2.2 K-vizinhos mais próximos

Em problemas de classificação, os objetos de natureza semelhante classificam-se num único grupo. Quando é introduzido um novo objeto, este irá ser colocado (classificado) num grupo apropriado.

Seguindo os princípios já usados na regressão, treina-se a máquina para classificar um objeto com base numa determinada característica. Quando a máquina aprender como são formados os grupos, será capaz de classificar qualquer novo objeto desconhecido corretamente.

Mais uma vez, usam-se os dados de teste para verificar se a máquina aprendeu a técnica de classificação antes de colocar o modelo desenvolvido em "produção", **"predictor"**.

Os K-vizinhos mais próximos, simplesmente chamados de kNN, são uma técnica estatística que pode ser usada para resolver problemas de classificação e regressão.

Discute-se a seguir o caso de classificação de um objeto desconhecido usando kNN.

Considera-se a distribuição de objetos conforme mostrado na imagem. Fonte - Tutorials

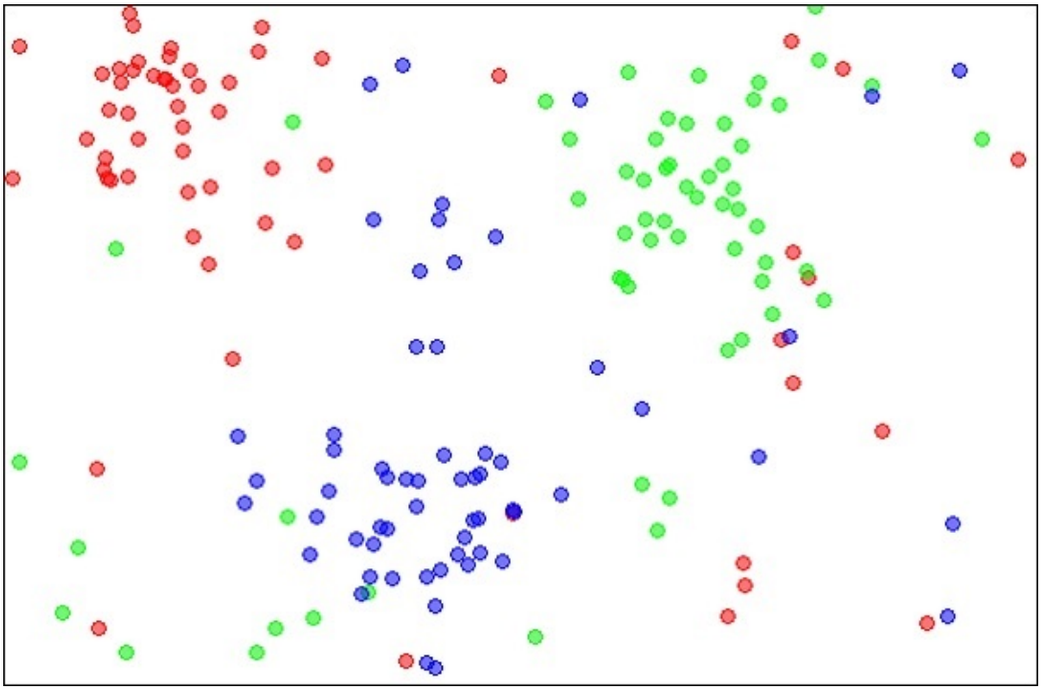

Figura 2.1: Classificação: k-vizinhos mais próximos

O diagrama mostra três tipos de objetos, marcados com cor vermelha, azul e verde. Quando se executa o classificador kNN no conjunto de dados acima, os limites para cada tipo de objeto serão marcados como mostrado

na figura em baixo.

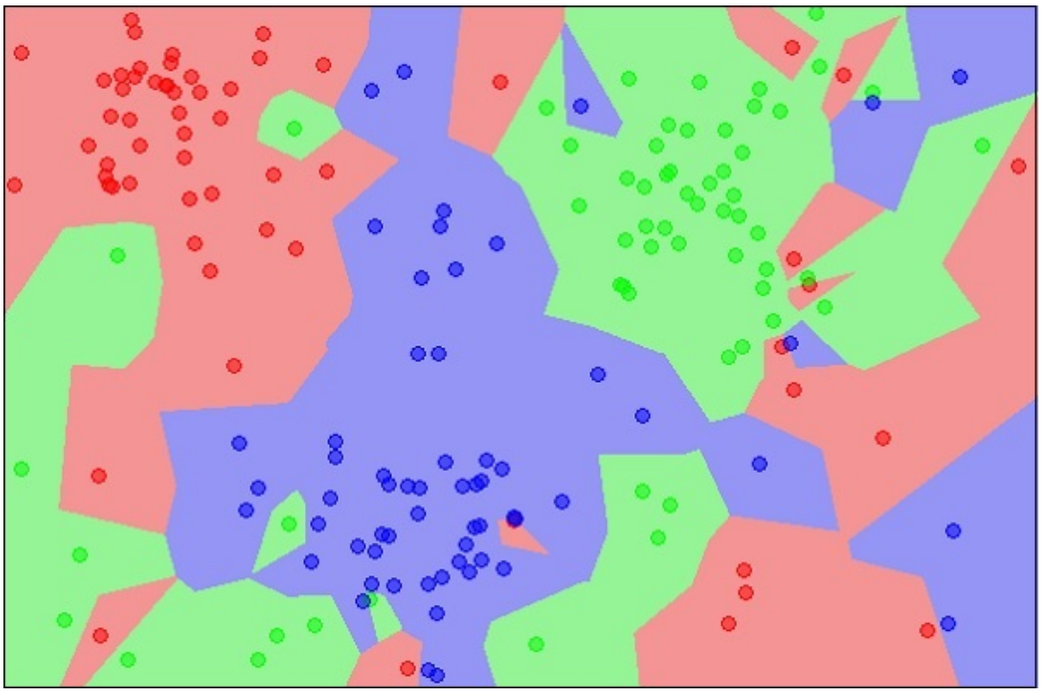

Figura 2.2: Limites das categorias de dados

Considera-se agora um novo objeto desconhecido que se deseja classificar como vermelho, verde ou azul (ilustração da Figura 2.3).

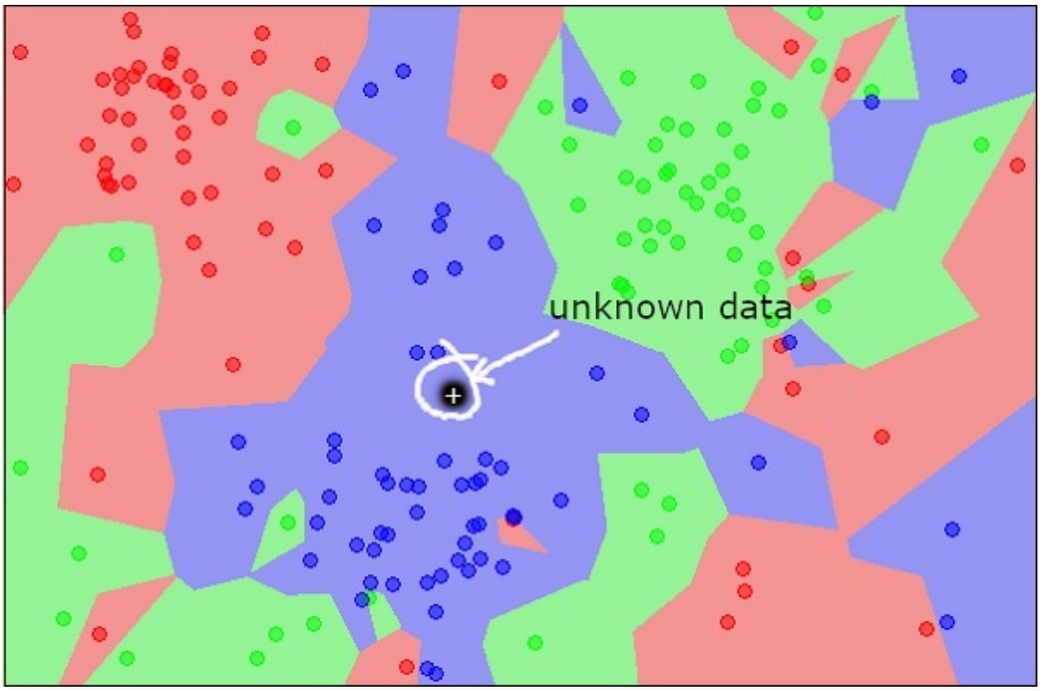

Figura 2.3: Classificação de objeto desconhecido

Como é visível, o ponto de dados desconhecido pertence a uma classe de objetos azuis. Matematicamente, isso pode ser concluído medindo a distância desse ponto desconhecido com todos os outros pontos no conjunto de dados. Ao fazer isso, sabe-se que a maioria de seus vizinhos é azul. A distância média para objetos vermelhos e verdes seria definitivamente maior do que a distância média aos objetos azuis. Desta forma, este objeto desconhecido será classificado como pertencente à classe azul.

Os algoritmos KNN também podem ser usados para problemas de regressão, "KNeighborsRegressor", ou em muitas outras aplicações (neighbors.KNeighborsTransformer, neighbors.RadiusNeighborsRegressor, neighbors.NearestCentroid, neighbors.NearestNeighbors, neighbors.NeighborhoodComponentsAnalysis). O algoritmo *KNeighborsClassifier* está disponível para utilização na maioria das bibliotecas de ML.

KNeighborsClassifier

from sklearn import neighbors	
KNeighborsClassifier( ) . métodos	
fit(X, y[, sample_weight])	Ajusta o classificador de k vizinhos mais próximos do conjunto de dados de treino
get_params([deep])	Indica os parâmetros deste estimador.
kneighbors([X, n_neighbors, return_distance])	Encontra os K-vizinhos de um ponto.
kneighbors_graph([X, n_neighbors, mode])	Calcula o grafo[1] (ponderado) de k-Vizinhos para pontos em X.
predict(X)	Prevê os rótulos de classe para os dados fornecidos.
predict_proba(X)	Devolve estimativas de probabilidade para os dados de teste X.
score(X, y[, sample_weight])	Devolve a precisão média nos dados e rótulos de teste fornecidos.
set_params(**params)	Define os parâmetros deste estimador ("predictor").

Apresenta-se a seguir um possível código Python para a solução de um problema de classificação de um objeto desconhecido usando `KNeighborsClassifier( )`, neste caso relativo ao conjunto de dados `Iris`, que representa 3 tipos de flores *Iris* (*Iris setosa, virginica e versicolor*) com 4 atributos (comprimento da sépala, largura da sépala, comprimento da pétala e largura da pétala), semelhante ao descrito na Figura 2.1.

Antes de mais os pacotes necessários para correr o algoritmo.

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets
```

Para além dos pacotes (livrarias) já referidos anteriormente, é carregado o *"seaborn"*, uma ferramenta de visualização de dados Python baseada em matplotlib, que permite representar dados estatísticos de uma forma esteticamente atraente, mantendo o rigor informativo.
Ainda assim, para ser possível a visualização dos dados é necessário reduzir a dimensão do vetor input `iris.data` — neste caso optou-se por considerar, de forma perfeitamente aleatória, as duas primeiras colunas, `X=iris.data[:, :2]`. No parágrafo 2.2.1 é definido um método de redução da dimensão do vetor input mais criterioso, através da ferramenta `NeighborhoodComponentsAnalysis`.

De seguida definem-se os dados de treino e mostra-se a sua distribuição no plano *xoy* (*sns.scatterplot*, Figura 2.4) com as cores que irão ser usadas de acordo com a respetiva classificação "cmap_bold".

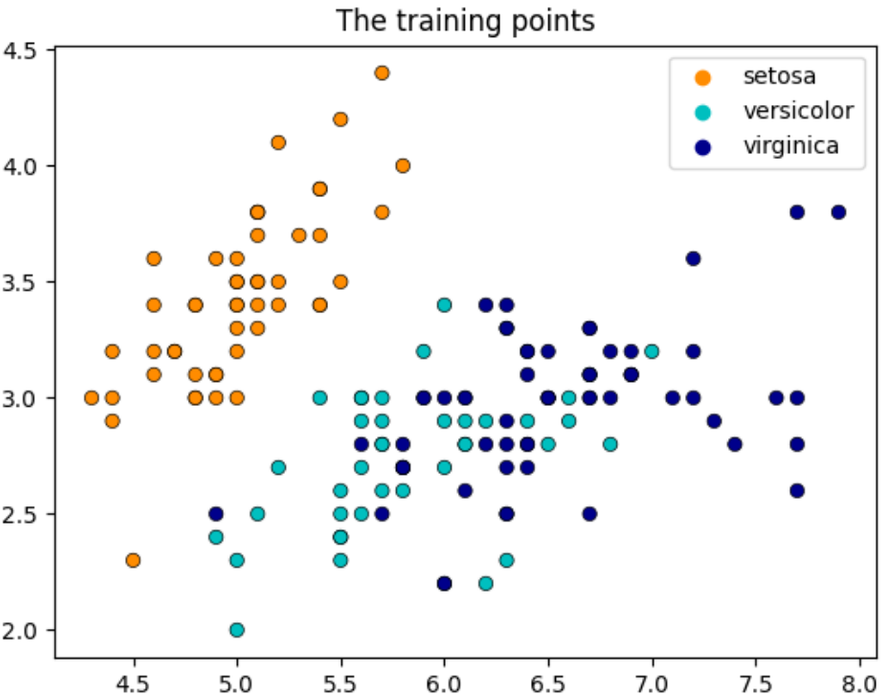

Figura 2.4: Classificação dos dados de treino

São também introduzidos desde já alguns parâmetros:

- O parâmetro $k = n\_neighbors$ que implementa a aprendizagem KNeighborsClassifier com base nos k vizinhos mais próximos de cada ponto de consulta, onde k é um valor inteiro.

```
n_neighbors = 15
# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2]
y = iris.target
h = 0.02  # step size in the mesh
```

A escolha ideal do valor de k é altamente dependente dos dados em questão: em geral, um maior valor de k suaviza os limites da classificação, mas também torna essa fronteira menos precisa em relação aos dados (Figura 2.5).

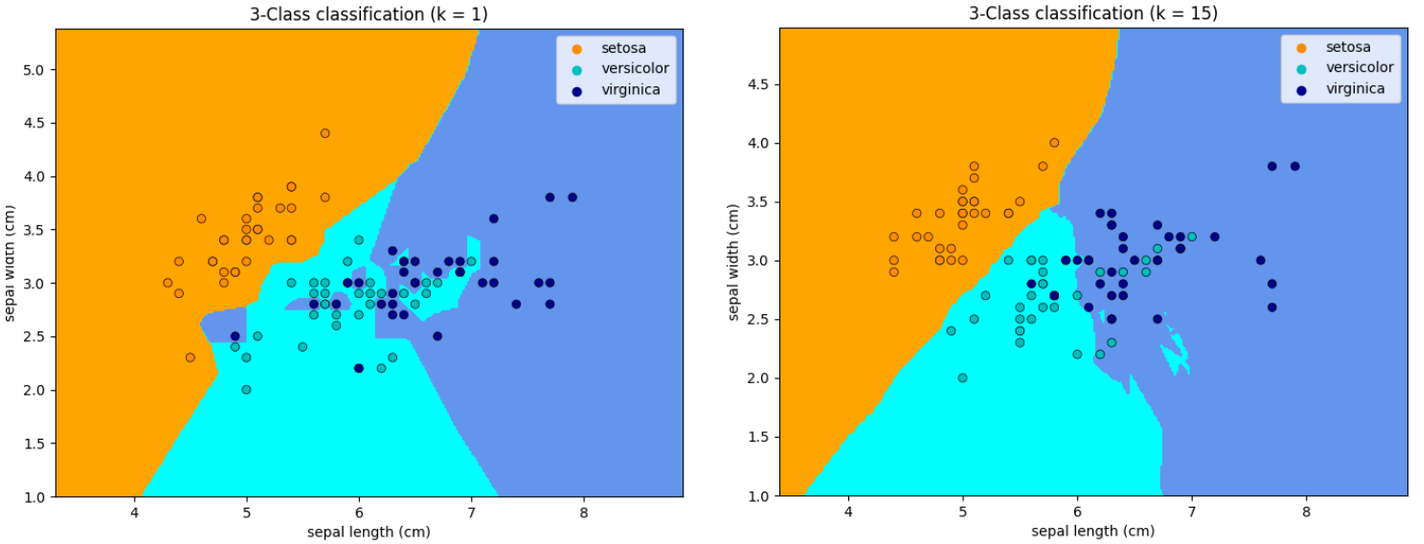

Figura 2.5: Fronteira da classificação para diferentes valores de "k_neighbors"

- O espaçamento h de uma malha a ser construída;
- As cores que irão ser usadas na malha para distinguir a classificação obtida "cmap_light".

```
# Create color maps
cmap_light = ListedColormap(["orange", "cyan", "cornflowerblue"])
cmap_bold = ["darkorange", "c", "darkblue"]
```

Para se poder visualizar a classificação obtida cria-se uma malha (*meshgrid*) dentro do intervalo $[x\_min, x\_max] \times [y\_min, y\_max]$ e com um determinado espaçamento h.
Feita a classificação de cada ponto dessa malha através de,

$$Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()]),$$

esta será associada a uma determinada cor "cmap_light" com vista à sua visualização $plt.contourf(xx, yy, Z, cmap = cmap\_light)$.
Na realidade são feitas duas classificações em simultâneo,

$$clf = neighbors.KNeighborsClassifier(n\_neighbors, weights = weights),$$

onde,

$$weights\ in\ [\text{"}uniform\text{"}, \text{"}distance\text{"}].$$

```
for weights in ["uniform", "distance"]:
    clf = neighbors.KNeighborsClassifier(n_neighbors, weights=weights)
    clf.fit(X, y)
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.figure(figsize=(8, 6))
    plt.contourf(xx, yy, Z, cmap=cmap_light)
```

São acrescentados os pontos de treino e mostra-se a sua distribuição (*sns.scatterplot*) dentro da malha Z.

```
    # Plot also the training points
    sns.scatterplot(
        x=X[:, 0],
        y=X[:, 1],
        hue=iris.target_names[y],
        palette=cmap_bold,
        alpha=1.0,
        edgecolor="black",)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title(
        "3-Class classification (k = %i, weights = '%s')" % (n_neighbors, weights))
    plt.xlabel(iris.feature_names[0])
    plt.ylabel(iris.feature_names[1])
```

Por fim, é acrescentada a previsão da cor (a classificação) onde se encaixa um objeto desconhecido (x,y), Figura 2.6.

Neste caso considerou-se o ponto médio de dois dados de treino $(X[1,:]+X[43,:])/2$, isto é,

$$x = (X[1,0]+X[43,0])/2,\ y = (X[1,1]+X[43,1])/2.$$

```
    # Plot a predict point
    sns.scatterplot(
        x=(X[1,0]+X[43,0])/2,
        y=(X[1,1]+X[43,1])/2,
        marker="X",
        s=90,
        hue=iris.target_names[y],
        palette=cmap_bold,
        alpha=1.0,
        edgecolor="w",)
plt.show()
```

A Figura 2.6 apresenta a classificação e localização, através do simbolo x, do objeto desconhecido (x, y).

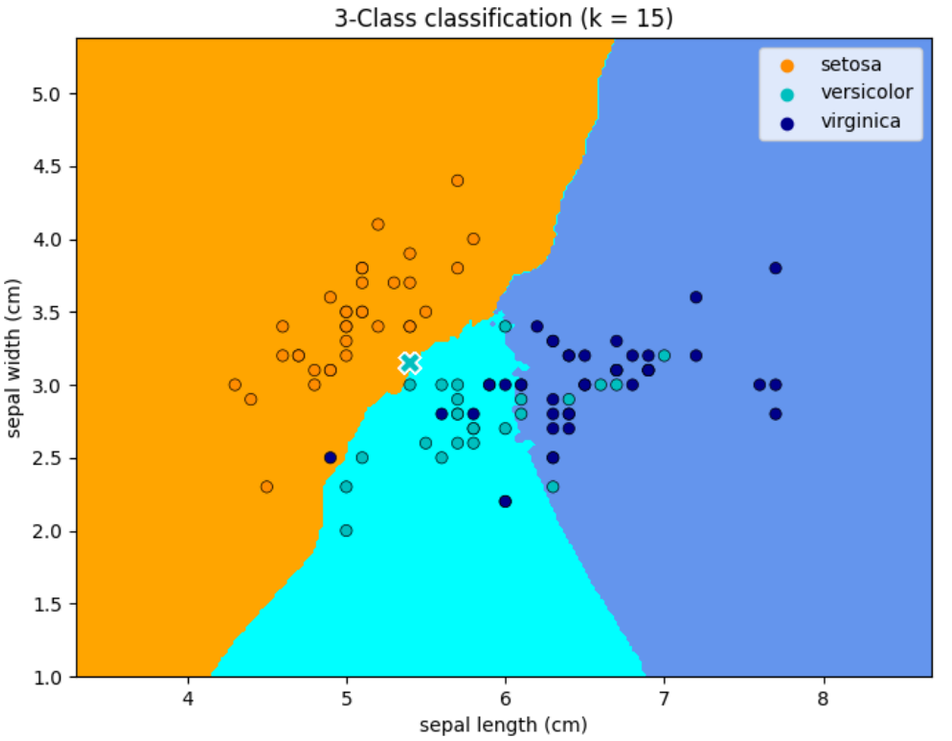

Figura 2.6: Classificação de objeto desconhecido

Laboratório 2