INFO03
Curso: UFCD **10810**
UFCD/Módulo/Temática: UFCD 10810 - **Fundamentos do desenvolvimento de modelos analíticos em Python**
Ação: 10810_1L
Formador/a: Sandra Liliana Meira de Oliveira
Data:
Nome do Formando/a:

Modelos Supervisionados: Desenvolvimento de Classificadores e Avaliação

1. Fundamentos dos Modelos Supervisionados

Modelos supervisionados partem de um conjunto de dados rotulado, ou seja, onde cada entrada tem um rótulo ou valor alvo associado (X -> y). O objetivo é construir um modelo que **aprenda uma função f: X → y** com capacidade de **generalização**, ou seja, que produza previsões corretas em dados não observados.

Tipos de Tarefas Supervisionadas

- **Classificação**: prever uma classe discreta (ex: spam/não spam, doente/são).
- **Regressão**: prever um valor contínuo (ex: preço de uma casa).

Este documento foca-se na **classificação multiclasse**, mas as ideias são extensíveis a regressão.

2. Ciclo de Desenvolvimento de um Modelo Supervisionado

2.1. Pré-processamento e Exploração de Dados

Objetivos do Pré-processamento

- Reduzir viés e variância.
- Aumentar qualidade e consistência dos dados.
- Facilitar o treino e reduzir tempo de computação.

Técnicas Detalhadas

- **Imputação de valores ausentes**:
  - Numéricos: média, mediana, interpolação.
  - Categóricos: moda ou valor "desconhecido".
- **Remoção de outliers**:
  - Z-score, IQR, ou modelagem robusta.

Outliers são valores que se afastam significativamente da distribuição dos restantes dados. A sua presença pode distorcer análises estatísticas, influenciar o desempenho de modelos de machine learning e gerar conclusões erradas.

Por que remover outliers?

1. **Modelos sensíveis a distâncias** (como k-NN, SVM, Regressão) podem ser fortemente afetados.
2. Podem representar **erros de medição, entrada ou extrações raras**, dependendo do contexto.
3. Em casos justificados, pode ser mais apropriado **corrigir ou suavizar** em vez de remover.

Métodos Comuns para Identificação de Outliers

1. **Z-Score (pontuação padrão)**

- Mede o número de desvios padrão que um ponto está acima ou abaixo da média.
- Fórmula:

$$z = \\ fracx - \mu\sigma$$

- Ponto com |z| > 3 (ou 2.5, conforme o rigor) é considerado outlier.

Exemplo:

```
from scipy.stats import zscore
import pandas as pd

df = pd.DataFrame({'valor': [10, 12, 13, 12, 11, 110]})  # 110 é um outlier
df['zscore'] = zscore(df['valor'])
outliers = df[abs(df['zscore']) > 3]
print(outliers)
```

2. IQR (Intervalo Interquartil)

- Baseia-se nos quartis: Q1 (25%) e Q3 (75%).
- Cálculo:

$$\text{IQR} = Q3 - Q1$$

- Valores fora de:

$$[Q1 - 1.5 \times \text{IQR},\ Q3 + 1.5 \times \text{IQR}]$$

são considerados outliers.

Exemplo:

```
Q1 = df['valor'].quantile(0.25)
Q3 = df['valor'].quantile(0.75)
IQR = Q3 - Q1
limite_inferior = Q1 - 1.5 * IQR
limite_superior = Q3 + 1.5 * IQR
outliers = df[(df['valor'] < limite_inferior) | (df['valor'] > limite_superior)]
print(outliers)
```

3. Modelagem Robusta (ex: Isolation Forest, DBSCAN)

- Usa algoritmos que isolam pontos raros.
- Exemplo com **Isolation Forest**:

```
from sklearn.ensemble import IsolationForest

modelo = IsolationForest(contamination=0.1)
df['outlier'] = modelo.fit_predict(df[['valor']])
outliers = df[df['outlier'] == -1]
print(outliers)
```

- ***Codificação:***
  - *One-hot encoding*: cria colunas binárias.
  - *Ordinal encoding*: mantém hierarquia.

Vimos na aula passada o que era o *One-hot encoding*. Vamos agora ver no que consiste o *Ordinal encoding*.

Ordinal Encoding é uma técnica de codificação de **variáveis categóricas** onde cada categoria é substituída por um **número inteiro**. A codificação preserva uma **ordem lógica ou hierárquica** entre os valores.

Exemplo simples:

Suponhamos a variável Tamanho com as seguintes categorias:

Tamanho	Ordem esperada
Pequeno	0
Médio	1
Grande	2

Com **Ordinal Encoding**, seria codificada assim:

```
import pandas as pd
from sklearn.preprocessing import OrdinalEncoder
df = pd.DataFrame({'Tamanho': ['Médio', 'Pequeno', 'Grande', 'Médio']})
encoder = OrdinalEncoder(categories=[['Pequeno', 'Médio', 'Grande']])
df['Tamanho_cod'] = encoder.fit_transform(df[['Tamanho']])
print(df)
```

Resultado:

```
  Tamanho  Tamanho_cod
0   Médio          1.0
1 Pequeno          0.0
2  Grande          2.0
3   Médio          1.0
```

Quando usar Ordinal Encoding?

Usar quando:

1. As categorias **têm ordem** (ex: níveis de escolaridade, satisfação, tamanho, risco).
2. O modelo pode **beneficiar da noção de hierarquia**.

Evitar quando:

1. As categorias **não têm ordem** (ex: cor dos olhos, nomes de países, marcas).
2. Nesse caso, usar **One-Hot Encoding** para evitar inferência errada de hierarquia.

- ***Escalonamento:***
    - *StandardScaler*: média 0 e desvio padrão 1.
    - MinMaxScaler: escala entre 0 e 1.
- **Feature engineering**: combinação, transformação ou criação de atributos com base na lógica do problema.

2.2. Seleção e Comparação de Modelos

Critérios de Escolha

- **Complexidade computacional** (tempo de treino e predição).
- **Capacidade de interpretação** (transparência para auditoria).
- **Comportamento em dados ruidosos** (resistência ao overfitting).

Exemplo Comparativo:

Modelo	Interpretação	Escalamento Necessário	Sensível a Outliers	Robusto para Overfitting
Regressão Logística	Alta	Sim	Sim	Não
KNN	Média	Sim	Sim	Sim (em k baixo)
Árvore de Decisão	Alta	Não	Não	Não
Random Forest	Média	Não	Não	Sim (controlado)
SVM	Baixa	Sim	Sim	Sim

2.3. Treino, Validação e Ajuste de Hiperparâmetros

Divisão típica do dataset

- **Treino** (60–70%): para ajustar os pesos/estruturas.
- **Validação** (15–20%): para ajustar hiperparâmetros (como profundidade de árvore, C do SVM (No algoritmo **SVM**, o parâmetro C controla o **grau de penalização para erros de classificação** — ou seja, **o equilíbrio entre a margem larga e os erros cometidos no treino. C é um hiperparâmetro — deve ser ajustado com validação cruzada (ex: GridSearchCV) para encontrar o melhor valor para o teu caso**), etc).
- **Teste** (15–20%): apenas para avaliação final.

Validação Cruzada (k-fold)

Validação cruzada é uma técnica estatística usada para avaliar o desempenho de modelos de aprendizagem automática, especialmente quando temos poucos dados. Em vez de usar apenas uma partição fixa para treino e outra para teste, a validação cruzada permite uma avaliação mais robusta e confiável, reduzindo a variância associada à divisão dos dados.

- Reduz o risco de overfitting nos hiperparâmetros.
- Em k-fold:

1. O conjunto de dados é dividido em **k subconjuntos** (ou *folds*) de tamanho aproximadamente igual.
2. O processo é repetido **k vezes**:
    - Em cada iteração, **um fold é usado para validação** (teste), e os **k – 1 restantes para treino**.
    - O modelo é ajustado nos dados de treino e avaliado no fold de teste.
3. No final, obtém-se **k resultados de avaliação**, que podem ser **médios** para dar uma métrica final (ex.: acurácia média).

Vantagens:

- **Melhor estimativa generalizada** do desempenho do modelo.
- **Reduz o risco de overfitting** na escolha de hiperparâmetros (como profundidade de uma árvore, número de vizinhos no KNN, etc.).
- Usa **todos os dados para treino e teste**, ao longo das iterações.

Variações do k-fold:

1. **StratifiedKFold**

- Mantém a mesma proporção de classes (no caso de classificação) em cada fold.
- Muito útil quando há classes desbalanceadas (por ex., 90% classe A, 10% classe B).
- Evita que um fold contenha só exemplos de uma classe.

2**. Leave-One-Out Cross-Validation** (LOOCV)

- É um caso especial onde k = n (n = número de exemplos).
- Em cada iteração, um único exemplo é usado como teste, e o resto como treino.
- Excelente para conjuntos de dados muito pequenos.
- Muito caro computacionalmente, pois são necessárias n iterações.

3. **Repeated k-Fold**

- Repete o processo k-fold várias vezes com diferentes divisões aleatórias.
- Ajuda a obter uma estimativa mais estável do desempenho.

```
# Importar as bibliotecas necessárias
from sklearn.model_selection import KFold, StratifiedKFold,
cross_val_score
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier

# Carregar o dataset Iris (flores)
# X contém as características (features), y contém os rótulos (classes)
X, y = load_iris(return_X_y=True)

# Criar o modelo de classificação: uma floresta aleatória (Random Forest)
model = RandomForestClassifier()

# ------------------------------
# Validação Cruzada com KFold
# ------------------------------

# Criar o objeto KFold com 5 divisões (splits)
# shuffle=True mistura os dados antes de dividir, random_state garante
reprodutibilidade
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Aplicar cross-validation: o modelo será treinado e testado 5 vezes
scores = cross_val_score(model, X, y, cv=kf)

# Imprimir as acurácias obtidas em cada iteração
print("Acurácias (k-fold):", scores)

# Calcular e imprimir a média das acurácias
print("Acurácia média (k-fold):", scores.mean())

# ------------------------------
# Validação Cruzada com StratifiedKFold
# ------------------------------

# Criar o objeto StratifiedKFold para manter a proporção entre as classes
em cada fold
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Aplicar cross-validation com estratificação
strat_scores = cross_val_score(model, X, y, cv=skf)

# Imprimir os resultados
print("Acurácias (Stratified k-fold):", strat_scores)
print("Acurácia média (Stratified k-fold):", strat_scores.mean())
```

A validação cruzada, especialmente o k-fold, é essencial em pipelines (fluxo de trabalho) de machine learning, tanto para avaliar modelos como para selecionar hiperparâmetros (Número de árvores, profundidade máxima, taxa de aprendizagem) de forma justa. A escolha entre k-fold, Stratified ou Leave-One-Out depende dos dados disponíveis e da complexidade computacional aceitável.

Exemplos de Hiperparâmetros

Árvores de Decisão:

- max_depth: profundidade máxima da árvore.
- min_samples_split: número mínimo de amostras para dividir um nó.

Random Forest:

- n_estimators: número de árvores na floresta.
- max_features: número de atributos considerados em cada divisão.

KNN:

- n_neighbors: número de vizinhos a considerar.

Como se escolhem?

Os hiperparâmetros são escolhidos através de **técnicas de otimização**, como:

- **Validação Cruzada com Grid Search** (GridSearchCV)
- **Random Search**
- **Algoritmos Bayesianos (ex.: Optuna, Hyperopt)**

Exemplo simples com GridSearchCV:

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 10]
}

grid = GridSearchCV(RandomForestClassifier(), param_grid, cv=5)
grid.fit(X, y)

print("Melhores hiperparâmetros:", grid.best_params_)
print("Melhor score:", grid.best_score_)
```

2.4. Teste Final e Generalização

O **teste final** nunca deve influenciar decisões de treino/validação. A sua única função é medir a **capacidade real de generalização**.

Erro de Generalização

- Representa a diferença entre desempenho nos dados de treino e nos dados reais.
- Está diretamente relacionado com:
    - **Capacidade do modelo (VC Dimension)**.
    - **Quantidade de dados**.
    - **Qualidade dos dados**.

3. Conceitos-Chave: Overfitting e Underfitting

Overfitting

- Causa: excesso de complexidade (árvores muito profundas, kNN com k baixo, redes neurais com muitos parâmetros).
- Sinais:
    - Acurácia de treino ≫ acurácia de teste.
    - Baixa performance em novos dados.
- Soluções:
    - Regularização (L1, L2).
    - Poda em árvores.
    - Aumento de dados.
    - Dropout (em redes neurais).

Underfitting

- Causa: modelo demasiado simples ou dados mal preparados.
- Sinais:
    - Baixa performance em treino e teste.
- Soluções:
    - Modelos mais complexos.
    - Inclusão de mais atributos relevantes.

4. Métricas de Avaliação: Análise Técnica

Matriz de Confusão

	PRED. POSITIVO	**PRED. NEGATIVO**
REAL POSITIVO	True Positive	False Negative
REAL NEGATIVO	False Positive	True Negative

- TP = doentes corretamente diagnosticados.
- FN = doentes não diagnosticados (crítico em medicina).

Métricas Derivadas

- **Accuracy** = (TP + TN) / Total
  ➜ útil em classes equilibradas.
- **Precision** = TP / (TP + FP)
  ➜ "Quantos dos que identifiquei como positivos realmente são?"
- **Recall (Sensibilidade)** = TP / (TP + FN)
  ➜ "Quantos dos reais positivos consegui identificar?"
- **F1-score** = 2 × (Precision × Recall) / (Precision + Recall)
  ➜ Média harmónica entre precisão e recall.

Acurácia alta pode ser enganadora:
Exemplo com dados desbalanceados (95% classe A, 5% classe B):
Um modelo que prevê sempre classe A terá 95% de acurácia, mas 0% de recall para a classe B.

Curvas ROC e AUC (Para classificação binária)

- **ROC Curve**: gráfico entre *TPR* e *FPR*.
- **AUC (Área sob a curva)**: mede a capacidade de separação entre classes.
  ➜ AUC = 1 → separação perfeita.
  ➜ AUC = 0.5 → classificador aleatório.

Imagina que estamos a treinar um modelo para detectar se um e-mail é SPAM (1) ou NÃO-SPAM (0). O modelo dá uma probabilidade (por exemplo, "80% de ser SPAM").
Mas... onde traçamos a linha para decidir? A 50%? A 70%?

Aqui entra a Curva ROC.

O que é a Curva ROC?

ROC significa *Receiver Operating Characteristic*.
É um gráfico que mostra **o desempenho de um modelo de classificação binária** à medida que mudamos o **limiar (threshold)** de decisão.

No gráfico ROC:

- **Eixo Y**: **TPR** (True Positive Rate)
  → "Quantos SPAMs reais conseguimos detetar?"
  Também chamado de **Sensibilidade** ou **Recall**.
- **Eixo X**: **FPR** (False Positive Rate)
  → "Quantos e-mails legítimos marcámos **erradamente** como SPAM?"

Como se cria a curva?

1. O modelo calcula **probabilidades** para cada exemplo.
2. Para vários limiares (de 0 a 1), calcula-se:
   - Quantos SPAMs foram detetados (TPR)?
   - Quantos NÃO-SPAMs foram falsamente detetados (FPR)?
3. Cada ponto forma a curva ROC.

O que é o AUC?

AUC significa Area Under the Curve → Área sob a curva ROC.

É um número entre 0 e 1 que resume quão bom é o modelo a separar as duas classes (SPAM vs NÃO-SPAM).

AUC	Interpretação
1.0	Separação perfeita
0.9 ~ 1.0	Excelente
0.8 ~ 0.9	Muito bom
0.7 ~ 0.8	Razoável
0.5	Pior cenário útil (classificação aleatória)
< 0.5	Modelo está a inverter as classes

Exemplo simplificado:

E-mail	Real (SPAM?)	Probabilidade dada pelo modelo
A	1 (sim)	0.95
B	0 (não)	0.85
C	1 (sim)	0.65
D	0 (não)	0.40

Se o modelo classificar como SPAM qualquer e-mail com probabilidade ≥ 0.5, o desempenho pode mudar muito dependendo desse limiar. A **curva ROC** mostra **todos os possíveis desempenhos**, de forma visual.

Em resumo:

- **ROC Curve**: gráfico do desempenho do modelo à medida que se altera o limiar.
- **AUC**: número entre 0 e 1 que mede **a qualidade do modelo para distinguir as duas classes**.

5. Exemplo Prático com Interpretação

```
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris

# Carga do dataset
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data,
iris.target, test_size=0.3, stratify=iris.target)

# Treino do modelo
modelo = DecisionTreeClassifier(max_depth=3)
modelo.fit(X_train, y_train)

# Predição e avaliação
y_pred = modelo.predict(X_test)
print(classification_report(y_test, y_pred,
target_names=iris.target_names))
```

A função classification_report() fornece:

- Precision
- Recall

- F1-score
- Suporte (número de exemplos por classe)