

Teste de Machine Learning Iris



Estou fazendo alguns testes de Machine Learning com o dataset Iris. Conhece?



Sim, conheço o **dataset Iris**! Ele é um dos conjuntos de dados mais clássicos e amplamente utilizados em **aprendizado de máquina** e **estatística**. Foi introduzido por Ronald A. Fisher em 1936 para ilustrar a técnica de análise discriminante.

Informações do dataset Iris:

- **Objetivo:** Classificar flores da espécie **Iris** em três classes:
 1. **Iris setosa**
 2. **Iris versicolor**
 3. **Iris virginica**
- **Atributos (features):**
 1. Comprimento da sépala (**sepal length**)
 2. Largura da sépala (**sepal width**)
 3. Comprimento da pétala (**petal length**)
 4. Largura da pétala (**petal width**)
- **Tamanho:** 150 amostras, com 50 amostras de cada uma das três classes.

É muito usado para testes de algoritmos de classificação como **K-Nearest Neighbors (KNN)**, **SVM**, **Árvores de Decisão**, e **Redes Neurais**.

Se estiver realizando experimentos com esse dataset, posso ajudar com algoritmos, pré-processamento, ou visualizações. Como está utilizando o dataset?



Perfeito!

Gostaria que você me fornecesse a implementação do algoritmo de KNN para esse dataset. Não quero que use nenhuma biblioteca com esse algoritmo pronto, apenas bibliotecas auxiliares como pandas e numpy



Aqui está a implementação do algoritmo **K-Nearest Neighbors (KNN)** para o dataset **Iris** utilizando apenas bibliotecas auxiliares como ``pandas`` e ``numpy``. Essa implementação faz o cálculo das distâncias e a classificação manualmente, sem usar bibliotecas que já implementam o KNN.

Passos:

1. Carregar o dataset Iris.
2. Dividir em treino e teste.
3. Implementar o algoritmo KNN.
4. Calcular a acurácia do modelo.

Código:

python

```
import pandas as pd
import numpy as np
from collections import Counter
from sklearn.model_selection import train_test_split

# Carregar o dataset Iris
def load_iris_dataset():
    url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
    columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']
    iris = pd.read_csv(url, header=None, names=columns)
```

```

return iris

# Função para calcular a distância euclidiana
def euclidean_distance(row1, row2):
    return np.sqrt(np.sum((row1 - row2) ** 2))

# Implementação do KNN
def knn(train_data, test_data, k):
    predictions = []

    # Para cada amostra no conjunto de teste
    for test_row in test_data:
        distances = []

        # Calcular a distância de cada ponto do conjunto de treino
        for train_row in train_data:
            distance = euclidean_distance(test_row[:-1], train_row[:-1])
            distances.append((train_row, distance))

        # Ordenar por distância (os mais próximos vêm primeiro)
        distances.sort(key=lambda x: x[1])

        # Selecionar os 'k' mais próximos
        neighbors = [dist[0][-1] for dist in distances[:k]]

        # Contar a frequência das classes dos vizinhos e escolher a mais comum
        prediction = Counter(neighbors).most_common(1)[0][0]
        predictions.append(prediction)

    return predictions

# Função para calcular a acurácia
def accuracy(y_true, y_pred):
    return np.sum(y_true == y_pred) / len(y_true)

# Main
if __name__ == "__main__":
    # Carregar o dataset
    iris_data = load_iris_dataset()

    # Codificar as classes em valores numéricos
    iris_data['species'] = iris_data['species'].astype('category').cat.codes

    # Dividir o dataset em features e labels
    X = iris_data.iloc[:, :-1].values
    y = iris_data.iloc[:, -1].values

    # Dividir em treino e teste
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

```

```
# Concatenar as labels de treino com os dados para facilitar o processamento no algoritmo
train_data = np.column_stack((X_train, y_train))
test_data = np.column_stack((X_test, y_test))

# Definir o valor de k
k = 3

# Fazer previsões
predictions = knn(train_data, test_data, k)

# Calcular a acurácia
acc = accuracy(y_test, predictions)
print(f"Acurácia do KNN: {acc * 100:.2f}%")
```

Explicação:

1. **Carregar o dataset:** Utilizamos o `pandas` para carregar o dataset Iris diretamente da URL.
2. **Distância Euclidiana:** Calculamos a distância euclidiana entre duas amostras de forma manual.
3. **KNN:** Para cada amostra de teste, calculamos as distâncias em relação às amostras de treino, ordenamos por proximidade e pegamos os `k` vizinhos mais próximos. A classe mais comum entre esses vizinhos é a previsão do modelo.
4. **Acurácia:** Comparamos as previsões do modelo com as classes reais do conjunto de teste.

Resultados:

Essa implementação básica faz a previsão para cada amostra de teste com base nos `k` vizinhos mais próximos. Você pode ajustar o valor de `k` para observar como isso afeta a performance do modelo.

Posso ajudar com ajustes ou otimizações, caso precise!



Qual foi o critério de escolha $k = 3$?



A escolha de $k=3$ no algoritmo **K-Nearest Neighbors (KNN)** foi arbitrária no código fornecido, mas a seleção do valor de k é uma etapa crítica para o desempenho do modelo. Existem alguns critérios comuns que você pode usar para escolher o valor ideal de k :

Critérios para a escolha de

1. Tamanho do Dataset:

- Se o conjunto de dados é pequeno, valores menores de k (como 3 ou 5) são geralmente mais apropriados. Com mais dados, valores maiores de k podem ser usados para estabilizar a previsão.

2. Valores Ímpares para :

- Muitas vezes, é recomendado escolher k como um número ímpar, especialmente em problemas de classificação binária, para evitar empates. Isso não é tão relevante no dataset **Iris**, que tem três classes.

3. Cross-Validation:

- Um critério mais sólido é realizar **validação cruzada (cross-validation)**. Isso envolve testar diferentes valores de k em várias partições do conjunto de dados (treino/teste), e escolher o k que maximiza a acurácia média.

4. Bias-Variance Tradeoff:

- Valores menores de k (como $k=1$) podem levar a um modelo com **baixo viés** e **alta variância** — ou seja, o modelo pode sobreajustar (overfitting) os dados de treino.
- Valores maiores de k podem aumentar o **viés** e reduzir a **variância**, tornando o modelo mais estável, mas com maior risco de subajustar (underfitting) os dados.

5. Visualização e Testes:

- Em datasets de baixa dimensionalidade, como o Iris, você pode visualizar o efeito de diferentes valores de k para ter uma ideia mais intuitiva. Por exemplo, plote a acurácia ou a taxa de erro para diferentes valores de k.

Como testar diferentes valores de

Você pode alterar o valor de k no código anterior e observar o desempenho. Um processo mais robusto envolve realizar uma busca por validação cruzada para encontrar o melhor k. Aqui está uma modificação para testar valores diferentes de k usando validação cruzada:

Modificação para testar múltiplos valores de

```
python

from sklearn.model_selection import cross_val_score

def knn_cross_validation(train_data, k_values):
    accuracies = []
    for k in k_values:
        predictions = knn(train_data, train_data, k)
        accuracy_value = accuracy(train_data[:, -1], predictions)
        accuracies.append(accuracy_value)
    return accuracies

# Testar valores de k de 1 a 10
k_values = range(1, 11)
train_data = np.column_stack((X_train, y_train))
accuracies = knn_cross_validation(train_data, k_values)

# Exibir as acurácias para cada valor de k
for k, acc in zip(k_values, accuracies):
    print(f"k = {k}: Acurácia = {acc * 100:.2f}%")
```

Essa modificação avalia o desempenho de diferentes valores de k e exibe a acurácia para cada um. Em produção, é importante experimentar e ajustar o valor de k com base em **cross-validation** para obter o melhor resultado possível.

Resumo:

- A escolha de $k=3$ no exemplo foi arbitrária, mas o valor ideal de k pode ser encontrado usando validação cruzada ou experimentação.
- Testar múltiplos valores de k e analisar a acurácia média em diferentes partições do dataset é a maneira mais robusta de escolher o melhor k .



Para o dataset Iris, qual k você indicaria?



Para o dataset **Iris**, o valor de **k** ideal geralmente depende de experimentação e validação, mas com base nas características desse dataset (como o número de amostras e as classes bem separadas), aqui estão algumas sugestões:

Sugestão baseada no dataset Iris:

1. Tamanho do dataset:

- O dataset Iris tem **150 amostras** distribuídas igualmente entre três classes (50 amostras por classe). Um número pequeno de amostras favorece valores menores de k (como 3, 5 ou 7).

2. Número de classes:

- Como há três classes distintas, você pode escolher um valor ímpar para k , como $k=3$ ou $k=5$, para ajudar a desempatar a votação durante a classificação.

3. Experimentação com validação cruzada:

- Embora $k=3$ e $k=5$ sejam comumente usados, a escolha ótima de k

pode ser obtida com **validação cruzada**, testando diversos valores de kkk e observando o desempenho do modelo.

Teste de valores de

Um procedimento de validação cruzada pode ajudar a determinar o melhor valor de kkk. Aqui está uma sugestão para realizar essa busca e identificar o valor de kkk que maximiza a acurácia.

Código para testar múltiplos valores de

```
python

import pandas as pd
import numpy as np
from collections import Counter
from sklearn.model_selection import train_test_split, cross_val_score

# Carregar o dataset Iris
def load_iris_dataset():
    url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
    columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']
    iris = pd.read_csv(url, header=None, names=columns)
    return iris

# Função para calcular a distância euclidiana
def euclidean_distance(row1, row2):
    return np.sqrt(np.sum((row1 - row2) ** 2))

# Implementação do KNN
def knn(train_data, test_data, k):
    predictions = []

    # Para cada amostra no conjunto de teste
    for test_row in test_data:
        distances = []

        # Calcular a distância de cada ponto do conjunto de treino
        for train_row in train_data:
            distance = euclidean_distance(test_row[:-1], train_row[:-1])
            distances.append((train_row, distance))

        # Ordenar por distância (os mais próximos vêm primeiro)
        distances.sort(key=lambda x: x[1])
```



```

# Selecionar os 'k' mais próximos
neighbors = [dist[0][-1] for dist in distances[:k]]

# Contar a frequência das classes dos vizinhos e escolher a mais comum
prediction = Counter(neighbors).most_common(1)[0][0]
predictions.append(prediction)

return predictions

# Função para calcular a acurácia
def accuracy(y_true, y_pred):
    return np.sum(y_true == y_pred) / len(y_true)

# Função para testar diferentes valores de k
def knn_cross_validation(train_data, test_data, k_values):
    accuracies = []
    for k in k_values:
        predictions = knn(train_data, test_data, k)
        acc = accuracy(test_data[:, -1], predictions)
        accuracies.append((k, acc))
    return accuracies

# Main
if __name__ == "__main__":
    # Carregar o dataset
    iris_data = load_iris_dataset()

    # Codificar as classes em valores numéricos
    iris_data['species'] = iris_data['species'].astype('category').cat.codes

    # Dividir o dataset em features e labels
    X = iris_data.iloc[:, :-1].values
    y = iris_data.iloc[:, -1].values

    # Dividir em treino e teste
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Concatenar as labels com os dados para facilitar o processamento no algoritmo
    train_data = np.column_stack((X_train, y_train))
    test_data = np.column_stack((X_test, y_test))

    # Testar valores de k de 1 a 10
    k_values = range(1, 11)
    results = knn_cross_validation(train_data, test_data, k_values)

    # Exibir as acurácias para cada valor de k
    for k, acc in results:
        print(f"k = {k}: Acurácia = {acc * 100:.2f}%")

```

Explicação:

- Este código testa valores de k de 1 a 10.
- Para cada valor de k , calcula-se a acurácia no conjunto de teste.
- Você pode observar o desempenho de diferentes valores e escolher aquele que maximiza a acurácia.

Resultados esperados:

- Valores de k menores (como $k=1$) podem apresentar maior variância (possivelmente overfitting), mas para o dataset Iris, isso pode ainda dar uma boa acurácia.
- Valores de k como **3, 5, ou 7** geralmente produzem bons resultados com uma estabilidade entre bias e variância.

Valor sugerido:

Na prática, para o dataset Iris, $k=3$ costuma ser um ótimo ponto de equilíbrio entre simplicidade e desempenho, mas é sempre importante testar e validar com base nos resultados de cross-validation.

Se precisar, posso ajudá-lo a interpretar os resultados e ajustar o modelo!



```
k = 1: Acurácia = 100.00%
k = 2: Acurácia = 100.00%
k = 3: Acurácia = 100.00%
k = 4: Acurácia = 100.00%
k = 5: Acurácia = 100.00%
k = 6: Acurácia = 100.00%
k = 7: Acurácia = 96.67%
k = 8: Acurácia = 100.00%
k = 9: Acurácia = 100.00%
k = 10: Acurácia = 100.00%
```

Executei o programa e a saída foi essa



Esses resultados são surpreendentemente altos, indicando que o algoritmo KNN está classificando o dataset **Iris** de maneira muito eficaz, independentemente do valor de k . Aqui estão algumas observações e interpretações sobre os resultados:

Interpretação dos resultados:

1. **Acurácia de 100% para vários valores de k :** Para $k=1,2,3,4,5,6,8,9,10$, o modelo atingiu **100% de acurácia** no conjunto de teste. Isso mostra que o dataset Iris, em geral, é muito fácil de classificar, provavelmente devido à separação clara entre as classes.
2. **Queda leve na acurácia para $k=7$:** Para $k=7$, a acurácia caiu para **96,67%**, o que ainda é muito bom. Essa pequena queda pode ser atribuída ao fato de que, com k maior, o algoritmo está considerando mais vizinhos, o que pode introduzir algum erro ao incorporar amostras de outras classes que não são tão próximas no espaço de características.

O que esses resultados indicam:

- O dataset **Iris** é bem comportado em termos de separação das classes. As três classes de flores são bastante distintas em relação às características medidas (comprimento e largura das pétalas e sépalas), permitindo uma classificação quase perfeita com KNN.
- **Valores menores de k** (como $k=1$) conseguem capturar as divisões claras entre as classes, resultando em alta acurácia. Entretanto, k ligeiramente maiores (como 5 ou 7) ajudam a suavizar um pouco as previsões, agregando mais amostras, o que pode ser útil em conjuntos de dados mais complexos.

Sugestão:

Com base nesses resultados, qualquer valor de k entre **1 e 10** funciona muito bem para o dataset Iris. Entretanto, eu recomendaria manter **$k=3$** ou **$k=5$** como valores ideais, pois eles trazem o melhor equilíbrio entre simplicidade e robustez, além de serem menos suscetíveis a outliers em datasets maiores ou mais complexos.

Consideração adicional:

Se precisar validar esses resultados em outro conjunto de dados ou testar o modelo em diferentes partições de treino/teste, você pode realizar uma **validação cruzada k-fold** para garantir que o modelo tenha um desempenho consistente.

Se quiser explorar mais alguma variação ou análise, fique à vontade para perguntar!