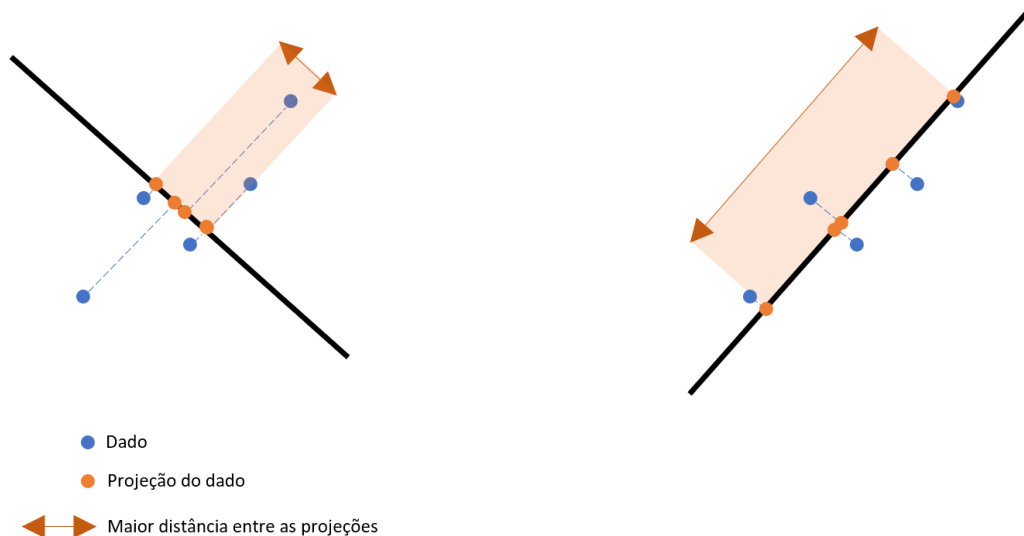


Figura 11-1: Pontos sendo ajustados por dois hiperplanos de projeção diferentes (e perpendiculares). O ajuste da direita é o ajuste que minimiza a distância quadrática média entre os pontos e o hiperplano de projeção, resultando, assim, em uma maior variância das projeções sobre o hiperplano em questão (como pode ser visto pela seta laranja, que mostra a extensão do espalhamento dos dados). No ajuste da esquerda, feita por um hiperplano ortogonal ao da direita, vemos que as projeções dos dados têm uma variância menor.

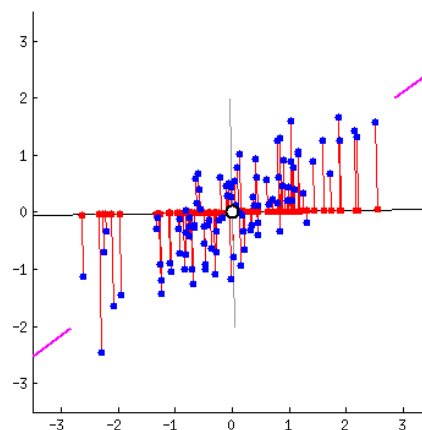


Figura 11-2: Figura animada mostrando as projeções (pontos vermelhos) dos dados (pontos azuis) à medida que giramos o hiperplano de projeção.

Ao reduzirmos a dimensionalidade maximizando a variância nessa projeção, estamos garantindo a menor perda de representatividade dos dados possível. A Figura 11-3 ilustra essa relação entre a maximização da variância e a representatividade da projeção.

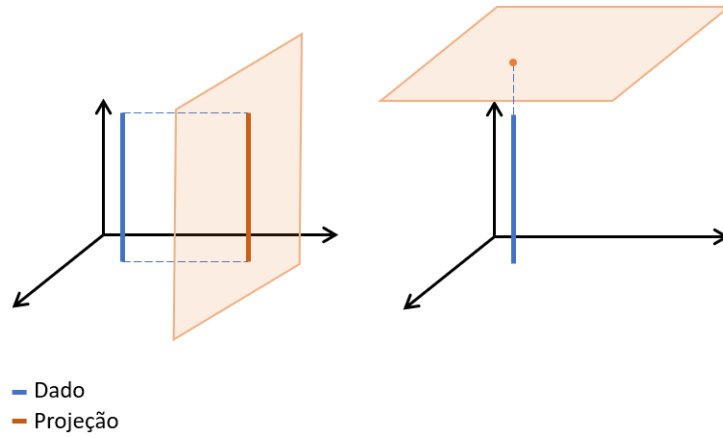


Figura 11-3: Do lado esquerdo, temos uma projeção bastante representativa dos dados, pois o plano foi escolhido de forma a maximizar a variância da projeção. Por outro lado, do lado direito, o plano escolhido minimiza a variância da projeção, resultado em uma projeção pouco representativa dos dados.

A seguir, são mostrados os cálculos envolvidos no método de PCA.

11.1 FORMULAÇÃO DA MÁXIMA VARIÂNCIA

Consideremos um espaço de dimensão D . Nosso objetivo é projetar os dados em um espaço de dimensão $M < D$ de forma que a variância dos dados projetados seja máxima.

Por simplicidade, vamos considerar, inicialmente, que $M = 1$. Podemos representar a direção desse espaço usando o vetor unitário \mathbf{u}_1 de D dimensões. Assim, a média das projeções dos dados é dada por

$$\bar{x}_p = \mathbf{u}_1^T \bar{\mathbf{x}},$$

em que $\bar{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n$ e N é a quantidade total de dados.

A variância dos dados projetados é dada por

$$S_x = \frac{1}{N} \sum_{n=1}^N \left(\mathbf{u}_1^T \mathbf{x}_n - \widehat{\mathbf{u}_1^T \bar{\mathbf{x}}} \right)^2 = \mathbf{u}_1^T \left[\frac{1}{N} \sum_{n=1}^N (\mathbf{x}_n - \bar{\mathbf{x}})^2 \right] \mathbf{u}_1 = \mathbf{u}_1^T \overbrace{\left[\frac{1}{N} \sum_{n=1}^N (\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T \right]}^{\mathbf{S}} \mathbf{u}_1$$

$$S_x = \mathbf{u}_1^T \mathbf{S} \mathbf{u}_1,$$

em que \mathbf{S} é a matriz de covariância dos dados.

Tendo o vetor \mathbf{u}_1 onde os dados serão projetados e a expressão da variância S_x , basta fazermos

$$\max_{\mathbf{u}_1} S_x = \max_{\mathbf{u}_1} \mathbf{u}_1^T \mathbf{S} \mathbf{u}_1, \quad \|\mathbf{u}_1\| = 1.$$

O Lagrangiano desse problema é

$$L(\mathbf{u}_1) = \mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 + \lambda_1 (1 - \mathbf{u}_1^T \mathbf{u}_1),$$

em que λ_1 é um multiplicador de Lagrange.

Para maximizarmos essa expressão, derivamo-la com relação a \mathbf{u}_1 e a igualamos a zero, encontrando como resultado a seguinte equação

$$\mathbf{S}\mathbf{u}_1 = \lambda_1\mathbf{u}_1 \Rightarrow \mathbf{u}_1^T \mathbf{S}\mathbf{u}_1 = S_x = \lambda_1.$$

Da equação acima, notamos que \mathbf{u}_1 é um autovetor de \mathbf{S} , cujo autovalor é λ_1 . Além disso, vemos que a variância dos dados projetados é igual ao autovalor λ_1 . Como está sendo maximizada, então λ_1 é o maior autovalor de \mathbf{S} . O vetor \mathbf{u}_1 recebe o nome de **primeira componente principal**.

Se considerarmos o caso genérico em que o espaço H onde os dados estão sendo projetados tem dimensão M , então $H = \langle \mathbf{u}_1, \dots, \mathbf{u}_M \rangle$ em que $\mathbf{u}_1, \dots, \mathbf{u}_M$ são os autovetores correspondentes aos maiores autovalores $\lambda_1, \dots, \lambda_M$ da matriz de covariância \mathbf{S} dos dados. Além disso, como \mathbf{S} é simétrica, assim como toda matriz de covariância, seus autovetores são todos ortogonais entre si [22]. Lembrando que a variância da projeção dos dados associada a cada autovetor é igual ao seu autovalor correspondente.

Por fim, para encontrar as coordenadas dos novos pontos projetados no espaço H , basta projetar os pontos originais \mathbf{x} nos autovetores $\mathbf{u}_1, \dots, \mathbf{u}_M$. De forma geral,

$$[\mathbf{X}]_H = \mathbf{X}\mathbf{V},$$

em que \mathbf{X} apresenta em cada linha uma instância/ um ponto \mathbf{x}_i , \mathbf{V} é a matriz contendo os autovetores $\mathbf{u}_1, \dots, \mathbf{u}_M$ nas colunas e $[\mathbf{X}]_H$ é a matriz contendo em suas linhas os pontos \mathbf{x}_i projetados na base H .

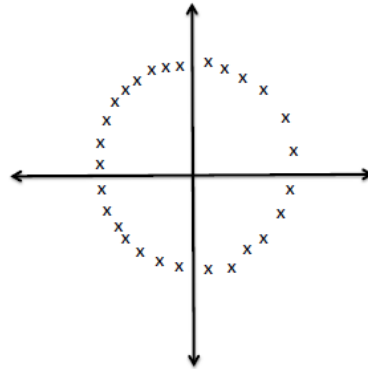
11.2 DETALHES IMPORTANTES

Como foi visto na Seção 11.1, projetamos os dados em vetores \mathbf{u}_i que geram o espaço de projeção H . O problema consiste basicamente em encontrar esses vetores \mathbf{u}_i cujas retas geradas minimizem a distância média quadrática aos pontos. No entanto, para fazer isso, precisamos que os dados estejam centralizados em zero, ou seja, precisamos subtrair dos dados a média deles. Observar na Figura 11-2 como os dados estão centralizados. Se, nessa figura, por exemplo, os dados estivessem centrados em $(0, -3)$, a melhor reta seria diferente, assim como o vetor \mathbf{u} que a gerou. Não seria apenas diferente, como também geraria uma projeção com variância menor. Portanto, é possível ver a importância de centralizar os dados antes de aplicar o PCA.

Outros detalhes que podem estar associados a maus resultados do método PCA são listados abaixo:

- Centralização/escalamento mal realizados. A importância da centralização dos dados já foi discutida acima. O escalamento, por sua vez, também é importante, pois o resultado do PCA pode variar bruscamente com as unidades adotadas nas coordenadas dos dados [23]. Normalmente, realiza-se o escalamento utilizando o desvio padrão. Além disso, podem ser utilizadas outras técnicas de pré-processamento, como remoção de *outliers*.
- Não linearidade dos dados. Se os dados possuírem uma estrutura não linear, é bem possível que essa relação não-linear entre os dados seja perdida ao realizar a projeção

no plano H . Na imagem abaixo, a projeção dos dados em qualquer vetor \mathbf{u} resultaria numa perda da estrutura não linear.



11.3 RELAÇÃO COM A DECOMPOSIÇÃO SVD

Se considerarmos \mathbf{X} a matriz dos dados com dimensões $n \times m$, em que n é o número de dados e m é a dimensão desses dados, podemos escrever a matriz de covariância dos dados como [24]

$$\mathbf{S} = \frac{\mathbf{X}^T \mathbf{X}}{n-1}.$$

Ao realizar a decomposição SVD de $\frac{\mathbf{X}}{\sqrt{n-1}}$, encontramos $\frac{\mathbf{X}}{\sqrt{n-1}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, em que \mathbf{V}^T é a matriz cujas linhas são os autovetores de \mathbf{S} e formam uma base ortonormal (conforme 0). Além disso, na diagonal de $\mathbf{\Sigma}$ estão os valores singulares de $\frac{\mathbf{X}}{\sqrt{n-1}}$, que correspondem a $\sigma_i = \sqrt{\lambda_i}$, com λ_i os autovalores de \mathbf{S} .

Conhecer esses fatos é mais interessante numericamente para fazer os cálculos da PCA, pois, normalmente, é mais fácil determinar os autovalores e autovetores de \mathbf{S} por meio da decomposição SVD do que por meio da multiplicação $\frac{\mathbf{X}^T \mathbf{X}}{n-1}$, que costuma ser mais custosa.

Capítulo 12: AVALIAÇÃO DE DESEMPENHO E SELEÇÃO DE MODELO

12.1 MÉTRICAS PARA CLASSIFICAÇÃO BINÁRIA

As principais métricas usadas para classificação binária são apresentadas a seguir.

12.1.1 Acurácia

$$A = \frac{TP + TN}{TP + FN + TN + FP}$$

É a taxa de acertos do modelo.

Desvantagens:

- Pode levar a conclusões errôneas caso as taxas de acertos de diferentes classes tiverem importâncias diferentes.
- Pouco adequada quando há um desbalanceamento nos dados que favorece muito uma única classe. A taxa de acerto da classe favorecida domina o resultado da acurácia, mascarando a contribuição das taxas de acerto das outras classes. Em um conjunto de dados em que apenas 1% das amostras são da classe 0 e 99% da classe 1, caso o modelo sempre dê uma predição de 1, sua acurácia será de 99%, apesar de ter errado a classificação de todos os dados da classe 0.

12.1.2 True Positive Rate (TPR) ou Sensibilidade

$$TPR = \frac{TP}{TP + FN}$$

Mede quão bom o modelo é com relação não deixar de “detectar” amostras positivas (FN). O nome “sensibilidade” vem de um contexto médico [25, p. 95], em que essa métrica é utilizada para medir a capacidade do modelo de detectar a presença de uma doença (sem deixar ela passar).

Uma tática para lembrar rápido da expressão é perceber que a TPR é 100% quando acertamos a classificação de todas amostras positivas. Com isso, o denominador tem que ser o número total de exemplos positivos.

12.1.3 False Positive Rate (FPR)

$$FPR = \frac{FP}{TN + FP}$$

Mede a taxa de falsos positivos, ou seja, quantas das amostras negativas são classificadas como positivas do total de amostras negativas. Uma forma de lembrar amostras negativas. Uma forma de lembrar dessa expressão, é só pensar que o caso em que o modelo sempre dá uma predição positiva, teremos FPR igual a 100%, pois todas as amostras negativas serão FP e o numerador se igualará ao denominador. Se o numerador fosse a soma das amostras positivas, não teríamos FPR igual a 100% nesse caso.

A TPR e a FPR, geralmente, são reportadas em pares. Outro par equivalente, seria a FNR e a TNR, que são os complementares da TPR e FPR, respectivamente.

12.1.4 Precisão

$$P = \frac{TP}{TP + FP}$$

Mnemônico: *P*recisão = TP / *P*reditas Positivas.

Mede quão preciso é o modelo na sua seleção de amostras positivas, ou seja, quantas das amostras preditas como positivas são de fato positivas. Está relacionada à capacidade do modelo de evitar falsos positivos.

A precisão e a FPR fornecem informação sobre a capacidade do modelo de evitar falsos positivos. No entanto, quando o número de negativos é muito maior do que o de positivos (o que é muito comum em problemas de diagnóstico de uma doença rara), é preferível utilizar a precisão. Isso porque, nesse caso, o denominador da FPR tende a ficar muito grande e a minimizar o valor de FPR. Por outro lado, a precisão não é afetada por isso, já que o denominador leva em conta apenas as amostras classificadas como positivas.

12.1.5 Recall

$$R = \frac{TP}{TP + FN}$$

Mnemônico: *R*ecall = TP / *R*eal Positive.

É equivalente a TPR. Logo, mede a capacidade do modelo de identificar amostras positivas do total de amostras positivas, ou seja, a capacidade do modelo de evitar falsos negativos.

12.1.6 F1-score

$$F_1 = 2 \frac{PR}{P + R}$$

Mede o equilíbrio entre precisão e *recall* ou, de forma equivalente, o equilíbrio entre falsos positivos e falsos negativos. Varia de 0 a 1. Se o valor for baixo, a métrica indica que pelo menos *R* ou *P* são baixos. Sendo *R* e *P* altos, F_1 também será alto.

É interessante notar que o F1-score é a média harmônica de precisão e do *recall*. Reparar que $F_1 = \frac{2}{\frac{1}{P} + \frac{1}{R}}$. O motivo para se escolher a média harmônica e não a média aritmética, por

exemplo, é que a primeira penaliza os casos em que há desequilíbrio entre *P* e *R*, ou seja, *P* elevado e *R* baixo e vice-versa. Isso pode ser visto por meio dos gráficos das médias aritmética e harmônica para diferentes combinações de *P* e *R*, como mostrado na Figura 11-4. Isso penaliza casos como o do exemplo da Seção 12.1.1, em que o modelo sempre prevê a mesma classe. Nesse caso, a média harmônica penaliza mais o modelo do que a aritmética, o que é razoável, já que ele é muito ruim.

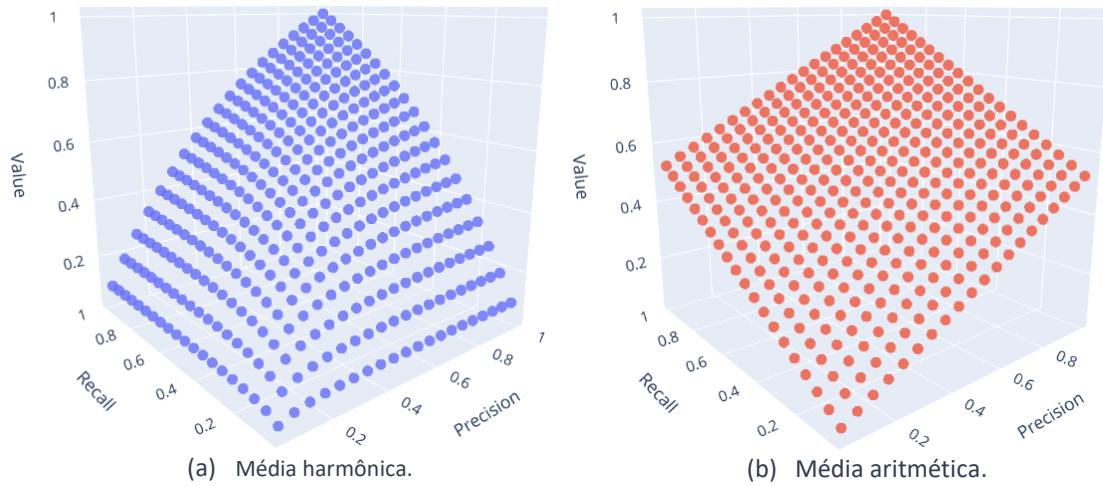


Figura 11-4: Comparação entre a média harmônica e aritmética da precisão e do recall.

12.1.7 F-score

$$F_{\alpha} = \frac{(1 + \alpha)PR}{\alpha P + R}$$

É a forma geral do F1-score, que permite atribuir graus de importância diferentes para R e P . Para $\alpha > 1$, estamos dando mais importância para R e, para $\alpha < 1$, estamos dando mais importância para P (apesar de isso não ser muito intuitivo, olhando apenas para a expressão). $\alpha = 2$ indica que estamos dando um peso duas vezes maior para R . Isso fica mais fácil de compreender quando percebemos que essa expressão corresponde à média harmônica ponderada de P e R , pois temos

$$F_{\alpha} = \frac{1 + \alpha}{\alpha R^{-1} + P^{-1}} = \left(\frac{\alpha}{1 + \alpha} R^{-1} + \frac{1}{1 + \alpha} P^{-1} \right)^{-1}.$$

Com a primeira igualdade, fica mais simples de ver que isso se trata de uma média harmônica ponderada. Com a segunda igualdade, vemos que o termo R^{-1} tem mais importância quando α maior que 1, pois nesse caso, $\frac{\alpha}{1+\alpha} > \frac{1}{1+\alpha}$. No caso limite em que $\alpha \rightarrow \infty$, $\frac{\alpha}{1+\alpha} = 1$ e $\frac{1}{1+\alpha} = 0$ e o termo associado a P tem nenhuma importância. Portanto, não importa se P é grande ou pequeno, pois ele terá uma porcentagem menor de participação no resultado final quando comparado com R . Se por exemplo, tivéssemos $\alpha = 3$, então $F_2 = (0,75R^{-1} + 0,25P^{-1})^{-1}$.

12.1.8 Receiver Operating Characteristic (ROC)

É o gráfico da TPR pela FPR, como mostrado na Figura 11-5. Em problemas em que temos que decidir um limiar de probabilidade para classificar uma amostra como positiva, essa curva é útil, pois ela resume a relação da TPR e da FRP para diferentes valores de limiar. Se as duas métricas tiverem importâncias iguais, o melhor ponto dessa curva é na parte superior esquerda, já que quanto menor a FPR, melhor.

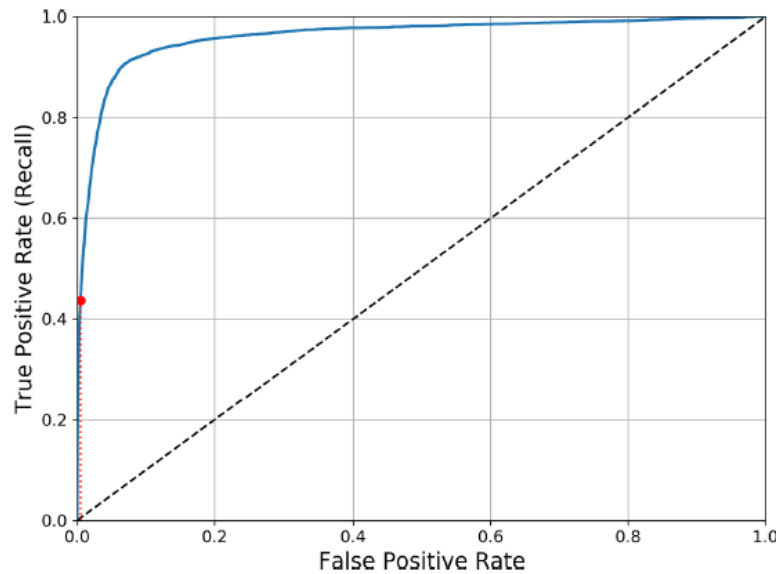


Figura 11-5: Curva ROC.

Para diferentes valores de hiperparâmetros, teremos diferentes curvas ROC. Uma forma de comparar diferentes curvas, é observar a forma de cada uma delas e determinar a que fornece a melhor combinação de TPR e FPR. Usualmente, quanto mais perto do extremo superior esquerdo estiver o ponto melhor. No entanto, isso não é muito conveniente. Uma forma mais prática de comparar essas curvas é usar a métrica AUC (*Area Under the Curve*). Quanto maior o AUC, melhor é o modelo. Um modelo perfeito terá AUC igual a 1.

Uma alternativa à curva ROC é a curva Precisão x *Recall*, que normalmente é utilizada quando temos um número de amostras negativas muito maior do que o de amostras positivas.

12.2 MÉTRICAS PARA CLASSIFICAÇÃO MULTICLASSE

Em um problema de classificação multiclasse, ainda podemos utilizar as métricas para o problema binário, mas para cada classe. Por exemplo, se tivermos 3 classes, A, B e C, poderíamos analisar cada classe individualmente, calculando as métricas da Seção 12.1 para cada uma das classes. Para isso usamos uma estratégia *one-vs-the-rest*, ou seja, consideramos a classe em questão como positiva e todas as outras como negativa. Por exemplo, para calcular a precisão para a classe A, consideramos a classe A como positiva e as classes B e C como negativas.

Apesar de podermos usar essa estratégia, isso faz com que tenhamos várias métricas, o que torna a análise complicada, principalmente quando o número de classes é grande. Por isso, é importante definir métricas que resumam a performance global do modelo. Essas métricas são apresentadas a seguir. Na definição dessas métricas, consideraremos um conjunto de classes Ω , com cardinalidade $K \equiv |\Omega|$, e utilizaremos o termo C_{rp} para representar a contagem de casos pertencentes à classe r ('real') que foram classificados como classe p ('predito').

12.2.1 Acurácia

$$A = \frac{\sum_{r=p} C_{rp}}{\sum_{r,p \in \Omega} C_{rp}}$$

É o total de acertos sobre o total de casos.

Quando o número de exemplos de cada classe é desbalanceado, essa métrica pode mascarar o desempenho de classes com baixa contagem de exemplos. Assim, se uma classe minoritária tiver uma taxa de erros muito elevada, mas todas as outras classes tiverem bom desempenho, a acurácia terá a assumir valores mais altos, mascarando o mau desempenho da classe minoritária.

12.2.2 Acurácia Balanceada

$$A_b = \frac{1}{K} \sum_{\omega \in \Omega} R_{\omega},$$

em que $R_{\omega} = \frac{C_{\omega\omega}}{\sum_{p \in \Omega} C_{\omega p}}$ é o *recall*/TPR de uma classe ω .

Logo, a acurácia balanceada é a média das dos valores de TPR para as diferentes classes.

Essa métrica atribui importâncias iguais para cada classe, em vez de atribuir importâncias iguais para cada amostra independentemente da classe, como é o caso da acurácia. Portanto, se considerarmos um exemplo em que há uma classe com poucas amostras e muitos erros de classificação e outras classes com várias amostras e boa performance, teremos $A > A_b$. Isso porque a classe minoritária terá um papel para o cálculo da acurácia balanceada tão importante quanto as outras.

Para conjuntos de dados balanceados, não faz sentido usar essa métrica, pois ela tende a se igual à acurácia. Além disso, mesmo que o conjunto de dados seja desbalanceado, se a taxa de acertos global for mais importante do que controlar a taxa de acertos em cada classe, talvez a acurácia seja uma métrica mais recomendada do que a acurácia balanceada.

12.2.3 Métricas Macro

Macro é uma maneira de calcular métricas que considera cada classe como a unidade básica para o cálculo, ou seja, é a média das métricas calculadas para cada classe no esquema *one-vs-all* (uma classe é considerada positiva e as outras negativas).

Consideremos que $P_{\omega} = \frac{C_{\omega\omega}}{\sum_{r \in \Omega} C_{r\omega}}$, $R_{\omega} = \frac{C_{\omega\omega}}{\sum_{p \in \Omega} C_{\omega p}}$ e $F_{1\omega} = \frac{2P_{\omega}R_{\omega}}{P_{\omega}+R_{\omega}}$ são, respectivamente, a precisão, o *recall* e o f1-score para uma certa classe ω , considerando que essa classe é a positiva e todas as outras negativas.

Dessa forma, a precisão, o *recall* e o f1-score macro são dados, respectivamente, por:

$$\begin{aligned} P_{macro} &= \frac{1}{K} \sum_{\omega \in \Omega} P_{\omega}; \\ R_{macro} &= \frac{1}{K} \sum_{\omega \in \Omega} R_{\omega}; \\ F_{1macro} &= \frac{1}{K} \sum_{\omega \in \Omega} F_{1\omega}. \end{aligned} \tag{11.1}$$

É interessante mencionar que em [26], é dada uma definição diferente para o f1-score macro, em que essa métrica é calculada a partir da média harmônica de P_{macro} e R_{macro} , ou seja,

$2 \frac{P_{macro} R_{macro}}{P_{macro} + R_{macro}}$. No entanto, a definição usada na biblioteca *sklearn* do Python não é a que foi apresentada na eq. (11.1).

Além disso, é interessante observar que a acurácia balanceada é igual ao *recall* macro. No entanto, a acurácia macro não é igual à acurácia balanceada.

12.2.4 Métricas Macro Ponderadas (*weighted*)

São métricas calculadas de maneira similar às macro, porém ponderando as métricas de cada classe usando o número de amostras de cada uma delas. Dessa forma, a precisão, o *recall* e o f1-score macro ponderados são dados por

$$P_w = \frac{\sum_{\omega \in \Omega} N_{\omega} P_{\omega}}{\sum_{\omega \in \Omega} N_{\omega}};$$

$$R_w = \frac{\sum_{\omega \in \Omega} N_{\omega} R_{\omega}}{\sum_{\omega \in \Omega} N_{\omega}};$$

$$F_{1w} = \frac{\sum_{\omega \in \Omega} N_{\omega} F_{1\omega}}{\sum_{\omega \in \Omega} N_{\omega}}.$$

N_{ω} é o total de amostras pertencentes à classe ω .

Essas métricas devem ser utilizadas se for desejável ter métricas que resumam o desempenho global do modelo levando em consideração a quantidade de amostras em cada classes. As classes com poucas amostras receberão menos importância e as classes com mais amostras receberam mais importância no resultado final. Caso se queira dar a mesma importância para todas as classes, mas ainda assim ter métricas globais, é melhor utilizar as métricas macro.

12.2.5 Métricas Micro

Essas métricas são calculadas de maneira global, sem diferenciar amostras individuais pela classe a qual elas pertencem, considerando o número total de TP, FP e FN. Dessa forma, a precisão micro e o *recall* são dados por

$$P_{micro} = \sum_{\omega \in \Omega} \frac{C_{\omega\omega}}{\sum_{\gamma \in \Omega} C_{\gamma\omega}} = \frac{\sum_{\omega \in \Omega} C_{\omega\omega}}{\sum_{\omega \in \Omega} \sum_{\gamma \in \Omega} C_{\gamma\omega}};$$

$$R_{micro} = \sum_{\omega \in \Omega} \frac{C_{\omega\omega}}{\sum_{\gamma \in \Omega} C_{\omega\gamma}} = \frac{\sum_{\omega \in \Omega} C_{\omega\omega}}{\sum_{\omega \in \Omega} \sum_{\gamma \in \Omega} C_{\omega\gamma}}.$$

Reparar que $P_{micro} = R_{micro}$. Portanto, o F1-score é dado por

$$F_{1micro} = 2 \frac{P_{micro} R_{micro}}{P_{micro} + R_{micro}} = P_{micro} = R_{micro}.$$

Por fim, reparar que $P_{micro} = R_{micro} = F_{1micro}$ são iguais à acurácia, pois essas métricas consistem na razão entre os acertos e o total de amostras.

12.3 MÉTRICAS PARA REGRESSÃO

As principais métricas usadas para regressão são apresentadas abaixo.

12.3.1 Mean Squared Error (MSE)

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N (y_i - t_i)^2$$

Vantagem:

- É diferenciável.

Desvantagem:

- Não é medido nas mesmas unidades que t_i e y_i .
- Como os resíduos são elevados ao quadrado, é dado um peso maior para *outliers*.

12.3.2 Root Mean Squared Error (RMSE)

$$\text{RMSE} = \sqrt{\text{MSE}}$$

Vantagem:

- É diferenciável.
- É medido nas mesmas unidades que t_i e y_i .

Desvantagem:

- Como os resíduos são elevados ao quadrado, é dado um peso maior para *outliers*.
- Apesar de ser medido nas mesmas unidades que t_i , o RMSE não representa a diferença média entre o *target* e o valor predito. Um RMSE igual a 10 não significa que o modelo erra em 10 na média.

12.3.3 Mean Absolute Error (MAE)

$$\text{MAE} = \frac{1}{N} \sum_{i=1}^N |y_i - t_i|$$

Vantagem:

- Como os resíduos não são elevados ao quadrado, todos os erros são pesados de maneira igual.
- É medido nas mesmas unidades que os resíduos, t_i e y_i .

Desvantagem:

- Não é facilmente diferenciável.

12.3.4 Mean Absolute Percentage Error (MAPE)

$$\text{MAPE} = \frac{1}{N} \sum_{i=1}^N \left| \frac{y_i - t_i}{t_i} \right|$$

Vantagem:

- É uma porcentagem. Logo, é independente de escala e pode ser comparado com MAE de modelos de diferentes conjuntos de dados.

Desvantagem:

- Não é facilmente diferenciável.

12.3.5 Coeficiente de Correlação de Pearson (PCC)

$$r_{yt} = \frac{S_{yt}}{S_y S_t}$$

S_y e S_t são as variâncias amostrais de y e t , respectivamente, e S_{yt} é a covariância amostral entre y e t .

12.3.6 Coeficiente de Determinação (R^2)

12.3.6.1 Modelos lineares

O coeficiente de determinação surge naturalmente para avaliação de modelos de regressão lineares. Considerando que \bar{t} e S_t são, respectivamente, a média e a variância amostrais dos *targets*, a seguinte expressão é válida para o coeficiente de determinação para modelos de regressão linear [27, p. 68]

$$R^2 = 1 - \frac{\sum_{i=1}^N (y_i - t_i)^2}{\sum_{i=1}^N (t_i - \bar{t})^2} = 1 - \frac{\text{MSE}}{S_t} = 1 - \frac{\text{SS}_{\text{res}}}{\text{SS}_{\text{tot}}} = \frac{\text{SS}_{\text{reg}}}{\text{SS}_{\text{tot}}}, \quad (11.2)$$

em que [27, p. 44]

$$\begin{aligned} \text{SS}_{\text{res}} &= \text{MSE} \times N = \sum_{i=1}^N (y_i - t_i)^2 && \text{:Sum of Squared Residuals (também chamada de} \\ &&& \text{variância não explicada pelo modelo)} \\ \text{SS}_{\text{tot}} &= S_t \times N = \sum_{i=1}^N (t_i - \bar{t})^2 && \text{:Total Sum of Squared Deviations} \\ \text{SS}_{\text{reg}} &= \sum_{i=1}^N (y_i - \bar{t})^2 && \text{:Sum of Squared residuals when considering} \\ &&& \text{Regression (também chamada de variância} \\ &&& \text{explicada pelo modelo)} \end{aligned}$$

O que esse coeficiente indica é a proporção de desvio quadrático quando consideramos que os valores y_i da regressão são verdadeiros (mas mantendo a média verdadeira \bar{t}) com relação ao desvio quadrático real (mantendo os valores reais t_i). Apesar de SS_{reg} compor SS_{tot} (observar que $\text{SS}_{\text{tot}} = \text{SS}_{\text{res}} + \text{SS}_{\text{reg}}$), essa interpretação dificulta um pouco o entendimento. Podemos enxergar essa divisão apenas como uma forma de reportar o SS_{reg} de maneira proporcional. Sozinho, o SS_{reg} não tem muito valor interpretativo. No entanto, quando o dividimos por SS_{tot} , estamos comparando-o a uma referência.

Outra forma de interpretar essa métrica é olhando para a primeira igualdade. O termo $\frac{\text{MSE}}{S_t}$ pode ter uma interpretação análoga a que criamos no parágrafo anterior: é uma forma de reportar o MSE, porém, comparado a uma grandeza de referência, como a variância amostral S_t . O R^2 portanto, é o complementar disso.

Por fim, uma última interpretação pode ser feita. O termo SS_{tot} representa o SS'_{reg} de um modelo de regressão que sempre tem como saída a média \bar{t} . Portanto, estamos comparando o SS_{reg} do modelo com o SS'_{reg} de um modelo trivial que sempre prevê a média.

$R^2 = 1$ quando o modelo sempre acertar as previsões, ou seja, quando $\text{MSE} = \text{SS}_{\text{reg}} = 0$.

$R^2 = 0$ quando o modelo for substituível pelo modelo que sempre prevê a média, ou seja, $SS_{\text{reg}} = SS_{\text{tot}}$.

$R^2 < 0$ quando o modelo for pior do que o modelo que sempre prevê a média, ou seja, $MSE > S_t$. Nesse caso, seria melhor simplesmente prever a média.

12.3.6.2 Modelos não-lineares

Para modelos não-lineares, a equação (11.2) não é válida [28]. Isso porque $SS_{\text{tot}} \neq SS_{\text{res}} + SS_{\text{reg}}$ e, por isso, $1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}} \neq \frac{SS_{\text{reg}}}{SS_{\text{tot}}}$. Dessa forma, o coeficiente de determinação é dado por

$$R^2 = 1 - \frac{MSE}{S_t} = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}.$$

Assim como no caso linear, $R \leq 1$, sendo

$R^2 = 1$ quando o modelo acertar todas as previsões;

$R^2 = 0$ quando o modelo for equivalente a prever sempre a média;

$R^2 < 0$ quando o modelo for pior do que o modelo que só prevê a média.

12.4 MÉTODOS DE VALIDAÇÃO

Capítulo 13:

Capítulo 14: APÊNDICES

Abaixo são apresentados os apêndices.

TEORIA DA INFORMAÇÃO

CASO DISCRETO

INFORMAÇÃO

A **informação** é uma grandeza que quantifica a “surpresa” causada por um valor observado x da variável aleatória $X: \Omega \rightarrow \mathcal{X}$, com p.m.f $p(x) = \mathbb{P}(X = x)$. A expressão da informação é dada por

$$I(x) = -\log p(x). \quad (0.1)$$

Quando a base do logaritmo é 2, a informação é medida em **bits**. Quando o logaritmo é natural, a informação é medida em **nats**. Em *machine learning*, usualmente utilizamos o logaritmo natural.

Reparar que, como $p(x) \in [0,1]$, $I(x) > 0$.

Com isso, vemos que quanto mais raro for um evento, maior é a informação associada a ele. Então, a informação de que “hoje o sol nasceu”, que tem probabilidade praticamente 1, teria informação nula, já que não é nenhuma surpresa de que o sol nasce todos os dias. Se por acaso, algum dia o sol não nascesse, a informação teria valor infinito, pois é algo com probabilidade praticamente 0 de acontecer.

Tentando dar outra interpretação intuitiva para essa grandeza, é como se ela medisse o quanto uma informação é informativa. Assim, “o sol nasceu” é pouco informativo e, por isso, tem informação nula.

De maneira similar, podemos definir a informação condicional como sendo a informação que ainda resta de um valor observado x quando se conhece um valor observado y da v.a. Y . Ela é dada por

$$I(x|y) = -\log p(x|y). \quad (0.2)$$

Continuamos tendo a mesma interpretação da informação. Só trocamos a p.m.f $p(x)$ por outra p.m.f $g(x) = p(x|y)$.

É importante ressaltar que, a informação condicional NÃO é a parcela de informação (ou “surpresa”) de x que se esvai por conhecer y , ou seja, não é a parcela da incerteza de x compartilhada com y . Isso ficará claro mais para frente.

ENTROPIA

A entropia é a informação média (ou “surpresa” média) de uma variável aleatória, ou seja, é a soma da informação para cada valor observado x ponderado pela sua probabilidade (ou “frequência”, sob uma ótica frequentista). Sua expressão é dada por

$$H(X) = \mathbb{E}[I(X)] = -\sum_{x \in \mathcal{X}} p(x) \log p(x). \quad (0.3)$$

Observar que $H(X) \geq 0$, pois $\ln p(x_i) \leq 0$, já que $p(x) \in [0,1]$.

A entropia é mínima quando $X \sim \delta_c$, ou seja, $\mathbb{P}(X = c) = p(c) = 1$. Intuitivamente, apenas um evento ocorre com probabilidade 1, de forma que não temos nenhuma “surpresa”.

A entropia é máxima quando $X \sim \mathcal{U}(a, b)$. Nesse caso, a entropia é dada por $H(X) = \ln M$, com $M = b - a + 1$.

ENTROPIA CONDICIONAL

É a informação condicional média. Dessa forma, nos baseando na eq. (0.3) e na *Law of The Unconscious Statistician* (LOTUS) [3, p. 170], também conhecida como *Rule of the Lazy Statistician* [2, p. 48], podemos escrever

$$H(X|Y) = \mathbb{E}[I(X|Y)] = - \sum_{y \in \mathcal{Y}} \sum_{x \in \mathcal{X}} p(x, y) \log p(x|y). \quad (0.4)$$

Lembrando que estamos encarando $I(X|Y) = g(X, Y)$ como uma função das variáveis aleatórias X e Y .

INFORMAÇÃO MÚTUA

Definimos a seguinte quantidade

$$I(x; y) = \log \frac{p(x|y)}{p(x)} = \log \frac{p(x, y)}{p(x)p(y)} = \log \frac{p(y|x)}{p(y)} = I(y; x), \quad (0.5)$$

que, em [5, p. 56], é chamada de informação mútua. Ela quantifica a informação que temos de x por conhecer y e vice-versa (já que $I(x; y) = I(y; x)$), ou seja, é a parcela da informação compartilhada por x e y . Isso fica mais claro quando escrevemos essa quantidade como $I(x; y) = I(x) - I(x|y)$. $I(x|y)$ é a “surpresa” de x que ainda resta depois de conhecermos y . Ao subtrairmos essa quantidade de $I(x)$, que é toda a incerteza associada a x , ficamos com a parcela da informação de x que conhecemos plenamente só por conhecermos y . A informação mútua ainda pode ser vista como a redução da incerteza ou “surpresa” associada a x por conhecer y .

Se $Y = X$, então $I(x; y) = \log \frac{1}{p(x)} = I(x)$, o que indica que conhecer o valor observado y reduz a “surpresa” associada a x de $I(x)$ unidades de informação, ou seja, reduz a incerteza de x a zero. Por outro lado, se X e Y forem independentes, então $I(x; y) = 0$, o que indica que conhecer o valor y não reduz em nada a incerteza sobre x .

De forma análoga ao que foi feito para a entropia e para a entropia condicional, podemos calcular a informação mútua média associada às variáveis aleatória X e Y . A informação mútua média é dada pela expressão

$$I(X; Y) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}. \quad (0.6)$$

Pode ser mostrado que $I(X; Y) \geq 0$, sendo nula somente quando X e Y forem independentes.

De maneira geral, chamamos a quantidade definida pela eq. (0.6) de **informação mútua**. Apesar de em [5, p. 56] haver uma diferenciação de nomenclatura entre a eq. (0.5) e a eq. (0.6), normalmente a eq. (0.5) não é muito comum e preferimos reservar o nome “informação mútua” para a eq. (0.6).

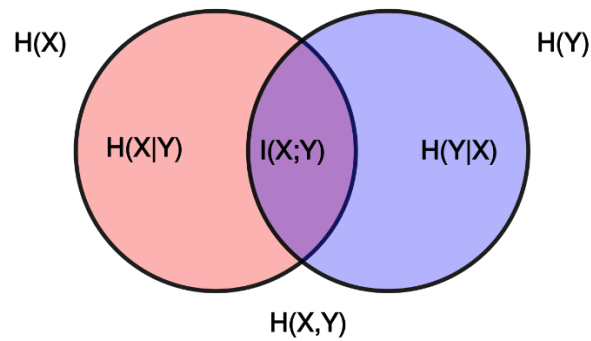
Assim, podemos ver que a informação mútua é uma ótima forma para quantificar a independência entre duas variáveis aleatórias. Com isso, vemos que outra interpretação que poderíamos fazer da informação mútua é que ela quantifica o quanto de informação X e Y compartilham entre si.

Por fim, reparamos que também podemos escrever

$$I(X; Y) = H(X) - H(X|Y), \quad (0.7)$$

o que está de acordo com a interpretação dada no início desta subseção, de que a informação mútua mede a redução da “surpresa” média associada a v.a. X por conhecermos a v.a. Y .

O diagrama abaixo ilustra a relação da informação mútua com as outras quantidades vistas nesta seção:



DIVERGÊNCIA KULLBACK-LEIBLER

A **divergência Kullback-Leibler** ou **entropia relativa** é uma medida utilizada para quantificar a diferença entre uma distribuição $p(x)$ e uma distribuição de referência $q(x)$. Ela é dada pela expressão

$$KL(p||q) = \mathbb{E}[I_q(x) - I(x)] = \mathbb{E}\left[\log \frac{p(x)}{q(x)}\right] = \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)}, \quad (0.8)$$

que se lê como “divergência KL de p para q ” e com isso estamos calculando o quanto a distribuição p diverge de q . $I(x)$ é a informação de X , que tem distribuição $p(x)$, e $I_q(x)$ é a informação de X quando aproximamos sua distribuição por q . Vemos que p é tido como referência, até mesmo porque multiplicamos $\log \frac{p(x)}{q(x)}$ por $p(x)$ no somatório, pois estamos considerando que a distribuição de X é descrita por $p(x)$ nesse referencial. É importante notar que $KL(p||q) \neq KL(q||p)$, ou seja, ao mudarmos o referencial, a divergência muda. Logo, não podemos dizer que a divergência KL representa uma métrica de distância.

Observar que, como $KL(p||q) = \mathbb{E}[I_q(x) - I(x)]$, temos que

$$KL(p||q) = H_q(X) - H(X),$$

em que $H_q(X)$ é a entropia de X quando aproximamos sua distribuição por q , também chamada de **entropia cruzada**, como veremos adiante.

Em *machine learning*, muitas vezes utilizamos a expressão $KL(p||q)$ para quantificar o quanto uma distribuição desconhecida p é divergente de uma distribuição q que tenta aproximá-la.

É importante reparar que $KL(p||q) \geq 0$ [16, p. 55].

Finalmente, reparamos que existe uma relação entre a informação mútua e a divergência KL, que é ilustrada na equação:

$$I(X; Y) = KL(p(x, y) || p(x)p(y)) \quad (0.9)$$

ENTROPIA CRUZADA

Às vezes, em problemas de *machine learning*, podemos nos deparar com a expressão da **entropia cruzada**, que é dada por

$$H_q(X) = - \sum_{x \in \mathcal{X}} p(x) \log q(x). \quad (0.10)$$

Ela é a “surpresa” média que temos ao codificarmos uma v.a. X que tem distribuição $p(x)$ com uma distribuição $q(x)$.

Se repararmos bem, ela é muito parecida com a equação da divergência KL. De fato, temos que

$$KL(p||q) = H_q(X) - H(X).$$

Considerando que $p(x)$ é a distribuição que dá a codificação ótima para X (que tem menor informação média necessária associada), então a divergência KL é a “surpresa” média por tentarmos utilizar uma distribuição q , que não é ótima (vai exigir mais informação para codificar X), menos a “surpresa” média ao utilizarmos a distribuição ótima p (entropia).

A equação da entropia cruzada surge quando temos uma variável que tem uma distribuição p e queremos aproximá-la com outra distribuição q_θ , parametrizada pelo vetor de parâmetros θ a ser otimizado. Para isso, o mais óbvio a se fazer é minimizar a divergência $KL(p||q)$, o que resulta em

$$\min_{\theta} KL(p||q) = \min_{\theta} H_q(X) - H(X) = \min_{\theta} H_q(X),$$

pois $H(X)$ não depende do vetor de parâmetros θ , já que $p(x)$ também não depende.

Geralmente, a entropia cruzada surge no contexto de classificação, pois ela é usualmente utilizada no cálculo da função custo. Se considerarmos o caso da classificação com K classes $\omega_1, \omega_2, \dots, \omega_K$, teríamos um problema modelado da seguinte forma: $\Omega = \{\omega_1, \omega_2, \dots, \omega_K\}$, representando o espaço amostral; $\mathcal{X} = [1, K]$ representando o espaço imagem da variável aleatória $X(\omega)$; $q_x = q(x) = \mathbb{P}_q(X = x)$ sendo a p.m.f estimada de X ; $p_x = p(x) = \mathbb{P}(X = x)$ sendo a p.m.f real de X .

Considerando que estejamos avaliando um ponto \mathbf{v} , que representa um dado, pertencente a uma classe ω_j , teremos a seguinte expressão para a entropia cruzada:

$$H_q(X) = - \sum_{x=1}^K p_x \log q_x$$

Como a classe do ponto \mathbf{v} é conhecida, a distribuição X das classes para esse ponto é $\mathbb{P}(X = j) = 1$, ou seja, $\mathbb{P}(X = x) = p_x = 0$ para $x \neq j$. Logo,

$$H_q(X) = - \sum_{x=1}^K \mathbb{1}_{\{\mathbf{v} \in \omega_x\}} \log q_x = - \log q_j.$$

Observar que, no desenvolvimento acima, x não representa os dados! Para evitar confusões, podemos reescrever a equação da entropia cruzada para um determinado ponto \mathbf{v} trocando as nomenclaturas da seguinte forma:

$$H_q(W_{\mathbf{v}}) = - \sum_{k=1}^K p_k \log q_k$$

em que $W_{\mathbf{v}}$ é a variável aleatória que representa o número da classe a qual pertence o ponto \mathbf{v} .

CASO CONTÍNUO

As equações para o caso discreto podem ser generalizadas para o caso contínuo. O desenvolvimento necessário para isso está explicado em [5, p. 59].

ENTROPIA

Também chamada de **entropia diferencial** é dada pela expressão

$$H(X) = - \int_{-\infty}^{\infty} p(x) \log p(x) dx \quad (0.11)$$

A entropia é máxima quando X segue uma distribuição normal.

Observar que, no caso contínuo, $p(x)$ pode assumir valores maiores do que 1, o que pode fazer com que possamos obter valores negativos de entropia.

ENTROPIA CONDICIONAL

A entropia condicional é dada pela expressão

$$H(X|Y) = - \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} p(x, y) \log p(x|y) dx dy \quad (0.12)$$

INFORMAÇÃO MÚTUA

A informação mútua é dada pela expressão

$$I(X; Y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} dx dy \quad (0.13)$$

DIVERGÊNCIA KULLBACK-LEIBLER

A divergência Kullback-Leibler é dada pela expressão

$$\text{KL}(p||q) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} dx \quad (0.14)$$

DECOMPOSIÇÃO SVD

Qualquer matriz A de $m \times n$ pode ser decomposta da seguinte maneira

$$A = U\Sigma V^T \quad (0.1)$$

em que

U é uma matriz ortogonal $m \times m$ cujas colunas são os autovetores de AA^T e são chamadas de *left singular vectors de A* [29],

V^T é uma matriz ortogonal $n \times n$ cujas colunas são os autovetores de $A^T A$ e são chamadas de *right singular vectors de A* e

Σ é uma matriz diagonal (possivelmente retangular) $m \times n$ cujos valores σ_i da diagonal principal são as raízes quadradas dos autovalores de AA^T e de $A^T A$ e são chamados de *valores singulares*.

Essa decomposição é chamada de SVD [30]. A Figura 11-6 ilustra essa decomposição.

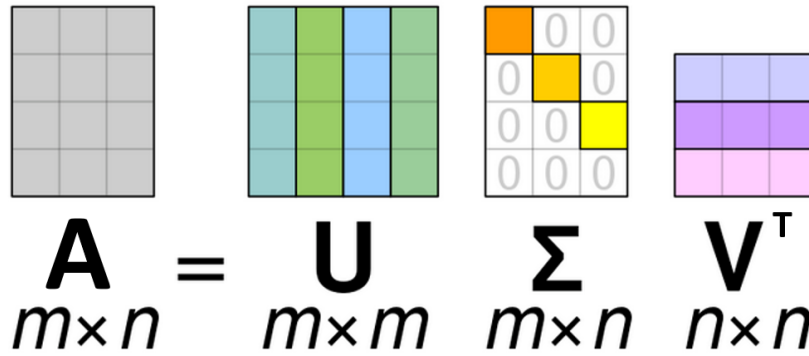


Figura 11-6: Representação gráfica da decomposição SVD.

Lembrando que AA^T e de $A^T A$ são matrizes diferentes (normalmente com dimensões diferentes) que apresentam autovetores diferentes, mas os mesmos autovalores. Lembrando também que, o número de autovalores é igual ao posto (*rank*) da matriz.

A partir da eq. (0.1), podemos escrever

$$AA^T = U\Sigma V^T A^T$$

$$AA^T = U\Sigma V^T V \Sigma U^T = U\Sigma^2 U^T = U\Lambda U^T,$$

lembrando que $\Sigma = \Sigma^T$ e que $VV^T = I$, pois V é ortogonal. A expressão acima é a diagonalização da matriz AA^T (decomposição espectral). Logo, fica claro que as colunas de U são de fato os autovetores de AA^T . Além disso, vemos que os valores σ_i da diagonal de Σ correspondem a

$$\sigma_i = \sqrt{\lambda_i},$$

em que λ_i são os valores da diagonal da matriz Λ de autovalores de AA^T .

É possível provar que toda matriz A $m \times n$ de posto 1 pode ser escrita como produto externo de um vetor de tamanho m por outro de tamanho n [29]:

$$A = \mathbf{u}\mathbf{v}^T = \begin{bmatrix} - & u_1\mathbf{v}^T & - \\ \vdots & \vdots & \vdots \\ - & u_m\mathbf{v}^T & - \end{bmatrix} = \begin{bmatrix} | & \cdots & | \\ v_1\mathbf{u} & \cdots & v_n\mathbf{u} \\ | & \cdots & | \end{bmatrix}.$$

Além disso, toda matriz de posto k pode ser escrita como a soma de k matrizes de posto 1 [29]. A título de ilustração, uma matriz A $m \times n$ de posto 2 ficaria

$$A = \mathbf{u}\mathbf{v}^T + \mathbf{w}\mathbf{z}^T = \begin{bmatrix} - & u_1\mathbf{v}^T + w_1\mathbf{z}^T & - \\ \vdots & \vdots & \vdots \\ - & u_m\mathbf{v}^T + w_m\mathbf{z}^T & - \end{bmatrix} = \begin{bmatrix} | & | \\ \mathbf{u} & \mathbf{w} \\ | & | \end{bmatrix} \begin{bmatrix} - & \mathbf{v}^T & - \\ \mathbf{z}^T & - \end{bmatrix}. \quad (0.2)$$

Com isso, podemos escrever a decomposição SVD como

$$A = \sum_{i=1}^{\min(m,n)} \sigma_i \mathbf{u}_i \mathbf{v}_i^T. \quad (0.3)$$

Logo, podemos escrever a decomposição SVD como a soma dos produtos externos da coluna i da matriz U com a linha i da matriz V^T escalados pelo valor singular i . Isso faz sentido quando olhamos para o termo da direita da eq. (0.2).

LOW-RANK APROXIMATION POR SVD

A *low-rank approximation* de uma matriz usando SVD consiste em aproximar a matriz A por uma matriz \tilde{A} de posto $k < \text{rank}(A)$. Isso é feito ao organizar a decomposição da eq. (0.1) de forma que os valores singulares estejam dispostos em Σ em ordem decrescente (o primeiro valor da diagonal é o maior) e zerando os menores valores singulares (e as colunas/linhas correspondentes nas matrizes U e V^T) até restarem k valores singulares não-nulos. Isso é equivalente, na eq. (0.3), a eliminar as parcelas com os menores valores de σ_i até restarem apenas k parcelas.

Isso faz sentido porque, como \mathbf{u}_i e \mathbf{v}_i^T são unitários, a norma de Frobenius do seu produto externo é $\|\mathbf{u}_i \mathbf{v}_i^T\|_F = 1$.

Lembrando que $\|A\|_F$ é o equivalente a colocar todos os elementos de A em um único vetor \mathbf{w} e calcular $\|\mathbf{w}\|_2$. Assim, a norma de Frobenius transmite uma ideia da magnitude dos elementos armazenados na matriz.

Logo, a contribuição da parcela $\sigma_i \mathbf{u}_i \mathbf{v}_i^T$ na matriz final \tilde{A} tende a ser menor quanto menor for σ_i . Portanto, eliminamos as parcelas da eq. (0.3) em ordem crescente de σ_i .

Ao fazer isso, encontramos uma matriz \tilde{A} aproximada tal que $\|A - \tilde{A}\|_F \leq \|A - B\|_F$, sendo B uma matriz de dimensões $m \times n$, assim como A . Isso quer dizer que \tilde{A} é a matriz mais próxima de A se usarmos a norma de Frobenius como medida de distância [29].

Exemplo:

Decomposição da matriz

$$A = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 & 9 \\ 10 & 11 & 12 & 13 & 14 \\ 15 & 16 & 17 & 18 & 19 \\ 20 & 21 & 22 & 23 & 24 \end{bmatrix}.$$

Ao realizarmos a decomposição SVD de A encontramos uma matriz de valores singulares

$$\Sigma = \begin{bmatrix} 69,91 & 0 & 0 & 0 & 0 \\ 0 & 3,58 & 0 & 0 & 0 \\ 0 & 0 & 6,38 \times 10^{-15} & 0 & 0 \\ 0 & 0 & 0 & 1,24 \times 10^{-15} & 0 \\ 0 & 0 & 0 & 0 & 2,39 \times 10^{-16} \end{bmatrix}.$$

A matriz aproximada \tilde{A}_2 de posto 2 é

$$\tilde{A}_2 = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 & 9 \\ 10 & 11 & 12 & 13 & 14 \\ 15 & 16 & 17 & 18 & 19 \\ 20 & 21 & 22 & 23 & 24 \end{bmatrix} + \epsilon$$

$$\epsilon = \begin{bmatrix} -0,2 & -5,4 & -9,1 & -9,8 & -5,3 \\ 0,9 & -4 & -5 & -6 & -2 \\ -2 & -7 & -7 & -9 & -5 \\ -4 & -9 & -10,7 & -10,7 & -7,1 \\ -3,6 & -10,7 & -7,1 & -14,2 & -14,2 \end{bmatrix} \times 10^{-15}.$$

Assim, em vez de ter que armazenar os 25 valores da matriz A original, bastaria armazenar os 2 valores singulares de \tilde{A} mais 20 valores das linhas e colunas de U e V^T correspondentes, totalizando 22 valores. O ganho parece pouco nesse exemplo, mas em aplicações em que a matriz A é muito grande, ao guardarmos apenas alguns valores singulares, temos um ganho muito significativo.

OTIMIZAÇÃO E DUALIDADE

Um problema de otimização (não necessariamente convexo) na forma padrão tem a seguinte forma:

$$\begin{array}{ll} \text{minimizar} & f_0(\mathbf{x}) \\ \text{sujeito a} & f_i(\mathbf{x}) \leq 0, \quad i = 1, \dots, m \\ & h_i(\mathbf{x}) = 0, \quad i = 1, \dots, p \end{array} \quad (0.1)$$

em que $f_0(\mathbf{x})$ é chamada de **função objetivo** e as funções $f_i(\mathbf{x})$ e $h_i(\mathbf{x})$ são chamadas de **funções de restrição**. Denotamos o valor ótimo desse problema p^* . O valor \mathbf{x}^* , valor de \mathbf{x} que leva a p^* , é chamado de **ótimo primal**. Além disso, todo $\tilde{\mathbf{x}}$ que pertence ao domínio \mathcal{D} de f_0 e respeita as restrições do problema (0.1) é chamado de **ponto viável**. Por fim, muitas vezes chamamos o problema (0.1) de **problema primal** (essa nomenclatura fará sentido mais para frente).

A ideia básica da dualidade de Lagrange é levar em conta as restrições em uma única equação estendendo a expressão da função objetivo [31]. Essa expressão estendida é chamada de **Lagrangiano** (ou função de Lagrange) e é dada por:

$$L(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{v}) = f_0(\mathbf{x}) + \sum_{i=1}^m \lambda_i f_i(\mathbf{x}) + \sum_{i=1}^p v_i h_i(\mathbf{x}). \quad (0.2)$$

Os termos λ_i e v_i são chamados de **multiplicadores de Lagrange**. Os primeiros associados às restrições de desigualdade e os segundos às restrições de igualdade. Observar que, quando $\lambda_i \geq 0$, estamos premiando os valores de \mathbf{x} que tornam f_i muito negativa, pois, quanto mais negativa f_i menos vale $L(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{v})$.

Dando prosseguimento, ao tentar minimizar $L(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{v})$, encontramos a **função de Lagrange dual**, que é definida como o mínimo do Lagrangiano com relação a \mathbf{x} . Ela é dada por

$$g(\boldsymbol{\lambda}, \mathbf{v}) = \inf_{\mathbf{x} \in \mathcal{D}} L(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{v}) = \inf_{\mathbf{x} \in \mathcal{D}} \left(f_0(\mathbf{x}) + \sum_{i=1}^m \lambda_i f_i(\mathbf{x}) + \sum_{i=1}^p v_i h_i(\mathbf{x}) \right). \quad (0.3)$$

A função de Lagrange dual é côncava, mesmo que o problema definido em (0.1) não seja (chegar as condições necessárias para a convexidade de um problema de otimização em [31, p. 136]).

Para entender a afirmação acima, precisamos reparar que estamos falando da concavidade de $g(\boldsymbol{\lambda}, \mathbf{v})$ com relação a $(\boldsymbol{\lambda}, \mathbf{v})$, e não a \mathbf{x} . Levando isso em conta, podemos escrever o Lagrangiano como $L(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{v}) = v_{\mathbf{x}}(\boldsymbol{\lambda}, \mathbf{v})$, colocando \mathbf{x} como um parâmetro e $(\boldsymbol{\lambda}, \mathbf{v})$ como as únicas variáveis. Nesse caso, ao fixarmos um \mathbf{x} , podemos dizer que $v_{\mathbf{x}}(\boldsymbol{\lambda}, \mathbf{v})$ é uma soma de funções afins, pois com $f_0(\mathbf{x})$, $f_i(\mathbf{x})$ e $h_i(\mathbf{x})$ tornam-se constantes. Com isso, podemos dizer que $\{v_{\mathbf{x}}(\boldsymbol{\lambda}, \mathbf{v}) : \mathbf{x} \in \mathcal{D}\}$ é uma família de funções afins. Logo, $g(\boldsymbol{\lambda}, \mathbf{v}) = \inf_{\mathbf{x} \in \mathcal{D}} \{v_{\mathbf{x}}(\boldsymbol{\lambda}, \mathbf{v}) : \mathbf{x} \in \mathcal{D}\}$, que é uma função côncava, pois para cada valor de \mathbf{x} , $v_{\mathbf{x}}(\boldsymbol{\lambda}, \mathbf{v})$ é côncava [31, pp. 80, 87].

A função de Lagrange dual representa um limite inferior para o valor ótimo, ou seja, $g(\boldsymbol{\lambda}, \mathbf{v}) \leq p^*$. Isso porque, para um ponto viável $\tilde{\mathbf{x}}$ e para $\boldsymbol{\lambda} \geq 0$ (\geq é igual a \geq elemento a elemento),

$L(\tilde{x}, \lambda, \nu) = f_0(\tilde{x}) + \overbrace{\sum_{i=1}^m \lambda_i f_i(\tilde{x}) + \sum_{i=1}^p \nu_i h_i(\tilde{x})}^{<0} \leq f_0(\tilde{x})$, já que $f_i(\tilde{x}) < 0$ e $h_i(\tilde{x}) = 0$. Logo, $g(\lambda, \nu) = \inf_{x \in \mathcal{D}} L(x, \lambda, \nu) \leq L(x, \lambda, \nu) \leq f_0(x)$. Assim, para cada par (λ, ν) com $\lambda \geq 0$, $g(\lambda, \nu)$ representa um limite inferior diferente para p^* .

Poderíamos nos perguntar qual é o melhor limite inferior que poderíamos encontrar a partir de $g(\lambda, \nu)$. Essa pergunta dá origem ao **problema dual de Lagrange** que é definido como:

$$\begin{aligned} & \text{maximizar } g(\lambda, \nu) \\ & \text{sujeito a } \lambda \geq 0 \end{aligned} \tag{0.4}$$

Nos referimos a (λ^*, ν^*) como **ótimo dual** e denotamos seu valor ótimo correspondente d^* . Reparar que o problema (0.4) é convexo, pois a função objetivo é côncava e a restrição é convexa.

Podemos afirmar que $d^* \leq p^*$, já que $g(\lambda, \nu)$ é um limite inferior para p^* . Essa propriedade é chamada de **dualidade fraca**. Quando temos uma igualdade, ou seja, $d^* \leq p^*$, falamos em **dualidade forte**. Além disso, chamamos de **gap de dualidade** a diferença $p^* - d^*$. Em geral, a dualidade forte não é válida. No entanto, quando o problema primal é convexo, normalmente temos dualidade forte [31, p. 226].

Quando a dualidade forte é válida, podemos escrever

$$\begin{aligned} f_0(x^*) &= g(\lambda^*, \nu^*) \\ &= \inf_{x \in \mathcal{D}} \left(f_0(x) + \sum_{i=1}^m \lambda_i^* f_i(x) + \sum_{i=1}^p \nu_i^* h_i(x) \right) \\ &\leq f_0(x^*) + \sum_{i=1}^m \lambda_i^* f_i(x^*) + \sum_{i=1}^p \nu_i^* h_i(x^*) \\ &\leq f_0(x^*) \end{aligned}$$

Com isso, podemos concluir que, na verdade, as desigualdades são igualdades. Logo, quando o gap de dualidade é zero, temos $\sum_{i=1}^m \lambda_i^* f_i(x^*) = 0$. Mas, como todos os termos dessa soma são não-positivos, temos que

$$\lambda_i^* f_i(x^*) = 0.$$

Essa propriedade é chamada de **complementary slackness**. Além disso, podemos ver que x^* minimiza $L(x, \lambda^*, \nu^*)$ com relação a x .

Por fim, todo problema de otimização na forma (0.1) e com restrições e função objetivo diferenciáveis devem respeitar as chamadas **condições de KKT** (Karush-Kuhn-Tucker), que são as seguintes:

1. $f_i(x^*) \leq 0$
2. $h_i(x^*) = 0$
3. $\lambda_i^* \geq 0$
4. $\lambda_i^* f_i(x^*) = 0$
5. $(\nabla_x L)(x^*, \lambda^*, \nu^*) = 0 \Rightarrow (\nabla_x f_0)(x^*) + \sum_{i=1}^m \lambda_i^* (\nabla_x f_i)(x^*) + \sum_{i=1}^p \nu_i^* (\nabla_x h_i)(x^*) = 0$

As duas primeiras condições são as restrições definidas inicialmente em (0.1). A condição 3 é necessária para que a função de Lagrange dual seja um limite inferior para p^* , como visto anteriormente. A condição 4 vem da *complementary slackness*. A condição vem do fato de que $\inf_x L(\mathbf{x}, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^*) = L(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^*)$, como visto anteriormente.

Se o problema primal for convexo, a validade das condições de KKT já são suficientes para afirmar que existe dualidade forte.

Portanto, vemos que o problema de otimização apresentando em (0.1) se resume a $\max_{(\boldsymbol{\lambda}, \boldsymbol{\nu})} \inf_x L(\boldsymbol{\lambda}, \boldsymbol{\nu}, \mathbf{x})$ com $\boldsymbol{\lambda} \geq 0$ e, sob condição de gap de dualidade nulo, respeitando as condições de KKT.

REPRODUCING KERNEL HILBERT SPACES

Para entender o conceito de *Reproducing Kernel Hilbert Spaces* (RKHS), é necessário entender outros conceitos mais básicos, como o conceito de espaço de Hilbert.

Um espaço de Hilbert \mathbb{H} é um espaço com um produto interno $\langle \cdot, \cdot \rangle_{\mathbb{H}}$ que é completo com respeito à norma $\|\cdot\|_{\mathbb{H}}$ definida pelo produto interno. Um espaço completo é aquele em toda sequência de Cauchy converge para um membro desse espaço, sendo uma sequência de Cauchy é uma sequência cujos elementos se tornam arbitrariamente muito próximos uns dos outros à medida que a sequência progride [32].

O espaço de Hilbert é uma generalização do espaço Euclidiano para um espaço de dimensão finita ou infinita (o espaço Euclidiano tem dimensão finita) [32]. Aparentemente, um espaço de Hilbert finito é bem parecido com o espaço Euclidiano. Parece que as diferenças começam a surgir quando falamos de um espaço de Hilbert de dimensão infinita.

Por simplicidade, o produto interno e a norma do espaço de Hilbert serão escritos sem o subscrito \mathbb{H} .

Um **reproducing kernel Hilbert space** (RKHS) é um espaço de Hilbert \mathbb{H} de funções $f : \mathcal{X} \rightarrow \mathbb{R}$ com um *reproducing kernel* (ou **kernel**) $\kappa : \mathcal{X}^2 \mapsto \mathbb{R}$ em que $\kappa(x, \cdot) \in \mathbb{H}$ e $f(x) = \langle \kappa(x, \cdot), f \rangle$. Essa igualdade é chamada **reproducing property**.

Considere uma função kernel $\kappa(x, y)$ de duas variáveis. Suponha que, para n pontos, fixemos a variável y para ter $\kappa(x_1, y), \kappa(x_2, y), \dots, \kappa(x_n, y)$, que são todas funções da variável y . RKHS é um espaço de funções que é o conjunto de todas as possíveis combinações lineares dessas funções, ou seja,

$$\mathbb{H} := \left\{ f(\cdot) = \sum_{i=1}^n \alpha_i \kappa(x_i, \cdot) \right\} = \left\{ f(\cdot) = \sum_{i=1}^n \alpha_i \kappa_{x_i}(\cdot) \right\}.$$

Trazendo para algo concreto, poderíamos ter o kernel $\kappa(x, y) = xe^{2y}$. As funções $g(y) = 3e^{2y}$ e $h(y) = 5e^{2y}$ seriam exemplo de funções pertencentes ao RKHS $\mathbb{H} = \{f(y) = \sum_{i=1}^n \alpha_i x_i e^{2y}\}$.

Então, vemos que o kernel é a base para gerar o RKHS.

Dado um kernel, o RKHS é único e, dado um RKHS, o kernel correspondente é único [32].

A partir da *reproducing property*, se $f(\cdot) = \kappa(\cdot, x_1), x_1 \in \mathcal{X}$, então

$$\langle \kappa(\cdot, x_1), \kappa(\cdot, x_2) \rangle = \kappa(x_2, x_1) = \kappa(x_1, x_2), \quad (0.1)$$

ou seja, a ordem dos argumentos não importa.

Na primeira igualdade, temos o produto interno entre $f(\cdot)$ e $\kappa(\cdot, x_2)$, ou seja, uma função de um único parâmetro e outra com dois parâmetros, mas o segundo parâmetro está fixado em x_2 . Logo, o resultado é $f(x_2) = \kappa(x_2, x_1)$. A segunda igualdade vem do fato que $\langle \kappa(\cdot, x_1), \kappa(\cdot, x_2) \rangle = \overline{\langle \kappa(\cdot, x_2), \kappa(\cdot, x_1) \rangle} = \langle \kappa(\cdot, x_2), \kappa(\cdot, x_1) \rangle = \kappa(x_2, x_1)$, em que $\overline{\langle \kappa(\cdot, x_2), \kappa(\cdot, x_1) \rangle} = \langle \kappa(\cdot, x_2), \kappa(\cdot, x_1) \rangle$ vem do fato de esse espaço de Hilbert ser composto apenas de funções reais.

Seja \mathbb{H} um RKHS, associado a uma função kernel $\kappa(\cdot, \cdot)$ e com \mathcal{X} sendo um conjunto de elementos. Então, o mapeamento

$$\begin{aligned}\phi: \mathcal{X} &\rightarrow \mathbb{H} \\ x &\mapsto \phi(x)\end{aligned}$$

é chamado de **feature map** e o espaço \mathbb{H} é chamado de **espaço das features**. \mathcal{X} é chamado de **espaço das entradas** (*input space*).

O mapa das *features*, que pode ser finito ou infinito, é um vetor $\phi(x) = [\phi_1(x), \phi_2(x), \dots]$. Um exemplo de *feature map* finito de dimensão 3 é $\phi(x) = [\phi_1(x), \phi_2(x), \phi_3(x)] = [x_1^2, x_1 x_2 \sqrt{2}, x_2^2]$, com $x = [x_1, x_2]^T$. Nesse espaço, temos apenas três *features*: x_1^2 , $x_1 x_2 \sqrt{2}$ e x_2^2 .

O seguinte produto interno é chamado de **kernel trick**:

$$\langle \phi(x), \phi(y) \rangle = \kappa(x, y).$$

A prova para isso pode ser encontrada em [32].

A partir da eq. (0.1), vemos que $\phi(x) = \kappa(x, \cdot) = \kappa(\cdot, x)$. Com isso, fica claro que o *feature map* é uma função $\kappa(x, \cdot)$ em que fixamos um dos argumentos em x . Logo, o *feature map* leva os dados $x \in \mathbb{R}^p$ do espaço das entradas para o espaço das *features* \mathbb{H} composto por funções, que costuma ter um número de dimensões muito mais elevado, possivelmente infinito.

No exemplo do espaço finito apresentado acima, o kernel é dado por $\kappa(x, y) = \langle \phi(x), \phi(y) \rangle = \phi^T(x) \phi(y) = (x^T y)^2$, com $y = [y_1, y_2]^T$. No caso específico desse exemplo, vemos que o *features map* $\phi(x_n) = \kappa(x_n, z)$ avaliado no ponto x_n não é função da variável z , apesar de isso não ser sempre verdade (aparentemente).

Com isso, vemos que, podemos realizar produtos internos de maneira bastante eficiente quando empregamos esse mapeamento, pois basta avaliar o kernel $\kappa(\cdot, \cdot)$ nos pontos x e y .

Abaixo, são listados alguns kernel mais conhecidos.

Kernel linear

$$\kappa(x, y) := x^T y$$

Nesse tipo de kernel, conhecemos o *feature map*, que é $\phi(x) = x$.

Kernel Gaussiano ou Radial Basis Function (RBF)

$$\kappa(x, y) := e^{-\gamma \|x-y\|_2^2} = e^{-\frac{\|x-y\|_2^2}{\sigma^2}}$$

Um valor típico para γ é $1/d$, em que d é a dimensionalidade dos dados.

Kernel polinomial

$$\kappa(x, y) := (\gamma x^T y + c)^d$$

Com $\gamma > 0$ sendo o coeficiente angular, c o coeficiente linear e d a dimensionalidade dos dados. Valores típicos para os parâmetros são $\gamma = 1/d$ e $c = 1$.

Capítulo 15: REFERÊNCIAS

- [1] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189-1232, 2001.
- [2] L. Wasserman, *All of Statistics*, Pittsburgh: Springer, 2003.
- [3] J. K. Blitzstein e J. Hwang, *Introduction to Probability*, 2 ed., Boca Raton: CRC Press, 2019.
- [4] "1.6 Nearest Neighbors," scikit-learn, [Online]. Available: <https://scikit-learn.org/stable/modules/neighbors.html#classification>. [Acesso em 12 06 2024].
- [5] S. Theodoridis, *Machine Learning: a bayesian and optimization perspective*, 2nd ed., London: Elsevier, 2020.
- [6] N. El Gayar, F. Schwenker e G. Palm, "A Study of the Robustness of KNN Classifiers Trained Using Soft Labels," *Artificial Neural Networks in Pattern Recognition: Second IAPR Workshop, ANNPR 2006*, pp. 67-80, 2006.
- [7] T. Hastie, R. Tibshirani e J. Friedman, *The Elements of Statistical Learning*, 2nd ed., New York: Springer, 2009.
- [8] J. L. Bentley, "Multidimensional Binary Search Trees Used fo Associative Searching," *Communications of the ACM*, vol. 18, nº 9, pp. 509-517, 01 09 1975.
- [9] K. Weinberger, "16: KD Trees," Cornell University, [Online]. Available: <http://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote16.html>. [Acesso em 15 06 2024].
- [10] "KDTree," Scikit-learn, [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KDTree.html>. [Acesso em 16 06 2024].
- [11] H. Hristov, "Introduction to K-D Trees," Baeldung, 2023 03 29. [Online]. Available: <https://www.baeldung.com/cs/k-d-trees>. [Acesso em 15 06 2024].
- [12] S. M. Omohundro, *Five balltree construction algorithms*, Berkeley: International Computer Science Institute, 1989.
- [13] J. Adamczyk, "k nearest neighbors computational complexity," Medium, 06 08 2020. [Online]. Available: <https://towardsdatascience.com/k-nearest-neighbors-computational-complexity-502d2c440d5>. [Acesso em 20 06 2024].
- [14] K. Weinberger, "KD Trees," [Online]. Available: <http://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote16.html>. [Acesso em 18 06 2024].

- [15] K. Weinberger, "Lecture 2: k-nearest neighbors," Cornell University, [Online]. Available: https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote02_kNN.html. [Acesso em 20 06 2024].
- [16] C. M. Bishop, *Pattern Recognition and Machine Learning*, New York: Springer, 2006.
- [17] A. Géron, *Hands-on Machine Learning with Scikit-learn, Keras & Tensorflow: Concepts, Tools and Techniques to Build Intelligent Systems*, Sebastopol: O'Reilly, 2019.
- [18] J. H. Friedman, "Greedy function approximation: a gradient boosting machine.," *Annals of statistics*, pp. 1189-1232, 2001.
- [19] M. N. Bernstein, "Functionals and functional derivatives," 10 04 2022. [Online]. Available: <https://mbernste.github.io/posts/functionals/>. [Acesso em 26 06 2024].
- [20] L. Mason, J. Baxter, P. Bartlett e M. Frean, "Boosting algorithms as gradient descent," *Advances in neural information processing systems*, vol. 12, 1999.
- [21] J. Li, K. Cheng, S. Wang, F. Morsttater, R. P. Trevino, J. Tang e H. Liu, "Feature Selection: A Data Perspective," *ACM Computing Surveys*, vol. 50, 2017.
- [22] M. Cabral e P. Goldfeld, *Curso de Álgebra Linear: Fundamentos e Aplicações*, 3 ed., Rio de Janeiro: Instituto de Matemática, 2012.
- [23] T. Roughgarden e G. Valiant, "Lecture #7: Understanding and using Principal Component Analysis (PCA)," 2023.
- [24] "Covariance Matrix," Wikipédia, [Online]. Available: https://en.wikipedia.org/wiki/Covariance_matrix. [Acesso em 14 10 2023].
- [25] N. Japkowicz e M. Shah, *Evaluating Learning Algorithms: A Classification Perspective*, Cambridge University Press, 2011.
- [26] M. Grandini, E. Bagli e G. Visani, "Metrics for Multi-Class Classification: An Overview," *arXiv preprint arXiv:2008.05756*, 2020.
- [27] S. Chatterjee e A. S. Hadi, *Regression Analysis by Example*, 5 ed., Cairo: Wiley, 2012.
- [28] T. O. Kvalseth, "Note on the R2 measure of goodness of fit for nonlinear models," *Bulletin of the Psychonomic Society*, vol. 21, nº 1, pp. 79-80, 1983.
- [29] T. Roughgarden e G. Valiant, "The Singular Value Decomposition (SVD) and Low-Rank Matrix Approximations," 2023.
- [30] G. Strang, *Linear Algebra and Its Applications*, 4 ed., Cengage Learning, 2005.
- [31] S. Boyd e L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2004.
- [32] B. Ghoggh, A. Ghodsi, F. Karray e M. Crowley, "Reproducing Kernel Hilbert Space, Mercer's Theorem, Eigenfunctions, Nyström Method, and Use of Kernels in Machine Learning: Tutorial and Survey," *arXiv preprint arXiv:2106.08443*, 2021.

