Atividade 6 - Aprendizado de Máquina

AUTHOR
Rennan Dalla Guimarães

Exercício 1

1. Carregar e preparar os dados

```
import numpy as np
import pandas as pd
from pathlib import Path

from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler, Funct
from sklearn.model_selection import StratifiedKFold, GridSearchCV
from sklearn.metrics import make_scorer, mean_squared_error, r2_score

from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import KFold

csv_path = Path("forestfires.csv")
if not csv_path.exists():
    raise FileNotFoundError(
        "Coloque o dataset CSV (forestfires.csv) na mesma pasta deste 
        "ou ajuste o caminho em 'csv_path'."
    )

df = pd.read_csv(csv_path)

# A variável-alvo
y_raw = df["area"].values
# Apliquei log1p porque 'area' é extremamente assimétrica
y = np.log1p(y_raw)

# Atributos
X = df.drop(columns=["area"])

numeric_cols = X.select_dtypes(include=["int64", "float64"]).columns.t
categorical_cols = ["month", "day"]

preprocess = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), numeric_cols),
        ("cat", OneHotEncoder(handle_unknown="ignore"), categorical_co
    ]
)
```

2. Configurar validação cruzada e métrica para avaliar resultado

```
cv = KFold(n_splits=5, shuffle=True, random_state=42) # Usei o KFold
# porque tive problemas com StratifiedKFold: ValueError: Supported tar
# types are: ('binary', 'multiclass'). Got 'continuous' instead.

rmse = make_scorer(
    lambda yt, yp: np.sqrt(mean_squared_error(yt, yp)), greater_is_bet
)
```

3. Definir modelos e grades de hiperparâmetros

```
pipelines = {
    "KNN": Pipeline(
        steps=[
            ("prep", preprocess),
            ("reg", KNeighborsRegressor()),
        ]
    ),
    "DecisionTree": Pipeline(
        steps=[
            ("prep", preprocess),
            ("reg", DecisionTreeRegressor(random_state=42)),
        ]
    ),
    "SVR": Pipeline(
        steps=[
            ("prep", preprocess),
            ("reg", SVR()),
        ]
    ),
    "MLP": Pipeline(
        steps=[
            ("prep", preprocess),
            ("reg", MLPRegressor(max_iter=1000, random_state=42, early
        ]
    ),
}

param_grids = {
    "KNN": {
        "reg__n_neighbors": [3, 5, 7, 11],
        "reg__weights": ["uniform", "distance"],
        "reg__p": [1, 2],  # Manhattan ou Euclidiana
    },
    "DecisionTree": {
        "reg__max_depth": [None, 4, 6, 8, 12],
        "reg__min_samples_leaf": [1, 3, 5, 10],
        "reg__criterion": ["squared_error", "friedman_mse"],
    },
    "SVR": {
        "reg__kernel": ["rbf", "poly", "sigmoid"],
        "reg__C": [1, 10, 100],
```

```
        "reg__gamma": ["scale", "auto"],
        "reg__epsilon": [0.1, 0.5, 1.0],
    },
    "MLP": {
        "reg__hidden_layer_sizes": [(50,), (100,), (50, 50)],
        "reg__activation": ["relu", "tanh"],
        "reg__alpha": [1e-4, 1e-3],
        "reg__learning_rate_init": [0.001, 0.01],
    },
}
```

---

4. Treinar com Grid Search + CV

```
results = []

for name, pipe in pipelines.items():
    print(f"\n🔍 {name} — grid-search...")
    grid = GridSearchCV(
        estimator=pipe,
        param_grid=param_grids[name],
        scoring=rmse,  # negativo (quanto mais próximo de 0, melhor)
        cv=cv,
        n_jobs=-1,
        verbose=0,
    )
    grid.fit(X, y)
    best_rmse = -grid.best_score_  # invertendo sinal
    best_r2 = grid.cv_results_[
        "mean_test_score"
    ]  # usamos RMSE, mas podemos recalcular R^2
    # Recalcula R^2 para o melhor estimador
    r2_val = r2_score(y, grid.best_estimator_.predict(X))
    results.append(
        {
            "Modelo": name,
            "Melhor_RMSE_CV": round(best_rmse, 3),
            "R2_no_treino": round(r2_val, 3),
            "Best_Params": grid.best_params_,
        }
    )
    print(f"→ RMSE (CV): {best_rmse:.3f}")
```

```
🔍 KNN — grid-search...
→ RMSE (CV): 1.437

🔍 DecisionTree — grid-search...
→ RMSE (CV): 1.455

🔍 SVR — grid-search...
→ RMSE (CV): 1.392

🔍 MLP — grid-search...
→ RMSE (CV): 1.427
```

---

5. Resumo dos modelos

```
print("\n===== Resumo =====")
res_df = pd.DataFrame(results).set_index("Modelo")
print(res_df)
```

```
===== Resumo =====
              Melhor_RMSE_CV  R2_no_treino  \
Modelo
KNN                    1.437         0.993
DecisionTree           1.455         0.093
SVR                    1.392         0.129
MLP                    1.427         0.017

                                                  Best_Params
Modelo
KNN           {'reg__n_neighbors': 11, 'reg__p': 2, 'reg__we...
DecisionTree  {'reg__criterion': 'squared_error', 'reg__max_...
SVR           {'reg__C': 1, 'reg__epsilon': 1.0, 'reg__gamma...
MLP           {'reg__activation': 'tanh', 'reg__alpha': 0.00...
```

6. Análise do resultado

Conclusão: usando “menor RMSE médio na validação cruzada” como critério, o Support Vector Regressor leva a melhor, seguido de perto pelo KNN.

Exercício 2

Aspecto	Regressão “clássica”	Séries temporais
Independência das amostras	Assume i.i.d. (independentes e identicamente distribuídas); ordem irrelevante.	Observações autocorrelacionadas; ordem **crucial**.
Divisão treino-teste	Aleatória / estratificada.	Respeita cronologia (treino → valida → teste).
Validação cruzada	`KFold`, `ShuffleSplit`.	`TimeSeriesSplit` (walk-forward).
Modelos típicos	KNN, Árvore, SVM, MLP, Regressão Linear.	ARIMA/SARIMA, Prophet, LSTM, Transformer-TS.
Features	Variáveis explicativas arbitrárias.	Lags, tendências, sazonalidade; variáveis exógenas defasadas.

Resumo — Regressão tradicional trata cada linha como independente, enquanto predição de séries temporais considera dependências ao longo do eixo temporal e exige métodos, divisões e validações próprias.