Universidade Federal do Rio Grande do Norte
Departamento de Engenharia Elétrica
ELE0606 - TOPICOS ESPECIAIS EM INTELIGENCIA ARTIFICIAL

Classificação da Base de Dados Wine com kNN

Aluno: Pedro Artur F. Varela de Lira
Professor: José Alfredo Ferreira Costa

Setembro
2023

Conteúdo

1 Introdução e Contexto **1**

2 Metodologia **1**

3 Pseudocódigo **2**

4 Resultados **4**

5 Conclusão e Discussões **6**

1 Introdução e Contexto

A análise e classificação de vinhos desempenham um papel fundamental na indústria vinícola, ajudando a identificar a qualidade e as características dos vinhos com base em diversos atributos. Neste relatório, descreveremos um experimento de classificação de vinhos utilizando o algoritmo k-Nearest Neighbors (kNN) implementado com a biblioteca Scikit-Learn em Python.

Para este experimento, utilizamos o Wine Dataset, uma base de dados amplamente conhecida e utilizada na área de aprendizado de máquina. Este conjunto de dados contém informações sobre vinhos de três diferentes classes, representando três cultivares de uvas da região da Itália. Cada amostra é caracterizada por 13 atributos, incluindo teores de álcool, acidez málica e intensidade de cor. O objetivo é treinar um modelo de classificação capaz de distinguir as três classes de vinhos com base nessas características.

O algoritmo k-Nearest Neighbors é um método de aprendizado de máquina supervisionado que se baseia na proximidade entre os pontos de dados. A ideia central do kNN é classificar um novo ponto de dados com base na maioria das classes dos k pontos mais próximos a ele no espaço de atributos. Isso significa que o kNN toma decisões com base na vizinhança dos dados e não requer uma fase de treinamento complexa.

O algoritmo k-Nearest Neighbors é uma abordagem intuitiva e versátil para problemas de classificação e regressão. Sua simplicidade reside na ideia de que objetos semelhantes tendem a estar próximos no espaço de atributos. Ao determinar as k instâncias mais próximas de um novo ponto de dados a ser classificado, o kNN toma decisões com base na votação da maioria dessas instâncias vizinhas. O valor de k é um hiperparâmetro crucial, pois afeta a sensibilidade do modelo a ruídos e pode influenciar a sua capacidade de generalização. Este algoritmo é especialmente útil quando a distribuição dos dados é complexa ou não linear, permitindo que ele se adapte a padrões intrincados nos dados de forma natural. Durante este experimento, exploraremos como ajustar o valor de k e examinaremos seu impacto no desempenho do modelo de classificação de vinhos.

O objetivo principal deste experimento é aplicar o algoritmo kNN para classificar corretamente as amostras de vinho em suas respectivas classes.

2 Metodologia

A fim de aplicar o algoritmo kNN para classificar corretamente as amostras de vinho em suas respectivas classes, realizaremos as seguintes etapas:

- **Pré-processamento de Dados**: Faremos uma análise exploratória dos dados, verificando se há valores ausentes, normalizando os atributos e dividindo o conjunto de dados em conjuntos de treinamento e teste.
- **Treinamento do Modelo kNN**: Implementaremos o algoritmo kNN com o Scikit-Learn e ajustaremos o modelo utilizando o conjunto de treinamento.
- **Avaliação do Modelo**: Usaremos o conjunto de teste para avaliar o desempenho do modelo kNN em termos de métricas de classificação, como precisão, recall e F1-score. Também podemos ajustar o valor de k para otimizar o desempenho.
- **Visualização dos Resultados**: Utilizaremos gráficos e visualizações para apresentar os resultados de forma clara e interpretável.

3 Pseudocódigo

O pseudocódigo abaixo reflete as principais etapas do código implementado no experimento, desde a importação das bibliotecas até a análise dos resultados com diferentes configurações de normalização e valores de k para o kNN.

```
# Importacao das bibliotecas necesscrias
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 StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

# Carregamento do conjunto de dados Wine
wine_data = load_wine()
wine_df = pd.DataFrame(data=wine_data['data'], columns=
    wine_data['feature_names'])
wine_df['target'] = wine_data['target']

# Analise exploratoria dos dados
wine_df.info()
wine_df.describe()

# Visualizacao da correlacao entre os atributos
corr = wine_df.corr()
```

```
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)] = True
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Pre-processamento dos dados
wine_df.drop(['ash'], axis=1, inplace=True)
result = wine_df['target']
wine_df.drop(['target'], axis=1, inplace=True)

# Normalizacao dos dados usando escalonamento linear
normalized_wine_df = wine_df.apply(escalonamento_linear)

# Divisao dos dados em conjuntos de treinamento e teste
X_train, X_test, Y_train, Y_test = train_test_split(
    normalized_wine_df, result, test_size=0.30, random_state
    =11)

# Treinamento do modelo k-Nearest Neighbors (kNN)
k = 5
pred_knn = apply_knn(k)
print(f'A acuracia do modelo para K={k} e de {accuracy_score(
    Y_test, pred_knn):.4f}')

# Analise dos resultados
k_values = [1, 3, 5, 7, 9]
execution_counts = [10, 20, 30, 40, 50]

result_df = pd.DataFrame(columns=k_values, index=
    execution_counts)

for exec_count in execution_counts:
    x_train, x_test, y_train, y_test = train_test_split(
    normalized_wine_df, result, train_size=0.0056 * exec_count
    , random_state=11)
    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 k_values:
        knn = KNeighborsClassifier(n_neighbors=k, weights='
    uniform')
        knn.fit(x_train, y_train)
        pred_knn = knn.predict(x_test)
        acc = accuracy_score(y_test, pred_knn)
        result_df.at[exec_count, k] = acc

# Matriz de Confusao
cm = confusion_matrix(Y_test, apply_knn(3))

```

```
# Normalizacao Z-score dos dados
normalized_wine_df = wine_df.apply(z_score_normalize_column)

# Analise dos resultados com normalizacao Z-score
result_df = pd.DataFrame(columns=k_values, index=
    execution_counts)

for exec_count in execution_counts:
    x_train, x_test, y_train, y_test = train_test_split(
    normalized_wine_df, result, train_size=0.0056 * exec_count
    , random_state=11)
    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 k_values:
        knn = KNeighborsClassifier(n_neighbors=k, weights='
    uniform')
        knn.fit(x_train, y_train)
        pred_knn = knn.predict(x_test)
        acc = accuracy_score(y_test, pred_knn)
        result_df.at[exec_count, k] = acc
```

Listing 1: Pseudocódigo

É importante ressaltar que o pseudocódigo não foi implementado para funcionar efetivamente como solução do problema, mas sim como um meio de representar facilmente as etapas principais realizadas no Jupyter Notebook. O código para o projeto completo e o resultados estão no link: `https://github.com/pedro-varela1/Arquivos_ELE-606/blob/main/Atividade_2_kNN_ELE606.ipynb`.

4 Resultados

Para realizar a validação geral do modelo, foram realocados 70% o dataset Wine para treinamento e 30% para teste e a acurácia foi medida para cada elemento presente nos dados de teste e treinamento variando o valor de K.

No gráfico da Figura 1, vemos que, tanto a quantidade de pacotes de treinamento quanto a valor de k influenciaram significativamente na acurácia, mas, claramente, nos dados de treinamento, a acurácio sempre foi maior, já que o modelo de kNN já tem acesso a esses dados. No gráfico, observa-se que, dentre os valores de k utilizados, o que obteve a acurácia mais alta foi $k = 3$. Com valores de K mais altos, houve uma menor acurácia devido a um possível overfitting. O modelo não generalizou bem para $k > 3$. Já para $k = 1$ houve uma melhor adequação que $k > 3$, mais pior que para $k = 3$,

isso pode mostrar um underfitting do modelo.

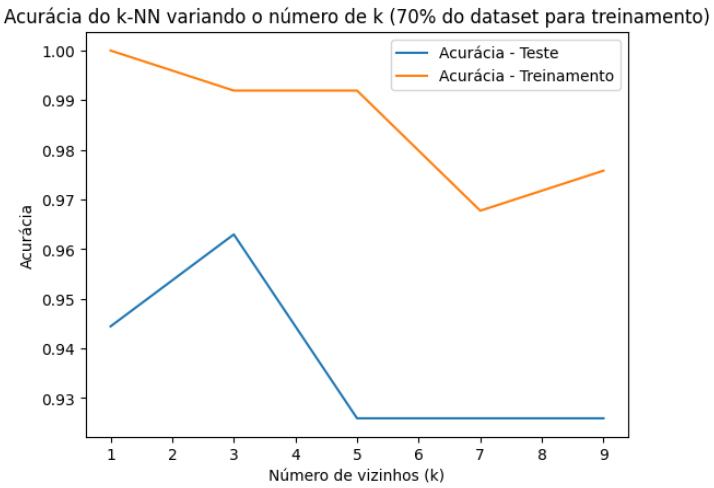

Figura 1: Acurácia do k-NN variando o número de K

Foi realizado, também, a construção de uma tabela que expôe as acurácias variando tanto o número de K, quanto o número de pacotes ou linhas do dataset utilizadas no treinamento. Essa tabela mostra que o valor de K não influenciou significativamente para muitos pacotes de treinamento, mas em geral, sua melhor acurácia foi entre $k = 3$ e $k = 5$. Além disso, observa-se um resultado melhor no geral para um valor de k menor. Porém, o máximo valor de acurácia permaneceu na mesma faixa, quando comparado ao outro tipo de normalização.

Número de Pacotes Utilizados no Treinamento	1	3	5	7	9
10	0.721893	0.621302	0.384615	0.384615	0.384615
20	0.930818	0.955975	0.955975	0.880503	0.672956
30	0.932886	0.939597	0.95302	0.946309	0.885906
40	0.942446	0.935252	0.94964	0.942446	0.935252
50	0.96124	0.968992	0.968992	0.945736	0.937984

Figura 2: Acurácia do k-NN variando o número de K e com o número de pacotes de treinamento

Por fim, uma matriz de confusão foi feita (Figura 3), mostrando que, dentre 54 linhas de teste, 52 vinhos foram classificados corretamente. Apenas 1 vinho da Classe 0 foi classificado incorretamente como da Classe 1 e um

vinho da Classe 2 foi incorretamente classificado como integrante da Classe 1. O ideal é que essa matriz fosse diagonal, ou seja, 100% dos dados classificados corretamente.

	Classe 0	Classe 1	Classe 2
Classe 0	21	0	0
Classe 1	1	20	1
Classe 2	0	0	11

Figura 3: Matriz de Confusão

5 Conclusão e Discussões

Os resultados obtidos neste estudo sobre o conjunto de dados Wine usando o algoritmo k-NN fornecem insights valiosos sobre como o modelo se comporta em diferentes configurações. Alguns pontos de discussão a serem considerados incluem:

- **Escolha de K**: o fato de K=3 ter produzido o melhor desempenho sugere que considerar um número intermediário de vizinhos mais próximos foi mais benéfico em comparação com valores muito altos ou muito baixos. Isso destaca a importância de ajustar cuidadosamente o hiperparâmetro K ao usar o k-NN.

- **Overfitting vs. Underfitting**: a análise dos resultados para diferentes valores de K também indicou a possibilidade de overfitting (para valores altos de K) e underfitting (para K=1). Encontrar um equilíbrio é crucial para obter um modelo que generalize bem para dados não vistos.

- **Impacto do Tamanho do Conjunto de Treinamento**: a variação no tamanho do conjunto de treinamento não teve um impacto significativo na escolha de K, mas mostrou que um valor menor de K tende a produzir resultados melhores em geral. Isso sugere que um modelo treinado com uma quantidade menor de vizinhos pode ser mais eficaz.

- **Matriz de Confusão**: embora a acurácia geral tenha sido alta, a matriz de confusão destacou as classes em que o modelo teve dificuldades. É importante considerar o contexto em que esses erros ocorreram, pois eles podem ter implicações práticas, dependendo do problema real.

Bibliografia

IBM. (s.d.). k-Nearest Neighbors (kNN). Disponível em: https://www.ibm.com/topics/knn.Acessoem:18/09/2023.

SCIKIT-LEARN. (s.d.). sklearn.datasets.load_wine. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_wine.html. Acesso em: 18/09/2023.