

# Aprendizado de Máquina

## Aula 3.4 - Algoritmos de Classificação


Adriano Rivolli

[rivolli@utfpr.edu.br](mailto:rivolli@utfpr.edu.br)

**Especialização em Inteligência Artificial**


Universidade Tecnológica Federal do Paraná (UTFPR)  
Câmpus Cornélio Procópio  
Departamento de Computação

# Conteúdo


- 
- 1 Regressão Logística
  - 2 K-nearest Neighbors
  - 3 Árvore de decisão
  - 4 Support Vector Machine

# Regressão Logística

## Visão geral do algoritmo

- 
- Método de Regressão Linear usando uma saída logística
  - Prediz a probabilidade de uma instância pertencer a cada classe
  - Seus coeficientes são interpretáveis
  - A fronteira de decisão gerada é linear ou polinomial

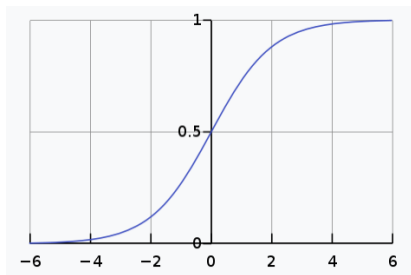
# Treinamento

- 
- Aprende uma função linear que melhor aproxima a entrada em saída
    - ▶ Encontra os valores dos coeficientes
  - Utiliza o método de Mínimos Quadrados (*Least Squares*)
  - Um *solver* roda o algoritmo de otimização

# Predição (Função Logística)

- Também chamada de Função *sigmoid*
- Gera um valor de probabilidade entre 0 e 1

$$f(x) = \frac{1}{1 + e^{-\beta_0 + \sum_{j=1}^d \beta_j x_j}}$$



# Hyperparâmetros

- **penalty**: Define o tipo de regularização
  - ▶ Opções: 'None', 'l1', 'l2', 'elasticnet'
  - ▶ Valor padrão: 'l2'
- **C**: Inverso da força de regularização
  - ▶ Opções: um *float* positivo
  - ▶ Valores pequenos definem uma regularização forte
  - ▶ Valor padrão: 1
- **multi\_class**: Transformação binária
  - ▶ Opções: 'auto', 'ovr', 'multinomial'
  - ▶ Valor padrão: 'auto'

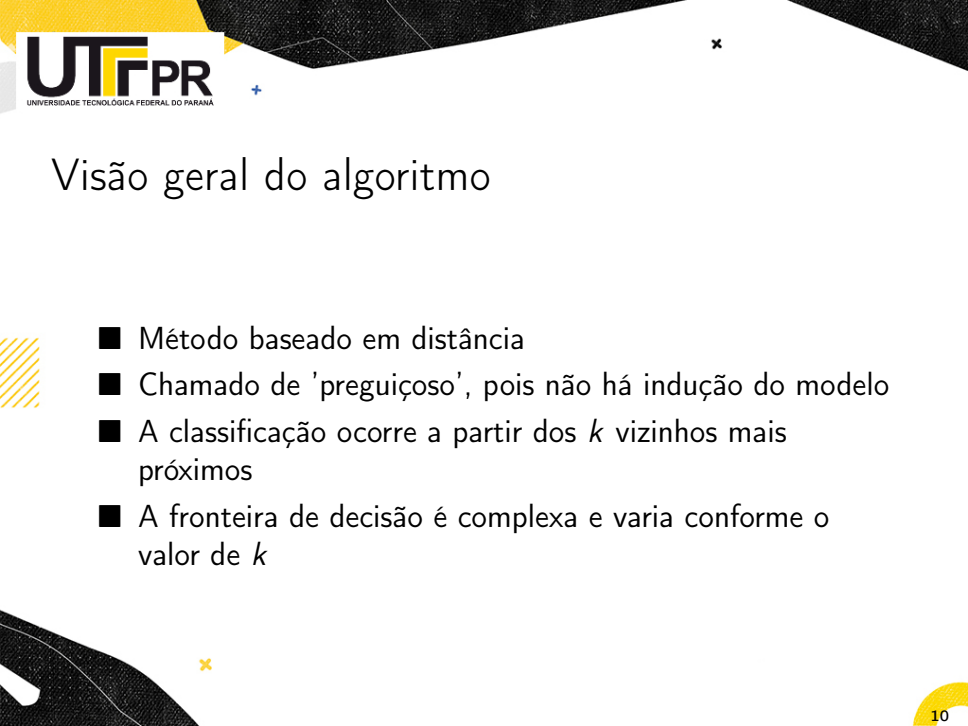
# Regularização

- **L1 (Lasso) :**
  - ▶ Remove atributos irrelevantes e redundantes
- **L2 (Ridge) :**
  - ▶ Diminui o *overfitting* deixando os coeficientes com valores menores
- **ElasticNet :**
  - ▶ Combina L1 e L2



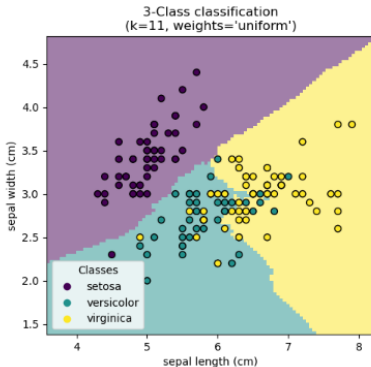
## K-nearest Neighbors

## Visão geral do algoritmo

- 
- Método baseado em distância
  - Chamado de 'preguiçoso', pois não há indução do modelo
  - A classificação ocorre a partir dos  $k$  vizinhos mais próximos
  - A fronteira de decisão é complexa e varia conforme o valor de  $k$


# Simulador

<http://vision.stanford.edu/teaching/cs231n-demos/knn/>



Fonte: <https://scikit-learn.org/stable/modules/neighbors.html>

# Treinamento

- 
- Existe um processo de construção de estruturas de dados para otimizar o cálculo das distâncias
  - Este processo é dispensável, portanto, o algoritmo Knn é apenas de predição

# Predição

- 1 Calcula a distância do ponto a ser classificado com todos os pontos do conjunto de treinamento
- 2 Seleciona os  $k$  vizinhos mais próximos
- 3 Retorna a classe mais frequente entre os  $k$  vizinhos
  - ▶ É possível ponderar pela distância

# Hyperparâmetros

- **n\_neighbors** : Número de vizinhos
  - ▶ Opções: um inteiro positivo
  - ▶ Valor padrão: 5
- **weights** : Como será calculado a classe
  - ▶ Opções: 'uniform', 'distance', callback
  - ▶ Valor padrão: 'uniform'
- **metric** : Medida de distância que será utilizada
  - ▶ Opções: 'cityblock', 'cosine', 'euclidean', 'minkowski'
  - ▶ Valor padrão: 'minkowski'


## A escolha do $k$

- Escolha que exerce grande impacto nas predições
- Valores pequenos a decisão fica restrita a áreas específicas  
sussetível ao *overfitting*
- Valores grandes a decisão fica genérica e dominado pela classe majoritária sussetível ao *underfitting*
- Uso de validação para determinar seu valor
  - ▶ Valor baixo e ímpar

# Distância Euclidiana

■ Medida padrão utilizado pelo algoritmo

■ Fórmula:

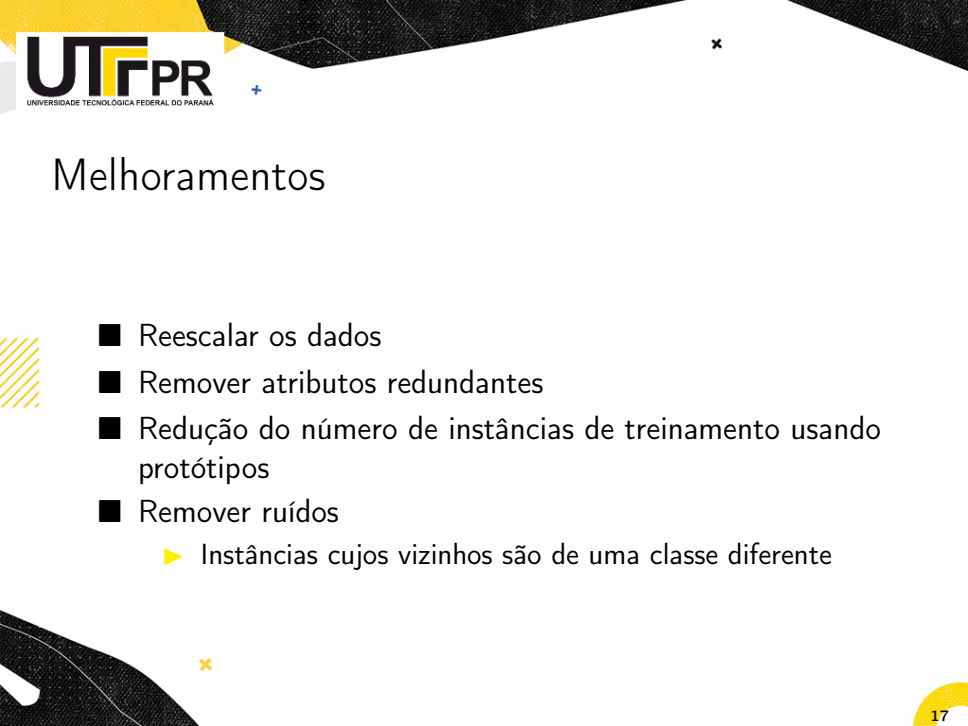

$$d(p, q) = \sqrt{\sum_{i=1}^n (q_i - p_i)^2}$$

■ Cuidados:

- ▶ Tipo de dados
- ▶ Escala dos números



# Melhoramentos

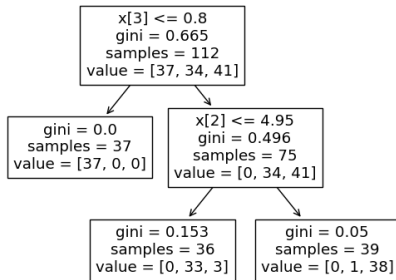
- 
- Reescalar os dados
  - Remover atributos redundantes
  - Redução do número de instâncias de treinamento usando protótipos
  - Remover ruídos
    - ▶ Instâncias cujos vizinhos são de uma classe diferente

# Árvore de decisão

## Visão geral do Algoritmo

- Modelo simbólico nativamente interpretável
- Os nós internos representam as decisões e as folhas representam as classes
- Há diferentes algoritmos para indução da árvore:
  - ▶ C4.5, C5.0, CART, ID3
- Principais diferenças:
  - ▶ Critério de divisão
  - ▶ Tipos dos nós internos
  - ▶ Critério de parada
  - ▶ Estratégia de poda

# Modelo



**Fonte:** [https://scikit-learn.org/stable/auto\\_examples/tree/plot\\_unveil\\_tree\\_structure.html](https://scikit-learn.org/stable/auto_examples/tree/plot_unveil_tree_structure.html)

# Treinamento

- ① Escolher um atributo que maximiza o critério de divisão
  - ▶ Gini
  - ▶ Ganho de informação
- ② Para cada parte da divisão, faça:
  - ▶ Verifica se o critério de parada foi alcançado
  - ▶ Repete a atividade 1
  - ▶ Repete a atividade 2
- ③ Podar a árvore gerada (opcional)

## Critério de divisão

- O critério de divisão é avaliado por uma função de qualidade
- Permite avaliar o critério de pureza da divisão
  - ▶ Quanto mais puro, melhor é a divisão
  - ▶ Quanto mais as instâncias de mesma classes são separadas mais puro
- Medidas:
  - ▶ *Gini Index*
  - ▶ *Information Gain*

## Gini Index

- Uma medida de dispersão que mensura a desigualdade do conjunto
- Mede a impureza, portanto quanto menor o valor melhor
- Fórmula:

$$Gini(nó) = 1 - \sum_{i=1}^q (P_i)^2$$

onde  $P_i$  é a probabilidade da classe  $i$  e  $q$  é o número de classes

## Cálculo Gini - Exemplo 1

■  $P(C1) = 50\%$  e  $P(C2) = 50\%$

$$\begin{aligned}
 Gini(nó) &= 1 - \sum_{i=1}^q (P_i)^2 \\
 &= 1 - (0.5^2 + 0.5^2) \\
 &= 1 - (0.25 + 0.25) \\
 &= 0.5
 \end{aligned}$$



## Cálculo Gini - Exemplo 2

■  $P(C1) = 65\%$  e  $P(C2) = 35\%$

$$\begin{aligned}
 Gini(nó) &= 1 - \sum_{i=1}^q (P_i)^2 \\
 &= 1 - (0.65^2 + 0.35^2) \\
 &= 1 - (0.4225 + 0.1225) \\
 &= 1 - 0.545 \\
 &\approx 0.45
 \end{aligned}$$

## Cálculo Gini - Exemplo 3

■  $P(C1) = 94\%$  e  $P(C2) = 6\%$

$$\begin{aligned}
 Gini(nó) &= 1 - \sum_{i=1}^q (P_i)^2 \\
 &= 1 - (0.94^2 + 0.06^2) \\
 &= 1 - (0.8836 + 0.0036) \\
 &= 1 - 0.8872 \\
 &\approx 0.11
 \end{aligned}$$

## Cálculo Gini - Exemplo 4

■  $P(C1) = 100\%$  e  $P(C2) = 0\%$

$$\begin{aligned}
 Gini(nó) &= 1 - \sum_{i=1}^q (P_i)^2 \\
 &= 1 - (1^2 + 0^2) \\
 &= 1 - (1 + 0) \\
 &= 0
 \end{aligned}$$

## Divisão com Gini

■ Para identificar a melhor divisão:

- ▶ Avaliar todos os atributos
- ▶ Avaliar todos os pontos de cortes

■ Fórmula:

$$Gini(Split) = \frac{N_{left}}{N_{total}} Gini(Left) + \frac{N_{right}}{N_{total}} Gini(Right)$$

## Exemplo de divisão

Atributo: [1.2, 1.8, 2.1, 2.5, 2.9, 3.1, 3.5, 3.7, 4.2, 4.5]  
Classe: ['A', 'A', 'B', 'A', 'A', 'B', 'B', 'B', 'B', 'A']

$$\text{Gini}(1.8): (2/10 * 0) + (8/10 * .469) = 0.375$$

$$\text{Gini}(2.1): (3/10 * 0.44) + (7/10 * .489) = 0.474$$


$$\text{Gini}(2.9): (5/10 * 0.32) + (5/10 * 0.32) = 0.32$$

$$\text{Gini}(4.2): (9/10 * 0.49) + (1/10 * 0) = 0.441$$



Melhor ponto de corte para o atributo analisado

## Critério de parada

- 
- Todas as instâncias de um nó são da mesma classe
  - Número de instâncias de um nó é menor do que um limiar pré-determinado
  - Não há um critério de divisão capaz de separar as classes

# Predição

- 1 Percorre a árvore da raiz a folha
  - ▶ Cada nó interno é uma decisão baseada no valor de um atributo
- 2 Prediz a respectiva classe da folha

## Hiperparâmetros

- **criterion** : Função para medir a pureza dos nós
  - ▶ Opções: 'entropy', 'log\_loss', 'gini', Padrão: 'gini'
- **max\_depth** : Profundidade máxima da árvore
  - ▶ Padrão: None
- **min\_samples\_split** : Número mínimo de instâncias para realizar uma divisão
  - ▶ Padrão: 2
- **min\_samples\_leaf** : Número mínimo de instâncias que um nó folha deve ter
  - ▶ Padrão: 1



## Vantagens e Desvantagens

- Interpretabilidade
- Suporte nativo para:
  - ▶ Multi-classe
  - ▶ Dados categóricos
  - ▶ Seleção de atributos
- Árvores profundas levam ao *overfitting*
- Sensível a ruídos
- Instabilidade a pequenas variações

# Support Vector Machine

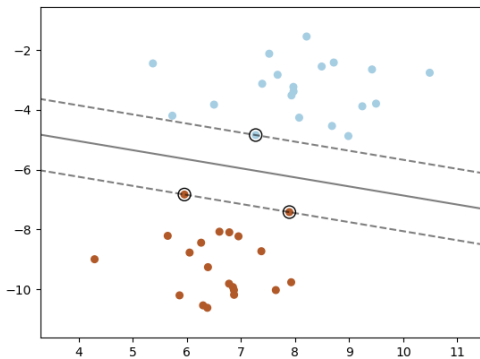
## Visão Geral do Algoritmo

- Algoritmo criado a partir da teoria do aprendizado estatístico
- Encontra a melhor fronteira linear (hiperplano) no espaço de atributos
  - ▶ Utiliza técnicas de otimização para encontrar a melhor margem de separação
- Utiliza transformações nos dados (*kernels*)

# SVM Linear

- Formaliza o problema de encontrar a melhor margem de decisão usando otimização com restrições usando Lagrange
- Os vetores de suporte são as instâncias posicionadas na margem
- Tipos de margem:
  - ▶ *Hard*: Classes devem ser linearmente separáveis
  - ▶ *Soft*: Permite que algumas instâncias ultrapassem a margem

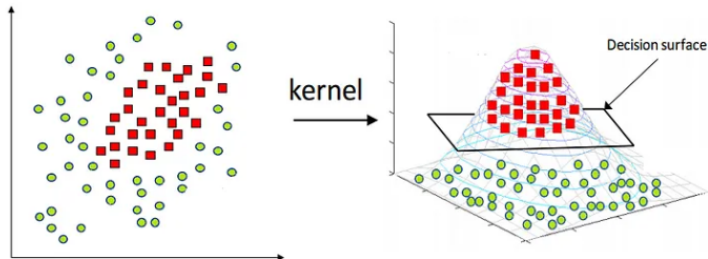
# Fronteira de decisão



# SVM Não linear

- Utiliza o conceito de *kernels* para transformação dos dados
  - ▶ Não converte as instâncias, mas o produto entre elas
- Kernels:
  - ▶ Linear
  - ▶ Polinomial
  - ▶ Gaussiano (*radial basis function*)
  - ▶ Customizável

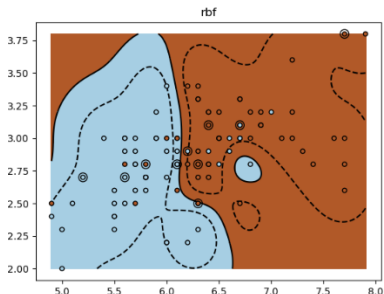
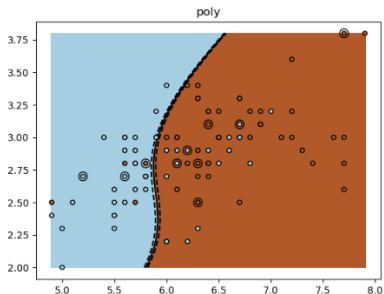
# Transformação com Kernels



Fonte:

<https://medium.com/@zxr.nju/what-is-the-kernel-trick-why-is-it-important-98a98db0961d>

# Fronteira de decisão com Kernels



Fonte: [https://scikit-learn.org/stable/auto\\_examples/exercises/plot\\_iris\\_exercise.html](https://scikit-learn.org/stable/auto_examples/exercises/plot_iris_exercise.html)



# Hiperparâmetros

- **kernel** : Escolha do Kernel
  - ▶ Opções: 'linear', 'poly', 'rbf', 'sigmoid', Padrão: 'rbf'
- **C** : Parâmetro de regularização
  - ▶ Padrão: 1
- **gamma** : Coeficiente do kernel
  - ▶ Opções: 'scale', 'auto', float, Padrão: 'scale'
  - ▶ Scale:  $\frac{1}{n\_features * X.var()}$
  - ▶ Auto:  $\frac{1}{n\_features}$
  - ▶ O float deve ser não negativo

## Hiperparâmetro de custo (C)

### ■ Valores baixos (ex: 1) fazem a fronteira de decisão suave

- ▶ Permite que existam erros de classificação
- ▶ Útil quando há ruídos e *outliers*
- ▶ Irá aumentar a largura da margem


### ■ Valores altos (ex: 1000) penaliza erros de classificação

- ▶ Não tolera erros de classificação
- ▶ Gera fronteiras de decisão mais complexas
- ▶ Pode levar ao *overfitting*
- ▶ Irá diminuir a largura da margem

## Hiperparâmetro Gamma

- Valores baixos (ex: 0.1) usam mais pontos para definir a fronteira de decisão
  - ▶ Cada instância possui menos influência
  - ▶ Recomendado para grandes conjuntos de dados ou dados com ruídos
- Valores altos (ex: 10) usam menos pontos para a definir a fronteira de decisão
  - ▶ Algumas instâncias possuem mais influência
  - ▶ Não recomendado para pequenos conjuntos de dados

## Vantagens e Desvantagens

- 
- Versatilidade com o uso dos Kernels
  - Generalização robusta (evita *overfitting*)
  - Suporta espaços com alta dimensões
  - Interpretabilidade reduzida
  - Sensível a ruídos
  - Demanda grande custo computacional
  - Escolha correta dos hiperparâmetros