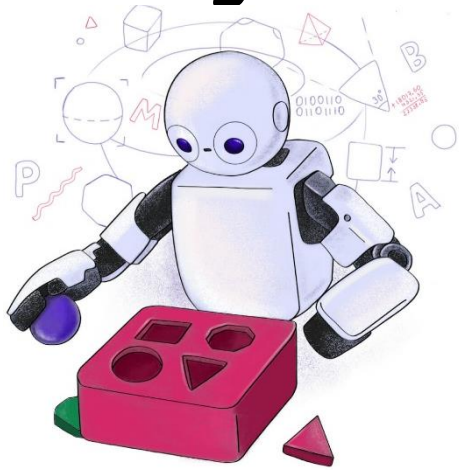
TP555 - Inteligência Artificial e Machine Learning:

Classificação Linear

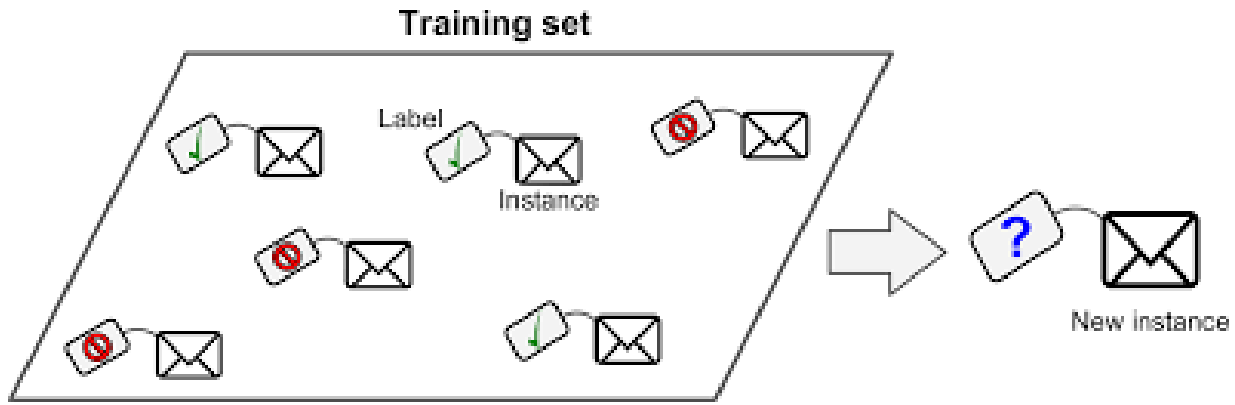


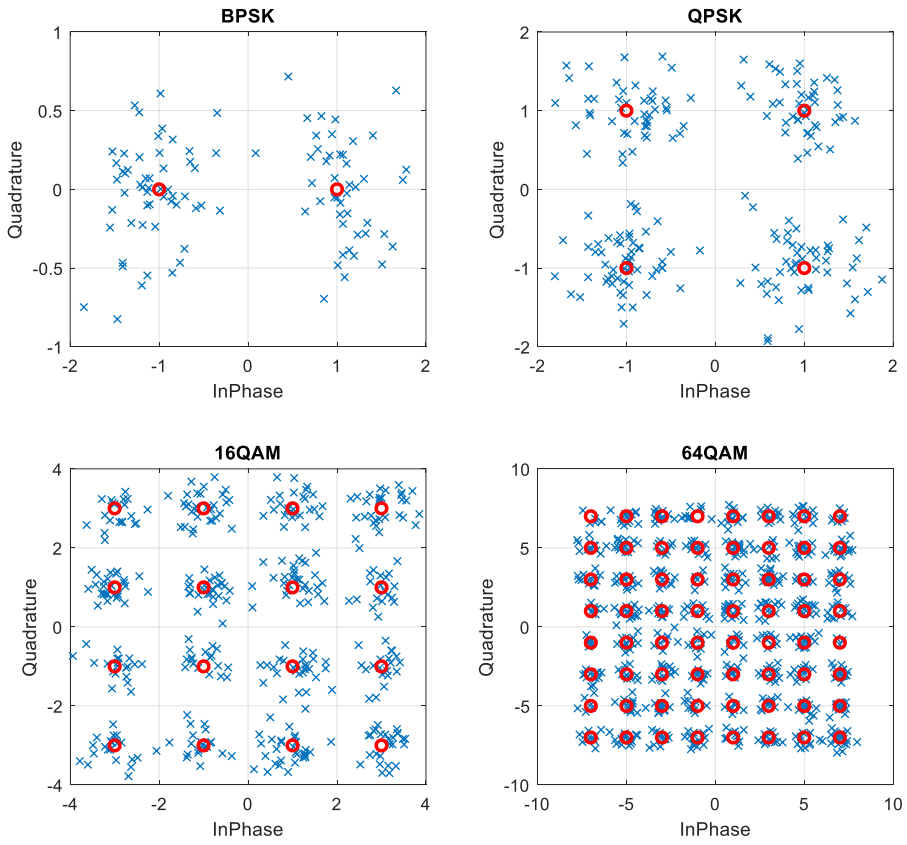
Felipe Augusto Pereira de Figueiredo

Tópicos abordados

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

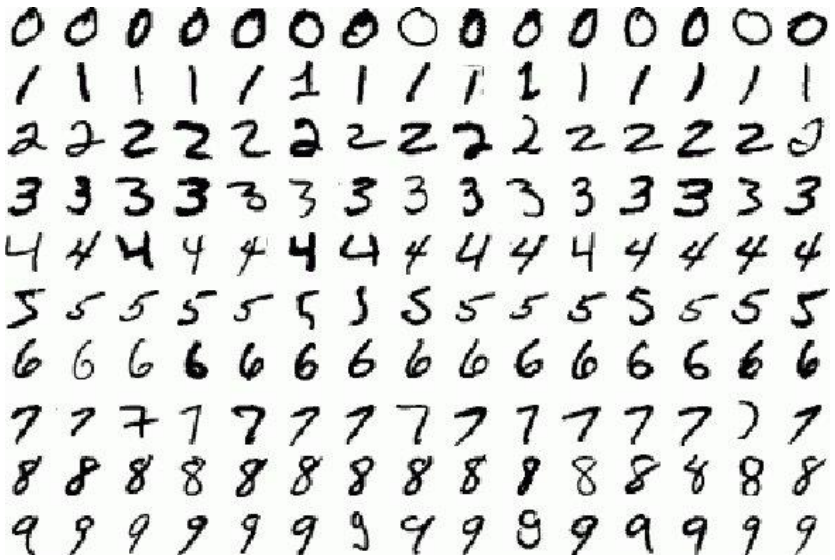
Motivação





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

Motivação



- 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 $\{\boldsymbol{x}(i); y(i)\}_{i=0}^{N-1}$ que é utilizado para treinar um ***classificador***, onde
  - $\boldsymbol{x}(i) = [x_1(i) \quad \cdots \quad x_K(i)]^T \in \mathbb{R}^{K\times 1}$ representa o *i*-ésimo vetor exemplo de entrada, o qual é caraterizado 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:

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

- 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 $\boldsymbol{x}(i) \in C_1$.

Representação da saída desejada

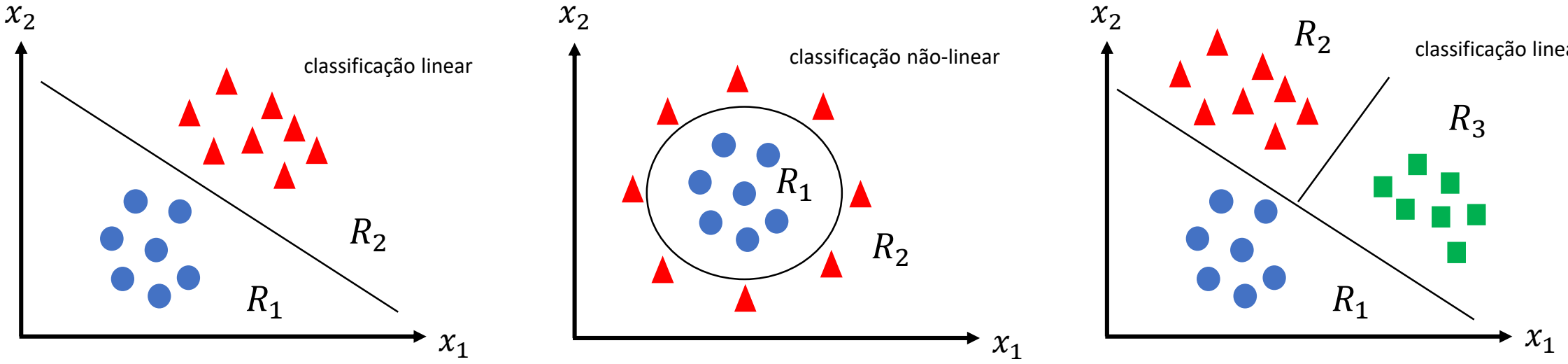
- **Classificação multi-classes**: existem mais de 2 classes possíveis ($Q > 2$).
- Uma estratégia bastante utilizada para representar estas classes é conhecida como ***one-hot encoding***.
- ***one-hot encoding***: utiliza uma representação binária para cada uma das variáveis categóricas.
- Neste caso, 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***?

$$\left.\begin{array}{ll} \textit{esporte:} & [1 \quad 0 \quad 0 \quad 0]^T \\ \textit{política:} & [0 \quad 1 \quad 0 \quad 0]^T \\ \textit{ciências:} & [0 \quad 0 \quad 1 \quad 0]^T \\ \textit{variedades:} & [0 \quad 0 \quad 0 \quad 1]^T \end{array}\right\}$$

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

Fronteiras de decisão de um classificador

- O espaço de entrada $\mathbb{R}^K$ é dividido em ***regiões de decisão*** $R_i, i = 1, \ldots, \mathrm{Q}$, as quais são delimitadas ou separadas pelas ***fonteiras 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

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

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

- ***Classificadores lineares*** sãos 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 $\boldsymbol{x} \in \mathbb{R}^{K\times 1}$. 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 $\boldsymbol{x}$ 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, $\boldsymbol{x}$.
  - 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 $P(C_q|\boldsymbol{x})$.
  - Como calculamos $P(C_q|\boldsymbol{x})$?
- **Teorema de Bayes**

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

onde o termo $P(\boldsymbol{x}|C_q)$ é denominado de ***verossimilhança*** (***likelihood***) e o termo $P(\boldsymbol{x})$ 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 $\boldsymbol{x}$ é dada pela classe C_q que maximiza $P(C_i|\boldsymbol{x})$, ou seja, em forma matemática:

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

Probabilidade a posteriori

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

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

já que o denominador $P(\boldsymbol{x})$ 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 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 $P(\boldsymbol{x}|C_i)$. Neste sentido, o ML escolhe a classe C_q mais plausível ou mais verossímil em relação ao padrão observado:

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

- **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 probabilistica: $P(\text{doença}|+)$, 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 $\boldsymbol{x}$ pode ser reescrita como

$$P\left(C_q|\boldsymbol{x} = [x_1 \quad \cdots \quad x_K]^T\right) = \frac{P\left(\boldsymbol{x}|C_q\right)P\left(C_q\right)}{P(\boldsymbol{x})} = \frac{P\left(x_1|C_q\right)\ldots P\left(x_K|C_q\right)P\left(C_q\right)}{P(x_1)\ldots P(x_K)}$$

- Com a independência dos atributos, os decisores MAP e ML são dados por

$$\text{MAP: } C_q = \arg\max_{C_i, i=1,\ldots,Q} P(x_1|C_i)\ldots P(x_K|C_i)P(C_i),$$

$$\text{ML: } C_q = \arg\max_{C_i, i=1,\ldots,Q} P(x_1|C_i)\ldots P(x_K|C_i).$$

- Aplicações típicas do classificador naive Bayes incluem filtragem de spam, classificação de documentos e previsão de sentimentos.

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.

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, Multinominal 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 naïve Bayes Gaussiano

- Quando lidamos com ***atributos*** x_1, ..., 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, ..., x_K, de acordo com a classe a que pertencem;
  - Em seguida, calcula-se a média, μ_{x_k,C_q}, e a variância, $\sigma^2_{x_k,C_q}$, 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 μ_{x_k,C_q} e $\sigma^2_{x_k,C_q}$.

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

- 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.

Exemplo: Probabilidade da prática de esportes

Nesse exemplo vamos usar o classficador 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

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

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

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

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

Agora calculamos as probabilidades:

- *P*(jogar=sim | temp.=25, hum.=62) = *P*(temp.=25 | jogar=sim) *P*(hum.=62 | jogar=sim)*P*(jogar=sim) = 5.83e-4
- *P*(jogar=não | temp.=25, hum.=62) = *P*(temp.=25 | jogar=não) *P*(hum.=62 | jogar=não)*P*(jogar=não) = 5.78e-5

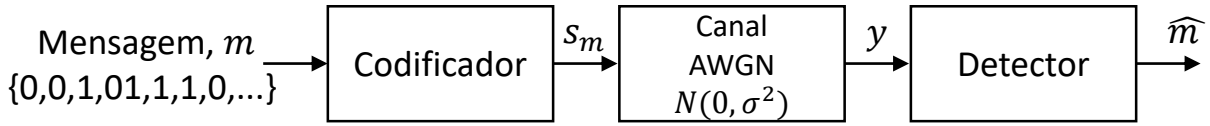
Portanto, a probabilidade é maior para o caso deles jogarem.

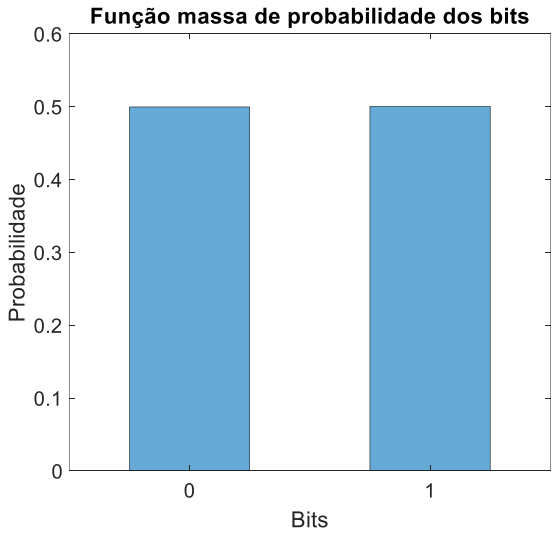
Exemplo: Detecção de símbolos BPSK em canais AWGN

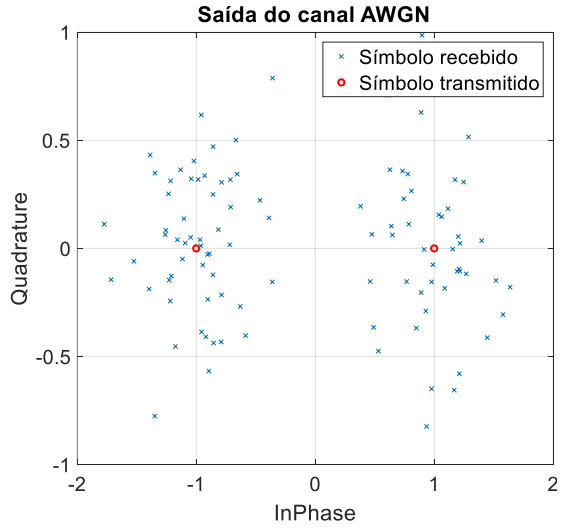
- Imagine um codificador que converte o m-ésimo bit de uma mensagem composta por 0s e 1s nos símbolos s_0 = -1 e s_1 = 1, para $m = 0$ e 1, respectivamente.
- Em seguida, os símbolos codificados passam por um canal AWGN cuja saída é dada por

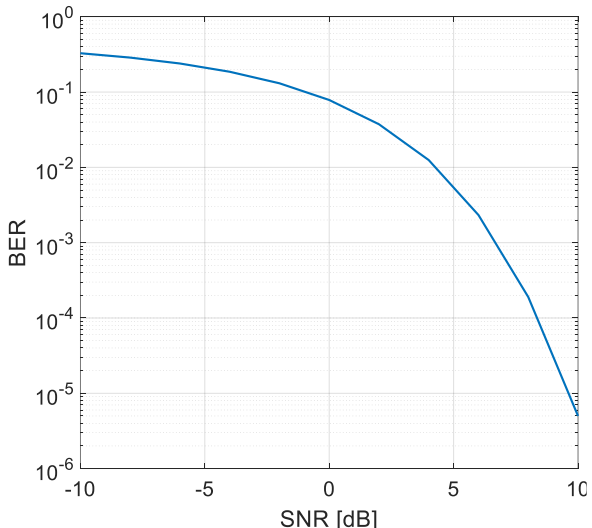
$$y = s_m + w, \text{ onde } w \sim N(0, \sigma^2).$$

- 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: $y = s_0 + w$

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

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

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

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

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

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()

# Create Es/N0 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))))
```

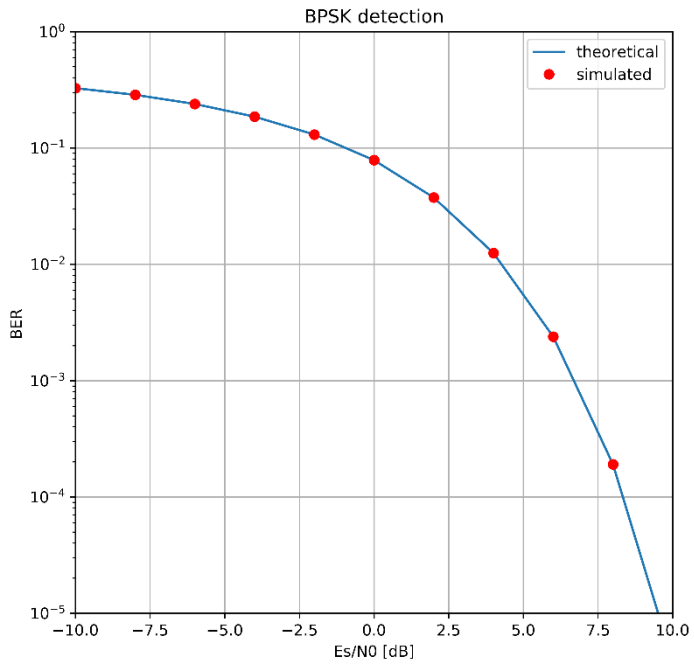
Importa classe GaussianNB do módulo naive_bayes do biblioteca SciKit-Learn.

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.

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 $\boldsymbol{x} = [x_1, x_2, \dots, x_K]^T$ 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 $\boldsymbol{x}$ é dada por

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

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

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)
```

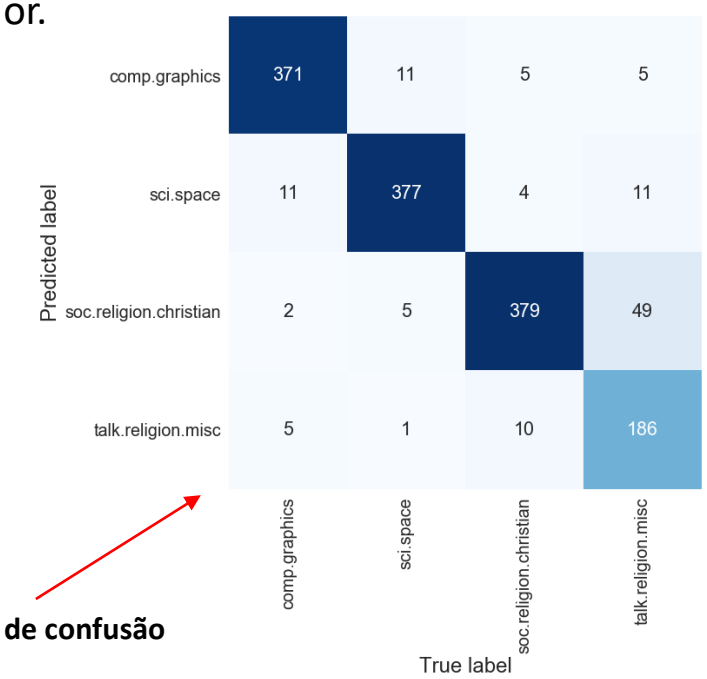
Base com 20 tópicos diferentes de discussão.

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.



Matriz de confusão

[Exemplo: ClassifyingTextMultinomialNB.ipynb]

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

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

  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.

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.

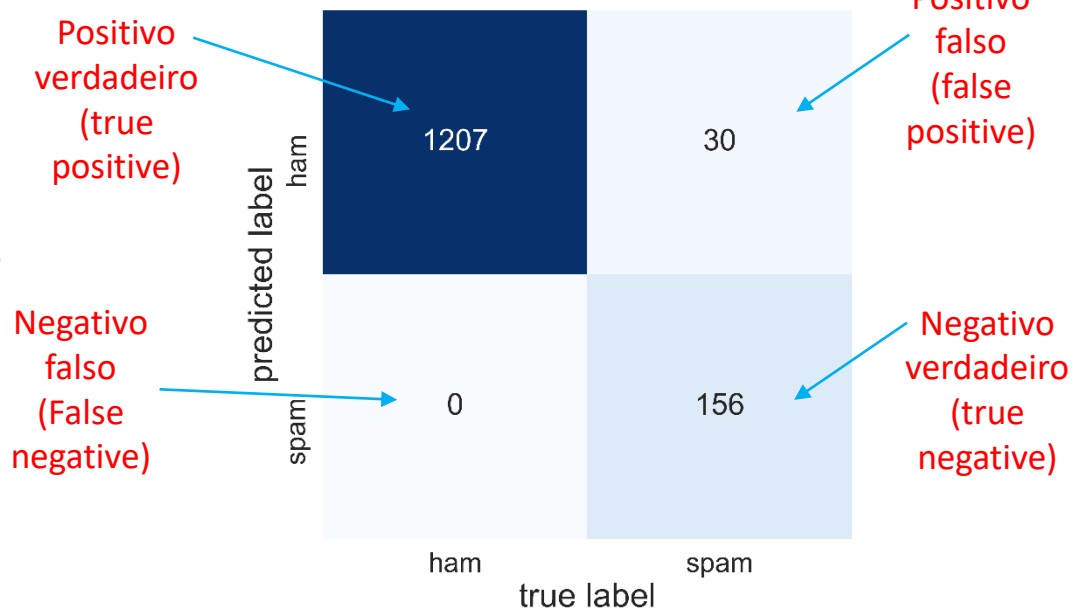
Cria matriz booleana indicando ou não a presença de uma palavra.

Treinamento do classificador.

Cria matriz boolena 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.



[Exemplo: SPAMClassificationBernoulliNB.ipynb]

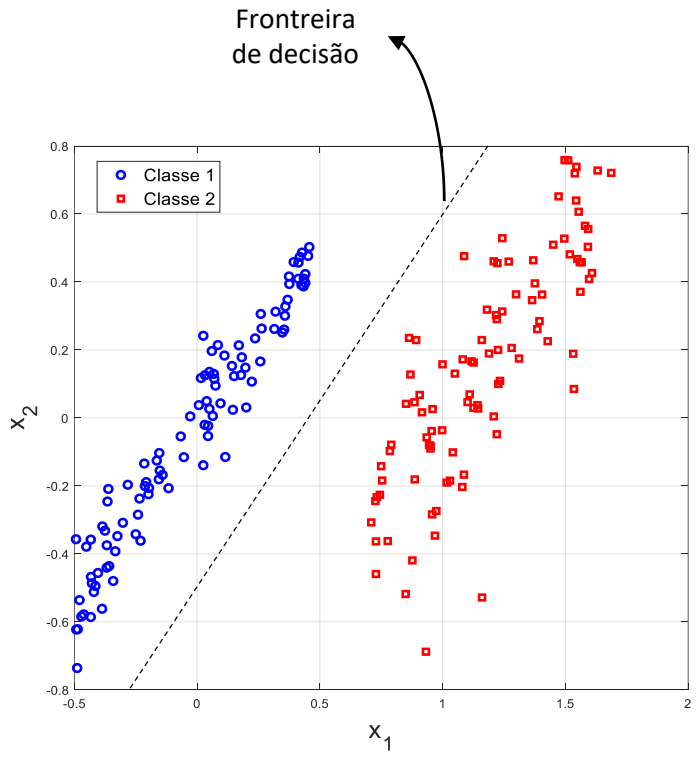
Classificadores lineares com limiar de decisão rígido

- 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 ou ***1*** caso ela seja C_2.
- 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.
- 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

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

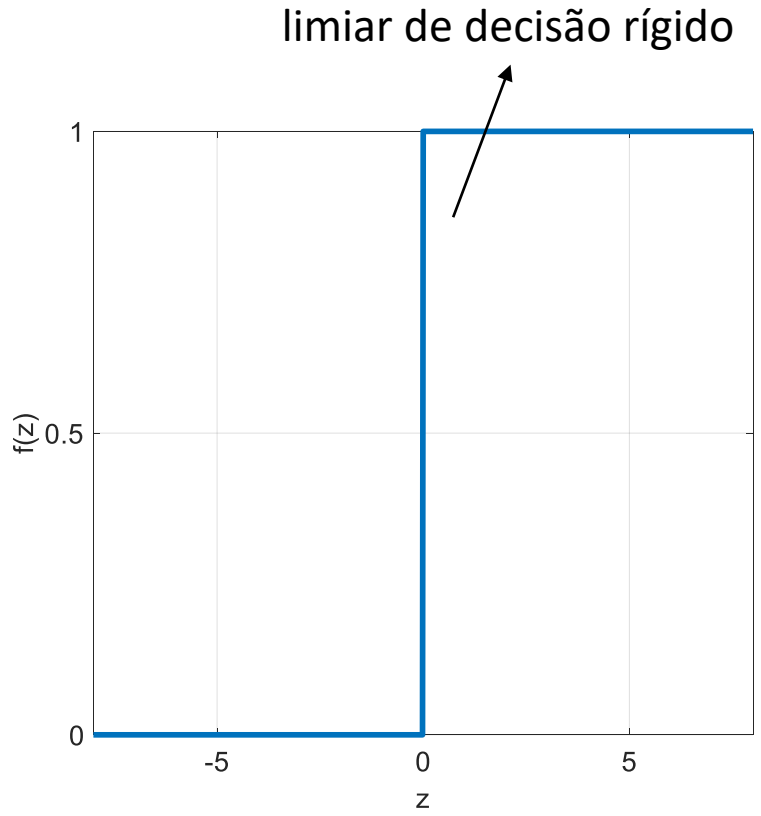
- Alternativamente, nós podemos expressar h como sendo o resultado de passar a função linear $\boldsymbol{a}^T\boldsymbol{x}$ através de uma ***função de limiar*** $f(.)$

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



Classificadores lineares com limiar de decisão rígido

- A **função de limiar** é mostrada na figura ao lado.
- Agora que a ***hipótese*** $h_{\boldsymbol{a}}(\boldsymbol{x})$ tem uma forma matemática bem definida, nós podemos pensar em como escolher os pesos $\boldsymbol{a}$ que minimizem o erro.
- No caso da regressão linear, nós fizemos isso de duas maneiras: (i) de forma fechada (eq. 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 escolhemso 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 $\boldsymbol{x}^T \boldsymbol{a} = 0$, e mesmo assim, o gradiente é indeterminado nesses pontos.
- **Portanto, o que podemos fazer?**



Classificação linear com regressão logística

- Além do fato de que a **hipótese** $h_{\boldsymbol{a}}(\boldsymbol{x})$ 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), definida como

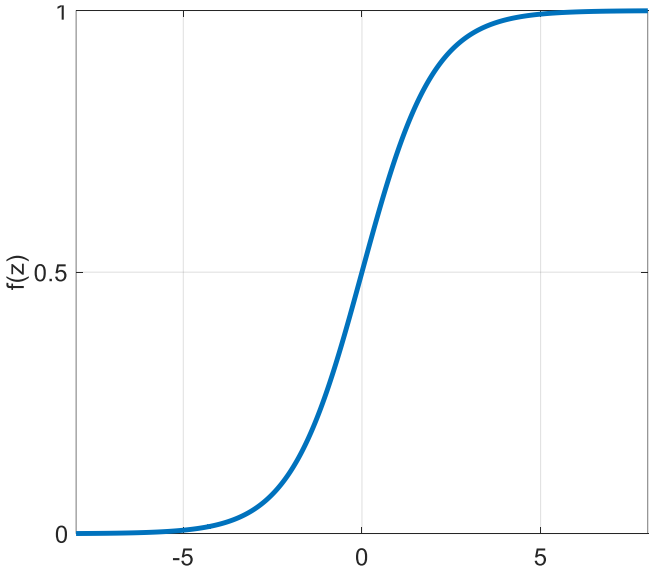
$$Logistic(z) = \frac{1}{1+e^{-z}},$$

apresenta tais propriedades matemáticas. A função é mostrada na figura ao lado.

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

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

- 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 1.
- A hipótese $h_{\boldsymbol{a}}(\boldsymbol{x})$ 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 distancie da fronteira.



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

Regressão logística

- A ***regressão logística*** (também chamada de regressão logit) é um método para ***classificação binária***. Ela classifica os pontos de um conjunto de dados em duas classes distintas, ou seja, é ótima para situações em que você precisa classificar entre duas classes.
- A ***regressão logística*** estima a probabilidade de um exemplo pertencer a uma classe específica (por exemplo, qual é a probabilidade de uma 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

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

- 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 1 se $\boldsymbol{x}^T \boldsymbol{a}$ for positivo e 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.

Propriedades da regressão logística

- Os valores de saída da hipótese ficam restritos entre $0 \leq h_{\boldsymbol{a}}(\boldsymbol{x}) \leq 1$.
- A saída de $h_{\boldsymbol{a}}(\boldsymbol{x})$ representa a probabilidade de $\boldsymbol{x}$ pertencer à classe positiva, C_2, para qual a saída desejada é $y = 1$. Ou seja, $h_{\boldsymbol{a}}(\boldsymbol{x}) = P(C_2 \,|\boldsymbol{x}; \boldsymbol{a})$.
- Assim, consequentemente, $(1 - h_{\boldsymbol{a}}(\boldsymbol{x}))$ = $P(C_1 \,|\boldsymbol{x}; \boldsymbol{a})$.
- A ***fronteira de decisão*** é determinada quando quando há uma indecisão entre as classes, ou seja, quando $P(C_1 \,|\boldsymbol{x}; \boldsymbol{a}) = P(C_2 \,|\boldsymbol{x}; \boldsymbol{a})$.
- Isto ocorre quando $P(C_2 \,|\boldsymbol{x}; \boldsymbol{a}) = h_{\boldsymbol{a}}(\boldsymbol{x}) = 0.5$.
- Observando a figura da função logística, nós percebemos que *Logistic*$(z) = 0.5$ quando $z = 0$.
- Desta forma, a ***fronteira de*** *decisão* é caracterizada por

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

  e corresponde a um hiperplano.
- A saída do classificador, $\hat{y}$, é dada por

$$\hat{y} = \begin{cases} 0 \text{ (classe } C_1), \text{se } h_{\boldsymbol{a}}(\boldsymbol{x}) < 0.5 \\ 1 \text{ (classe } C_2), \textit{se } h_{\boldsymbol{a}}(\boldsymbol{x}) \geq 0.5 \end{cases}$$

Função de erro

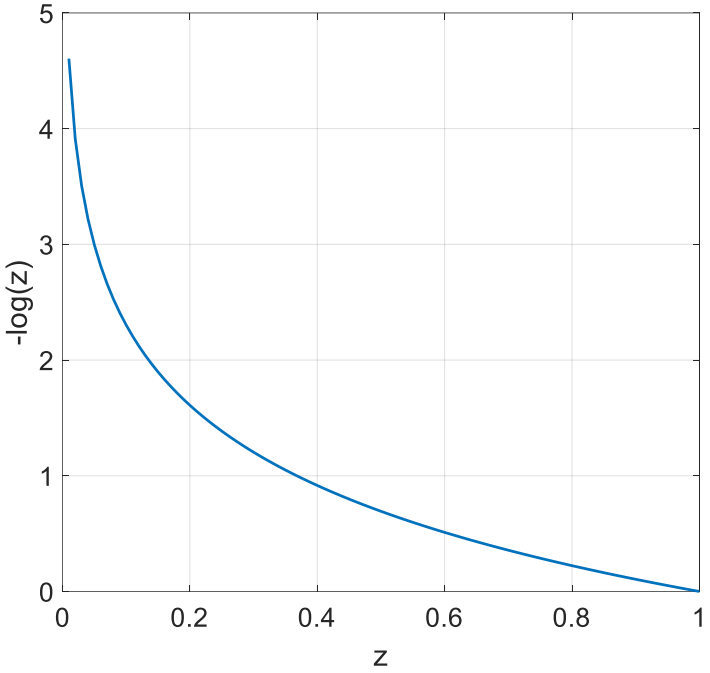
- Adotar o ***erro quadrático médio*** como função de erro não é uma escolha muito acertada para a adaptação dos pesos no caso ***da regressão logística***.
- A função de erro utilizando o ***erro quadrático médio*** é dada por

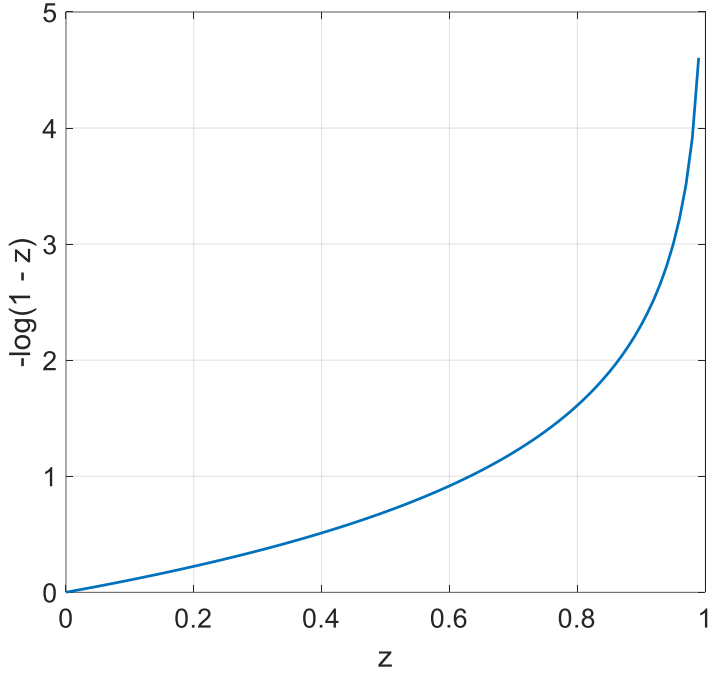
$$J_e(\boldsymbol{a}) = \frac{1}{N}\sum_{i=1}^{N}\big(y(i) - h_{\boldsymbol{a}}(\boldsymbol{x})\big)^2 = \frac{1}{N}\sum_{i=1}^{N}\big(y(i) - \mathit{Logistic}(\boldsymbol{x}^T\boldsymbol{a})\big)^2.$$

- Como *Logistic(.)* é uma função não-linear, $J_e(\boldsymbol{a})$ 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

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

Função de erro





As figuras ao lado mostram as duas situações possíveis para a funçãode 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 modelo 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 modelo 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 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 reduzir a definição da ***função de erro*** a uma expressão única:

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

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

$$J_e(\boldsymbol{a}) = -\frac{1}{N} \sum_{i=0}^{N-1} y(i) \log\left(h_{\boldsymbol{a}}(\boldsymbol{x}(i))\right) + (1-y(i)) \log(1-h_{\boldsymbol{a}}(\boldsymbol{x}(i))).$$

- A má notícia é que não existe uma equação de forma fechada conhecida para calcular o valor dos pesos $\boldsymbol{a}$ que minimiza 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).

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 $J_e(\boldsymbol{a})$, vamos reescrever a ***função de erro*** utilizando as seguintes equivalências

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

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

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

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

Processo de treinamento

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

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

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

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

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

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

Processo de treinamento

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

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

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

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

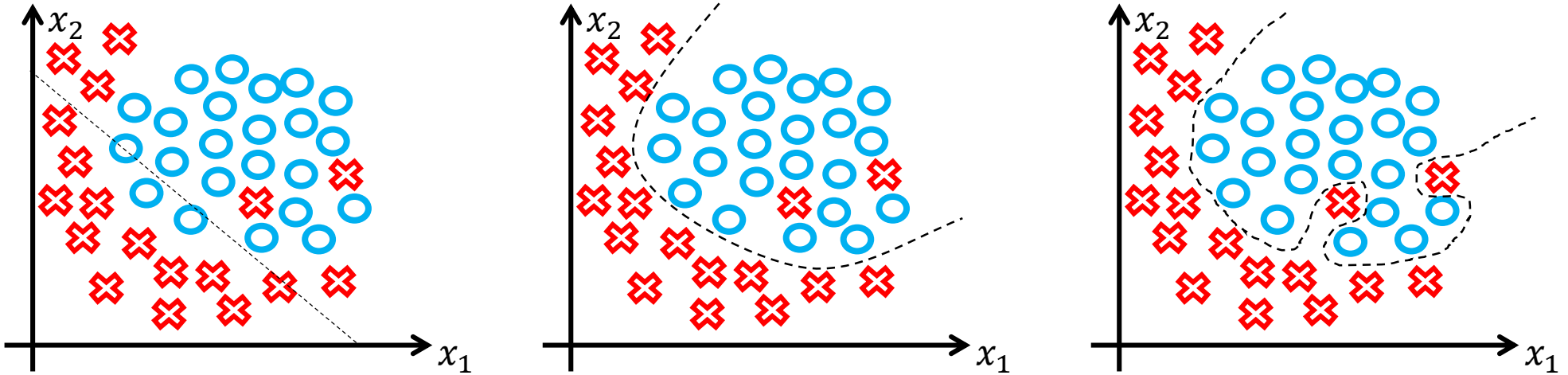
- Portanto, combinando os 2 resultados acima, temos que o ***vetor gradiente*** da ***função de erro*** é dado por

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

- 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.

Observações

- A expressão do ***vetor gradiente*** da ***função de erro***, para a ***regressão logística*** é idêntica 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 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 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(solver='lbfgs')
# 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.

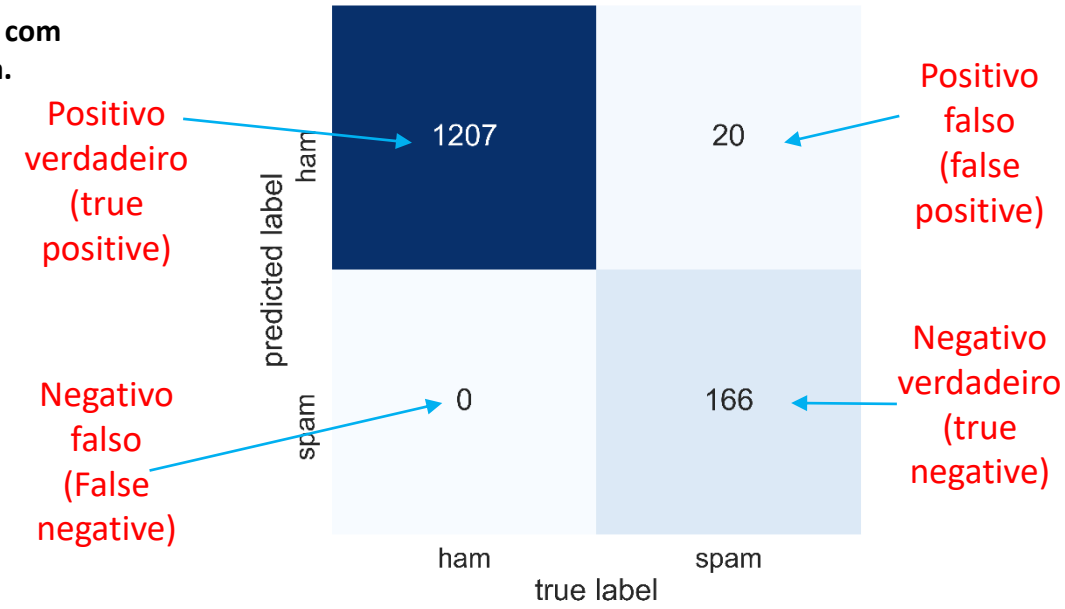
Divide a base de dados em 75% treinamento e 25% validação.

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 matriz registrando o número de vezes (frequência) que cada palavra aparece na mensagem.
- A ***matriz de confusão*** mostra a performance do classificador.



[Exemplo: SPAMClassificationLogisticRegressionSciKit.ipynb]

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) $h_{\boldsymbol{a}}(\boldsymbol{x}(i))^{(q)}$ para cada classe q para predizer a probabilidade de $y = q$.
- Ou seja, 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, $\boldsymbol{x}$, realiza-se as predições e escolhe-se a classe que maximize

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

- Vantagem 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 estratégia, 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 classficar entre C_1/C_2, C_1/C_3, C_1/C_4, C_2/C_3, C_3/C_4, e C_3/C_4).
- No final, cada exemplo é classificado conforme o voto majoritário entre os classificadores.
- A principal vantagem do ***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***.
- Úma estratégia 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***, definida por

$$P\left(C_q \mid \boldsymbol{x}(i)\right) = h_{\boldsymbol{a}}^{q}\left(\boldsymbol{x}(i)\right) = \frac{e^{\boldsymbol{x}(i)^T \boldsymbol{a}_q}}{\sum_{i=1}^{Q} e^{\boldsymbol{x}(i)^T \boldsymbol{a_i}}},$$

onde $\boldsymbol{a_q} = \begin{bmatrix} a_0^q & a_1^q & \cdots & a_K^q \end{bmatrix}^{\boldsymbol{T}}$ é o vetor de pesos associado à q-ésima saída e $h_{\boldsymbol{a}}^{q}\left(\boldsymbol{x}(i)\right)$ é a função hipótese associada à q-ésima classe.

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

$$C_q = \arg.\max_q h_{\boldsymbol{a}}^{q}\left(\boldsymbol{x}(i)\right) = \arg.\max_q \boldsymbol{x}(i)^T \boldsymbol{a}_q.$$

- A ***função de erro*** é dada por

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

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 $\boldsymbol{A} \in \mathbb{R}^{K+1\times Q}$ é a matriz com os pesos para todas as Q classes.

Propriedades da regressão softmax

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

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

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(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.

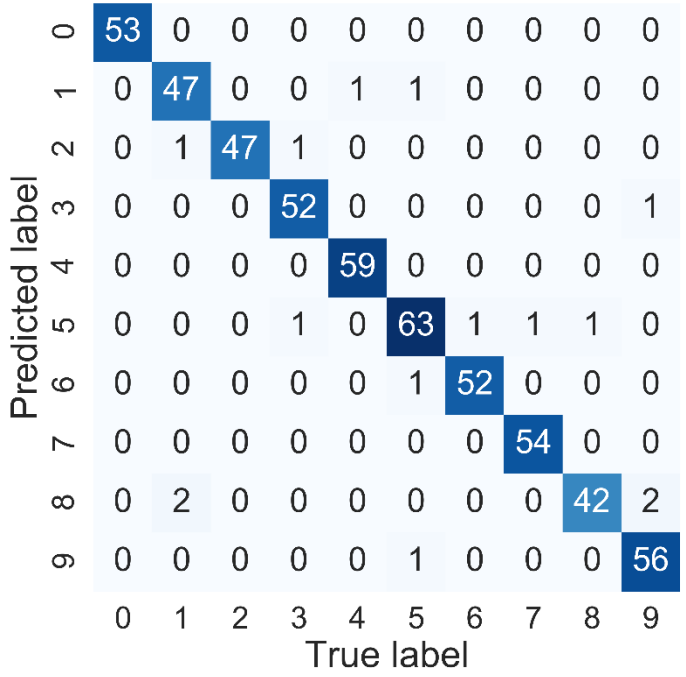
Carrega base de dados da biblioteca SciKit-Learn.

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

Instancia objeto da classe LogisticRegression Multinomial.

Treinamento e validação do classificador.

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



Exemplo: 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

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

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 $p_e(\hat{y}(\boldsymbol{x})) \in [0,1]$.

- O complemento da ***taxa de erro*** é conhecido como ***acurácia***, e é definida por

$$\text{acc}\big(\hat{y}(\boldsymbol{x})\big) = 1 - p_e(\hat{y}(\boldsymbol{x})).$$

Métricas de avaliação da classificação

Matrix de Confusão

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

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

onde o elemento C_{ij} indica quantos padrões da classe i foram designados à classe j. Em sua diagonal, portanto, temos o número de classificações corretas.

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

Métricas de avaliação da classificação

Matrix de Confusão

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

- ***Verdadeiro Positivo*** (TP): número de exemplos da classe positiva classificados corretamente.
- ***Verdadeiro Negativo*** (TN): número de exemplos da classe negativa 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

Matrix 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

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

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

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

- ***Taxa de erro***:

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

- ***Acurácia***:

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

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

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

Sensibilidade (recall)

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

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

Especificidade

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

$$\text{especificidade}\big(\hat{y}(\boldsymbol{x})\big) = \frac{\text{T}N}{\text{TN+F}P} = 1 - p_e^-\big(\hat{y}(\boldsymbol{x})\big).$$

Métricas de avaliação da classificação

Observações Importantes

- É possível estender estas métricas para o cenário multi-classe: basta tomar, uma vez, cada classe $C_q, q = 1, \ldots, Q$ 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 quantas amostras desta classe foram classificadas de forma incorreta (***falso negativo***).
  - 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 (***falso positivo***).

Métricas de avaliação da classificação

Pontuação-F

- As medidas de ***precisão*** e ***recall*** costumam ser analisadas juntas 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***:

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

onde m é o ***fator de ponderação***.

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

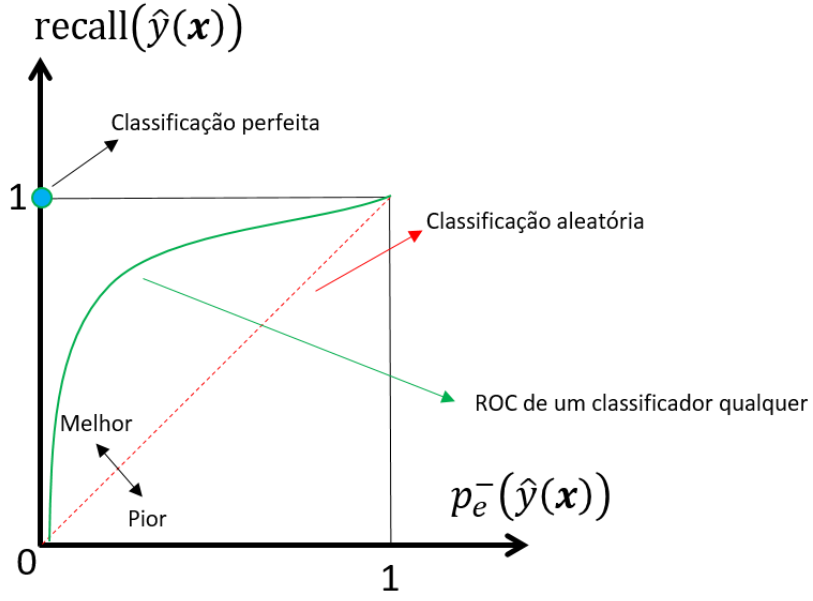
$$F_1 = 2\,\frac{\text{recall}(\hat{y}(\boldsymbol{x}))\times \text{precisão}\left(\hat{y}(\boldsymbol{x})\right)}{\text{recall}\left(\hat{y}(\boldsymbol{x})\right)+\text{precisão}\left(\hat{y}(\boldsymbol{x})\right)} = \frac{\text{TP}}{\text{TP}+\frac{\text{FN}+\text{FP}}{2}}.$$

- Valores de F_1 próximos de 1 indicam que o classificador obteve bons resultados tanto na ***precisão*** quanto no ***recall***.

Métricas de avaliação da classificação

Característica Operacional do Receptor (ROC)

- É um gráfico 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 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).



Métricas de avaliação da classificação

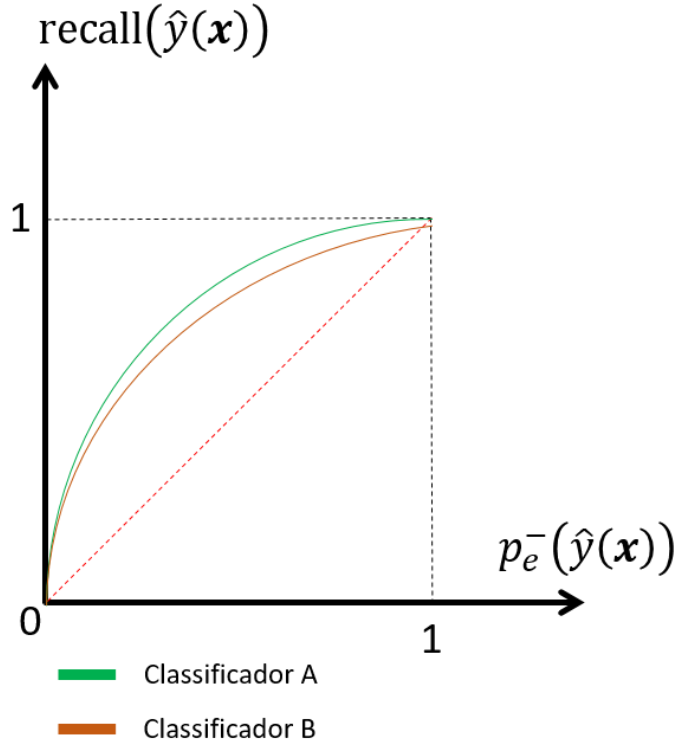
Característica Operacional do Receptor (ROC)

- 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 $h_{\boldsymbol{a}}\big(\boldsymbol{x}(i)\big)$ ultrapassa um determinado ***limiar***, ela é mapeada no valor 1 (classe positiva); caso contrário, ela é mapeada no valor 0 (classe negativa).
- 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

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.
- **Vantagens**
  - 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***.



Exemplo: Métricas 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(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])
# Claculate 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.

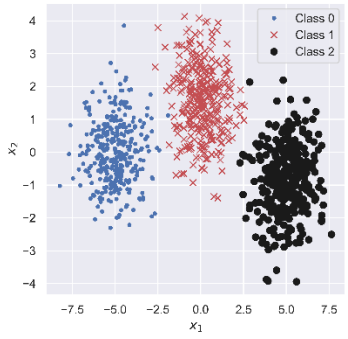
Cria objeto da classe LogisticRegression para múltiplas classes.

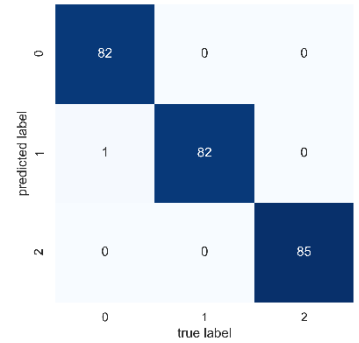
Treinamento e predição do classificador.

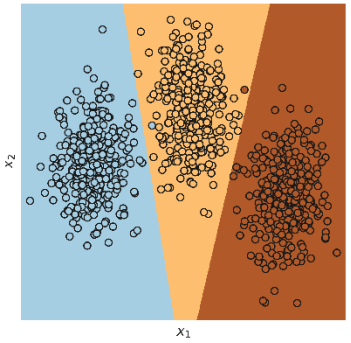
Cria matriz de confusão.

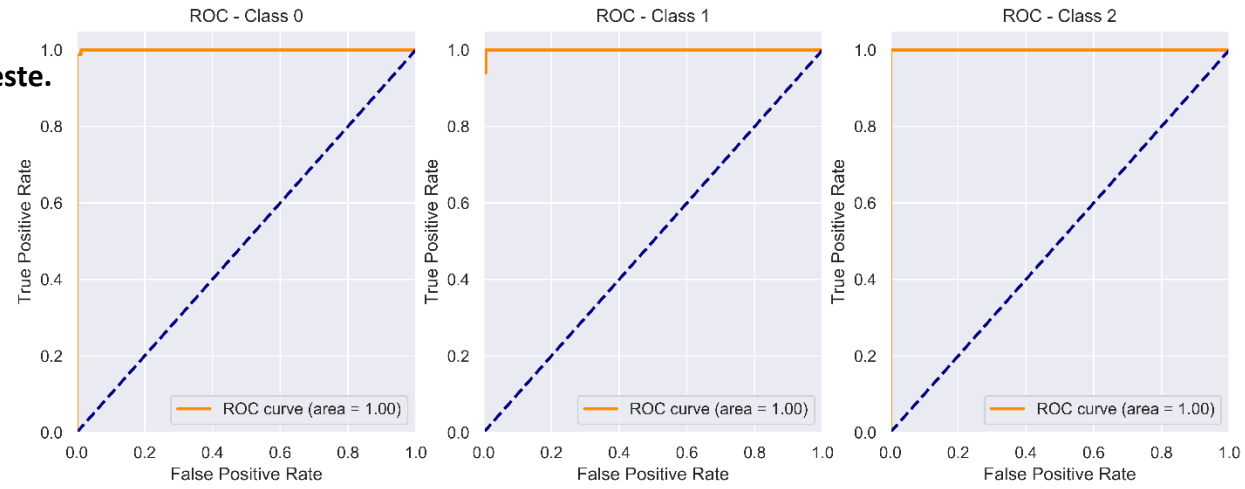
Curva ROC.

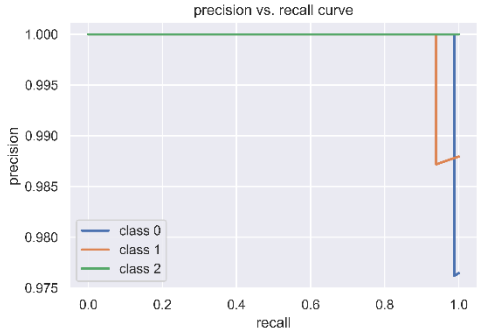
Calcula várias métricas.







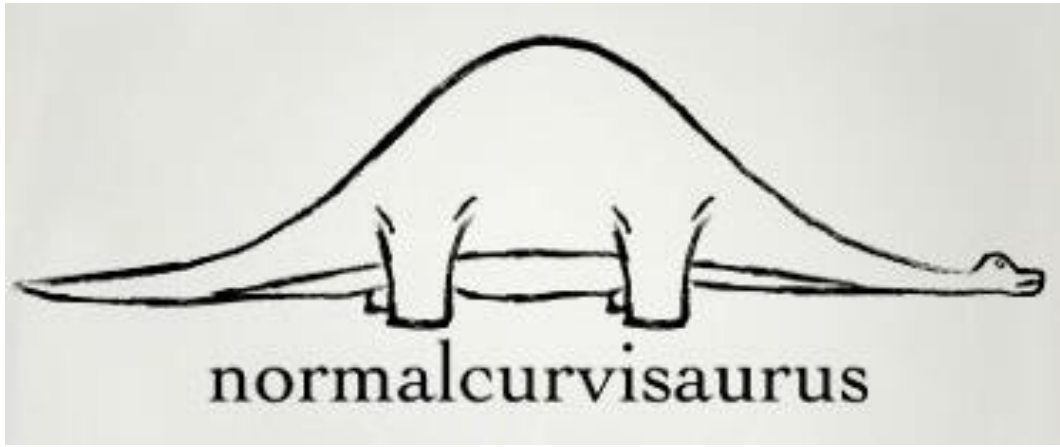


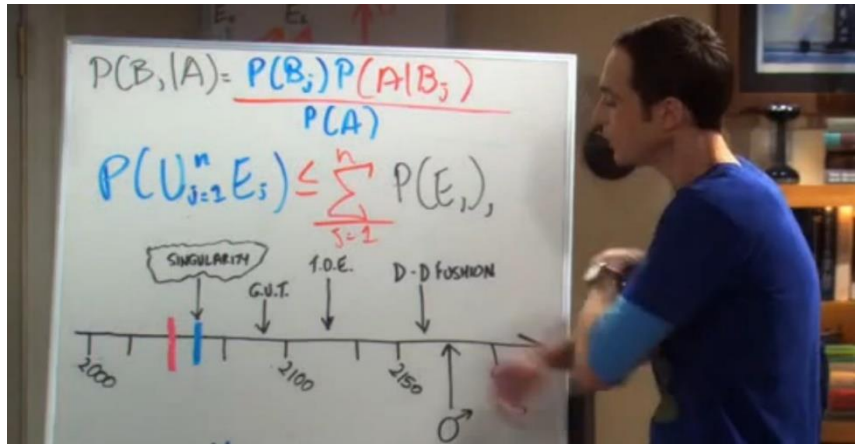


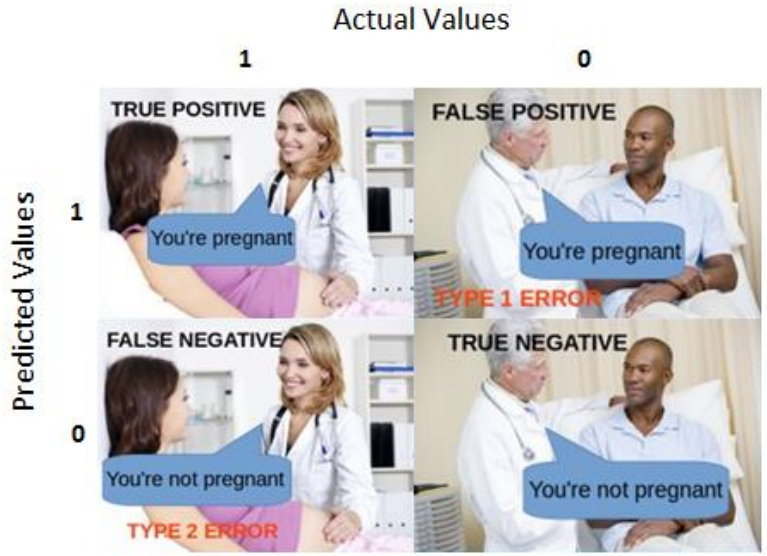
[Exemplo: ClassificationMetrics.ipynb]

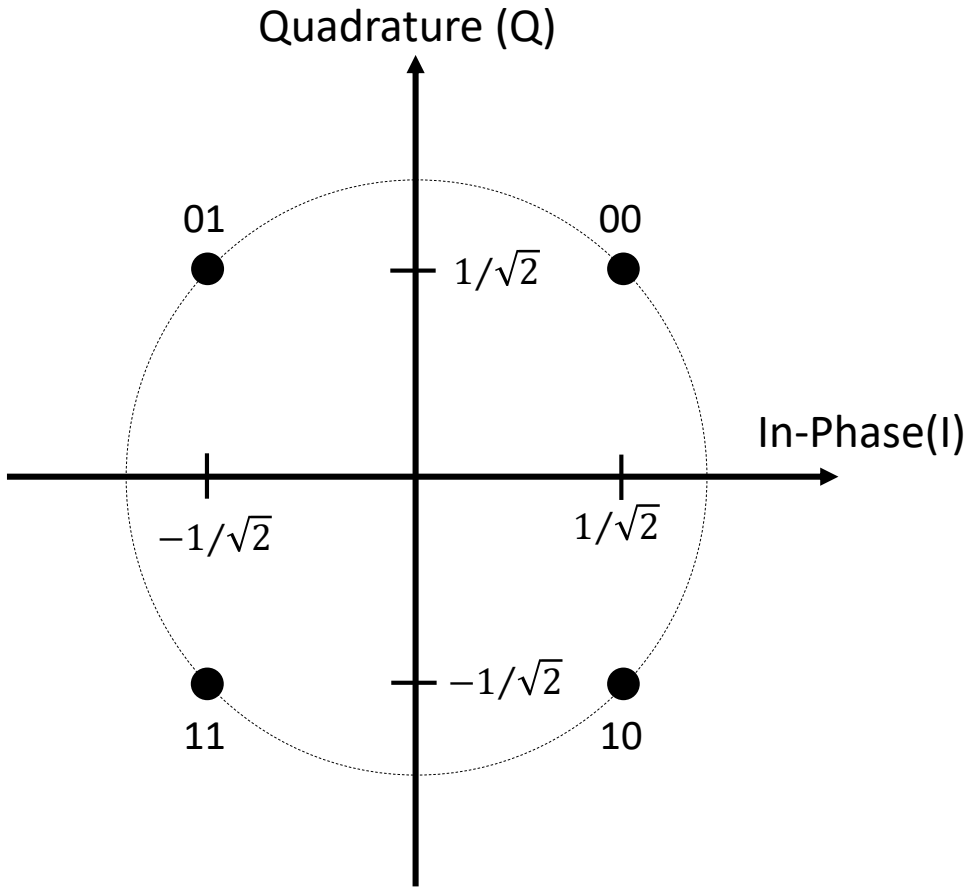
Obrigado!

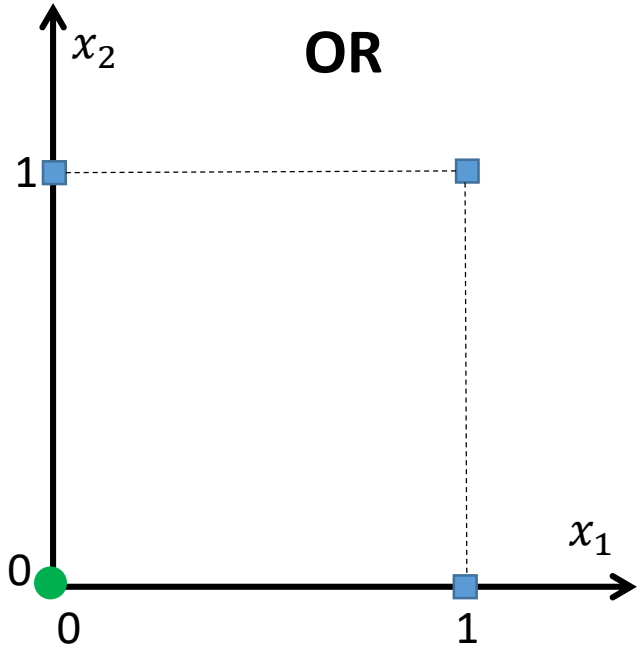

THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS


normalcurvisaurus


SINGULARITY
G.U.T.
T.O.E.
D-D FOSHION
2000
2100
2150

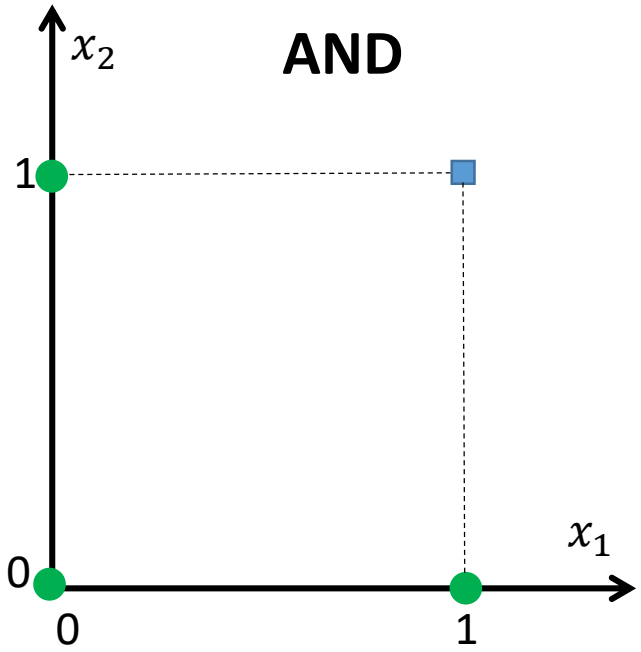

Actual Values
1
0
Predicted Values
1
0
TRUE POSITIVE
You're pregnant
FALSE POSITIVE
You're pregnant
TYPE 1 ERROR
FALSE NEGATIVE
You're not pregnant
TYPE 2 ERROR
TRUE NEGATIVE
You're not pregnant


Quadrature (Q)
01
00
$1/\sqrt{2}$
In-Phase(I)
$-1/\sqrt{2}$
$1/\sqrt{2}$
$-1/\sqrt{2}$
11
10



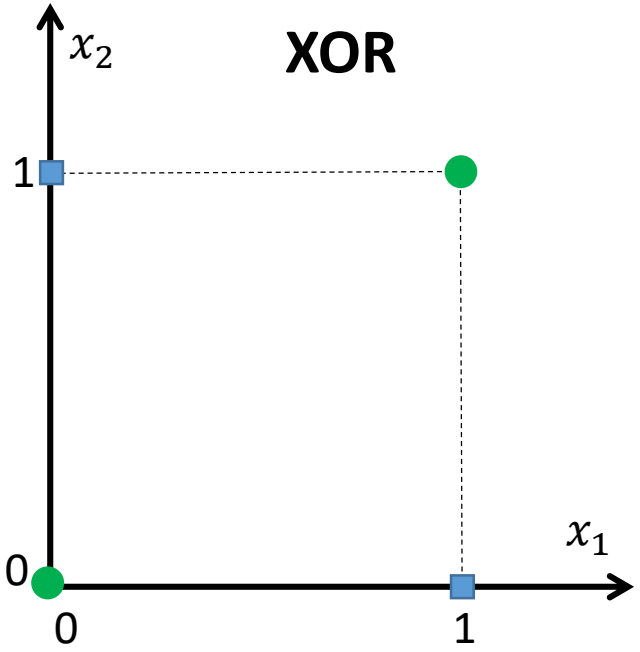
Classe 0 (nível lógico 0)

Classe 1 (nível lógico 1)



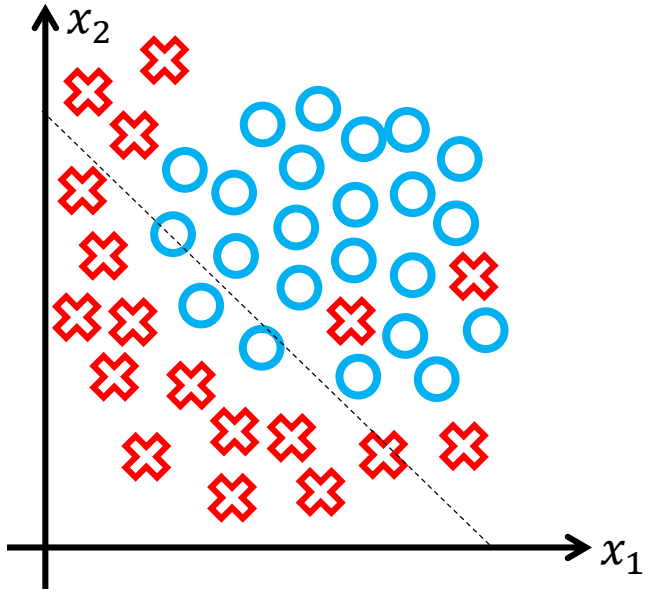
Classe 0 (nível lógico 0)

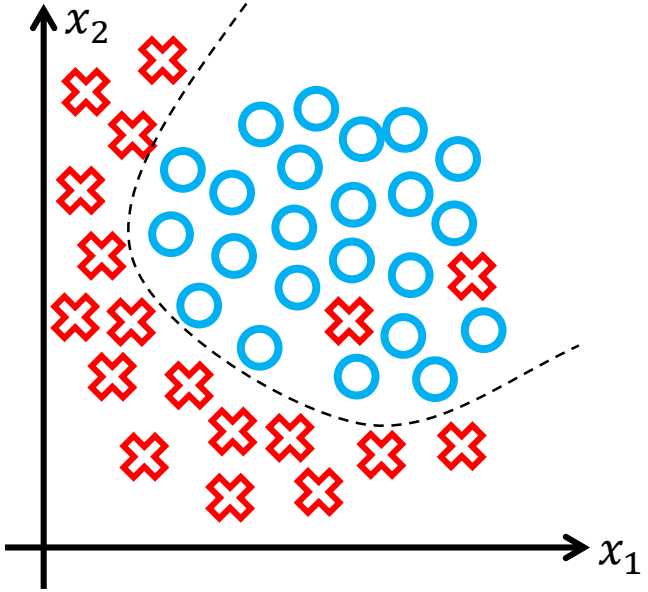
Classe 1 (nível lógico 1)

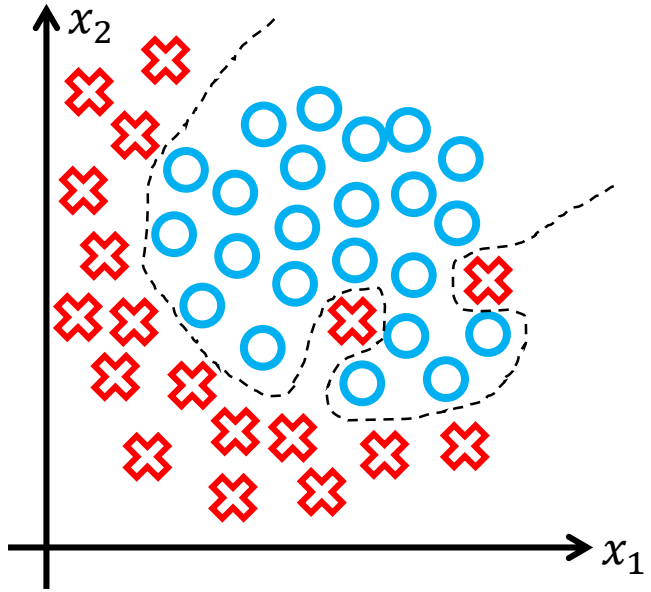


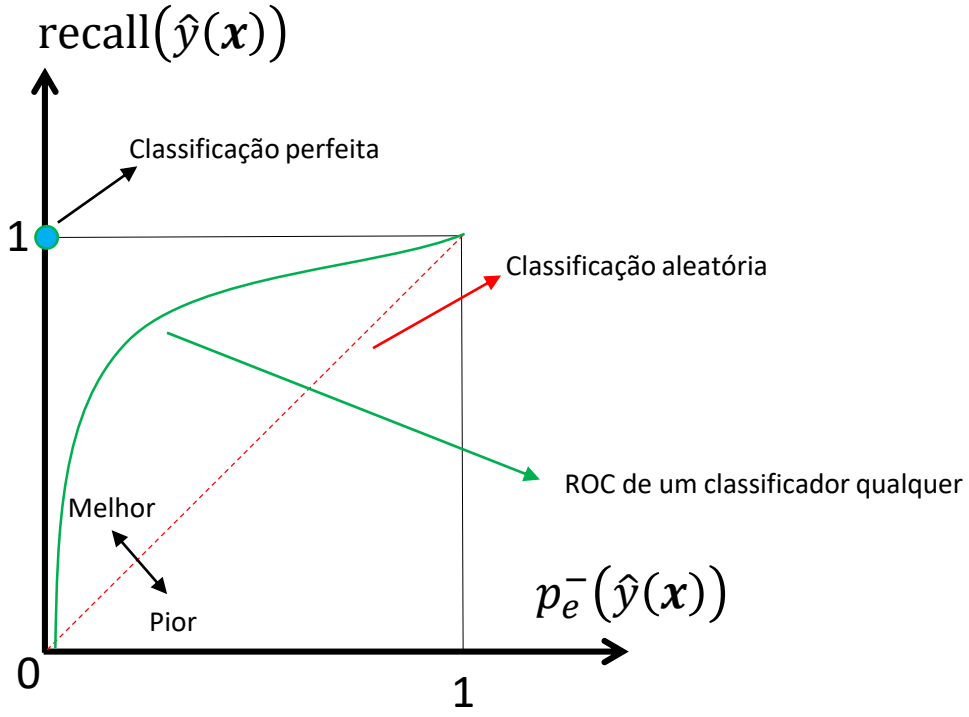
Classe 0 (nível lógico 0)

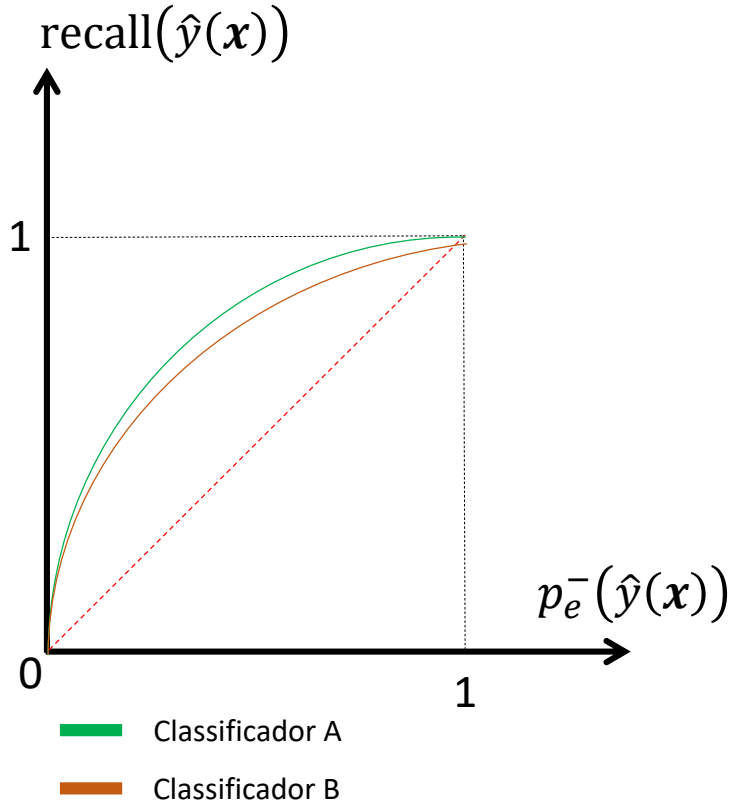
Classe 1 (nível lógico 1)


x_2
x_1


x_2
x_1


x_2
x_1


recall$(\hat{y}(\boldsymbol{x}))$
Classificação perfeita
1
Classificação aleatória
ROC de um classificador qualquer
Melhor
Pior
$p_e^-(\hat{y}(\boldsymbol{x}))$
0
1


recall$\left(\hat{y}(\boldsymbol{x})\right)$
1
0
1
$p_e^-\left(\hat{y}(\boldsymbol{x})\right)$
Classificador A
Classificador B