# K-Vizinhos Próximos

## Sobre aprendizado de máquina

Como já falamos durante o curso, **aprendizado de máquina** é o conjunto de modelos baseados em dados estatísticos ou probabilísticos que tomam conclusões referentes a dados baseados em outros dados já conhecidos.

Temos principalmente dois tipos de modelos:

- Supervisionados, nos quais já possuo dados sore o meu alvo, e estou tentando obter uma função que associe esses dados ("podemos encontrar grupos de clientes com maior probabilidade de cancelarem seus serviços logo após o vencimento dos contratos?");
- Não supervisionados, que não possuem um objetivo específico, ou alvo ("nossos clientes naturalmente se encaixam em grupos diferentes?").

Durante o restante do curso, vamos falar um pouco sobre alguns modelos de aprendizado de máquina. Não é nossa pretensão abordar todos os modelos, ou abordar uma base teórica muito forte. Nas aulas, iremos discutir um pouco da teoria por trás, mas nosso foco será muito maior no desenvolvimento de aplicações práticas que utilizem esses modelos.

Para a aplicação dos modelos, além das bibliotecas `pandas`, `numpy`, `matplotlib` e `seaborn`, que já utilizamos em aulas anteriores, vamos usar também a biblioteca `scikit-learn`, que disponibiliza uma série de modelos de aprendizado de máquina, bem como diversas métricas estatísticas para avaliarmos o desempenho dos modelos escolhidos. A documentação oficial contém mais informações sobre a biblioteca.

Para instalar o `scikit-learn`, basta entrar no terminal e digitar o seguinte comando:

```
pip install sklearn
```

## Modelo dos k-vizinhos

Vamos pensar no seguinte cenário: você quer tentar prever como uma pessoa vai votar na próxima eleição, entre os candidatos A ou B. Você não sabe nada sobre a pessoa, exceto o bairro em que a pessoa mora. Uma possível abordagem lógica é: se os demais moradores do bairro vão votar no candidato A, faz sentido que aquela pessoa também vote em A.

Agora, imagine que você sabe mais sobre a pessoa do que apenas o endereço - idade, rendimentos, composição da família, religião, etc. Considerando que o comportamento dessa pessoa é influenciado ou caracterizado por essas características, considerar apenas os

vizinhos com essas características pode ser uma classificação melhor do que considerar todos os vizinhos. Essa é a ideia por trás da classificação dos vizinhos mais próximos.

Esse modelo é um dos modelos preditivos mais simples que existe. Ele não possui premissas matemáticas, e requer apenas uma noção de distância, e a premissa de que pontos que estão pertos uns dos outros são similares.

O modelo não segue a lógica de outros modelos preditivos que vamos ver, já que normalmente eles consideram todo o conjunto de pontos para aprender um padrão nos dados. Os modelos de vizinhos mais próximos, por outro lado, só consideram um conjunto pequeno de elementos.

Para rodar o modelo, é necessário ter um conjunto de pontos (em qualquer dimensão), e cada ponto deve estar associado a um rótulo. Por exemplo, uma empresa de TV por assinatura pode tentar prever, baseado na idade do cliente, a opção pela contratação ou não de um pacote específico de canais.

| Idade | Resposta |
|---|---|
| 20 | S |
| 50 | N |
| 36 | S |
| 70 | S |
| 46 | N |
| 25 | N |

Ao fazer uma oferta para um cliente com, por exemplo, 40 anos, o modelo pode buscar um subconjunto de k-vizinhos próximos, onde `k` é um número escolhido pelo analista.

Resumindo, o modelo de k-vizinhos utiliza as seguintes etapas:

- Calcula a distância entre o novo dado e os demais dados do treinamento;
- Escolhe `k` entradas de dados mais próximas ao novo dado;
- Pega o rótulo mais votado entre aqueles vizinhos.

Para determinar a distância entre dois elementos, utiliza-se a distância euclidiana. Dado um ponto O em um sistema de duas dimensões ($x_o$, $y_o$), a distância euclidiana de um ponto (x, y) ao ponto O é dada por:

$$d = \sqrt{(x - x_o)^2 + (y - y_o)^2}$$

A fórmula deve ser alterada conforme a dimensão do sistema que estamos determinando. No caso do nosso exemplo anterior, consideremos a idade do nosso novo cliente, 40 anos. Considerando que o sistema possui apenas uma dimensão (idade), podemos calcular a distância euclidiana simplesmente calculando o valor absoluto da subtração de 40 em cada idade:

| Entrada | Idade | Resposta | Distância |
|---|---|---|---|
| 1 | 20 | S | 20 |
| 2 | 50 | N | 10 |
| 3 | 36 | S | 4 |
| 4 | 70 | S | 30 |
| 5 | 46 | N | 6 |
| 6 | 25 | N | 15 |

Com as distâncias calculadas, basta identificar as `k` entradas com os menores valores. Usualmente, adota-se um número ímpar de vizinhos, de forma a eliminar a possibilidade de um empate nos resultados. Portanto, caso adotemos 3 vizinhos, buscaríamos os rótulos das entradas 2, 3 e 5, o que nos levaria a concluir que a provável resposta do nosso cliente à opção de contratar um pacote adicional de canais seja "não".

## Aplicação do modelo

Como falamos, vamos usar a biblioteca `scikit-learn` para elaborar nossos modelos de aprendizado de máquina. Ela fornece, além de diversas ferramentas para apoiar a mineração de dados, alguns conjuntos de exemplos para aplicação. Nesse modelo, vamos usar um conjunto de dados sobre a flor íris. Esse conjunto possui 50 amostras de três espécies diferentes de íris, um total de 150 amostras. Para cada amostra, o conjunto armazena o comprimento e a largura da sépala (uma parte da flor), o comprimento e a largura da pétala e o nome da espécie.

Então, tendo as quatro dimensões de uma nova amostra, podemos usar o modelo dos k-vizinhos para prever qual seria a sua espécie. Esse é um problema de classificação, já que a resposta é categórica (é apenas uma dentre três espécies possíveis).

À medida que formos desenvolvendo os modelos, vamos ver que uma análise de aprendizado de máquina envolve alguns passos muito bem definidos:

- Importação das bibliotecas e análise preliminar dos dados
- Tratamento e organização dos dados
- Divisão dos dados e treino do modelo
- Análise dos resultados e conclusões

Vamos abordar três desses quatro passos nesta aula. O passo 2, de tratamento e organização dos dados, não será necessário, já que utilizaremos uma base de dados já formatada e fornecida pela `scikit-learn`.

## Passo 1: import e análise preliminar dos dados

Para essa análise, vamos importar a `matplotlib` para criar alguns gráficos ao longo do exemplo. Também vamos importar a classe `load_iris` da `scikit-learn`. Em seguida, vamos criar um objeto `iris`, a partir da classe `load_iris`.

```
%matplotlib inline
import matplotlib.pyplot as plt

from sklearn.datasets import load_iris

iris = load_iris()
type(iris)
```

```
sklearn.utils.Bunch
```

O tipo do objeto `iris` é um "bunch". Esse é um tipo específico da `scikit-learn` para armazenar conjuntos de dados e seus atributos. Vamos ver agora quais tipos de dados esse objeto `iris` contém.

```
iris.data
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.2],
       [5. , 3.2, 1.2, 0.2],
       [5.5, 3.5, 1.3, 0.2],
       [4.9, 3.6, 1.4, 0.1],
       [4.4, 3. , 1.3, 0.2],
       [5.1, 3.4, 1.5, 0.2],
       [5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
```

```
       [4.4, 3.2, 1.3, 0.2],
       [5. , 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4],
       [4.8, 3. , 1.4, 0.3],
       [5.1, 3.8, 1.6, 0.2],
       [4.6, 3.2, 1.4, 0.2],
       [5.3, 3.7, 1.5, 0.2],
       [5. , 3.3, 1.4, 0.2],
       [7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
       [6.9, 3.1, 4.9, 1.5],
       [5.5, 2.3, 4. , 1.3],
       [6.5, 2.8, 4.6, 1.5],
       [5.7, 2.8, 4.5, 1.3],
       [6.3, 3.3, 4.7, 1.6],
       [4.9, 2.4, 3.3, 1. ],
       [6.6, 2.9, 4.6, 1.3],
```

Cada observação representa uma flor e 4 colunas representam 4 medidas. Nós conseguimos ver as medidas dentro do atributo `data`, e os títulos dessas medidas dentro de `feature_names`. Já para os rótulos podemos usar os atributos `target` e `target_names`, para os valores e o título. Como podemos ver abaixo, os rótulos são categorizados como 0, 1 e 2, para que esses dados possam ser usados, por exemplo, pela numpy.

```
# Nomes das medidas
iris.feature_names
```

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

```
# Inteiros representando as espécies:
# - 0: setosa
# - 1: versicolor
# - 2: virginica
iris.target
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
# Rótulos do alvo
iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

```
# Temos 150 observações e 4 medidas
iris.data.shape
```

```
(150, 4)
```

Para visualizar melhor, vamos pegar os dados das sépalas e plotar num gráfico, para entender a distribuição pelas espécies. Para isso, vamos criar um gráfico de dispersão usando a `matplotlib`.

```
# Dividir o array de dados para pegar apenas as sépalas
sepal_info = iris.data[:, :2]
sepal_info
```

```
[6.1, 3. ],
[5.8, 2.6],
[5. , 2.3],
[5.6, 2.7],
[5.7, 3. ],
[5.7, 2.9],
[6.2, 2.9],
[5.1, 2.5],
[5.7, 2.8],
[6.3, 3.3],
[5.8, 2.7],
[7.1, 3. ],
[6.3, 2.9],
[6.5, 3. ],
[7.6, 3. ],
[4.9, 2.5],
[7.3, 2.9],
[6.7, 2.5],
[7.2, 3.6],
[6.5, 3.2],
[6.4, 2.7],
[6.8, 3. ],
[5.7, 2.5],
[5.8, 2.8],
[6.4, 3.2],
[6.5, 3. ],
[7.7, 3.8],
[7.7, 2.6],
[6. , 2.2],
[6.9, 3.2],
[5.6, 2.8],
[7.7, 2.8],
[6.3, 2.7],
[6.7, 3.3],
[7.2, 3.2],
[6.2, 2.8],
[6.1, 3. ],
[6.4, 2.8],
[7.2, 3. ],
[7.4, 2.8],
[7.9, 3.8],
[6.4, 2.8],
[6.3, 2.8],
[6.1, 2.6],
[7.7, 3. ],
[6.3, 3.4],
[6.4, 3.1],
[6. , 3. ],
[6.9, 3.1],
[6.7, 3.1],
```

```
[6.9, 3.1],
[5.8, 2.7],
[6.8, 3.2],
[6.7, 3.3],
[6.7, 3. ],
[6.3, 2.5],
[6.5, 3. ],
[6.2, 3.4],
[5.9, 3. ]])
```

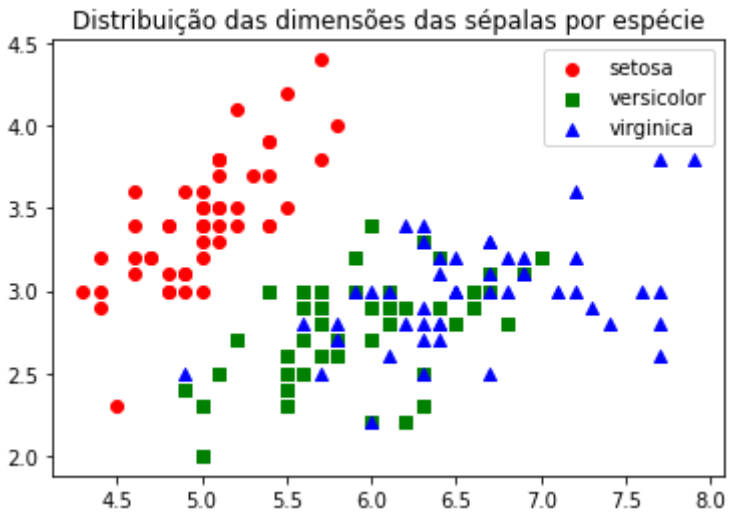
```
# Montar um dicionário contendo os dados
plots = {0: ([], []), 1: ([], []), 2: ([], [])}

# Queremos que cada linguagem tenha marcador e cor diferentes
markers = {0: "o", 1: "s", 2: "^"}
colors = {0: "r", 1: "g", 2: "b"}

for (sepal_length, sepal_width), species in zip(sepal_info, iris.target):
  plots[species][0].append(sepal_length)
  plots[species][1].append(sepal_width)

# Cria uma série de dispersão para cada espécie
for species, (x, y) in plots.items():
  plt.scatter(
    x,
    y,
    color=colors[species],
    marker=markers[species],
    label=iris.target_names[species]
  )

plt.legend(loc=0)  # A matplotlib vai escolher o local da legenda
plt.title("Distribuição das dimensões das sépalas por espécie")
plt.show()
```




## Passo 2: tratamento e organização dos dados

Como falamos anteriormente, esse passo não será necessário para esse exemplo em particular. Nas aulas seguintes usaremos exemplos nos quais precisaremos criar um processo de transformação dos nossos dados extraídos.

## Passo 3: divisão dos dados e treino do modelo

Já falamos que treinar e testar em cima dos mesmos dados não é a melhor abordagem, portanto vamos dividir os dados em duas partes, `training_set` e `testing_set`. Vamos usar a função `train_set_split` para dividir os dados. Essa função possui um parâmetro opcional `test_size` para definir o percentual da divisão. O argumento `random_state` faz com que os dados sejam divididos da mesma forma todas as vezes que você rodar o código. O inteiro passado é um gerador que é definido pelo usuário (qualquer valor inteiro).

```
# Dividir os dados em conjuntos de treino e de teste (80:20)
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(
  iris.data,
  iris.target,
  test_size=0.2,
  random_state=4
)
```

```
# Shape dos objetos
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(120, 4)
(30, 4)
(120,)
(30,)
```

O pacote scikit-learn é organizado em módulos, para que possamos importar as classes relevantes de forma mais fácil. Vamos importar a classe `KNeighborsClassifier` do módulo `neighbors` e instanciar o estimador (termo usado pela biblioteca para fazer referência ao modelo).

Vamos criar uma instância `knn` da classe `KNeighborsClassifier`, que saberá como classificar pelos k-vizinhos quando fornecermos os dados. O parâmetro `n_neighbors` é o parâmetro de refinamento/ajuste (k). Todos os outros argumentos vamos manter como os valores-padrão, porém é possível consultar a documentação para mais informações.

O método `fit` é usado para treinar o modelo nos dados de treinamento (`x_train, y_train`), enquanto que o método `predict` é usado para o teste dos dados (`x_test`). Escolher o valor ótimo de k é crítico, portanto vamos rodar o método `fit` para diferentes valores de k (de 1 a 25) usando um loop, e vamos gravar a acurácia do modelo em uma variável `scores`.

```
# Importar a KNeighborsClassifier do sklearn
from sklearn.neighbors import KNeighborsClassifier

# Importa o modelo de métricas para verificar a acurácia
```

```
from sklearn import metrics


# Roda k de 1 a 25 e grava a acurácia dos testes
k_range = range(1, 26)
scores = {}
scores_list = []

for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train, y_train)
    y_pred = knn.predict(x_test)
    scores[k] = metrics.accuracy_score(y_test, y_pred)
    scores_list.append(scores[k])
```

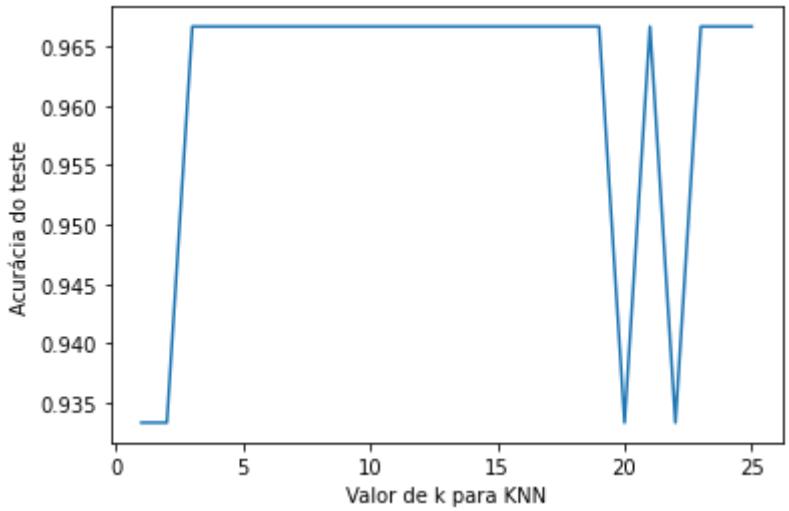
## Passo 4: análise dos resultados

Vamos agora plotar a relação entre os valores de k e a acurácia correspondente usando a matplotlib. Como podemos ver abaixo , existem altas e baixas na acurácia e isso é usual quando examinamos a complexidade do modelo pela acurácia. Normalmente, à medida que o valor de k aumenta, há uma tendência de alta da acurácia até certo ponto, e a partir deste há novamente uma queda.

No gráfico conseguimos ver que para valores de k entre 3 e 19 temos um modelo com 96,5% de acurácia.

```
# Plota a relação entre k e a acurácia do teste
plt.plot(k_range, scores_list)
plt.xlabel("Valor de k para KNN")
plt.ylabel("Acurácia do teste")
```

```
Text(0, 0.5, 'Acurácia do teste')
```




Para o nosso modelo final podemos adotar um valor entre 3 e 19, como k = 5, por exemplo, e retreinamos o modelo com todos os dados disponíveis. Esse será o nosso modelo final, no qual estamos prontos para fazer predições.

```
knn = KNeighborsClassifier(n_neighbors=5)
```

```
knn.fit(iris.data, iris.target)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

```
labels = {0: "setosa", 1: "versicolor", 2: "virginica"}

# Vamos montar a previsão em cima de dados novos
# Prevê os rótulos das duas observações aleatórias abaixo
x_new = [
    [3, 4, 5, 2],
    [5, 4, 2, 2]
]
y_predict = knn.predict(x_new)

print(labels[y_predict[0]])
print(labels[y_predict[1]])
```

```
versicolor
setosa
```

Com a nossa análise, agora podemos utilizar o objeto `knn` para prever quaisquer dados que estejam no mesmo formato dos nossos dados de referência.