UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
ELE0606 - TÓPICOS ESPECIAIS EM IA

Docente:
José Alfredo Ferreira Costa

Autor:
Vinícius Venceslau Venancio da Penha

KNN - Vizinhos mais próximos

Natal - RN
Setembro de 2023

Sumário

1. Introdução

O algoritmo k-NN, abreviação de "k-Nearest Neighbors" ou "Vizinhos Mais Próximos", é um dos métodos mais simples e eficazes de aprendizado de máquina supervisionado. É usado principalmente para classificação e regressão. O k-NN é um algoritmo de aprendizado baseado em instância, o que significa que ele toma decisões com base nos dados de treinamento em vez de aprender explicitamente um modelo.

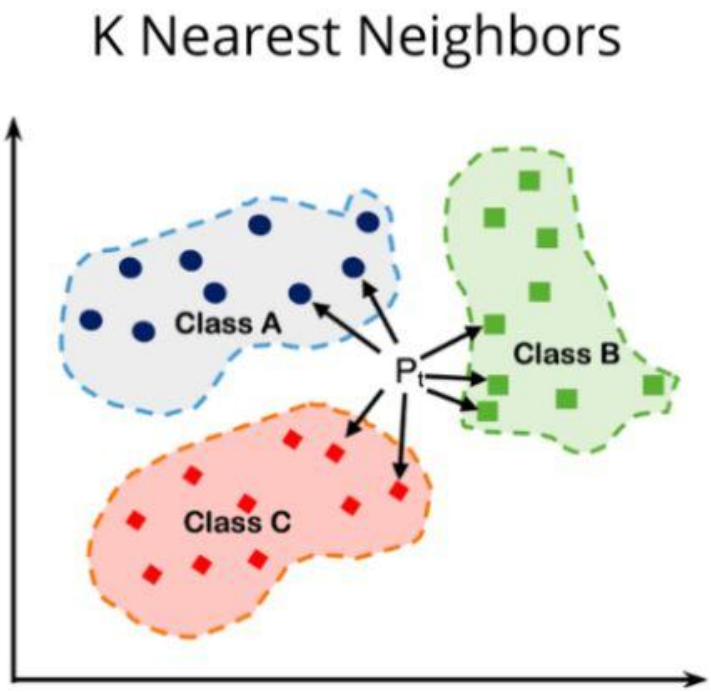

Figura 1: Imagem ilustrativa do funcionamento do método KNN.

2. Princípio Básico

O princípio fundamental subjacente ao k-NN é notavelmente elementar. Inicia-se o algoritmo com a computação das distâncias euclidianas entre o ponto cuja classificação é desconhecida e todos os pontos existentes no conjunto de dados. Subsequentemente, procede-se à seleção dos k pontos mais próximos, facultando a flexibilidade de escolha do valor de k. A atribuição da categoria ao ponto de interesse é, então, realizada com base nas categorias dos k vizinhos mais próximos.

Além disso, é imperativo a composição de um conjunto de dados que abranja as características a serem utilizadas no processo de treinamento do modelo. Neste contexto, é prática comum converter e normalizar as características, como parte da etapa de pré-processamento.

Paralelamente, no desenvolvimento do conjunto de dados, também se procede à atribuição de cada amostra a uma classe específica. Tomemos como exemplo um conjunto de dados simples, onde se encontram duas características denominadas como "x" e "y," que serão empregadas na tarefa de determinar a classe (classificação) à qual pertencem os valores, podendo ser 0 ou 1, que essas características representam.

x	y	Classe
12	20	1
3	12	0
4	5	0
7	15	1
9	3	1
5	1	0
8	8	1
5	10	0
10	15	1
1	20	0

Tabela 1: Exemplo de dataset simples.

Nesse contexto, apresenta-se a representação gráfica da distribuição desses pontos, utilizando a ferramenta de software PlanMaker para a criação do gráfico correspondente. É relevante observar que a construção dessa tabela ocorreu de maneira aleatória.

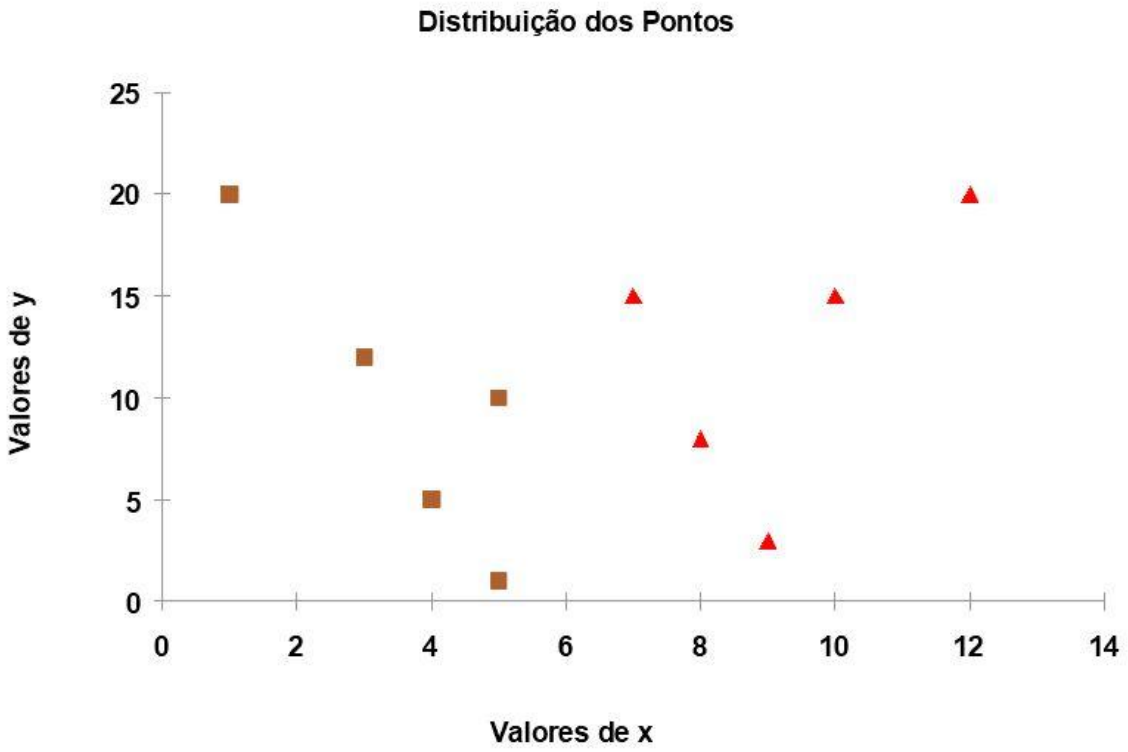

Figura 2: Distribuição dos Pontos graficamente.

Nessa situação, uma vez que as amostras utilizadas na criação do conjunto de dados inicial foram previamente categorizadas, torna-se possível identificar quais dados correspondem ao triângulo vermelho, representado pelo valor 1, e quais dados se relacionam ao quadrado marrom, representado pelo valor 0. Assim, o método KNN emprega um procedimento de treinamento supervisionado, pois contou com a assistência de um especialista para a determinação das categorias de cada amostra.

Por fim, para estabelecer quais amostras são vizinhas umas das outras, torna-se imperativo utilizar uma métrica de distância. Essa métrica avalia a similaridade entre as duas amostras em consideração. Quando o resultado dessa métrica é de baixo valor, indica que as duas amostras estão próximas, denotando maior similaridade; por outro lado, quando o valor é elevado, sugere que as duas amostras estão distantes, indicando menor similaridade entre elas.

2.1 Distâncias

1. **Distância de Manhattan:** A distância de Manhattan vem da ideia de calcular a distância de quarteirões que devem ser percorridos por um carro entre dois locais na cidade de Manhattan. Desse modo, essa distância pode ser definida matematicamente da seguinte maneira:

$$\sum_{i=0}^{n-1} |a_i - b_i|$$

Equação 1: Distância de Manhattan.

Onde, **a** e **b** são duas amostras que deseja-se calcular a similaridade e representam o vetor de características de cada amostra. Outrossim, **n** é a quantidade de características.

2. **Distância Euclidiana:** A distância Euclidiana é similar à distância de Manhattan. Para não ter um resultado com valor negativo, a diferença entre a mesma característica de cada amostra é elevada ao quadrado, e no final da somatória é calculado a raiz quadrada, é como se ao invés de somar as retas dos quarteirões, tem o intuito de traçar um caminho reto entre as duas amostras.

$$\sqrt{\sum_{i=0}^{n-1} (a_i - b_i)^2}$$

Equação 2: Distância Euclidiana.

2.2 Como funciona o K-NN

- **Escolha de k:** O primeiro passo é escolher o valor de k, que representa o número de vizinhos mais próximos a serem considerados. Um valor típico é 3, 5 ou qualquer número ímpar.

- **Medida de Distância:** O algoritmo usa uma métrica de distância, como a distância euclidiana, para calcular a distância entre o novo ponto de dados e todos os pontos de dados no conjunto de treinamento.

- **Seleção dos Vizinhos:** Os k pontos de dados mais próximos com base na métrica de distância são selecionados como os vizinhos.

- **Tomada de Decisão:** Para classificação, o algoritmo faz uma votação entre os k vizinhos para determinar a classe mais comum. Para regressão, ele calcula a média ou a mediana dos valores dos k vizinhos.

2.3 Vantagens e Desvantagens

Vantagens:

- **Simplicidade**: Fácil de entender e implementar.

- **Versatilidade:** Pode ser usado para classificação e regressão.

- **Não paramétrico:** Não faz suposições sobre a distribuição dos dados.

Desvantagens:

- **Sensível a Outliers:** Pode ser sensível a pontos de dados extremos.

- **Custoso em Termos de Memória:** Armazena todos os dados de treinamento.
- **Requer Escolha de k**: A escolha de um valor adequado de k é crucial.

3. Conclusões

O algoritmo k-NN é uma ferramenta valiosa no arsenal de algoritmos de aprendizado de máquina. Sua simplicidade e eficácia o tornam uma escolha popular para problemas de classificação e regressão. No entanto, é importante ajustar os parâmetros adequadamente e considerar suas limitações ao aplicá-lo a problemas do mundo real.

4. Bibliografia

ALFREDO, José. **Algoritmo KNN - K Vizinhos Mais Próximos**. 2023. Meio de Publicação: Documento em PDF. Universidade Federal do Rio Grande do Norte.

Machine Learning Tutorial 13 - K-Nearest Neighbours (KNN algorithm) implementation in Scikit-Learn. Disponível em: https://www.youtube.com/watch?v=OO7Y5wQWnQs.

Wine classification Project using KNN | Machine Learning Project Python | Data Science with Python. Disponível em: https://www.youtube.com/watch?v=IQhh6myW6Fw.

OPENAI. **ChatGPT**. 2023. Disponível em: https://openai.com/.

knn-algoritmo-12-09-2023

September 22, 2023

1 Atividade 2 - KNN Vizinhos mais próximos (Base Wine)

Normalização de Dataframe usando **MinMaxScaler()** e **Z-SCORE**

Aluno: Vinícius Venceslau Venancio da Penha

ELE0606 - Tópicos Especiais em IA

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_wine
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

wine_data = load_wine()
wine_df = pd.DataFrame(wine_data['data'], columns = wine_data['feature_names'])

#Adicionar a coluna de 'target' ou classe no dataframe criado.
wine_df['classe'] = wine_data['target']

#Salvar os valores de classe, antes de removê-los do dataframe.
wine_classe = wine_df['classe']

#Remover a coluna de classe, pois é a saída (resposta) do nosso sistema e a␣
 ↪máquina deve prever esses valores.
wine_df.drop(['classe'], axis=1, inplace=True)

#Normalizar o meu dataframe.
normalizar = MinMaxScaler()

wine_df_normalizado = normalizar.fit_transform(wine_df)

#Converter novamente para dataframe.
wine_df_normalizado = pd.DataFrame(wine_df_normalizado, columns=wine_df.columns)
```

```
#Agora SIM, de fato vamos desenvolver a parte de treinamento:
X_train, X_test, Y_train, Y_test = train_test_split(wine_df_normalizado,␣
 ↪wine_classe, test_size=0.40, random_state=13)

#Convertendo os conjuntos de treinamento e teste do pandas DataFrame para␣
 ↪matrizes NumPy. Isso é comum quando se trabalha com bibliotecas de machine␣
 ↪learning como o scikit-learn, que frequentemente esperam matrizes NumPy como␣
 ↪entrada.
X_train = X_train.to_numpy()
Y_train = Y_train.to_numpy()
X_test = X_test.to_numpy()
Y_test = Y_test.to_numpy()

#Função KNN, cujo os parâmetros são: número de vizinhos e O valor padrão é␣
 ↪'uniform', o que significa que todos os vizinhos têm o mesmo peso.
def aplicacao_knn(neigh, weight='uniform'):
    knn = KNeighborsClassifier(n_neighbors=neigh, weights=weight)
    knn.fit(X_train, Y_train) #Esta linha treina o modelo k-NN com os dados de␣
 ↪treinamento. X_train são os recursos de treinamento e Y_train são os rótulos␣
 ↪de classe correspondentes.
    pred_knn = knn.predict(X_test) #Depois de treinar o modelo, você usa o␣
 ↪conjunto de teste (X_test) para fazer previsões usando o modelo k-NN␣
 ↪treinado. As previsões são armazenadas em pred_knn.
    return pred_knn #Finalmente, a função retorna as previsões feitas pelo k-NN␣
 ↪no conjunto de teste.

'''
Primeiro Teste (Apenas uma simulação e/ou execução!):

k = 7

pred_knn = aplicacao_knn(k)

print(f'A acurácia do modelo para K={k} é de {accuracy_score(Y_test, pred_knn):.
 ↪4f}')
'''

'''
Treinamento voltado para variação nos valores k, bem como na quantidade de␣
 ↪simulações para cada k:
'''

#Valores de k para coleta dos resultados.
valores_k = [1, 3, 5, 7, 9]
```

```
#Número de representantes por classe, entende-se como número de informações de␣
 ↪classes fornecida para máquina.
qnt_simulacoes = [10, 20, 30, 40, 50]

#DataFrame para armazenar as médias da acurácia.
df_precisao = pd.DataFrame(columns=valores_k, index=qnt_simulacoes)

#Lista para armazenar as matrizes confusões geradas.
matrizes_confusoes = []

for qnt_exec in qnt_simulacoes:
  x_train, x_test, y_train, y_test = train_test_split(wine_df_normalizado,␣
 ↪wine_classe, train_size=0.0056*qnt_exec, random_state=13)

  #Conversão para numpy array
  y_train = y_train.to_numpy()
  x_train = x_train.to_numpy()
  x_test = x_test.to_numpy()
  y_test = y_test.to_numpy()

  for k in valores_k:
    knn = KNeighborsClassifier(n_neighbors=k, weights='uniform')
    knn.fit(x_train, y_train)
    previsao_knn = knn.predict(x_test)
    acc = accuracy_score(y_test, previsao_knn)  #Calcula a acurácia, ou seja,␣
 ↪compara y_test com a previsão feita pela máquina.
    df_precisao.at[qnt_exec, k] = acc
    matriz_confusao = confusion_matrix(y_test, previsao_knn)  #Calcula a matriz␣
 ↪de confusão.
    matrizes_confusoes.append(matriz_confusao)  #Adiciona a matriz de confusão␣
 ↪à lista.

#Solicitar ao usuário o índice da matriz de confusão que deseja visualizar:
while True:
    try:
        indice = int(input("Digite o índice da matriz de confusão que deseja␣
 ↪visualizar (0 a {}), dado que 0 até 4 refere-se a 10 objetos por classe, 5␣
 ↪até 8 está associado a 20 objetos por classe...\n".
 ↪format(len(matrizes_confusoes) - 1)))
        print('O valor informado foi:', indice)
        print('\n')
        if 0 <= indice < len(matrizes_confusoes):
            break
        else:
            print("Índice fora do intervalo válido. Tente novamente.\n")
    except ValueError:
```

```
        print("Entrada inválida. Digite um número inteiro válido.\n")

#Exibir a matriz de confusão escolhida:
matriz_escolhida = matrizes_confusoes[indice]
print("Matriz de Confusão:\n")
print(matriz_escolhida)
print("\n")
print("A tabela de precisão final ficou desta maneira:\n")
df_precisao
```

```
Digite o índice da matriz de confusão que deseja visualizar (0 a 24), dado que 0
até 4 refere-se a 10 objetos por classe, 5 até 8 está associado a 20 objetos por
classe…
0
O valor informado foi: 0


Matriz de Confusão:

[[49  9  0]
 [ 1 59  7]
 [ 0  0 44]]


A tabela de precisão final ficou desta maneira:
```

[1]:

	1	3	5	7	9
10	0.899408	0.621302	0.579882	0.591716	0.39645
20	0.91195	0.91195	0.867925	0.566038	0.54717
30	0.926174	0.926174	0.932886	0.919463	0.892617
40	0.928058	0.935252	0.956835	0.942446	0.956835
50	0.953488	0.96124	0.976744	0.968992	0.96124

Para usar normalização z-score:

from sklearn.preprocessing import StandardScaler

Substituir a linha onde usa-se MinMaxScaler, ou seja, "normalizar = MinMaxScaler()"

Pela normalização Z-score usando StandardScaler, isso é, "normalizar = StandardScaler()"

wine_df_normalizado = normalizar.fit_transform(wine_df)

Consideração Importante:

Em suma, a normalização é uma etapa importante no processamento de dados que ajuda a melhorar a eficiência e o desempenho dos algoritmos de aprendizado de máquina, garantindo que os dados estejam em uma escala adequada e comparável. A escolha de como normalizar os dados depende do algoritmo e das características dos dados em questão.

Normalização Min-Max:

1. Redimensiona os dados para um intervalo específico, geralmente [0, 1] ou [-1, 1].
2. É sensível a outliers, pois os valores extremos podem afetar significativamente a escala dos dados.
3. É apropriado quando você tem a priori conhecimento sobre a faixa de valores que seus dados devem estar.

Normalização Z-score (padronização):

1. Redimensiona os dados para que tenham média zero e desvio padrão igual a um.
2. É menos sensível a outliers, pois utiliza a média e o desvio padrão para a escala, tornando-a mais robusta.
3. É apropriado quando você não tem informações sobre a escala ideal dos seus dados e deseja remover o efeito das unidades de medida.

A melhor escolha pode ser determinada empiricamente por meio de experimentação e validação cruzada, avaliando qual normalização se ajusta melhor ao seu conjunto de dados e ao seu algoritmo de aprendizado de máquina.

Referências:

ALFREDO, José. **Algoritmo KNN - K Vizinhos Mais Próximos**. 2023. Meio de Publicação: Documento em PDF. Universidade Federal do Rio Grande do Norte.

Machine Learning Tutorial 13 - K-Nearest Neighbours (KNN algorithm) implementation in Scikit-Learn. Disponível em: https://www.youtube.com/watch?v=OO7Y5wQWnQs.

Wine classification Project using KNN | Machine Learning Project Python | Data Science with Python. Disponível em: https://www.youtube.com/watch?v=IQhh6myW6Fw.

OPENAI. ChatGPT. 2023. Disponível em: https://openai.com/.