Argos Maia - Paulo José

As questões respondidas estarão no final desse notebook

Bibliotecas necessárias para rodar o programa

```
from scipy.io import arff
import pandas as pd
import numpy as np
import warnings
import statistics as sts
import seaborn as sns
import platform
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectKBest, mutual_info_classif, RFE, SelectFromModel, VarianceThreshc
from sklearn.neighbors import LocalOutlierFactor
from sklearn.neighbors import KNeighborsClassifier as knn
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.inspection import permutation_importance
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import MinMaxScaler, LabelEncoder, StandardScaler, KBinsDiscretizer, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.metrics import accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import NeighborhoodComponentsAnalysis as nca
```

Inicializa o modulo base necessário para se vizualizar os dados

Foi mudado para que o programa rode em um SO e dentro de um ambiente Jupyter

```
# Verifica se está executando no Google Colab
if 'google.colab' in str(get_ipython()):
    caminho = "/content/sample_data/arritmias_treino.arff"
else:
    sistema_operacional = platform.system()

    # Verifica o sistema operacional
    if sistema_operacional == "Windows":
        caminho = r"C:\Users\seu_usuario\Área de Trabalho\Trabalho\arritmias_treino.arff"
    elif sistema_operacional == "Linux":
        caminho = "/home/seu_usuario/Área de Trabalho/Trabalho/arritmias_treino.arff"
    elif sistema_operacional == "Darwin":
        caminho = "/Users/seu_usuario/Área de Trabalho/Trabalho/arritmias_treino.arff"
    else:
        print("Sistema operacional não suportado.")

# Carrega os dados do arquivo
dados, meta = arff.loadarff(caminho)
base = pd.DataFrame(dados)
base
```

Descrição de alguns valores da base

```
base.describe()
```

Checa-se quais bases tem valores nulos

```
base.isnull().values.any()
#checa onde se há colunas com valores vazios no dataframe
vazios = base.columns[base.isnull().any()].tolist()
print(vazios)
```

```
['T', 'P', 'QRST', 'J']
```

Portanto, aplicaremos a limpeza desses dados

Visualização de outliers, missing values, discretização e seleção de variáveis

```
outliers = [40, 121, 296]
for outlier in outliers:
    print(base.loc[outlier, :],"\n")
```

```
Age              1.0
Sex             b'0'
Height         100.0
Weight          10.0
QRS_duration    80.0
                ...
Amp_V6_7         0.8
Amp_V6_8         0.9
Amp_V6_9        -1.8
Amp_V6_10        5.2
class           b'5'
Name: 40, Length: 279, dtype: object

Age              1.0
Sex             b'1'
Height         100.0
Weight           6.0
QRS_duration    85.0
                ...
Amp_V6_7         1.3
Amp_V6_8         0.7
Amp_V6_9         2.7
Amp_V6_10        5.5
class           b'5'
Name: 121, Length: 279, dtype: object

Age              0.0
Sex             b'0'
Height         100.0
Weight          10.0
QRS_duration    83.0
                ...
Amp_V6_7         0.5
Amp_V6_8         2.5
Amp_V6_9       -11.8
Amp_V6_10        1.7
class           b'5'
Name: 296, Length: 279, dtype: object
```

```
# Alterar a altura nas posições específicas
base.at[40, 'Height'] = 100.0
base.at[121, 'Height'] = 100.0
base.at[296, 'Height'] = 100.0

# Verificar as alterações
print(base.loc[40])
print(base.loc[121])
print(base.loc[296])
```

```
Age              1.0
Sex             b'0'
Height          100.0
Weight           10.0
QRS_duration     80.0
                 ...
Amp_V6_7          0.8
Amp_V6_8          0.9
Amp_V6_9         -1.8
Amp_V6_10         5.2
class            b'5'
Name: 40, Length: 280, dtype: object
Age              1.0
Sex             b'1'
Height          100.0
Weight           6.0
QRS_duration     85.0
                 ...
Amp_V6_7          1.3
Amp_V6_8          0.7
Amp_V6_9          2.7
Amp_V6_10         5.5
class            b'5'
Name: 121, Length: 280, dtype: object
Age              0.0
Sex             b'0'
Height          100.0
Weight           10.0
QRS_duration     83.0
                 ...
Amp_V6_7          0.5
Amp_V6_8          2.5
Amp_V6_9        -11.8
Amp_V6_10         1.7
class            b'5'
Name: 296, Length: 280, dtype: object
```

```
def analise_missing(base):
    # Número de NaN em cada coluna
    na_count = base.isna().sum()

    # Porcentagem de valores faltantes em cada coluna
    na_percentage = na_count / len(base) * 100

    # DataFrame com as informações sobre valores faltantes
    missing_data = pd.DataFrame({'Número de NaN': na_count, 'Porcentagem de faltantes': na_percentage})
    missing_data.sort_values(by='Número de NaN', ascending=False, inplace=True)

    # Exibir o resultado
    print(missing_data)

# Chamando a função para analisar os valores faltantes na base
analise_missing(base)
```

```
           Número de NaN  Porcentagem de faltantes
J                    334                83.084577
P                     22                 5.472637
T                      8                 1.990050
QRST                   1                 0.248756
```

```
Amp_AVR_4              0                 0.000000
...                  ...                      ...
V2_3                   0                 0.000000
V2_4                   0                 0.000000
V2_5                   0                 0.000000
V2_6                   0                 0.000000
class                  0                 0.000000

[280 rows x 2 columns]
```

Apagaremos a coluna J pois ela é vazia e não tem utilidades para o programa

```
base = base.drop("J", axis=1)
```

```
# Get the number of missing values for each column
missing_values = base.isna().sum()

# Get the frequency of each value in each column
frequencia = base.value_counts()

# Replace the missing values with the frequency of each value
base.fillna(frequencia, inplace=True)

base
```

```
# Verificar valores únicos nas colunas
unique_values_P = base['P'].unique()
unique_values_T = base['T'].unique()
unique_values_QRST = base['QRST'].unique()

# Calcular frequência de cada elemento nas colunas
frequency_P = base['P'].value_counts()
frequency_T = base['T'].value_counts()
frequency_QRST = base['QRST'].value_counts()

# Preencher valores faltantes com a frequência de cada elemento
base['P'].fillna(frequency_P, inplace=True)
base['T'].fillna(frequency_T, inplace=True)
base['QRST'].fillna(frequency_QRST, inplace=True)
```

```
# Verificar valores faltantes nas colunas 'P', 'T' e 'QRST'
missing_values_P = base['P'].isna().sum()
missing_values_T = base['T'].isna().sum()
missing_values_QRST = base['QRST'].isna().sum()

# Exibir a quantidade de valores faltantes
print("Valores faltantes na coluna 'P':", missing_values_P)
print("Valores faltantes na coluna 'T':", missing_values_T)
print("Valores faltantes na coluna 'QRST':", missing_values_QRST)
```

```
Valores faltantes na coluna 'P': 20
Valores faltantes na coluna 'T': 6
Valores faltantes na coluna 'QRST': 1
```

```
for i, valor in base["P"].items():
    if pd.isnull(valor):
        print(f"NaN encontrado na posição {i}")
```

```
NaN encontrado na posição 86
NaN encontrado na posição 88
NaN encontrado na posição 96
NaN encontrado na posição 113
NaN encontrado na posição 154
NaN encontrado na posição 157
NaN encontrado na posição 173
NaN encontrado na posição 184
NaN encontrado na posição 197
NaN encontrado na posição 199
NaN encontrado na posição 223
NaN encontrado na posição 233
NaN encontrado na posição 259
NaN encontrado na posição 264
NaN encontrado na posição 278
NaN encontrado na posição 280
NaN encontrado na posição 288
NaN encontrado na posição 290
NaN encontrado na posição 330
NaN encontrado na posição 400
```

```
#Faz a mediana de valores P
mediana_P = base["P"].dropna().median()
print(mediana_P)
```

```
55.5
```

```
base["P"]
```

```
0      -5.0
1      75.0
2       8.0
3      78.0
4      49.0
       ...
397    34.0
398    49.0
399    59.0
400     NaN
401    63.0
Name: P, Length: 402, dtype: float64
```

```
#insere as medianas onde há valores nulos
base["P"].fillna(mediana_P, inplace=True)
base["P"]
```

```
0      -5.0
1      75.0
2       8.0
3      78.0
4      49.0
       ...
397    34.0
398    49.0
399    59.0
400    55.5
401    63.0
Name: P, Length: 402, dtype: float64
```

```
for i, valor in base["QRST"].items():
    if pd.isnull(valor):
      print(f"NaN encontrado na posição {i}")
```

```
#Faz a mediana de valores QRST
mediana_QRST = base["QRST"].dropna().median()
print(mediana_QRST,"\n\n")
```

```
#insere as medianas onde há valores nulos
base["QRST"].fillna(mediana_QRST, inplace=True)
base["QRST"]
```

```
40.0


0      20.0
1      65.0
2      51.0
3      66.0
4      26.0
       ...
397    13.0
398    38.0
399    48.0
400    81.0
401    20.0
Name: QRST, Length: 402, dtype: float64
```

```
mediana = base["T"].dropna().median()
base["T"].fillna(mediana, inplace=True)
base["T"]
```

```
0       11.0
1       49.0
2        7.0
3       69.0
4       37.0
       ...
397    136.0
398     33.0
399     61.0
400   -132.0
401      0.0
Name: T, Length: 402, dtype: float64
```

```
if 'J' in base.columns:
    if base['J'].empty:
        print("A coluna 'J' existe, mas está vazia.")
    else:
        print("A coluna 'J' existe e contém valores.")
else:
    print("A coluna 'J' não existe no DataFrame.")
```

```
A coluna 'J' não existe no DataFrame.
```

```
print(f"Max: {base['Height'].max()} cm")
print(f"Min: {base['Height'].min()} cm")
```

```
Max: 188.0 cm
Min: 100.0 cm
```

```
print(f"Max: {base['Weight'].max()}")
print(f"Min: {base['Weight'].min()}")
```

```
Max: 176.0
Min: 6.0
```

```
linha_weight_6 = base.loc[base['Weight'] == 6.0]
print(linha_weight_6)
```

```
      Age   Sex  Height  Weight  QRS_duration    P-R    Q-T  T_interval  \
121  1.0  b'1'   100.0     6.0          85.0  165.0  237.0       150.0
```

```
     P_interval   QRS  ...  Amp_V6_2  Amp_V6_3  Amp_V6_4  Amp_V6_5  Amp_V6_6  \
121       106.0  88.0  ...       0.0       5.0      -4.6       0.0       0.0

     Amp_V6_7  Amp_V6_8  Amp_V6_9  Amp_V6_10  class
121       1.3       0.7       2.7        5.5   b'5'

[1 rows x 279 columns]
```

```
# Verificar se o usuário quer imprimir apenas uma linha ou todas as linhas
print_todas = input("Deseja imprimir todas as linhas? (S/N): ").lower() == "s"

# Verificar se o usuário quer imprimir apenas uma linha
if not print_todas:
    linha_desejada = int(input("Digite o número da linha desejada: "))

# Loop para percorrer as linhas do DataFrame
for i, linha in base.iterrows():
    # Verificar se o usuário quer imprimir apenas uma linha
    if not print_todas and i != linha_desejada:
        continue  # Pular para a próxima iteração do loop

    # Imprimir os valores da idade, altura e peso
    idade = linha["Age"]
    altura = linha["Height"]
    peso = linha["Weight"]
    print(f"Linha {i + 1}: Idade={idade}, Altura={altura}, Peso={peso}")

    # Parar o loop se o usuário quiser imprimir apenas uma linha
    if not print_todas:
        break
```

```
 Deseja imprimir todas as linhas? (S/N): N
 Digite o número da linha desejada: 130
 Linha 131: Idade=48.0, Altura=156.0, Peso=62.0
```

```
base['class'] = base['class'].astype(str)

# Selecionar as colunas desejadas
X = base[['T', 'P', 'QRST']].copy()
y = base['class']

# Preencher valores faltantes com a frequência de cada elemento
X['T'].fillna(X['T'].value_counts(), inplace=True)
X['P'].fillna(X['P'].value_counts(), inplace=True)
X['QRST'].fillna(X['QRST'].value_counts(), inplace=True)

# Dividir o conjunto de dados em treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Criar o modelo de árvore de decisão
dt_classifier = DecisionTreeClassifier()

# Treinar o modelo
dt_classifier.fit(X_train, y_train)

# Fazer previsões
previsoes = dt_classifier.predict(X_test)

# Calcular a acurácia do modelo
acuracia = accuracy_score(y_test, previsoes)
print(f"Acurácia do modelo: {acuracia:.4f}")
```

```
Acurácia do modelo: 0.3827
```

1. a) Descrição dos dados

Age: *Numérico*(razão, continuo) = Mostra a idade dos 402 pacientes presentes na lista

Sex: *Categórico* (binário) **-** Gênero do paciente (0: Masculino, 1: Feminino) = Mostra o sexo do paciente

Height: *Numérico* (razão, continuo) = Mostra a altura dos pacientes

Weight: *Numérico* (razão, continuo) = Mostra o peso dos pacientes

QRS_duration: *Numérico* (razão, continuo) = Duração do complexo QRS

P-R: *Numérico* (razão, continuo) = intervalo P-R

Q-T: *Numérico* (razão, continuo) = Intervalo Q-T

T_interval: *Numérico* (razão, continuo) = intervalo T

P_interval: *Numérico* (razão, continuo) = intervalo P

QRS: *Numérico* (razão, continuo) = valor do complexo de QRS

Categoria numérica (razão, continuo) de J até Amp_V6_10: Representando valores de amplitude de canais diferentes de ESG

Class: Categórico (nominal, dicreta) = Classe de dados de arritmia cardíaca de 1 até 15

Abaixo tem a códificação de cada dado

```
# Verificar se o usuário deseja imprimir todas as colunas ou apenas uma coluna específica
imprime_tudo = input("Deseja imprimir todas as colunas? (S/N): ").lower() == "s"

# Verificar se o usuário deseja imprimir apenas uma coluna específica
if not imprime_tudo:
    coluna_escolhida = input("Digite o nome da coluna desejada: ")

# Percorrer as colunas da base de dados
for coluna in base.columns:
    # Verificar se o usuário deseja imprimir apenas uma coluna específica
    if not imprime_tudo and coluna != coluna_escolhida:
        continue  # Pular para a próxima iteração do loop

    nome = coluna
    tipo = base[coluna].dtype

    # Verificar o tipo de dado da coluna
    if tipo == "object":
        # Atributo categórico (nominal)
        escala = "nominal"

        # Verificar a cardinalidade
        cardinalidade = "discreta" if base[coluna].nunique() <= 10 else "contínua"

    elif tipo in ["int64", "float64"]:
        # Atributo numérico (razão)
        escala = "razão"

        # Verificar a cardinalidade
        cardinalidade = "discreta" if base[coluna].nunique() <= 10 else "contínua"

        # Verificar se é uma coluna binária
        if base[coluna].nunique() == 2:
            cardinalidade = "binária"

    else:
        # Tipo de dado não reconhecido
```

```
            escala = "desconhecida"
            cardinalidade = "desconhecida"

        # Imprimir a descrição do atributo
        print("Atributo:", nome)
        print("Tipo:", tipo)
        print("Escala:", escala)
        print("Cardinalidade:", cardinalidade)
        print("---------------------------------")
```

```
 Deseja imprimir todas as colunas? (S/N): S
 Atributo: Age
 Tipo: float64
 Escala: razão
 Cardinalidade: contínua
 ---------------------------------
 Atributo: Sex
 Tipo: object
 Escala: nominal
 Cardinalidade: discreta
 ---------------------------------
 Atributo: Height
 Tipo: float64
 Escala: razão
 Cardinalidade: contínua
 ---------------------------------
 Atributo: Weight
 Tipo: float64
 Escala: razão
 Cardinalidade: contínua
 ---------------------------------
 Atributo: QRS_duration
 Tipo: float64
 Escala: razão
 Cardinalidade: contínua
 ---------------------------------
 Atributo: P-R
 Tipo: float64
 Escala: razão
 Cardinalidade: contínua
 ---------------------------------
 Atributo: Q-T
 Tipo: float64
 Escala: razão
 Cardinalidade: contínua
 ---------------------------------
 Atributo: T_interval
 Tipo: float64
 Escala: razão
 Cardinalidade: contínua
 ---------------------------------
 Atributo: P_interval
 Tipo: float64
 Escala: razão
 Cardinalidade: contínua
 ---------------------------------
 Atributo: QRS
 Tipo: float64
 Escala: razão
 Cardinalidade: contínua
 ---------------------------------
 Atributo: T
 Tipo: float64
 Escala: razão
 Cardinalidade: contínua
 ---------------------------------
 Atributo: P
 Tipo: float64
```

Questão 1. b) Frequência dos dados

Optamos pela codificação desses dados pois seria mais pratico para que se possa analisar o conteúdo preenchido no dataset, ao se descrever os dados de cada atributo presente segundo sua ***frequência, max, min e dp***, analisa-se cada coluna presente

```
# Verificar se o usuário deseja imprimir todas as colunas ou apenas uma coluna específica
imprime_tudo = input("Deseja imprimir todas as colunas? (S/N): ").lower() == "s"

# Verificar se o usuário deseja imprimir apenas uma coluna específica
if not imprime_tudo:
    coluna_escolhida = input("Digite o nome da coluna desejada: ")

# Percorrer as colunas da base de dados
for coluna in base.columns:
    # Verificar se o usuário deseja imprimir apenas uma coluna específica
    if not imprime_tudo and coluna != coluna_escolhida:
        continue  # Pular para a próxima iteração do loop

    nome = coluna
    tipo = base[coluna].dtype

    # Verificar o tipo de dado da coluna
    if tipo == "object":
        # Atributo categórico (nominal)
        escala = "nominal"

        # Verificar a cardinalidade
        cardinalidade = "discreta" if base[coluna].nunique() <= 10 else "contínua"

        # Descrever a frequência dos valores
        frequencia = base[coluna].value_counts()

        # Imprimir a frequência
        print("Frequência dos valores:")
        print(frequencia)

    elif tipo in ["int64", "float64"]:
        # Atributo numérico (razão)
        escala = "razão"

        # Verificar a cardinalidade
        cardinalidade = "discreta" if base[coluna].nunique() <= 10 else "contínua"

        # Descrever estatísticas descritivas
        minimo = base[coluna].min()
        maximo = base[coluna].max()
        desvio_padrao = base[coluna].std()

        # Imprimir estatísticas descritivas
        print(f"Valor mínimo: {minimo}")
        print(f"Valor máximo: {maximo}")
        print(f"Desvio padrão: {desvio_padrao:.2f}")

    else:
        # Tipo de dado não reconhecido
        escala = "desconhecida"
        cardinalidade = "desconhecida"

    # Imprimir a descrição do atributo
    print("Atributo:", nome)
    print("Tipo:", tipo)
    print("Escala:", escala)
```

```
print("Cardinalidade:", cardinalidade)
print("--------------------------------")
```

```
Desvio padrão: 0.00
Atributo: Amp_V6_6
Tipo: float64
Escala: razão
Cardinalidade: discreta
--------------------------------
Valor mínimo: -0.8
Valor máximo: 2.4
Desvio padrão: 0.35
Atributo: Amp_V6_7
Tipo: float64
Escala: razão
Cardinalidade: contínua
--------------------------------
Valor mínimo: -6.0
Valor máximo: 6.0
Desvio padrão: 1.42
Atributo: Amp_V6_8
Tipo: float64
Escala: razão
Cardinalidade: contínua
--------------------------------
Valor mínimo: -36.6
Valor máximo: 88.8
Desvio padrão: 13.41
Atributo: Amp_V6_9
Tipo: float64
Escala: razão
Cardinalidade: contínua
--------------------------------
Valor mínimo: -38.6
Valor máximo: 97.6
Desvio padrão: 17.45
Atributo: Amp_V6_10
Tipo: float64
Escala: razão
Cardinalidade: contínua
--------------------------------
Frequência dos valores:
b'1'     225
b'2'      41
b'10'     39
b'6'      20
b'16'     20
b'5'      13
b'3'      13
b'4'      13
b'9'       8
b'15'      5
b'14'      2
b'7'       2
b'8'       1
Name: class, dtype: int64
Atributo: class
Tipo: object
Escala: nominal
Cardinalidade: contínua
--------------------------------
```

Questão 1.c)

implementando o algoritmo J48 no dataset

Em cima, checamos se os valores discrepantes ainda existem ou se já foram substituídos

```
warnings.filterwarnings("ignore", category=pd.errors.PerformanceWarning)
warnings.filterwarnings("ignore", category=UserWarning)

# Selecionar as colunas desejadas
X = base.drop('class', axis=1)
y = base['class']

# Dividir o conjunto de dados em treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Criar o modelo de árvore de decisão (J48)
dt_classifier = DecisionTreeClassifier(criterion='entropy')

# Treinar o modelo
dt_classifier.fit(X_train, y_train)

# Fazer previsões
previsoes = dt_classifier.predict(X_test)

# Calcular a acurácia do modelo
acuracia = accuracy_score(y_test, previsoes)
print(f"Acurácia do modelo: {acuracia:.4f}")
```

```
Acurácia do modelo: 0.6667
```

Questão 2

Modelo | 1º atributo | 2º atributo | N atributo

Gain |

ReliefF |

InfoGain |

CFS |

O código abaixo vai demorar para executar

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_selection import SelectKBest, mutual_info_classif
from sklearn.metrics import accuracy_score

# Separar atributos e classe
X = base.drop('class', axis=1)
y = base['class']

# Dividir o conjunto de dados em treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Definir o modelo de árvore de decisão para avaliação de acurácia
dt_classifier = DecisionTreeClassifier()

# Definir os métodos de seleção de variáveis
methods = ['gain', 'relieff', 'infogain', 'cfs']

# Dicionário para armazenar os resultados de acurácia
results = {}

# Loop pelos métodos de seleção de variáveis
```

```
for method in methods:
    # Selecionar as K melhores variáveis utilizando o método específico
    if method == 'gain':
        selector = SelectKBest(score_func=mutual_info_classif, k=1)
    elif method == 'infogain':
        selector = SelectKBest(score_func=mutual_info_classif, k=1)
    elif method == 'cfs':
        selector = SelectKBest(score_func=mutual_info_classif, k=1)

    # Aplicar a seleção de variáveis no conjunto de treinamento
    X_train_selected = selector.fit_transform(X_train, y_train)

    # Obter o índice do atributo selecionado
    selected_index = selector.get_support(indices=True)[0]

    # Obter o nome do atributo selecionado
    selected_attribute = X.columns[selected_index]

    # Treinar o modelo de árvore de decisão com a variável selecionada
    dt_classifier.fit(X_train_selected, y_train)

    # Aplicar a seleção de variáveis no conjunto de teste
    X_test_selected = X_test.iloc[:, selected_index].values.reshape(-1, 1)

    # Fazer previsões com o modelo treinado
    previsoes = dt_classifier.predict(X_test_selected)

    # Calcular a acurácia para o atributo selecionado
    accuracy = accuracy_score(y_test, previsoes)

    # Armazenar o resultado de acurácia no dicionário de resultados
    results[selected_attribute] = accuracy

# Imprimir os resultados
for attribute, accuracy in results.items():
    print(f"Acurácia para o atributo {attribute}: {accuracy:.4f}")

# Identificar o atributo com maior acurácia
best_attribute = max(results, key=results.get)
print(f"Atributo com maior acurácia: {best_attribute}")
```

```
Acurácia para o atributo Amp_AVR_8: 0.5309
Acurácia para o atributo V3_3: 0.5926
Atributo com maior acurácia: V3_3
```