Atividade 0: Classificação e Clustering com o Dataset Iris

Bem-vindos à aula prática! Hoje vocês vão aprender, de forma guiada e passo a passo, como utilizar Python para aplicar conceitos básicos de aprendizado supervisionado e não supervisionado. Vamos usar o famoso dataset Iris para praticar.

Objetivos

- **Aprendizado Supervisionado:** Criar e avaliar um modelo de classificação utilizando uma árvore de decisão.
- **Aprendizado Não Supervisionado:** Agrupar os dados com o algoritmo K-Means e comparar os resultados com as classes reais.

Pré-Requisitos

- **Ambiente de Programação:** Jupyter Notebook (ou similar) com Python instalado.
- **Bibliotecas Necessárias:**
  Para instalar as bibliotecas, abra o terminal e execute:

  ```
  pip install pandas numpy matplotlib seaborn scikit-learn
  ```

Parte 1: Aprendizado Supervisionado – Classificação

Passo 1: Preparando o Ambiente

Abra seu Jupyter Notebook e crie uma nova célula. Digite o código abaixo para importar as bibliotecas que iremos utilizar:

```
# Importa bibliotecas necessárias para manipulação de dados e criação de gráficos
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Importa ferramentas do scikit-learn para carregar dados, dividir conjuntos e criar modelos
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

> O que está acontecendo aqui?

> - Estamos carregando bibliotecas para manipulação de dados (NumPy, Pandas), para visualização (Matplotlib, Seaborn) e para construir modelos de machine learning (scikit-learn).

Passo 2: Carregando o Dataset Iris

Em uma nova célula, copie e cole o seguinte código para carregar o dataset Iris e transformá-lo em um DataFrame:

```
# Carrega o dataset Iris
iris = load_iris()

# Cria um DataFrame com os dados (features) do dataset
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)

# Adiciona a coluna de rótulo (target) e converte os números em nomes das espécies
df['target'] = iris.target
df['species'] = df['target'].apply(lambda x: iris.target_names[x])

# Exibe as primeiras 5 linhas para verificar se os dados foram carregados corretamente
display(df.head())
```

> Explicação:
>
> - Transformamos os dados do Iris em uma tabela (DataFrame) para facilitar a visualização e manipulação.
> - A coluna `species` mostra o nome da espécie correspondente a cada registro.

Passo 3: Explorando e Visualizando os Dados

Para entender melhor os dados, vamos criar gráficos que mostram as relações entre as variáveis:

```
# Cria um pairplot para visualizar as relações entre todas as variáveis, colorindo pelos nomes das espécie
s
sns.pairplot(df, hue='species')
plt.show()
```

> O que o código faz?
>
> - O `pairplot` gera gráficos de dispersão para cada par de variáveis, ajudando a identificar padrões e diferenças entre as espécies.

Passo 4: Dividindo os Dados para Treinamento e Teste

Agora, vamos separar os dados em duas partes: uma para treinar o modelo e outra para testar sua performance.

```
# Define as variáveis independentes (X) e a variável dependente (y)
X = df[iris.feature_names]
y = df['target']

# Divide os dados: 70% para treinamento e 30% para teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

> Dica:
>
> - O parâmetro `random_state` garante que a divisão seja sempre a mesma, facilitando a comparação dos resultados.

Passo 5: Treinando o Modelo de Classificação

Utilize uma árvore de decisão para criar um modelo que classifique as flores:

```
# Cria o modelo de árvore de decisão
clf = DecisionTreeClassifier(random_state=42)

# Treina o modelo utilizando os dados de treinamento
clf.fit(X_train, y_train)
```

> Nota:
>
> - O método `fit` é responsável por ajustar o modelo aos dados de treinamento.

Passo 6: Avaliando o Modelo

Depois do treinamento, vamos avaliar o desempenho do nosso modelo:

```
# Faz as predições com o modelo treinado utilizando os dados de teste
y_pred = clf.predict(X_test)

# Calcula e exibe a acurácia (percentual de acertos)
print("Acurácia:", accuracy_score(y_test, y_pred))

# Mostra a matriz de confusão, que compara os valores reais com os preditos
print("Matriz de Confusão:\n", confusion_matrix(y_test, y_pred))

# Exibe um relatório detalhado com métricas de avaliação
report_dict = classification_report(y_test, y_pred, output_dict=True)
report_df = pd.DataFrame(report_dict).transpose()
display(report_df)
```

> Explicação:
>
> - Essas métricas ajudam a entender a eficácia do modelo na hora de classificar novas amostras.

Parte 2: Aprendizado Não Supervisionado – Clustering com K-Means

Passo 7: Aplicando o K-Means para Agrupamento

Nesta etapa, vamos usar o K-Means para agrupar os dados sem utilizar os rótulos das espécies.

```
# Importa o algoritmo K-Means do scikit-learn
from sklearn.cluster import KMeans

# Cria o modelo K-Means com 3 clusters (pois o dataset possui 3 espécies)
kmeans = KMeans(n_clusters=3, random_state=42)

# Aplica o modelo ao dataset e armazena os clusters obtidos
clusters = kmeans.fit_predict(iris.data)

# Adiciona a coluna 'cluster' ao DataFrame para identificar a qual grupo cada registro pertence
df['cluster'] = clusters

# Exibe as primeiras linhas para ver a nova coluna com os clusters
display(df.head())
```

> Importante:
>
> - O K-Means agrupa os dados com base em semelhanças, sem considerar os nomes das espécies.

Passo 8: Visualizando os Clusters

Crie um gráfico para visualizar como os dados foram agrupados:

```
# Cria um gráfico de dispersão utilizando a primeira e a segunda característica do dataset
sns.scatterplot(x=df[iris.feature_names[0]], y=df[iris.feature_names[1]], hue='cluster', palette='viridis')
plt.title("Clusters formados pelo K-Means")
plt.show()
```

> Observação:
>
> - Esse gráfico permite ver se os clusters formados se aproximam das verdadeiras espécies do dataset.

Passo 9: Comparando os Clusters com as Classes Reais

Para verificar a correspondência entre os clusters e as espécies originais, crie uma tabela cruzada:

```
# Cria uma tabela que compara os clusters formados com as espécies reais
print(pd.crosstab(df['cluster'], df['species']))
```

> Dica para Discussão:
>
> - Analise se os clusters se alinham com as espécies e discuta possíveis discrepâncias com os colegas.

Conclusão e Próximos Passos

O que aprendemos hoje:

- **Aprendizado Supervisionado:**
  - Carregamos dados, treinamos um modelo de árvore de decisão e avaliamos sua performance.
- **Aprendizado Não Supervisionado:**
  - Aplicamos o K-Means para agrupar dados e comparamos os resultados com as classes reais.