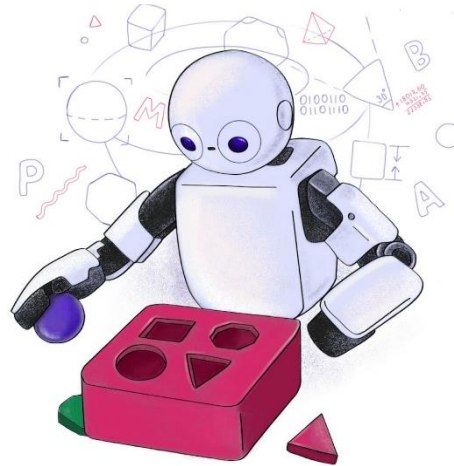


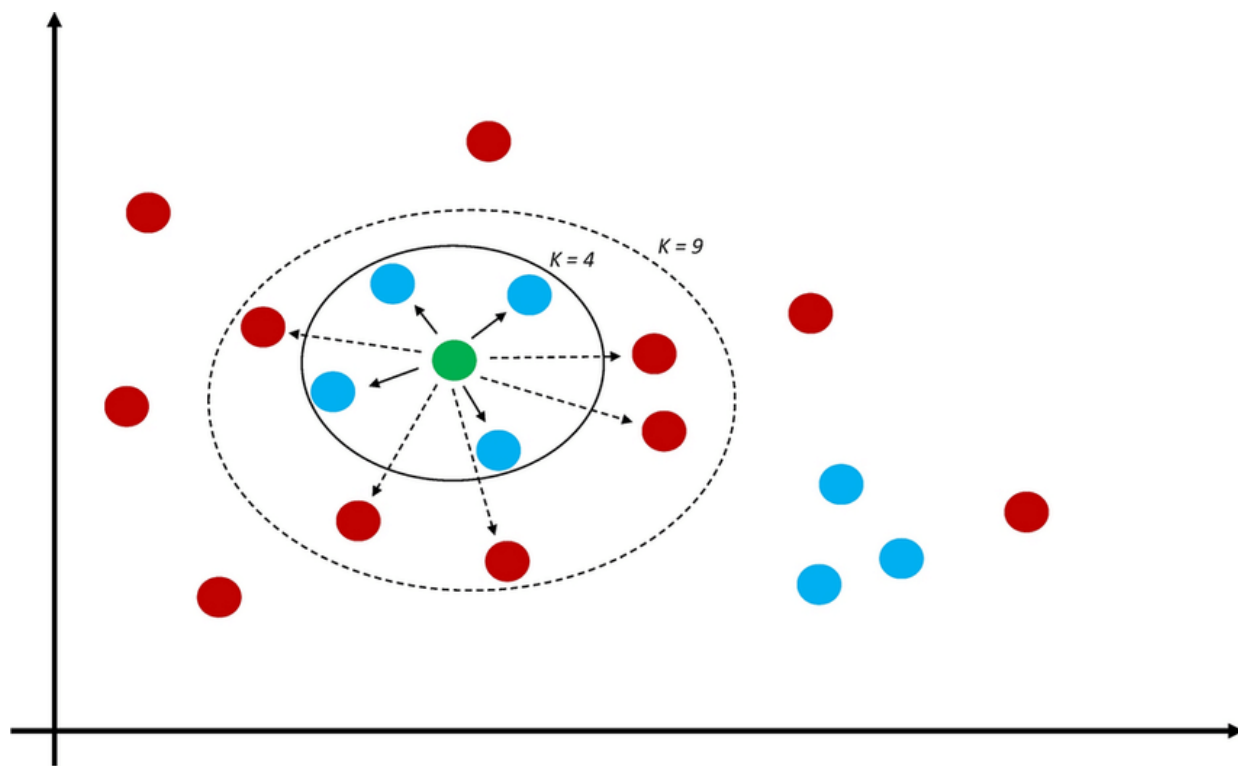
TP555 - Inteligência Artificial e Machine Learning: *k-Vizinhos mais Próximos*



Inatel

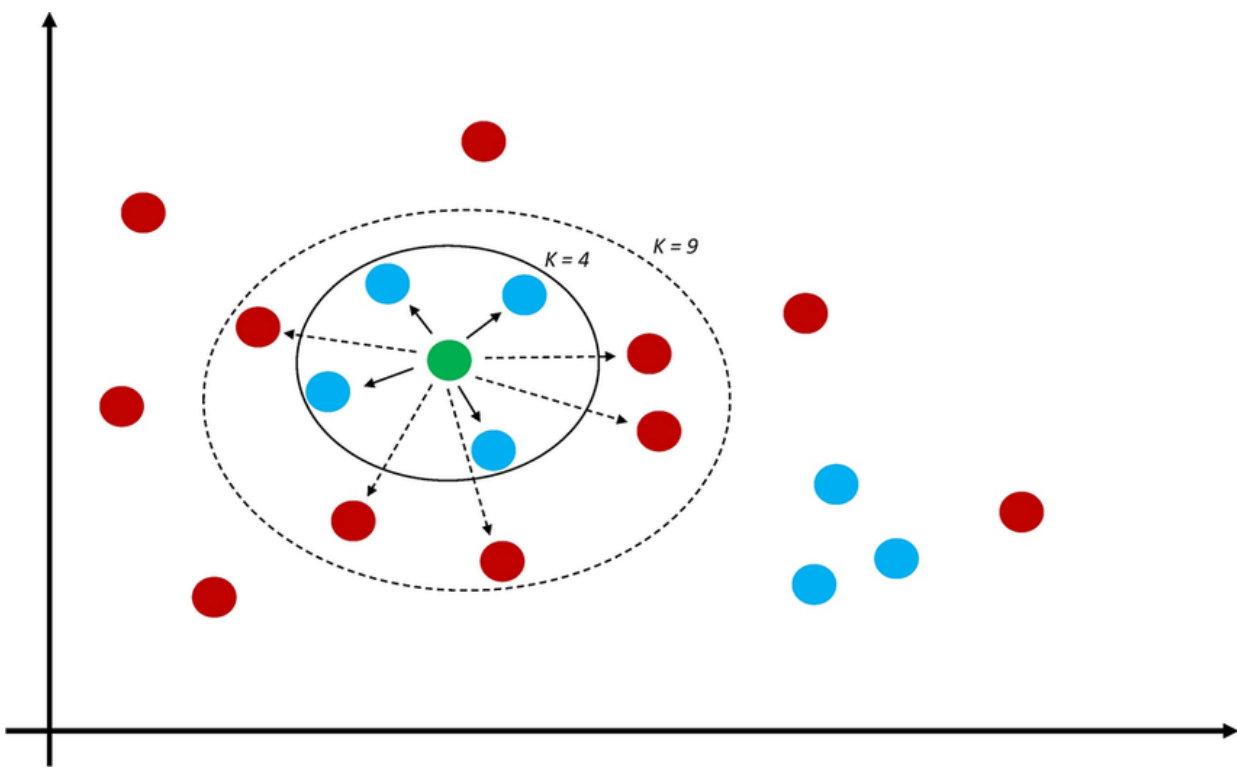
Felipe Augusto Pereira de Figueiredo
felipe.figueiredo@inatel.br

k-vizinhos mais próximos (k-NN)



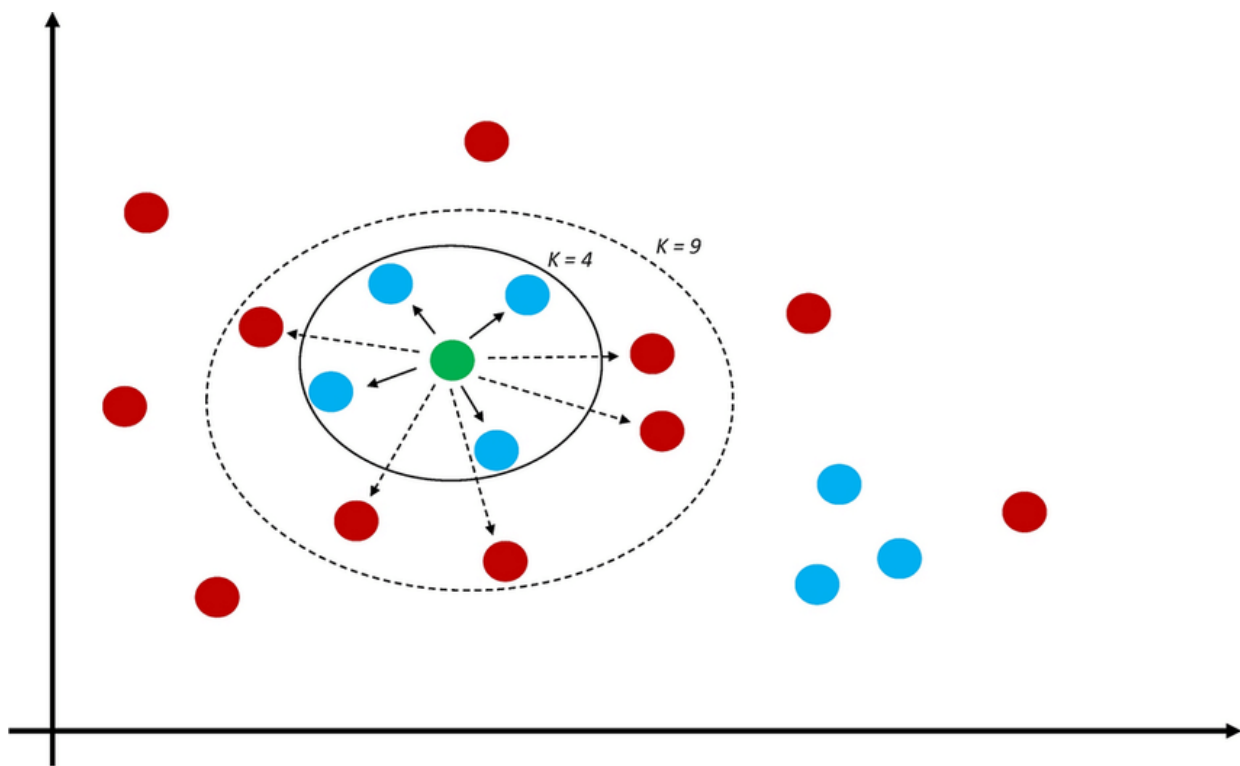
- O algoritmo k-NN (do inglês, *k-Nearest Neighbours*) é um dos algoritmos mais simples de **aprendizado supervisionado** para se resolver problemas tanto de **classificação** quanto de **regressão**.

k-vizinhos mais próximos (k-NN)



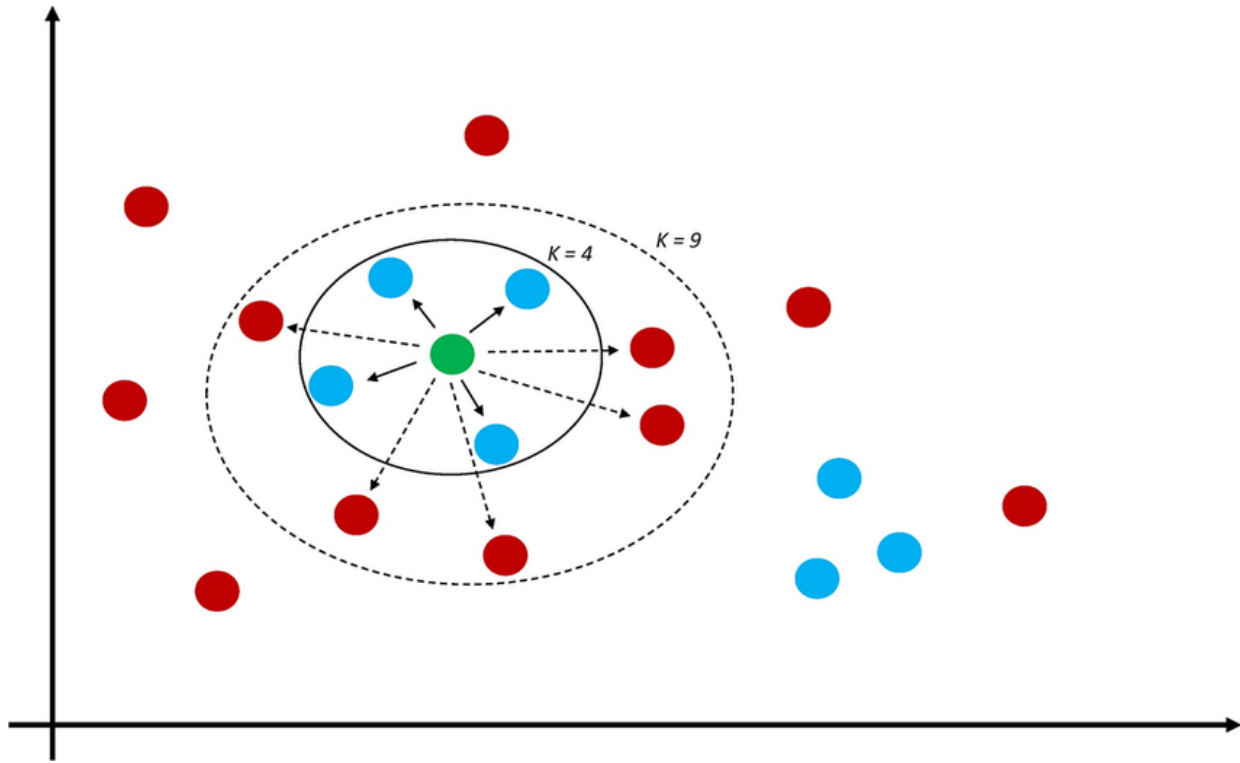
- É um algoritmo do tipo **não-paramétrico**, pois diferentemente dos outros algoritmos que vimos até o momento
 - **não há um modelo a ser treinado** (e.g., polinômio com um número definido de pesos - Regressão/Classificação logística),
 - **tampouco se faz qualquer suposição a respeito da distribuição dos dados** (e.g., modelos Naive Bayes - Bernoulli ou Gaussiano).

k-vizinhos mais próximos (k-NN)



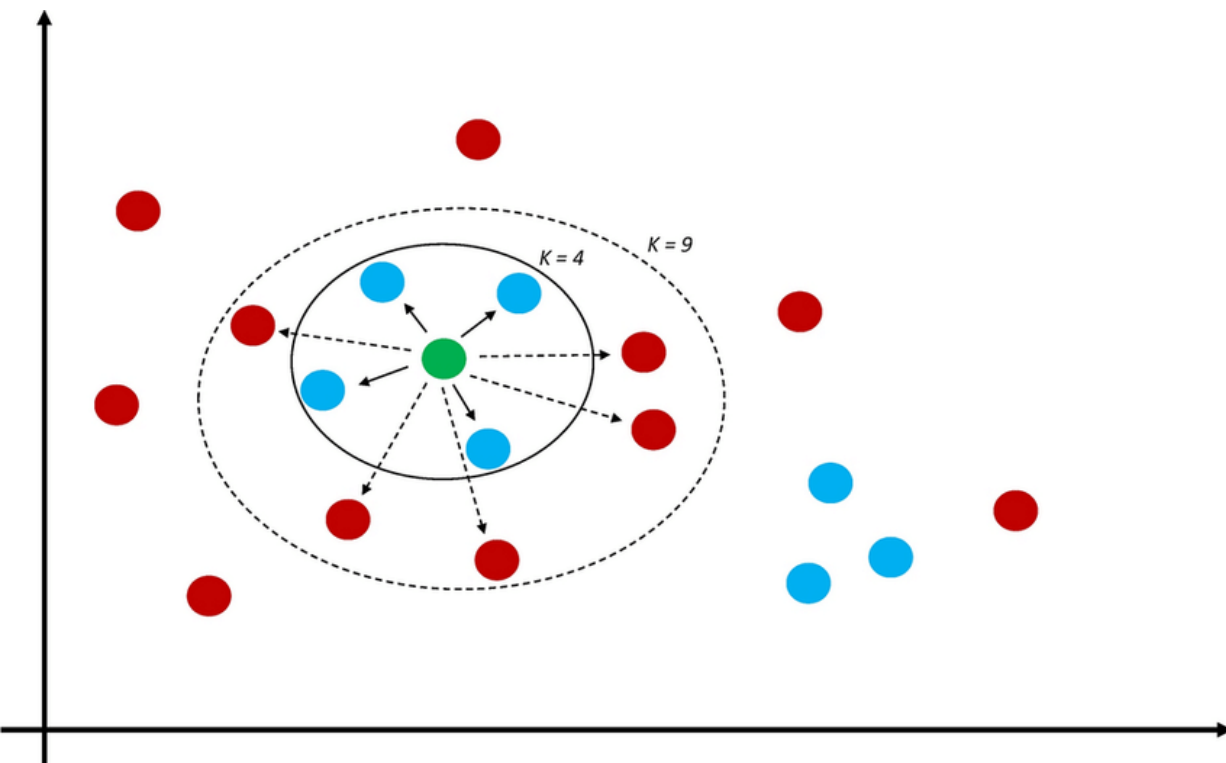
- Ele *usa diretamente os exemplos de treinamento* para tomar decisões.
- A única suposição é que uma *medida de distância entre dois exemplos* (i.e., vetores de atributos) *possa ser calculada*.

Funcionamento



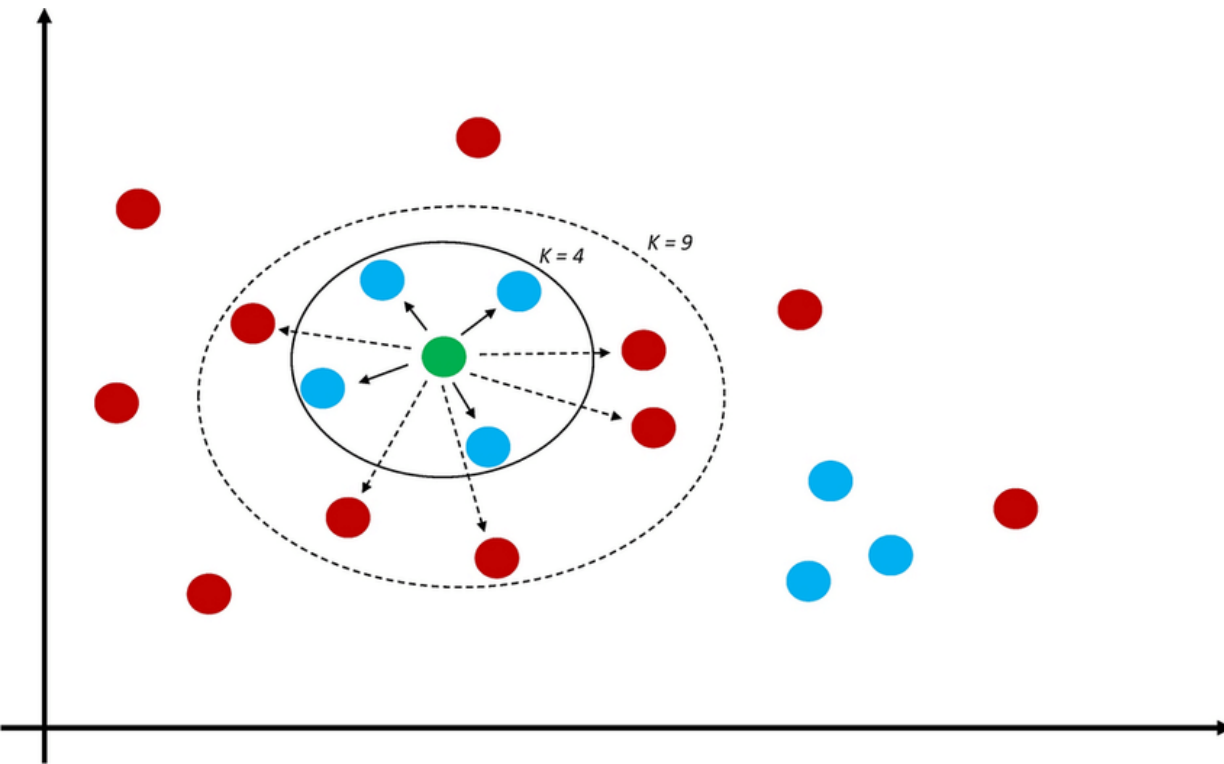
- O algoritmo necessita que todos os *exemplos de treinamento*, $\mathbf{x}(i) = [x_1(i) \quad \cdots \quad x_K(i)] \in \mathbb{R}^{K \times 1}$ e seus respectivos rótulos, $y(i)$, $i = 0, \dots, N - 1$, sejam *armazenados em memória*.

Funcionamento



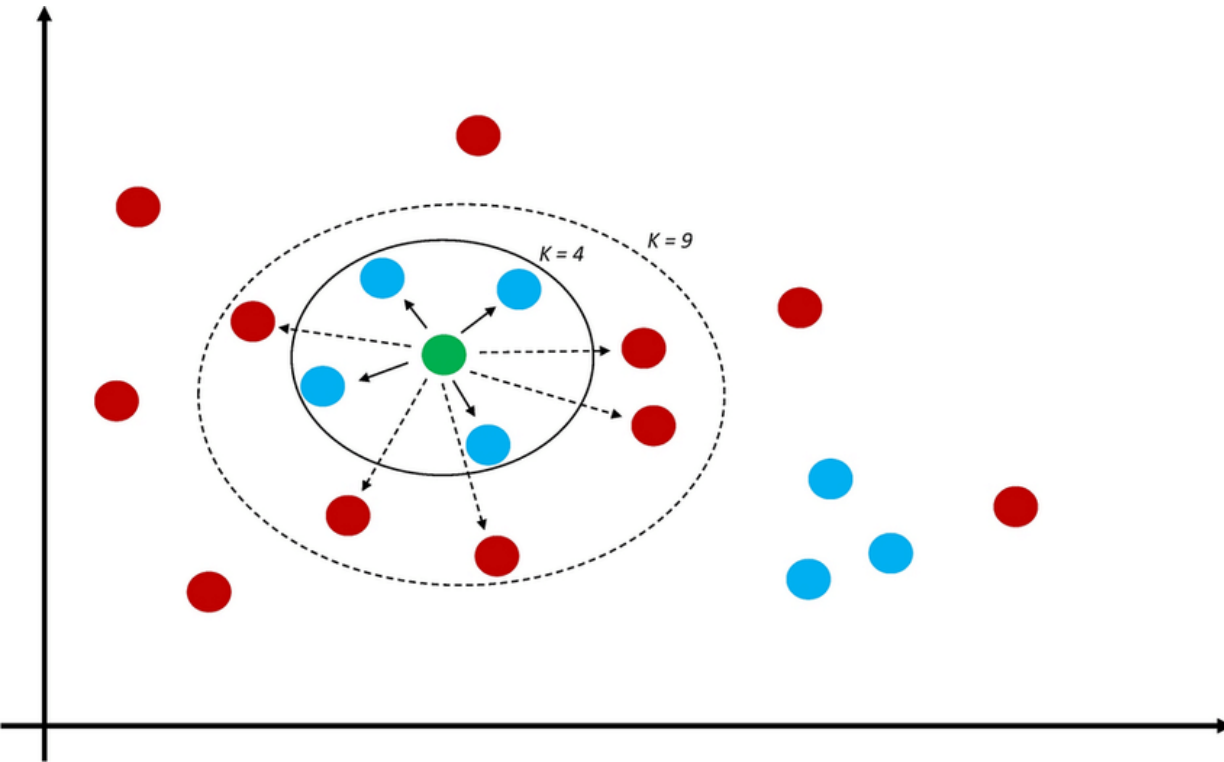
- Em seguida, dado um **exemplo de entrada inédito**, x' (e.g., círculo verde), a saída para este exemplo **dependerá dos rótulos** associados aos **k exemplos de treinamento mais próximos** do exemplo de entrada x' no espaço de atributos.

Funcionamento



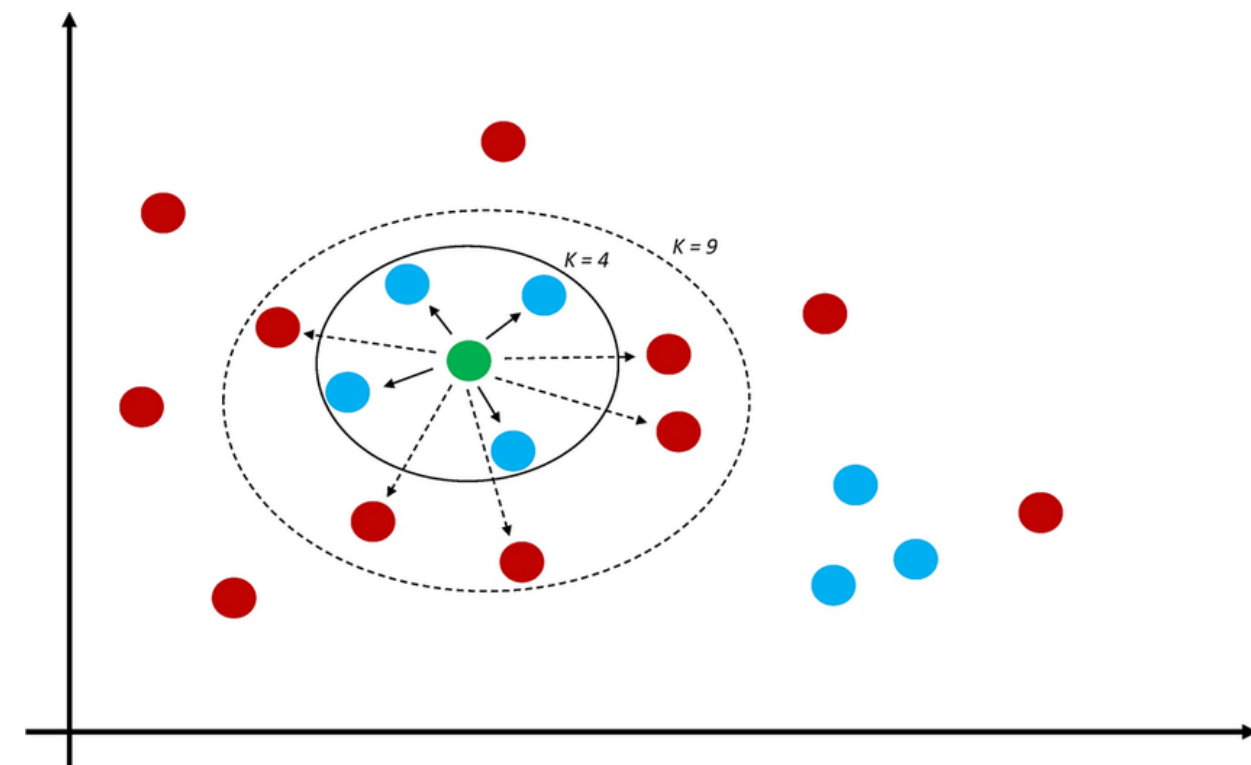
- Para *classificação*, a *classe mais frequente* entre os k vizinhos mais próximos é escolhida como a classe do exemplo de entrada.
- Para *regressão*, os valores associados aos k vizinhos mais próximos são usados para calcular um *valor médio ou ponderado*, que será a estimativa para o exemplo de entrada.

Funcionamento



- Ou seja, o algoritmo usa *similaridade/proximidade entre vetores de atributos* para prever os valores de quaisquer novos exemplos.
- Isso significa que um novo exemplo de entrada recebe um valor de saída com base na sua *proximidade com os exemplos do conjunto de treinamento*.

Funcionamento

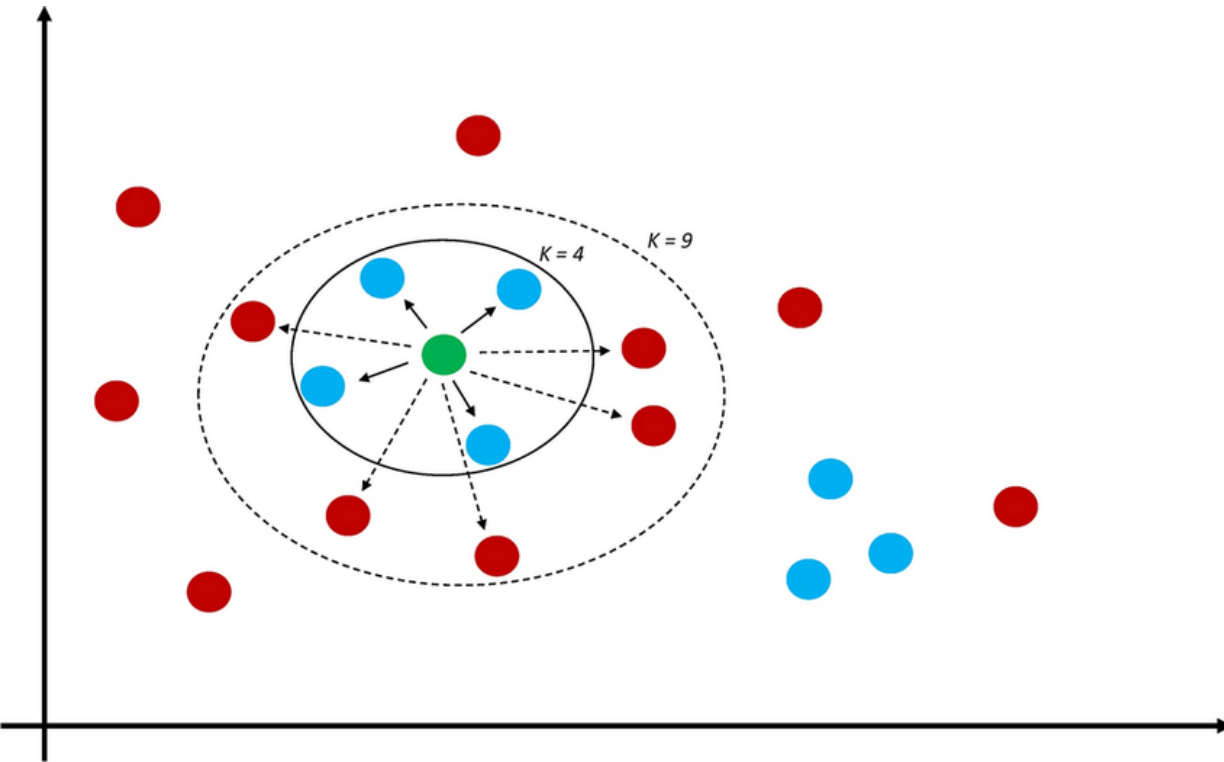


- Por exemplo, para regressão, nós podemos tomar a **média aritmética dos rótulos dos k vizinhos mais próximos**:

$$\hat{y}(x') = \frac{1}{k} \sum_{x(i) \in N_k(x')} y(i),$$

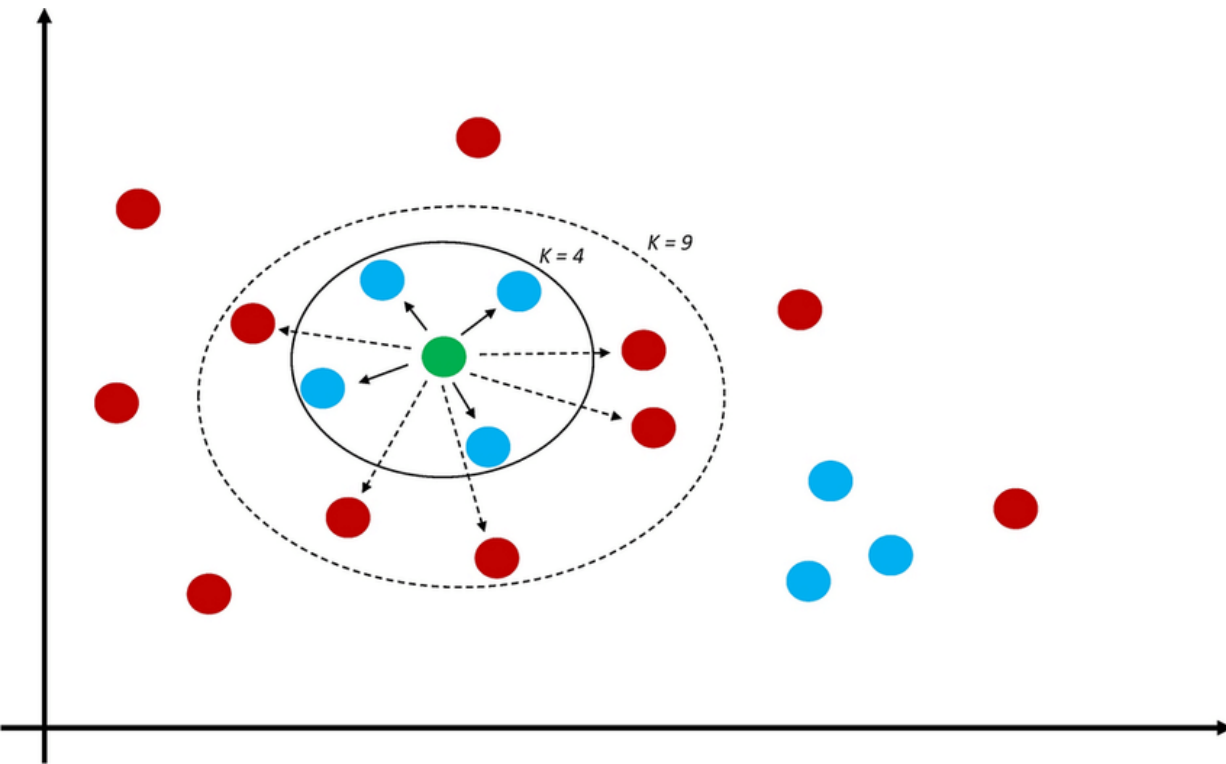
onde $N_k(x')$ é a vizinhança de x' , formada pelos exemplos de treinamento $x(i)$ que correspondem aos k vizinhos mais próximos de x' .

Funcionamento



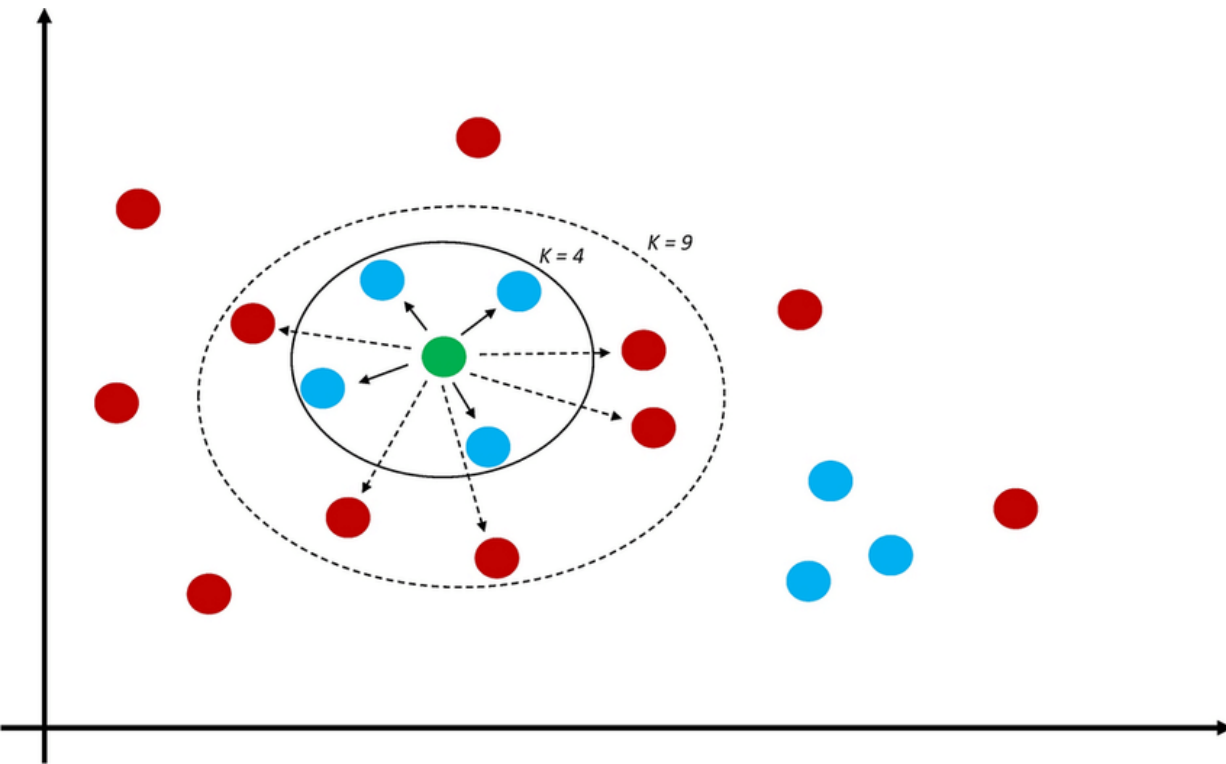
- Para classificação, por exemplo, dentre k vizinhos mais próximos, escolhemos a classe com maior número de exemplos (i.e., voto majoritário).
- **OBS.:** Não confundam o *número de atributos*, K , com o *número de vizinhos mais próximos*, k .

k-Vizinhos mais Próximos (k-NN)



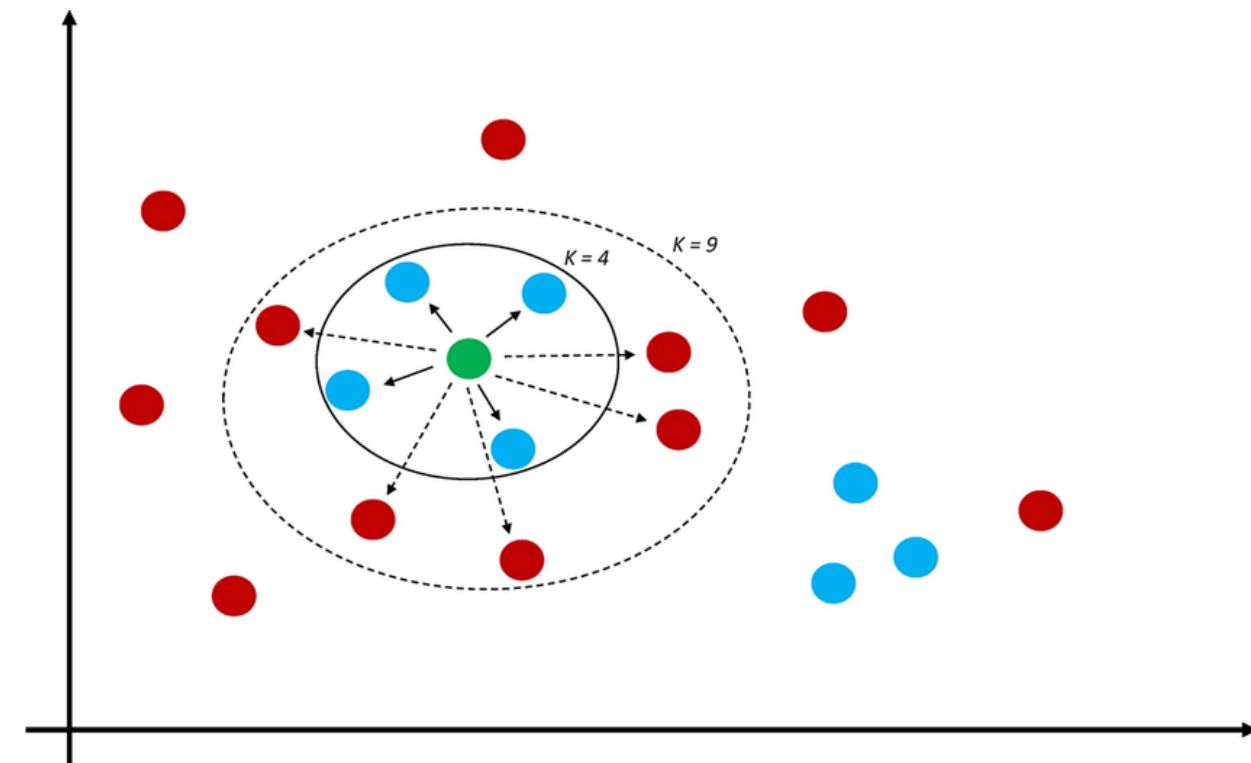
- Portanto, o uso do k-NN envolve a **definição** de:
 - Uma **métrica de distância** que deve ser **calculada no espaço de atributos** a fim de identificar os vizinhos mais próximos.
 - Um **valor para o hiperparâmetro k** , ou seja, a escolha do número de vizinhos que devem ser levados em consideração para a geração da saída correspondente ao exemplo de entrada, x' .

k-Vizinhos mais Próximos (k-NN)



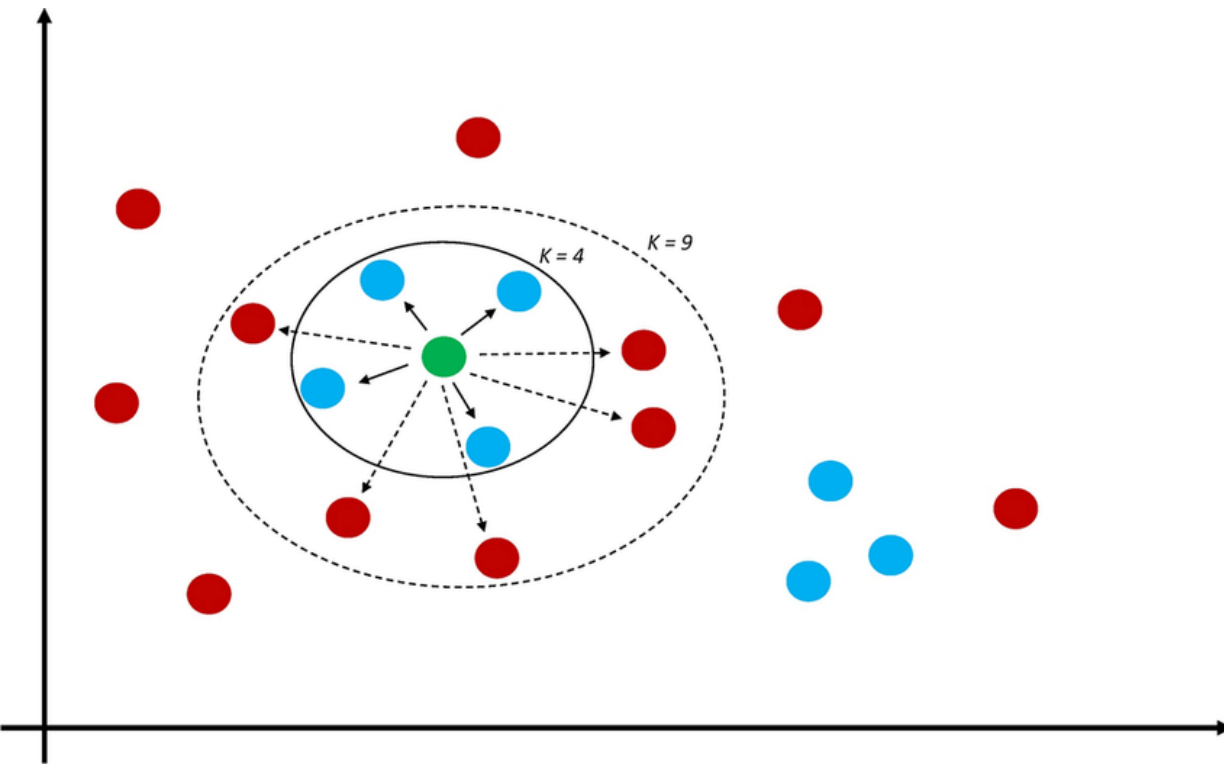
- Como k é um **hiperparâmetro** do k-NN, devemos usar técnicas de **validação cruzada** para encontrar o melhor valor de k .
- Podemos utilizar por exemplo **Grid Search** ou **Random Search com k-fold**.

k-Vizinhos mais Próximos (k-NN)



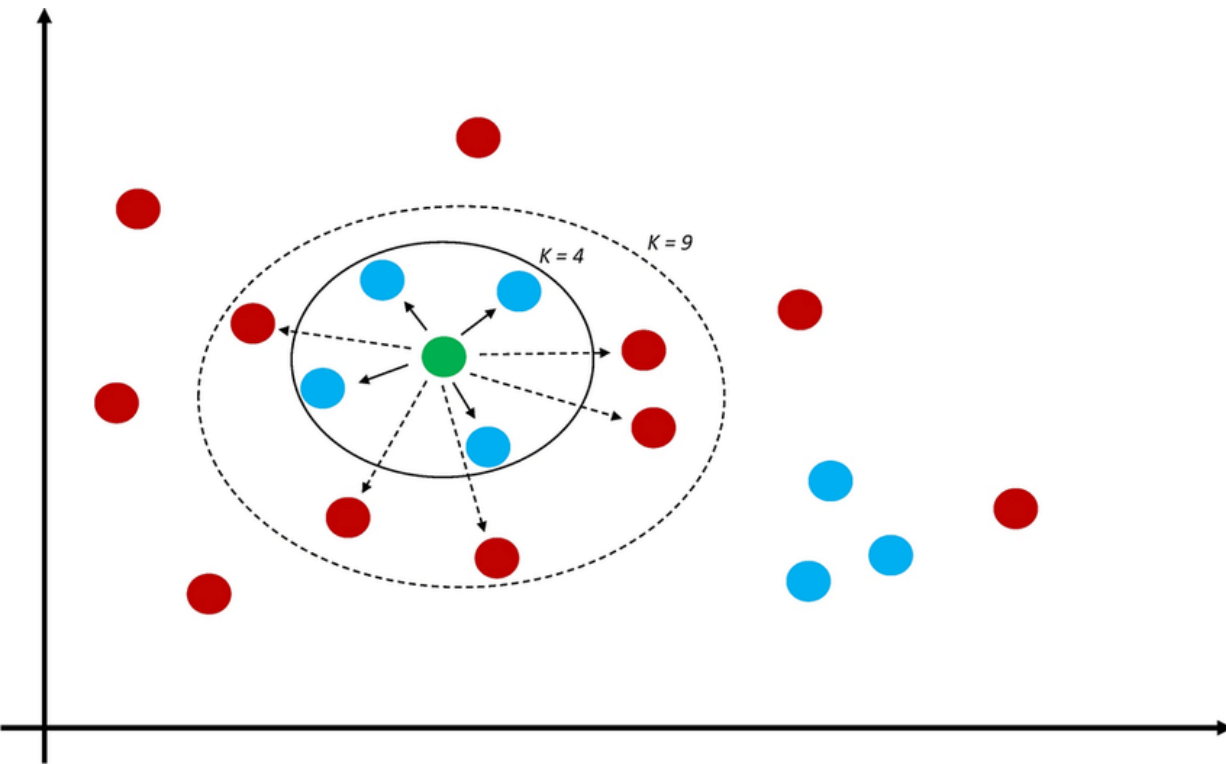
- Devido às suas características, o k-NN é visto como um **algoritmo de aprendizado competitivo**, uma vez que os **elementos do modelo** (que são os próprios exemplos de treinamento) **competem entre si pelo direito de influenciar a saída do algoritmo** quando a **medida de similaridade (distância)** é calculada para cada novo dado de entrada.

k-Vizinhos mais Próximos (k-NN)



- O k-NN explora a ideia conhecida como *lazy learning*, uma vez que o algoritmo *não “constrói” um modelo até o instante em que uma predição é solicitada*.
 - Ou seja, *não existe uma etapa explícita de treinamento/aprendizado*.
 - Em vez disso, todo *o aprendizado é adiado até a fase de predição*.

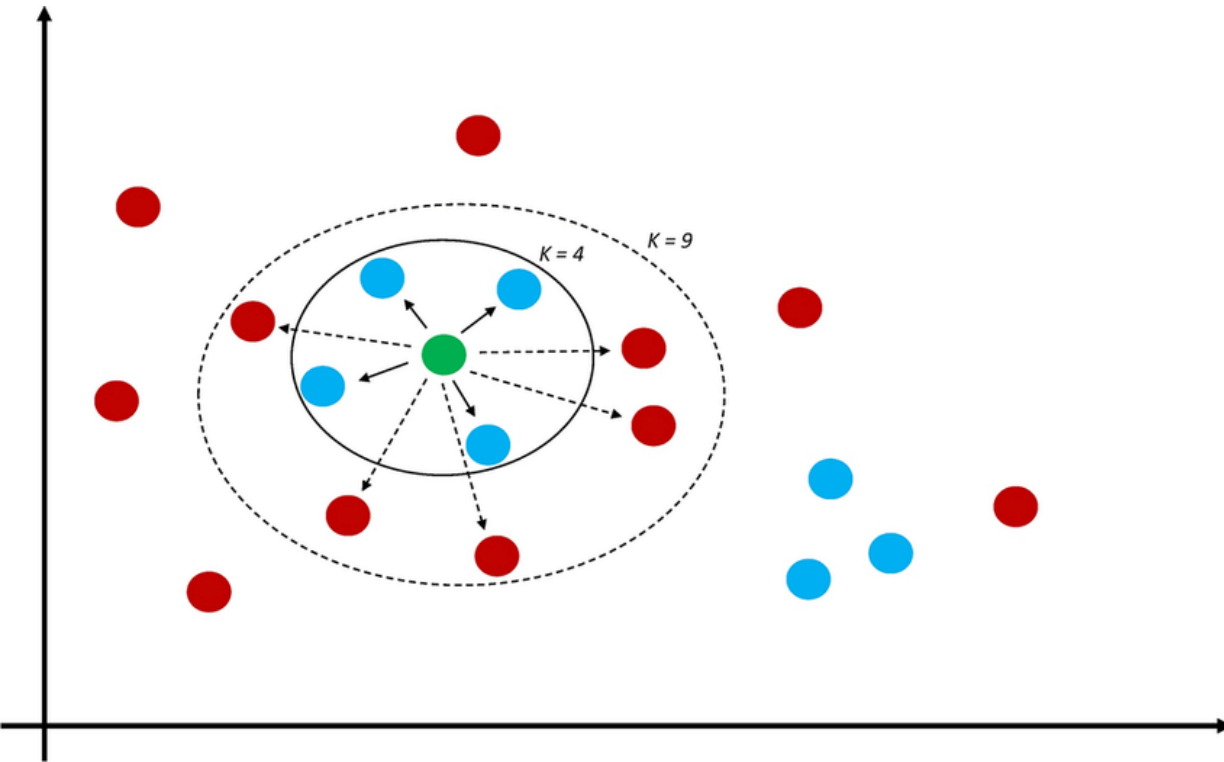
k-Vizinhos mais Próximos (k-NN)



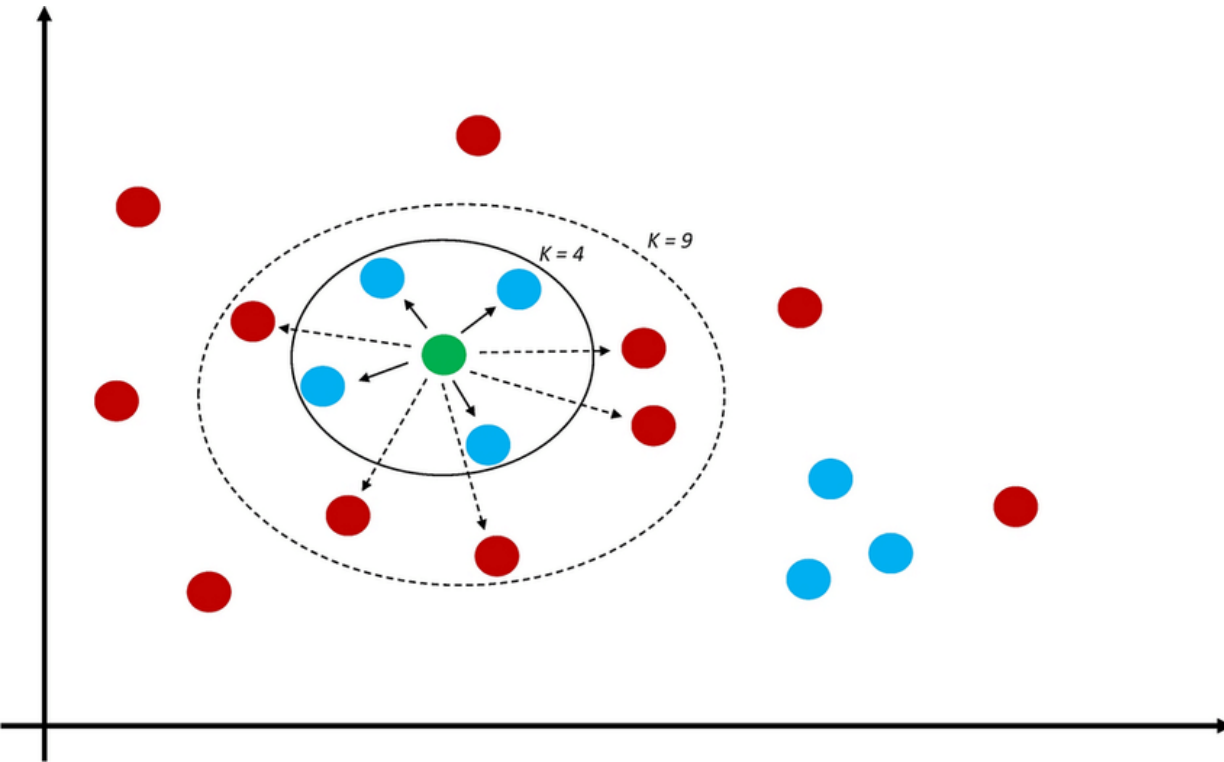
- O k-NN segue o paradigma de ***aprendizado-baseado em exemplos***, onde ao invés de se treinar um modelo a partir do conjunto de treinamento, ele ***compara exemplos de entrada com os exemplos do conjunto de treinamento armazenados em memória***.

Desvantagens

- Todos os dados de treinamento precisam ser *armazenados* e *consultados* para se identificar os k vizinhos mais próximos.

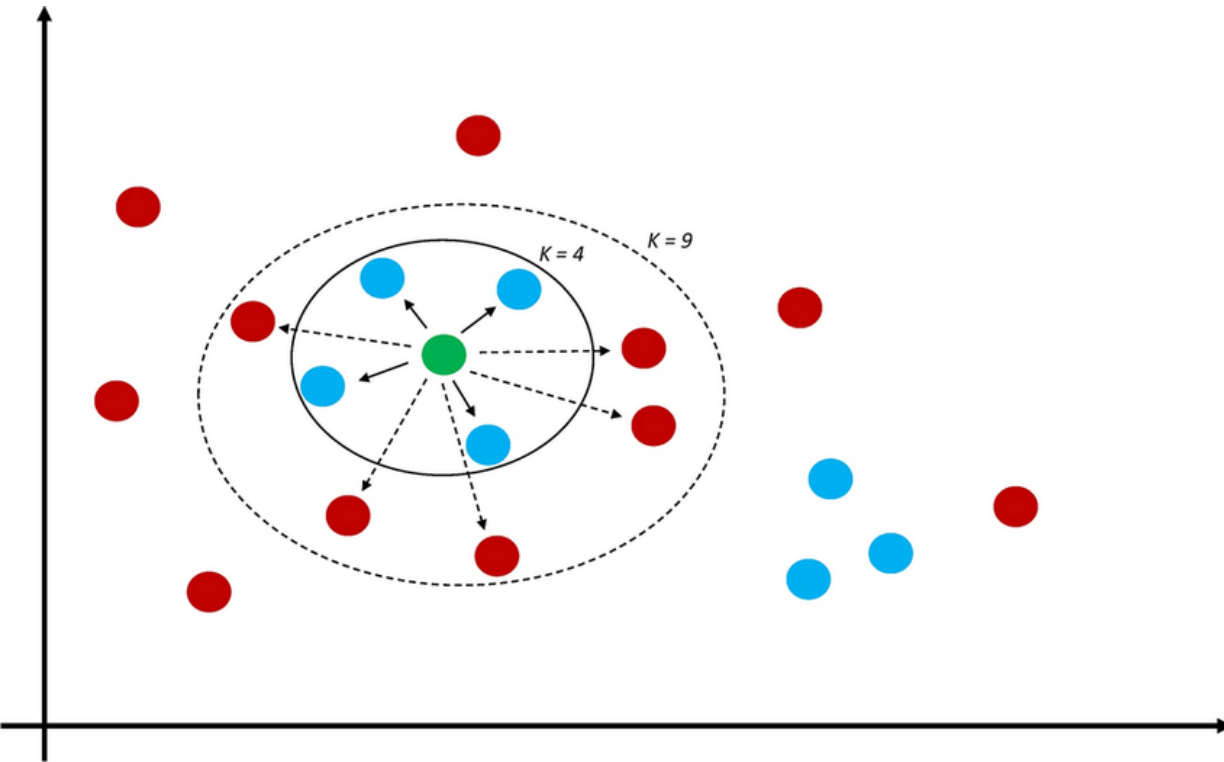


Desvantagens



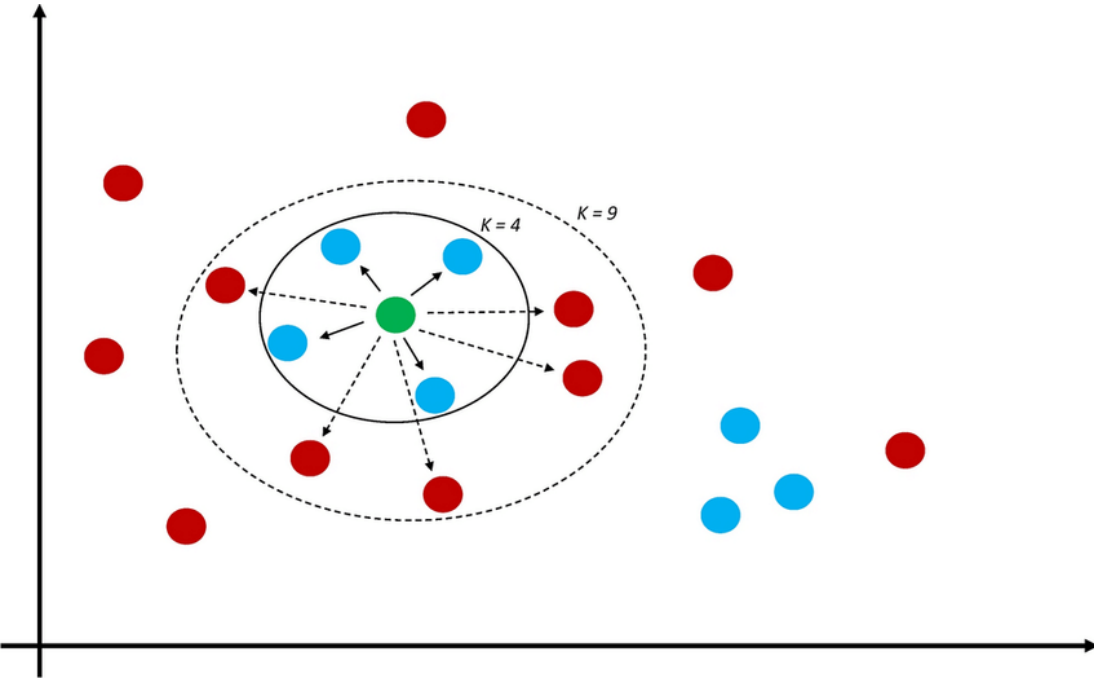
- Portanto, a ***predição*** poderá ser ***demorada*** dependendo do tamanho do conjunto de treinamento, pois deve-se ***calcular a distância entre o exemplo de entrada e todos os exemplos do conjunto de treinamento***.

Desvantagens



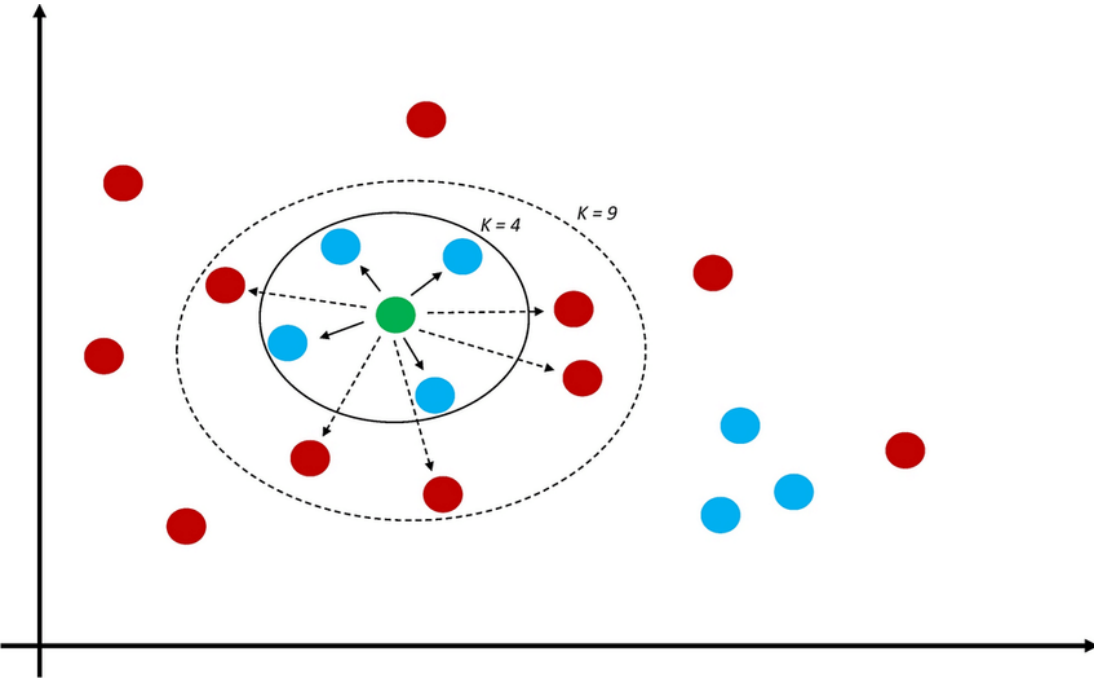
- Além disto, como vimos, o **conjunto de treinamento deve ser armazenado em memória**, e caso esse conjunto seja muito grande, pode não haver memória o suficiente para armazená-lo.

Outras desvantagens: Sensibilidade à dimensionalidade



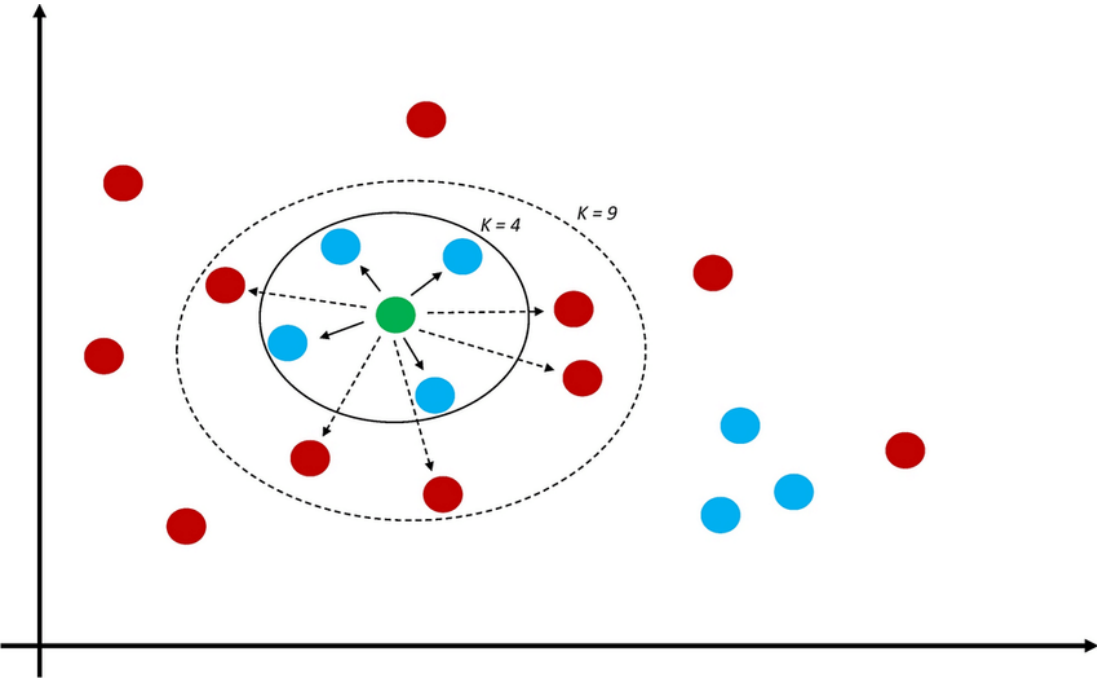
- O desempenho do k-NN pode ser afetado quando o conjunto de dados possui *alta dimensionalidade*.
- Isso ocorre porque, em espaços de alta dimensão, *a noção de proximidade entre exemplos pode se tornar menos significativa, levando a resultados menos precisos*.

Outras desvantagens: Sensibilidade à dimensionalidade



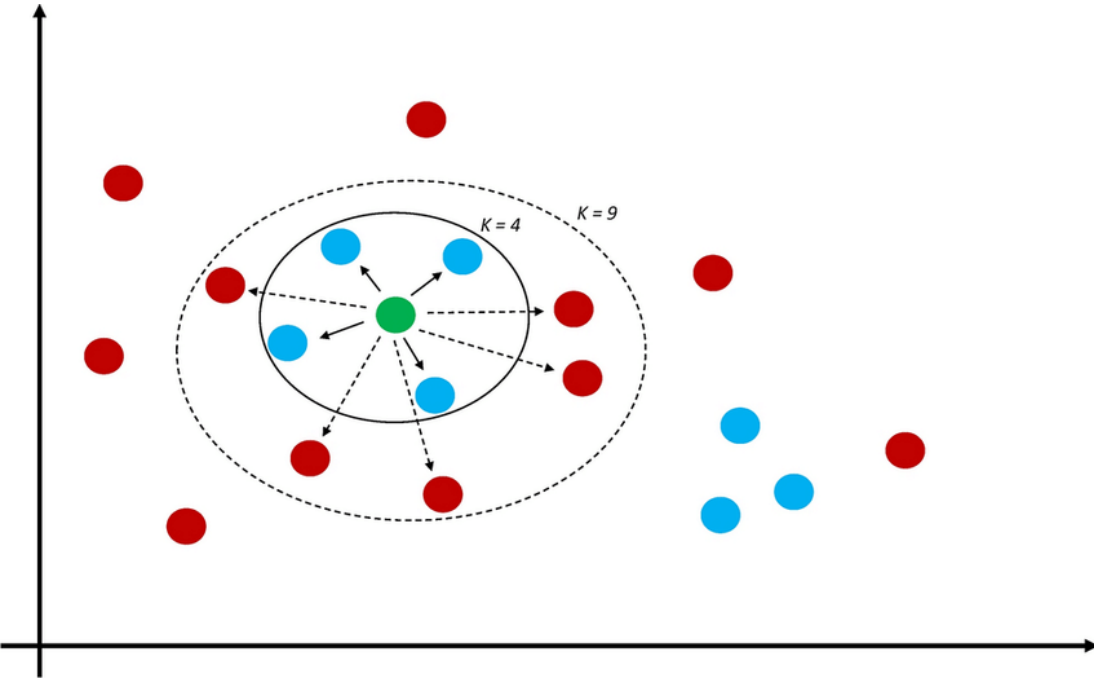
- À medida que as dimensões aumentam, os dados tornam-se mais **esparsos**, dificultando a busca por vizinhos próximos significativos.
 - **Esparsidade**: exemplos ficam mais distantes uns dos outros.
- A esparsidade faz com que existam **menos pontos vizinhos dentro de uma determinada distância**.

Outras desvantagens: Sensibilidade à dimensionalidade



- Além disso, em *espaços de alta dimensionalidade* as *distâncias* entre pontos *se tornam mais uniformes* (i.e., *equidistante*), o que *reduz a relevância do conceito de vizinhança*.
- Isso degrada o desempenho do k-NN, pois sua *suposição de que pontos próximos têm rótulos semelhantes pode não se aplicar* em alta dimensionalidade.
- Além disso, o *cálculo da distância se torna mais custoso computacionalmente*.

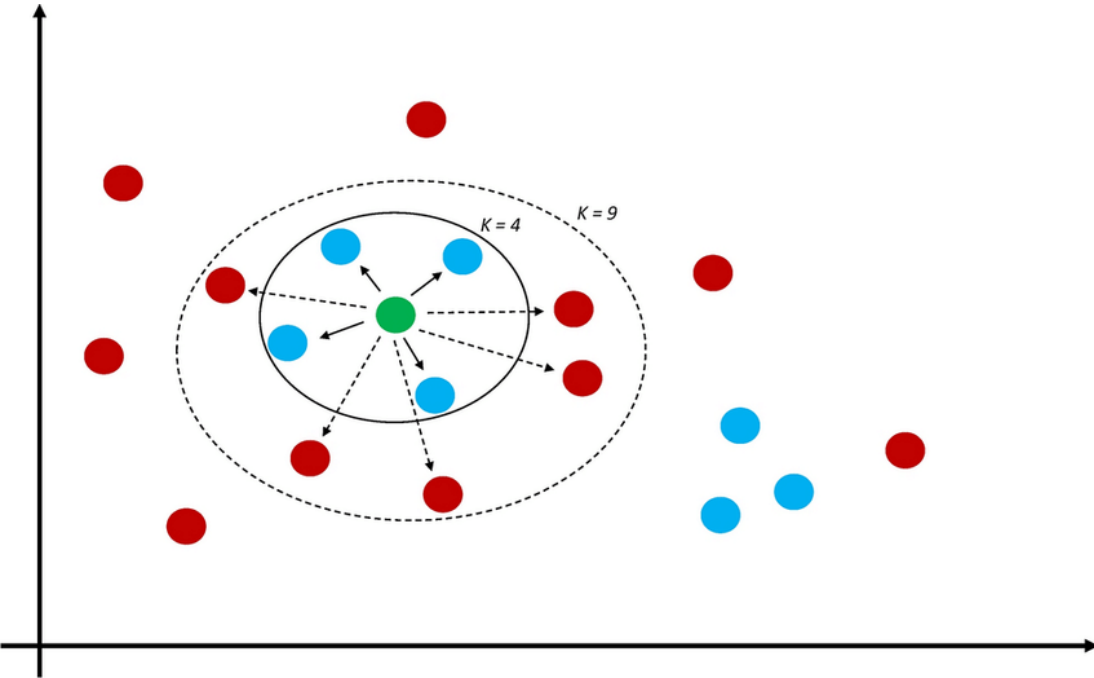
Outras desvantagens: Sensibilidade a dados desbalanceados



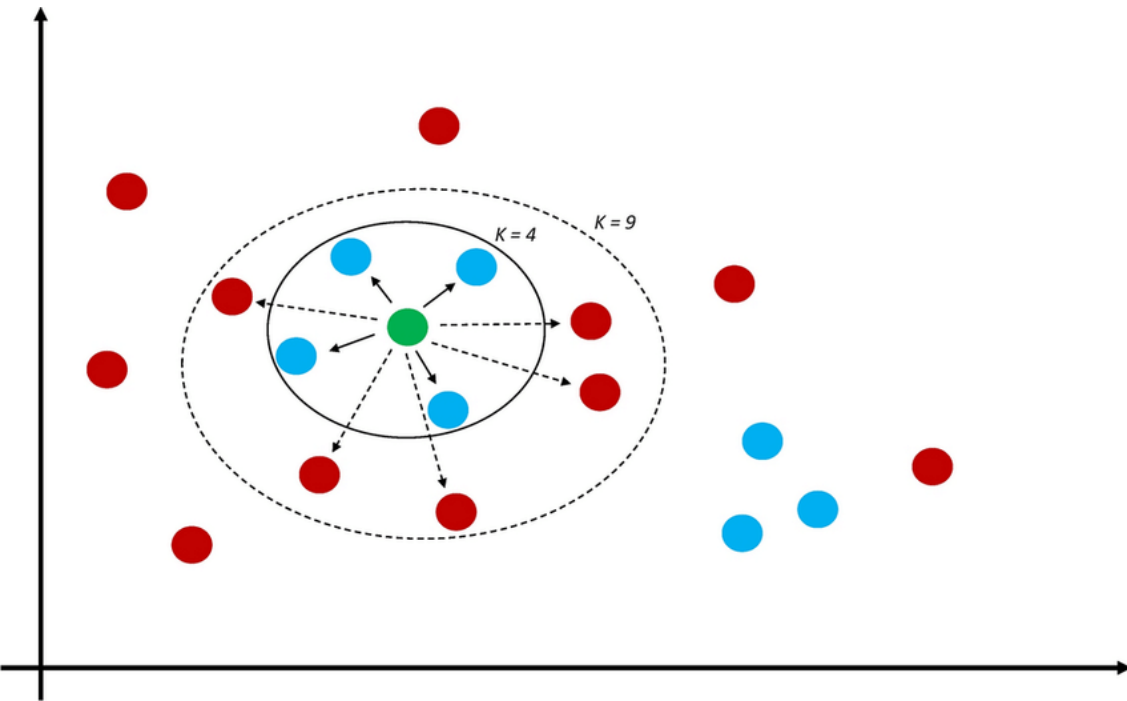
- Se as classes estiverem desbalanceadas, o k-NN pode se tornar *enviesado em direção à classe majoritária*.
- Isso ocorre porque, ao selecionar os k vizinhos mais próximos, é mais provável que sejam selecionados exemplos da classe majoritária.

Outras desvantagens: Necessidade de pré-processamento de dados

- O k-NN é *sensível a atributos com diferentes escalas*.
- Portanto, em geral, *é necessário realizar escalonamento*, como normalização ou padronização, para garantir que todos os *atributos* tenham uma *contribuição similar no cálculo da distância*.

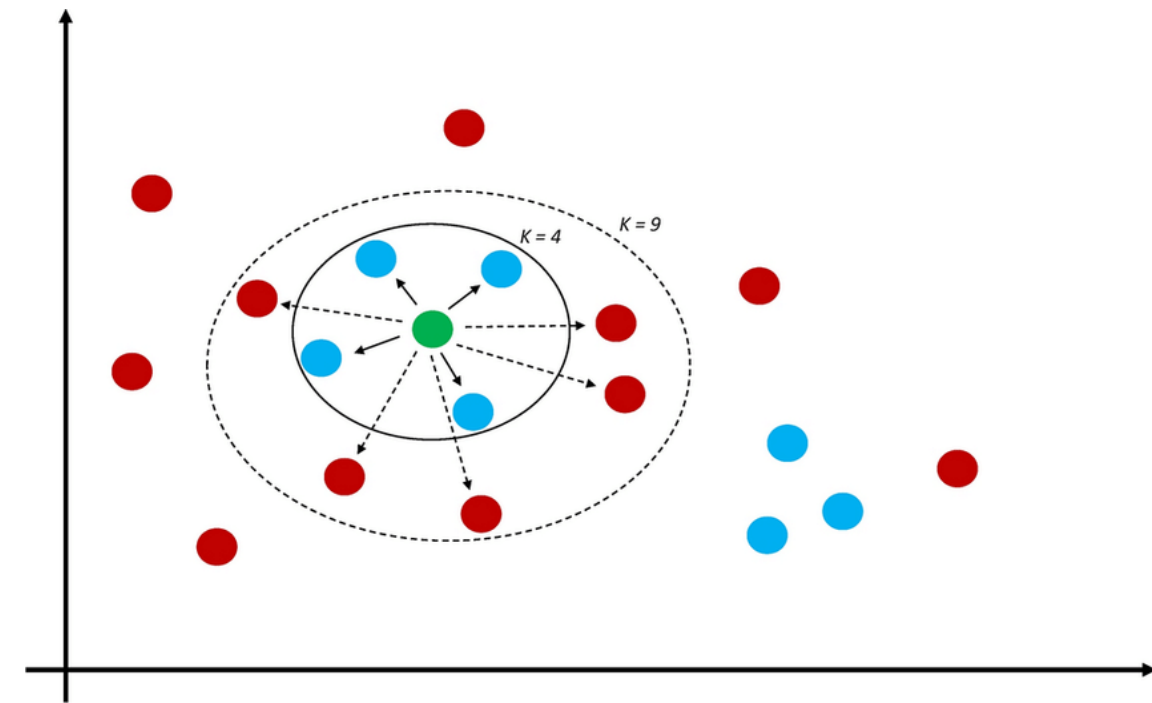


Métricas de distância



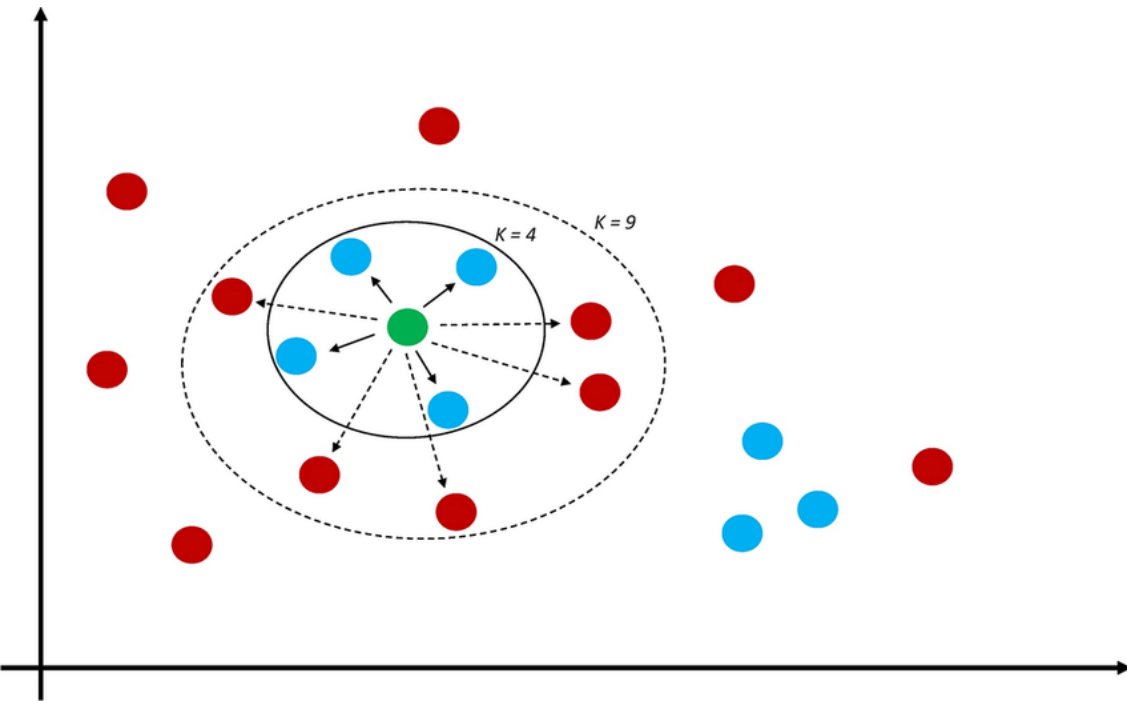
- **Definição:** Uma métrica de distância fornece a *distância entre os elementos de um conjunto*.
- Se a *distância* é igual a *zero*, os *elementos são equivalentes*, *caso contrário, os elementos são diferentes* uns dos outros.

Métricas de distância



- No nosso caso, a métrica serve para *medir a distância/similaridade* entre os *K atributos do vetor de entrada* e os *K atributos dos vetores do conjunto de treinamento*.
- Existem várias *métricas de distância*, mas vamos discutir apenas as mais utilizadas através de uma *métrica de distância generalizada*, chamada de *distância de Minkowski*.

Métricas de distância

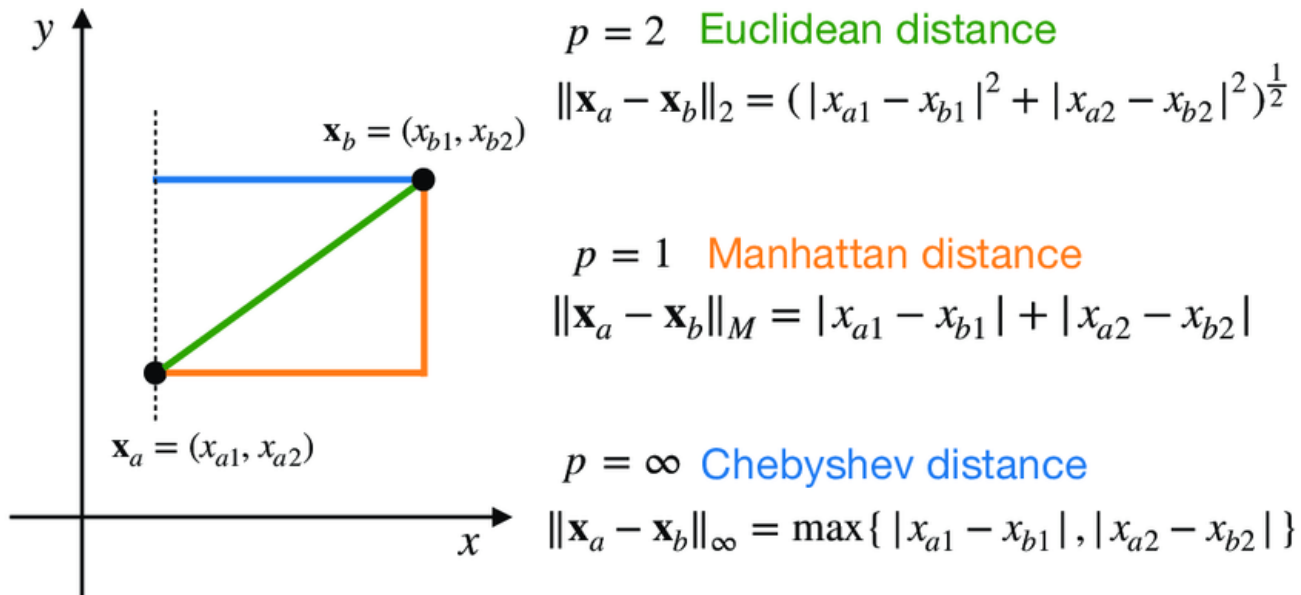


- **Distância de *Minkowski***: é uma métrica definida no ***espaço vetorial normado*** (ou seja, um ***espaço vetorial*** no qual uma ***norma vetorial***, $p(\cdot)$, é definida) que satisfaz algumas propriedades.
 - É um ***espaço onde podemos medir o tamanho ou magnitude dos vetores***.

Métricas de distância

- A **norma vetorial**, $p(\cdot)$, é uma **função que mapeia um vetor em um número real não negativo**, i.e., $\mathbb{R}^{K \times 1} \rightarrow \mathbb{R}_+$, e que possui algumas propriedades.
- Sejam 2 vetores, \mathbf{v} e $\mathbf{u} \in \mathbb{R}^{K \times 1}$, a norma $p(\cdot)$ dos vetores é uma **função com valores não-negativos** com as seguintes propriedades:
 - A norma satisfaz a **desigualdade do triângulo**:
 - $p(\mathbf{v} + \mathbf{u}) \leq p(\mathbf{v}) + p(\mathbf{u})$
 - A norma é **absolutamente escalável**:
 - $p(a\mathbf{v}) = |a|p(\mathbf{v})$, para todo $a \in \mathbb{R}$
 - A norma é **positiva definida**, i.e., os valores são sempre maiores ou no mínimo iguais a zero:
 - Se $p(\mathbf{v}) = 0$, então $\mathbf{v} = \mathbf{0}$ (**vetor nulo**).

Distância de Minkowski

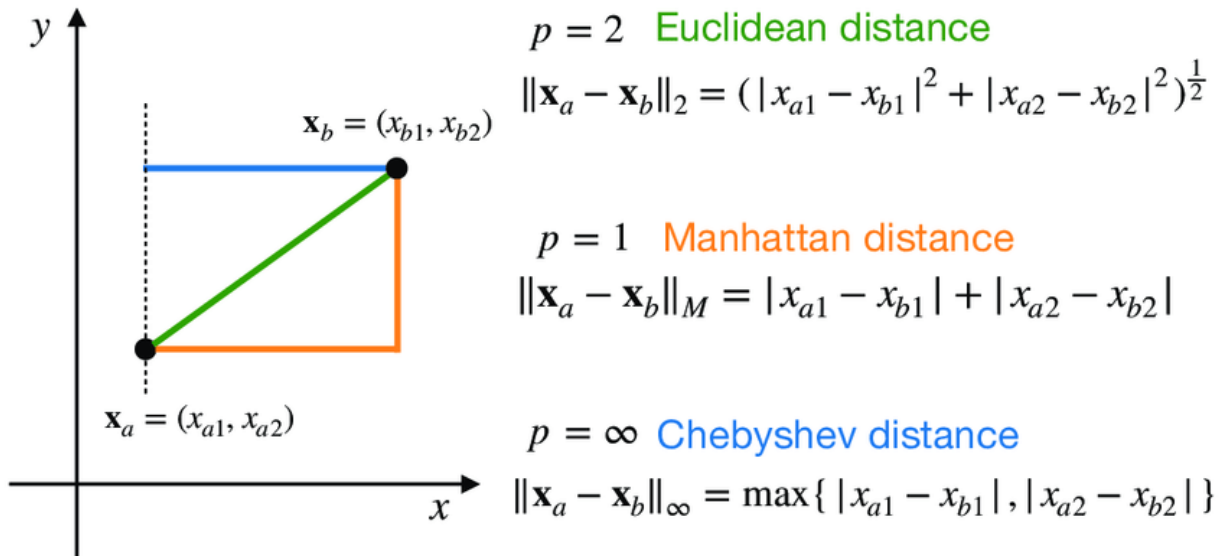


- A **distância de Minkowski** de ordem p é calculada com equação abaixo

$$d(\mathbf{x}; \mathbf{y}) = \left(\sum_{i=1}^K |x_i - y_i|^p \right)^{1/p}.$$

- É uma métrica de **distância generalizada**, ou seja, podemos alterar o parâmetro p para calcular a distância entre dois pontos de formas diferentes.

Distância de Minkowski: Casos particulares



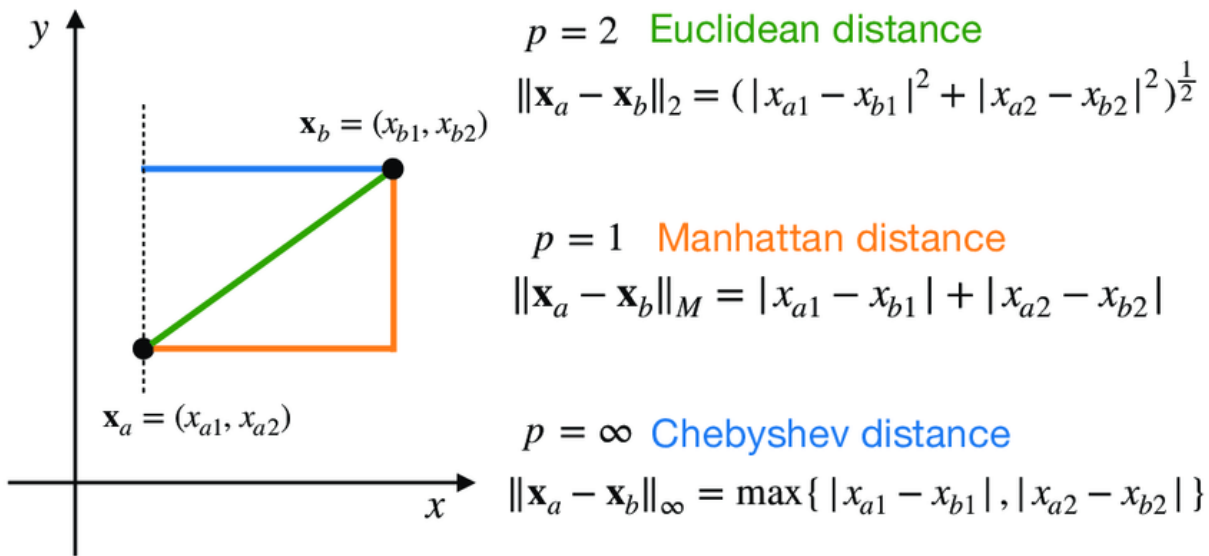
- Para $p = 1$, temos a ***distância de Manhattan***:

$$d(\mathbf{x}; \mathbf{y}) = \sum_{i=1}^K |x_i - y_i|.$$

- Para $p = 2$, temos a ***distância Euclidiana***:

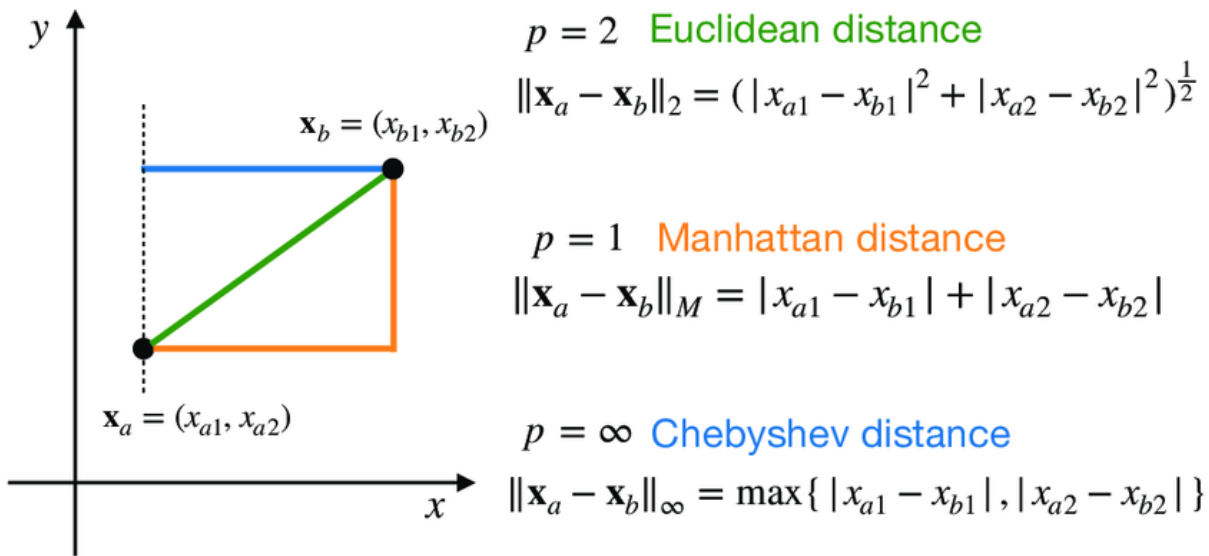
$$d(\mathbf{x}; \mathbf{y}) = \sqrt{\sum_{i=1}^K |x_i - y_i|^2}.$$

Distância Euclidiana



- Mede a **menor distância direta entre dois pontos** em um espaço, como se fosse uma **linha reta**.
- A distância Euclidiana é usada quando os **atributos têm uma relação linear**.

Distância Manhattan



- A distância de Manhattan mede a distância **movendo-se apenas ao longo de eixos horizontais e verticais**, como em uma grade de ruas de uma cidade (daí o nome Manhattan).
- A distância de Manhattan é mais adequada quando os **atributos não têm uma relação linear clara**.
- Usada quando o **movimento só pode ocorrer em direções perpendiculares**, como em redes urbanas ou circuitos.

k-NN para classificação

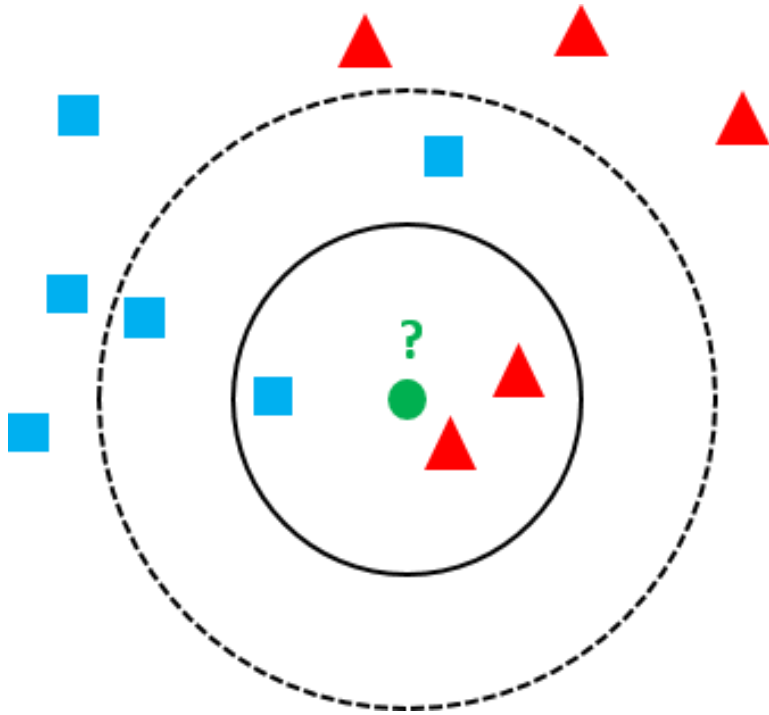
- Com relação ao problema da classificação, a saída da equação

$$\hat{y}(\mathbf{x}') = \underset{\substack{\text{Voto majoritário} \nearrow \\ q \in Q}}{\arg \max} \sum_{\mathbf{x}(i) \in N_k(\mathbf{x}')} \delta(q, y(i)),$$

gerada pelo k-NN equivale a tomar o **voto majoritário** dos k vizinhos mais próximos de \mathbf{x}' , onde

- q é uma das classes do conjunto de classes Q ,
 - $N_k(\mathbf{x}')$ são os k vizinhos mais próximos de \mathbf{x}' , ou seja, os k exemplos de treinamento, $\mathbf{x}(i)$, mais próximos de \mathbf{x}' ,
 - $y(i)$ são as classes correspondentes aos k vizinhos mais próximos de \mathbf{x}' ,
 - $\delta(i, j)$ é o delta de Kronecker onde $\delta(i, j) = 1$ se $i == j$ e 0 caso contrário.
- Em resumo, um novo exemplo de entrada, \mathbf{x}' , é classificado como sendo pertencente à classe que contiver o maior número de vizinhos de \mathbf{x}' .

Exemplo de classificação com k-NN



- Na figura ao lado, o exemplo de teste (**ponto verde**) pode ser classificado como pertencente à classe dos **quadrados azuis** ou à classe dos **triângulos vermelhos** dependendo do valor de k .
- Se $k = 3$ (círculo com linha sólida), ele é atribuído à classe dos triângulos vermelhos pois existem 2 triângulos e apenas 1 quadrado dentro do círculo interno.
- Se $k = 5$ (círculo tracejado), ele é atribuído à classe dos quadrados azuis (3 quadrados vs. 2 triângulos dentro do círculo externo).

Exemplo de classificação com k-NN

- Uma desvantagem da classificação por **votação majoritária** ocorre quando a **distribuição das classes é desbalanceada**.
- Ou seja, **exemplos de uma classe mais frequente tendem a dominar a predição de um exemplo de entrada**, pois tendem a ser comuns entre os k vizinhos mais próximos devido ao seu maior número.
- Portanto, nestas circunstâncias, uma técnica bastante utilizada para se classificar os exemplos de entrada é **atribuir pesos diferentes à contribuição de cada vizinho à decisão final** de tal forma que **vizinhos mais próximos contribuam mais do que vizinhos mais distantes**.
- Uma forma usual é definir os **pesos** como sendo **inversamente proporcionais às distâncias dos vizinhos ao exemplo de entrada, x'** .

Exemplo: Classificação k-NN com SciKit-Learn

Import all necessary libraries.

`import numpy as np`

`import matplotlib.pyplot as plt`

`from matplotlib.colors import ListedColormap`

`from sklearn.neighbors import KNeighborsClassifier`

`from sklearn.datasets import make_blobs`

Importa classe
KNeighborsClassifier.

Number of examples.

`N = 100`

`centers = [[-1, 0], [0, 0]]`

Create a 2-class dataset for classification.

`x, y = make_blobs(n_samples=N, centers=centers, random_state=42)`

Cria duas classes de dados
que se sobrepõem.

Create an instance of Neighbours Classifier and fit the data.

`clf = KNeighborsClassifier(k, weights='distance')`

Train the classifier.

`clf.fit(x, y)`

Predict.

`clf.predict(x)`

Armazena conjunto de
dados na memória.

Realiza predição.

Peso de cada vizinho é o
inverso da distância para o
exemplo de entrada.

Número de vizinhos
mais próximos a
serem considerados.

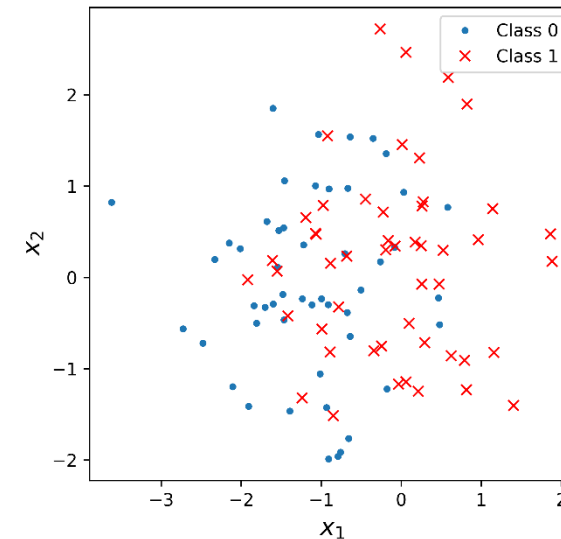
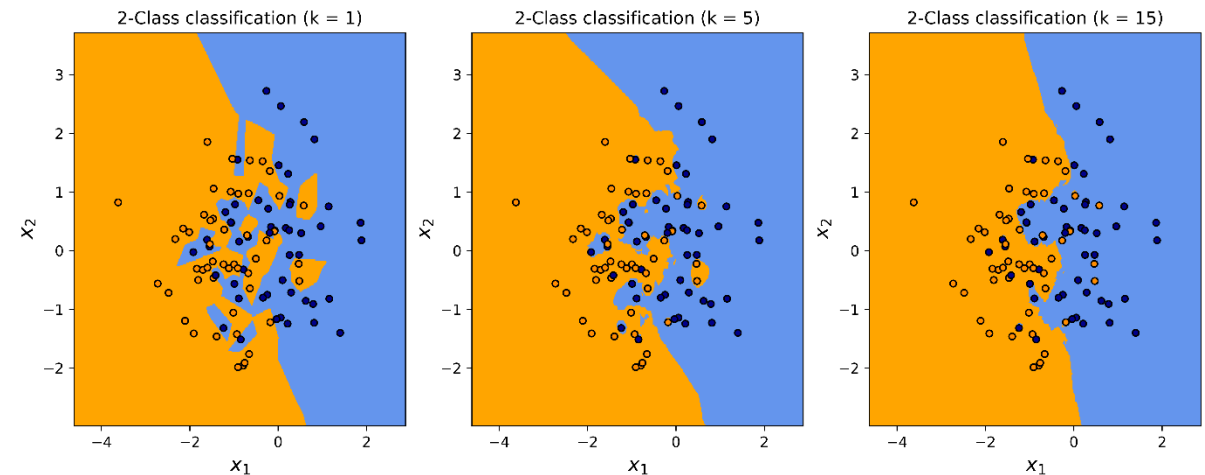


Figura com a
distribuição dos
exemplos de
treinamento.



A figura mostra a fronteira de decisão criada pelo k-NN para diferentes valores de k .

Como podemos ver, à medida que k aumenta, a fronteira tende a ficar mais suave e menos regiões isoladas são criadas para cada classe.

k-NN para regressão

- Seja $N_k(\mathbf{x}')$ o conjunto formado pelos k exemplos de treinamento, $\mathbf{x}(j) \in \mathbb{R}^{K \times 1}, j = 1, \dots, k$, mais próximos ao exemplo de entrada \mathbf{x}' .
- As saídas associadas (i.e., rótulos) a estes exemplos de treinamento são denotadas por $y_j(\mathbf{x} \in N_k(\mathbf{x}')), j = 1, \dots, k$.
- Desta forma, quando utilizado para **regressão**, a saída do algoritmo k-NN para um novo exemplo de entrada, \mathbf{x}' , pode ser escrita de forma geral como

$$\hat{y}(\mathbf{x}') = \frac{\sum_{j=1}^k w_j y_j(\mathbf{x} \in N_k(\mathbf{x}'))}{\sum w_j},$$

onde $w_j, j = 1, \dots, k$ representa o peso associado ao j -ésimo vizinho de \mathbf{x}' .

- Os pesos associados aos vizinhos podem ser **uniformes** ou **inversamente proporcionais à distância**.

Exemplo: Regressão k-NN com SciKit-Learn

```
# Import all necessary libraries.
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsRegressor

# Generate sample data.
N = 40
np.random.seed(42)
X = np.sort((6*np.random.rand(N, 1) - 3), axis=0)
T = np.linspace(-3, 3, 100)[:, np.newaxis]
y = (0.5*X**2 + X + 2).ravel()
y_orig = (0.5*X**2 + X + 2).ravel()

y += np.random.randn(N)

# Fit regression model
n_neighbors = 5

plt.figure(figsize=(15,5))
for i, weights in enumerate(['uniform', 'distance']):
    knn = KNeighborsRegressor(n_neighbors, weights=weights)
    y_ = knn.fit(X, y).predict(T)

    plt.subplot(1, 2, i + 1)
    plt.scatter(X, y, color='darkorange', label='noisy data')
    plt.plot(X, y_orig, color='red', label='original data')
    plt.plot(T, y_, color='navy', label='prediction')
    plt.axis('tight')
    plt.legend()
    plt.title("KNeighborsRegressor (k = %i, weights = '%s')" % (n_neighbors, weights))

plt.show()
```

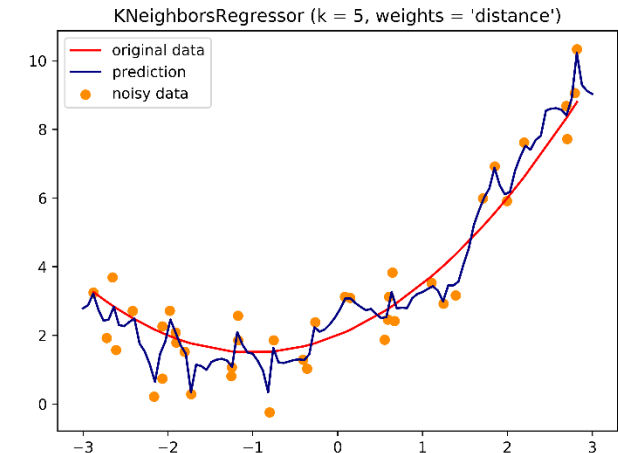
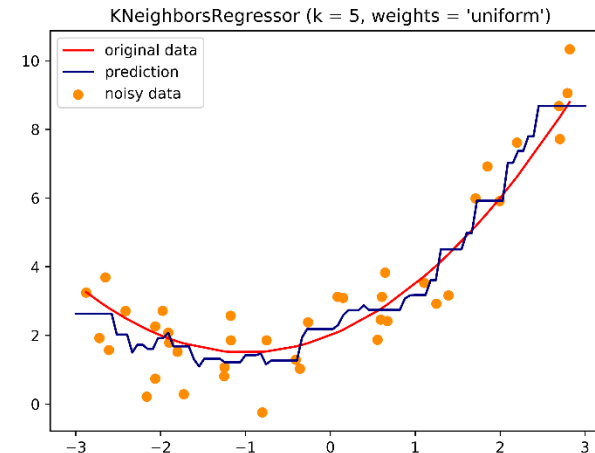
Importa classe KNeighborsRegressor.

Cria dados para regressão.

Peso de cada vizinho é uniforme ou o inverso da distância para o exemplo de entrada.

Armazena conjunto de dados na memória e realiza predição.

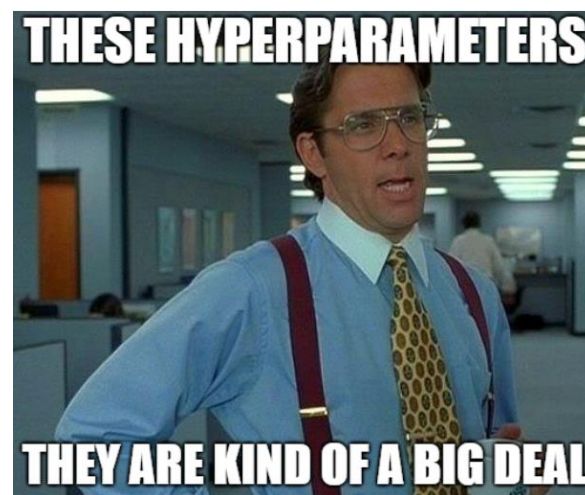
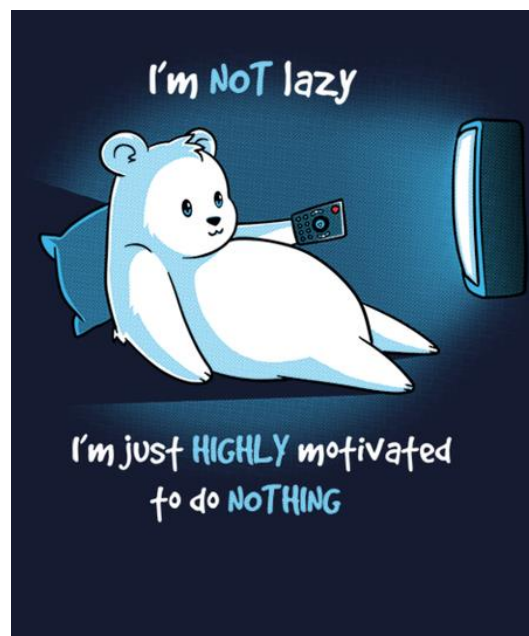
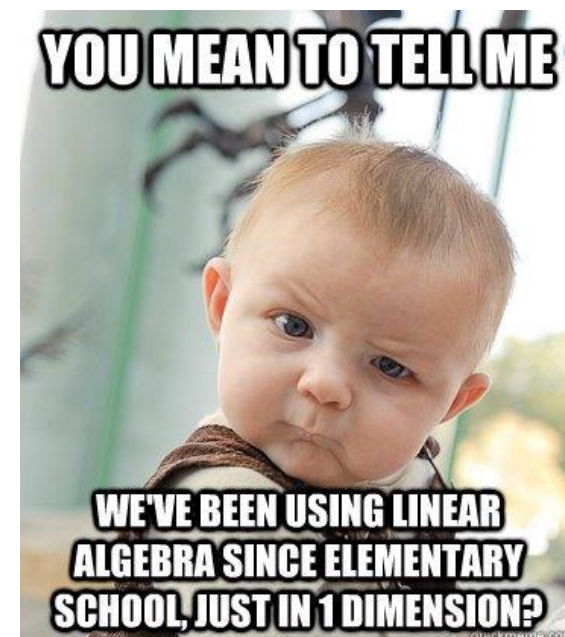
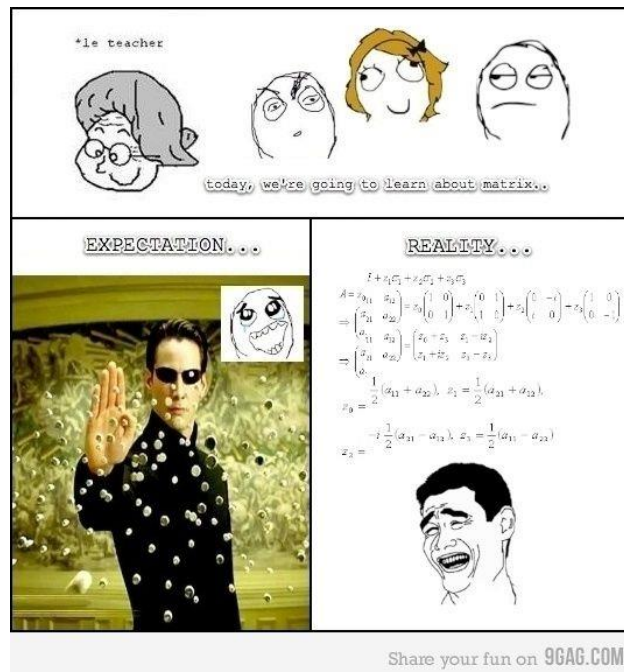
- A figura abaixo compara a regressão feita com o algoritmo k-NN quando os pesos associados aos vizinhos são **uniformes** (figura da esquerda) e **inversamente proporcionais à distância** (figura da direita).
- Pesos uniformes resultam em uma aproximação mais suave, pois o valor de saída será a média dos k valores, porém, com pesos inversamente proporcionais à distância, amostras próximas ao exemplo de entrada terão grande influência no valor de saída, fazendo com que ele seja bem próximo desse valor.



Tarefas e Avisos

- Vocês já podem fazer a lista #6.

Obrigado!



Figuras

