title: "Algoritmos na Prática" subtitle: "Regressão, Árvores, K-Means e PCA — Da Intuição ao Código de Produção" author: "Nexus Affil'IA'te MMN_IA" collection: "Curso Universo IA — Coletânea Técnica Vol. 03" version: "1.0" date: "2026-06" classification: "Acadêmico · Técnico · Hands-On · PhD-Level"

Capa

> "Teoria sem código é filosofia. Código sem teoria é hacking. Os dois juntos são engenharia." — Nexus Affil'IA'te MMN_IA

Sumário

1. **Prólogo — O Algoritmo Não Importa. O Problema Importa.**
2. **Capítulo 1 — Setup do Ambiente Profissional**
3. **Capítulo 2 — Pipeline de Dados: Do Raw ao Model-Ready**
4. **Capítulo 3 — Análise Exploratória (EDA)**
5. **Capítulo 4 — Feature Engineering**
6. **Capítulo 5 — Regressão Linear com NumPy (Do Zero)**
7. **Capítulo 6 — Regressão Linear com Scikit-Learn**
8. **Capítulo 7 — Regressão Logística — Detecção de Spam**
9. **Capítulo 8 — Árvores de Decisão — Risco de Crédito**
10. **Capítulo 9 — Random Forest — A Força do Ensemble**
11. **Capítulo 10 — Gradient Boosting (XGBoost, LightGBM)**
12. **Capítulo 11 — KNN — Sistema de Recomendação**
13. **Capítulo 12 — K-Means — Segmentação de Clientes**
14. **Capítulo 13 — PCA — Compressão de Imagens**
15. **Capítulo 14 — DBSCAN — Detecção de Fraude**
16. **Capítulo 15 — Isolation Forest — Outliers em Séries Temporais**
17. **Capítulo 16 — SVM — Classificação de Texto**
18. **Capítulo 17 — Naive Bayes — Filtro de Spam (Clássico)**
19. **Capítulo 18 — Validação Cruzada e Hyperparameter Tuning**
20. **Capítulo 19 — Pipelines e MLflow**
21. **Capítulo 20 — Deploy: Do Notebook à API**

22. **Capítulo 21 — Monitoramento em Produção (Drift)**
23. **Capítulo 22 — MLOps: O Ciclo de Vida Completo**
24. **Capítulo 23 — Algoritmos para Time Series**
25. **Capítulo 24 — Algoritmos para NLP Clássico**
26. **Capítulo 25 — Quando NÃO Usar ML**
27. **Glossário**
28. **Referências**

Prólogo — O Algoritmo Não Importa. O Problema Importa.

É tentador cair na armadilha de **otimizar o algoritmo** quando o verdadeiro gargalo está em outro lugar:

- **Dados ruins** (mais comum que se imagina).
- **Problema mal formulado** (métrica errada).
- **Distribuição diferente em produção.**
- **Falta de feature engineering.**

Este volume é **hands-on**. Cada capítulo traz: - Conceito teórico - Código Python executável - Caso de uso real - Armadilhas comuns

Vou usar datasets públicos (Iris, Boston Housing, MNIST, Titanic) e casos práticos do **ecossistema Nexus Affil'IA'te**.

Capítulo 1 — Setup do Ambiente Profissional

1.1 Python + Anaconda/Mamba

```
# Instalar Mamba (mais rápido que conda)
curl -L https://mamba.readthedocs.io/en/latest/installation/mamba-installation.sh | b
```

```
# Criar ambiente
mamba create -n mt-prod python=3.11 -y
mamba activate mt-prod

# Instalar libs core
pip install numpy pandas scikit-learn matplotlib seaborn
pip install xgboost lightgbm catboost optuna
pip install mlflow fastapi uvicorn
```

1.2 Hardware Recomendado

- **Treino básico:** 16GB RAM, CPU multi-core.
- **Deep Learning:** GPU NVIDIA com 12GB+ VRAM (RTX 3060+).
- **Produção:** Multi-GPU, Kubernetes, GPUs A100/H100.

1.3 Estrutura de Projeto

```
ml-project/
├── data/
│   ├── raw/              # Dados originais (read-only)
│   ├── processed/        # Dados limpos
│   └── external/         # Dados de terceiros
├── notebooks/            # Jupyter de exploração
├── src/
│   ├── data/             # Scripts de data prep
│   ├── features/         # Feature engineering
│   ├── models/           # Treino e inferência
│   └── visualization/    # Plots
├── tests/                # Unit tests
├── configs/              # YAML de configuração
├── mlruns/               # MLflow artifacts
├── models/               # Modelos serializados
├── reports/              # Outputs finais
└── pyproject.toml        # Dependências
```

1.4 Boas Práticas

- **Reprodutibilidade:** `np.random.seed(42)`, `random_state` em tudo.
- **Versionamento:** Git + DVC para dados.
- **Logging:** MLflow, Weights & Biases.

- **Documentação:** Docstrings em tudo, README claro.
- **Testing:** pytest, data validation com Great Expectations.

Capítulo 2 — Pipeline de Dados: Do Raw ao Model-Ready

2.1 Etapas Fundamentais

```
Raw Data → Profiling → Cleaning → Transformation → Feature Engineering → Splitting →
```

2.2 Profiling com Pandas Profiling

```
import pandas as pd
from ydata_profiling import ProfileReport

df = pd.read_csv('data/raw/customers.csv')
profile = ProfileReport(df, title='Customer Data Profile')
profile.to_file('reports/eda.html')
```

Gera relatório com: tipos, missing, distribuição, correlações, alertas.

2.3 Tratamento de Missing Values

```
# Remoção
df.dropna()                    # Remove linhas com qualquer NaN
df.dropna(thresh=5)            # Mantém linhas com ≥5 valores não-NaN

# Imputação
df['age'].fillna(df['age'].median(), inplace=True)        # Mediana
df['category'].fillna(df['category'].mode()[0])           # Moda
df.fillna(method='ffill')                                 # Forward fill
```

Imputação avançada: - **KNN Imputer:** Usa vizinhos para imputar. - **Iterative Imputer (MICE):** Modela cada feature com missing como função das outras.

2.4 Detecção de Outliers

```
# IQR Method
Q1, Q3 = df['price'].quantile([0.25, 0.75])
IQR = Q3 - Q1
outliers = df[(df['price'] < Q1 - 1.5*IQR) | (df['price'] > Q3 + 1.5*IQR)]

# Z-Score
from scipy.stats import zscore
z_scores = zscore(df['price'])
df_no_outliers = df[abs(z_scores) < 3]
```

2.5 Encoding de Variáveis Categóricas

```
# One-Hot Encoding
df_encoded = pd.get_dummies(df, columns=['city'])

# Label Encoding (ordem importa)
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['size_encoded'] = le.fit_transform(df['size'])  # S, M, L → 0, 1, 2

# Target Encoding (média da variável target por categoria)
df['city_target'] = df.groupby('city')['target'].transform('mean')

# Embedding (para alta cardinalidade >50 categorias)
# Use rede neural com embedding layer
```

2.6 Scaling

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler

scaler = StandardScaler()        # Média 0, std 1
df_scaled = scaler.fit_transform(df[numeric_cols])

minmax = MinMaxScaler()          # [0, 1]
robust = RobustScaler()          # Usa mediana/IQR, robusto a outliers
```

Quando usar: - **StandardScaler:** Regressão, SVM, KNN, PCA. - **MinMaxScaler:** Redes neurais, quando há limites conhecidos. -

RobustScaler: Dados com outliers. - **Não escalar:** Árvores, Gradient Boosting (já são invariantes a escala).

Capítulo 3 — Análise Exploratória (EDA)

3.1 Os 5 Mandamentos do EDA

1. **Sempre olhe os dados crus** — `df.head()`, `df.sample(10)`.
2. **Entenda as distribuições** — Histogramas, KDE, boxplots.
3. **Procure correlações** — Heatmap, pairplot.
4. **Analise por segmento** — Groupby + agregação.
5. **Questione outliers** — Investigar antes de remover.

3.2 Visualizações Essenciais

```
import matplotlib.pyplot as plt
import seaborn as sns

# Univariada
df['age'].hist(bins=30)
sns.boxplot(x=df['salary'])

# Bivariada
sns.scatterplot(x='age', y='salary', data=df, hue='churn')
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')

# Multivariada
sns.pairplot(df[['age', 'salary', 'churn']], hue='churn')

# Categóricas
sns.countplot(x='city', data=df, order=df['city'].value_counts().index)
```

3.3 Estatísticas-Chave

```
df.describe()                    # Summary
df['col'].skew()                 # Assimetria
df['col'].kurtosis()             # Curtose
df.corr()                        # Correlação Pearson
df.corr(method='spearman')       # Não-linear
```

3.4 Quando Suspeitar

- **Distribuição bimodal:** Pode haver dois grupos misturados.
- **Skew extremo:** Variável precisa de transformação (log, sqrt).
- **Valores constantes:** Feature inútil.
- **Alta correlação (>0.95):** Multicolinearidade.

Capítulo 4 — Feature Engineering

4.1 A Ciência Mais Importante

> "Better features beat better algorithms." — Pedro Domingos

4.2 Transformações Numéricas

```
# Log (skew positivo)
df['log_income'] = np.log10(df['income'])

# Box-Cox (otimiza lambda)
from scipy.stats import boxcox
df['income_bc'], lam = boxcox(df['income'] + 1)

# Binning
df['age_bin'] = pd.cut(df['age'], bins=[0, 18, 30, 50, 100],
                       labels=['child', 'young', 'adult', 'senior'])
```

4.3 Interações

```
df['price_per_km'] = df['price'] / (df['distance'] + 1)
df['age_income_ratio'] = df['age'] / (df['income'] + 1)
```

4.4 Encoding de Datas

```
df['date'] = pd.to_datetime(df['date'])
df['year'] = df['date'].dt.year
df['month'] = df['date'].dt.month
df['day_of_week'] = df['date'].dt.dayofweek
df['is_weekend'] = df['day_of_week'].isin([5, 6]).astype(int)
df['days_since'] = (pd.Timestamp.now() - df['date']).dt.days
```

4.5 Text Features (Básico)

```
df['text_length'] = df['text'].str.len()
df['num_words'] = df['text'].str.split().str.len()
df['has_question'] = df['text'].str.contains('?').astype(int)
```

4.6 Feature Selection

```
# Filter: correlação com target
corr_with_target = df.corr()['target'].abs().sort_values(ascending=False)

# Embedded: usar feature importance do model
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
model.fit(X, y)
importance = pd.Series(model.feature_importances_, index=X.columns)
top_features = importance.nlargest(10).index

# Wrapper: Recursive Feature Elimination (RFE)
from sklearn.feature_selection import RFE
rfe = RFE(estimator=model, n_features_to_select=10)
X_selected = rfe.fit_transform(X, y)
```

Capítulo 5 — Regressão Linear com NumPy (Do Zero)

5.1 O Algoritmo

```
import numpy as np

class LinearRegression:
    def __init__(self, learning_rate=0.01, n_iterations=1000):
        self.lr = learning_rate
        self.n_iters = n_iterations

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(self.n_iters):
            y_pred = np.dot(X, self.weights) + self.bias

            dw = (1/n_samples) * np.dot(X.T, (y_pred - y))
            db = (1/n_samples) * np.sum(y_pred - y)

            self.weights -= self.lr * dw
            self.bias -= self.lr * db

    def predict(self, X):
        return np.dot(X, self.weights) + self.bias
```

5.2 Aplicação: Predição de Preços de Imóveis

```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

data = fetch_california_housing()
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
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

model = LinearRegression(learning_rate=0.01, n_iterations=1000)
model.fit(X_train, y_train)
predictions = model.predict(X_test)

from sklearn.metrics import mean_squared_error, r2_score
print(f"MSE: {mean_squared_error(y_test, predictions):.3f}")
print(f"R²: {r2_score(y_test, predictions):.3f}")
```

5.3 Lições

- Implementação do zero revela a matemática.
- Convergência depende de learning rate e scaling.
- Scikit-learn é ~100x mais otimizado (usa LAPACK).

Diagrama: Regressão e Gradient Descent

Capítulo 6 — Regressão Linear com Scikit-Learn

6.1 API Padrão

```
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

# Pipeline
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('model', Ridge(alpha=1.0))
])

pipeline.fit(X_train, y_train)
score = pipeline.score(X_test, y_test)
```

6.2 Regularização

```
# L2 (Ridge) - penaliza pesos grandes
ridge = Ridge(alpha=1.0)

# L1 (Lasso) - produz esparsidade
lasso = Lasso(alpha=0.1)

# L1 + L2 (ElasticNet)
elastic = ElasticNet(alpha=0.1, l1_ratio=0.5)
```

6.3 Diagnosticando o Modelo

```
import matplotlib.pyplot as plt
import seaborn as sns

residuals = y_test - predictions

# Residuos vs Predições (deve ser random)
plt.scatter(predictions, residuals)
plt.axhline(0, color='red', linestyle='--')

# QQ Plot (normalidade)
from scipy.stats import probplot
probplot(residuals, dist="norm", plot=plt)

# Heterocedasticidade
sns.scatterplot(x=predictions, y=residuals**2
```

Capítulo 7 — Regressão Logística — Detecção de Spam

7.1 O Dataset

Usaremos o **SMS Spam Collection** (5574 mensagens).

```
import pandas as pd

url = "https://raw.githubusercontent.com/justmarkham/DAT8/master/data/sms.tsv"
df = pd.read_csv(url, sep='\t', names=['label', 'message'])
df['label_num'] = (df['label'] == 'spam').astype(int)
print(df['label'].value_counts())
# ham    4825
# spam    747
```

7.2 Feature Engineering com TF-IDF

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(max_features=3000, stop_words='english', ngram_range=(1,2))
X = tfidf.fit_transform(df['message'])
y = df['label_num']
```

7.3 Modelo

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

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

model = LogisticRegression(max_iter=1000, class_weight='balanced')
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

7.4 Threshold Tuning

Por padrão, threshold = 0.5. Em problemas desbalanceados, podemos ajustar:

```
from sklearn.metrics import precision_recall_curve

y_proba = model.predict_proba(X_test)[:, 1]
```

```
precisions, recalls, thresholds = precision_recall_curve(y_test, y_proba)

# Escolher threshold com F1 ótimo
import numpy as np
f1_scores = 2 * (precisions * recalls) / (precisions + recalls + 1e-10)
optimal_threshold = thresholds[np.argmax(f1_scores)]
```

7.5 Interpretação dos Coeficientes

```
feature_names = tfidf.get_feature_names_out()
coefs = pd.Series(model.coef_[0], index=feature_names)
print("Top 10 palavras que indicam SPAM:")
print(coefs.nlargest(10))
print("\nTop 10 palavras que indicam HAM:")
print(coefs.nsmallest(10))
```

Capítulo 8 — Árvores de Decisão — Risco de Crédito

8.1 O Problema

Prever se um cliente **inadimplente** (default) com base em: - Idade - Renda - Histórico de crédito - Dívida atual - Score de crédito

8.2 Dataset Sintético

```
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=2000, n_features=10, n_informative=6,
                           n_redundant=2, random_state=42)
```

8.3 Modelo

```
from sklearn.tree import DecisionTreeClassifier, export_text, plot_tree

```

```
tree = DecisionTreeClassifier(max_depth=5, min_samples_split=20, random_state=42)
tree.fit(X_train, y_train)

# Visualizar
print(export_text(tree, feature_names=feature_names))
plot_tree(tree, feature_names=feature_names, class_names=['OK', 'Default'], filled=Tr
```

8.4 Análise de Importância

```
import matplotlib.pyplot as plt
import numpy as np

importances = tree.feature_importances_
indices = np.argsort(importances)[::-1]

plt.figure(figsize=(10, 6))
plt.title("Feature Importances")
plt.bar(range(X.shape[1]), importances[indices])
plt.xticks(range(X.shape[1]), [feature_names[i] for i in indices], rotation=45)
plt.tight_layout()
plt.show()
```

Diagrama: Árvore de Decisão

Capítulo 9 — Random Forest — A Força do Ensemble

9.1 O Algoritmo

Random Forest (Breiman, 2001) combina **muitas árvores** treinadas em subamostras dos dados.

Por que funciona: - **Bagging:** Cada árvore vê ~63% dos dados (com reposição). - **Feature randomness:** Cada split considera apenas √p features. - **Voting:** Predição = média (regressão) ou voto (classificação).

9.2 Código

```
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor

rf_clf = RandomForestClassifier(
    n_estimators=200,
    max_depth=10,
    min_samples_split=5,
    max_features='sqrt',
    n_jobs=-1,
    random_state=42
)
rf_clf.fit(X_train, y_train)
y_pred = rf_clf.predict(X_test)
```

9.3 Out-of-Bag (OOB) Score

```
rf_oob = RandomForestClassifier(n_estimators=200, oob_score=True, random_state=42)
rf_oob.fit(X_train, y_train)
print(f"OOB Score: {rf_oob.oob_score_:.3f}")
```

9.4 Hyperparâmetros Críticos

- `n_estimators`: Mais árvores = mais estabilidade (diminishing returns).
- `max_depth`: Profundidade máxima (evita overfit).
- `min_samples_leaf`: Mínimo de amostras em folha.
- `max_features`: Features por split (sqrt para classificação, n/3 para regressão).

Capítulo 10 — Gradient Boosting (XGBoost, LightGBM)

10.1 O Paradigma

Boosting treina modelos **sequencialmente**, cada um corrigindo os erros do anterior.

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

Onde h_m é o modelo fraco (geralmente uma árvore rasa) que minimiza os **pseudo-resíduos** (negativo do gradiente da loss).

10.2 XGBoost

```
import xgboost as xgb

xgb_model = xgb.XGBClassifier(
    n_estimators=300,
    max_depth=6,
    learning_rate=0.05,
    subsample=0.8,
    colsample_bytree=0.8,
    reg_alpha=0.1,
    reg_lambda=1.0,
    use_label_encoder=False,
    eval_metric='logloss',
    n_jobs=-1,
    random_state=42
)

xgb_model.fit(X_train, y_train,
              eval_set=[(X_test, y_test)],
              early_stopping_rounds=50,
              verbose=100)
```

10.3 LightGBM

```
import lightgbm as lgb

lgb_model = lgb.LGBMClassifier(
    n_estimators=300,
    max_depth=-1,              # Sem limite
    num_leaves=31,             # 2^max_depth - 1
    learning_rate=0.05,
    feature_fraction=0.8,
    bagging_fraction=0.8,
    bagging_freq=5,
    n_jobs=-1,
    random_state=42,
```

```
)
xgb_model.fit(X_train, y_train,
              eval_set=[(X_test, y_test)],
              callbacks=[lgb.early_stopping(50)])
```

10.4 CatBoost

Especializado em **variáveis categóricas** sem necessidade de encoding:

```
from catboost import CatBoostClassifier

cat_model = CatBoostClassifier(
    iterations=500,
    depth=6,
    learning_rate=0.05,
    cat_features=['city', 'category'],
    verbose=100
)
cat_model.fit(X_train, y_train)
```

10.5 Comparação

Algoritmo	Velocidade	Categóricas	Memória
XGBoost	Média	Não (precisa encoding)	Média
LightGBM	Rápida	Não	Baixa
CatBoost	Lenta	Sim (nativas)	Alta

Capítulo 11 — KNN — Sistema de Recomendação

11.1 O Problema

Recomendar filmes similares a um filme dado.

11.2 Features de Filmes

```
movies = pd.DataFrame(
    {'title': ['Matrix', 'Inception', 'Avengers', 'Titanic'],
    'action': [0.9, 0.95, 0.9, 0.2],
    'romance': [0.1, 0.3, 0.2, 0.95],
    'scifi': [0.95, 0.95, 0.7, 0.1],
    'drama': [0.3, 0.5, 0.3, 0.9]
)
```

11.3 Modelo

```
from sklearn.neighbors import NearestNeighbors
from sklearn.preprocessing import StandardScaler

X = movies.drop('title', axis=1)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

knn = NearestNeighbors(n_neighbors=2, metric='cosine')
knn.fit(X_scaled)

# Para "Matrix", recomendar vizinho
query = X_scaled[0].reshape(1, -1)
distances, indices = knn.kneighbors(query)
print(f"Para {movies.iloc[0]['title']}, recomendamos: {movies.iloc[indices[0][1]]['title
```

11.4 Similaridade Cosseno vs. Euclidiana

- **Euclidiana:** Distância absoluta. Boa para features em mesma escala.
- **Cosseno:** Ângulo entre vetores. Ignora magnitude, bom para "perfil".
- **Para recomendação de itens, cosseno quase sempre é melhor.**

Capítulo 12 — K-Means — Segmentação de Clientes

12.1 O Problema

Segmentar base de clientes de e-commerce em grupos acionáveis.

12.2 Dados Sintéticos RFM

```
import numpy as np
import pandas as pd

np.random.seed(42)
n = 1000

df = pd.DataFrame(
    'recency': np.random.exponential(30, n),        # Dias desde última compra
    'frequency': np.random.poisson(5, n),           # Compras/mês
    'monetary': np.random.exponential(100, n)       # Valor médio
)
```

12.3 Modelo

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score

scaler = StandardScaler()
X_scaled = scaler.fit_transform(df)

# Encontrar k ótimo
inertias = []
silhouettes = []
k_range = range(2, 11)
for k in k_range:
    km = KMeans(n_clusters=k, n_init=10, random_state=42)
    km.fit(X_scaled)
    inertias.append(km.inertia_)
```

```
    silhouettes.append(silhouette_score(X_scaled, km.labels_))

# Plot
import matplotlib.pyplot as plt
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
ax1.plot(K_range, inertias, 'bx-')
ax1.set_xlabel('k')
ax1.set_ylabel('Inertia')
ax1.set_title('Elbow Method')
ax2.plot(K_range, silhouettes, 'rx-')
ax2.set_xlabel('k')
ax2.set_ylabel('Silhouette')
ax2.set_title('Silhouette Method')
plt.tight_layout()
```

12.4 K-Means++ e Múltiplas Inicializações

```
# K-Means++ é default no scikit-learn
km = KMeans(n_clusters=4, init='k-means++', n_init=20, random_state=42)
df['cluster'] = km.fit_predict(X_scaled)

# Análise dos clusters
cluster_profile = df.groupby('cluster').mean()
print(cluster_profile)
```

12.5 Interpretação de Negócio

```
# Nomear clusters
cluster_names = {
    0: 'Champions',
    1: 'At-Risk',
    2: 'New Customers',
    3: 'Hibernating'
}
df['segment'] = df['cluster'].map(cluster_names)
```

Diagrama: K-Means Clustering

Capítulo 13 — PCA — Compressão de Imagens

13.1 Visualizando o Efeito

```
from sklearn.decomposition import PCA
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt

digits = load_digits()
X = digits.data  # 1797 imagens 8x8 = 64 features

# Variância explicada cumulativa
pca = PCA().fit(X)
cumsum = np.cumsum(pca.explained_variance_ratio_)
plt.plot(cumsum)
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.axhline(y=0.95, color='r', linestyle='--')
```

13.2 Reconstrução

```
# Manter 95% da variância
pca_95 = PCA(n_components=0.95)
X_reduced = pca_95.fit_transform(X)
X_reconstructed = pca_95.inverse_transform(X_reduced)

# Visualizar
fig, axes = plt.subplots(2, 10, figsize=(15, 3))
for i in range(10):
    axes[0, i].imshow(X[i].reshape(8, 8), cmap='gray')
    axes[0, i].axis('off')
    axes[1, i].imshow(X_reconstructed[i].reshape(8, 8), cmap='gray')
    axes[1, i].axis('off')
axes[0, 0].set_title('Original')
axes[1, 0].set_title(f'Reconstruído ({pca_95.n_components_} PCs)')
```

13.3 Taxa de Compressão

- **Original:** 64 features por imagem
- **Reduzido:** ~30 features (95% de variância)
- **Compressão:** ~2.1x sem perda perceptual significativa

13.4 Aplicações

- Pré-processamento para SVM/KNN (acelera treino).
- Visualização 2D/3D de datasets complexos.
- Remoção de ruído.
- Compressão de imagens.

Diagrama: PCA Dimensionality Reduction

Capítulo 14 — DBSCAN — Detecção de Fraude

14.1 Por que DBSCAN para Fraude?

- Não precisa especificar K.
- Identifica outliers como **ruído** (label -1).
- Encontra clusters de forma arbitrária.

14.2 Implementação

```
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

# Credit Card Fraud (dataset Kaggle)
df = pd.read_csv('creditcard.csv')
X = df.drop('Class', axis=1)
y = df['Class']

# Padronizar (Amount e Time têm escalas muito diferentes)
scaler = StandardScaler()
```

```
# Scaled = scaler.fit_transform(X)

# DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=10, n_jobs=-1)
clusters = dbscan.fit_predict(X_scaled)

# Anomalias = label -1
n_anomalies = (clusters == -1).sum()
print(f"Anomalias detectadas: {n_anomalies} ({n_anomalies/len(X)*100:.2f}%)")

# Comparar com fraudes reais
fraudes_reais = y.sum()
fraudes_detectadas = ((clusters == -1) & (y == 1)).sum()
print(f"Fraudes detectadas: {fraudes_detectadas} / {fraudes_reais}")
```

14.3 Tuning de ε e min_samples

Use o **k-distance plot**:

```
from sklearn.neighbors import NearestNeighbors

neighbors = NearestNeighbors(n_neighbors=5)
neighbors.fit(X_scaled)
distances, _ = neighbors.kneighbors(X_scaled)
distances = np.sort(distances[:, -1])

plt.plot(distances)
plt.xlabel('Pontos ordenados')
plt.ylabel('Distância ao 5º vizinho')
plt.axhline(y=0.5, color='r', linestyle='--')
```

O "cotovelo" no gráfico indica o bom valor de ε.

Capítulo 15 — Isolation Forest — Outliers em Séries Temporais

15.1 O Algoritmo

Isolation Forest (Liu et al., 2008) isola anomalias por meio de **partições aleatórias**. Anomalias são isoladas em **menos splits**.

15.2 Implementação

```
from sklearn.ensemble import IsolationForest

# Série temporal de requisições HTTP
np.random.seed(42)
n = 10000
X = np.random.normal(0, 1, (n, 2))
# Adicionar outliers
X[500] += np.random.normal(5, 1, (50, 2))

iso_forest = IsolationForest(contamination=0.01, random_state=42)
labels = iso_forest.fit_predict(X)

anomalies = X[labels == -1]
print(f"Anomalias: {len(anomalies)} ({len(anomalies)/n*100:.2f}%)")
```

15.3 Vantagens

- Tempo linear: O(n).
- Não usa distância (bom em alta dimensão).
- Funciona bem para séries temporais.

15.4 Caso de Uso Real: API Monitoring

```
# Detectar picos anômalos de latência
latencies = np.array(df['latency_ms'].).reshape(-1, 1)
iso = IsolationForest(contamination=0.001)
```

```
df['is_anomaly'] = iso.fit_predict(latencies)

# Alerta
anomalies = df[df['is_anomaly'] == -1]
if len(anomalies) > 0:
    send_alert(f"Detectados {len(anomalies)} picos anômalos")
```

Capítulo 16 — SVM — Classificação de Texto

16.1 O Algoritmo

SVM encontra o **hiperplano de margem máxima** que separa as classes.

Para dados não-linearmente separáveis, usa o **kernel trick**: - **Linear:** $K(x, y) = x^T y$ - **RBF (Gaussiano):** $K(x, y) = \exp(-\gamma|x - y|^2)$ - **Polinomial:** $K(x, y) = (x^T y + c)^d$

16.2 Aplicação: Classificação de Notícias

```
from sklearn.svm import SVC
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer

news = fetch_20newsgroups(subset='train', categories=['alt.atheism', 'soc.religion.ch
X, y = news.data, news.target

tfidf = TfidfVectorizer(max_features=5000)
X_tfidf = tfidf.fit_transform(X)

svm = SVC(kernel='linear', C=1.0)
svm.fit(X_tfidf, y)
score = svm.score(X_tfidf, y)
```

16.3 Tuning de C e gamma

```
from sklearn.model_selection import GridSearchCV

params = {
    'C': [0.1, 1, 10],
    'gamma': ['scale', 'auto', 0.1, 1],
    'kernel': ['rbf', 'linear']
}
grid = GridSearchCV(SVC(), params, cv=5, n_jobs=-1)
grid.fit(X_tfidf, y)
print(grid.best_params_)
```

Capítulo 17 — Naive Bayes — Filtro de Spam (Clássico)

17.1 O Algoritmo

Aplica o **Teorema de Bayes** com a suposição (ingênua) de independência entre features:

$$P(c|x) = \frac{P(x|c) P(c)}{P(x)}$$

17.2 Variantes

- **MultinomialNB:** Para contagens (ex: frequência de palavras).
- **BernoulliNB:** Para features binárias (palavra presente/ausente).
- **GaussianNB:** Para features contínuas (assume normalidade).

17.3 Código

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
X_counts = vectorizer.fit_transform(df['message'])
```

```
# Treinamento
nb = MultinomialNB(alpha=1.0)  # alpha = Laplace smoothing
nb.fit(X_counts, y)
```

17.4 Vantagens e Limitações

Vantagens: Treino ultra-rápido - Funciona bem com poucos dados - Interpretabilidade direta - Bom baseline para texto

Limitações: Suposição de independência raramente é válida - Não captura interações entre features - Pode ser superado por modelos mais complexos

---

Capítulo 18 — Validação Cruzada e Hyperparameter Tuning

18.1 Cross-Validation

```
from sklearn.model_selection import cross_val_score, KFold, StratifiedKFold

# K-Fold padrão
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(model, X, y, cv=kfold, scoring='accuracy')
print(f"CV Accuracy: {scores.mean():.3f} ± {scores.std():.3f}")

# Stratified (mantém proporção de classes)
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(model, X, y, cv=skf, scoring='f1')
```

18.2 Grid Search

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [100, 200, 500],
    'max_depth': [3, 5, 10, None],
    'min_samples_split': [2, 5, 10],
```

```
)

grid = GridSearchCV(
    RandomForestClassifier(random_state=42),
    param_grid,
    cv=5,
    scoring='f1',
    n_jobs=-1,
    verbose=1
)
grid.fit(X_train, y_train)
print(grid.best_params_)
print(grid.best_score_)
```

18.3 Random Search

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint, uniform

param_dist = {
    'n_estimators': randint(50, 500),
    'max_depth': randint(3, 20),
    'min_samples_split': randint(2, 20),
    'learning_rate': uniform(0.01, 0.3)
}

random_search = RandomizedSearchCV(
    XGBClassifier(),
    param_dist,
    n_iter=100,
    cv=3,
    scoring='f1',
    n_jobs=-1,
    random_state=42,
)
random_search.fit(X_train, y_train)
```

18.4 Optuna (Bayesian Optimization)

```
import optuna

def objective(trial):
```

```
    params = {
        'n_estimators': trial.suggest_int('n_estimators', 100, 1000),
        'max_depth': trial.suggest_int('max_depth', 3, 10),
        'learning_rate': trial.suggest_float('learning_rate', 0.01, 0.3, log=True),
        'subsample': trial.suggest_float('subsample', 0.6, 1.0),
        'colsample_bytree': trial.suggest_float('colsample_bytree', 0.6, 1.0)
    }
    model = XGBClassifier(**params)
    scores = cross_val_score(model, X_train, y_train, cv=5, scoring='f1')
    return scores.mean()

study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=50, show_progress_bar=True)
print(study.best_params)
print(study.best_value)
```

Capítulo 19 — Pipelines e MLflow

19.1 Pipelines Reproduzíveis

```
from sklearn.pipeline import Pipeline

pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=0.95)),
    ('classifier', RandomForestClassifier(n_estimators=200))
])

pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)

# Salvar pipeline inteiro
import joblib
joblib.dump(pipeline, 'models/pipeline_v1.pkl')
```

19.2 ColumnTransformer (Heterogêneo)

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
```

```
numeric_features = ['age', 'salary']
categorical_features = ['city', 'gender']

preprocessor = ColumnTransformer(
    ('num', StandardScaler(), numeric_features),
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
)

pipeline = Pipeline(
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier())
)
```

19.3 MLflow — Tracking de Experimentos

```
import mlflow
import mlflow.sklearn

mlflow.set_experiment('customer-churn')

with mlflow.start_run():
    # Hiperparâmetros
    n_estimators = 200
    max_depth = 14
    mlflow.log_param('n_estimators', n_estimators)
    mlflow.log_param('max_depth', max_depth)

    # Modelo
    model = RandomForestClassifier(n_estimators=n_estimators, max_depth=max_depth)
    model.fit(X_train, y_train)

    # Métricas
    y_pred = model.predict(X_test)
    from sklearn.metrics import f1_score, roc_auc_score
    mlflow.log_metric('f1', f1_score(y_test, y_pred))
    mlflow.log_metric('auc', roc_auc_score(y_test, model.predict_proba(X_test)[:, 1]))

    # Modelo
    mlflow.sklearn.log_model(model, 'model')

    # Artefatos
    mlflow.log_artifact('confusion_matrix.png')
```

```
mlflow ui  # http://localhost:5000
```

Capítulo 20 — Deploy: Do Notebook à API

20.1 Salvando o Modelo

```
import joblib

joblib.dump(model, 'model.pkl')

# Carregar depois
model = joblib.load('model.pkl')
```

20.2 FastAPI

```
# main.py
from fastapi import FastAPI
from pydantic import BaseModel
import joblib
import numpy as np

app = FastAPI()
model = joblib.load('model.pkl')

class PredictionRequest(BaseModel):
    features: List

@app.post("/predict")
def predict(request: PredictionRequest):
    X = np.array(request.features).reshape(1, -1)
    prediction = model.predict(X)
    probability = model.predict_proba(X).max()
    return {
        "prediction": int(prediction[0]),
        "probability": float(probability)
    }
```

```
|
+ Run: uvicorn main:app --reload
```

20.3 Docker

```
FROM python:3.11-slim
|
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
|
COPY . .
|
EXPOSE 8000
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

20.4 Batch Inference

```
import pandas as pd
import joblib
|
model = joblib.load('model.pkl')
df_new = pd.read_parquet('data/new_data.parquet')
predictions = model.predict(df_new)
|
df_new['prediction'] = predictions
df_new.to_parquet('predictions.parquet')
```

Capítulo 21 — Monitoramento em Produção (Drift)

21.1 Data Drift

A distribuição dos dados de entrada muda ao longo do tempo.

```
from evidently import ColumnDriftMetric
from evidently.report import Report

report = Report(metrics=[ColumnDriftMetric(column_name='age')])
report.run(reference_data=df_train, current_data=df_prod)
report.save_html('drift_report.html')
```

21.2 Concept Drift

A relação entre X e y muda (modelo fica desatualizado).

21.3 Métricas em Produção

```
# Capturar feedback
predictions = model.predict(X_prod)
true_labels = get_labels_delayed()

# Comparar com baseline
from sklearn.metrics import f1_score
prod_f1 = f1_score(true_labels, predictions)

if prod_f1 < 0.85 * baseline_f1:
    trigger_retraining()
```

Capítulo 22 — MLOps: O Ciclo de Vida Completo

22.1 As Fases

```
1. Data Engineering → 2. Model Development → 3. Model Deployment
                                                    ↓
                    4. Monitoring → 5. Retraining (Loop)
```

22.2 Stack Moderna

Camada	Ferramentas
Orquestração	Airflow, Prefect, Dagster
Feature Store	Feast, Tecton
Tracking	MLflow, Weights & Biases, Neptune
Model Serving	BentoML, Seldon, Triton, KServe
Monitoring	Evidently, Arize, Whylabs
CI/CD	GitHub Actions, GitLab CI
Infra	Kubernetes, Docker, Terraform

22.3 CI/CD para ML

```
# .github/workflows/train.yml
name: Train Model
on:
  push:
    paths: ['data/**', 'src/**']

jobs:
  train:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Setup Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.11'
      - name: Install deps
        run: pip install -r requirements.txt
      - name: Train
        run: python src/train.py
      - name: Validate
        run: python src/validate.py
      - name: Deploy
        if: success()
```

```
    run: |
      aws s3 cp model.pkl s3://my-bucket/models/
```

22.4 Integração com Nexus Affil'IA'te

No ecossistema Nexus, **MLOps** é gerenciado pelo **SHO (Sistema Híbrido de Orquestração)**: - **Nível S2 (Supervisionado):** Modelo treina, humano revisa amostral. - **Nível S3 (Autônomo):** Retraining automático com guardrails. - **S4 (Federado):** Modelos sincronizam entre nós federados.

Capítulo 23 — Algoritmos para Time Series

23.1 Especificidades

- **Ordem temporal:** Não pode shuffle.
- **Sazonalidade:** Padrões cíclicos (dia, semana, ano).
- **Autocorrelação:** Observações dependentes.
- **Não-estacionariedade:** Média/variância mudam.

23.2 Modelos Clássicos

```
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX

# ARIMA(p, d, q)
model = ARIMA(series, order=(2, 1, 1))
fitted = model.fit()
forecast = fitted.forecast(steps=30)

# SARIMAX com sazonalidade
model = SARIMAX(series, order=(2, 1, 1), seasonal_order=(1, 1, 1, 12))
```

23.3 Prophet (Facebook)

```
from prophet import Prophet

df_prophet = df.rename(columns={'date': 'ds', 'value': 'y'})
model = Prophet(yearly_seasonality=True, weekly_seasonality=True)
model.fit(df_prophet)
future = model.make_future_dataframe(periods=30)
forecast = model.predict(future)
```

23.4 Deep Learning para Séries

- **LSTM/GRU:** Captura dependências de longo prazo.
- **Transformer-based:** Informer, Autoformer, PatchTST.
- **N-BEATS:** Especializado em forecasting.

Capítulo 24 — Algoritmos para NLP Clássico

24.1 Pipeline Clássico

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression

vectorizer = TfidfVectorizer(
    max_features=10000,
    ngram_range=(1, 2),
    min_df=2,
    max_df=0.95,
    stop_words='english'
)
X = vectorizer.fit_transform(texts)
model = LogisticRegression(max_iter=1000)
model.fit(X, labels)
```

24.2 Técnicas Clássicas

- **Bag-of-Words (BoW):** Contagem de palavras.
- **TF-IDF:** Frequência ponderada por inverso da frequência em documentos.
- **Word2Vec/GloVe:** Embeddings pré-treinados.
- **Topic Modeling:** LDA, NMF.

24.3 Quando Usar vs. Transformers

Critério	Clássico	Transformer
Dataset pequeno		(overfit)
Velocidade		
Interpretabilidade		
Performance em texto longo		
Estado da arte		

Capítulo 25 — Quando NÃO Usar ML

25.1 As 7 Mortes do Projeto ML

1. **Regras claras funcionam:** Se dá para escrever `if/else`, faça.
2. **Dados insuficientes:** <100 amostras por classe.
3. **Causalidade confundida com correlação:** ML não descobre causalidade.
4. **Stakeholders não entendem o modelo:** Problema de gestão.
5. **Custo de erro é catastrófico:** Medicina crítica, aviação.
6. **Mudança de conceito rápida:** Modelo precisa retreinar a cada hora.
7. **Métrica de negócio vs. métrica de ML:** Foco errado.

25.2 O Teste "É ML?"

Faça estas perguntas: - O problema é uma **otimização** ou uma **regressão/classificação**? - Há **dados históricos** suficientes? - O **valor** de melhorar a baseline justifica o investimento? - A métrica é **mensurável** e **disponível**?

Se 3+ respostas forem sim, considere ML.

25.3 A Regra de Ouro

> "Comece com o modelo mais simples (baseline). Só escale de complexidade se a métrica exigir."

Linha de base: **Regressão Logística** ou **Árvore de Decisão**. Se 80% do valor sai dela, pare aí.

Glossário

Termo	Definição
Bagging	Treinar modelos em subamostras (Random Forest).
Boosting	Treinar modelos sequencialmente corrigindo erros.
Cross-Validation	Avaliação usando múltiplos folds.
Drift	Mudança na distribuição dos dados ao longo do tempo.
Ensemble	Combinação de modelos.
Hyperparameter	Configuração externa ao modelo.
MLOps	Operações de ML em produção.
Pipeline	Sequência encadeada de transformações.
TF-IDF	Term Frequency-Inverse Document Frequency.

Termo	Definição
Train/Test Split	Divisão do dataset em treino e teste.

Referências

1. **Géron, A.** (2022). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly.
2. **Hastie, T., Tibshirani, R., Friedman, J.** (2009). The Elements of Statistical Learning. Springer.
3. **Chen, T., Guestrin, C.** (2016). XGBoost: A Scalable Tree Boosting System. KDD.
4. **Ke, G. et al.** (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. NeurIPS.
5. **Prokhorenkova, L. et al.** (2018). CatBoost: Unbiased Boosting with Categorical Features. NeurIPS.
6. **Breiman, L.** (2001). Random Forests. Machine Learning.
7. **Liu, F. et al.** (2008). Isolation Forest. ICDM.
8. **Ester, M. et al.** (1996). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. KDD.
9. **scikit-learn Documentation** (2024). https://scikit-learn.org.
10. **MLflow Documentation** (2024). https://mlflow.org.

Fim do Volume 03

> "O melhor modelo é o que resolve o problema. Não o mais complexo, não o mais novo — o que resolve." — Nexus Affil'IA'te MMN_IA

Imagem de Encerramento

© 2026 Nexus Affil'IA'te MMN_IA · Curso Universo IA · Coletânea Técnica