

# TP555 - Inteligência Artificial e Machine Learning: *Classificação Linear*



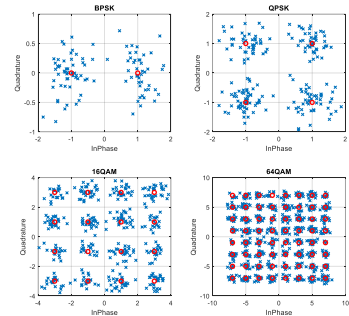
**Inatel**

Felipe Augusto Pereira de Figueiredo  
felipe.figueiredo@inatel.br

## Tópicos abordados

- Abordagens para classificação linear:
  - Classificação Bayesiana
  - Regressão logística
- Métricas para avaliação de classificadores.

## Motivação



- Classificação de emails entre SPAM e pessoal (HAM).
- Detecção de símbolos (classificação de símbolos).

Motivação: classificação de e-mails, detecção de símbolos

## Motivação

0 0 0 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 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  
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  
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



- Reconhecimento de dígitos escritos à mão.
- Classificação de texto.

## Definição do problema

**Problema:** atribuir a cada exemplo de entrada o **rótulo** correspondente a uma das  $Q$  classes existentes,  $C_q, q = 1, \dots, Q$ , à qual o exemplo pertence.

○ Este tipo de desafio é característico de problemas conhecidos como **classificação**.

- Semelhante ao problema da regressão linear, existe um conjunto de treinamento  $\{\mathbf{x}(i); y(i)\}_{i=0}^{N-1}$  que é utilizado para treinar um **classificador**, onde

- $\mathbf{x}(i) = [x_1(i) \ \dots \ x_K(i)]^T \in \mathbb{R}^{K \times 1}$  representa o  $i$ -ésimo vetor exemplo de entrada, o qual é caracterizado por  $K$  atributos,  $x_1, \dots, x_K$
- e  $y(i) \in \mathbb{R}$  representa o  $i$ -ésimo **rótulo**. Como veremos à seguir,  $y$  pode ser um escalar  $\mathbb{R}^1$  ou um vetor  $\mathbb{R}^{Q \times 1}$ .

## Representação da saída desejada

- A saída desejada para o exemplo de entrada deve ser o **rótulo** da classe à qual ele pertence.
- Sendo assim, a saída  $y$  de um **classificador**, é uma variável categórica (ou seja, discreta).
- Portanto, para realizarmos o treinamento do modelo, é necessário escolher uma **representação numérica** para a saída desejada, ou seja,  $y$ .
- Assim, como veremos a seguir, duas opções podem ser adotadas, dependendo do tipo de classificação a ser feita.

## Representação da saída desejada

- **Classificação binária:** existem apenas duas classes possíveis,  $C_1$  e  $C_2$ . Portanto, neste caso, podemos utilizar ***uma única saída escalar binária*** para indicar a classe correspondente ao exemplo de entrada:

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

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

## Representação da saída desejada

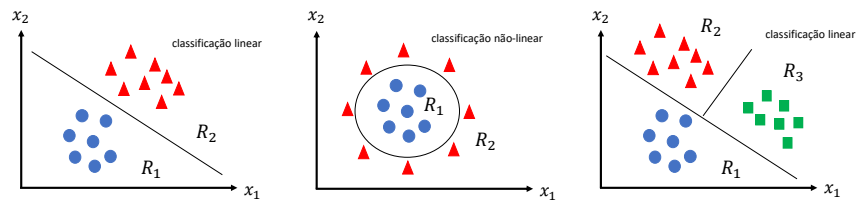
- **Classificação multi-classes:** existem mais de 2 classes possíveis ( $Q > 2$ ).
- Uma estratégia bastante utilizada para representar estas classes é conhecida como **one-hot encoding**.
- **one-hot encoding:** utiliza uma representação binária para cada uma das variáveis categóricas.
- Neste caso, o **classificador** produz múltiplas saídas, cada uma representando a **possibilidade** (ou probabilidade) do padrão pertencer a uma classe específica.
- **Exemplo:** imaginemos um classificador de texto com quatro classes possíveis: *esporte*, *política*, *ciências* e *variedades*. Como vocês as representariam com o **one-hot encoding**?

<i>esporte:</i>	$[1 \ 0 \ 0 \ 0]^T$	Assim, $\mathbf{y}(i) \in \mathbb{R}^{Q \times 1}$ , de maneira que o classificador realiza um mapeamento $\mathbb{R}^K \rightarrow \mathbb{R}^Q$ .
<i>política:</i>	$[0 \ 1 \ 0 \ 0]^T$	
<i>ciências:</i>	$[0 \ 0 \ 1 \ 0]^T$	
<i>variedades:</i>	$[0 \ 0 \ 0 \ 1]^T$	



## Fronteiras de decisão de um classificador

- O espaço de entrada  $\mathbb{R}^K$  é dividido em **regiões de decisão**  $R_i, i = 1, \dots, Q$ , as quais são delimitadas ou separadas pelas **fronteiras de decisão**, que correspondem a **superfícies** (ou **superfícies de decisão**) no espaço dos atributos onde ocorre uma indeterminação, ou, analogamente, um empate entre diferentes classes possíveis.
- As **fronteiras de decisão** podem ser **lineares** (e.g., retas e planos) ou **não-lineares** (e.g., círculos e esferas).



Regiões de decisão em problemas de classificação binária e multi-classes.

## 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 (ou grupo) ele pertence.
- Um **classificador linear** atinge esse objetivo tomando uma decisão de classificação com base no valor de uma **combinação linear** dos atributos.
- A saída de um classificador linear é dada por

$$y = f\left(\sum_{k=1}^K a_k x_k\right) = f(\mathbf{a}^T \mathbf{x}),$$

onde  $\mathbf{x} = [1, x_1, \dots, x_K]^T$  e  $f(\cdot)$  é a **função de limiar**, que é uma função que converte o produto escalar dos dois vetores,  $\mathbf{a}^T \mathbf{x}$ , na saída desejada, ou seja, na classe  $C_q$  do objeto.

- **Classificadores lineares** são frequentemente usados em situações em que a velocidade da classificação é um problema, pois ele geralmente é o classificador mais rápido.
- Além disso, os **classificadores lineares** geralmente funcionam muito bem quando o número de atributos é grande, como no caso da classificação de documentos.

## Teoria bayesiana de decisão

- A teoria bayesiana de decisão é uma **abordagem estatística** para o problema de **classificação**.
- Ela explora o conhecimento de probabilidades ligadas às classes e aos atributos, bem como dos **custos** associados a cada decisão, para realizar a classificação de cada novo exemplo.
- **Definições:** Considere que um exemplo (conjunto de atributos) a ser classificado seja descrito por um 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  $C_q$ .

## Teoria Bayesiana de decisão

- Agora suponha, que um exemplo  $x$  seja observado. De posse do conhecimento das características 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 específicos do exemplo,  $x$ .
  - Ou seja, a decisão é tomada em favor da classe cuja probabilidade **a posteriori** (ou seja, já levando em consideração o conhecimento do vetor de atributos) seja máxima.
  - A probabilidade **a posteriori** corresponde à probabilidade condicional  $P(C_q|x)$ .
  - Como calculamos  $P(C_q|x)$ ?
- **Teorema de Bayes**

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



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)

- A opção intuitiva sugerida anteriormente é conhecida como o critério ou decisor da **máxima probabilidade a posteriori** (MAP, do inglês **maximum a posteriori probability**), cuja decisão para o padrão  $\mathbf{x}$  é dada pela classe  $C_q$  que maximiza  $P(C_i|\mathbf{x})$ , ou seja, em forma matemática:

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

- Observe que, com base no teorema de Bayes, a solução para a equação acima é equivalente àquela que maximiza o numerador,  $P(\mathbf{x}|C_i)P(C_i)$ , de forma que:

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

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

O termo  $P(C_i|\mathbf{x})$  é chamado de **probabilidade a posteriori**

## Máxima verossimilhança (ML)

- O **decisor de máxima verossimilhança** (ML, do inglês *maximum likelihood*) parte do pressuposto de que não há informação estatística consistente sobre as classes, i.e., sobre  $P(C_i)$ .
- Portanto, o critério ML toma a decisão em favor da classe que apresenta o maior valor para a probabilidade  $P(x|C_i)$ . Neste sentido, o ML escolhe a classe  $C_q$  mais plausível ou mais verossímil em relação ao padrão observado:

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

- **OBS.:** se compararmos as expressões associadas aos critérios MAP e ML, percebemos que a diferença fundamental entre eles reside no fato de o MAP explicitamente incorporar o conhecimento das **probabilidades a priori**, i.e.,  $P(C_i)$ . Curiosamente, quando temos um cenário em que as classes são equiprováveis, i.e.,  $P(C_i) = 1/Q$  e independentes do índice,  $i$ . Então, maximizar a **probabilidade a posteriori** fornecerá 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 possuíam a doença (20%) e 80 estavam saudáveis (80%), sendo que dos indivíduos que possuíam a doença, 90% receberam positivo no teste da doença, e 30% deles que não possuíam a doença também receberam o teste positivo.

- **Pergunta:** Se uma novo indivíduo realizar o teste e receber um resultado positivo, qual a probabilidade de ele realmente possuir 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.

Podemos concluir que se o resultado do teste do indivíduo for positivo, ele possui aproximadamente 43% (0.429) de chance de ter a doença e que a chance dele não ter a doença mesmo tendo um teste positivo é de aproximadamente 57% (0.571).



## Classificador naíve 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

$$P(C_q | \mathbf{x} = [x_1 \ \dots \ x_K]^T) = \frac{P(\mathbf{x} | C_q)P(C_q)}{P(\mathbf{x})} = \frac{P(x_1 | C_q) \dots P(x_K | C_q)P(C_q)}{P(x_1) \dots P(x_K)}$$

- Com a independência dos atributos, os decisores 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 naíve Bayes incluem filtragem de spam, classificação de documentos e previsão de sentimentos.

O nome naíve (ingênuo em português) é usado porque assume que os atributos do modelo são 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 usados no algoritmo.

A independência dos atributos geralmente não é o caso de problemas do mundo real, onde os atributos podem ter relacionamentos complexos.

Teoria é baseada nos trabalhos do Reverendo Thomas Bayes (1702 a 1761).

Embora a independência dos atributos possa parecer uma suposição excessivamente simplista (ingênuo) em relação aos dados, na prática, o classificador naíve Bayes é competitivo com técnicas mais sofisticadas e possui suporte teórico para sua eficácia [1].

As aplicações típicas incluem filtragem de spam [2], classificação de documentos, previsão de sentimentos [3], etc.

**Classificação de textos/Filtragem de spam/Análise de sentimento:** classificadores Naíve Bayes utilizados principalmente em classificação de textos (devido a um melhor resultado em problemas de classes múltiplas e regra de independência) têm maior taxa de sucesso em comparação com outros algoritmos. Como resultado, é amplamente utilizado na filtragem de spam (identificar spam) e Análise de Sentimento (em análise de mídia social, para identificar sentimentos positivos e negativos dos clientes, identificar se o usuário está feliz ou triste ao publicar determinado texto)

**References:**

- [1] Zhang, H. (2004). The Optimality of Naive Bayes. *American Association for Artificial Intelligence*, 1-6.
- [2] Samahi, M. (1998). A Bayesian approach to filtering junk e-mail. *AAAI'98 Workshop on Learning for Text Categorization*, 1-8.
- [3] Choudhary, V. (2013). Vocal Emotion Recognition Using Naive Bayes Classifier. *Proceession of International Conference on Advances in Computer Science, AETACS*, 378-382.

## Classificador naïve Bayes

### Vantagens

- Fácil de ser implementado e altamente escalável.
- Funciona bem mesmo com poucos dados.
- Rápido para realizar as classificações, e portanto, pode ser utilizado em aplicações de tempo-real.
- Além de simples, ele é conhecido por apresentar performance 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 seja igual a zero, mas existem formas de se driblar esse problema (e.g., técnica da suavização de Laplace).
- É necessário se conhecer ou se assumir as probabilidades condicionais dos atributos.

Devido à suposição de independência, os classificadores naïve Bayes são altamente escaláveis e podem aprender rapidamente a usar atributos de alta dimensão com dados limitados de treinamento. Isso é útil para muitos conjuntos de dados do mundo real, onde a quantidade de dados é pequena em comparação com o número de atributos.

Como determinar a classe mais provável para um exemplo/amostra,  $\mathbf{x} = [x_1, x_2, \dots, x_K]^T$ , consiste em calcular o produto de  $K + 1$  fatores  $Q$  vezes, a notação big-O para a complexidade da classificação em tempo de execução é  $O(KQ)$ . Isso é computacionalmente muito eficiente e garante ao classificador naïve Bayes sua alta escalabilidade, uma vez que o tempo de execução é escalado linearmente no número de atributos  $K$  e no número de classes  $Q$ . Isso é especialmente útil com dados apresentando altas dimensões (ou seja,  $K$  é grande), como classificação de imagens de alta resolução, como, por exemplo, imagens de ressonância magnética [1].

Na prática, as probabilidades condicionais dos atributos de uma classe são geralmente modeladas usando-se o mesmo tipo de distribuição de probabilidade, como a distribuição binomial ou a distribuição Gaussiana.

Apesar de suas suposições aparentemente simplificadas, os classificadores naïve de Bayes têm funcionado muito bem em muitas situações/aplicação do mundo real, sendo famosa sua aplicação em classificação de documentos e filtragem de spam. Eles exigem uma pequena quantidade de dados de treinamento para estimar os parâmetros necessários. (Para as razões teóricas pelas quais classificadores naïve Bayes funcionam bem e em quais tipos de dados, consulte a referência [2] abaixo.)

### Referencias:

- [1] Al-Aidaroos, K. (2012). Medical Data Classification with Naive Bayes Approach. *Information Technology Journal*, 11, 1166-1174.
- [2] H. Zhang (2004). The optimality of Naive Bayes. Proc. FLAIRS.

## Tipos de classificadores naïve Bayes

- Na prática, as probabilidades condicionais 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 3 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 naíve Bayes Gaussiano

- 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,  $\mu_{x_k, C_q}$ , e a variância,  $\sigma_{x_k, C_q}^2$ , de cada atributo  $x_k$  em relação à classe,  $C_q$ , a que pertence.
- 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  $\mu_{x_k, C_q}$  e  $\sigma_{x_k, C_q}^2$ .

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

- Essa 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.

Até agora, vimos os cálculos quando os atributos ( $x_1, x_2, \dots, x_K$ ) são categóricos. Mas como calcular as probabilidades quando o atributo é uma variável contínua?

Se assumirmos que o vetor de atributos  $x = [x_1, \dots, x_K]^T$  segue uma distribuição específica, é possível utilizar a função de densidade de probabilidade dessa distribuição para calcular a probabilidade condicional.

Se você assumir que os atributos,  $x_1, \dots, x_K$ , seguem uma distribuição normal (também conhecida como gaussiana), o que é bastante comum, substituímos a densidade de probabilidade correspondente de uma distribuição normal e a chamamos de classificador naíve Bayes Gaussiano. Você precisa apenas da média e variância dos atributos para calcular as probabilidades condicionais.

O classificador naíve Bayes Gaussiano assume que os atributos seguem uma distribuição normal. Essa é outra suposição forte, pois muitos atributos não seguem uma distribuição normal. Embora isso seja verdade, supondo uma distribuição normal torna nossos cálculos muito mais fáceis. Portanto, usamos modelos gaussianos quando os atributos podem assumir infinitos valores.

## Exemplo: Probabilidade da prática de esportes

Nesse exemplo vamos usar o classificador Naive 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[temp.   jogar=sim]	22.78
std(temp.   jogar=sim)	3.42
E[temp.   jogar=não]	23.67
std(temp.   jogar=não)	4.39

Humidade [%]	
E[hum.   jogar=sim]	79.11
std(hum.   jogar=sim)	10.22
E[hum.   jogar=não]	86.20
std(hum.   jogar=não)	9.73

P(jogar=sim)	9/14
P(jogar=não)	5/14

$$P(\text{temp.}=25 \mid \text{jogar=sim}) = \frac{1}{3.42\sqrt{2\pi}} e^{-\frac{(25-22.78)^2}{2(3.42)^2}} = 0.0944 \quad P(\text{hum.}=62 \mid \text{jogar=sim}) = \frac{1}{10.22\sqrt{2\pi}} e^{-\frac{(62-79.11)^2}{2(10.22)^2}} = 0.0096$$

$$P(\text{temp.}=25 \mid \text{jogar=não}) = \frac{1}{4.39\sqrt{2\pi}} e^{-\frac{(25-23.67)^2}{2(4.39)^2}} = 0.0869 \quad P(\text{hum.}=62 \mid \text{jogar=não}) = \frac{1}{9.73\sqrt{2\pi}} e^{-\frac{(62-86.2)^2}{2(9.73)^2}} = 0.0019$$

Agora calculamos as probabilidades:

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

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

**OBS.:** a variância foi calculada como  $s^2 = \sum (x - \text{mean})^2 / (n - 1)$ , porém, alguns autores utilizam a seguinte definição  $s^2 = \sum (x - \text{mean})^2 / n$ .

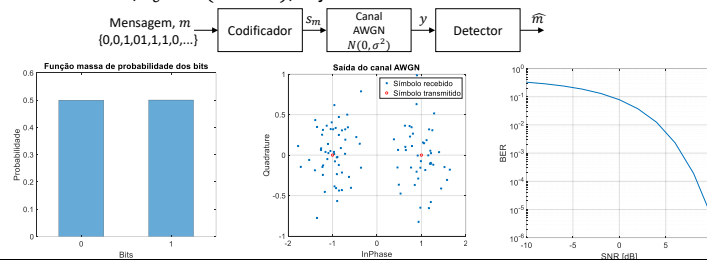
### Referência:

<https://mathworld.wolfram.com/SampleVariance.html>



## 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  $y = s_m + w$ , 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.



O detector MAP é o detector ideal para modulação BPSK:

<http://shannon.cm.nctu.edu.tw/digitalcom/Chap04.pdf>

<https://blogs.cornell.edu/info2040/2018/11/27/bayes-rule-in-digital-communication-with-bspk-modulation/>

### Referências:

[1] <https://dsp.stackexchange.com/questions/10222/qpsk-soft-decoding>

[2] <https://www.tutorialsp>

[3] <http://www.dsplog.com/2009/09/29/hamming-74-code-with-hard-decision-decoding/ojnt.com/hard-and-soft-decision-decoding>

[4] <https://www.gaussianwaves.com/2009/12/hard-and-soft-decision-decoding-2/>

### 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

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

O detector MAP é o detector ideal para modulação BPSK:

<https://blogs.cornell.edu/info2040/2018/11/27/bayes-rule-in-digital-communication-with-bpsk-modulation/>

#### Referências:

<https://ee.stanford.edu/~cioffi/doc/book/chap1.pdf>

<https://dsp.stackexchange.com/questions/10222/qpsk-soft-decoding>

<https://www.tutorialspoint.com/dsp/bpsk.htm>

<http://www.dsblog.com/2009/09/29/hamming-74-code-with-hard-decision-decoding/ojnt.com/hard-and-soft-decision-decoding>

<https://www.gaussianwaves.com/2009/12/hard-and-soft-decision-decoding-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()

# 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
    # Split array into random train and test subsets.
    x_test, x_train, y_test, y_train = train_test_split(x, y, random_state=42)
    # Fit.
    gnb.fit(y_train, x_train.ravel())
    # Predict.
    detected_x = gnb.predict(y_test).reshape(len(y_test), 1)
    # Simulated BPSK BER.
    ber_simu[idx] = 1.0 * ((x_test != detected_x).sum()) / len(y_test)
# Theoretical BPSK BER.
ber_theo[idx] = 0.5*erfc(np.sqrt(10.0**((EsN0dB[idx]/10.0))))
```

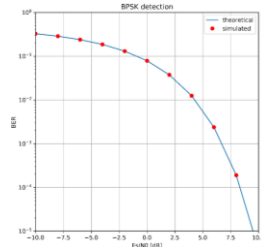
Importa classe GaussianNB do módulo naive\_bayes da biblioteca Scikit-Learn.

Instancia objeto da classe GaussianNB.

Divide o conjunto em treinamento e validação.

Treina o classificador com conjunto de treinamento.

Executa a classificação com conjunto de validação.



Exemplo: bpsk\_detection.ipynb

- Curva simulada se aproxima da teórica.



Exemplo: bpsk\_detection.ipynb

Link: [https://colab.research.google.com/github/zz4fap/tp555-machine-learning/blob/master/exemplos/classification/linear/bpsk\\_detection.ipynb](https://colab.research.google.com/github/zz4fap/tp555-machine-learning/blob/master/exemplos/classification/linear/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 que o evento  $k$  foi observado em uma instância específica.
- Este classificador é normalmente usado para classificação de documentos, com eventos representando a ocorrência de uma palavra no documento.
- A probabilidade de observar um histograma  $\mathbf{x}$  é dada por

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

onde  $p_{qk}$  é a probabilidade da classe  $C_q$  gerar o atributo  $x_k$ .

Outro classificador útil é o multinomialNB, onde se supõe que os atributos sejam gerados a partir de uma distribuição multinomial. A distribuição multinomial descreve a probabilidade de observar contagens entre várias categorias e, portanto, multinomialNB é mais apropriado para atributos que representam contagens ou taxas de contagem.

O classificador multinomial Naive Bayes (multinomialNB) é adequado para classificação onde os atributos são discretos (por exemplo, contagem de palavras para classificação de texto).

O classificador multinomialNB se baseia em uma distribuição discreta usada sempre que um atributo deva ser representado por um número inteiro (por exemplo, no processamento de linguagem natural, pode ser a frequência de um termo).

## 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
```

Base com 20 tópicos diferentes de discussão.

# Use the "20 Newsgroups corpus" from `sklearn` to show how we might classify these short documents into categories.

```
data = fetch_20newsgroups()
data.target_names
```

Treinamos e validamos com apenas 4 tópicos.

# 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)
```

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.

Matriz de confusão

Predicted \ True	comp.graphics	sci.space	soc.religion.christian	talk.religion.misc
comp.graphics	371	11	5	5
sci.space	11	372	4	11
soc.religion.christian	2	5	379	49
talk.religion.misc	5	1	10	195

Exemplo: [ClassifyingTextMultinomialNB.ipynb](#)

**Exemplo:** `ClassifyingTextMultinomialNB.ipynb`

**Link:** <https://colab.research.google.com/github/zz4fap/tp555-machine-learning/blob/master/exemplos/classification/linear/ClassifyingTextMultinomialNB.ipynb>

**Referência:** <https://jakevdp.github.io/PythonDataScienceHandbook/05.05-naive-bayes.html>

Evidentemente, mesmo esse classificador muito simples pode separar com sucesso a conversa sobre o **espaço** da conversa sobre **computadores**, mas fica um pouco confuso entre conversa sobre **religião** e conversa sobre **cristianismo**. Esta seja talvez uma área esperada de confusão.

## Classificador naïve Bayes Bernoulli

- Esse classificador é baseado na distribuição de Bernoulli, que é uma **distribuição binária**.
- Portanto, esse classificador considera que os **atributos** são **variáveis binárias** (i.e., booleanos) independentes, ou seja, o **atributo** pode estar presente (True) ou ausente (False).
- Assim como o classificador multinomial, esse classificador é utilizado para tarefas de classificação de documentos, onde atributos binários da ocorrência de termos são usados em vez da frequências de termos.
- Se  $x_k$  é um atributo booleano que expressa a ocorrência ou ausência do  $i$ -ésimo termo de um vocabulário de termos (i.e., palavras), então a probabilidade de um documento pertencer à classe  $C_q$  é dado por

$$P(\mathbf{x}|C_q) = \prod_k p_{qk}^{x_k} (1 - p_{qk})^{(1-x_k)},$$

onde  $p_{qk}$  é a probabilidade da classe  $C_q$  gerar o termo  $x_k$ .

- Esse classificador é bastante utilizado para classificar textos curtos e tem o benefício de classificar explicitamente a ausência de termos.

O classificador naïve Bayes Bernoulli (BernoulliNB) é baseado na distribuição de Bernoulli, que é uma distribuição binária. BernoulliNB é útil quando um atributo pode estar presente ou ausente. Esse classificador é utilizado quando estamos trabalhando com contagens discretas. Ele conta se um atributo ocorreu ou não.

Assim como o MultinomialNB, esse classificador é adequado para dados discretos. A diferença é que, enquanto o MultinomialNB trabalha com contagens de ocorrências, o BernoulliNB foi projetado para atributos binários/booleanos, ou seja, presença ou ausência do atributo.

## 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/junmarkham/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

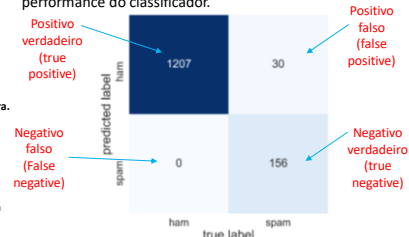
# Split array into random train and test subsets.
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

# 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)
```

- 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.

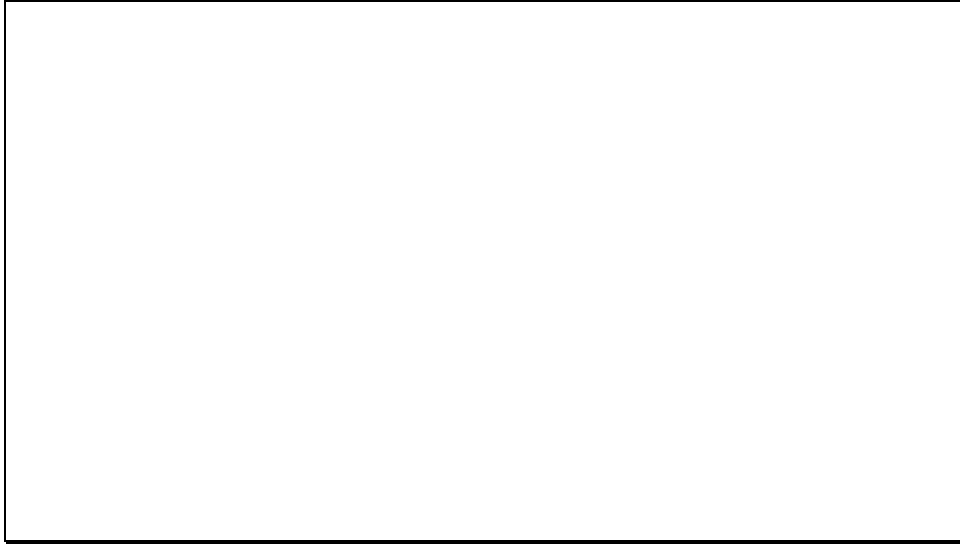


Exemplo: SPAMClassificationBernoulliNB.ipynb

**Exemplo:** SPAMClassificationBernoulliNB.ipynb

**Link:** <https://colab.research.google.com/github/zz4fap/tp555-machine-learning/blob/master/exemplos/classification/linear/SPAMClassificationBernoulliNB.ipynb#scrollTo=ARMnYAzud4fo>

Slide 30





## Tópicos abordados na aula de hoje

- Classificadores lineares com limiar rígido.
- Regressão logística.
- Métricas para avaliação de classificadores.

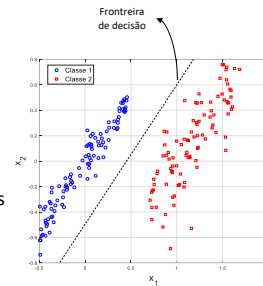
## Classificadores lineares com limiar de decisão rígido

- Funções lineares podem ser utilizadas em tarefas de **regressão** e **classificação**.
- Dado um conjunto de treinamento, a tarefa do classificador é a de **aprender** uma **hipótese**  $h$  que receberá um novo e desconhecido exemplo (e.g.,  $x_1$  e  $x_2$ ) e retornará **0** caso sua classe seja  $C_1$  (**classe negativa**) ou **1** caso ela seja  $C_2$  (**classe positiva**).
- Uma **fronteira de decisão** é uma linha (ou superfície, em altas dimensões) que separa as classes.
- No exemplo da figura ao lado, a **fronteira de decisão** é uma linha reta.
- Uma **fronteira de decisão linear** é chamada de **separador linear** e dados que admitem tal separador são chamados de **linearmente separáveis**.
- Portanto, para o exemplo ao lado, podemos definir a **hipótese de classificação** como

$$h_a(x) = 1 \text{ se } x^T a \geq 0 \text{ e } 0 \text{ caso contrário.}$$

- Alternativamente, nós podemos expressar  $h$  como sendo o resultado de passar a função linear  $a^T x$  através de uma **função de limiar rígido**  $f(\cdot)$

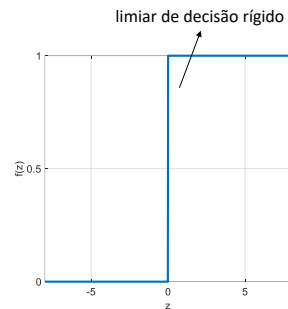
$$h_a(x) = f(x^T a), \text{ onde } f(z) = 1 \text{ se } z \geq 0 \text{ e } 0 \text{ caso contrário.}$$



## Classificadores lineares com limiar de decisão rígido

- A **função de limiar rígido** é mostrada na figura ao lado.
- Agora que a **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 o gradiente igual a zero e resolvendo a equação para os pesos; (ii) e através do **gradiente descendente**.
- Entretanto, com a **função de limiar rígido** que escolhemos e mostrada na figura, nenhuma das duas abordagens é possível devido ao fato do **gradiente** ser zero em todos os pontos do espaço de pesos exceto nos pontos onde  $x^T a = 0$ , e mesmo assim, o **gradiente** é indeterminado nesses pontos.

• Portanto, o que podemos fazer?



o classificador sempre anuncia uma previsão completamente confiante de 0 ou 1, mesmo para exemplos muito próximos do limite. Em muitas situações, precisamos realmente de previsões mais graduadas.

## Classificação linear com regressão logística

- Além do fato de que a hipótese  $h_a(x)$  não ser *diferenciável* e ser de fato, uma função *descontínua*, nós percebemos que com a *função de limiar rígida* o *classificador* sempre anuncia uma *previsão* completamente confiante de 0 ou 1, mesmo para exemplos muito próximos da *fronteira de decisão*.

- Em muitas situações, nós precisamos de previsões mais graduadas.

- Todos esses problemas podem ser resolvidos em grande parte com a *suavização* da *função de limiar rígida* através de sua aproximação com uma função que seja contínua e diferenciável.

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

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

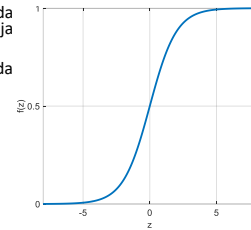
apresenta tais propriedades matemáticas.

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

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

- Perceba que a saída será um número entre 0 e 1, o qual pode ser interpretado como uma *probabilidade* de um dado exemplo pertencer à classe  $C_2$  (ou seja, a *classe positiva*).

- A hipótese  $h_a(x)$  forma uma *fronteira de decisão* suave, a qual confere a probabilidade de 0.5 para entradas no centro da região de decisão e se aproxima de 0 ou 1 conforme a posição do exemplo se distancia da fronteira.



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

A regressão logística, apesar de seu nome, é um modelo linear para classificação, em vez de regressão. A regressão logística também é conhecida na literatura como regressão logit, classificação de entropia máxima (MaxEnt) ou classificador log-linear. Nesse modelo, as probabilidades que descrevem os possíveis resultados de um único estudo são modeladas usando uma função logística.

A regressão logística é um método para classificação binária. Ela classifica os pontos de um conjunto de dados em duas classes ou categorias distintas.

Por simplicidade, vamos chamá-las de classe 1 e classe 2. O modelo nos dará a probabilidade de um determinado ponto (ou exemplo) pertencer à classe 2. Se a probabilidade for baixa (inferior a 50%), então o classificaremos na categoria 1. Caso contrário, ele cai na classe 2.

A regressão logística é ótima para situações em que você precisa classificar entre duas categorias/classes.

A regressão logística funciona usando uma combinação linear de atributos, para que várias fontes de informação possam governar a saída do modelo. Os parâmetros do modelo são os pesos dos vários atributos e representam sua importância relativa para o resultado.

Mesmo sendo uma técnica simples, a regressão logística é muito utilizada em aplicações do mundo real em áreas como medicina, propaganda, análise de crédito, saúde pública entre outras.



## Regressão logística

- A **regressão logística** (também chamada de **regressão logit**) é um método para **classificação binária**. Ela classifica os exemplos (ou pontos) de um conjunto de dados em duas classes distintas, ou seja, é ótima para situações em que você precisa classificar entre duas classes,  $C_1$  e  $C_2$ .
- A **regressão logística** estima a **probabilidade** de um exemplo pertencer a uma classe específica (por exemplo, qual é a probabilidade de um dado email ser spam?).
- Se a **probabilidade** estimada para o exemplo for maior que ou igual a 50%, o classificador prediz que o exemplo pertence a essa classe (denominada **classe positiva**, rotulada como 1), ou então prediz que não pertence (ou seja, pertence à **classe negativa**, rotulada como 0). Ou seja

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

- Note que  $Logistic(z) < 0.5$  quando  $z < 0$  e  $Logistic(z) \geq 0.5$  quando  $z \geq 0$ , portanto, o modelo de **regressão logística** prediz a classe  $C_2$  (i.e.,  $\hat{y} = 1$ ) se  $x^T a$  for positivo e  $C_1$  (i.e.,  $\hat{y} = 0$ ) se for negativo.
- Como vimos, a **regressão logística** funciona usando uma **combinação linear** de **atributos**, para que várias fontes de informação possam governar a saída do modelo. Os **parâmetros do modelo** são os **pesos** dos vários **atributos** e representam sua importância relativa para o resultado.
- Mesmo sendo uma técnica simples, a **regressão logística** é muito utilizada em várias aplicações do mundo real em áreas como medicina, marketing, análise de crédito, saúde pública entre outras.

Outra abordagem de classificação que promove a separação das classes com base em fronteiras de decisão lineares. Porém, diferentemente do que ocorre na regressão linear, a saída do modelo (ou classificador) é gerada a partir de um mapeamento não-linear (i.e., um mapeamento através da função de limiar).

Naive Bayes e a Regressão logística produzem assintoticamente o mesmo modelo se a suposição de Naive Bayes se mantiver. Isso é demonstrado no link abaixo:

<https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote05.html>

## Propriedades da regressão logística

- Os valores de saída da **função hipótese**  $h_a(\mathbf{x})$  ficam restritos entre o intervalo  $0 \leq h_a(\mathbf{x}) \leq 1$ .
- A saída de  $h_a(\mathbf{x})$  representa a **probabilidade** do vetor de atributos  $\mathbf{x}$  pertencer à classe positiva,  $C_2$ , para qual a saída desejada é  $y = 1$ . Ou seja,  $h_a(\mathbf{x}) = P(C_2 | \mathbf{x}; \mathbf{a})$ .
- Assim, consequentemente,  $(1 - h_a(\mathbf{x})) = P(C_1 | \mathbf{x}; \mathbf{a})$ .
- A **fronteira de decisão** é determinada quando 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  $P(C_2 | \mathbf{x}; \mathbf{a}) = h_a(\mathbf{x}) = 0.5$ .
- Observando a figura da **função logística**, nós percebemos que  $\text{Logistic}(z) = 0.5$  quando  $z = 0$ .
- Desta forma, a **fronteira de decisão** é caracterizada por

$$\mathbf{x}^T \mathbf{a} = a_0 + a_1 x_1 + \dots + a_K x_K = 0,$$

e corresponde a um **hiperplano**.

- A saída do classificador,  $\hat{y}$ , já discretizada, é dada por

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

## Função de erro

- Adotar o **erro quadrático médio** como **função de erro** não é uma escolha muito acertada para a **adaptação dos pesos** no caso da **regressão logística**.
- A **função de erro** utilizando o **erro quadrático médio** é dada por

$$J_e(\mathbf{a}) = \frac{1}{N} \sum_{i=1}^N (y(i) - h_{\mathbf{a}}(\mathbf{x}))^2 = \frac{1}{N} \sum_{i=1}^N (y(i) - \text{Logistic}(\mathbf{x}^T \mathbf{a}))^2.$$

- Como  $\text{Logistic}(\cdot)$  é uma função **não-linear**,  $J_e(\mathbf{a})$  não será consequentemente uma função convexa, de forma que a **superfície de erro** pode apresentar mínimos locais que vão dificultar o aprendizado.
- **Ideia:** adotar uma função que melhor se adapte às características do problema de tal forma que a **superfície de erro** resultante seja **convexa**.
- Uma proposta intuitiva para a **função de erro** é dada por

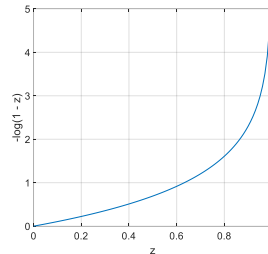
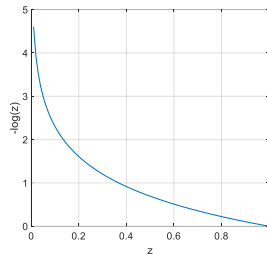
$$\text{Erro}(h_{\mathbf{a}}(\mathbf{x}); y) = \begin{cases} -\log(h_{\mathbf{a}}(\mathbf{x})), & \text{se } y = 1 \\ -\log(1 - h_{\mathbf{a}}(\mathbf{x})), & \text{se } y = 0 \end{cases}$$

O objetivo do treinamento é definir o vetor de pesos  $\mathbf{a}$  para que o modelo atribua valores altos de probabilidade para exemplos positivos (i.e.,  $y \geq 0.5$ ) e valores baixos de probabilidade para exemplos negativos (i.e.,  $y < 0.5$ ).

Essa função de erro faz sentido porque  $-\log(z)$  se torna muito grande quando  $z$  se aproxima de 0, então o erro será grande se o modelo estimar uma probabilidade próxima a 0 para um exemplo positivo e também será muito grande se o modelo estimar uma probabilidade próxima a 1 para um exemplo negativo. 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 0 para um exemplo negativo ou próxima de 1 para um exemplo positivo, que é exatamente o que queremos.



## Função de erro



As figuras ao lado mostram as duas situações possíveis para a **função de erro**. Como podemos observar, a penalização aplicada a cada saída reflete o **erro de classificação**.

- 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.

- Na figura da esquerda, o custo é nulo somente se a saída é  $h(\mathbf{a} \cdot \mathbf{x}) = 1$  (ou seja, se a classe  $C_2$  é corretamente identificada). À medida que  $h(\mathbf{a} \cdot \mathbf{x}) \rightarrow 0$ , o custo tende a infinito.
- Na figura da direita, o custo é nulo quando a saída é  $h(\mathbf{a} \cdot \mathbf{x}) = 0$  (ou seja, quando a classe  $C_1$  é corretamente identificada). Conforme  $h(\mathbf{a} \cdot \mathbf{x}) \rightarrow 1$ , o custo tende a infinito.

## Função de erro

- Nós podemos reduzir a definição da **função de erro** a uma expressão única, dada por

$$Erro(h_a(x); y) = \frac{-y \log(h_a(x))}{\text{Só exerce influência no erro se } y=1} + \frac{-(1-y) \log(1-h_a(x))}{\text{Só exerce influência no erro se } y=0}.$$

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

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

- A má notícia aqui é que não existe uma **equação de forma fechada** conhecida para calcular o valor dos pesos  $\mathbf{a}$  que minimize essa **função de erro** (ou seja, não há um equivalente da **Equação Normal**).
- A boa notícia é que essa **função de erro** 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 você espere tempo suficiente).

A função de erro para todo o conjunto de treinamento é simplesmente o erro médio para todos os exemplos de treinamento.

Ela pode ser escrita em uma única expressão como mostrado acima.

## Processo de treinamento

- Semelhante ao que fizemos com a **regressão linear**, vamos apresentar em seguida o algoritmo do **gradiente descendente** para a **minimização** da **função de erro** apresentada anteriormente.
- Antes de encontrarmos o **vetor gradiente** de  $J_e(\mathbf{a})$ , vamos reescrever a **função de erro** utilizando as seguintes equivalências

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

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

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

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

### Referência:

<https://math.stackexchange.com/questions/477207/derivative-of-cost-function-for-logistic-regression>

## Processo de treinamento

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

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

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

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

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

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

## Processo de treinamento

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

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

- O **vetor gradiente** do segundo termo é dado por

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

- Portanto, combinando os 2 resultados acima, temos que o **vetor gradiente** da **função de erro** é dado por

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

- Depois de obter o **vetor gradiente**, podemos usá-lo no algoritmo do **gradiente descendente em batelada**.
- Para o **gradiente descendente estocástico**, usamos apenas um exemplo de cada vez, e para o **gradiente descendente em mini-batch**, usamos um mini-batch por vez.
- A atualização dos pesos é dada por

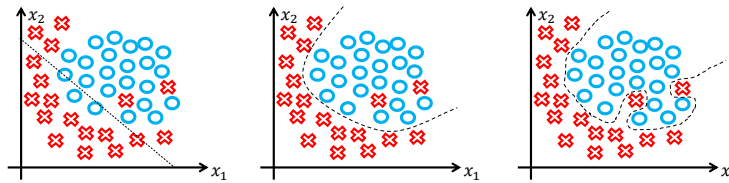
$$\mathbf{a} = \mathbf{a} - \alpha \frac{\partial J_e(\mathbf{a})}{\partial \mathbf{a}}.$$

**Exemplo:** SPAMClassificationLogisticRegressionGD.ipynb

**Link:** <https://colab.research.google.com/github/zz4fap/tp555-machine-learning/blob/master/exemplos/classification/linear/logistic/SPAMClassificationLogisticRegressionGD.ipynb>

## Observações

- A expressão do **vetor gradiente** da **função de erro**, para a **regressão logística** é similar àquela obtida para a **regressão linear** com o critério dos **quadrados mínimos**.
- A **regressão logística** também pode ser facilmente estendida para incorporar termos **polinomiais** envolvendo os atributos de entrada (e.g.,  $x_1^2, x_2^2$ ). Desta forma, pode-se produzir **fronteiras de decisão não-lineares**.
- Assim como nós discutimos no caso da **regressão linear**, modelos de **regressão logística** também estão sujeitos à ocorrência de **sobreajuste** e/ou **subajuste**. Por isso, **técnicas de regularização** podem ser empregadas em seu treinamento, assim como **validação cruzada** e **early stopping**.
  - Na figura mais à direita, 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 susceptível a erros de classificação para novos dados.
  - Na figura mais à esquerda, a falta de flexibilidade da reta usada faz com que o erro de classificação seja alto.
  - Já a figura do meio mostra o que seria uma boa **hipótese de classificação**.



## Exemplo: Regressão Logística com SciKit-Learn

```
# Import all necessary libraries.
import pandas as pd
from sklearn.linear_model 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()
x_train_dtm = vect.fit_transform(x_train)
# Learn the vocabulary dictionary and return term-document matrix for the training set.
# Transform validation set into document-term matrix.
x_test_dtm = vect.transform(x_test)

# Instantiate Logistic classifier.
classifier = LogisticRegression(solver='lbfgs')
# Train the model.
classifier.fit(x_train_dtm, y_train.ravel())

# Trainamento e validação do classificador.
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.

- 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) que cada palavra aparece na mensagem.
- A **matriz de confusão** mostra a performance do classificador.

predicted label	ham	spam
	1207	20
actual label	ham	spam
	0	166

Positivo verdadeiro (true positive) → 1207

Positivo falso (false positive) → 20

Negativo falso (false negative) → 0

Negativo verdadeiro (true negative) → 166

Exemplo: [SPAMClassificationLogisticRegressionSciKit.ipynb](#)

**Exemplo:** SPAMClassificationLogisticRegressionSciKit.ipynb

**Link:** <https://colab.research.google.com/github/zz4fap/tp555-machine-learning/blob/master/exemplos/classification/linear/logistic/SPAMClassificationLogisticRegressionSciKit.ipynb>

**OBS.:** Assim como os outros modelos lineares, os modelos de Regressão Logística podem ser regularizados usando penalidades de L1 ou L2. O Scikit-Learn adiciona uma penalidade L2 por padrão.

## Casos multi-classe

- Até agora nós vimos como classificar com a ***regressão logística*** quando os dados pertencem a apenas 2 classes (i.e.,  $Q = 2$ ), mas e quando existem mais de 2 classes (i.e.,  $Q > 2$ )?
- Existem algumas abordagens para classificação multi-classe:
  - Um-contra-todos
  - Um-contra-um
  - Regressão softmax



## Um-Contra-Todos

- Nesta abordagem, nós treinamos um **classificador** de **regressão logística** (ou seja, um **classificador binário**)  $h_a(\mathbf{x}(i))^{(q)}$  para cada classe  $q$  para prever a probabilidade de  $y = q$ , ou seja,  $P(y = q | \mathbf{x}; \mathbf{a})$ .
- Em outras palavras, cria-se  $Q$  **classificadores binários** onde a classe positiva  $C_2 = q$  e a classe negativa  $C_1$  é a junção de todas as outras  $Q - 1$  classes.
- Portanto, o **classificador** deve indicar a classe positiva caso o exemplo pertença à classe  $q$ , e a classe negativa caso o exemplo pertença a qualquer outra classe.
- Para cada novo exemplo de entrada,  $\mathbf{x}$ , realiza-se as predições e escolhe-se a classe que maximize

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

- Vantagem desta abordagem é que se treina apenas  $Q$  **classificadores**.
- 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.

## 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.
  - Se  $Q = 4$ , então treina-se **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 na parte do conjunto de treinamento para as duas classes que ele deve distinguir.
- A desvantagem é que por exemplo, se  $Q = 10$ , temos que treinar 45 **classificadores**.

## Regressão softmax

- Também conhecida como **regressão logística multinomial**.
- Uma abordagem mais robusta que as anteriores consiste em montar um modelo que produza saídas, em que cada saída representa a probabilidade de cada exemplo pertencer a uma classe específica.
- Isto pode ser feito a partir de uma generalização da **regressão logística**, explorando a **função softmax**, que é definida por

$$P(C_q | \mathbf{x}(i)) = h_a^q(\mathbf{x}(i)) = \frac{e^{\mathbf{x}(i)^T \mathbf{a}_q}}{\sum_{t=1}^Q e^{\mathbf{x}(i)^T \mathbf{a}_t}},$$

onde  $\mathbf{a}_q = [a_0^q \ a_1^q \ \dots \ a_K^q]^T$  é o vetor de pesos associado à  $q$ -ésima saída e  $h_a^q(\mathbf{x}(i))$  é a **função hipótese** associada à  $q$ -ésima classe.

- Para cada novo exemplo de entrada,  $\mathbf{x}$ , realiza-se as predições e escolhe-se a classe que maximiza

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

- A **função de erro** é dada por

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

onde  $y^q(i)$  é a probabilidade de que o  $i$ -ésimo exemplo pertença à classe  $q$ . Em geral,  $y^q(i)$  é igual a 1 ou 0, dependendo se o exemplo pertence à classe ou não e  $\mathbf{A} \in \mathbb{R}^{K+1 \times Q}$  é a matriz com os pesos para todas as  $Q$  classes.

O classificador Softmax prevê/prediz apenas uma classe de cada vez (ou seja, ele é multiclasse, e não multi-saída), portanto, ele deve ser usado apenas com classes mutuamente exclusivas, como por exemplo diferentes tipos de plantas, dígitos, categorias de notícias, etc. Portanto, você não pode usá-lo para reconhecer várias pessoas em uma foto, por exemplo.

Observe que, quando existem apenas duas classes ( $Q = 2$ ), a função de erro acima é equivalente à função de erro da regressão logística.

## Propriedades da regressão softmax

- $\sum_{q=1}^Q h_a^q(\mathbf{x}(i)) = 1$ , ou seja, o somatório da probabilidade de todas as classes é igual a 1.
- $0 \leq h_a^q(\mathbf{x}(i)) \leq 1$ , ou seja, temos, um vetor  $\mathbf{h}_a(\mathbf{x}(i)) = [h_a^1(\mathbf{x}(i)) \ \dots \ h_a^Q(\mathbf{x}(i))] \in \mathbb{R}^{Q \times 1}$  que atende os requisitos de uma **função probabilidade de massa** (PMF, do inglês **probability mass function**).
- A derivada de  $J_e(\mathbf{A})$  com respeito a cada vetor de pesos  $\mathbf{a}_q$  segue uma expressão semelhante àquela obtida para a **regressão logística**:

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

O objetivo do treinamento do classificador softmax é ter um modelo que estima uma alta probabilidade para a classe-alvo (e consequentemente uma baixa probabilidade para as outras classes).

A classe `LogisticRegression` da biblioteca Scikit-Learn usa a estratégia um-contra-todos por padrão quando você o treina com dados pertencentes a mais de duas classes, mas você pode definir o parâmetro **mult\_class** como *"multinomial"* para alternar para regressão Softmax. Você também deve especificar um **solver** que suporte a regressão Softmax, como o **solver** "lbfgs" (consulte a documentação do Scikit-Learn para obter mais detalhes). Ele também aplica regularização L2 por padrão, que você pode controlar usando o parâmetro **C**.

## 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 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(solver='lbfgs', max_iter=10000, multi_class='multinomial')

# Train model.
model.fit(x_train, y_train)

# Predict.
y_pred = model.predict(x_test)
```

Importa classe de regressão Logística.

Carrega base de dados da biblioteca SciKit-Learn.

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

Instancia objeto da classe LogisticRegression 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.

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

Exemplo: MultiClassDigitClassification. ipynb

**Exemplo:** MultiClassDigitClassification. ipynb

**Link:** <https://colab.research.google.com/github/zz4fap/tp555-machine-learning/blob/master/exemplos/classification/linear/logistic/MultiClassDigitClassification.ipynb>

## Métricas de avaliação da classificação

### Taxa de erro e acurácia

- A **taxa de erro**, é intuitivamente, 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

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

onde  $\delta(i, j) = \begin{cases} 0, & \text{se } i \neq j \\ 1, & \text{se } i = j \end{cases}$  é o delta de Kronecker. Observe que  $p_e(\hat{y}(\mathbf{x})) \in [0, 1]$ .

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

## Métricas de avaliação da classificação

### Matrix de Confusão

- A **matriz de confusão**  $\mathbf{C} \in \mathbb{R}^{Q \times Q}$  contabiliza o número de classificações corretas e incorretas para cada uma das  $Q$  classes existentes.

$$\mathbf{C} = \begin{bmatrix} C_{ii} & C_{ij} \\ C_{ji} & C_{jj} \end{bmatrix},$$

onde o elemento  $C_{ij}$  indica quantos padrões da classe  $i$  foram designados à classe  $j$ . Portanto, a diagonal de  $\mathbf{C}$ , nos fornece o número de classificações corretas.

- A informação apresentada nesta matriz permite verificar quais classes o **classificador** tem maior dificuldade em classificar.

Uma matriz de confusão, também conhecida como tabela de contingência ou matriz de erros, é um layout de tabela específico que permite a visualização do desempenho de um classificador. Cada coluna da matriz representa as instâncias em uma classe prevista, enquanto cada linha representa as instâncias em uma classe real. O nome deriva do fato de tornar fácil ver se o classificador está confundindo duas classes (ou seja, geralmente rotulando incorretamente uma como a outra).

## Métricas de avaliação da classificação

### Matrix de Confusão

Exemplo para  $Q = 2$ .

Classe Estimada	+	Verdadeiro Positivo (TP)	Falso Positivo (FP)
	-	Falso Negativo (FN)	Verdadeiro Negativo (TN)
		+	-
		$C_2$	$C_1$
		Classe Verdadeira	

- **Verdadeiro Positivo** (TP): número de exemplos da classe positiva (+),  $C_2$ , classificados corretamente.
- **Verdadeiro Negativo** (TN): número de exemplos da classe negativa (-),  $C_1$ , identificados corretamente.
- **Falso Positivo** (FP): exemplos classificados como positivos (+), mas que, na verdade, pertencem à classe negativa (-).
- **Falso Negativo** (FN): exemplos atribuídos à classe negativa (-), mas que, na verdade, pertencem à classe positiva (+).

➤ **Observe que:**

- $N_+$  define o número de padrões pertencentes à classe positiva = TP + FN.
- $N_-$  define o número de padrões pertencentes à classe negativa = FP + TN.
- $N$  define o número total de padrões = TP + FN + FP + TN.



## Métricas de avaliação da classificação

### Matriz de Confusão

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

- **Taxa de falso negativo:** proporção de exemplos da classe positiva (+) classificados incorretamente. É definida por

$$\text{Taxa de falso negativo} = p_e^+(\hat{y}(x)) = \frac{FN}{TP+FN} = \frac{FN}{N_+}.$$

- **Taxa de falso positivo:** proporção de exemplos da classe negativa (-) classificados incorretamente. É definida por

$$\text{Taxa de falso positivo} = p_e^-(\hat{y}(x)) = \frac{FP}{TF+FP} = \frac{FP}{N_-}.$$

- **Taxa de erro:**

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

- **Acurácia:**

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

## Métricas de avaliação da classificação

### Precisão

- Corresponde à proporção de exemplos da classe positiva (+) corretamente classificados em relação a todos os exemplos atribuídos à classe positiva (+). É definida por

$$\text{precisão}(\hat{y}(x)) = \frac{TP}{TP+FP}.$$

### Sensibilidade (recall)

- Também conhecida como **taxa de verdadeiro positivo**, a sensibilidade corresponde à proporção de exemplos da classe positiva (+) corretamente classificadas.

$$\text{recall}(\hat{y}(x)) = \frac{TP}{TP+FN}.$$

### Especificidade

Também conhecida como taxa de verdadeiros negativos, a especificidade é dada pela proporção de exemplos da classe negativa (-) corretamente classificadas.

$$\text{especificidade}(\hat{y}(x)) = \frac{TN}{TN+FP} = 1 - p_e^-(\hat{y}(x)).$$

# Métricas de avaliação da classificação

## Observações Importantes

- É possível estender estas métricas obtidas com a **matriz de confusão** para o cenário multi-classe (i.e.,  $Q > 2$ ): basta tomar, 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, obtem-se os valores das métricas para cada classe.
- Veja o exemplo abaixo para  $Q = 3$ .

Classe Estimada	+	Verdadeiro Positivo (TP)	Falso Positivo (FP)	Falso Positivo (FP)
	-	Falso Negativo (FN)	Verdadeiro Negativo (TN)	Verdadeiro Negativo (TN)
	-	Falso Negativo (FN)	Verdadeiro Negativo (TN)	Verdadeiro Negativo (TN)
		+	-	-
		Classe Verdadeira		

Classe Estimada	-	Verdadeiro Negativo (TN)	Falso Negativo (FN)	Verdadeiro Negativo (TN)
	+	Falso Positivo (FP)	Verdadeiro Positivo (TP)	Falso Positivo (FP)
	-	Verdadeiro Negativo (TN)	Falso Negativo (FN)	Verdadeiro Negativo (TN)
		-	+	-
		Classe Verdadeira		

Classe Estimada	-	Verdadeiro Negativo (TN)	Verdadeiro Negativo (TN)	Falso Negativo (FN)
	-	Verdadeiro Negativo (TN)	Verdadeiro Negativo (TN)	Falso Negativo (FN)
	+	Falso Positivo (FP)	Falso Positivo (FP)	Verdadeiro Positivo (TP)
		-	-	+
		Classe Verdadeira		

## Métricas de avaliação da classificação

### Observações Importantes

- Para o caso multi-classe, a **acurácia global** é obtida a partir das informações presentes na diagonal principal da **matriz de confusão**.
- A **precisão** pode ser entendida como uma medida da qualidade de um **classificador**, enquanto a **sensibilidade** (ou **recall**) dá uma noção de sua completude/qualidade.
  - Um valor de **precisão** = 1 significa que, para uma determinada classe, cada exemplo classificado como sendo pertencente a esta classe realmente pertence a ela. Entretanto, isso 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 valor de **recall** = 1 indica que todos os exemplos da classe  $C_q$  foram classificados como sendo pertencentes a  $C_q$ . Porém, isso não traz informações a respeito de quantos exemplos associados a outras classes foram classificados como sendo pertencentes a  $C_q$  (ou seja, quantidade de **falsos positivos**).

## Métricas de avaliação da classificação

### Pontuação-F

- As medidas de **precisão** e **recall** costumam ser analisadas conjuntamente através de uma métrica que combina ambas medidas, chamada de **pontuação-F** (ou **F-score**), denotada por  $F_m$ , que combina as duas medidas através de uma **média harmônica ponderada** dada pela equação 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  $m$  é o **fator de ponderação**.

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

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

- Valores de  $F_1$  próximos de 1 indicam que o **classificador** obteve bons resultados tanto de **precisão** quanto de **recall**.

A pontuação-F1 é a **média harmônica** entre **precisão** e **recall**. Enquanto a **média aritmética** trata todos os valores igualmente, a **média harmônica** atribui muito mais peso aos valores pequenos. Como resultado, o classificador só obterá uma pontuação-F1 alta se as medidas **recall** e **precisão** forem altas.

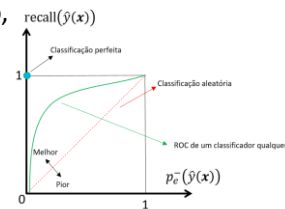
A pontuação-F1 favorece classificadores que têm **precisão** e **recall** semelhantes.

Aumentar a **precisão** reduz o **recall** e vice-versa. Isso é chamado de balanço (tradeoff) de precisão/recall.

## Métricas de avaliação da classificação

### Curva Característica Operacional do Receptor (ROC)

- É um gráfico, conforme mostrado na figura ao lado, em que a **taxa de verdadeiro positivo**, a qual equivale ao **recall**, é exibida em função da **taxa de falso positivo**.
- Quanto mais à esquerda e para cima estiver a **curva ROC** de um **classificador**, melhor será o seu desempenho.
- A linha diagonal, em vermelho, está associada a um **classificador aleatório**. Um bom **classificador** fica o mais longe possível dessa linha (em direção ao canto superior esquerdo).



Em estatística, uma característica operacional do receptor (ROC), ou curva ROC, é um gráfico que ilustra o desempenho de um classificador binário. A curva é criada plotando a taxa de verdadeiro positivo (recall) contra a taxa positiva falsa (especificidade) em várias configurações de limite.

A linha pontilhada representa a curva ROC de um classificador puramente aleatório; um bom classificador fica o mais longe possível dessa linha (em direção ao canto superior esquerdo).

Referência:

[1] [https://joparga3.github.io/standford\\_logistic\\_regression/#what-is-logistic-regression](https://joparga3.github.io/standford_logistic_regression/#what-is-logistic-regression)

## Métricas de avaliação da classificação

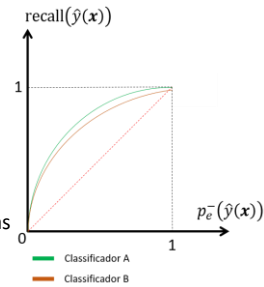
### Curva Característica Operacional do Receptor (ROC)

- A forma usual de se comparar **classificadores** consiste em criar uma **curva ROC** para cada um.
- Em geral, **classificadores** produzem uma saída real para cada exemplo de entrada. Normalmente, estas saídas são, então, discretizadas para que se tenha a decisão final: por exemplo, se  $h_a(x(i))$  ultrapassa um determinado **limiar**, ela é mapeada no valor 1 (classe positiva,  $C_2$ ); caso contrário, ela é mapeada no valor 0 (classe negativa,  $C_1$ ).
- Sendo assim, ao plotar a **taxa de verdadeiro positivo** (ou **recall**) versus a **taxa de falso positivo** para diferentes valores de **limiar**, obtemos a **curva ROC** associada a um **classificador**.

## Métricas de avaliação da classificação

### Curva Característica Operacional do Receptor (ROC)

- Por exemplo, considere as **curvas ROC** exibidas na figura ao lado. Para decidir qual o melhor **classificador**, podemos tomar como base a **área sob a curva ROC**.
- Neste exemplo, o classificador A seria o de melhor desempenho, pois tem maior **área sob a curva ROC**.
- **Vantagens da curva ROC**
  - Possibilita a análise de diferentes métricas de desempenho independente do **limiar** escolhido.
  - Auxilia o estudo de diferentes **limiares** para lidar com problemas de desbalanceamento nos dados (i.e., nos quais as classes possuem tamanhos discrepantes).
- **Desvantagens**
  - Apropriada para problemas de **classificação binária**.
  - No caso multi-classe, devemos utilizar a estratégia **um-contra-todos** e plotar várias **curvas ROC**.



A área sob a curva ROC é uma medida da qualidade do classificador.



## Exemplo: Métricas com SciKit-Learn

```
import numpy as np
from sklearn.datasets import make_blobs
from sklearn.linear_model 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(solver='lbfgs', 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.

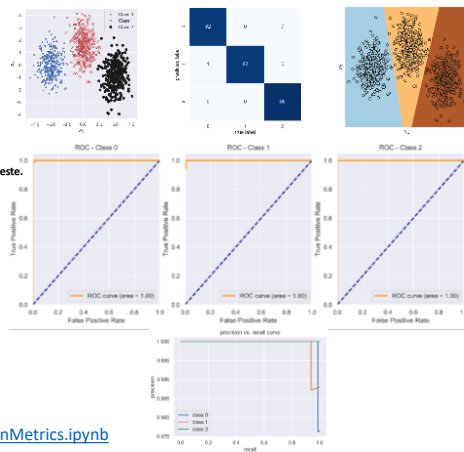
→ Cria objeto da classe LogisticRegression para múltiplas classes.

→ Cria matriz de confusão.

→ Curva ROC.

→ Calcula várias métricas.

Exemplo: [ClassificationMetrics.ipynb](#)



**Exemplo:** [ClassificationMetrics.ipynb](#)

**Link:** <https://colab.research.google.com/github/zz4fap/tp555-machine-learning/blob/master/exemplos/classification/linear/logistic/ClassificationMetrics.ipynb>

## Tarefas e Avisos

- Exemplos já estão disponíveis no site.
- Lista #5 já está disponível no site.
- Lista #4 pode ser entregue até dia 05/05.
- Prova: 23/06 (entrega dia 30/06)
- Apresentações do Projeto Final: 23/06 e 30/06 (ainda não defini se online ou gravadas).
- **IMPORTANTE:** todos os exercícios que envolvam programação devem estar implementados em um notebook do Jupyter. Eu não terei tempo de rodar todos os repos e através do Notebook eu consigo ver os gráficos e resultados.
- Atendimento às quartas-feiras das 8 as 10 da manhã (Skype: zz4fap)
- <https://www.inatel.br/docentes/felipefigueiredo/>


Obrigado!

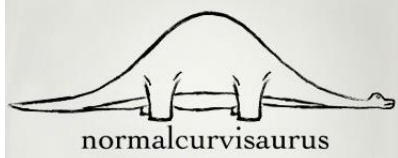
THIS IS YOUR MACHINE LEARNING SYSTEM?

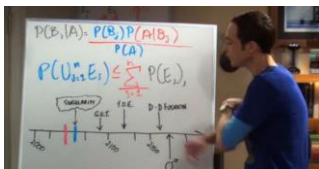
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.

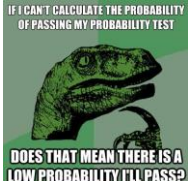
WHAT IF THE ANSWERS ARE WRONG?

JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.









		Actual Values	
		1	0
Predicted Values	1	TRUE POSITIVE You're pregnant	FALSE POSITIVE You're pregnant TYPE 1 ERROR
	0	FALSE NEGATIVE You're not pregnant TYPE 2 ERROR	TRUE NEGATIVE You're not pregnant

