| FT04 |
|---|
| **Curso**: UFCD **10810** |
| **UFCD/Módulo/Temática:** UFCD 10810 |
| **Ação: 10810_1L Fundamentos do desenvolvimento de modelos analíticos em Python** |
| **Formador/a**: Sandra Liliana Meira de Oliveira |
| **Nome do Formando/a**: |

**Exercício 01:**

Consideta o ficheiro train.csv, que contém o famoso **dataset do Titanic**, com informações de passageiros e se sobreviveram ou não. Eis um resumo completo:

| Coluna | Tipo | Descrição |
|---|---|---|
| PassengerId | int64 | Identificador único do passageiro |
| Survived | int64 | Variável alvo: 1 = Sobreviveu, 0 = Não sobreviveu |
| Pclass | int64 | Classe do bilhete (1ª, 2ª, 3ª) |
| Name | object | Nome completo do passageiro |
| Sex | object | Sexo (masculino/feminino) |
| Age | float64 | Idade |
| SibSp | int64 | Nº de irmãos/cônjuges a bordo |
| Parch | int64 | Nº de pais/filhos a bordo |
| Ticket | object | Número do bilhete |
| Fare | float64 | Valor pago pelo bilhete |
| Cabin | object | Número da cabine (muitos valores em falta) |
| Embarked | object | Porto de embarque: C = Cherbourg, Q = Queenstown, S = Southampton |

**Este dataset é ideal para aplicar classificação supervisionada, pois a variável Survived é binária (0 ou 1).**

**No código seguinte fazemos:**

1. Pré-processamento
2. Detecção de outliers
3. Codificação
4. Escalonamento
5. Treino e avaliação do modelo com Random Forest
6. Validação cruzada
7. Ajuste de hiperparâmetros com GridSearchCV
8. Testes adicionais: Curva ROC/AUC e comparação com Decision Tree

Modelos aplicados:

1. **Random Forest Classifier**
    - É o modelo principal que está a ser treinado.
    - Usado para prever uma variável alvo categórica (como Survived, 0 ou 1).
    - Avaliado com:
        - classification_report() (Precision, Recall, F1-score)
        - confusion_matrix()
        - Validação cruzada StratifiedKFold
        - GridSearchCV para otimizar hiperparâmetros
        - Se binário: cálculo da Curva ROC e AUC
2. **Decision Tree Classifier**
    - Serve como modelo de comparação.
    - Aplicado com profundidade máxima de 5.

**Reproduz e analisa o seguinte código:**

```
# classificacao_supervisionada.py

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score,
StratifiedKFold, GridSearchCV
from sklearn.preprocessing import OrdinalEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier, IsolationForest
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.impute import SimpleImputer
import matplotlib.pyplot as plt
```

```
import seaborn as sns

# 1. Carregar o dataset
df = pd.read_csv("train.csv")
print("Primeiras linhas do dataset:")
print(df.head())

# 2. Exploração inicial
print("\nResumo estatístico:")
print(df.describe())

print("\nValores ausentes por coluna:")
print(df.isnull().sum())

# 3. Tratamento de valores ausentes (numéricos com mediana, categóricos com
moda)
num_cols = df.select_dtypes(include=np.number).columns
cat_cols = df.select_dtypes(include='object').columns

imputer_num = SimpleImputer(strategy='median')
df[num_cols] = imputer_num.fit_transform(df[num_cols])

imputer_cat = SimpleImputer(strategy='most_frequent')
df[cat_cols] = imputer_cat.fit_transform(df[cat_cols])

# 4. Detecção de outliers com Z-score e remoção
from scipy.stats import zscore
z_scores = np.abs(zscore(df[num_cols]))
df = df[(z_scores < 3).all(axis=1)]

# 5. Codificação Ordinal (exemplo simplificado)
# NOTA: aplicar apenas a colunas que têm ordem (simular no exemplo)
if 'Tamanho' in df.columns:
    encoder = OrdinalEncoder(categories=[['Pequeno', 'Médio', 'Grande']])
    df['Tamanho'] = encoder.fit_transform(df[['Tamanho']])

# 6. Codificação One-hot para restantes variáveis categóricas
df = pd.get_dummies(df, columns=cat_cols)

# 7. Separar variáveis independentes e alvo
# Atenção: substitui "target" pelo nome da variável alvo do teu dataset
target = "Survived" if "Survived" in df.columns else df.columns[-1]
X = df.drop(columns=[target])
y = df[target]

# 8. Escalonamento
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
# 9. Divisão treino/teste
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.2, stratify=y, random_state=42)

# 10. Treino de modelo base
modelo = RandomForestClassifier(random_state=42)
modelo.fit(X_train, y_train)

# 11. Avaliação
y_pred = modelo.predict(X_test)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("Matriz de Confusão:")
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d',
cmap='Blues')
plt.title("Matriz de Confusão")
plt.xlabel("Previsto")
plt.ylabel("Real")
plt.show()

# 12. Validação Cruzada com StratifiedKFold
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(modelo, X_scaled, y, cv=cv)
print("\nAcurácias (Stratified K-Fold):", scores)
print("Acurácia média:", scores.mean())

# 13. Otimização de hiperparâmetros com GridSearchCV
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, None],
    'min_samples_split': [2, 5, 10]
}

grid = GridSearchCV(RandomForestClassifier(random_state=42), param_grid,
cv=cv, n_jobs=-1)
grid.fit(X_scaled, y)
print("\nMelhores hiperparâmetros:", grid.best_params_)
print("Melhor score:", grid.best_score_)

# 14. Teste com Curva ROC e AUC (se classificação binária)
from sklearn.metrics import roc_auc_score, roc_curve
if len(np.unique(y)) == 2:
    y_proba = modelo.predict_proba(X_test)[:, 1]
    auc = roc_auc_score(y_test, y_proba)
    print(f"ROC AUC Score: {auc:.4f}")
```

```
    fpr, tpr, thresholds = roc_curve(y_test, y_proba)
    plt.figure()
    plt.plot(fpr, tpr, label=f"AUC = {auc:.2f}")
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Curva ROC')
    plt.legend(loc="lower right")
    plt.show()
else:
    print("Curva ROC apenas aplicável para classificação binária.")

# 15. Comparação com outro modelo: Decision Tree
from sklearn.tree import DecisionTreeClassifier

tree_model = DecisionTreeClassifier(max_depth=5, random_state=42)
tree_model.fit(X_train, y_train)
y_tree_pred = tree_model.predict(X_test)
print("\nRelatório de classificação - Decision Tree:")
print(classification_report(y_test, y_tree_pred))
```

**Exercício 02:**

O ficheiro heart.csv contém dados médicos usados para prever doenças cardíacas. Aqui está o resumo detalhado.

**Colunas e Tipos de Dados:**

| Coluna | Tipo | Descrição resumida |
|---|---|---|
| age | int64 | Idade do paciente |
| sex | int64 | Sexo (1 = masculino, 0 = feminino) |
| cp | int64 | Tipo de dor no peito (4 categorias) |
| trestbps | int64 | Pressão arterial em repouso |
| chol | int64 | Colesterol sérico em mg/dl |
| fbs | int64 | Açúcar no sangue > 120 mg/dl (1 = sim, 0 = não) |
| restecg | int64 | Resultados do ECG em repouso |
| thalach | int64 | Frequência cardíaca máxima atingida |
| exang | int64 | Angina induzida por exercício (1 = sim, 0 = não) |
| oldpeak | float64 | Depressão ST induzida por exercício |

| | | |
|---|---|---|
| slope | int64 | Inclinação do segmento ST |
| ca | int64 | Nº de vasos principais com coloração |
| thal | int64 | 3 = normal, 6 = defeito fixo, 7 = defeito reversível |
| target | int64 | 1 = Doença cardíaca presente, 0 = ausente |

Dados Limpos

- Sem valores nulos – todos os campos estão completos.
- Variáveis numéricas categóricas (como cp, thal, slope, etc.) devem ser tratadas como categóricas codificadas.

Exemplo de Registos

| age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 63 | 1 | 3 | 145 | 233 | 1 | 0 | 150 | 0 | 2.3 | 0 | 0 | 1 | 1 |
| 37 | 1 | 2 | 130 | 250 | 0 | 1 | 187 | 0 | 3.5 | 0 | 0 | 2 | 1 |
| 41 | 0 | 1 | 130 | 204 | 0 | 0 | 172 | 0 | 1.4 | 2 | 0 | 2 | 1 |

Este dataset é ideal para treinar modelos de classificação supervisionada como:

- Regressão logística
- Random Forest
- KNN
- SVM

O script completo para treinar e avaliar modelos de classificação com o ficheiro `heart.csv` encontra-se a seguir.

Este script inclui:

- Escalonamento dos dados
- Treino e avaliação com Random Forest
- Curva ROC e cálculo da AUC
- Validação cruzada (StratifiedKFold)

- Comparação com Decision Tree e Regressão Logística
- Ajuste de hiperparâmetros com GridSearchCV

No script `classificacao_heart.py`, foram aplicados **três modelos supervisionados de classificação**, todos adequados à variável alvo `target` (binária: 0 = sem doença, 1 = com doença cardíaca):

1. **Random Forest Classifier (RandomForestClassifier)**
    - Modelo principal.
    - Utiliza uma floresta de árvores de decisão.
    - Robusto ao overfitting, bom para dados com misto de variáveis numéricas/categóricas.
    - Avaliado com:
        - classification_report()
        - Matriz de confusão (confusion_matrix)
        - Curva ROC e AUC
        - Validação cruzada (StratifiedKFold)
        - Otimização de hiperparâmetros com GridSearchCV
2. **Decision Tree Classifier (DecisionTreeClassifier)**
    - Modelo de comparação.
    - Usado com profundidade máxima de 5 (max_depth=5)
    - Útil pela sua interpretabilidade.
    - Avaliado com classification_report()
3. **Regressão Logística (LogisticRegression)**
    - Modelo clássico para classificação binária.
    - Simples e eficaz com dados bem escalonados.
    - Avaliado com classification_report()

Todos os modelos usam:

- Dados escalonados (StandardScaler)
- Divisão treino/teste estratificada

- Avaliação com métricas padrão da aprendizagem supervisionada

Reproduz e analisa o seguinte código, num jupyter notebook:

```
# classificacao_heart.py

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, StratifiedKFold,
cross_val_score, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix,
roc_auc_score, roc_curve
import matplotlib.pyplot as plt
import seaborn as sns

# 1. Carregar o dataset
df = pd.read_csv("heart.csv")

# 2. Separar variáveis independentes e alvo
X = df.drop(columns=["target"])
y = df["target"]

# 3. Escalonamento
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 4. Divisão treino/teste
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, stratify=y, random_state=42
)

# 5. Treino com Random Forest
modelo = RandomForestClassifier(random_state=42)
modelo.fit(X_train, y_train)

# 6. Avaliação
y_pred = modelo.predict(X_test)
print("Random Forest Report:")
print(classification_report(y_test, y_pred))
print("Matriz de Confusão:")
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d',
cmap='Greens')
plt.title("Matriz de Confusão - Random Forest")
plt.xlabel("Previsto")
plt.ylabel("Real")
```

```
plt.show()

# 7. ROC e AUC
y_proba = modelo.predict_proba(X_test)[:, 1]
auc = roc_auc_score(y_test, y_proba)
fpr, tpr, _ = roc_curve(y_test, y_proba)
plt.plot(fpr, tpr, label=f"AUC = {auc:.2f}")
plt.plot([0, 1], [0, 1], 'k--')
plt.title("Curva ROC - Random Forest")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.grid()
plt.show()
print(f"ROC AUC: {auc:.4f}")

# 8. Validação Cruzada
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(modelo, X_scaled, y, cv=cv)
print("Acurácias (CV):", scores)
print("Acurácia Média (CV):", scores.mean())

# 9. Comparação com Árvore de Decisão
tree_model = DecisionTreeClassifier(max_depth=5, random_state=42)
tree_model.fit(X_train, y_train)
y_tree_pred = tree_model.predict(X_test)
print("Decision Tree Report:")
print(classification_report(y_test, y_tree_pred))

# 10. GridSearchCV para Random Forest
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, None],
    'min_samples_split': [2, 5, 10]
}
grid = GridSearchCV(RandomForestClassifier(random_state=42), param_grid,
cv=cv, n_jobs=-1)
grid.fit(X_scaled, y)
print("Melhores hiperparâmetros:", grid.best_params_)
print("Melhor score:", grid.best_score_)

# 11. Comparação adicional: Regressão Logística
log_model = LogisticRegression(max_iter=1000)
log_model.fit(X_train, y_train)
y_log_pred = log_model.predict(X_test)
print("Logistic Regression Report:")
print(classification_report(y_test, y_log_pred))
```