

<b>FT03</b>
<b>Curso:</b> UFCD 10810
<b>UFCD/Módulo/Temática:</b> UFCD 10810 - Fundamentos do desenvolvimento de modelos analíticos em Python
<b>Ação:</b> 10810_1L
<b>Formador/a:</b> Sandra Liliana Meira de Oliveira
<b>Data:</b>
<b>Nome do Formando/a:</b>

## Sessão 3 : Introdução à Aprendizagem Automática e Componentes

### Objetivos

- Compreender os fundamentos da aprendizagem automática
- Conhecer os principais componentes de sistemas de ML
- Aplicar regressão logística em dados reais
- Avaliar performance e mitigar overfitting
- Explorar datasets reais através de projetos práticos

## Parte I – Fundamentos Teóricos

### 1. O que é Inteligência Artificial (IA)?

2.

A **Inteligência Artificial** (IA) consiste na criação de sistemas que simulam capacidades humanas como:

- Reconhecimento de fala e imagem
- Tomada de decisões
- Resolução de problemas
- Aprendizagem com a experiência

### 2. Divisões da IA

Tipo de IA	Descrição
IA Simbólica	Baseada em lógica e regras explícitas. Ex: sistemas especialistas.
IA Estatística	Baseada em dados e modelos matemáticos. Aqui insere-se o <b>Machine Learning</b> .

### 3. O que é Machine Learning (ML)?

É um ramo da IA que permite que **computadores aprendam com dados**, sem serem explicitamente programados para cada tarefa.

#### Exemplos:

- Previsão de vendas
- Diagnóstico médico
- Detecção de fraudes
- Recomendação de filmes

### 4. Tipos de Aprendizagem Automática

Tipo	Descrição	Exemplos
Supervisionado	Usa dados com rótulos	Classificação, Regressão
Não-Supervisionado	Não tem rótulos, procura padrões	Agrupamento (clustering)
Por Reforço	Aprende com tentativa/erro e recompensas	Jogos, robótica

### 5. Componentes Essenciais de um Sistema de ML

Componente	Função
Dados	Entradas (features) e saídas (labels)
Modelo	Algoritmo que aprende os padrões (ex: Regressão Logística)
Função Objetivo	Mede o erro entre a previsão e a realidade (ex: erro quadrático médio)
Ciclo treino/teste	Separação de dados para evitar avaliação enviesada

## Parte II – Regressão Logística com Titanic (passo a passo)

### Conceito

A **Regressão Logística** é um modelo de **classificação binária**, que prevê a **probabilidade** de um resultado entre duas classes (ex: sobreviver ou não).

Apesar do nome, **não serve para prever valores contínuos**. A saída é uma **probabilidade** entre 0 e 1, convertida depois numa classe 0 ou 1.

A regressão logística usa a função **sigmóide**:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

### Por que usamos Regressão Logística aqui?

- É **simples e eficaz** para introdução ao ML
- Ideal para **classificação binária**
- Funciona bem com **variáveis numéricas ou categóricas codificadas**
- **Interpretável**: permite perceber o peso de cada variável

### Explicação passo a passo do código

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

- pandas: biblioteca para manipulação de dados tabulares (DataFrames)
- train\_test\_split: separa os dados em conjuntos de treino e teste
- LogisticRegression: classe que define o modelo de regressão logística
- accuracy\_score, confusion\_matrix, classification\_report: métricas para avaliação do modelo

```
# Carregar e preparar o dataset
df =
pd.read_csv("https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv")
df = df[["Survived", "Pclass", "Sex", "Age"]].dropna()
df["Sex"] = df["Sex"].map({"male": 0, "female": 1})
```

- read\_csv: lê o ficheiro CSV diretamente da internet
- Selecionam-se apenas as colunas relevantes
- dropna(): remove linhas com valores em falta

- map(...): converte "male"/"female" para 0/1 (necessário para ML)

```
X = df[["Pclass", "Sex", "Age"]] # Features (entradas)
y = df["Survived"] # Target (saída binária)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

- X e y são definidos separadamente
- train\_test\_split: 70% treino e 30% teste, com semente (random\_state) para repetibilidade

```
modelo = LogisticRegression(max_iter=200)
modelo.fit(X_train, y_train)
```

- Criação do modelo com número máximo de iterações fixado
- fit(...): treino do modelo com dados de treino

```
y_pred = modelo.predict(X_test)
print("Acurácia:", accuracy_score(y_test, y_pred))
print("Matriz de Confusão:\n", confusion_matrix(y_test, y_pred))
print("Relatório:\n", classification_report(y_test, y_pred))
```

- predict(...): previsões com os dados de teste
- accuracy\_score: percentagem de classificações corretas
- confusion\_matrix: matriz com TP, TN, FP, FN

A função confusion\_matrix do scikit-learn calcula uma **matriz de confusão**, que é uma tabela que permite avaliar o desempenho de um modelo de **classificação**, comparando os **valores reais** com os **valores previstos**.

- **TP (True Positive)**: O modelo previu 1 e o rótulo real era 1 → previsão correta de positivo.
- **TN (True Negative)**: O modelo previu 0 e o rótulo real era 0 → previsão correta de negativo.
- **FP (False Positive)**: O modelo previu 1 mas o rótulo real era 0 → "falso alarme".
- **FN (False Negative)**: O modelo previu 0 mas o rótulo real era 1 → "falha na deteção".
- classification\_report: métricas detalhadas como precisão, revocação e F1-score

## Interpretação dos coeficientes

```
for nome, coef in zip(X.columns, modelo.coef_[0]):  
    print(f"{nome}: {coef:.4f}")
```

- Mostra o peso de cada variável no modelo
- Coeficiente positivo → aumenta probabilidade de classe 1 (sobreviveu)
- Coeficiente negativo → diminui probabilidade

## Parte III – Overfitting e Validação Cruzada

### Overfitting

- Modelo aprende demais os dados de treino → falha em dados novos

### Estratégias para evitar:

- Separar treino/teste
- Validação cruzada
- Regularização

### Validação Cruzada

```
from sklearn.model_selection import cross_val_score  
  
modelo = LogisticRegression(max_iter=200)  
scores = cross_val_score(modelo, X, y, cv=5, scoring="accuracy")  
  
print("Scores:", scores)  
print("Média da acurácia:", scores.mean())
```

- Divide o treino em 5 partes e avalia a estabilidade do modelo

## Parte IV – Atividade Resolvida: Simular Overfitting

### Exemplo 1: Treino com 5 exemplos

```
amostra = df.sample(n=5, random_state=42)  
X_amostra = amostra[["Pclass", "Sex", "Age"]]  
y_amostra = amostra["Survived"]
```

```

modelo_pequeno = LogisticRegression()
modelo_pequeno.fit(X_amostra, y_amostra)

print("Acurácia treino (5 exemplos):", accuracy_score(y_amostra,
modelo_pequeno.predict(X_amostra)))

```

- O modelo "memoriza" → overfitting

### **Exemplo 2: Adicionar ruído**

```

import numpy as np
df_100 = df.sample(n=100, random_state=42)
X_100 = df_100[["Pclass", "Sex", "Age"]].copy()
for i in range(5):
    X_100[f"ruído_{i}"] = np.random.randn(100)
y_100 = df_100["Survived"]

modelo_ruido = LogisticRegression()
modelo_ruido.fit(X_100, y_100)
print("Acurácia com ruído:", accuracy_score(y_100, modelo_ruido.predict(X_100)))

```

- O modelo ajusta-se a variáveis sem sentido

### **Exemplo 3: Aplicar regularização**

```

modelo_reg = LogisticRegression(C=0.01)
modelo_reg.fit(X_100, y_100)
print("Acurácia com regularização:", accuracy_score(y_100, modelo_reg.predict(X_100)))
for nome, coef in zip(X_100.columns, modelo_reg.coef_[0]):
    print(f"{nome}: {coef:.4f}")

```

- Coeficientes de variáveis irrelevantes tendem a 0 → modelo mais robusto

## **Parte V – Exercícios com Datasets Reais do Kaggle**

### **Exercício 1 – Titanic (clássico)**

- <https://www.kaggle.com/c/titanic/data>
- Tarefa:
  - Usar outras variáveis (e.g. Fare, Embarked)
  - Codificar variáveis categóricas
  - Comparar performance com/sem normalização

### **NOTA:**

Modelos de machine learning como regressão logística, KNN, ou árvores de decisão trabalham apenas com valores numéricos.

Logo, variáveis categóricas (texto) como "profissão", "sexo" ou "cidade" precisam ser transformadas em números antes de alimentar o modelo.

## Duas abordagens comuns

Método	Quando usar	Exemplo
<b>Label Encoding</b>	Variáveis <b>ordinais</b> (com ordem)	"baixo", "médio", "alto" → 0, 1, 2
<b>OneHotEncoding</b>	Variáveis <b>nominais</b> (sem ordem)	"admin", "tech", "blue-collar" → colunas binárias

## OneHotEncoding – Exemplo passo a passo

```
#1. Criar dataset de exemplo
import pandas as pd
df = pd.DataFrame({
    "idade": [25, 45, 35, 33],
    "profissao": ["admin", "blue-collar", "technician", "admin"],
    "subscrito": [0, 1, 0, 1]
})

#2. Separar features e target
X = df[["idade", "profissao"]] # variáveis independentes
y = df["subscrito"]           # variável dependente (classe)

#3. Codificar com OneHotEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
# Definir transformador
preprocessador = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(drop='first'), ['profissao']) # cria colunas binárias
    ],
    remainder='passthrough' # mantém a coluna "idade"
)
# Criar pipeline completo
modelo = Pipeline(steps=[
    ('preprocessamento', preprocessador),
    ('classificador', LogisticRegression())
])
```

```
# Treinar modelo
modelo.fit(X, y)

#4. Ver variáveis codificadas
print(modelo.named_steps['preprocessamento'].get_feature_names_out())
#Exemplo de saída:
['cat_blue-collar' 'cat_technician' 'idade']
```

## Conclusão

- **OneHotEncoder** é a forma correta de codificar variáveis nominais (sem ordem).
- **LabelEncoder** só deve ser usado com variáveis ordinais.
- Com Pipeline e ColumnTransformer, o pré-processamento é automatizado e organizado.

## Exercício 2 – Heart Disease Prediction UCI

- <https://www.kaggle.com/datasets/priyanka841/heart-disease-prediction-uci>
- Tarefa:
  - Prever presença de doença cardíaca com regressão logística
  - Usar regularização: C=1, 0.1, 0.01
  - Avaliar com classification\_report e confusion\_matrix

### NOTA:

O parâmetro **C** é usado para controlar a regularização em modelos como **SVM (Support Vector Machine)** e **Regressão Logística** no scikit-learn. Ele está inversamente relacionado à força da regularização:

- **C grande (ex: C=1)** → Menor regularização, o modelo tenta ajustar mais precisamente os dados de treino.
- **C pequeno (ex: C=0.1 ou C=0.01)** → Maior regularização, o modelo generaliza mais, evita overfitting.

## Exemplo prático com LogisticRegression:

```
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Dados
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
```

```
# Testar diferentes valores de C
for C in [1, 0.1, 0.01]:
    model = LogisticRegression(C=C, max_iter=1000)
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    acc = accuracy_score(y_test, predictions)
    print(f"Acurácia com C={C}: {acc:.4f}")
```

### Exercício 3 – Bank Marketing

- <https://www.kaggle.com/datasets/henriqueyamahata/bank-marketing>

- Tarefa:

- Prever se o cliente adere ao produto bancário
- Codificar variáveis com OneHotEncoder
- Comparar desempenho com regressão logística e árvore de decisão

**NOTA:**

O OneHotEncoder transforma variáveis categóricas nominais (como "azul", "verde", "vermelho") em colunas binárias (0 ou 1), criando uma coluna por categoria.

**Exemplo:**

Se tiveres a variável Cor = ["azul", "verde", "azul", "vermelho"], o resultado será:

azul	verde	vermelho
1	0	0
0	1	0
1	0	0
0	0	1

**Como aplicar OneHotEncoder no Scikit-Learn**

Vamos usar um exemplo prático com o dataset Bank Marketing (ou similar).

**1. Importar bibliotecas**

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
```

**2. Criar dataset de exemplo**

```
df = pd.DataFrame({
    "idade": [25, 45, 35, 33],
    "profissao": ["admin", "blue-collar", "technician", "admin"],
    "subscrito": [0, 1, 0, 1]
})
```

### 3. Separar features e target

```
X = df[["idade", "profissao"]] # variáveis independentes
y = df["subscrito"] # variável dependente (alvo)
```

### 4. Definir o ColumnTransformer com OneHotEncoder

```
preprocessador = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(drop='first'), ['profissao']) # drop='first' evita
multicolinearidade
    ],
    remainder='passthrough' # mantém as colunas numéricas
)
```

### 5. Criar um pipeline com regressão logística

```
modelo = Pipeline(steps=[
    ('preprocessamento', preprocessador),
    ('classificador', LogisticRegression())
])
```

### 6. Treinar o modelo

```
modelo.fit(X, y)
```

### 7. Visualizar variáveis codificadas

```
# Ver os nomes das colunas após transformação
nome_colunas = modelo.named_steps['preprocessamento'].get_feature_names_out()
print(nome_colunas)
```

Isto pode devolver algo como:

```
['cat_blue-collar' 'cat_technician' 'idade']
```

Notas Importantes

- O drop='first' evita criar uma coluna redundante (multicolinearidade)
- O OneHotEncoder é preferível ao LabelEncoder para variáveis nominais sem ordem (ex: profissão, cidade)
- Para dados reais com múltiplas categorias, o uso do ColumnTransformer permite misturar colunas numéricas e categóricas no mesmo pipeline

## Exercício 4 – Breast Cancer Wisconsin

- <https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data>
- Tarefa:
  - Criar pipeline com StandardScaler + LogisticRegression
  - Avaliar com cross-validation

- Visualizar a matriz de confusão com seaborn