

Machine Learning Engineer

Python para ML e IA

Modelos em API

Leonardo Pena

/ Seja muito bem vindo



OBJETIVO

Treinar um modelo de
classificação (Iris) e
salvar com joblib



COMPARATIVO

Foco no notebook
modelo.ipynb



EXEMPLO

Demonstração
simples do fluxo
load_iris ->
train_test_split ->
LogisticRegression

Objetivo dessa parte



Passo 1

Carregar o dataset Iris com `sklearn.datasets.load_iris`



Passo 2

Separar dados em treino e teste usando `train_test_split`



Passo 3

Treinar um modelo simples de regressão logística



Passo 4

Avaliar desempenho (accuracy) no conjunto de teste



Passo 5

Salvar o modelo treinado em `modelo_iris.pkl` com `joblib`



Machine Learning e o Dataset





/ Conceito

- Iris dataset: 150 amostras de flores (Iris setosa, versicolor, virginica)
- Quatro features: sepal_length, sepal_width, petal_length, petal_width
- Tarefa: classificar a espécie (target = 0,1,2)
- Regressão logística: método de classificação linear
- Exemplo didático para APIs de ML

A decorative graphic at the bottom of the slide consisting of numerous thin, light gray wavy lines that create a sense of movement and depth, resembling a stylized landscape or water ripples.

/ Preparando o ambiente

- Crie o venv
- Ative o ambiente
- Instale joblib e scikit-learn

/

Vamos começar com um arquivo notebook:
modelo.ipynb



/ Importando Bibliotecas



```
import joblib
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

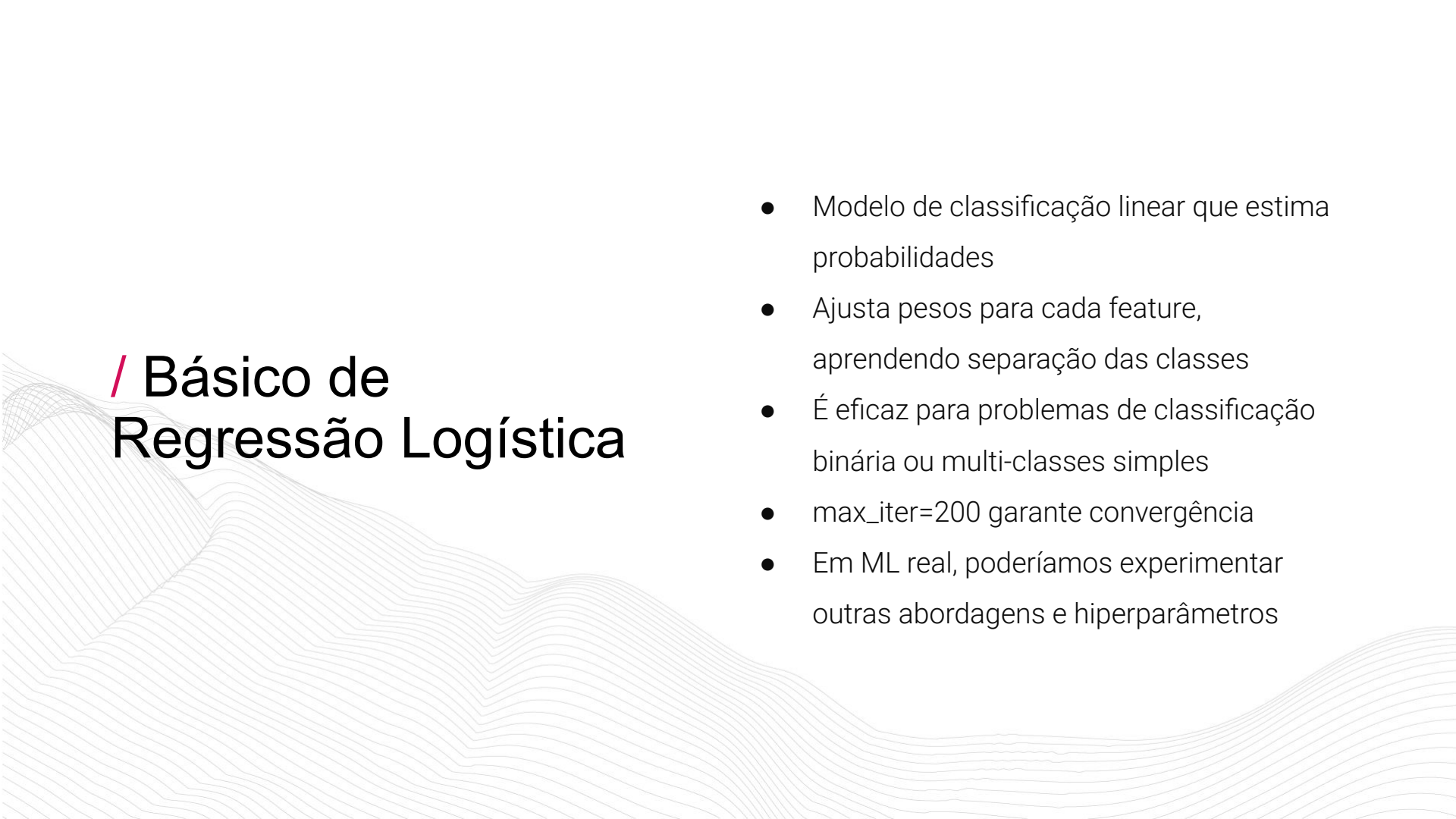
[]

/ Carregando e Dividindo o Dataset

```
data = load_iris()
X, y = data.data, data.target

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

✓ 0.0s



/ Básico de Regressão Logística

- Modelo de classificação linear que estima probabilidades
- Ajusta pesos para cada feature, aprendendo separação das classes
- É eficaz para problemas de classificação binária ou multi-classes simples
- `max_iter=200` garante convergência
- Em ML real, poderíamos experimentar outras abordagens e hiperparâmetros

/

Treinando o Modelo

```
model = LogisticRegression(max_iter=200, random_state=42)  
model.fit(X_train, y_train)
```

[11] ✓ 0.0s

...

▼ LogisticRegression ⓘ ⓘ
LogisticRegression(max_iter=200, random_state=42)

/

Avaliando o modelo

```
▷ score = model.score(X_test, y_test)  
print("Acurácia no conjunto de teste:", score)  
[13] ✓ 0.0s
```



/ Interpretação do Score

- Se a acurácia for ~90% ou mais, indica bom acerto para esse dataset
- Modelos avançados podem chegar a 97-98% no Iris
- Para APIs de produção, métricas adicionais (F1, recall) podem ser relevantes
- Comparar baseline: “será que precisamos de algo mais complexo?”
- Como é aula de exemplo, RegL é suficiente

/

Salvando o Modelo

```
joblib.dump(model, "modelo_iris.pkl")  
print("Modelo salvo em modelo_iris.pkl")
```

[14] ✓ 0.0s

... Modelo salvo em modelo_iris.pkl

> venv
modelo_iris.pkl
modelo.ipynb



/ Confirmando Geração do Arquivo

- Verificar no Colab/VSCode se `modelo_iris.pkl` foi criado
- Tamanho do arquivo deve ser relativamente pequeno
- Em pipelines de ML, esse artefato é enviado para repositório ou storage
- Possibilidade de versionar modelos (diferentes seeds, parâmetros)
- Base para “API de predição”



/ Vantagens de Salvar um Modelo

- Evita treinar toda vez que a API for subir
- Facilita reuso do mesmo modelo em diferentes linguagens
- Possível armazenar múltiplas versões e revertê-las em caso de problemas
- Ajuda a escalar: várias réplicas da API usam o mesmo modelo.pkl
- Desacopla parte de “treino” da parte “deploy”



/ Comparações com Outros Modelos

- Poderíamos usar RandomForestClassifier ou SVC no Iris
- A ideia é a mesma: .fit() e .score()
- joblib salva do mesmo jeito
- Sempre teste se a complexidade extra compensa a performance
- Em aula, regressão logística é suficiente para ilustrar



/ Documentando Experimentos

- Em ML real, é comum registrar seed, data do treinamento, versão do dataset
- Ferramentas como MLflow, DVC, Weights & Biases ajudam no tracking
- Aqui, basta anotar manualmente ou manter em repositório
- Minimiza riscos de perda do “caminho” e reprodutibilidade
- Prática recomendada em times de Data Science

POSTECH

FIAP + alura