Anotações sobre k-NN

Raphael Valdetaro

November 11, 2023

1 Introdução

O *k-NN* (k-Nearest Neighbors) é um método de aprendizado supervisionado versátil. Ele se destaca em tarefas de classificação, regressão e detecção de outliers. Este método é não paramétrico, o que significa que não faz suposições explícitas sobre a forma funcional dos dados, tornando-o especialmente útil em situações com limites de decisão complexos e não lineares.

O k-NN opera na ideia de proximidade: pontos de dados semelhantes tendem a ter rótulos semelhantes. A escolha do valor de "k" influencia a suavidade da decisão e a sensibilidade ao ruído. Valores pequenos podem levar a modelos mais sensíveis a outliers, enquanto valores grandes podem suavizar demais as fronteiras de decisão.

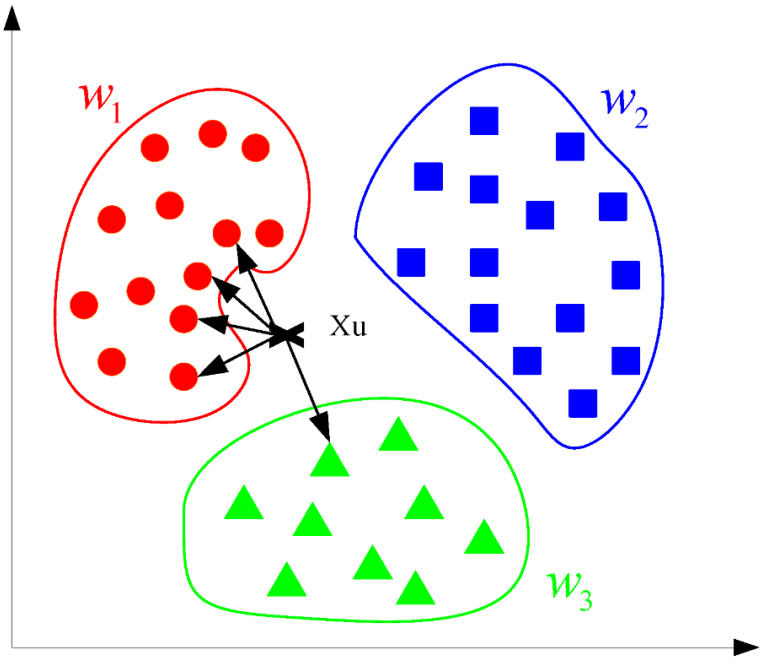

Figure 1: Exemplo do KNN.

O algoritmo k-NN (k-Nearest Neighbors) é um método de aprendizado supervisionado que determina a classe de um ponto de dados com base na maioria das classes encontradas em seus k vizinhos mais próximos. Nesse contexto, a "proximidade" é geralmente medida pela distância euclidiana entre os pontos, que é uma métrica comum para avaliar a dissimilaridade entre instâncias. Apesar de sua simplicidade e interpretabilidade, a escolha do valor de k é crucial, impactando diretamente o desempenho do modelo. Um k muito pequeno pode levar a sobreajuste, enquanto um k muito grande pode suavizar demais as fronteiras entre as classes

2 Preparação dos Dados

Vamos exemplificar o uso do k-NN com o conjunto de dados Iris, uma escolha clássica. O conjunto de dados contém medidas de sépalas e pétalas de três espécies de íris. Importamos o conjunto de dados usando `scikit-learn` e o organizamos em um DataFrame.

```
from sklearn import datasets
import matplotlib.pyplot as plt
import pandas as pd

iris = datasets.load_iris()
df_iris = pd.DataFrame(iris.data, columns=iris.feature_names)

# Adicionando a coluna de classes
df_iris['class'] = iris.target

# Dividindo os dados em atributos (x) e rótulos (y)
x = df_iris.iloc[:, :-1].values
y = df_iris.iloc[:, -1].values
```

3 Divisão dos Dados

A divisão do conjunto de dados em treino e teste é essencial para avaliar o desempenho do modelo em dados não vistos. O uso de uma porcentagem dedicada ao conjunto de teste ajuda a simular a aplicação do modelo em condições do mundo real, evitando o sobreajuste.

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)
```

4 Normalização dos Dados

Normalizar os dados é crucial para o k-NN, pois ele depende da distância entre os pontos. O `StandardScaler` padroniza as características, garantindo que todas contribuam igualmente para as distâncias.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

5 Treinamento e Avaliação do Modelo

Agora, treinamos o modelo k-NN e avaliamos seu desempenho. O relatório de classificação fornece detalhes sobre a precisão, recall e F1-score para cada classe, enquanto a matriz de confusão visualiza os resultados de forma mais intuitiva.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report

k = 8  # Escolha inicial de k
classifier = KNeighborsClassifier(n_neighbors=k)
classifier.fit(x_train, y_train)
```

```
y_pred = classifier.predict(x_test)

# Avaliando o modelo
cm = confusion_matrix(y_test, y_pred)
cr = classification_report(y_test, y_pred)

print("Matriz de Confusão:")
print(cm)
print("\nRelatório de Classificação:")
print(cr)
```

6 Ajustando Hiperparâmetros

A escolha adequada do hiperparâmetro "k" é vital. Utilizamos a validação cruzada e a busca em grade para explorar diferentes valores de "k" e encontrar o mais adequado para nosso conjunto de dados.

```
from sklearn.model_selection import GridSearchCV

# Definindo os parâmetros para ajustar
parametros = {'n_neighbors': [3, 5, 7, 9, 11]}

# Criando o classificador k-NN
knn = KNeighborsClassifier()

# Realizando a validação cruzada para encontrar o melhor valor de "k"
grid_search = GridSearchCV(knn, parametros, cv=5)
grid_search.fit(x_train, y_train)

# Obtendo o melhor valor de "k"
melhor_k = grid_search.best_params_['n_neighbors']

# Utilizando o melhor modelo
melhor_modelo = grid_search.best_estimator_
```

7 Avaliação de Desempenho Adicional

Métricas adicionais, como acurácia, precisão, recall e F1-score, fornecem uma visão abrangente do desempenho do modelo em diferentes aspectos.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Calculando métricas adicionais
acuracia = accuracy_score(y_test, y_pred)
precisao = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print(f'Acurácia: {acuracia}')
print(f'Precisão: {precisao}')
print(f'Recall: {recall}')
print(f'F1 Score: {f1}')
```

8 Visualização dos Resultados

Visualizar os resultados é crucial para entender o desempenho do modelo. Utilizamos *seaborn* para criar uma matriz de confusão visualmente informativa.

```
import seaborn as sns

# Visualizando a matriz de confusão
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=iris.target_names, yticklabels=iris.target_names)
plt.title("Matriz de Confusão")
plt.xlabel("Rótulos Preditos")
plt.ylabel("Rótulos Reais")
plt.show()
```