Cheat Sheet - Classificação: melhorando o desempenho com XGBoost

* Criando um modelo

*# importa a biblioteca XGBoost*

import xgboost as xgb

*link documentação:*

*https://xgboost.readthedocs.io/en/stable/index.html*

*# inicializa um modelo de classificação binária*

modelo = xgb.XGBClassifier(objective='binary:logistic')

* Ajustando um modelo

*# ajusta um modelo usando dados de treinamento*

modelo.fit(X\_treino, y\_treino)

* Fazendo previsões

*# fazendo previsões do modelo a partir dos dados de validação*

y\_pred = modelo.predict(X\_teste)

* Avaliando um modelo

*# avalia a taxa de acerto do modelo usando dados de teste*

acuracia = accuracy\_score(y\_teste, preds)

print(f'Acuracia: {acuracia:.2f}')

*link documentação:* [*https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy\_score.html*](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html)

* Relatório de classificação

*# importa a função para gerar o relatório de métricas de classificação*

*from sklearn.metrics import classification\_report*

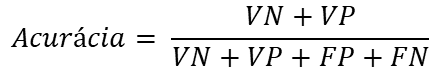
*link documentação:* [*https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification\_report.html*](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html)

*# gera o relatório de métricas de classificação usando os dados reais e previstos*

report = classification\_report(y\_teste, preds)

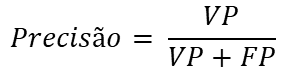
print(report)

*# A acurácia mede a proporção de predições corretas (verdadeiros positivos e verdadeiros negativos) em relação ao total de amostras. É uma métrica geral que avalia o desempenho global do modelo, mas pode ser enganosa em problemas de desequilíbrio de classe.*

**

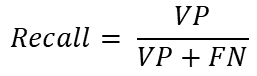
*link documentação:* [*https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy\_score.html*](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html)

*# A precisão é a proporção de previsões corretas entre todas as previsões positivas feitas pelo modelo. Em outras palavras, ela mede a qualidade das previsões positivas.*

**

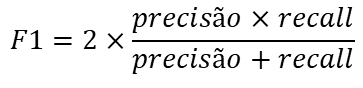
*link documentação:* [*https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision\_score.html*](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html)

*# O recall é a proporção de previsões corretas entre todos os exemplos positivos reais. Ela mede a capacidade do modelo de identificar todos os exemplos positivos.*

**

*link documentação:* [*https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall\_score.html*](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html)

*# O F1-Score é a média harmônica da precisão e o recall, fornecendo um equilíbrio entre essas duas métricas.*

**

*link documentação:* [*https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1\_score.html*](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html)

* Curva ROC AUC

*# importa a função para criação da visualização da curva ROC AUC*

from sklearn.metrics import RocCurveDisplay

*link documentação: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.RocCurveDisplay.html*

*# gerando a visualização da curva ROC AUC*

RocCurveDisplay.from\_predictions(y\_teste, preds, name='XGBoost');

* Matriz de confusão

*# importa a função para criação da visualização da matriz de confusão*

*from sklearn.metrics import ConfusionMatrixDisplay*

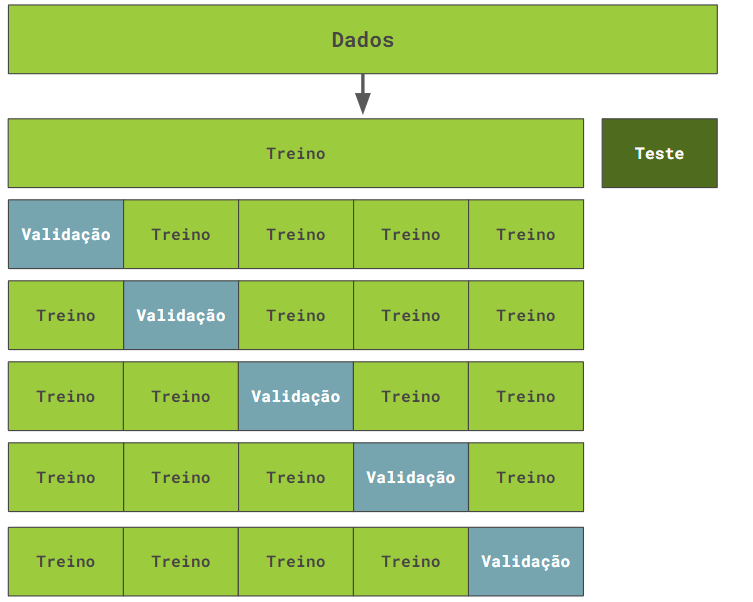
*link documentação:* [*https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html*](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html)

*# gerando a visualização da matriz de confusão a partir dos dados reais e previstos, com os valores normalizados e uma paleta de cores em tons de azul*

from sklearn.metrics import ConfusionMatrixDisplay

ConfusionMatrixDisplay.from\_estimator(xg\_cl, X\_teste, y\_teste, display\_labels=['Ausencia', 'Presenca'], normalize='true', cmap='Blues');

* Validação cruzada



*# Cria uma matriz otimizada chamada dmatrix:*

dmatrix = xgb.DMatrix(data=X, label=y)

*# Define os parâmetros*

params = {'objective': 'binary:logistic'}

*# Inicializa a função xgb.cv():*

cv\_resultados = xgb.cv(dtrain=dmatrix, params=params, nfold=3, num\_boost\_round=100, early\_stopping\_rounds=5, metrics='error', as\_pandas=True, seed=123)

# Imprime os resultados e a acurácia

print(cv\_resultados)

acuracia = 1 - cv\_resultados['test-error-mean'].iloc[-1]

print(f'Acuracia: {acuracia}')

* Ajuste de hiperparâmetros

*# Faz a busca aleatória (Random Search)*

from sklearn.model\_selection import RandomizedSearchCV

params = {

'learning\_rate': [0.1, 0.2, 0.3],

'max\_depth': [0, 2, 4, 6, 10],

'colsample\_bytree': [0.1, 0.5, 1],

'n\_estimators': [10, 25, 50]

}

modelo\_xgb = xgb.XGBClassifier()

random\_search\_cv = RandomizedSearchCV(estimator=modelo\_xgb, param\_distributions=params, n\_iter=5, cv=4, scoring='accuracy',

verbose=1)

random\_search\_cv.fit(X,y)

print('Melhores parametros encontrados: ', random\_search\_cv.best\_params\_)

print('Acuracia:', random\_search\_cv.best\_score\_)

*link documentação: https://scikit-learn.org/stable/modules/generated/sklearn.model\_selection.RandomizedSearchCV.html*

*# Faz a busca em grade (Grid Search)*

from sklearn.model\_selection import GridSearchCV

params = {

'learning\_rate': [0.1, 0.2, 0.3],

'max\_depth': [0, 2, 4, 6, 10],

'colsample\_bytree': [0.1, 0.5, 1],

'n\_estimators': [10, 25, 50]

}

modelo\_xgb = xgb.XGBClassifier()

grid\_search\_cv = GridSearchCV(estimator=modelo\_xgb, param\_grid=params, cv=4, scoring='accuracy', verbose=1)

grid\_search\_cv.fit(X,y)

print('Melhores parametros encontrados: ', grid\_search\_cv.best\_params\_)

print('Acuracia:', grid\_search\_cv.best\_score\_)

*link documentação: https://scikit-learn.org/stable/modules/generated/sklearn.model\_selection.GridSearchCV.html*

* Pipeline de dados

*# Importa a função de criação de pipelines*

from sklearn.pipeline import Pipeline

*link documentação: https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html*

*# Cria uma pipeline*

pipeline = Pipeline([('xgboost', xgb.XGBClassifier())])

*# Faz a busca em grade com a pipeline*

xgb\_param\_grid = {

'xgboost\_\_learning\_rate': [0.1, 0.2, 0.3],

'xgboost\_\_max\_depth': [0, 2, 4, 6, 10],

'xgboost\_\_colsample\_bytree': [0.1, 0.5, 1],

'xgboost\_\_n\_estimators': [10, 25, 50]

}

grid\_search\_cv = GridSearchCV(estimator=pipeline, param\_grid=xgb\_param\_grid, scoring='accuracy', cv=4, verbose=1)

grid\_search\_cv.fit(X, y)

print('Acuracia:', grid\_search\_cv.best\_score\_)

print('Melhor estimador:', grid\_search\_cv.best\_estimator\_)

* Salvar e carregar o modelo

*# Salva o modelo*

import joblib

joblib.dump(grid\_search\_cv, 'modelo\_pipeline.pkl')

*link documentação: https://joblib.readthedocs.io/en/stable/*

*# Carrega o modelo*

modelo = joblib.load('modelo\_pipeline.pkl')