ANA RAQUEL

PÓS TECH

DATA ANALYTICS

# DEPLOY DE APLICAÇÕES

# AULA 04

**SUMÁRIO**

## O QUE VEM POR AÍ?

Você aprendeu a importância de garantir a consistência dos dados, agora chegou a hora de escolher a cereja do bolo de um projeto de Machine Learning: o modelo preditivo!

Apesar de ser um momento tão desejado, passamos muito tempo antes de colocar as mãos no modelo, analisando e pré-processando dados. Será que a escolha do modelo também é uma etapa minuciosa? Vem comigo nesse capítulo e vamos conversar sobre essa etapa essencial do pipeline!

## HANDS ON

Chegou o momento de ver, na prática, a escolha do melhor modelo de Machine Learning para classificar clientes bons e maus pagadores. Antes de testar os modelos, aplicaremos as pipelines criadas na aula anterior nas bases de treino e teste para, em seguida, testar os modelos e avaliá-los com as métricas de desempenho.

## SAIBA MAIS

### CROSS INDUSTRY STANDARD PROCESS FOR DATA MINING (CRISP-DM)

Cross Industry Standard Process for Data Mining (CRISP-DM) é uma metodologia para solucionar problemas em Ciência dos Dados. Podemos listar algumas fases para a implementação da metodologia:

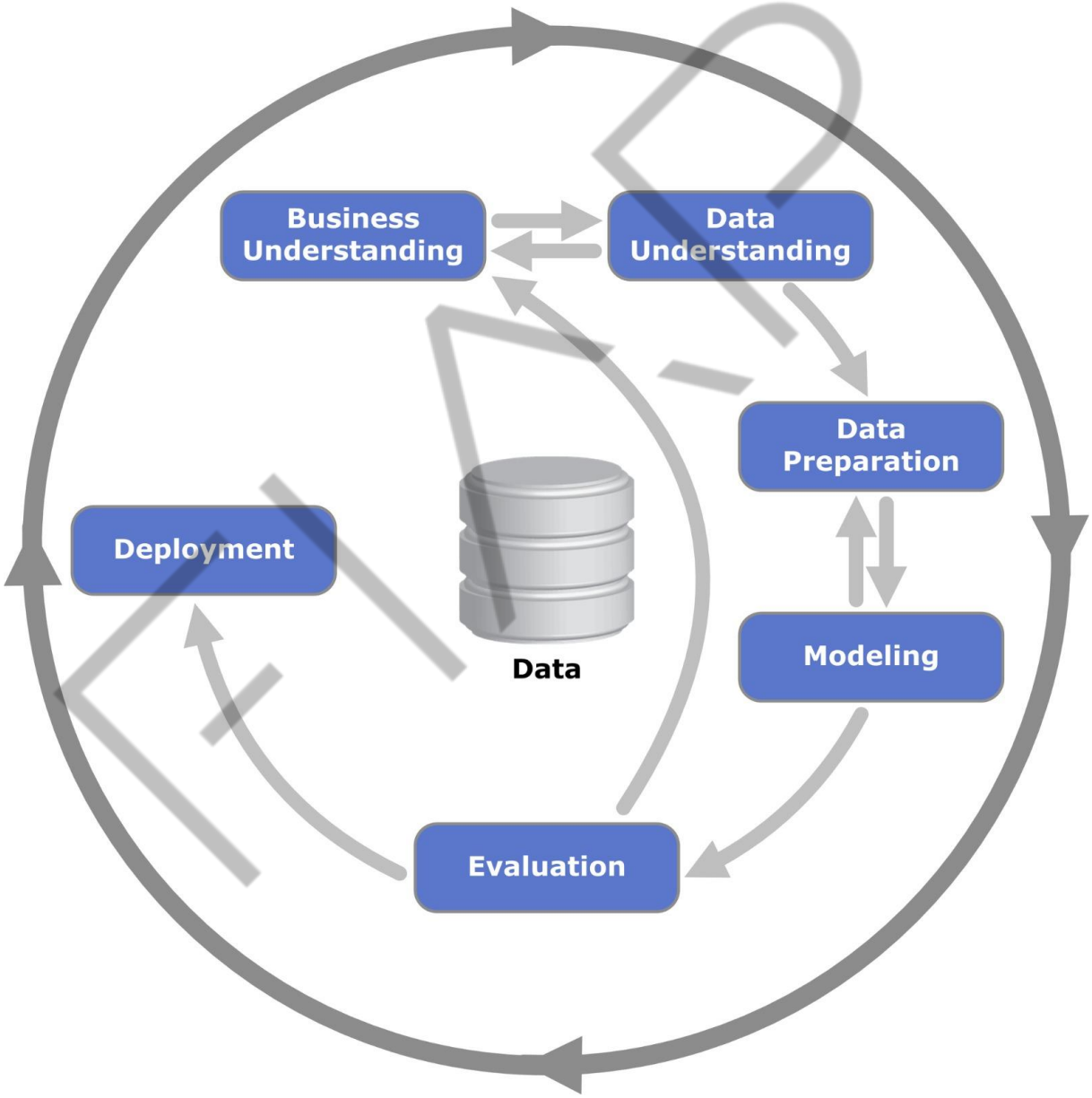


Figura 1 - Modelo do CRISP-EDM

Fonte: Paulo Domingues (2020)

- **Entendimento do negócio:** essa é a etapa mais importante, pois definirá todos os próximos passos da metodologia. Ela consiste em identificar os objetivos do negócio, sem vieses inconscientes.
- **Entendimento dos dados:** nessa etapa, aplicamos a famosa análise exploratória dos dados, para descobrir insights e comprovar algumas hipóteses realizadas sobre o negócio.
- **Preparação dos dados:** nessa etapa, temos o famoso pré-processamento dos dados, normalizando as escalas, limpando nulos, transformando variáveis do tipo texto para tipo numérico e assim por diante. Aqui, temos todas as demais características de feature engineering, que você já aprendeu em aula.
- **Modelagem:** essa é a etapa dessa aula! Na modelagem, nós treinamos e configuramos vários testes de modelos de Machine Learning, até chegar no ideal para solucionar o tipo de problema de negócio estudado e preparado nas etapas anteriores.
- **Avaliação do modelo:** essa etapa é decorrente da etapa anterior, sendo o momento de validar a eficácia do modelo. Será que o modelo criado atende todas as necessidades definidas inicialmente?

**Deployment:** é a etapa da nossa próxima aula, onde colocaremos o modelo disponível para os(as) stakeholders! Como será a entrega do seu modelo: um relatório? Em um software? Em uma aplicação?

Perceba que, durante as aulas dessa disciplina, estamos aplicando as etapas dessa metodologia para concluir a entrega de um projeto de Machine Learning! Seguir esses passos pode ser uma boa escolha para obter sucesso na entrega do projeto.

Focaremos nas duas penúltimas etapas: a modelagem do modelo e avaliação! Vamos lá?

**PREPARANDO AS BIBLIOTECAS DE VALIDAÇÃO**

Para compreender melhor essa etapa de escolha do modelo, vamos continuar utilizando os exemplos do case das videoaulas, ok?

Depois de criar as pipelines para tratar os dados, vamos importar algumas bibliotecas necessárias para a construção do modelo. Para analisar se os modelos estão performando bem, precisamos importar algumas bibliotecas essenciais:

- **metrics** do Sklearn;
- **classification_report** do Sklearn;
- **plot_confusion_matrix** do Sklearn;
- **roc_auc_score** e **plot_roc_curve** do Sklearn;
- **stats** do Scipy.

Existe uma métrica bem específica utilizada no mercado financeiro como um dos indicadores de eficiência de modelos de credit scoring: a **KS (Teste Kolmogorov-Smirnov)**, que mede a distância entre as probabilidades de classes. Nesse caso, estamos querendo medir a probabilidade de um cliente ser bom ou mau pagador. Para mensurar seu desempenho, quanto maior for o valor de KS, melhor! O mercado considera um bom modelo aquele que apresenta um valor de KS igual ou superior a 0.3. Isso significa que o modelo está conseguindo distinguir bem as classes.

Se o problema de negócio aplicado estiver fora do contexto do mercado financeiro, você pode optar por utilizar métricas como a **curva ROC** e a **pontuação ROC AUC** como medidas similares a KS, exemplificada acima. Também é muito importante avaliar as medidas contidas no **classification_report.**

Quando estamos avaliando escopo de um classificador (nesse caso, o quão bom o seu modelo é em distinguir os padrões de classes diferentes), é necessário analisar se você procura, realmente, classificar com precisão todas as classes contidas no modelo ou se você necessita que o modelo seja preciso apenas em uma das classes contidas na target.

Para saber se seu modelo está performando bem em todas as classes, analise o desempenho principalmente da métrica **recall** (sensibilidade ou Taxa de Verdadeiros Positivos) e **F1-Score** (média harmônica da precisão e do recall). Aqui, vale a pena também tomar cuidado com o overfitting!

Na etapa de validação, também é interessante aplicar a **validação cruzada**, pois com ela é possível testar o modelo em vários k-folds (conjunto de dados aleatórios da base de treinamento) e tirar uma média entre todos os testes e validar o desempenho.

## CRIANDO UMA FUNÇÃO PARA ESCOLHER O MELHOR MODELO

Nesse momento, partimos para a escolha do melhor modelo para resolver nosso problema de negócio. Mas por onde começar?

Aqui é necessário identificar qual é o **tipo de aprendizado de máquina** que está pensando criar no projeto, para iniciar os testes dentro dos algoritmos disponíveis para esse tipo de aprendizado. Abaixo, segue um exemplo de um fluxo de escolha de algoritmos:

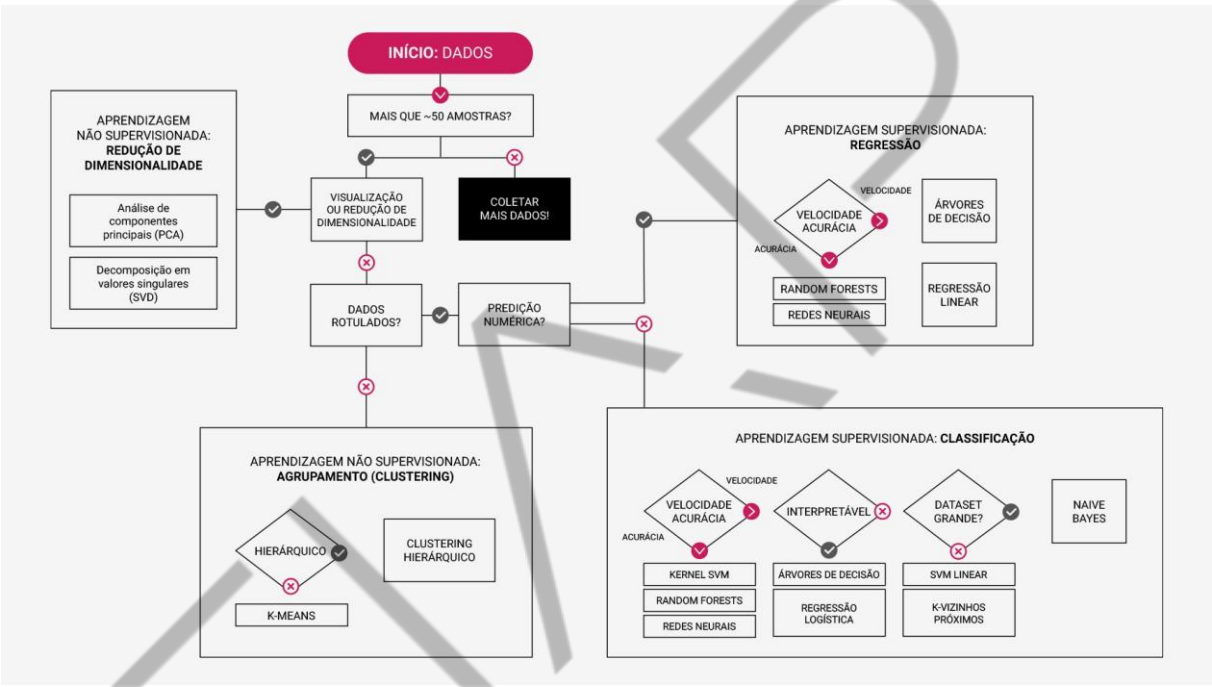


Figura 2 - Tipos de aprendizado de máquina
Fonte: Gabriel R. Schleder Adalberto Fazzio (2021)

Para exemplificar essa etapa fundamental, vamos criar uma função dentro do case explorado nas aulas para testarmos qual será o melhor modelo de Machine Learning na classificação de clientes bons e maus pagadores.

Criaremos uma função nomeada como **“roda_modelo”,** que irá receber como parâmetro um modelo de classificação de Machine Learning e nos retornar algumas métricas de validação como a pontuação AUC, a métrica KS, a matriz de confusão e a curva ROC.

SEED = 1561651

```
def roda_modelo(modelo):


    # Treinando modelo com os dados de treino
```

Aqui, nós faremos o fit com os dados de treino e, para isso, vamos usar o comando:

```
    modelo.fit(X_treino, y_treino)
```

Em seguida, podemos estimar a probabilidade das nossas classes (0 e 1) em X_test utilizando o método predict_proba e armazenar na variável prob_predic. Com esse método, as estimativas retornadas para todas as classes são ordenadas pelo rótulo das classes. No nosso caso, primeiro teremos a probabilidade da classe 0 e, em seguida, da classe 1. Para definir qual é a classe, a maior probabilidade é considerada:

```
    # Calculando a probabilidade e calculando o AUC
    prob_predic = modelo.predict_proba(X_teste)
```

Para deixar os resultados esteticamente agradáveis, vou fazer um print colocando a palavra “Resultados” e formatar para que saia o nome do modelo usado:

```
    print(f"\n------------------------------Resultados {modelo}------------------------------\n")
```

Em seguida, vamos calcular o valor de AUC. Para isso, usamos o método roc_auc_score e passamos o y_test e a variável que criamos para armazenar o predict_proba: prob_predic.

Entre colchetes, especificamos com :,1 para pegar a probabilidade da classe 1:

```
auc = roc_auc_score(y_teste, prob_predic[:,1])
print(f"AUC {auc}")
```

Agora, podemos aplicar a estatística que expliquei anteriormente: o teste KS. Então, vamos separar a probabilidade de ser bom e mau e calcular o KS. Para isso, aplicamos o **sort** do Numpy em **modelo.predict_proba(X_test)** e especificamos que **data_bom** é a classe 0 e **data_mau** é a classe 1.

Em seguida, calculamos o ks com stats.ks_2samp e passamos a ele data_bom e data_mau.

```
# Separando a probabilidade de ser bom e mau e calculando o KS
#métrica KS: probabilidade de um cliente ser classificado como bom ou mau.
data_bom = np.sort(modelo.predict_proba(X_teste)[:, 0])
data_mau = np.sort(modelo.predict_proba(X_teste)[:, 1])
kstest = stats.ks_2samp(data_bom, data_mau)
```

Podemos imprimir o resultado:

```
print(f"Métrica KS: {kstest}")
```

Agora, vamos adicionar mais métricas, começando pela matriz de confusão. Vou começar adicionando um título:

```
print("\nConfusion Matrix\n")
```

```
# Criando matriz de confusão
```

Para criar uma matriz normalizada vamos fazer o seguinte:

```
fig, ax = plt.subplots(figsize=(7,7))
matriz_confusao = plot_confusion_matrix(modelo, X_teste, y_teste, normalize='true',
                                        display_labels=['Bom pagador', 'Mau pagador'],
                                        ax=ax, cmap=plt.cm.Blues)
ax.set_title("Matriz de Confusão\n Normalizada", fontsize=16, fontweight="bold")
ax.set_xlabel("Label predita", fontsize=18)
ax.set_ylabel("Label verdadeira", fontsize=18)
plt.grid(False)
plt.show(matriz_confusao)
```

Podemos fazer a predição dos dados de teste e calcular o classification report. **zero_division** define o valor a ser retornado quando houver uma divisão zero.

```
# Fazendo a predição dos dados de teste e calculando o classification report
predicao = modelo.predict(X_teste)
print("\nClassification Report\n")
print(classification_report(y_teste, predicao, zero_division=0))
```

A última métrica que vamos pedir para o modelo executar é a curva ROC:

```
print("\nRoc Curve\n")
metrics.plot_roc_curve(modelo, X_teste, y_teste)
```

Função criada! Agora, o próximo passo é testar os modelos. Como estamos trabalhando com um problema de classificação binário, que tal começar pela **regressão logística**?

Para dar um “refresh” na sua memória, vamos relembrar como funciona o modelo? Basicamente, esse tipo de modelo linear para classificação tem o objetivo de prever a probabilidade de pertencimento de uma instância a uma certa classe específica (por exemplo, qual é a probabilidade de um cliente ser um mau pagador). Se a probabilidade estimada for maior que o limiar determinado pelo modelo (50%), então o modelo prevê que a instância analisada pertence a classe de clientes maus pagadores.

Vamos importar o modelo e armazenar em uma variável “modelo_logistico”:

```
from sklearn.linear_model import LogisticRegression
modelo_logistico = LogisticRegression()
```

Aplicando a nossa função criada anteriormente:

```
roda_modelo(modelo_logistico)
```

Vamos analisar os resultados?

```
----------Resultados LogisticRegression()----------
AUC 0.6565409896078054
Métrica KS: Ks_2sampResult(statistic=0.04132045811812261, pvalue=4.962147563460462e-07)
Confusion Matrix
```

Figura 3 – Resultados LogisticRegression.
Fonte: Notas de aula Machine Learning (2023)

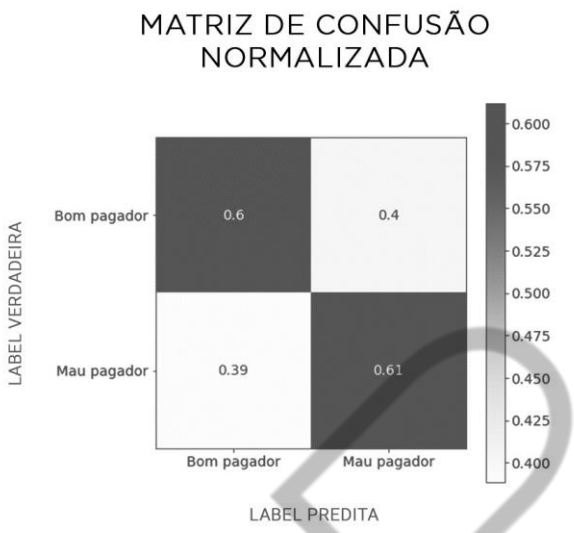


Figura 4 – Matriz de confusão LogisticRegression.

Fonte: Notas de aula Machine Learning (2023)

Classification Report

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.61 | 0.60 | 0.60 | 4453 |
| 1 | 0.60 | 0.61 | 0.61 | 4453 |
| accuracy | | | 0.60 | 8906 |
| macro avg | 0.60 | 0.60 | 0.60 | 8906 |
| weighted avg | 0.60 | 0.60 | 0.60 | 8906 |

Roc Curve

Figura 5 – Classification Report LogisticRegression.

Fonte: Notas de aula Machine Learning (2023)

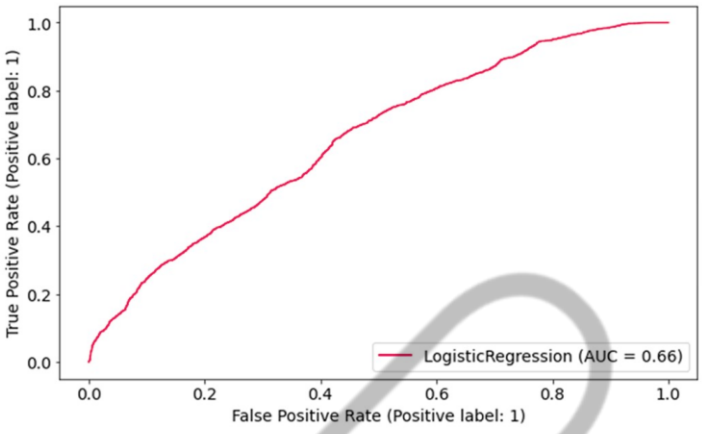


Figura 6 – Curva ROC LogisticRegression.
Fonte: Notas de aula Machine Learning (2023)

Analisando os resultados do modelo, consigo fazer algumas considerações. Vamos começar pela métrica KS, que está baixíssima **(0.04)**. A **matriz de confusão** mostra que o modelo está acertando **0.6** dos maus pagadores e **0.6** dos bons pagadores. Por outro lado, temos **0.4** de falso positivo e falso negativo. Acredito que podemos melhorar esses resultados de verdadeiros positivos, já que o modelo está acertando pouco. Analisando o **classification report,** temos valores de apenas **0.60** para precision, recall e f1-score.

Observem também como a curva roc está ruim, pois o **ideal é que ela seja mais voltada para o canto esquerdo superior, indicando que o modelo está separando bem as classes**.

Bem, já descartamos o modelo de regressão logística, mas que tal tentarmos com o modelo de árvore de decisão? Árvores de decisão são classificadores não lineares (ao contrário da regressão logística) e não exigem que os dados sejam linearmente separáveis. Acredito que esse modelo possa funcionar melhor, vamos testar?

Vamos importar o modelo e armazenar em uma variável “modelo_tree”:

```
from sklearn.tree import DecisionTreeClassifier

modelo_tree = DecisionTreeClassifier()

roda_modelo(modelo_tree)
```

Analisando os resultados:

```
----------Resultados DecisionTreeClassifier()----------
AUC 0.7921672266402559
Métrica KS: Ks_2sampResult(statistic=0.3648102402874467, pvalue=0.0)
Confusion Matrix
```

Figura 7 – Resultados DecisionTreeClassifier.
Fonte: Notas de aula Machine Learning (2023)

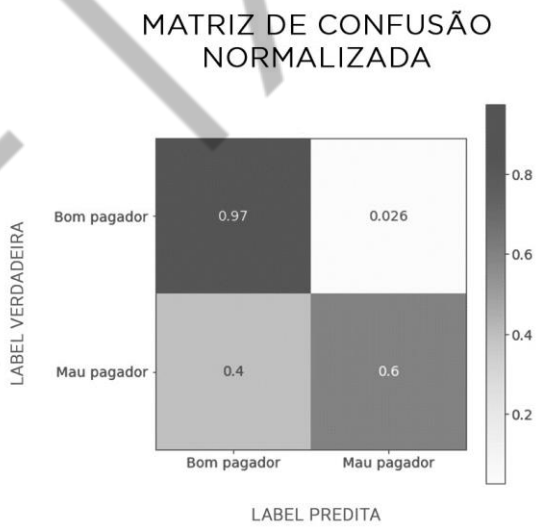


Figura 8 – Matriz de confusão DecisionTreeClassifier.
Fonte: Notas de aula Machine Learning (2023)

Classification Report

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.71 | 0.97 | 0.82 | 4453 |
| 1 | 0.96 | 0.60 | 0.74 | 4453 |
| accuracy | | | 0.79 | 8906 |
| macro avg | 0.83 | 0.79 | 0.78 | 8906 |
| weighted avg | 0.83 | 0.79 | 0.78 | 8906 |

Figura 9 – Classification Report DecisionTreeClassifier.
Fonte: Notas de aula Machine Learning (2023)

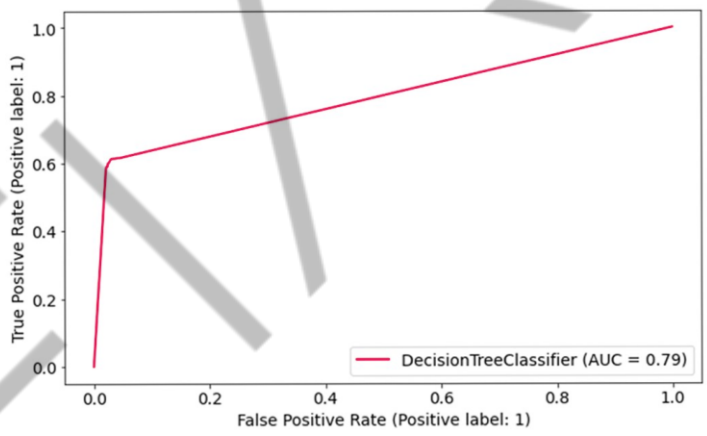


Figura 10 – Curva ROC DecisionTreeClassifier.
Fonte: Notas de aula Machine Learning (2023)

Agora temos um resultado melhor! O ROC AUC score está com **0.79**. O teste KS teve um valor de **0.36**, o que é **considerado bom em credit scoring**. A matriz de confusão também melhorou, mas agora nós temos um valor bem alto somente para a classe de bons pagadores (**0.97**). O falso positivo continua alto (**0.4**) e os maus pagadores positivos têm apenas **0.6**.

Perceba, com esses resultados, que o modelo está tendo dificuldade em prever se o cliente é um mau pagador. Isso pode gerar a liberação de crédito para clientes que não deveriam tê-la. No classification report, temos um recall de apenas **0.60** para classe 1, mostrando que o modelo vai identificar corretamente apenas 60% de maus pagadores. A curva roc deu uma boa melhorada, mas será que conseguimos resultados ainda melhores?

Vamos testar com o modelo GradientBoostingClassifier! Para isso, vamos importar o modelo e armazenar em uma variável “modelo_xgb”:

```
from sklearn.ensemble import GradientBoostingClassifier

modelo_xgb = GradientBoostingClassifier()

roda_modelo(modelo_xgb)
```

Vamos analisar os resultados:

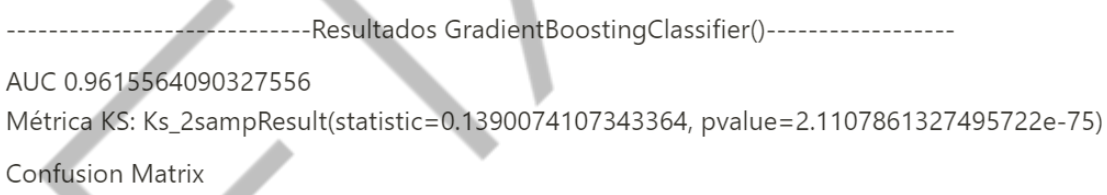


Figura 11 – Resultados GradientBoostingClassifier.
Fonte: Notas de aula Machine Learning (2023)

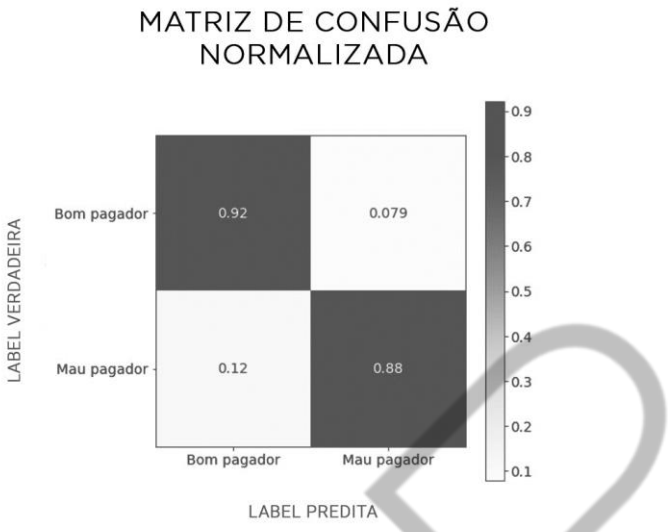


Figura 12 – Matriz de confusão GradientBoostingClassifier.

Fonte: Notas de aula Machine Learning (2023)

Classification Report

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.89 | 0.92 | 0.90 | 4453 |
| 1 | 0.92 | 0.88 | 0.90 | 4453 |
| accuracy | | | 0.90 | 8906 |
| macro avg | 0.90 | 0.90 | 0.90 | 8906 |
| weighted avg | 0.90 | 0.90 | 0.90 | 8906 |

Figura 13 – Classification Report GradientBoostingClassifier.

Fonte: Notas de aula Machine Learning (2023)

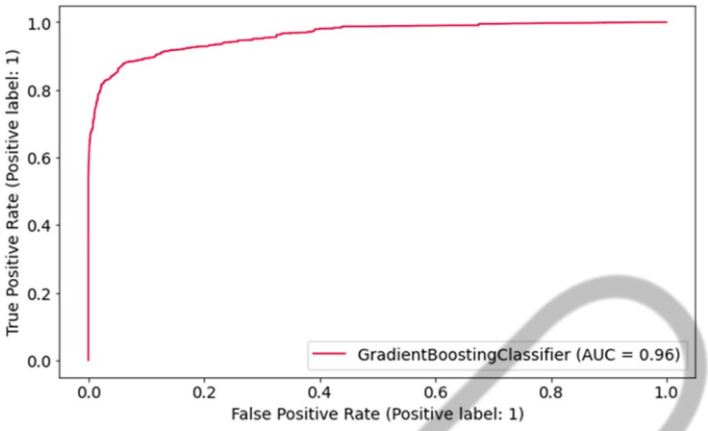


Figura 14 – Curva ROC GradientBoostingClassifier.
Fonte: Notas de aula Machine Learning (2023)

Agora temos resultados bem melhores! Nosso ROC AUC score está com **0.96**. O KS caiu para **0.13**, mas, por outro lado, a matriz de confusão está muito boa! Temos um acerto de **0.88** para maus-pagadores e **0.92** para bons pagadores. Além disso, os valores de falsos positivos e falsos negativos estão baixos. No classification report, temos bons valores para precision, recall e f1score. A curva ROC está ótima (muito próxima de 1), indicando que o modelo está separando bem as classes agora.

Para os nossos testes, não realizamos a alteração dos hiperparâmetros dos modelos. Optamos por utilizar a configuração default e deu tudo certo, mas deixo o convite para você **ajustar o modelo com novos hiperparâmetros e comparar nos testes** se o desempenho pode realmente melhorar utilizando, por exemplo, um método de ajuste como **Grid Search**.

Agora temos o modelo pronto para ser utilizado! O próximo passo é armazenar esse modelo para posteriormente utilizar na aplicação. Para armazenar o modelo, você pode utilizar bibliotecas como o **joblib, pickle**, **HDF5**, entre muitas outras.

```
import joblib
```

Para salvar o modelo, basta dar um **dump**, passar o modelo que será salvo e nomeá-lo:

```
joblib.dump(modelo_xgb, 'xgb.joblib')
```

## O QUE VOCÊ VIU NESTA AULA?

Nessa aula, você aprendeu como identificar e avaliar o melhor modelo de Machine Learning. Essa etapa é o coração do projeto, pois a entrega será um modelo preditivo. Se esse modelo preditivo não estiver bem avaliado, a chance de o algoritmo falhar é enorme. Sempre valide e teste os modelos, existem muitas opções, mas sempre haverá aquela que pode se encaixar melhor no seu problema de negócio.

Tem alguma dúvida ou quer conversar sobre o tema desta aula? Entre em contato conosco pela comunidade do Discord! Lá você pode fazer networking, receber avisos, tirar dúvidas e muito mais.

# REFERÊNCIAS

ALENCAR, Valquiria. Machine Learning: Criando a Pipeline. 2023. São Paulo. 1. Notas de aula.

CRISP-DM: A metodologia ideal para Ciência de Dados. **Escola DNC.** Disponível em: <https://www.escoladnc.com.br/blog/data-science/metodologia-crisp-dm/>. Acesso em: 21 ago. 2023.

DOCUMENTAÇÃO SCIKIT-LEARN. **Scikit-Learn.** Disponível em: <https://scikit-learn.org/stable/>. Acesso em: 21 ago. 2023.

DOMINGUES, Paulo. Você sabe o que é CRISP-DM? Disponível em: < https://medium.com/bexs-io/voc%C3%AA-sabe-o-que-%C3%A9-crisp-dm-a3c15975bd4c>. Acesso em: 30 ago. 2023.

GÉRON, A. **Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow**. Sebastopol: O'Reilly Media, 2019.

## PALAVRAS-CHAVE

Modelos de Machine Learning, métricas de validação, árvore de decisão.