Atividade 4 - Aprendizado de Máquina

AUTHOR
Rennan Dalla Guimarães

1 Classificação do Breast Cancer Wisconsin

Este relatório utiliza o *Breast Cancer Wisconsin (Diagnostic)* (569 instâncias, 30 atributos) disponível em `sklearn.datasets`.
Dois experimentos são conduzidos:

- **Experimento A – Dados brutos**
- **Experimento B – Pré-processamento** (remoção de outliers, seleção de features, PCA, padronização)

Todos os modelos são avaliados com validação cruzada estratificada (k-fold = 10) e múltiplas combinações de hiperparâmetros.

1.1 Configuração

```
import numpy as np
import pandas as pd

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import StratifiedKFold, cross_val_score, (
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest, f_classif

from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier

RNG    = 42
FOLDS  = 10
cv     = StratifiedKFold(n_splits=FOLDS, shuffle=True, random_state=RN(

data   = load_breast_cancer()
X, y   = data.data, data.target
print(f"Instâncias: {X.shape[0]}, Atributos: {X.shape[1]}")
```

```
Instâncias: 569, Atributos: 30
```

1.2 Experimento A – Dados brutos

```
models = {
    "KNN": KNeighborsClassifier(),
    "Naive Bayes": GaussianNB(),
    "Árvore Decisão": DecisionTreeClassifier(random_state=RNG)
}
```

```
param_grid = {
    "KNN": {
        "n_neighbors": [3,5,7,9],
        "weights": ["uniform","distance"],
        "metric": ["minkowski","euclidean","manhattan"]
    },
    "Naive Bayes": {  # GaussianNB não possui hiperparâmetros relevant
    },
    "Árvore Decisão": {
        "criterion": ["gini","entropy","log_loss"],
        "max_depth": [None, 3, 5, 10],
        "min_samples_split": [2, 4, 6]
    }
}

results_raw = []

for name, model in models.items():
    if param_grid[name]:
        grid = GridSearchCV(model, param_grid[name], cv=cv, n_jobs=-1)
        grid.fit(X, y)
        best_model = grid.best_estimator_
        acc = grid.best_score_
        params = grid.best_params_
    else:
        acc = cross_val_score(model, X, y, cv=cv, n_jobs=-1).mean()
        params = {}
    results_raw.append({"Modelo": name, "Acurácia média": acc, "Melhor

pd.DataFrame(results_raw).sort_values("Acurácia média", ascending=Fals
```

	Modelo	Acurácia média	Melhores parâmetros
0	KNN	0.938534	{'metric': 'manhattan', 'n_neighbors': 3, 'wei...
1	Naive Bayes	0.936811	{}
2	Árvore Decisão	0.934994	{'criterion': 'entropy', 'max_depth': None, 'm...

1.3 Experimento B – Dados pré-processados

- **Remoção de outliers** – RobustScaler
- **Seleção de features** – `SelectKBest` (teste F)
- **Redução de dimensionalidade** – PCA (95% da variância)

```
preprocess_pipe = Pipeline([
    ("scale", RobustScaler()),
    ("select", SelectKBest(score_func=f_classif, k=20)),
    ("pca",    PCA(n_components=0.95, random_state=RNG))
])

results_pp = []

for name, model in models.items():
    pipe = Pipeline([
```

```
        ("prep", preprocess_pipe),
        ("clf",  model)
    ])

    if param_grid[name]:
        # Adaptar nomes dos hiperparâmetros para o último estágio ('cl
        tuned_grid = {f"clf__{k}": v for k, v in param_grid[name].item
        grid = GridSearchCV(pipe, tuned_grid, cv=cv, n_jobs=-1)
        grid.fit(X, y)
        best_model = grid.best_estimator_
        acc = grid.best_score_
        params = grid.best_params_
    else:
        acc = cross_val_score(pipe, X, y, cv=cv, n_jobs=-1).mean()
        params = {}
    results_pp.append({"Modelo": name, "Acurácia média": acc, "Melhore

pd.DataFrame(results_pp).sort_values("Acurácia média", ascending=False
```

	Modelo	Acurácia média	Melhores parâmetros
0	KNN	0.963127	{'clf__metric': 'minkowski', 'clf__n_neighbors...
2	Árvore Decisão	0.943797	{'clf__criterion': 'gini', 'clf__max_depth': N...
1	Naive Bayes	0.931548	{}

2 Algoritmos de Árvores de Decisão – Principais Diferenças

2.1 Algoritmo de Hunt (1966) [1]

Hunt, Marin e Stone propuseram em 1966 um procedimento genérico de construção de árvores recursivas fundamentado no ciclo *dividir-testar-parar*. O trabalho não prescreveu critérios específicos de impureza, servindo como **arcabouço inicial** para que técnicas posteriores, mais especializadas, pudessem ser desenvolvidas.

2.2 ID3 (Quinlan, 1986) [2]

O **ID3** introduziu o uso do **Ganho de Informação** (entropia) como métrica para selecionar o atributo de divisão. Embora tenha representado grande avanço, possui duas limitações principais: (i) sensibilidade a atributos com muitos valores distintos e (ii) ausência de um mecanismo sistemático de poda, o que pode levar a sobre-ajuste. Ademais, atributos contínuos requerem discretização prévia.

2.3 C4.5 (Quinlan, 1993) [3]

O sucessor natural do ID3 trouxe quatro aperfeiçoamentos:

- **Gain Ratio**, que normaliza o Ganho de Informação, reduzindo o viés para atributos multi-válidos;

- **Tratamento direto de variáveis contínuas**, selecionando limiares ótimos durante a divisão;
- **Poda com estimativa de erro de generalização**, diminuindo o risco de sobre-ajuste;
- Capacidade de converter a árvore em um conjunto de **regras proposicionais**, facilitando a interpretação.

2.4 J4.8 (Witten & Frank, 1996) [7]

Popularizado no ambiente WEKA, o **J4.8** é essencialmente uma **implementação de código aberto do C4.5 em Java**. O algoritmo subjacente mantém a mesma lógica; a denominação distinta está associada apenas a questões de licenciamento de software.

2.5 C5.0 (Quinlan, 2000) [4]

O **C5.0** pode ser visto como uma evolução de engenharia sobre o C4.5:

- Aumento expressivo de velocidade e redução de uso de memória;
- Procedimento de **winnowing** para eliminar atributos irrelevantes antes da indução;
- Suporte a **pesos de instância** e a um componente de **boosting interno**.

Conceitualmente permanece alinhado ao C4.5, mas é mais eficiente em contextos de produção.

2.6 CART (Breiman et al., 1984) [5]

O **CART** difere substancialmente da linha ID3–C5.0:

- Utiliza **Índice de Gini** (classificação) ou **Soma dos Erros Quadráticos** (regressão) como critério de divisão;
- Gera **somente divisões binárias**, o que simplifica a análise da árvore resultante;
- Aplica **poda custo-complexidade (CCP)**, removendo ramos cujo benefício estatístico não compensa o aumento de complexidade;
- Possui suporte nativo a **modelagem de regressão**, além da classificação.

2.7 Random Forest (Breiman, 2001) [6]

A **Random Forest** estabelece um **conjunto (ensemble) de diversas árvores CART** combinadas por votação. Dois mecanismos geram diversidade entre as árvores: (i) *bootstrap* de amostras e (ii) sub-amostragem aleatória de atributos em cada divisão. Essa abordagem reduz significativamente a variância do modelo, fornecendo alto desempenho com necessidade mínima de ajuste fino.

2.8 Considerações Comparativas

- A família **ID3 → C4.5 → C5.0** progride em capacidade de lidar com atributos contínuos, estratégias de poda e eficiência computacional.

- O **CART** introduz árvores estritamente binárias e um esquema de poda fundamentado em teoria estatística robusta, tornando-o a base para diversos algoritmos de *ensembles*.
- A **Random Forest** demonstra que a agregação de modelos independentes é um caminho eficaz para aumentar a capacidade de generalização, sem sacrificar interpretabilidade em nível de atributo.
