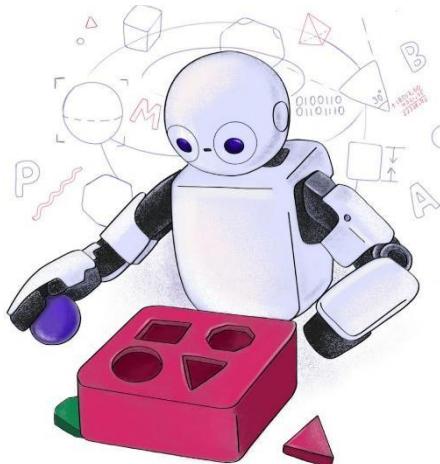


TP555 - Inteligência Artificial e Machine Learning: *Classificação*



Esse material foi desenvolvido e gentilmente cedido pelo Prof. Dr. Felipe Augusto Pereira de Figueiredo, do Inatel. (felipe.figueiredo@inatel.br)

Inatel

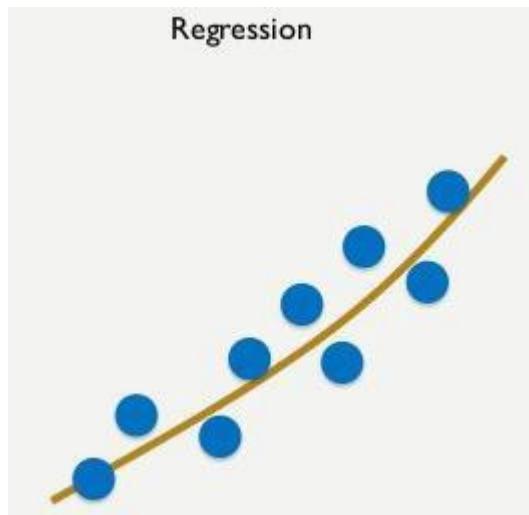
Prof. Dr. Luiz Augusto Melo Pereira
luiz.melo@inatel.br

Tópicos abordados

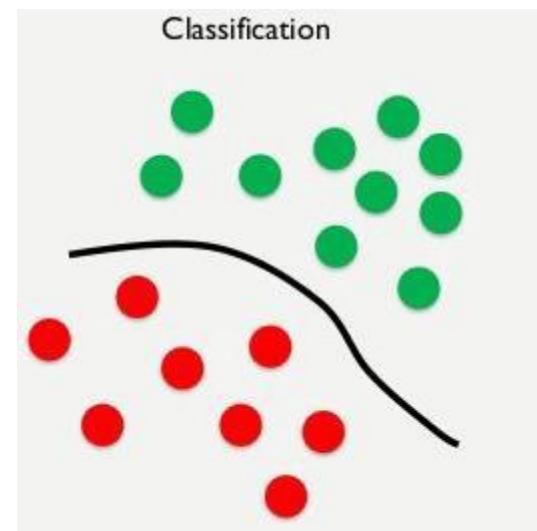
- Abordagens para classificação que veremos:
 - Classificação Bayesiana
 - Regressão logística
 - Regressão Softmax
- Métricas para avaliação de classificadores.

Classificação

- Tarefa (ou problema) de **aprendizado supervisionado**.
 - As saídas esperadas (rótulos) são conhecidas.
- Envolve encontrar uma função, $f(x)$, que **mapeie** os atributos de entrada em um **conjunto finito de valores discretos**, ou seja, em classes.



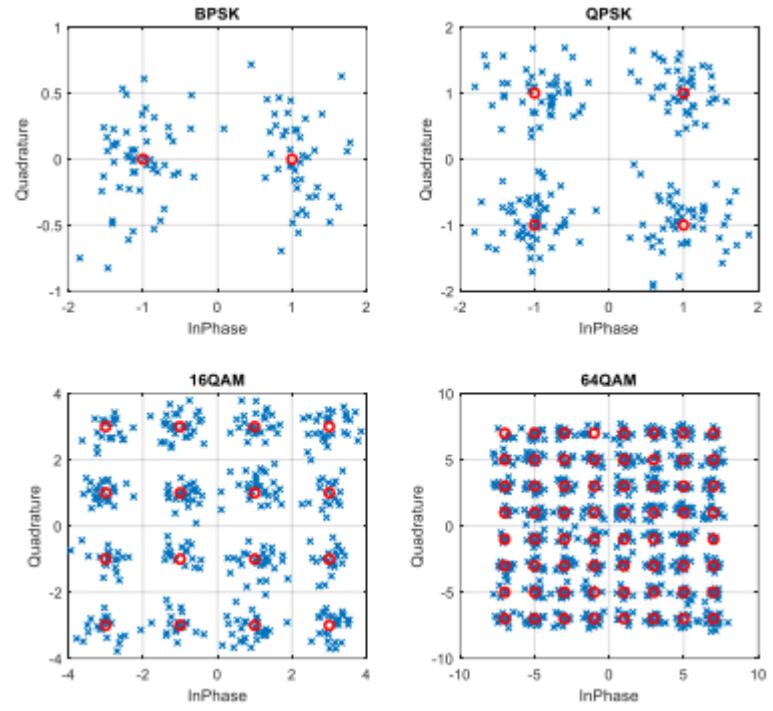
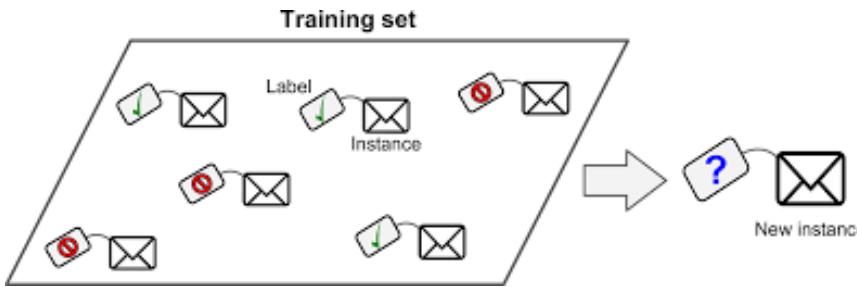
$f(x)$ **aproxima** o comportamento dos dados.



$f(x)$ **classifica** os dados.

$f(x)$ forma uma **fronteira de decisão**.

Tarefas de classificação



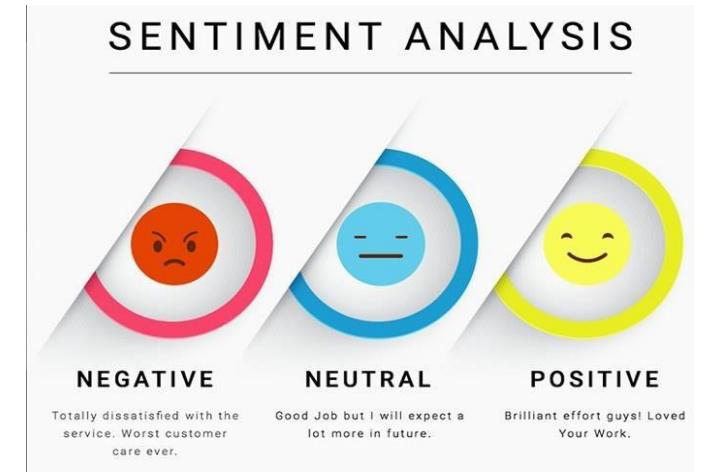
- Classificação de emails entre *spam* e *ham* (legítimo).
- Classificação de objetos em imagens ou vídeos.
- Detecção ou classificação de símbolos de modulações digitais.
- Classificação de modulações (QPSK, AM, FM, etc.).

Tarefas de classificação

0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9



- Reconhecimento de caracteres.
- Classificação de texto (e.g., notícias).
- Classificação de sentimentos.
- Classificação de doenças (e.g., pulmonares).



Tarefas de classificação



- Análise de crédito para diferenciar entre clientes de baixo e alto risco.
- Detecção de fraudes bancárias (e.g., fraudes com cartão de crédito).

Definição do problema de classificação

- **Problema:** encontrar uma função, $f(\mathbf{x})$, que atribua a um *exemplo de entrada*, \mathbf{x} , uma de Q classes possíveis, as quais denotaremos como $C_q, q = 1, \dots, Q$.
 - Por exemplo, as classes podem ser
 - *Spam* e *ham* (legítimo): $Q = 2$.
 - Dígitos de 0 a 9: $Q = 10$.
 - Símbolos de uma modulação específica (e.g., QPSK: $Q = 4$).
 - Objetos (carros, barcos, cães, gatos, etc.)
- Semelhante ao problema da *regressão linear*, existe um conjunto de treinamento com N pares de *vetores de atributos* e *rótulos* $\{\mathbf{x}(i); y(i)\}_{i=0}^{N-1}$ que é utilizado para treinar um *classificador*, onde
 - $\mathbf{x}(i) = [x_1(i) \quad \dots \quad x_K(i)]^T \in \mathbb{R}^{K \times 1}$ representa o i -ésimo vetor de atributos, o qual é composto por K atributos, $x_1(i), \dots, x_K(i)$;
 - e $y(i)$ representa o i -ésimo *rótulo*.

Representação da saída desejada

- **Classificação binária** ($Q = 2$): existem apenas duas classes possíveis, C_1 e C_2 , onde C_1 é chamada de **classe negativa** e C_2 de **classe positiva**.
- Portanto, nesse caso, o classificador possui **uma única saída escalar binária** para indicar a **classe** correspondente ao **vetor de atributos**:

$$y(i) = \begin{cases} 0, & x(i) \in C_1 \\ 1, & x(i) \in C_2 \end{cases}$$

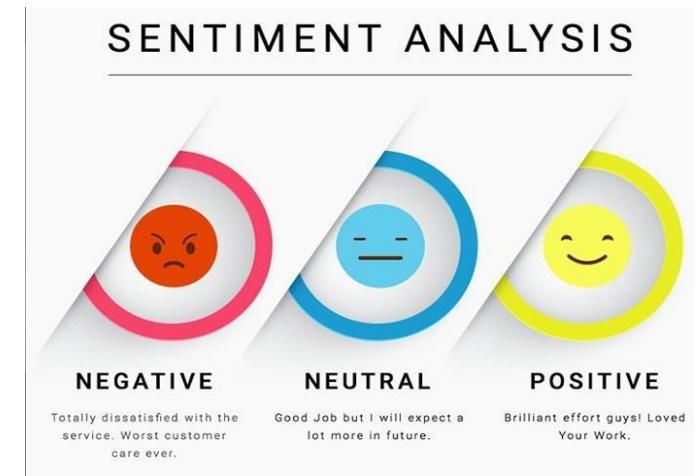
- Assim, $y(i) \in \mathbb{R}^1$, de maneira que **o classificador realiza um mapeamento** $\mathbb{R}^{K \times 1} \rightarrow \mathbb{R}^1$, ou seja, $y = f(x)$, onde $x \in \mathbb{R}^{K \times 1}$ e $y \in \mathbb{R}^1$.
- Também é possível utilizar $y(i) = -1$ para $x(i) \in C_1$, ou seja

$$y(i) = \begin{cases} -1, & x(i) \in C_1 \\ 1, & x(i) \in C_2 \end{cases}$$

Representação da saída desejada



- **Classificação multi-classes:** existem mais de 2 classes possíveis ($Q > 2$).
- Uma estratégia bastante utilizada para representar estas classes é conhecida como ***codificação one-hot***.



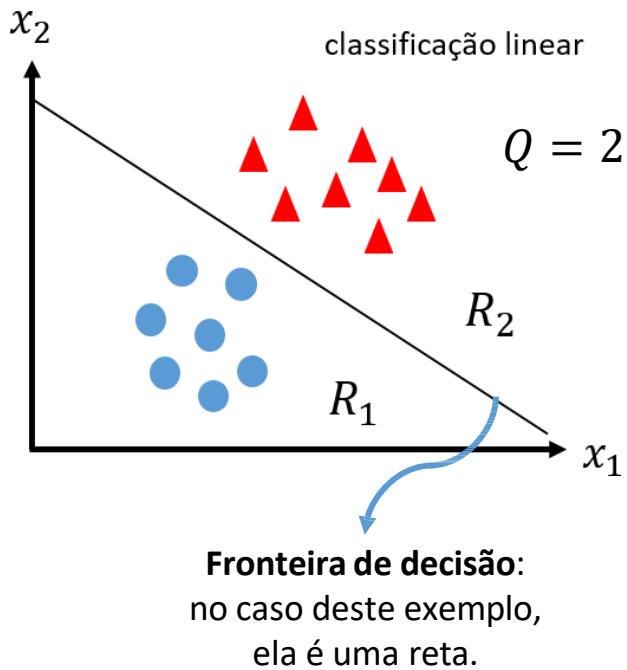
Representação da saída desejada

- **Codificação one-hot:** utiliza uma representação **vetorial binária** para as saídas.
 - Ou seja, as saídas são vetores com o valor 1 no elemento representando a classe do exemplo de entrada e 0 nos demais elementos.
 - Nesse caso, o **classificador possui múltiplas saídas** (Q saídas), cada uma representando uma classe específica.
 - **Exemplo:** imaginemos um **classificador de notícias** com quatro classes possíveis: *esportes, política, ciências* e *variedades*. Como seria a representação com a codificação **one-hot**?

$$\begin{array}{ll} \text{esportes: } & [1 \quad 0 \quad 0 \quad 0] \\ \text{política: } & [0 \quad 1 \quad 0 \quad 0] \\ \text{ciências: } & [0 \quad 0 \quad 1 \quad 0] \\ \text{variedades: } & [0 \quad 0 \quad 0 \quad 1] \end{array} \quad \left. \right\}$$

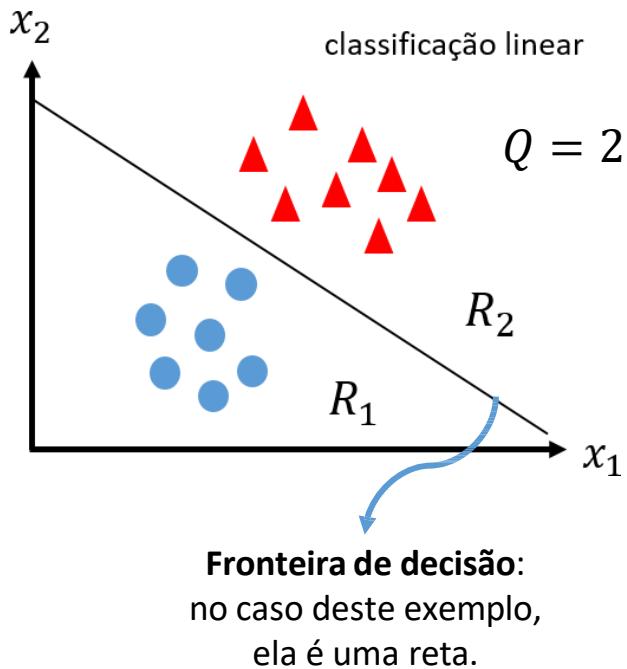
Assim, $y(i) \in \mathbb{R}^{Q \times 1}$, de maneira que o classificador realiza um mapeamento $\mathbb{R}^{K \times 1} \rightarrow \mathbb{R}^{Q \times 1}$

Fronteiras de decisão de um classificador



- Antes, nós usávamos **funções hipótese** para **aproximar o comportamento de um conjunto de dados**, agora, as usaremos para **separar grupos de dados (i.e., classes)**.
- Para facilitar o entendimento, vamos imaginar o **espaço bi-dimensional**, \mathbb{R}^2 , criado pelos **atributos** x_1 e x_2 , mostrado na figura ao lado.
- Os **pares de atributos** pertencem a duas classes ($Q = 2$):
 - Círculos azuis pertencem à classe C_1 .
 - Triângulos vermelhos pertencem à classe C_2 .

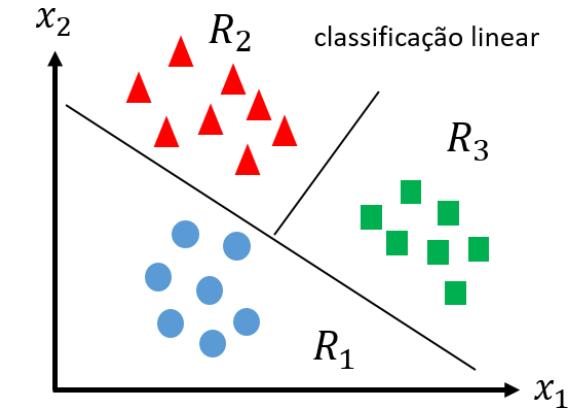
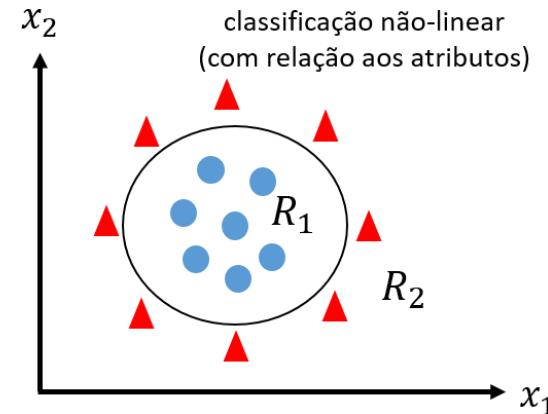
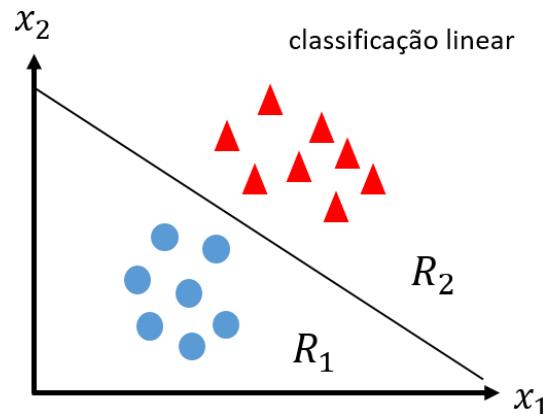
Fronteiras de decisão de um classificador



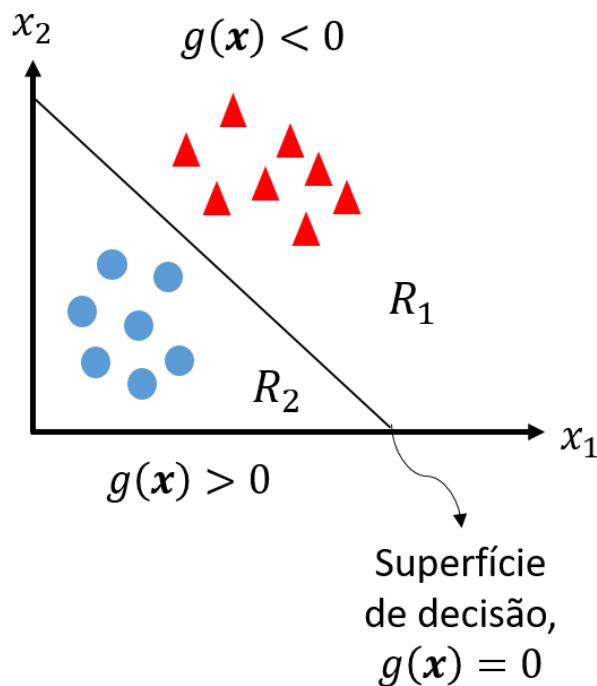
- Esse espaço pode ser dividido em **duas regiões de decisão**, R_1 e R_2 , onde cada **região** corresponde a uma classe.
- As regiões de decisão são separadas por **fronteiras de decisão**, que nada mais são do que **funções**.
- Na figura, como $Q = 2$, temos apenas uma fronteira de decisão.
- Uma **fronteira de decisão** corresponde a uma **superfície de separação** (1D, 2D, 3D, etc.) no **espaço de atributos** que separa as classes.

Fronteiras de decisão de um classificador

- As **superfícies de separação** podem ser **lineares** (e.g., retas e planos) ou **não-lineares** (e.g., círculos e elipses).
- As **superfícies de separação** são definidas por **funções** (lineares ou não) que separam as classes.
- Essas funções são normalmente chamadas de **funções discriminantes**, pois separam as classes.
- Abaixo temos **regiões de separação** para problemas de classificação **binária** e **multi-classes**.

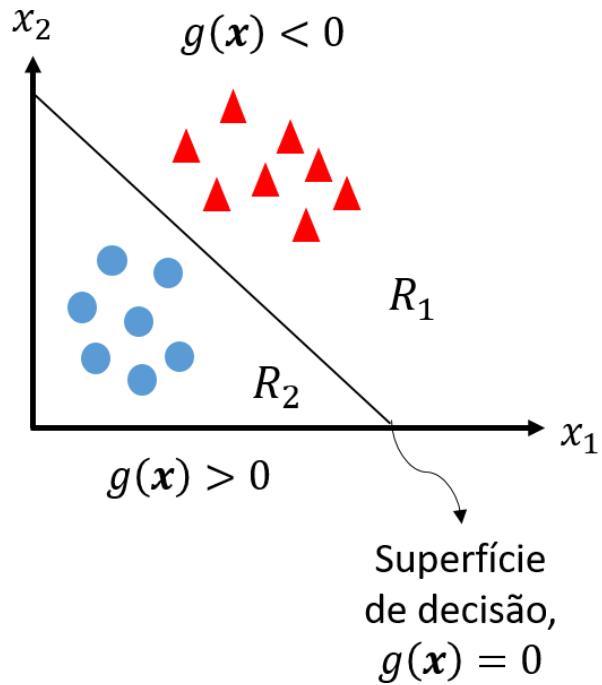


Funções discriminantes



- Uma **função discriminante linear** pode ser escrita da seguinte forma
$$g(\mathbf{x}) = a_0 + a_1x_1 + a_2x_2 + \cdots + a_Kx_K = \mathbf{a}^T\mathbf{x},$$
que nada mais é do que uma **combinacão linear dos atributos em relação aos pesos**, assim como nós vimos em regressão linear.
- $g(\mathbf{x})$ pode ser interpretada como um **hiperplano** que separa as classes.
- Um **hiperplano** pode ser 1 ponto em 1D, uma reta em 2D, um plano em 3D, etc.
 - O coeficiente a_0 (**bias**) dá o deslocamento com relação à origem.
 - E o restante dos pesos determina a orientação do **hiperplano**.

Funções discriminantes

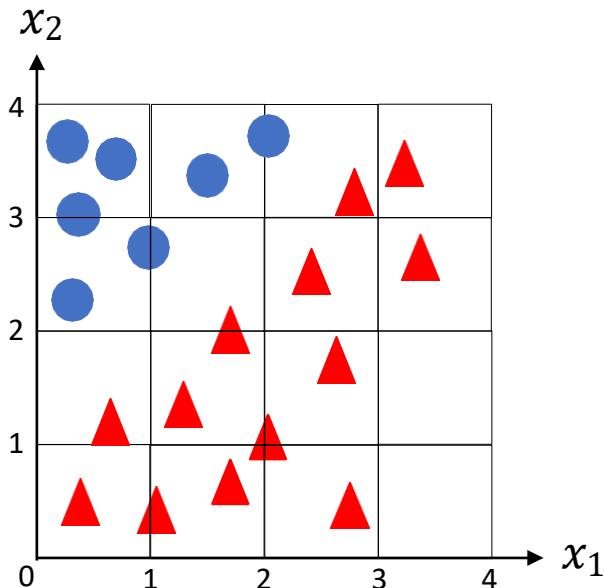


- Nosso **objetivo é encontrar os pesos da função discriminante** de tal forma que que a classe atribuída a um exemplo de entrada seja:

$$C_q = \begin{cases} 1, \\ 2, \\ \text{uma ou outra} \end{cases}$$

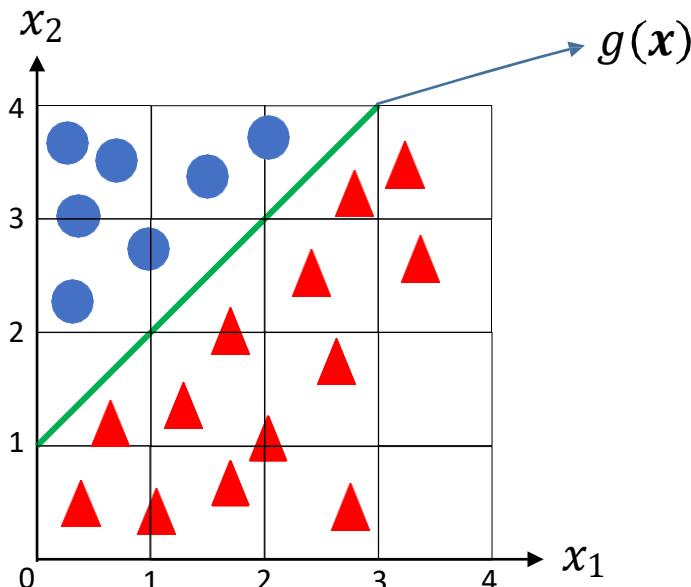
- OBS.: Podemos usar também *funções discriminantes não-lineares* **em relação aos atributos**, e.g., $g(x) = a_0 + x_1^2 + x_2^2$ (eq. de um círculo centrado na origem, onde $a_0 = -r^2$).

Exemplo: Encontrando os pesos da função discriminante, $g(x)$, visualmente



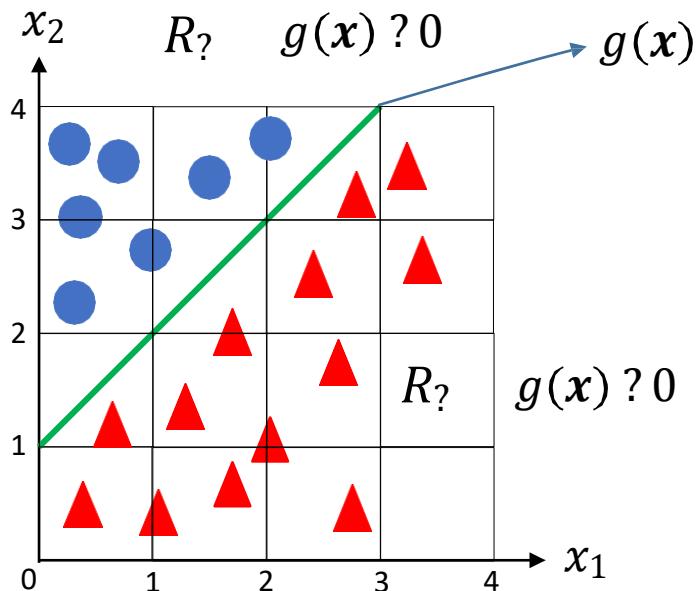
- Analisem a figura ao lado.
- Temos 2 classes, 2 atributos, x_1 e x_2 , e queremos encontrar uma **função discriminante**, $g(x)$, que as separe.
- Qual formato deve ter esta **função discriminante** para que ela tenha boa capacidade de generalização?
 - Lembrem-se do princípio da navalha de Occam: *a explicação mais simples (i.e., menos complexa) é geralmente a mais provável de estar correta.*

Exemplo: Encontrando os pesos da função discriminante, $g(x)$, visualmente



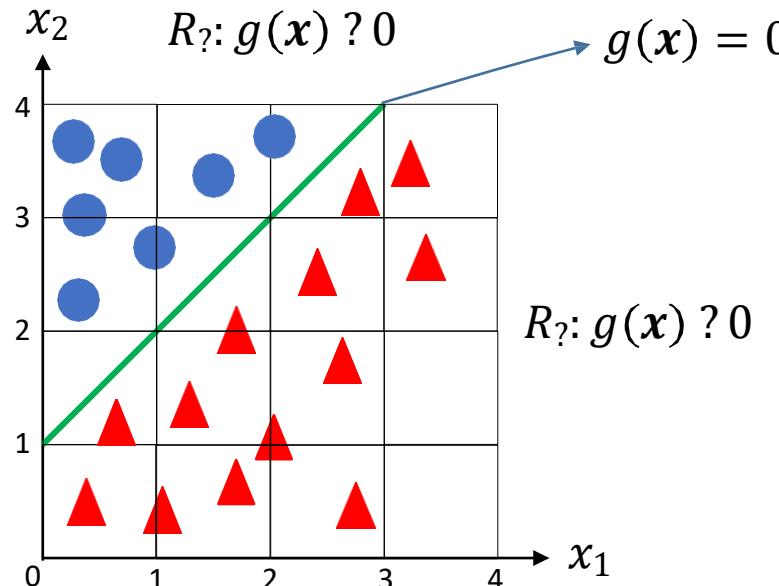
- Qual formato deve ter esta **função discriminante** para que ela tenha boa capacidade de generalização?
 - O formato mais simples, seguindo o princípio da navalha de Occam, é o de uma **reta** traçada no plano formado por x_1 e x_2 .

Exemplo: Encontrando os pesos da função discriminante, $g(x)$, visualmente



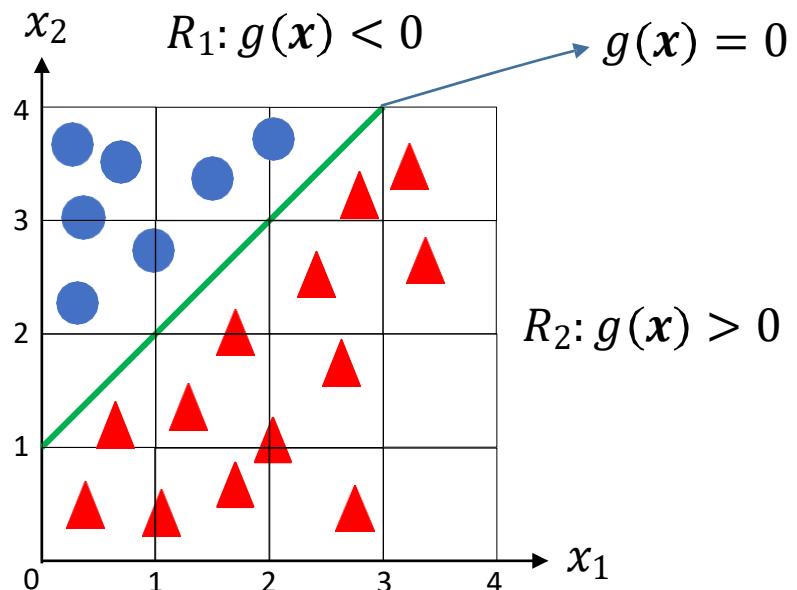
- **Visualmente**, nós traçamos a reta em uma posição que separe as classes da melhor forma possível.
- A **função discriminante** que representa esta reta no **espaço** criado por x_1 e x_2 é dada pelo **plano**
$$g(x) = a_0 + a_1x_1 + a_2x_2$$
- Agora que definimos o formato da função e sua posição no gráfico, precisamos encontrar os **pesos** e, com isso, definir as **regiões de decisão**.
- Como podemos encontrar os pesos?

Exemplo: Encontrando os pesos da função discriminante, $g(\mathbf{x})$, visualmente



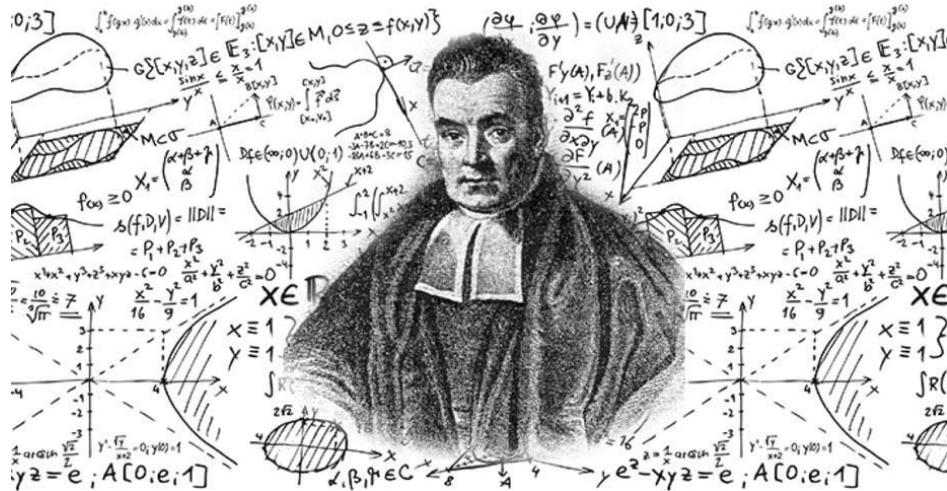
- Se temos 3 incógnitas, precisamos de um sistema com 3 equações:
 - $(x_1 = 0, x_2 = 1) \rightarrow 0 = a_0 + a_2 \therefore a_0 = -a_2$
 - $(x_1 = 1, x_2 = 2) \rightarrow 0 = a_0 + a_1 + 2a_2 \therefore a_1 = -(a_0 + 2a_2)$
 - $(x_1 = 2, x_2 = 3) \rightarrow 0 = a_0 + 2a_1 + 3a_2 \therefore a_1 = -(a_0 + 3a_2)/2$
- Resolvendo o sistema, encontramos $a_0 = 1$, $a_1 = 1$, $a_2 = -1$, então
 - $g(\mathbf{x}) = 1 + x_1 - x_2$

Exemplo: Encontrando os pesos da função discriminante, $g(x)$, visualmente



- Agora, vamos definir as **regiões de decisão** substituindo alguns valores em $g(x) = 1 + x_1 - x_2$.
 - $x_1 = 1$ e $x_2 = 1$ resulta em $g(x) > 0$.
 - ✓ Região da classe **positiva**, C_2 .
 - $x_1 = 1$ e $x_2 = 3$ resulta em $g(x) < 0$.
 - ✓ Região da classe **negativa**, C_1 .
 - $x_1 = 1$ e $x_2 = 2$ resulta em $g(x) = 0$.
 - ✓ **Indeterminação**: não podemos afirmar a qual classe o exemplo pertence.
 - ✓ Podemos atribuir arbitrariamente a uma das duas classes ou escolher a classe que possui maior número de exemplos.
- O classificador pode ser implementado como uma estrutura de controle de fluxo.

Teoria Bayesiana de decisão



- A teoria Bayesiana de decisão é uma **abordagem estatística** para o problema de **classificação**.
 - Ela explora o conhecimento
 - das **probabilidades das classes**,
 - dos **atributos** e
 - dos **custos associados a cada decisão**,
- para realizar a classificação dos exemplos de entrada.

Teoria Bayesiana de decisão



- Vamos considerar que um exemplo a ser classificado seja descrito pelo vetor de atributos $x \in \mathbb{R}^{K \times 1}$.
- Cada exemplo pertence a uma, e somente uma, classe C_q , sendo que existem ao todo Q classes possíveis.
 - $P(C_q)$ denota a **probabilidade a priori** associada à classe C_q .
 - Em outras palavras, $P(C_q)$ indica a **probabilidade de um exemplo arbitrário** (e desconhecido) **pertencer à classe C_q** .
- Agora vamos supor que um exemplo x seja observado.

Teoria Bayesiana de decisão



- De posse dos atributos deste exemplo, qual deve ser a decisão quanto à classe a que ele pertence?
 - Uma opção intuitiva e bastante poderosa é **escolher a classe que se mostre a mais provável tendo em vista os atributos do exemplo, x** .
 - Ou seja, a decisão é tomada em favor da classe cuja probabilidade ***a posteriori*** (i.e., probabilidade após a observação de x) seja máxima.
 - A probabilidade ***a posteriori*** corresponde à **probabilidade condicional**

$$P(C_q|x).$$

Como calculamos $P(C_q | x)$?

Teoria Bayesiana de decisão

Teorema de Bayes

$$P(C_q|x) = \frac{P(x|C_q)P(C_q)}{P(x)}, \forall q,$$

onde o termo $P(x|C_q)$ é denominado de **verossimilhança (likelihood)** e o termo $P(x)$ é normalmente chamado de **evidência**.

Máxima probabilidade a posteriori (MAP)

- Esta forma de classificação é conhecida como o **critério da máxima probabilidade a posteriori** (MAP, do inglês **maximum a posteriori probability**).
- Assim, a decisão para o exemplo x é dada pela classe C_q que maximiza $P(C_i|x)$, i.e., em forma matemática

$$\text{MAP: } C_q = \arg \max_{C_i, i=1, \dots, Q} P(C_i|x).$$

- A solução da equação acima é equivalente a maximizar seu numerador, $P(x|C_i)P(C_i)$, de forma que

$$\text{MAP: } C_q = \arg \max_{C_i, i=1, \dots, Q} P(x|C_i)P(C_i),$$

já que o denominador $P(x)$ **não depende das classes testadas**, servindo apenas como fator de escala no critério.

Máxima verossimilhança (ML)

- O **decisor de máxima verossimilhança** (ML, do inglês **maximum likelihood**) parte do pressuposto de que **não há informação estatística sobre as classes**, i.e., sobre $P(C_i)$.
- Portanto, o critério ML toma a decisão em favor da **classe C_q que maximiza a probabilidade de verossimilhança**, $P(x|C_i)$
ML: $C_q = \arg \max_{C_i, i=1, \dots, Q} P(x|C_i).$
- **OBS.1:** a diferença entre os critérios MAP e ML está em o MAP incorporar o conhecimento das **probabilidades a priori**, i.e., $P(C_i)$.
- **OBS.2:** caso as **classes sejam equiprováveis**, i.e., $P(C_i) = 1/Q, \forall i$, então, maximizar a **probabilidade a posteriori** dá a mesma solução que o ML.

Exemplo: diagnóstico de doenças

- Vamos supor que estamos trabalhando no diagnóstico de uma nova doença e que fizemos testes em 100 indivíduos distintos.
- Após coletarmos os resultados, descobrimos que 20 deles *realmente* possuíam a doença (20%) e 80 estavam saudáveis (80%).
- Dos indivíduos que realmente possuíam a doença, 90% receberam positivo no teste da doença.
- 30% dos que não possuíam a doença também receberam positivo no teste.
- **Pergunta:** Se um novo indivíduo realizar o teste e receber um resultado positivo, qual a probabilidade dele realmente ter a doença?

Exemplo: diagnóstico de doenças (solução)

Informações que possuímos:

2 classes: possui doença e não possui doença

1 atributo: resultado do teste: + ou -

- Pergunta em forma probabilística: $P(\text{doença} | +)$, ou seja, probabilidade do indivíduo ter a doença dado que o resultado observado é positivo?

- Probabilidades:

$P(+ \text{doença}) = 0.9$	$P(+ \text{sem_doença}) = 0.3$
$P(\text{doença}) = 0.2$	$P(\text{sem_doença}) = 0.8$
$P(+) = P(+ \text{doença})P(\text{doença}) + P(+ \text{sem_doença})P(\text{sem_doença}) = 0.42$	

- Usando o teorema de Bayes

$$P(\text{doença}|+) = \frac{P(+|\text{doença})P(\text{doença})}{P(+)} = 0.429$$

$$P(\text{sem_doença}|+) = \frac{P(+|\text{sem_doença})P(\text{sem_doença})}{P(+)} = 0.571$$

A probabilidade dele não ter a doença mesmo tendo seu teste positivo é de aproximadamente 57%, ou seja, a probabilidade de **falsos positivos** é alta. Portanto, este não é um teste confiável.

Classificador naïve Bayes

- São classificadores que assumem que os **atributos** são **estatisticamente independentes** uns dos outros.
- Ou seja, a alteração do valor de um **atributo**, não influencia diretamente ou altera o valor de qualquer um dos outros atributos.
- Assim a probabilidade da classe C_q dado o vetor de atributos \mathbf{x} pode ser reescrita como

$$\begin{aligned} P(C_q | \mathbf{x} = [x_1 \quad \cdots \quad x_K]^T) \\ = \frac{P(\mathbf{x}|C_q)P(C_q)}{P(\mathbf{x})} \\ = \frac{P(x_1|C_q) \dots P(x_K|C_q)P(C_q)}{P(x_1) \dots P(x_K)}. \end{aligned}$$

Classificador naïve Bayes

- Com a independência dos atributos, os critérios MAP e ML são dados por
MAP: $C_q = \arg \max_{C_i, i=1, \dots, Q} P(x_1|C_i) \dots P(x_K|C_i)P(C_i),$
ML: $C_q = \arg \max_{C_i, i=1, \dots, Q} P(x_1|C_i) \dots P(x_K|C_i).$
- Aplicações típicas do classificador naïve Bayes incluem
 - filtragem de spam,
 - classificação de documento,
 - de sentimentos,
 - detecção de modulações digitais, etc.

Classificador naïve Bayes

- **Vantagens**
 - Fácil de ser **implementado** e altamente **escalável**.
 - Funciona bem mesmo com poucos dados.
 - **Classificação rápida** e, portanto, pode ser utilizado em aplicações de **tempo real**.
 - Além de simples, pode apresentar desempenho melhor do que métodos de classificação altamente sofisticados em algumas aplicações.
- **Desvantagens**
 - Assume que todos os atributos são independentes, o que muitas vezes não é verdade na prática.
 - Não consegue classificar caso uma das probabilidades condicionais (i.e., **verossimilhanças**) seja igual a zero, mas existem formas de se driblar esse problema (e.g., técnica da suavização de Laplace).
 - É necessário que se conheça ou assuma as probabilidades condicionais dos atributos.

Tipos de classificadores naïve Bayes

- Na prática, as probabilidades condicionais, i.e., as *verossimilhanças*, dos atributos x_k de uma classe, C_q , $P(x_k|C_q)$, $\forall k$, são geralmente modeladas usando-se o mesmo tipo de distribuição de probabilidade, como as distribuições Gaussiana, Multinomial e de Bernoulli.
- Portanto, tem-se três tipos diferentes de classificadores dependendo da suposição feita para a probabilidade condicional $P(x_k|C_q)$:
 - Classificador naïve Bayes Gaussiano
 - Classificador naïve Bayes Multinomial
 - Classificador naïve Bayes Bernoulli

Classificador naïve Bayes Gaussiano

- Até agora, vimos os cálculos de probabilidade quando os **atributos** são **categóricos**.
- Mas como calcular as probabilidades quando os **atributos** são variáveis **contínuas**?
- Quando lidamos com **atributos**, x_1, \dots, x_K , que apresentam **valores contínuos**, uma suposição típica é que os valores dos atributos sejam distribuídos de acordo com uma **distribuição normal** (ou **Gaussiana**).
- Para se encontrar os **parâmetros** do classificador faz-se o seguinte:
 - Primeiro, segmenta-se os atributos, x_1, \dots, x_K , de acordo com a classe a que pertencem;
 - Em seguida, calcula-se a média, μ_{x_k, C_q} , e a variância, σ_{x_k, C_q}^2 , de cada atributo x_k em relação à classe, C_q , a que pertence.

Classificador naïve Bayes Gaussiano

- Assim, a probabilidade condicional $P(x_k|C_q)$ pode ser calculada inserindo-se o valor de x_k na equação da **distribuição Normal parametrizada** com μ_{x_k,C_q} e σ_{x_k,C_q}^2 .

$$P(x_k|C_q) = \frac{1}{\sigma_{x_k,C_q}\sqrt{2\pi}} e^{-\frac{(x_k - \mu_{x_k,C_q})^2}{2\sigma_{x_k,C_q}^2}}.$$

- OBS.:**
 - Além da suposição de independência dos atributos, esta é outra **suposição forte**, pois muitos **atributos não seguem uma distribuição normal**.
 - Embora isso seja verdade, supondo uma distribuição normal torna os cálculos muito mais fáceis.

Exemplo: Probabilidade da prática de esportes

Nesse exemplo vamos usar o classificador ***Naïve Bayes Gaussiano*** para calcular a probabilidade dos jogadores jogarem ou não, com base nas condições climáticas. Baseado nos dados abaixo, qual a probabilidade dos jogadores jogarem se temperatura = 25 °C e humidade = 62%?

Temperatura [°C]	Humidade [%]	Jogar?
29.44	85	Não
26.67	90	Não
28.33	86	Sim
21.11	96	Sim
20.00	80	Sim
18.33	70	Não
17.78	65	Sim
22.22	95	Não
20.56	70	Sim
23.89	80	Sim
23.89	70	Sim
22.22	90	Sim
27.22	75	Sim
21.67	91	Não

Exemplo: Probabilidade da prática de esportes

Primeiro, precisamos calcular a média e variância para cada atributo, ou seja, para temperatura e humidade.

Temperatura [°C]	
$E[\text{temp.} \text{jogar=sim}]$	22.78
$\text{std}(\text{temp.} \text{jogar=sim})$	3.42
$E[\text{temp.} \text{jogar=não}]$	23.67
$\text{std}(\text{temp.} \text{jogar=não})$	4.39

Humidade [%]	
$E[\text{hum.} \text{jogar=sim}]$	79.11
$\text{std}(\text{hum.} \text{jogar=sim})$	10.22
$E[\text{hum.} \text{jogar=não}]$	86.20
$\text{std}(\text{hum.} \text{jogar=não})$	9.73

$P(\text{jogar=sim})$	9/14
$P(\text{jogar=não})$	5/14

$$\sigma_{x_k, c_q}^2 = \sum_i \frac{(x_k(i) - \mu_{x_k, c_q})^2}{n - 1}$$

$$P(\text{temp.=25} | \text{jogar=sim}) = \frac{1}{3.42\sqrt{2\pi}} e^{-\frac{(25-22.78)^2}{2(3.42)^2}} = 0.0944$$

$$P(\text{hum.=62} | \text{jogar=sim}) = \frac{1}{10.22\sqrt{2\pi}} e^{-\frac{(62-79.11)^2}{2(10.22)^2}} = 0.0096$$

$$P(\text{temp.=25} | \text{jogar=não}) = \frac{1}{4.39\sqrt{2\pi}} e^{-\frac{(25-23.67)^2}{2(4.39)^2}} = 0.0869$$

$$P(\text{hum.=62} | \text{jogar=não}) = \frac{1}{9.73\sqrt{2\pi}} e^{-\frac{(62-86.2)^2}{2(9.73)^2}} = 0.0019$$

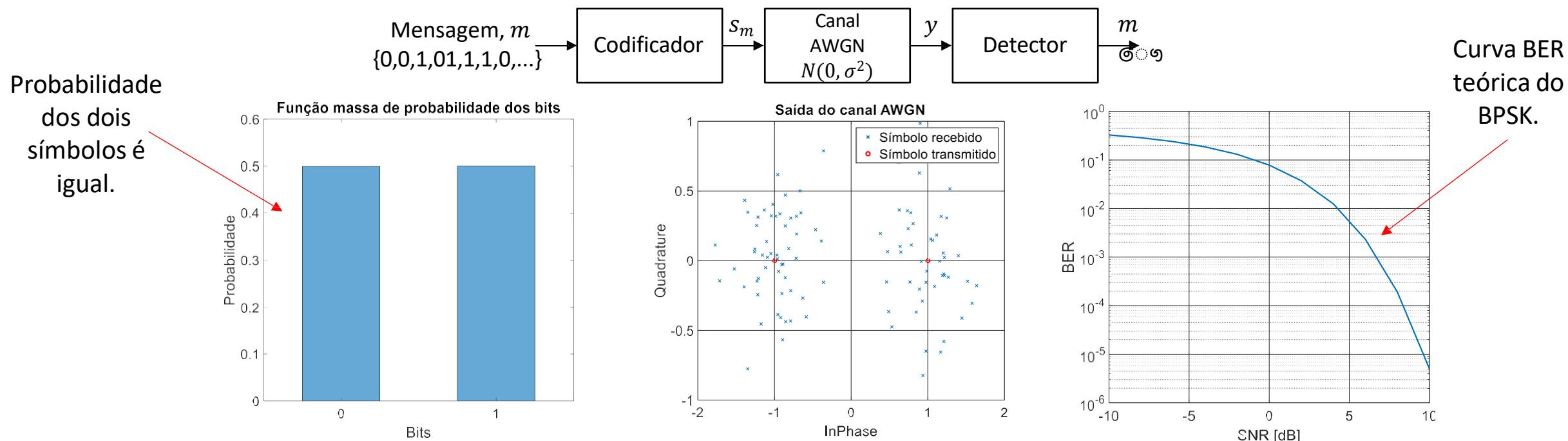
Agora calculamos as probabilidades:

- $P(\text{jogar=sim} | \text{temp.=25, hum.=62}) = P(\text{temp.=25} | \text{jogar=sim}) P(\text{hum.=62} | \text{jogar=sim}) P(\text{jogar=sim}) = 5.83e-4$
- $P(\text{jogar=não} | \text{temp.=25, hum.=62}) = P(\text{temp.=25} | \text{jogar=não}) P(\text{hum.=62} | \text{jogar=não}) P(\text{jogar=não}) = 5.78e-5$

Portanto, a probabilidade é maior para o caso deles jogarem.

Exemplo: Detecção de símbolos BPSK em canais AWGN

- Imagine um codificador que converte o m -ésimo bit de uma mensagem composta por 0s e 1s nos símbolos $s_0 = -1$ e $s_1 = 1$, para $m = 0$ e 1, respectivamente.
- Em seguida, os símbolos codificados passam por um canal AWGN cuja saída é dada por
$$y = s_m + w, \text{ onde } w \sim N(0, \sigma^2).$$
- Finalmente, o detector tem a tarefa de recuperar os bits transmitidos de tal forma que a probabilidade de erro, $P_e = P(\hat{m} \neq m)$, seja minimizada.



Exemplo: Detecção de símbolos BPSK em AWGN

- Detector MAP para esse problema é dado por:

$$S_m = \arg \max_{S_m, m=0,1} P(S_m|y) = \arg \max_{S_m, m=0,1} P(y|S_m)P(S_m)$$

- Se o símbolo s_0 é transmitido, então o sinal recebido é dado por: $y = s_0 + w$

$$P(y|s_0) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y+1)^2}{2\sigma^2}}.$$

- Se o símbolo s_1 é transmitido, então o sinal recebido é dado por: $y = s_1 + w$

$$P(y|s_1) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y-1)^2}{2\sigma^2}}.$$

- Como $P(S_0) = P(S_1) = 1/2$, então o detector MAP é equivalente ao ML, e assim

$$S_m = \arg \max_{S_m, m=0,1} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y-S_m)^2}{2\sigma^2}} = \arg \max_{S_m, m=0,1} (y - S_m)^2.$$

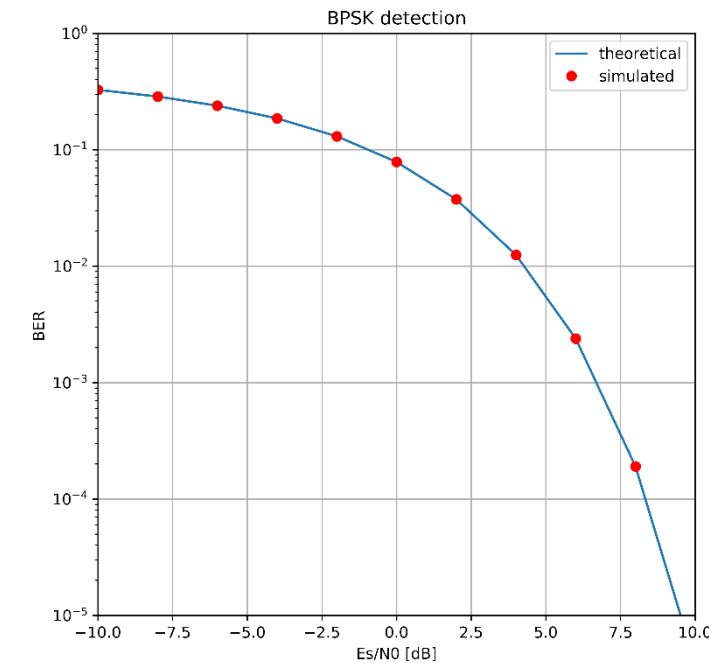
Exemplo: Detecção BPSK com Scikit-Learn

```
# Import all necessary libraries.  
import numpy as np  
from scipy.special import erfc  
from sklearn.naive_bayes import GaussianNB  
from sklearn.model_selection import train_test_split  
  
# Number of BPSK symbols to be transmitted.  
N = 1000000  
# Instantiate a Gaussian naive Bayes classifier.  
gnb = GaussianNB()  
  
# Train GNB model with high SNR value.  
  
# Create Es/No vector.  
EsN0dB = np.arange(-10,12,2)  
ber_theo = ber_simu = np.zeros(len(EsN0dB))  
for idx in range(0,len(EsN0dB)):  
    EsN0Lin = 10.0**(-(EsN0dB[idx]/10.0))  
    # Generate N BPSK symbols.  
    x = (2.0 * (np.random.rand(N) >= 0.5) - 1.0).reshape(N, 1)  
    # Generate noise vector  
    noise = np.sqrt(EsN0Lin/2.0)*np.random.randn(N, 1)  
    # Pass symbols through AWGN channel.  
    y = x + noise  
    # Predict.  
    detected_x = gnb.predict(y).reshape(len(y), 1)  
    # Simulated BPSK BER.  
    ber_simu[idx] = 1.0 * ((x != detected_x).sum()) / len(y)  
    # Theoretical BPSK BER.  
    ber_theo[idx] = 0.5*erfc(np.sqrt(10.0**((EsN0dB[idx]/10.0))))
```

Importa classe GaussianNB do módulo naive_bayes do biblioteca SciKit-Learn.

Instancia objeto da classe GaussianNB.

Executa a classificação.



Como podemos ver, a curva simulada se aproxima da teórica.

[Exemplo: bpsk_detection.ipynb](#)

Classificador naïve Bayes Multinomial

- Com um classificador naïve Bayes multinomial, os **atributos** são **discretos** e representam as frequências com as quais determinados eventos são gerados por uma **distribuição multinomial**, com probabilidades (p_1, p_2, \dots, p_K) , onde p_k é a probabilidade de que o evento k ocorra.
- Desta forma, o vetor de atributos $\mathbf{x} = [x_1, x_2, \dots, x_K]^T$ é então, um **histograma**, com x_k contando o número de vezes (i.e., frequência) que o evento k foi observado.
- Este classificador é normalmente usado para classificação de documentos, com eventos representando a ocorrência de uma palavra no documento.
- A probabilidade condicional de se observar um histograma \mathbf{x} dada a classe C_q é dada por

$$P(\mathbf{x}|C_q) = \frac{(\sum_k x_k)!}{\prod_k x_k!} \prod_k p_{qk}^{x_k} = \frac{(\sum_k x_k)!}{\prod_k x_k!} \prod_k P(x_k|C_q)^{x_k},$$

onde $\mathbf{x} = [x_1, \dots, x_K]$ é o vetor que representa a frequência dos eventos e p_{qk} é a probabilidade da classe C_q gerar o atributo x_k .

Exemplo: classificador multinomial com Scikit-Learn

```
# Import all necessary libraries.  
from sklearn.datasets import fetch_20newsgroups  
from sklearn.naive_bayes import MultinomialNB  
from sklearn.feature_extraction.text import CountVectorizer  
from sklearn.pipeline import make_pipeline  
from sklearn.metrics import confusion_matrix  
  
# Use the "20 Newsgroups corpus" from scikit to show how we might classify these  
short documents into categories.  
data = fetch_20newsgroups()  
data.target_names  
  
# Select just a few of these categories, and download the training and testing set.  
categories = ['talk.religion.misc', 'soc.religion.christian', 'sci.space', 'comp.graphics']  
train = fetch_20newsgroups(subset='train', categories=categories)  
test = fetch_20newsgroups(subset='test', categories=categories)  
  
# Convert a collection of text documents to a matrix of token counts.  
cv = CountVectorizer()  
# Naive Bayes classifier for multinomial models.  
mnb = MultinomialNB()  
# Create a pipeline that attaches the vectorizer to a multinomial naive Bayes  
classifier.  
model = make_pipeline(cv, mnb)  
  
# Train model. Apply the model to the training data.  
model.fit(train.data, train.target)  
# Run validation. Predict labels for the test data.  
labels = model.predict(test.data)
```

Base com 20 tópicos diferentes de discussão.

Treinamos e validamos com apenas 4 tópicos.

Converte o texto em um conjunto de valores numéricos representativos, ou seja, uma matriz com o número de ocorrências de cada palavra.

Treinamento e validação do classificador.

- Classificação de textos em categorias/classes.
- Esse exemplo usa uma base de dados de grupos de discussão disponibilizada pela biblioteca Scikit-learn.
- Ele classifica textos em 4 classes: 'religião', 'cristianismo', 'espaço' e 'computadores'.
- O objeto da classe **CountVectorizer** cria uma matriz registrando o número de vezes que cada palavra aparece.
- A **matriz de confusão** é usada para verificar a performance do classificador.

		comp.graphics	sci.space	soc.religion.christian	talk.religion.misc
True label	Predicted label	371	11	5	5
	comp.graphics	11	377	4	11
comp.graphics	sci.space	2	5	379	49
sci.space	soc.religion.christian	5	1	10	186
soc.religion.christian	talk.religion.misc	11	49	186	5
talk.religion.misc		5	11	186	5

[Exemplo: ClassifyingTextMultinomialNB.ipynb](#)

Classificador naïve Bayes Bernoulli

- Esse classificador é baseado na ***distribuição de Bernoulli***, que é uma ***distribuição discreta binária***.
- Portanto, esse classificador considera que os ***atributos*** são ***variáveis binárias*** (i.e., booleanas) ***independentes***, ou seja, o ***atributo*** pode estar ***presente*** ou ***ausente***.
- Assim como o classificador multinomial, esse classificador pode ser utilizado para tarefas de classificação de documentos, onde atributos binários da ocorrência de termos são usados ao invés da frequência dos termos.

Classificador naïve Bayes Bernoulli

- Se x_k é um **atributo booleano** que expressa a **presença** ou **ausência** do k -ésimo termo de um conjunto de termos (e.g., conjunto de palavras), então a probabilidade condicional (i.e., **verosimilhança**) de um determinado conjunto de atributos pertencer à classe C_q é dado por

$$\begin{aligned} P(x|C_q) &= \prod_k p_{qk}^{x_k} (1 - p_{qk})^{(1-x_k)} \\ &= \prod_k P(t_k|C_q)^{x_k} (1 - P(t_k|C_q))^{(1-x_k)} \\ &= \prod_k [P(t_k|C_q)x_k + (1 - P(t_k|C_q))(1 - x_k)], \end{aligned}$$

onde $x = [x_1 \dots x_K]$ é o **vetor binário** que representa a ocorrência ou não do termo, t_k , e p_{qk} é a probabilidade da classe C_q gerar o termo t_k , i.e., $P(t_k|C_q)$.

Naïve Bayes Bernoulli vs. Multinomial

- O classificador de Bernoulli usa informações binárias da ocorrência de termos, ignorando o número de ocorrências.
- Já o classificador multinomial usa informações de frequência dos termos, não ignorando múltiplas ocorrências.
- Como resultado, o modelo de Bernoulli normalmente comete muitos erros ao classificar documentos longos.
- Por exemplo, ele pode atribuir um documento inteiro à classe **Ficção-científica** devido a *uma única ocorrência* do termo *nave espacial*.

Naïve Bayes Bernoulli vs. Multinomial

- Analisando-se as verosimilhanças, percebe-se que diferentemente do classificador ***multinomial***, o ***Bernoulli*** penaliza explicitamente a não ocorrência de um termo, t_k , já o ***multinomial*** simplesmente ignora um termo que não ocorre.
- **Bernoulli**

$$P(x|C_q) = \prod_k [P(t_k|C_q)x_k + (1 - P(t_k|C_q))(1 - x_k)].$$

- **Multinomial**

$$P(x|C_q) = \frac{(\sum_k x_k)!}{\prod_k x_k!} \prod_k p_{qk} x_k = \frac{(\sum_k x_k)!}{\prod_k x_k!} \prod_k P(x_k|C_q)^{x_k}$$

- Portanto, o classificador de Bernoulli é bastante utilizado para classificar ***textos curtos***, pois geralmente tem desempenho melhor e tem o benefício de classificar explicitamente a ausência de termos.

Exemplo: Classificador Bernoulli com Scikit-Learn

```
# Import all necessary libraries.  
import pandas as pd  
from sklearn.naive_bayes import BernoulliNB  
from sklearn.feature_extraction.text import CountVectorizer  
from sklearn.model_selection import train_test_split  
  
# Read SMS data base with pandas.  
url = 'https://raw.githubusercontent.com/justmarkham/pycon-2016-tutorial/master/data/sms.tsv'  
sms = pd.read_table(url, header=None, names=['label', 'message'])  
  
# Convert label to a numerical variable  
sms['label_num'] = sms.label.map({'ham':0, 'spam':1})  
  
# Create feature and label vectors.  
X = sms.message  
y = sms.label_num  
  
# Split array into random train and test subsets.  
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)  
  
# Convert a collection of text documents into a matrix of token counts.  
vect = CountVectorizer(binary=True)  
# Learn the vocabulary dictionary and return term-document matrix.  
X_train_t = vect.fit_transform(X_train)  
  
# Instantiate a Bernoulli Naive Bayes model.  
nb = BernoulliNB(binarize=None)  
# Train the MultinomialNB model.  
nb.fit(X_train_t, y_train)  
  
# Transform document into document-term matrix.  
X_test_t = vect.transform(X_test)  
# Perform classification on an array of test vectors X_test_t.  
y_pred_class = nb.predict(X_test_t)
```

Download da base de dados.

Converte labels em valores discretos.

Divide a base de dados em 75% treinamento e 25% validação.

Cria matriz booleana indicando ou não a presença de uma palavra.

Treinamento do classificador.

Cria matriz booleana com a presença ou não de uma palavra para a base de validação

Validação do classificador.

- Classificação de mensagens entre SPAM e não-SPAM (HAM).
- Esse exemplo usa uma base de dados de mensagens SMS baixada do GitHub.
- Ele classifica as mensagens em 2 classes: 'SPAM' e 'HAM'.
- O objeto da classe **CountVectorizer** cria uma matriz registrando se uma palavra aparece ou não (booleano) em cada mensagem.
- A **matriz de confusão** é usada para verificar a performance do classificador.

		predicted label	
		ham	spam
true label	ham	1207	30
	spam	0	156

Positivo verdadeiro (true positive)

Negativo falso (False negative)

Negativo verdadeiro (true negative)

[Exemplo: SPAMClassificationBernoulliNB.ipynb](#)

Classificação linear

- Como vimos anteriormente, o objetivo da **classificação** é usar as características (i.e., atributos) de, por exemplo, um objeto para identificar a qual classe ele pertence.
- Um **classificador linear** atinge esse objetivo tomando uma decisão de classificação com base no valor de uma **combinação linear** dos **atributos**, ou seja, na saída de uma **função discriminante linear**.
- A saída de um **classificador linear** é dada por

$$\hat{y} = h_{\mathbf{a}}(\mathbf{x}) = f(g(\mathbf{x})) = f\left(\sum_{k=0}^K a_k x_k\right) = f(\mathbf{a}^T \mathbf{x})$$

onde $\mathbf{x} = [1, x_1, \dots, x_K]^T$ e $f(\cdot)$ é uma **função de limiar de decisão**.

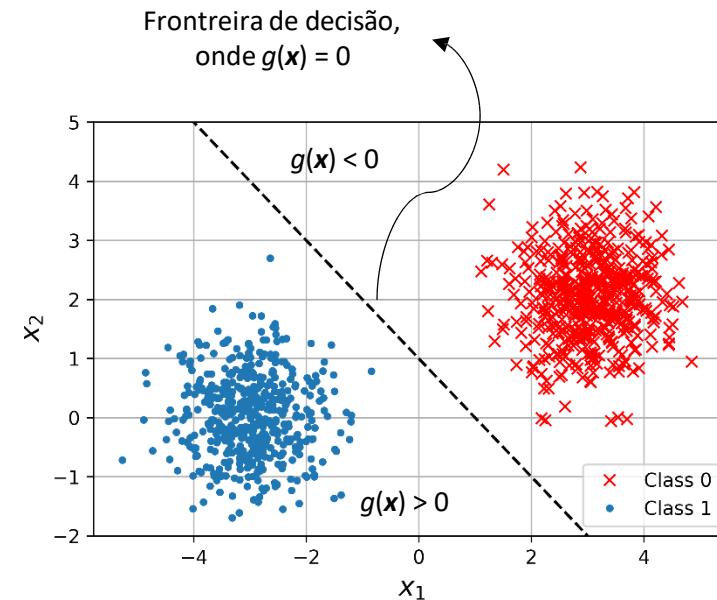
- **Função de limiar de decisão** é uma função que converte a saída da **função discriminante linear**, $\mathbf{a}^T \mathbf{x}$ (produto escalar), na saída desejada, ou seja, na classe C_q , $q = 1, \dots, Q$, do objeto.
- Ela é apenas uma formalização matemática para os **ifs** e **elses** que usamos para definir as classes.
- Originalmente, as **funções discriminantes** são formadas por equações de **hiperplanos**

$$g(\mathbf{x}) = a_0 + a_1 x_1 + \dots + a_K x_K = \sum_{k=0}^K a_k x_k$$

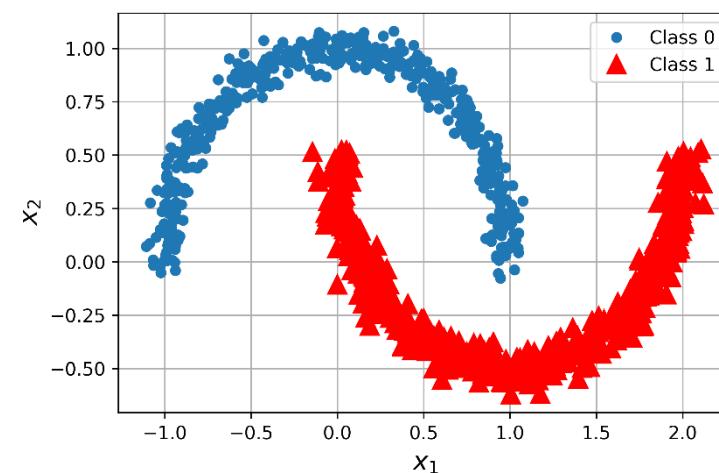
- $h_{\mathbf{a}}(\mathbf{x})$ é conhecida como **função hipótese de classificação**.

Classificação linear

- Dado um **conjunto de treinamento**, a tarefa do **classificador** é a de **aprender** uma **função hipótese de classificação**, $h_a(x)$, que receba um exemplo (e.g., x_1 e x_2) e retorne a classe do exemplo.
- **Classificadores binários** têm como saída o valor **0** caso o exemplo pertença à classe C_1 (também chamada de **classe negativa**) ou **1** caso ele pertença à classe C_2 (também chamada de **classe positiva**).
- Para que um **classificador linear** funcione corretamente, as duas classes devem ser **linearmente separáveis**.
- Isso significa que as classes devem ser **suficientemente separadas**umas das outras para garantir que a **superfície de decisão** seja um **hiperplano**.
- Classes que podem ser separadas por um **hiperplano** são chamadas de **linearmente separáveis**.
- Na primeira figura, a **fronteira de decisão** é definida por uma **função discriminante** que é uma **reta**: $g(x) = 1 - x_1 - x_2$.
- Na segunda figura, devido à proximidade das classes, não existe um **hiperplano** que as separe.



Classes linearmente separáveis.



Classes não-linearmente separáveis.

Classificação não-linear

- Originalmente, **classificação linear** é usada quando as classes podem ser separadas por **superfícies de decisão lineares**.
- Ou seja, as **funções discriminantes** são **hiperplanos**: $\sum_{k=0}^K a_k x_k$.
- Mas e se não pudermos separar as classes com um **hiperplano**, ou seja, se elas não forem **linearmente separáveis**?
- Nestes casos, podemos usar **funções discriminantes não-lineares**, como, por exemplo, **polinômios**:
 - $g(x) = (x_1 - a)^2 + (x_2 - b)^2 - r^2$, Círculo centrado em (a, b) e raio r .
 - $g(x) = \frac{(x_1 - a)^2}{c^2} + \frac{(x_2 - b)^2}{d^2} - 1$, Elipse centrada em (a, b) , largura $2c$ e altura $2d$.
 - $g(x) = (x_1 - a)(x_2 - b) - c$, Hipérbole retangular com eixos paralelos às suas assíntotas.
- Portanto, quando usamos uma **função discriminante não-linear**, convertemos **classificadores lineares** em **classificadores não-lineares** através da aplicação de uma **transformação dos atributos**.
- Esta transformação pode ser vista também como uma mudança do espaço de entrada, o que normalmente leva ao aumento das dimensões de entrada ou mudança dos eixos.

Limiar de decisão rígido

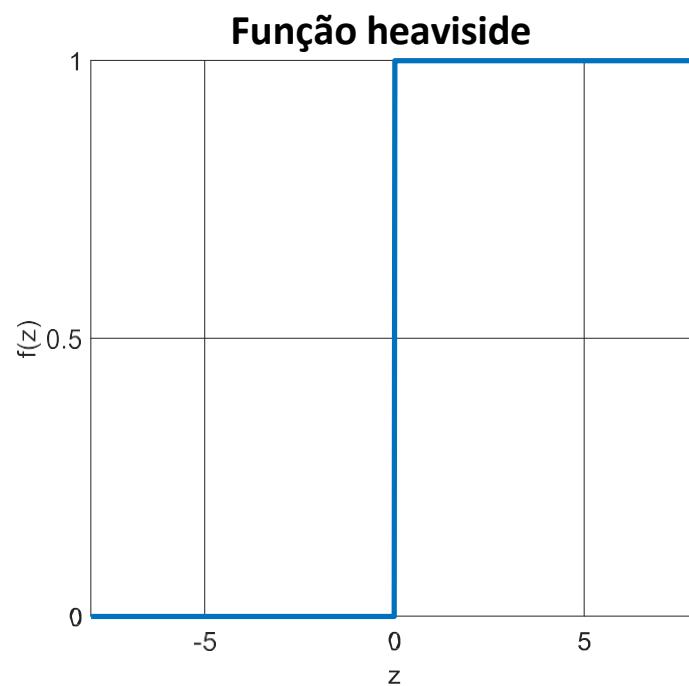
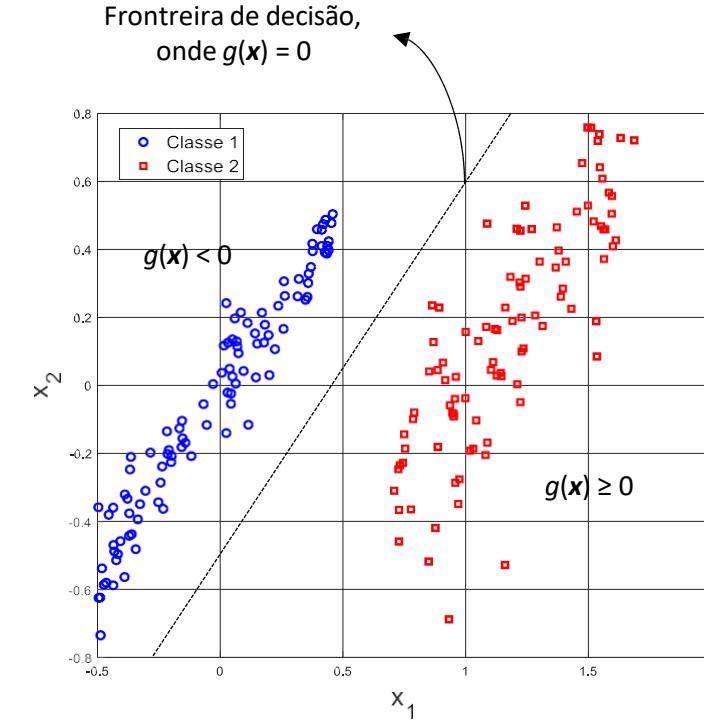
- Para o exemplo ao lado, podemos definir a **função hipótese de classificação** como

$$y = h_a(x) = \begin{cases} 0, & g(x) = x^T a < 0 \text{ (Classe 1)} \\ 1, & g(x) = x^T a \geq 0 \text{ (Classe 2)} \end{cases}$$

- Percebam que a saída da **função hipótese** é **binária**, ou seja, temos apenas 2 possíveis valores, 0 ou 1.
- O mapeamento entre o valor da função discriminante, $g(x)$, e a saída 0 ou 1 é feita através da **função de limiar de decisão**, $f(g(x))$.
- Uma **função de limiar de decisão** que faça o mapeamento do valor de $g(x)$ em apenas 2 valores é chamada de **função de limiar de decisão rígido**.
- A **função de limiar de decisão rígido** é mostrada na figura ao lado e é definida como

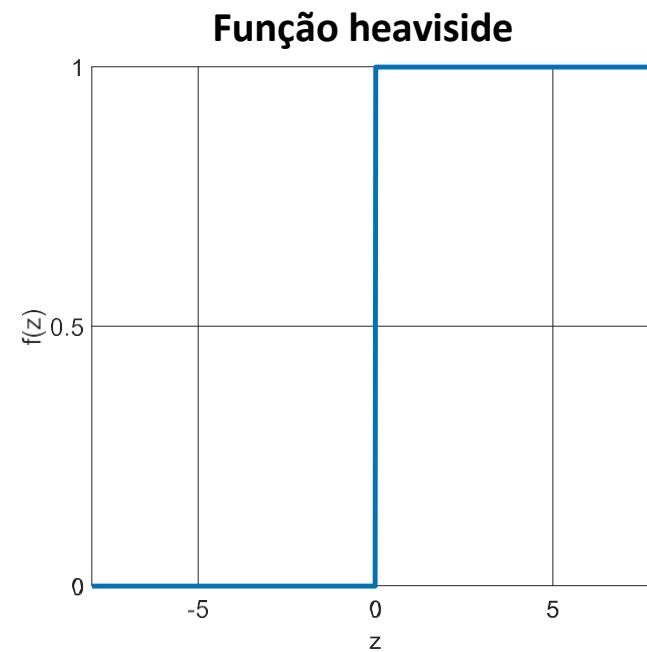
Conhecida
também como
função heaviside.

→ $f(z) = \begin{cases} 0, & z < 0 \\ 1, & z > 0 \\ \text{Indeterminado}, & z = 0 \end{cases}$



Classificação com limiar de decisão rígido

- Agora que a **função hipótese**, $h_a(x)$, tem uma forma matemática bem definida, nós podemos pensar em como escolher os pesos a que **minimizem o erro de classificação**.
- No caso da **regressão linear**, nós fizemos isso de duas maneiras:
 - i. de forma fechada (através da **equação normal**) fazendo a derivada parcial com relação aos pesos igual a zero e resolvendo a equação para os pesos;
 - ii. e através do algoritmo do **gradiente descendente**.
- Entretanto, com a **função de limiar rígido**, nenhuma das duas abordagens é possível devido ao fato do **gradiente** ser igual a zero em todos os pontos do espaço de pesos exceto no ponto onde $x^T a = 0$, e mesmo assim, o **gradiente** é indeterminado nesse ponto.
- **Portanto, o que podemos fazer?**



Classificação com limiar de decisão rígido

- Uma possível abordagem para o problema quando utilizamos um **limiar de decisão rígido** é utilizar uma **regra intuitiva** de atualização dos **pesos** que converge para uma solução **dado que exista uma função discriminante adequada e que as classes não se sobreponham**.
- A atualização dos **pesos** é dada pela seguinte equação

$$\mathbf{a} = \mathbf{a} + \alpha (y(i) - h_{\mathbf{a}}(\mathbf{x}(i))) \mathbf{x}(i), \forall i,$$

Os pesos são
atualizados a cada
novo exemplo.

a qual é essencialmente idêntica à regra de atualização para a **regressão linear** quando utilizamos o **gradiente descendente estocástico**.

- Esta regra é chamada de **regra de aprendizagem do perceptron**, por razões que discutiremos em breve.
- Essa regra de aprendizagem é aplicada a **um exemplo por vez**, escolhendo exemplos **aleatoriamente**, assim como fizemos com o **gradiente descendente estocástico**.
- Como estamos considerando classificadores com valores de saída 0 ou 1, o comportamento da regra de atualização será diferente do comportamento para a regressão linear, como veremos a seguir.

Classificação com limiar de decisão rígido

- Observem a equação de atualização dos pesos

$$\mathbf{a} = \mathbf{a} + \alpha \left(y(i) - h_{\mathbf{a}}(\mathbf{x}(i)) \right) \mathbf{x}(i).$$

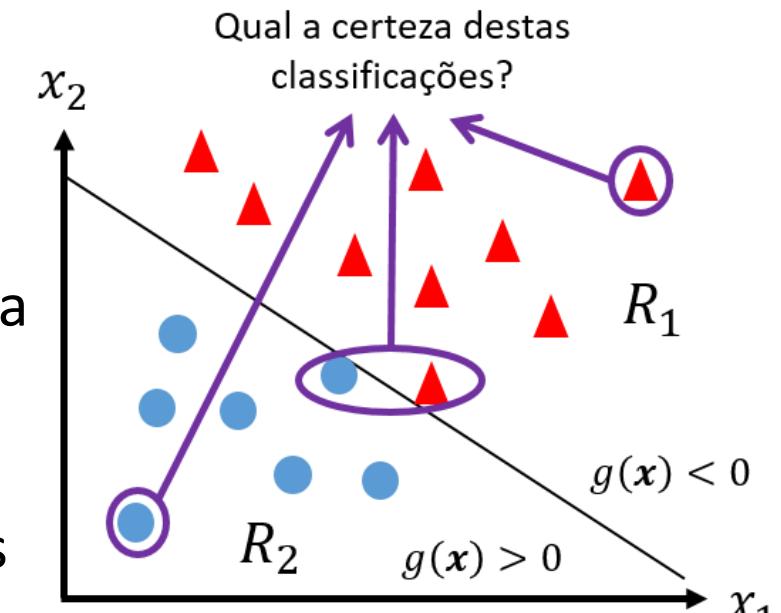
- Ambos, o valor desejado, y , e a saída da **função hipótese**, $h_{\mathbf{a}}(\mathbf{x})$, assumem os valores 0 ou 1, portanto, existem 3 possibilidades:
 - Se a saída estiver correta, i.e., $y = h_{\mathbf{a}}(\mathbf{x})$, então os pesos não são atualizados.
 - Se $y = 1$, mas $h_{\mathbf{a}}(\mathbf{x}) = 0$, então o peso a_k tem seu valor **aumentado** quando o valor de x_k é positivo e **diminuído** quando o valor de x_k é negativo.
 - Isso faz sentido pois nós queremos aumentar o valor do produto escalar $\mathbf{x}^T \mathbf{a}$, ou seja, $g(\mathbf{x})$, de tal forma que $h_{\mathbf{a}}(\mathbf{x})$ tenha como saída o valor 1.
 - Se $y = 0$, mas $h_{\mathbf{a}}(\mathbf{x}) = 1$, então o peso a_k tem seu valor **diminuido** quando o valor de x_k é positivo e **aumentado** quando o valor de x_k é negativo.
 - Isso faz sentido pois nós queremos diminuir o valor do produto escalar $\mathbf{x}^T \mathbf{a}$, ou seja, $g(\mathbf{x})$, de tal forma que $h_{\mathbf{a}}(\mathbf{x})$ tenha como saída o valor 0.

Classificação com limiar de decisão rígido

- A **regra de aprendizagem do perceptron** converge para um **separador perfeito** quando:
 - As classes são **suficientemente separadas** umas das outras, ou seja, não se sobrepõem.
 - Existe uma **função discriminante adequada para o problema**, mesmo que não seja um **hiperplano**.
- **Separador perfeito:** com erro de classificação igual a zero, ou seja, todos os exemplos são perfeitamente classificados.
- Porém, na prática essa situação não é muito comum.
- Nesse caso, a **regra de aprendizagem do perceptron** falha em convergir para uma solução perfeita.
- Em geral, essa regra não converge para uma solução estável para valores fixos do **passo de aprendizagem**, α , mas se α decresce de acordo com as iterações, então a regra tem uma chance de convergir para uma solução de erro mínimo quando os exemplos são apresentados de forma aleatória.
- Podemos também usar o **early-stopping** e utilizar os **pesos** que resultaram no menor erro de validação.

Classificação com limiar de decisão rígido

- Outro problema com classificadores que usam **limiar de decisão rígido** é a **falta de informação sobre a confiança do modelo sobre uma classificação**.
- No exemplo ao lado, dois exemplos estão bem próximos da **fronteira de decisão** enquanto outros dois estão bem distantes dela.
- O classificador com **limiar rígido**, faria uma previsão **completamente confiante** pelo valor 1 para os dois pontos azuis e 0 para os dois triângulos vermelhos, mesmo eles possuindo valores bem diferentes de $g(x)$.
- Em muitas situações, nós precisamos de **previsões mais graduadas**, que indiquem incertezas quanto à classificação.
- **Todos os problemas com a função de limiar rígido podem ser resolvidos com sua suavização** através de sua aproximação por uma função que seja **contínua, diferenciável e assuma valores reais dentro do intervalo de 0 a 1**.



- Os pontos distantes da **fronteira de decisão** têm valores absolutos de $g(x)$ bem maiores do que os dos pontos próximos, os quais têm valores de $g(x)$ muito próximos de 0.
- Ou seja, a **confiança** deveria ser maior para pontos distantes da fronteira.
- Porém, isso não é refletido na saída do classificador com **limiar rígido**.

Classificação com função de limiar logístico

- A **função logística** (ou **sigmóide**), mostrada na figura ao lado, é definida como

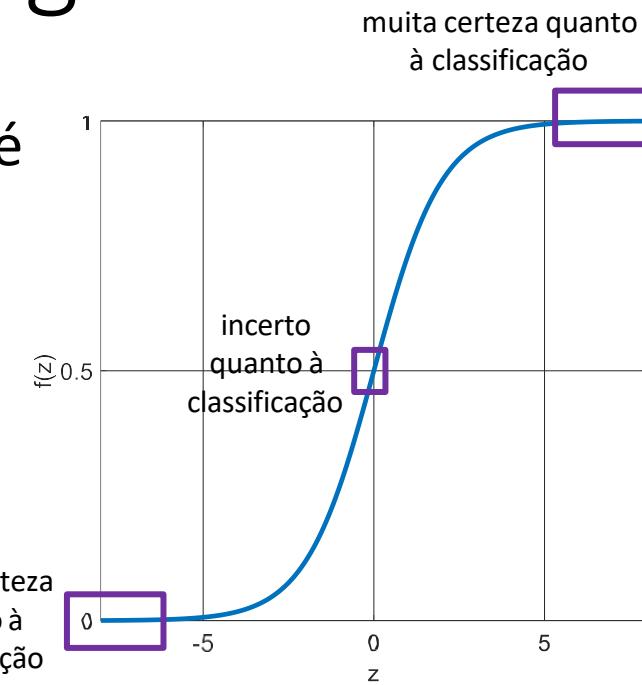
$$\text{Logistic}(z) = \frac{1}{1+e^{-z}} \in [0, 1],$$

apresenta tais propriedades matemáticas.

- Utilizando a **função logística** como **função de limiar**, temos

$$h_a(x) = \text{Logistic}(g(x)) = \frac{1}{1+e^{-g(x)}} \in [0, 1].$$

- $g(x)$ pode ser um **hiperplano** ou um **polinômio**.
- A saída será um número real entre 0 e 1, o qual pode ser interpretado como a **probabilidade** da classe C_2 (i.e., **classe positiva**) dado um exemplo e o vetor de pesos, a .
 - Por exemplo, qual é a probabilidade da classe spam (classe +) dado um email (i.e., vetor de atributos) e os pesos do modelo?
- A nova **função hipótese**, $h_a(x)$, forma uma **fronteira de decisão suave**, a qual confere a probabilidade de 0.5 para exemplos em cima da **fronteira de decisão** e se aproxima de 0 ou 1 conforme a posição do exemplo se distancia da fronteira de decisão.



A função logística realiza um mapeamento $\mathbb{R} \rightarrow [0, 1]$.

Quanto mais longe da **fronteira de decisão**, mais próximo o valor de saída da **função hipótese** será de 0 ou de 1 e, portanto, mais certeza teremos sobre uma classificação.

Em resumo, quanto mais longe da fronteira, maior será o valor absoluto de $g(x)$.

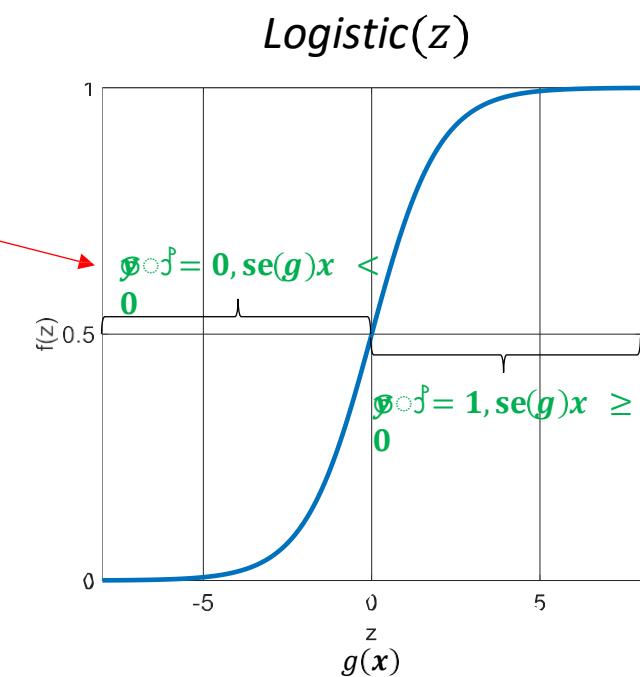
Regressão logística

- Esse classificador com função de **limiar logístico** é conhecido como **regressor logístico**.
 - É chamado de **regressor**, pois sua saída pode assumir infinitos valores dentro do intervalo 0 e 1.
- O **regressor logístico** pode ser usado para **classificação binária**, mas para isso, precisamos quantizar sua saída.
 - Se **quantiza** a saída da **função hipótese**, $h_a(x)$, nos valores 0 ou 1.
- Se a **probabilidade** estimada para um exemplo for igual ou maior do que 50%, o classificador **prediz** que o exemplo pertence à **classe positiva**, rotulada como 1, caso contrário, **prediz** que pertence à **classe negativa**, rotulada como 0.
- Ou seja, a saída **quantizada** do **regressor logístico** é dada por

$$\text{Classe} = \hat{y} = \begin{cases} 0 \text{ (classe } C_1 \text{ -- Negativa)}, & \text{se } h_a(x) < 0.5 \\ 1 \text{ (classe } C_2 \text{ -- Positiva)}, & \text{se } h_a(x) \geq 0.5 \end{cases}$$

Regressão logística

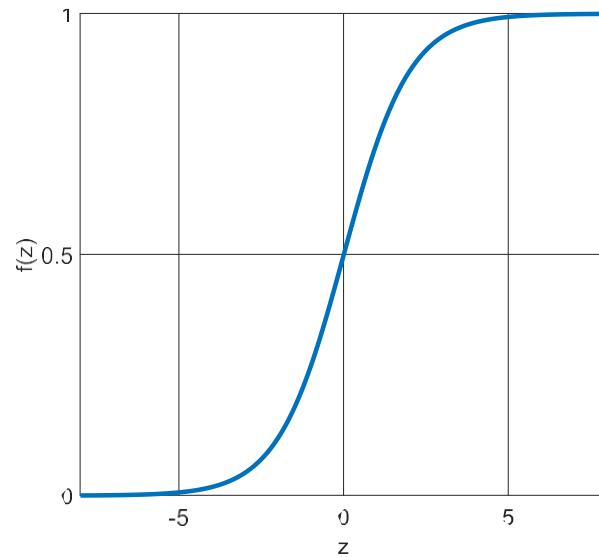
- Notem que $\text{Logistic}(z) < 0.5$ quando $z < 0$ e $\text{Logistic}(z) \geq 0.5$ quando $z \geq 0$, portanto, o modelo de **regressão logística** prediz a classe positiva, C_2 , (i.e., $\hat{y} = 1$) se $g(x) \geq 0$ e C_1 (i.e., $\hat{y} = 0$) se $g(x) < 0$.
- A **regressão logística** funciona usando **combinação linear dos atributos** (i.e., hiperplano ou polinômio), para que várias fontes de informação (i.e., atributos) possam ditar a saída do modelo.
- Os **parâmetros do modelo** são os **pesos** associados aos vários **atributos** e **representam a importância relativa de cada atributo para o resultado**.
- Mesmo sendo uma técnica bastante simples, a **regressão logística** é muito utilizada em várias aplicações do mundo real em áreas como medicina, marketing, análise de crédito, etc.
- Além disto, toda a teoria por trás da **regressão logística** foi a base para a criação das primeiras **redes neurais**.



Exemplos: classificar críticas de filmes como positivas ou negativas, probabilidade de um paciente desenvolver um doença, detecção de spam, classificar transações bancárias como fraudulentas ou não, etc.

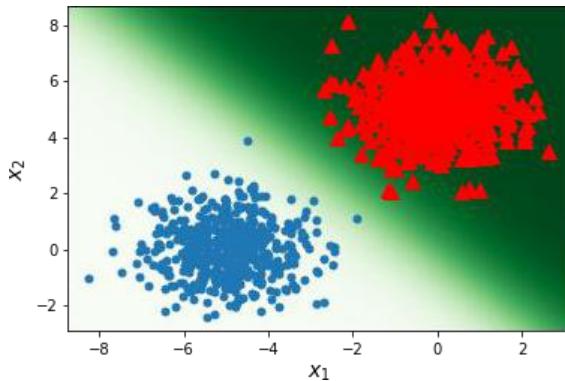
Propriedades da regressão logística

- Os valores de saída da **função hipótese**, $h_a(x)$, ficam restritos ao intervalo $0 \leq h_a(x) \leq 1$.
- A saída de $h_a(x)$ representa a **probabilidade da classe positiva**, C_2 , dado o vetor de atributos x e um dado vetor de pesos, a .
- Ou seja, $h_a(x)$ dá a probabilidade condicional da **classe positiva**, C_2 :
$$h_a(x) = P(C_2 | x; a).$$
- Consequentemente, o complemento de $h_a(x)$, ou seja,
$$(1 - h_a(x)) = P(C_1 | x; a)$$
- é a probabilidade condicional da **classe negativa**, C_1 .

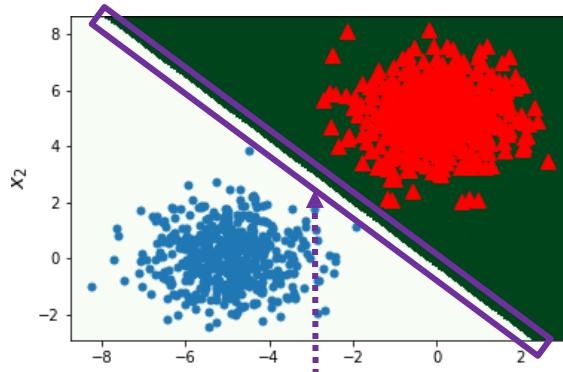


Propriedades da regressão logística

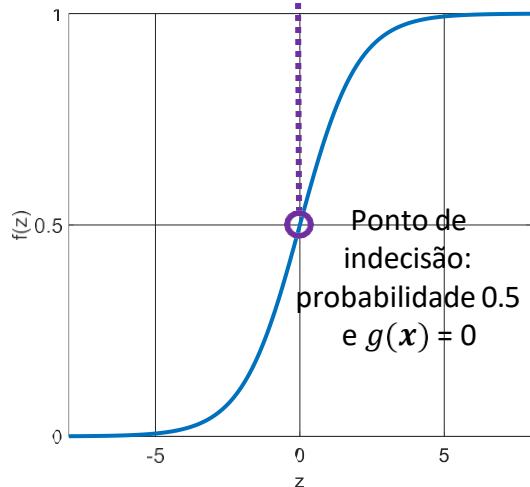
- A **fronteira de decisão** do regressor logístico é suave, mas após a **quantização** de sua saída, ela se torna rígida.
- A **fronteira de decisão rígida (classificador)** é determinada quando há uma **indecisão** entre as classes, ou seja, quando $P(C_1 | \mathbf{x}; \mathbf{a}) = P(C_2 | \mathbf{x}; \mathbf{a})$, que ocorre quando $P(C_2 | \mathbf{x}; \mathbf{a}) = h_{\mathbf{a}}(\mathbf{x}) = 0.5$.
- Observando a figura da **função logística**, nós percebemos que $\text{Logistic}(z) = 0.5$ quando $z = 0$.
- Ou seja, quando $g(\mathbf{x}) = 0$ (\mathbf{x} se encontra em cima da **função discriminante**), a probabilidade de \mathbf{x} pertencer à classe C_1 ou C_2 é de 50% para as duas classes, indicando que o classificador está indeciso.



Fronteira de decisão suave com função logística.



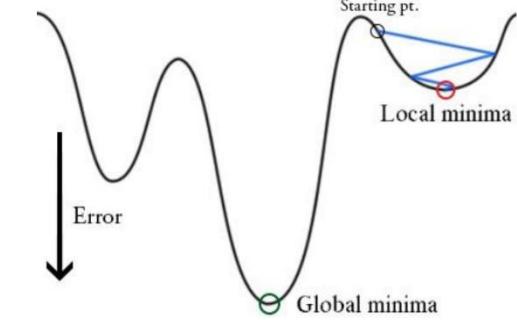
Fronteira de decisão rígida após a quantização



Função de erro

- Para treinarmos um **regressor logístico** e encontrarmos os **pesos** da **função hipótese**, nós precisamos, assim como fizemos com a **regressão linear**, definir uma **função de erro**.
- Porém, adotar o **erro quadrático médio** como **função de erro** não é uma boa escolha para a **adaptação dos pesos** no caso da **regressão logística** como veremos a seguir.
- A **função de erro** utilizando o **erro quadrático médio** é dada por

$$\begin{aligned} J_e(\mathbf{a}) &= \frac{1}{N} \sum_{i=1}^N (y(i) - h_{\mathbf{a}}(\mathbf{x}(i)))^2 \\ &= \frac{1}{N} \sum_{i=1}^N (y(i) - Logistic(g(\mathbf{x}(i))))^2 . \end{aligned}$$



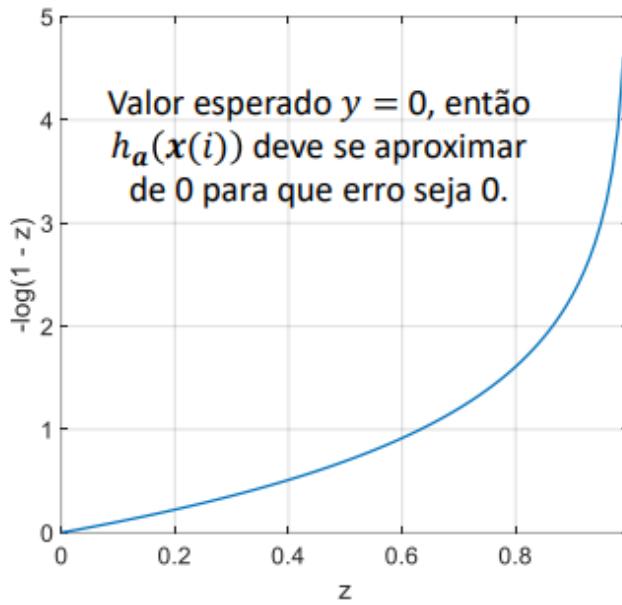
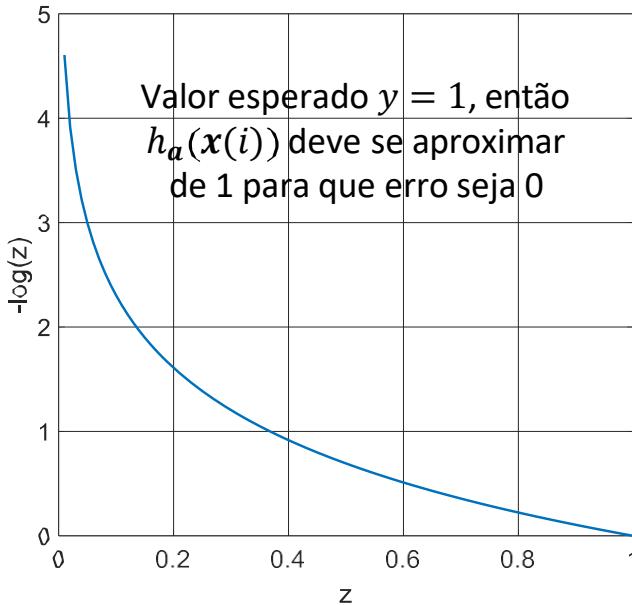
Função de erro

- Como $\text{Logistic}(\cdot)$ é uma função **não-linear**, $J_e(a)$ não será, consequentemente, uma **função convexa**, de forma que a **superfície de erro** poderá apresentar vários mínimos locais que vão dificultar o aprendizado (e.g., o algoritmo do gradiente descendente pode ficar preso em um mínimo local).
- **Ideia:** encontrar uma **função de erro** que tenha **superfície de erro convexa**, facilitando o encontro dos pesos por algoritmos baseados no gradiente descendente.
- Uma proposta **intuitiva** para a **função de erro para cada exemplo de entrada** é dada por

$$\text{Erro}(h_a(x(i)); y(i)) = \begin{cases} -\log(h_a(x(i))), & \text{se } y(i) = 1 \\ -\log(1 - h_a(x(i))), & \text{se } y(i) = 0 \end{cases}$$

- onde $y(i)$ é o i -ésimo valor esperado (i.e., rótulo) e $\log(\cdot)$ é o logaritmo natural.
- Veremos a seguir o motivo desta escolha.

Função de erro



- As figuras ao lado mostram as duas situações possíveis para a **função de erro**.
- Como podemos observar, a transformação aplicada a cada saída do regressor logístico reflete o **erro de classificação**.
- **Unindo-se as duas funções, obtém-se uma função convexa.**

- O uso dessa **função de erro** faz sentido pois:
 - O valor de $-\log(z)$ se torna muito grande quando z se aproxima de 0, então o erro será grande se o classificador estimar uma probabilidade próxima a 0 para um exemplo positivo (i.e., pertencente à classe C_2)
 - O valor de $-\log(1 - z)$ será muito grande se o classificador estimar uma probabilidade próxima de 1 para um exemplo negativo (i.e., pertencente à classe C_1) .
 - Por outro lado, $-\log(z)$ se torna próximo de 0 quando z se aproxima de 1, portanto, o erro será próximo de 0 se a probabilidade estimada for próxima de 1 para um exemplo positivo.
 - O valor $-\log(1 - z)$ se torna próximo de 0 quando z se aproxima de 0, portanto, o erro será próximo de 0 para um exemplo negativo.

Função de erro

- Nós podemos unir a **função de erro para cada exemplo** em uma expressão única, dada por

$$\text{Erro} \left(h_{\mathbf{a}}(\mathbf{x}(i)); y(i) \right) = \underbrace{-y(i) \log(h_{\mathbf{a}}(\mathbf{x}(i)))}_{\text{Só exerce influência no erro se } y=1} - \underbrace{(1 - y(i)) \log(1 - h_{\mathbf{a}}(\mathbf{x}(i)))}_{\text{Só exerce influência no erro se } y=0}.$$

← Erro para um único exemplo

- Com isto, podemos definir a seguinte **função de erro médio**:

$$\begin{aligned} J_e(\mathbf{a}) &= -\frac{1}{N} \sum_{i=0}^{N-1} y(i) \log(h_{\mathbf{a}}(\mathbf{x}(i))) + (1 - y(i)) \log(1 - h_{\mathbf{a}}(\mathbf{x}(i))) \\ &= -\frac{1}{N} \sum_{i=0}^{N-1} y(i) \log(P(C_2 | \mathbf{x}(i); \mathbf{a})) + (1 - y(i)) \log(P(C_1 | \mathbf{x}(i); \mathbf{a})) \\ &= -\frac{1}{N} \sum_{i=0}^{N-1} \sum_{q=1}^Q 1\{y(i) + 1 == q\} \log(P(C_q | \mathbf{x}(i); \mathbf{a})) \end{aligned}$$

• $y(i) \in \{0,1\} \forall i$
• $q \in \{1,2\}$

onde $1\{\cdot\}$ é a **função indicadora**: $1\{\text{uma afirmação verdadeira}\} = 1$ e $1\{\text{uma afirmação falsa}\} = 0$.

- A má notícia é que não existe uma **equação de forma fechada** conhecida para encontrar os pesos que minimizem essa **função de erro** (ou seja, não há um equivalente da **equação normal**).
- A boa notícia é que essa **função de erro também é convexa** e, portanto, é garantido que o algoritmo do **gradiente descendente** encontre o mínimo global (dado que a **taxa de aprendizagem** não seja muito grande e que se espere tempo suficiente para a convergência).

Processo de treinamento

- Portanto, da mesma forma como fizemos com a *regressão linear*, usamos o algoritmo do **gradiente descendente para encontrar os pesos que minimizam a função de erro médio**.
- Na sequência, vamos encontrar o vetor gradiente para implementar o gradiente descendente.
- Antes de encontrarmos o **vetor gradiente** de $J_e(\mathbf{a})$, vamos reescrever a **função de erro** utilizando as seguintes equivalências

$$\log(h_{\mathbf{a}}(\mathbf{x}(i))) = \log\left(\frac{1}{1+e^{-\mathbf{x}(i)^T \mathbf{a}}}\right) = -\log\left(1 + e^{-\mathbf{x}(i)^T \mathbf{a}}\right),$$

$$\log(1 - h_{\mathbf{a}}(\mathbf{x}(i))) = \log\left(1 - \frac{1}{1+e^{-\mathbf{x}(i)^T \mathbf{a}}}\right) = -\mathbf{x}(i)^T \mathbf{a} - \log\left(1 + e^{-\mathbf{x}(i)^T \mathbf{a}}\right).$$

- Assim, a nova expressão para a **função de erro médio** é dada por

$$J_e(\mathbf{a}) = -\frac{1}{N} \sum_{i=0}^{N-1} -y(i)\log\left(1 + e^{-\mathbf{x}(i)^T \mathbf{a}}\right) + (1 - y(i)) \left[-\mathbf{x}(i)^T \mathbf{a} - \log\left(1 + e^{-\mathbf{x}(i)^T \mathbf{a}}\right) \right]$$

Processo de treinamento

- O termo $-y(i) \log(1 + e^{-\mathbf{x}(i)^T \mathbf{a}})$ é cancelado com um dos elementos gerados a partir do produto envolvido no segundo termo, de forma que

$$J_e(\mathbf{a}) = -\frac{1}{N} \sum_{i=0}^{N-1} -\mathbf{x}(i)^T \mathbf{a} + y(i)\mathbf{x}(i)^T \mathbf{a} - \log(1 + e^{-\mathbf{x}(i)^T \mathbf{a}})$$

- Se $-\mathbf{x}(i)^T \mathbf{a} = -\log(e^{\mathbf{x}(i)^T \mathbf{a}})$, então

$$-\mathbf{x}(i)^T \mathbf{a} - \log(1 + e^{-\mathbf{x}(i)^T \mathbf{a}}) = -\log(1 + e^{\mathbf{x}(i)^T \mathbf{a}}).$$

- Desta forma, a **função de erro médio** se torna

$$J_e(\mathbf{a}) = -\frac{1}{N} \sum_{i=0}^{N-1} y(i)\mathbf{x}(i)^T \mathbf{a} - \log(1 + e^{\mathbf{x}(i)^T \mathbf{a}})$$

- Em seguida, encontramos o **vetor gradiente** de cada termo da equação acima.

Processo de treinamento

- Assim, o **vetor gradiente** do primeiro termo da equação anterior é dado por

$$\frac{\partial [y(i)\mathbf{x}(i)^T \mathbf{a}]}{\partial \mathbf{a}} = y(i)\mathbf{x}(i)^T$$

- O **vetor gradiente** do segundo termo da equação anterior é dado por

$$\begin{aligned}\frac{\partial [\log(1 + e^{\mathbf{x}(i)^T \mathbf{a}})]}{\partial \mathbf{a}} &= \frac{1}{1 + e^{\mathbf{x}(i)^T \mathbf{a}}} e^{\mathbf{x}(i)^T \mathbf{a}} \mathbf{x}(i)^T \\ &= \frac{1}{1 + e^{-\mathbf{x}(i)^T \mathbf{a}}} \mathbf{x}(i)^T \\ &= h_{\mathbf{a}}(\mathbf{x}(i)) \mathbf{x}(i)^T.\end{aligned}$$


- Usamos a **regra da cadeia** para encontrar o vetor gradiente do segundo termo.

Processo de treinamento

- Portanto, combinando os dois resultados anteriores, temos que o **vetor gradiente** da **função de erro médio** é dado por

$$\frac{\partial J_e(\mathbf{a})}{\partial \mathbf{a}} = -\frac{1}{N} \sum_{i=0}^{N-1} [y(i) - h_{\mathbf{a}}(\mathbf{x}(i))] \mathbf{x}(i)^T = -\frac{1}{N} \mathbf{X}^T (\mathbf{y} - \hat{\mathbf{y}}).$$

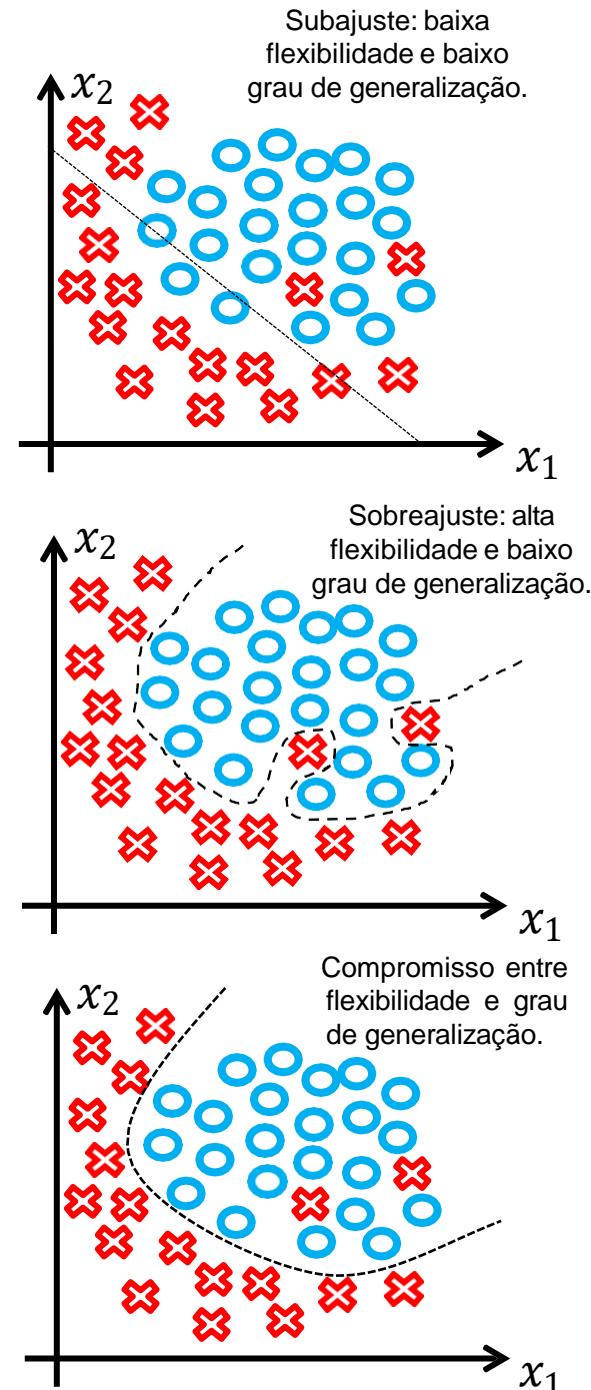
Forma matricial:
 $\mathbf{X} \in \mathbb{R}^{N \times K+1}$, \mathbf{y}
e $\hat{\mathbf{y}} \in \mathbb{R}^{N \times 1}$

- O **vetor gradiente** da **função de erro médio** para a **regressão logística** é idêntico àquele obtido para a **regressão linear** com a função de **erro quadrático médio**.
- Esse resultado pode ser utilizado com **funções discriminantes**, $g(x)$, lineares ou não-lineares, bastando apenas se alterar o formato da matriz de atributos, X .
- De posse do **vetor gradiente**, podemos usá-lo no algoritmo do **gradiente descendente** (nas versões em batelada, estocástico ou mini-batch).
- Assim, a atualização iterativa dos pesos é dada por

$$\mathbf{a} = \mathbf{a} - \alpha \frac{\partial J_e(\mathbf{a})}{\partial \mathbf{a}}.$$

Observações

- Como vimos, a **função discriminante**, $g(x)$, pode também assumir a forma de um **polinômio**, mas, muitas vezes, nós não sabemos qual a **melhor ordem** para este polinômio.
- Assim, como nós discutimos no caso da **regressão linear**, modelos de **regressão logística** usando **polinômios** também estão sujeitos à ocorrência de **sobreajuste** e **subajuste**.
 - Na primeira figura, a **falta de flexibilidade** da reta usada faz com que o erro de classificação seja alto.
 - Na segunda figura, a **flexibilidade excessiva** do modelo (explorando um polinômio de ordem elevada) dá origem a contorções na **fronteira de decisão** na tentativa de minimizar o erro de classificação junto aos dados de treinamento. Porém, o modelo ficou mais suscetível a erros de classificação para dados inéditos, ou seja, não irá generalizar bem.
 - Já a última figura mostra o que seria uma boa **hipótese de classificação**.
- Por isso, **técnicas de regularização** (e.g., LASSO, Ridge, Elastic-Net, Early-stopping) assim como **validação cruzada** também podem ser empregadas durante o treinamento quando não conhecemos a melhor ordem para o polinômio da **função discriminante**, $g(x)$.



Exemplo: Regressão Logistica com SciKit-Learn

```
# Import all necessary libraries.  
import pandas as pd  
from sklearn.linear_model.logistic import LogisticRegression  
from sklearn.feature_extraction.text import CountVectorizer  
from sklearn.model_selection import train_test_split  
  
# Read SMS data base with pandas.  
url = 'https://raw.githubusercontent.com/justmarkham/pycon-2016-tutorial/master/data/sms.tsv'  
sms = pd.read_table(url, header=None, names=['label', 'message'])  
  
# Split array into random train and test subsets.  
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=42)  
  
# Convert a collection of text documents into a matrix of token counts.  
vect = CountVectorizer()  
# Learn the vocabulary dictionary and return term-document matrix for the training set.  
x_train_dtm = vect.fit_transform(x_train)  
# Transform validation set into document-term matrix.  
x_test_dtm = vect.transform(x_test)  
  
# Instantiate Logistic classifier.  
classifier = LogisticRegression()  
# Train the model.  
classifier.fit(x_train_dtm, y_train.ravel())  
y_pred_class = classifier.predict(x_test_dtm)
```

Importa classe de regressão Logística.

Download da base de dados.

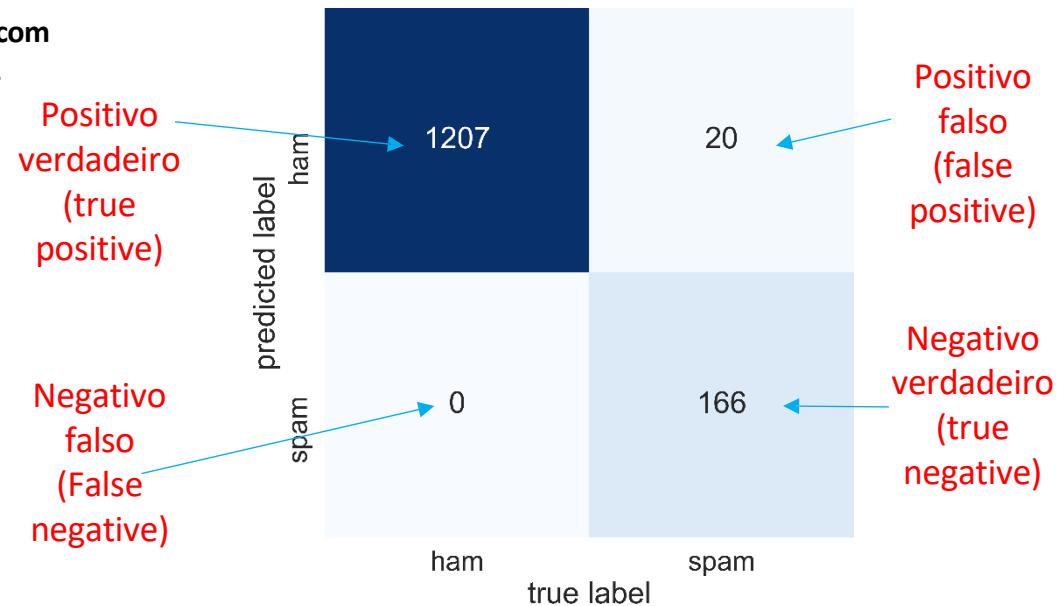
Divide a base de dados em 75% treinamento e 25% validação.

Converte as mensagens de treinamento em uma matriz com a frequência de cada palavra.

Converte as mensagens de validação em uma matriz com a frequência de cada palavra, baseado no vocabulário criado com o conjunto de treinamento.

Treinamento e validação do classificador.

- Classificação de mensagens entre SPAM e não-SPAM.
- Exemplo usa uma base de dados baixada do GitHub.
- Classifica as mensagens em 2 classes: 'SPAM' e 'HAM'.
- O objeto da classe **CountVectorizer** cria uma matriz registrando o número de vezes (frequência) com que cada palavra aparece na mensagem.
- A **matriz de confusão** mostra a performance do classificador.



Casos multi-classe

- Até agora, nós vimos como classificar utilizando **regressão logística** quando os dados pertencem a apenas 2 classes (i.e., $Q = 2$), mas e quando existem mais de 2 classes (i.e., $Q > 2$)? Por exemplo
 - Reconhecimento de dígitos escritos à mão: 10 dígitos.
 - Classificação de texto: Esportes, Economia, Política, Entretenimento, etc.
 - Classificação de sentimentos: Neutro, Positivo, Negativo.
- Existem algumas abordagens para a **classificação multi-classe**:
 - Um-Contra-o-Resto
 - Um-Contra-Um
 - Regressão Softmax
- As duas primeiras podem ser aplicadas a qualquer tipo de **classificador binário** e não apenas ao **regressor logístico**.
 - Outros classificadores binários: SVM, árvores de decisão, classificadores ingênuos de Bayes.
- A terceira abordagem é uma generalização do **classificador logístico** para problemas multi-classe.

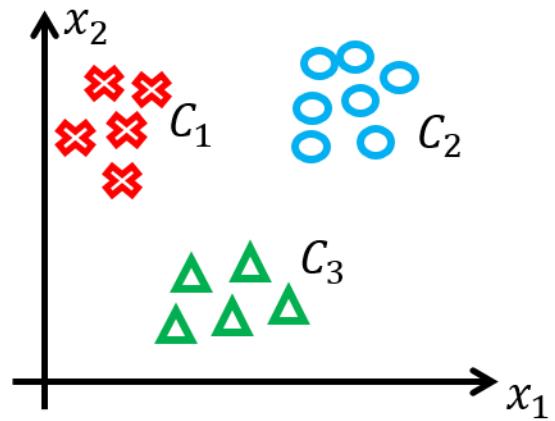
Um-Contra-o-Resto

- Nesta abordagem, treina-se **um classificador binário** (e.g., **regressor logístico**), representado pela função hipótese, $h_{\mathbf{a}}^q(x(i))$, para cada classe q , para predizer a probabilidade $P(C_q | x(i); \mathbf{a})$, $\forall q \in \{1, \dots, Q\}$, i.e., C_q é a classe positiva, C_2 .
- Em outras palavras, cria-se **Q classificadores binários**, onde para cada classificador, a classe positiva, C_2 , é a q -ésima classe e a classe negativa, C_1 , é a junção de todas as outras, $Q - 1$, classes.
 - *Transformamos um problema com Q classes em Q problemas binários.*
- Portanto, o q -ésimo **classificador** deve indicar a classe positiva caso o exemplo pertença à q -ésima classe, ou à classe negativa caso o exemplo pertença a qualquer outra classe.
- Após o treinamento, para cada exemplo de entrada, $x(i)$, realiza-se Q previsões e escolhe-se a classe que maximize $h_{\mathbf{a}}^q(x(i))$

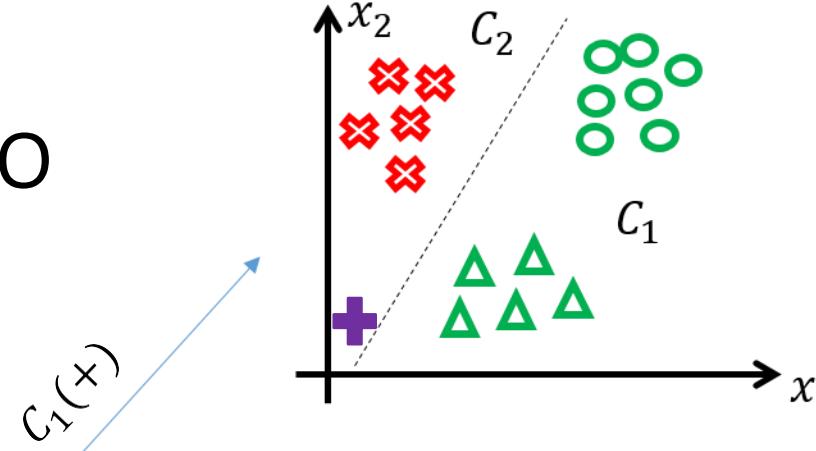
$$C_q = \arg \max_q h_{\mathbf{a}}^q(x(i)).$$

- A **vantagem** desta abordagem é que treina-se apenas Q **classificadores**.
- Uma **desvantagem** é que cada **classificador binário** precisa ser treinado com um conjunto negativo que é $Q-1$ vezes maior, o que pode aumentar o tempo de treinamento e a possibilidade de **classes desbalanceadas**.

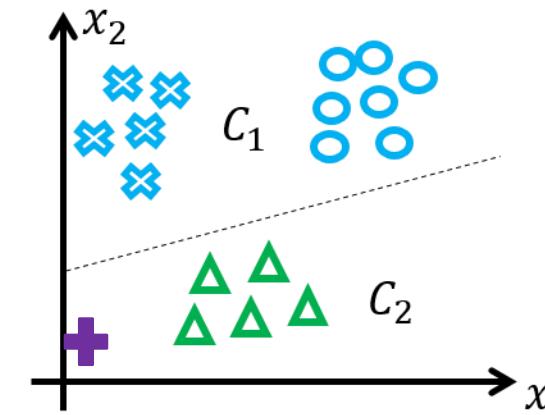
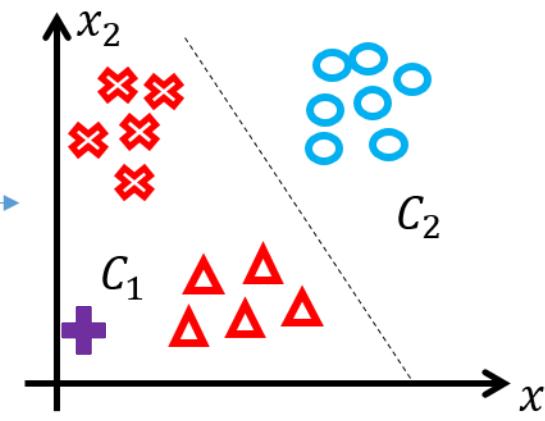
Um-Contra-o-Resto



Passamos a ter 3 classificadores binários



A qual classe + pertence?



Fronteira de decisão

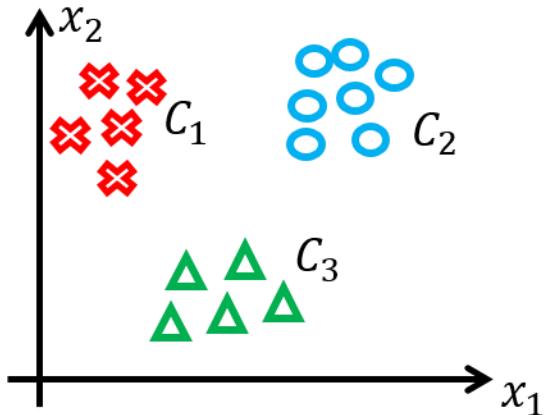


Exemplo de validação

Um-Contra-Um

- Nesta abordagem, treina-se $Q(Q - 1)/2$ **classificadores binários**.
- Cada **classificador** é construído para fazer a distinção entre exemplos pertencentes a cada um dos possíveis **pares** de classes.
 - **Transformamos um problema com Q classes em $Q(Q - 1)/2$ problemas binários.**
 - Se $Q = 4$, então treina-se 6 **classificadores** para classificar entre C_1/C_2 , C_1/C_3 , C_1/C_4 , C_2/C_3 , C_2/C_4 , e C_3/C_4 .
- No final, cada exemplo é classificado conforme o **voto majoritário** entre os **classificadores**.
- A principal **vantagem** da abordagem **Um-Contra-Um** é que cada **classificador** precisa ser treinado apenas com as duas classes que ele deve distinguir, portanto, a chance de desbalanceamento é reduzida.
- A desvantagem é que, por exemplo, se $Q = 10$, temos que treinar 45 **classificadores**.

Um-Contra-Um



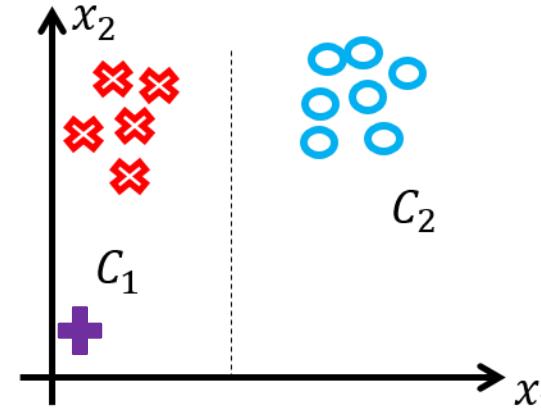
$$Q = 3$$

$$\frac{Q(Q-1)}{2} = 3$$

[Exemplo: ClassificationOfFourClassesWithOvAandOvO.ipynb](#)

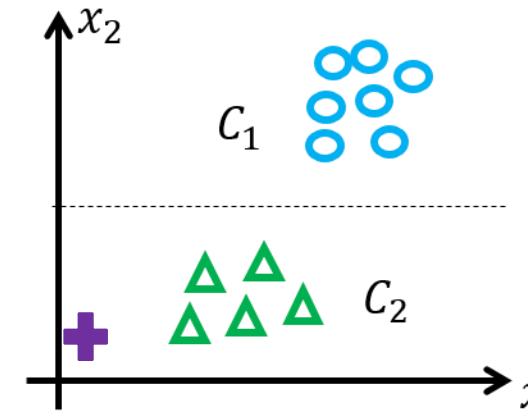
[Exemplo: ClassificationOfFourClassesWithOvAandOvO-SciKitLearn.ipynb](#)

$$\begin{matrix} C_2(+)/C_1(-) \\ C_3(+)/C_2(-) \\ C_1(+)/C_3(-) \end{matrix}$$

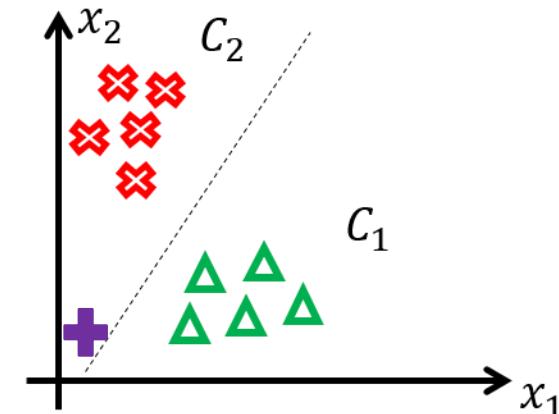


$$h_a^1(x(i)) = 0.05 \text{ (voto em } C_1\text{)}$$

A qual classe pertence?



$$h_a^2(x(i)) = 0.85 \text{ (voto em } C_3\text{)}$$



$$h_a^3(x(i)) = 0.55 \text{ (voto em } C_1\text{)}$$

----- Fronteira de decisão

Exemplo de validação

Regressão Softmax

- Também conhecida como **regressão logística multinomial**.
 - Porque as saídas do regressor podem ser interpretadas como as **probabilidades de uma variável categoricamente distribuída** (as classes) dado um conjunto de variáveis (atributos e pesos).
- É uma generalização do regressor logístico para problemas com **múltiplas classes**.
- A ideia é **treinar um único classificador com Q saídas**, onde **cada saída representa a probabilidade de um exemplo pertencer a uma das Q classes**.
 - Por exemplo, para um problema com 4 classes, teríamos um único classificador, mas com 4 saídas.
- Prediz **apenas uma classe de cada vez**, ou seja, ele é **multi-classe** e não **multi-label**, portanto, ele deve ser usado apenas com **classes mutuamente exclusivas**, como por exemplo diferentes tipos de plantas, dígitos, carros, etc.
 - **Classes mutuamente exclusivas**: exemplos devem pertencer a apenas uma das Q classes.
 - Já notícias e animais, por exemplo, podem pertencer a várias a várias classes.
- Para termos um **único** classificador, o **regressor softmax** possui uma **função hipótese de classificação**, $h_a^q(x(i))$, e uma **função discriminante**, $g_q(x(i))$, para cada classe q .

Regressão Softmax

- A ***q*-ésima função hipótese de classificação**, $h_{\mathbf{a}}^q(\mathbf{x}(i))$, é obtida passando-se a ***q*-ésima função discriminante**, $g_q(\mathbf{x}(i))$, através da **função softmax**,

Cada função discriminante tem seu próprio vetor de pesos.

$$P(C_q | \mathbf{x}(i); \mathbf{a}_q) = h_{\mathbf{a}}^q(\mathbf{x}(i)) = \frac{e^{g_q(\mathbf{x}(i))}}{\sum_{j=1}^Q e^{g_j(\mathbf{x}(i))}} = \frac{e^{\mathbf{x}(i)^T \mathbf{a}_q}}{\sum_{j=1}^Q e^{\mathbf{x}(i)^T \mathbf{a}_j}} \in \mathbb{R}[0,1],$$

O somatório de termos exponenciais normaliza o valor da *q*-ésima saída de tal forma que o somatório das *Q* saídas seja igual a 1.

onde $\mathbf{a}_q = [a_0^q \quad a_1^q \quad \dots \quad a_K^q]^T \in \mathbb{R}^{K+1 \times 1}$ é o **vetor de pesos** da *q*-ésima função discriminante, $g_q(\mathbf{x}(i))$, e *i* indica o número da amostra.

- Assim como com o regressor logístico, podemos usar **hiperplanos** ou **polinômios** como **funções discriminantes**.
- A **função softmax** atribui uma **probabilidade condicional**, $P(C_q | \mathbf{x}(i); \mathbf{a}_q)$, a cada classe, *q*, em um problema com múltiplas classes, onde a soma destas *Q* probabilidades deve ser igual a 1:

$$P(C_1 | \mathbf{x}(i); \mathbf{a}_1) + P(C_2 | \mathbf{x}(i); \mathbf{a}_2) + \dots + P(C_Q | \mathbf{x}(i); \mathbf{a}_Q) = 1.$$

Regressão Softmax

- Assim, o objetivo é encontrar um **modelo** (i.e., os **pesos** das Q funções hipótese) que atribua uma **alta probabilidade para a classe alvo** e, consequentemente, uma **baixa probabilidade para as demais classes**.
- Portanto, como fizemos anteriormente, precisamos definir uma **função de erro** e **minimizá-la** para encontrarmos os **pesos** das Q **funções hipóteses**.
- Usaremos a mesma função de erro definida para o regressor logístico, desta forma, a **função de erro médio** para a **regressão softmax** é dada por

$$J_e(\mathbf{A}) = -\frac{1}{N} \sum_{i=0}^{N-1} \sum_{q=1}^Q 1\{y(i) + 1 == q\} \log(h_{\mathbf{a}}^q(\mathbf{x}(i))),$$

O erro tende a 0 quando $h_{\mathbf{a}}^q(\mathbf{x})$ tende a 1, caso contrário, o erro aumenta

onde $1\{\cdot\}$ é a **função indicadora**, de modo que $1\{\text{uma condição verdadeira}\}=1$ e $1\{\text{uma condição falsa}\}=0$, $\mathbf{A} \in \mathbb{R}^{K+1 \times Q}$ é a matriz com os **pesos** das Q **funções hipótese** e $y(i)$ é o i -ésimo valor esperado.

- A matriz \mathbf{A} contém em suas **colunas** os vetores de pesos, \mathbf{a}_q , de cada uma das Q **funções discriminantes**.

Regressão Softmax

- Usando-se a codificação ***one-hot***, a equação anterior pode ser reescrita como

$$J_e(\mathbf{A}) = -\frac{1}{N} \sum_{i=0}^{N-1} \mathbf{y}(i)^T \log(\mathbf{h}_a(x(i))),$$

onde $\mathbf{y}(i) = [1\{y(i) + 1 == 1\}, \dots, 1\{y(i) + 1 == Q\}]^T \in \mathbb{R}^{Q \times 1}$ é o vetor utilizando a codificação ***one-hot*** e $\mathbf{h}_a(x(i)) \in \mathbb{R}^{Q \times 1}$ é o vetor com as saídas das Q ***funções hipóteses***

$$\begin{aligned}\mathbf{h}_a(x(i)) &= [h_a^1(x(i)), \dots, h_a^Q(x(i))]^T \\ &= [P(C_1 | x(i); \mathbf{a}_1), \dots, P(C_Q | x(i); \mathbf{a}_Q)]^T.\end{aligned}$$

- Notem que quando existem apenas duas classes ($Q = 2$), a ***função de erro*** acima é equivalente à ***função de erro médio*** do ***regressor logístico***.
 - Ou seja, o regressor softmax pode também ser usado quando $Q = 2$, caso binário.
- A ***função de erro médio não é linear*** e, portanto, ***não existe uma solução em forma fechada*** para encontrarmos os pesos. Porém, ela é ***convexa*** e, portanto, é garantido que o algoritmo do ***gradiente descendente*** encontre o mínimo global.

Regressão Softmax

- Agora que definimos uma **função de erro médio**, usamos o algoritmo do **gradiente descendente** para encontrar os **pesos** das Q funções discriminantes que a **minimizam**.
- A atualização iterativa dos **pesos** da q -ésima função discriminante, $g_q(\mathbf{x})$, é dada por

$$\mathbf{a}_q = \mathbf{a}_q - \alpha \frac{\partial J_e(\mathbf{A})}{\partial \mathbf{a}_q}, \forall q.$$

- Considerando uma **função discriminante linear** (i.e., **hiperplano**), a derivada parcial da **função de erro médio**, $J_e(\mathbf{A})$, com respeito a cada vetor de pesos, \mathbf{a}_q , tem uma expressão idêntica àquela obtida para a **regressão logística**:

$$\frac{\partial J_e(\mathbf{A})}{\partial \mathbf{a}_q} = -\frac{1}{N} \sum_{i=0}^{N-1} [1\{y(i) + 1 == q\} - h_{\mathbf{a}}^q(\mathbf{x}(i))] \mathbf{x}^T = -\frac{1}{N} \mathbf{X}^T (\mathbf{y}_q - \hat{\mathbf{y}}_q)$$

Forma matricial

Para outros formatos de função discriminante, basta alterarmos o formato da matriz \mathbf{X} .

onde \mathbf{y}_q e $\hat{\mathbf{y}}_q$ são vetores coluna com dimensão $N \times 1$.

[Exemplo: softmax_regressor_with_gradient_descent.ipynb](#)

Regressão Softmax: Observações

- O **vetor gradiente** da **função de erro médio** depende do formato da **função discriminante** adotada.
 - Entretanto, esta dependência afeta apenas a **matriz de atributos**, X .
 - Portanto, para outros formatos de **função discriminante**, basta alterar o formato da matriz de atributos, X .
- O regressor softmax apresenta duas propriedades:
 - $0 \leq h_a^q(x(i)) \leq 1$, ou seja, a saída da q -ésima função hipótese de classificação sempre será um valor dentro do intervalo $[0, 1]$;
 - $\sum_{q=1}^Q h_a^q(x(i)) = \sum_{q=1}^Q P(C_q | x(i), a_q) = 1$, ou seja, o somatório das **probabilidades condicionais** de todas as Q classes é igual a 1.
- Estas duas propriedades fazem com que o vetor

$$\mathbf{h}_a(x(i)) = [h_a^1(x(i)), \dots, h_a^Q(x(i))]^T \in \mathbb{R}^{Q \times 1},$$

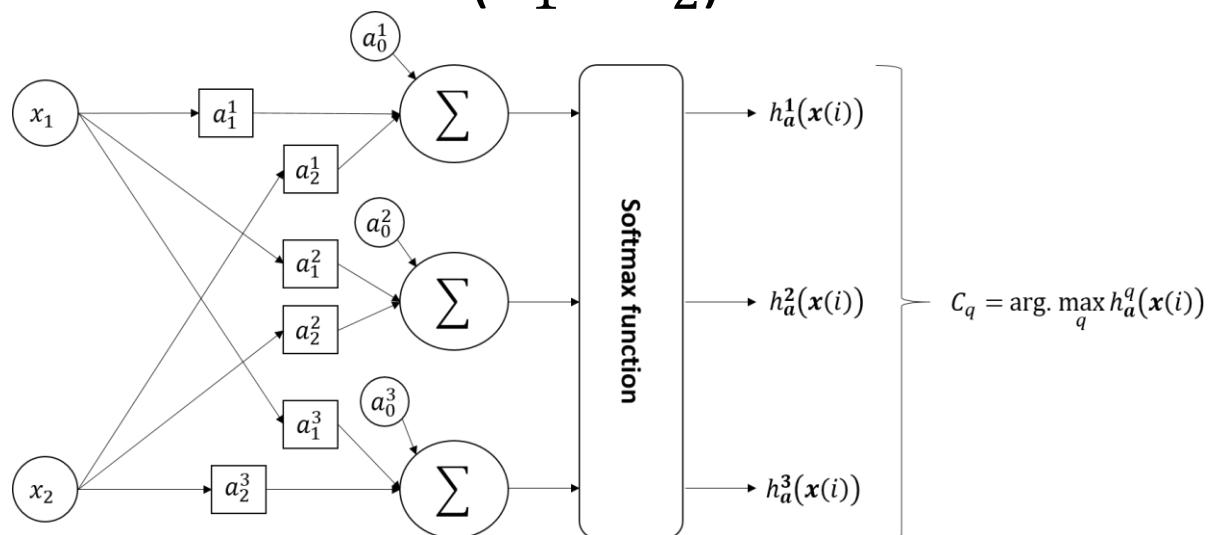
contendo todas as saídas do regressor softmax atenda aos requisitos de uma **função massa de probabilidade multinomial**.

Regressão Softmax

- Após o treinamento, durante a fase de predição, o classificador atribui ao i -ésimo exemplo de entrada, $\mathbf{x}(i)$, a classe, q , com a **maior probabilidade estimada**, que é simplesmente a classe com maior valor para $g_q(\mathbf{x}(i)) = \mathbf{x}(i)^T \mathbf{a}_q$

$$C_q = \arg \max_q h_{\mathbf{a}}^q(\mathbf{x}(i)) = \arg \max_q P(C_q \mid \mathbf{x}(i); \mathbf{a}_q) = \arg \max_q \mathbf{x}(i)^T \mathbf{a}_q .$$

- A arquitetura de um **regressor softmax** para três classes (i.e., $Q = 3$) e dois atributos (x_1 e x_2) é mostrada abaixo.



A ideia por trás da **regressão softmax** é bastante simples: dado um exemplo de entrada, \mathbf{x} , o regressor softmax primeiro calcula uma “**pontuação**”, $g_q(\mathbf{x}) = \mathbf{x}^T \mathbf{a}_q$, para cada classe q , em seguida, estima a probabilidade de cada classe aplicando a função softmax às “**pontuações**”, ou seja, as normaliza.

Exemplo: Regressão softmax com SciKit-Learn

```
# Import all necessary libraries.  
import matplotlib.pyplot as plt  
from sklearn.datasets import load_digits  
from sklearn.linear_model.logistic import LogisticRegression  
from sklearn.model_selection import train_test_split  
  
# Load digit data set.  
digits = load_digits()  
x = digits.data  
y = digits.target  
  
# Split the data set.  
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)  
  
# Instantiate LogisticRegression object.  
model = LogisticRegression(max_iter=10000, multi_class='multinomial')  
  
# Train model.  
model.fit(x_train, y_train)  
  
# Predict.  
y_pred = model.predict(x_test)
```

Carrega base de dados da biblioteca SciKit-Learn.

Importa classe de regressão Logística.

Divide a base de dados em 70% treinamento e 30% validação.

Instancia objeto da classe LogisticRegression para realizar regressão logistica Multinomial.

Treinamento e validação do classificador.

- Classificação de dígitos escritos à mão.
- Exemplo usa uma base de dados baixada do SciKit-Learn.
- Classifica os dígitos em 10 classes: 0 à 9.
- A **matriz de confusão** mostra a performance do classificador.

Predicted label	0	1	2	3	4	5	6	7	8	9
True label	53	0	0	0	0	0	0	0	0	0
0	53	0	0	0	0	0	0	0	0	0
1	0	47	0	0	1	1	0	0	0	0
2	0	1	47	1	0	0	0	0	0	0
3	0	0	0	52	0	0	0	0	0	1
4	0	0	0	0	59	0	0	0	0	0
5	0	0	0	1	0	63	1	1	1	0
6	0	0	0	0	0	1	52	0	0	0
7	0	0	0	0	0	0	0	54	0	0
8	0	2	0	0	0	0	0	0	42	2
9	0	0	0	0	0	1	0	0	0	56

- A classe **LogisticRegression** usa a abordagem **um-contra-todos** por padrão quando se treina com conjuntos com mais de duas classes, mas pode-se definir o hiperparâmetro **multi_class** como "**multinomial**" para alternar para Regressão Softmax.
- Deve-se também especificar um solver que suporte a Regressão Softmax, como por exemplo o solver "lbfgs" (consulte a documentação do Scikit-Learn). A classe **LogisticRegression** também aplica a regularização L2 por padrão, que pode ser controlada usando o hiperparâmetro **C**.

Métricas para avaliação de classificadores

- As métricas para avaliação do desempenho de classificadores que estudaremos são:
 - Taxa de erro e acurácia
 - Matriz de confusão
 - Várias métricas podem ser extraídas a partir da matriz.
 - Pontuação-F (*F-score*)
 - Curva Característica de Operação do Receptor (do inglês, *Receiver Operating Characteristic* - ROC)

Métricas para avaliação de classificadores

Taxa de erro e acurácia

- A **taxa de erro** é a métrica mais direta para se avaliar o desempenho de um classificador.
- Ela corresponde à **porcentagem de exemplos classificados incorretamente** considerando o conjunto de dados disponíveis para **validação**.
- A **taxa de erro** é dada por

$$p_e(\hat{y}(x)) = \frac{1}{N} \sum_{i=0}^{N-1} (1 - \delta(y(i), \hat{y}(x(i)))),$$

onde $\delta(i, j) = \begin{cases} 0, & \text{se } i \neq j \\ 1, & \text{se } i = j \end{cases}$ é o **delta de Kronecker**, $y(i)$ é o valor esperado e

$\hat{y}(x(i))$ é a saída do classificador. Observe que $p_e(\hat{y}(x)) \in [0, 1]$.

- O complemento da **taxa de erro** é conhecido como **acurácia**, e é definido por
- $$\text{acc}(\hat{y}(x)) = 1 - p_e(\hat{y}(x))$$

Métricas para avaliação de classificadores

Matriz de Confusão

- O nome, **matriz de confusão**, deriva do fato de que ela torna fácil verificar se o classificador está se **confundindo** (ou seja, **classificando incorretamente** os exemplos).
- A **matriz de confusão contabiliza o número de classificações corretas e incorretas** para cada uma das Q classes do problema.
- A **matriz de confusão**, $C \in \mathbb{R}^{Q \times Q}$, é definida como

$$C = \begin{bmatrix} C_{11} & C_{12} & \cdots & C_{1Q} \\ C_{21} & C_{22} & \ddots & C_{2Q} \\ \vdots & \ddots & \ddots & \vdots \\ C_{Q1} & C_{Q2} & \cdots & C_{QQ} \end{bmatrix}.$$

Quantidade de exemplos realmente pertencentes à classe 1.

Exemplos classificados como pertencentes à classe 1.

- C_{11} indica quantos exemplos da classe 1 foram corretamente atribuídos à classe 1.
- C_{12} indica quantos exemplos da classe 2 foram atribuídos à classe 1.

- A diagonal principal de C fornece o número de classificações corretas.
- A q -ésima **linha** indica o total de exemplos que foram classificados como pertencentes a q -ésima classe.
- A q -ésima **coluna** indica o total de exemplos realmente pertencentes à q -ésima classe.
- A informação apresentada na matriz permite verificar quais classes o **classificador** tem maior dificuldade em classificar.

Métricas para avaliação de classificadores

Matriz de Confusão

Exemplo para $Q = 2$.

Classes Preditas	+	Verdadeiro Positivo (TP)	Falso Positivo (FP)
	-	Falso Negativo (FN)	Verdadeiro Negativo (TN)
	+	-	
	C_2	C_1	Classes Verdadeiras

- **Verdadeiro Positivo (TP)**: número de exemplos da classe positiva, C_2 , classificados corretamente.
 - **Verdadeiro Negativo (TN)**: número de exemplos da classe negativa, C_1 , classificados corretamente.
 - **Falso Positivo (FP)**: número de exemplos classificados como positivos, mas que, na verdade, pertencem à classe negativa.
 - **Falso Negativo (FN)**: número de exemplos atribuídos à classe negativa, mas que, na verdade, pertencem à classe positiva.
- **IMPORTANTE:** Algumas definições que vamos precisar a seguir:
- N_+ define o número de exemplos pertencentes à classe positiva = TP + FN (coluna de C_2).
 - N_- define o número de exemplos pertencentes à classe negativa = FP + TN (coluna de C_1).
 - N define o número total de exemplos = TP + FN + FP + TN.

Métricas para avaliação de classificadores

Matriz de Confusão

Nós podemos calcular diversas métricas de desempenho a partir das informações contidas na **matriz de confusão**:

- **Taxa de falso negativo:** é a proporção de exemplos da classe positiva classificados incorretamente.

$$\text{Taxa de falso negativo} = p_e^+(\hat{y}(x)) = \frac{FN}{TP+FN} = \frac{FN}{N_+} \quad C = \begin{bmatrix} TP & FP \\ FN & TN \end{bmatrix}$$

- **Taxa de falso positivo:** é a proporção de exemplos da classe negativa classificados incorretamente.

$$\text{Taxa de falso positivo} = p_e^-(\hat{y}(x)) = \frac{FP}{TN+FP} = \frac{FP}{N_-} \quad C = \begin{bmatrix} TP & FP \\ FN & TN \end{bmatrix}$$

- **Taxa de erro:**

$$p_e(\hat{y}(x)) = \frac{FP + FN}{N}.$$

- **Acurácia:**

$$\text{acc}(\hat{y}(x)) = \frac{TP + TN}{N}.$$

Métricas para avaliação de classificadores

Matriz de Confusão

- **Precisão:** é a proporção de exemplos da classe positiva corretamente classificados (TP) em relação a todos os exemplos atribuídos à classe positiva (TP+FP).

$$\text{Precisão } (\hat{y}(x)) = \frac{\text{TP}}{\text{TP} + \text{FP}}. \quad C = \begin{bmatrix} \text{TP} & \text{FP} \\ \text{FN} & \text{TN} \end{bmatrix}$$

- **Sensibilidade (ou *recall*):** também conhecida como ***taxa de verdadeiros positivos***. É a proporção de exemplos da classe positiva corretamente classificados.

$$\text{Recall}(\hat{y}(x)) = \frac{\text{TP}}{\text{TP} + \text{FN}} = 1 - p_e^+(\hat{y}(x)). \quad C = \begin{bmatrix} \text{TP} & \text{FP} \\ \text{FN} & \text{TN} \end{bmatrix}$$

- **Especificidade:** também conhecida como ***taxa de verdadeiros negativos***. É a proporção de exemplos da classe negativa corretamente classificados.

$$\text{Especificidade } (\hat{y}(x)) = \frac{\text{TN}}{\text{TN} + \text{FP}} = 1 - p_e^-(\hat{y}(x)) \quad C = \begin{bmatrix} \text{TP} & \text{FP} \\ \text{FN} & \text{TN} \end{bmatrix}$$

Observações importantes quanto à matriz de confusão

- É possível estender as métricas obtidas com a **matriz de confusão** para o cenário multi-classes (i.e., $Q > 2$):
 - Por exemplo, usando a abordagem ***um-contra-o-resto***, basta selecionar, uma vez, cada classe $C_q, q = 1, \dots, Q$ como sendo a classe positiva, enquanto todas as demais classes formam a classe negativa. Assim, obtém-se os valores das métricas para cada uma das Q classes.
- Veja o exemplo abaixo para $Q = 3$, ou seja, C_1, C_2 e C_3 .

Classe C_1 é a positiva.

Classes Preditas	+ (C_1)	Verdadeiro Positivo (TP)	Falso Positivo (FP)	Falso Positivo (FP)
	- (C_2)	Falso Negativo (FN)	Verdadeiro Negativo (TN)	Verdadeiro Negativo (TN)
	- (C_3)	Falso Negativo (FN)	Verdadeiro Negativo (TN)	Verdadeiro Negativo (TN)
		+ (C_1)	- (C_2)	- (C_3)
Classes Verdadeiras				

Classe C_2 é a positiva.

Classes Preditas	- (C_1)	Verdadeiro Negativo (TN)	Falso Negativo (FN)	Verdadeiro Negativo (TN)
	+ (C_2)	Falso Positivo (FP)	Verdadeiro Positivo (TP)	Falso Positivo (FP)
	- (C_3)	Verdadeiro Negativo (TN)	Falso Negativo (FN)	Verdadeiro Negativo (TN)
		- (C_1)	+ (C_2)	- (C_3)
Classes Verdadeiras				

Classe C_3 é a positiva.

Classes Preditas	- (C_1)	Verdadeiro Negativo (TN)	Verdadeiro Negativo (TN)	Falso Negativo (FN)
	- (C_2)	Verdadeiro Negativo (TN)	Verdadeiro Negativo (TN)	Falso Negativo (FN)
	+ (C_3)	Falso Positivo (FP)	Falso Positivo (FP)	Verdadeiro Positivo (TP)
		- (C_1)	- (C_2)	+ (C_3)
Classes Verdadeiras				

Observações importantes quanto à matriz de confusão

- **Precisão** diz o *quão exato é o modelo em relação a todos os exemplos classificados como positivos*, ou seja, quantos deles são realmente positivos.

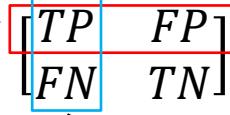
$$\text{Precisão} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$
$$C = \begin{bmatrix} TP & FP \\ FN & TN \end{bmatrix}$$

- A **precisão** é uma boa medida para determinar a qualidade de classificadores quando os custos de *falsos positivos* são altos.
 - Por exemplo, na classificação de *spams* (*verdadeiro positivo*), um *falso positivo* significa que um *ham*, email legítimo, (*verdadeiro negativo*) foi classificado como *spam*. O usuário de email pode perder emails importantes se a **precisão** for baixa.
- **Recall calcula quantas classificações positivas um classificador faz em relação a todos os exemplos positivos.**

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$
$$C = \begin{bmatrix} TP & FP \\ FN & TN \end{bmatrix}$$

- O **recall** é uma boa medida para determinar a qualidade de classificadores quando houver um alto custo associado a *falsos negativos*.
 - Por exemplo, na classificação de doenças, se um paciente doente (*verdadeiro positivo*) for classificado como não doente (*falso negativo*). O custo associado ao *falso negativo* será extremamente alto se a doença for contagiosa.

Observações importantes quanto à matriz de confusão

$$\text{Precisão} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$
$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$


- Uma **precisão** = 1 significa que todo exemplo classificado como pertencente à classe **positiva**, realmente pertence à ela, ou seja, o número de **falsos positivos** é igual a 0.
 - Entretanto, essa métrica não dá informações a respeito de quantos exemplos desta classe foram **classificados de forma incorreta**, ou seja, quantidade de **falsos negativos**.
- Por outro lado, um **recall** = 1 indica que todos os exemplos da classe positiva foram classificados como sendo pertencentes a ela, ou seja, o número de **falsos negativos** é igual a 0.
 - Porém, essa métrica não traz informações a respeito de **quantos exemplos da classe negativa foram classificados como sendo pertencentes à classe positiva**, ou seja, a quantidade de **falsos positivos**.
- Portanto, para analisarmos melhor o desempenho de um classificador, precisamos usar uma métrica que combine as duas.

Métricas para avaliação de classificadores

Pontuação-F

- A **precisão** e o **recall** podem ser analisados conjuntamente através de uma métrica que combina as duas, chamada de **pontuação-F** (ou **F-score**).
- Ela realiza uma **média harmônica ponderada** dada pela equação F_m abaixo:

$$F_m = \frac{(m + 1) \times \text{recall}(\hat{y}(x)) \times \text{precisão}(\hat{y}(x))}{\text{recall}(\hat{y}(x)) + m \times \text{precisão}(\hat{y}(x))}$$

onde $m \geq 0$ é o **fator de ponderação**.

- Quando $m = 1$, a **mesma importância** é dada para a **precisão** e para o **recall**:

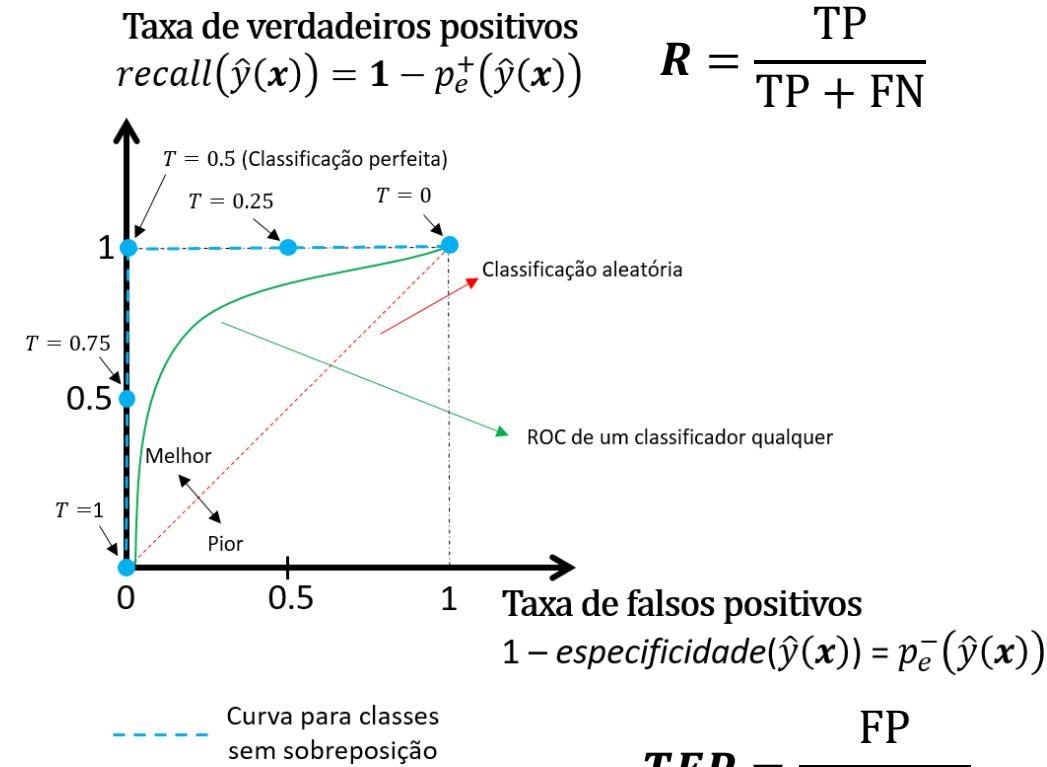
$$F_1 = 2 \frac{\text{recall}(\hat{y}(x)) \times \text{precisão}(\hat{y}(x))}{\text{recall}(\hat{y}(x)) + \text{precisão}(\hat{y}(x))} = \frac{\text{TP}}{\text{TP} + \frac{\text{FN} + \text{FP}}{2}}$$

- Valores de F_1 próximos de 1 indicam que o **classificador** obteve bons resultados tanto de **precisão** quanto de **recall**.
- Valores de $m > 1$ dão mais importância ao **recall** e valores de $m < 1$ dão mais importância à **precisão**.

Métricas para avaliação de classificadores

Curva Característica de Operação do Receptor (ROC)

- Gráfico que mostra a performance de um **classificador binário** conforme seu **limiar de discriminação (ou decisão)**, T , é variado.
- A curva é criada plotando-se o **recall** em função da **taxa de falsos positivos** para vários valores de T .
- Quanto mais à esquerda e para cima estiver a **curva ROC** de um **classificador**, melhor será o seu desempenho.
- A linha em vermelho, está associada a um **classificador puramente aleatório**. Um bom **classificador** fica o mais à esquerda possível dessa linha.
- Classes sem sobreposição apresentam uma curva ROC paralela aos eixos do recall e da TFP (linha azul tracejada).
- Classes sem sobreposição têm classificação perfeita quando $T = 0.5$, representando 100% de **recall** (ou seja, sem falsos negativos) e 100% de **especificidade** (ou seja, sem falsos positivos).
- Ela é robusta em relação ao desequilíbrio de classes, o que significa que pode ser usada efetivamente mesmo quando uma das classes é muito mais frequente do que a outra.



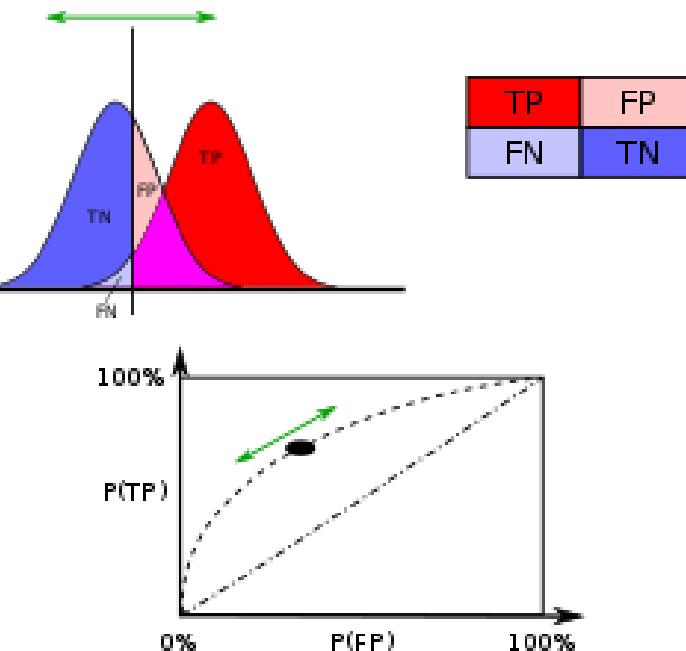
$$R = \frac{TP}{TP + FN}$$

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

Métricas para avaliação de classificadores

Curva Característica de Operação do Receptor (Curva ROC)

- A forma usual de se comparar **classificadores** consiste em criar uma **curva ROC** para cada um deles.
- Em geral, **classificadores** apresentam em sua saída uma probabilidade para cada exemplo de entrada.
- Normalmente, estas probabilidades são, então, discretizadas para que se tenha a decisão final: por exemplo, se o valor de $h_a(x(i))$ ultrapassa um determinado **limiar de decisão**, T , ele é mapeado no valor 1 (classe positiva, C_2); caso contrário, ele é mapeado no valor 0 (classe negativa, C_1).
- Sendo assim, ao plotarmos a **taxa de verdadeiro positivo** (ou **recall**) versus a **taxa de falso positivo** para diferentes valores de **limiar**, T , obtemos a **curva ROC** associada a um **classificador**.
- A forma da curva ROC é determinada pela sobreposição das duas distribuições das classes.



Métricas para avaliação de classificadores

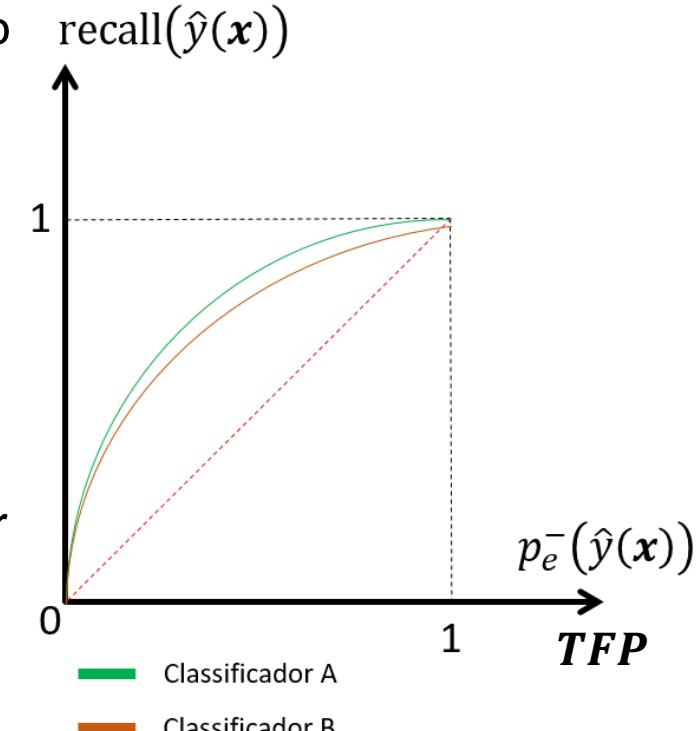
Curva Característica de Operação do Receptor (ROC)

- Por exemplo, considere as **curvas ROC** na figura ao lado. Para decidir qual o melhor **classificador**, podemos tomar como base a **área sob a curva (ASC)**.
- ASC** é outra métrica da qualidade de um classificador. É um número entre 0 e 1. Quanto maior a **ASC**, melhor será o classificador.
- Neste exemplo, o **classificador A** tem melhor desempenho, pois tem **área sob a curva ROC** maior do que a do **classificador B**.
- Vantagens da curva ROC**

- Permite comparar o desempenho de diferentes classificadores para o mesmo problema.
- Possibilita a análise de diferentes métricas de desempenho independente do **limiar de quantização, T**, escolhido.
- Auxilia na avaliação de diferentes **limiares** para lidar com problemas de **desbalanceamento e de diferentes custos de erros de classificação (FP/FN)**.
 - Aumentar o limiar resulta em menos falsos positivos (e mais falsos negativos) e vice versa.
 - Por exemplo, no caso desbalanceado, se a classe positiva é rara e os FP são mais custosos do que os FN, pode ser desejável escolher um limiar mais alto para reduzir a TFP.

- Desvantagens**

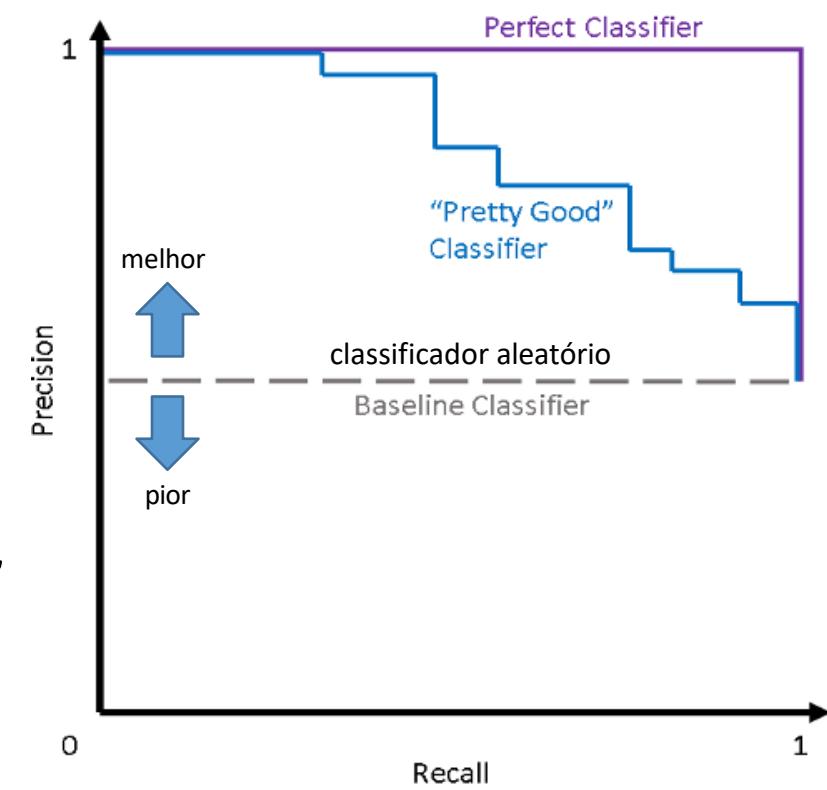
- Apropriada para problemas de **classificação binária**.
- No caso **multi-classes**, devemos utilizar as estratégias **um-contra-o-resto** ou **um-contra-um** e plotar várias **curvas ROC**.



Métricas para avaliação de classificadores

Curva de precisão-recall

- Assim como a curva ROC, a curva de **precisão-recall** é usada para avaliar o desempenho de classificadores binários.
- É frequentemente usada em situações em que as classes são **moderadamente ou fortemente desbalanceadas**.
- Assim como a curva ROC, fornece uma representação gráfica do desempenho de um classificador ao longo de vários valores para o **limiar de discriminação, T** .
- Ajuda a visualizar como a escolha do limiar afeta o desempenho do classificador, ajudando a selecionar o melhor limiar para um problema específico.



Exemplo: Métricas de classificação com SciKit-Learn

```
import numpy as np
from sklearn.datasets import make_blobs
from sklearn.linear_model.logistic import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import label_binarize
from sklearn.metrics import confusion_matrix, accuracy_score, auc, f1_score, roc_auc_score
from sklearn.metrics import classification_report, precision_score, recall_score
# make 3-class dataset for classification
centers = [[-5, 0], [0, 1.5], [5, -1]]
x, y = make_blobs(n_samples=1000, centers=centers, random_state=42)
# Split array into random train and test subsets.
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=23)
# Add column with ones regarding x0.
x_train = np.c_[np.ones([len(y_train), 1]), x_train]
x_test = np.c_[np.ones([len(y_test), 1]), x_test]
# Instantiate LogisticRegression object for multi-class case.
model = LogisticRegression(max_iter=10000, multi_class='multinomial')
# Train model.
model.fit(x_train, y_train)
# Predict.
y_pred = model.predict(x_test)
# Plot the confusion matrix.
confusion_matrix(y_test, y_pred)
# Getting the probabilities for each class.
y_prob = model.predict_proba(x_test)
# Binarize the test targets.
y_test_bin = label_binarize(y_test, classes=[0, 1, 2])
# Calculating ROC curve and ROC AUC only for class 0.
fpr, tpr, _ = roc_curve(y_test_bin[:, 0], y_prob[:, 0])
roc_auc = auc(fpr[0], tpr[0])
# Calculate metrics.
classification_report(y_test, y_pred)
accuracy_score(y_test, y_pred)*100
precision_score(y_test, y_pred, average=None)
recall_score(y_test, y_pred, average=None)
f1_score(y_test, y_pred, average=None)
```

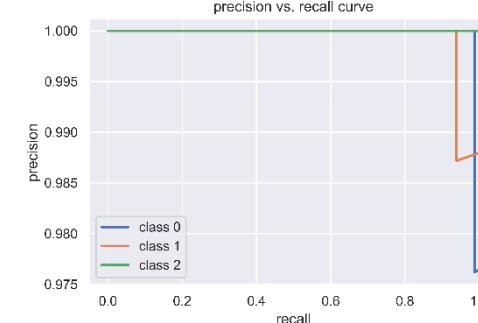
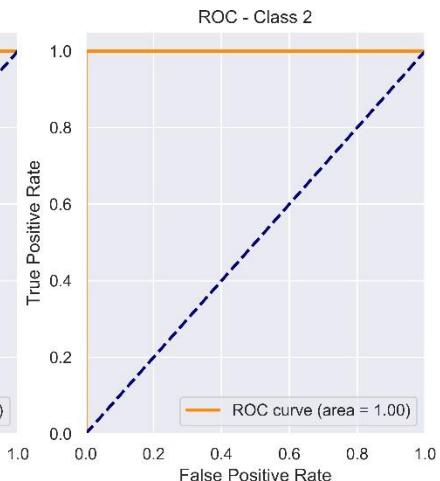
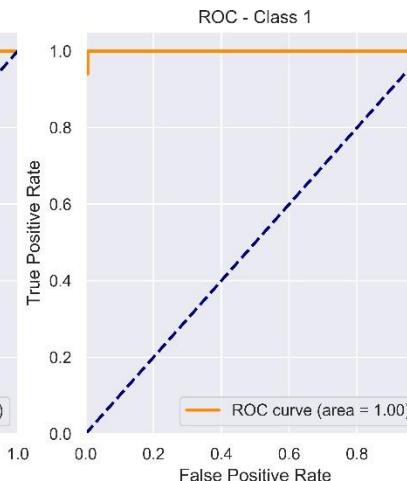
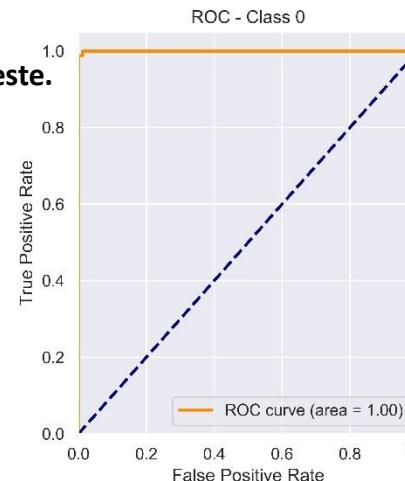
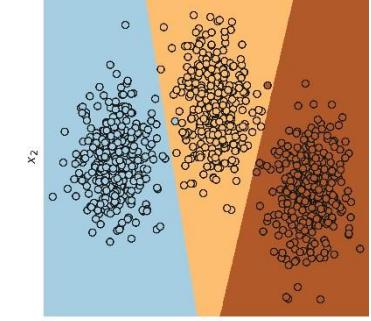
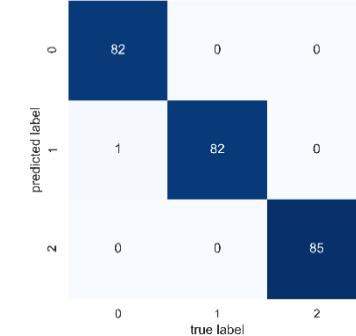
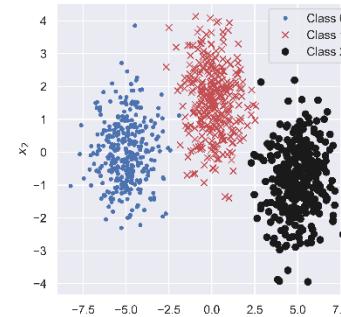
Cria 3 classes distintas.
Divide base de dados em treinamento e teste.

Treinamento e predição do classificador.
Cria objeto da classe LogisticRegression para múltiplas classes.

Cria matriz de confusão.

Curva ROC.

Calcula várias métricas.

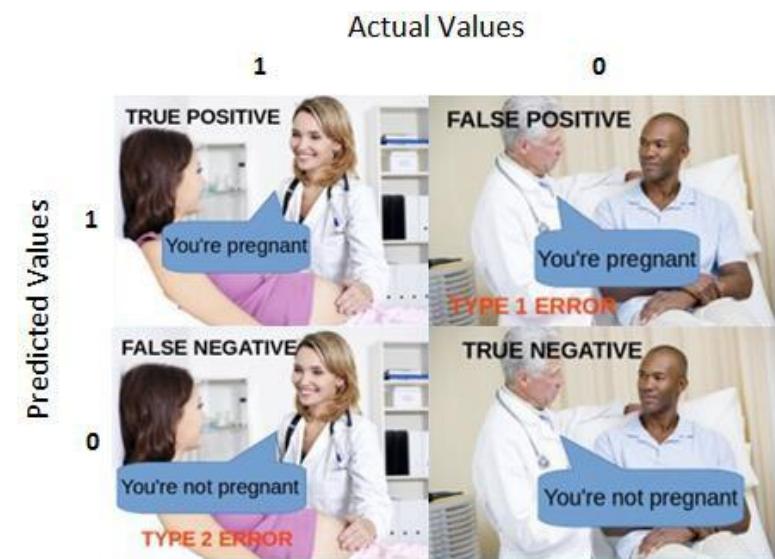
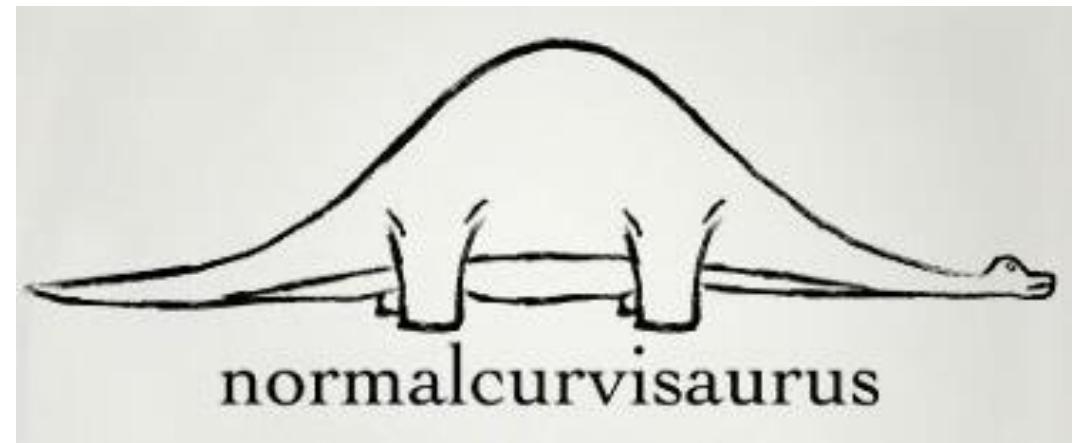
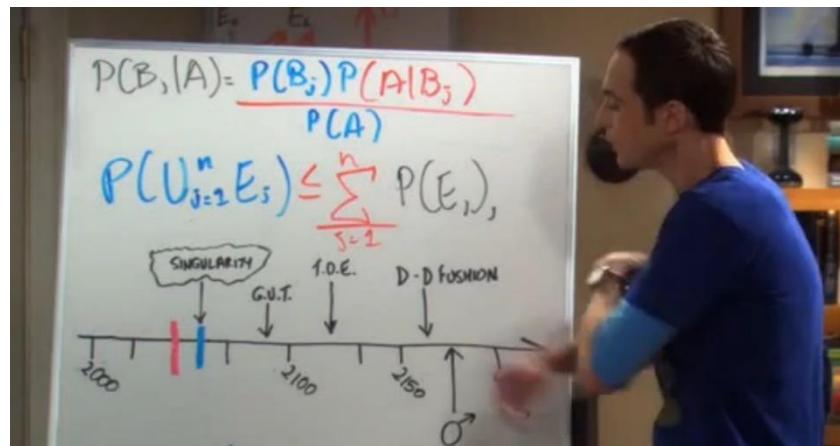
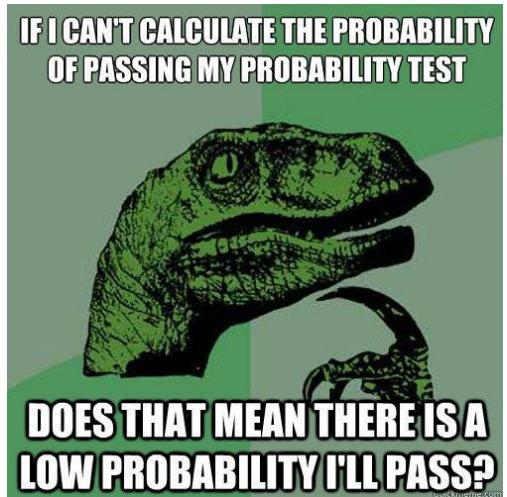


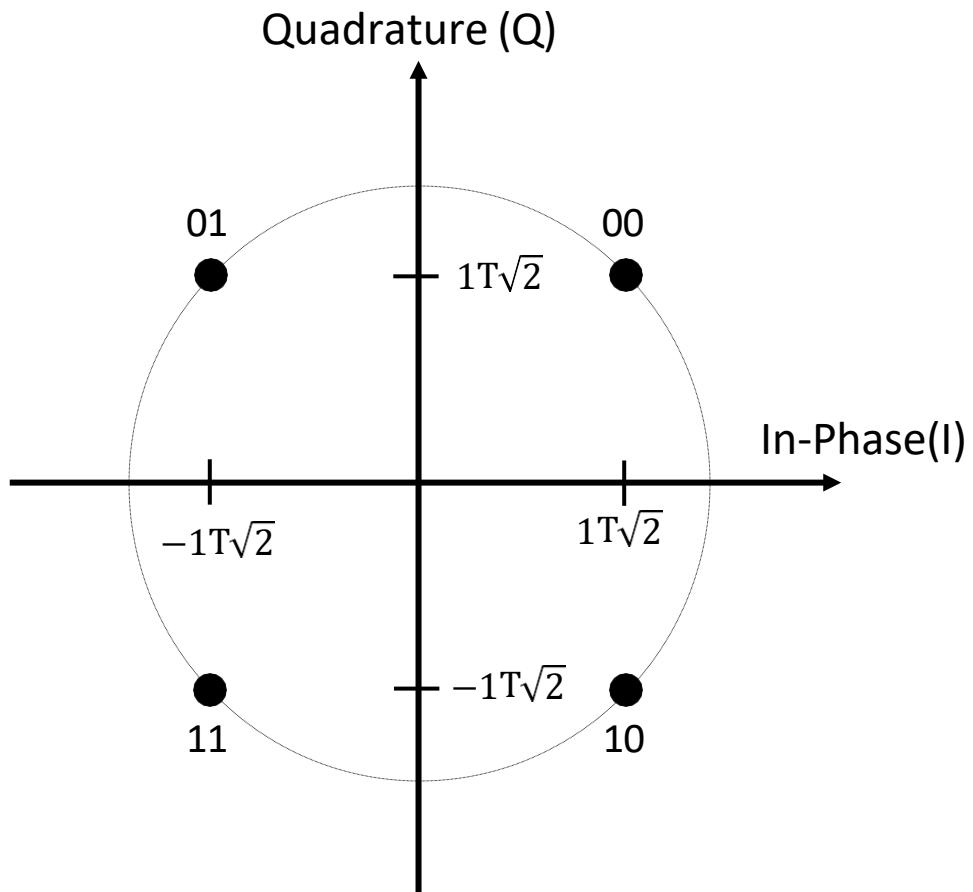
[Exemplo: classification_metrics.ipynb](#)

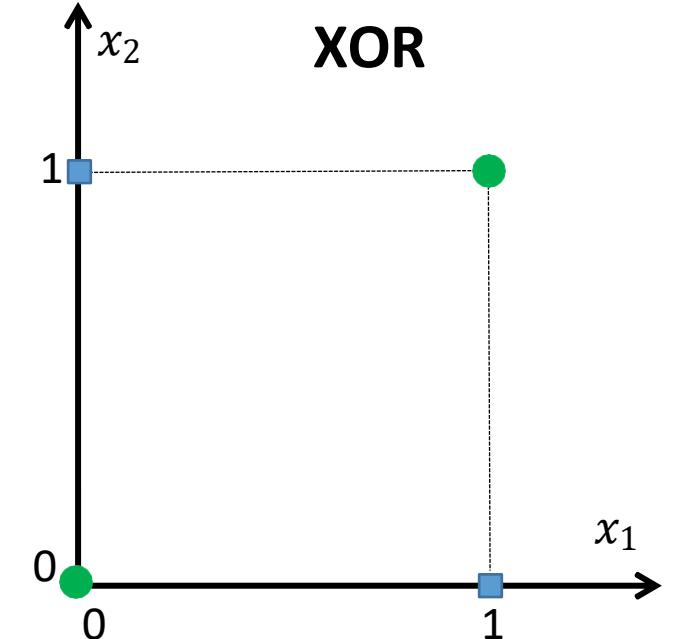
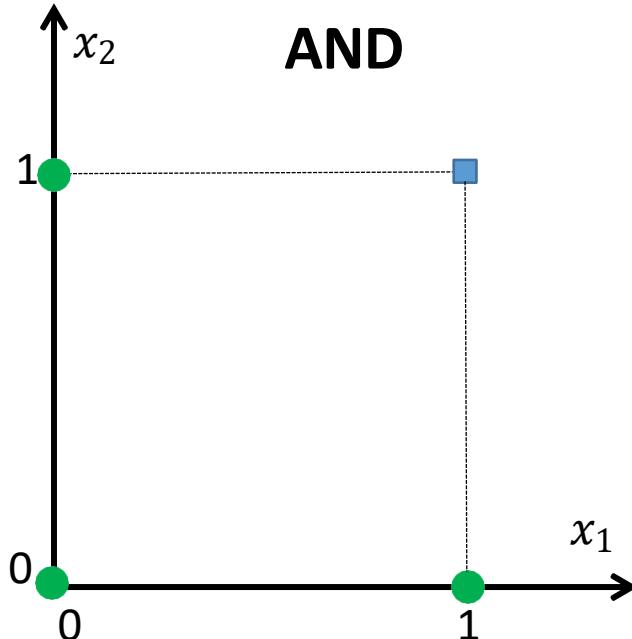
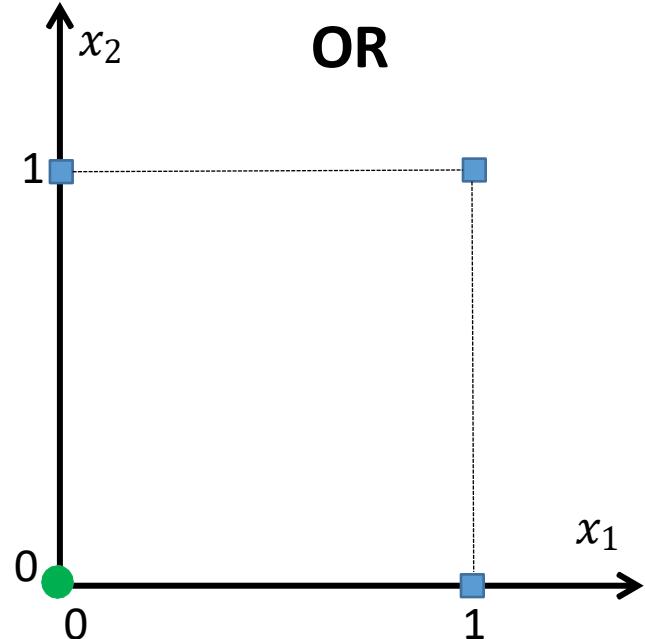
Tarefas e Avisos

- Você já podem fazer as listas #4 e #5.

Obrigado!



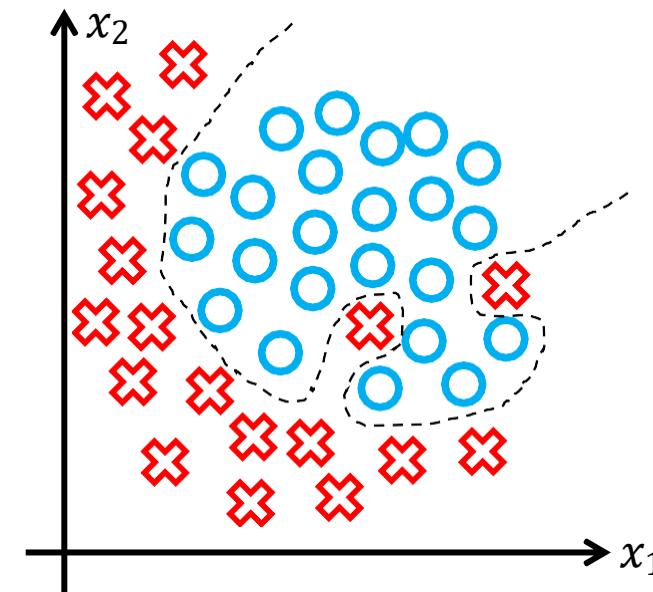
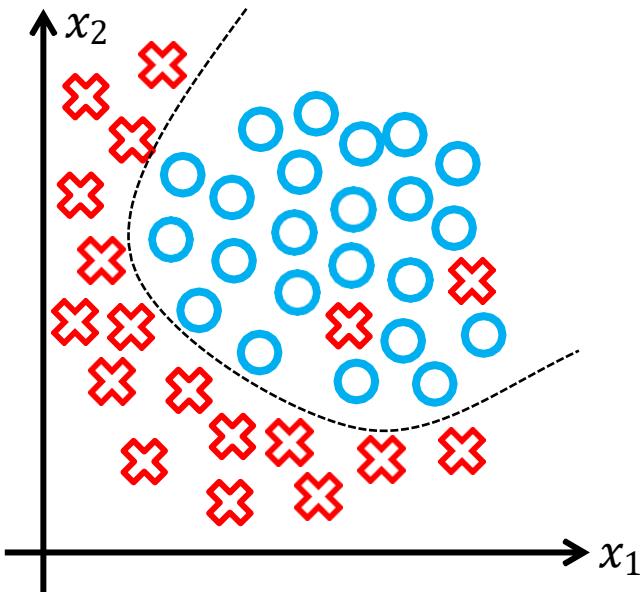
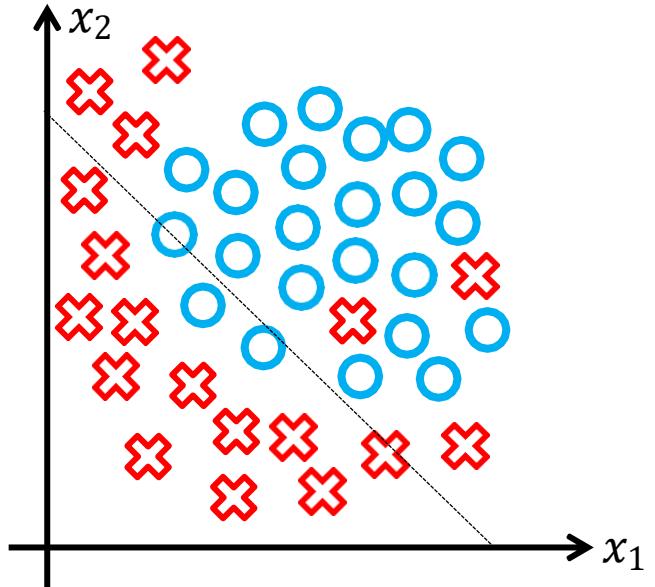


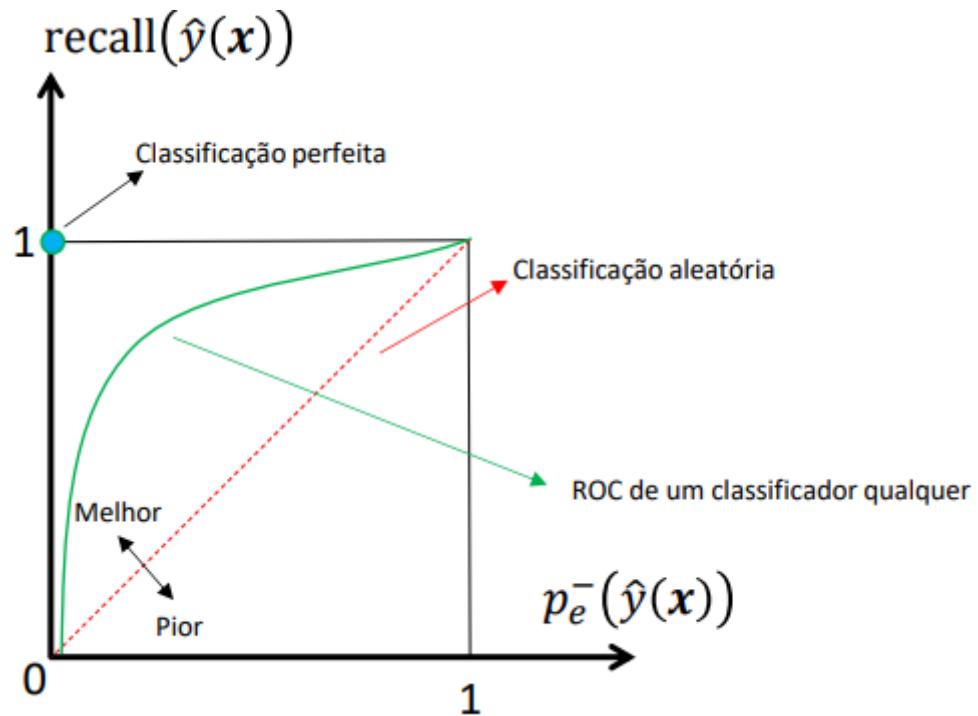


- Classe 0 (nível lógico 0)
- Classe 1 (nível lógico 1)

- Classe 0 (nível lógico 0)
- Classe 1 (nível lógico 1)

- Classe 0 (nível lógico 0)
- Classe 1 (nível lógico 1)





Taxa de verdadeiros positivos
 $recall(\hat{y}(\mathbf{x})) = \mathbf{1} - p_e^+(\hat{y}(\mathbf{x}))$

