Estudo de Caso: Identificação de Defeitos em Placas de Aço

ASA 2023

Ludmila Dias

Panorama Geral

Objetivo

Encontrar insights a partir dos dados e auxiliar a identificar se o defeito encontrado na placa é do tipo 0 ou do tipo 1.

Solução Propósta

Modelo de aprendizado supervisionado de Regressão Logística.

Etapas do Processo

01

Importação e Análise dos dados

02

Tratamento de Dados

- *Dados Nulos*
- *Dados Incorretos*
- *Outliers*

03

Encoding

04

Separação de Lotes e Treinamento do modelo

05

Avaliação do Modelo

06

Grid-Search

Importação e Análise dos Dados

Leitura de Dados

Descrição sobre cada coluna de acordo com a documentação.
Com exceção de 'slab_thickness' que não foi relatado sobre

- min_x_defect – Coordenada x inicial do defeito
- max_x_ defect – Coordenada x final do defeito
- min_y_ defect – Coordenada y inicial do defeito
- max_y_ defect – Coordenada y final do defeito
- area_pixels – Total de pixels presentes na placa
- slab_width – Largura da placa (eixo X)
- slab_length – Comprimento da placa (eixo Y)
- slab_thickness - Grossura ou Densidade da placa (?)
- sum_pixel_luminosity – Soma da luminosidade dos pixels
- min_pixel_luminosity – Mínima luminosidade dos pixels
- max_pixel_luminosity – Máxima luminosidade dos pixels
- conveyer_width – Largura da esteira (correia) transportadora (eixo X)
- type_of_steel – Identifica a classe do aço: pode pertencer à classe A300 ou A400
- defect_type – Tipo de defeito da classe. Pode ser do tipo 0 ou do tipo 1.

Características das Colunas

Valores únicos

```
defect_type              2
type_of_steel            4
slab_thickness          23
conveyer_width          75
max_pixel_luminosity    84
min_pixel_luminosity   115
slab_length            142
slab_width             151
area_pixels            420
min_x_defect           705
max_x_defect           706
sum_pixel_luminosity   951
min_y_defect           966
max_y_defect           966
dtype: int64
```

CSV Importado

	min_x_defect	max_x_defect	min_y_defect	max_y_defect	area_pixels	slab_width	slab_length	sum_pixel_luminosity	min_pixel_luminosity	max_pixel_luminosity	conveyer_width
0	38.0	49.0	735612.0	735624.0	113.0	11.0	12.0	12652.0	93.0	130.0	1707.0
1	1252.0	1348.0	355940.0	356016.0	1812.0	119.0	135.0	196003.0	NaN	132.0	1687.0
2	193.0	210.0	612201.0	612252.0	588.0	18.0	51.0	62182.0	73.0	135.0	1353.0
3	1159.0	1170.0	32914.0	32926.0	106.0	11.0	12.0	12792.0	100.0	134.0	1353.0
4	366.0	392.0	228379.0	228429.0	612.0	46.0	52.0	71337.0	103.0	127.0	1687.0
5	837.0	850.0	231429.0	231443.0	155.0	13.0	14.0	16093.0	55.0	134.0	1687.0
6	390.0	402.0	2513153.0	2513182.0	247.0	14.0	29.0	26419.0	NaN	126.0	1387.0
7	1351.0	1360.0	4807459.0	4807479.0	135.0	12.0	21.0	13096.0	NaN	109.0	1387.0
8	1325.0	1336.0	4848223.0	4848269.0	376.0	13.0	47.0	37703.0	NaN	117.0	1387.0
9	542.0	564.0	51943.0	51952.0	132.0	32.0	20.0	14760.0	104.0	119.0	1227.0

Como podemos ver acima, temos valores numéricos, categóricos string e o nosso target é binário. No dataframe acima já podemos ver que existem valores nulos, que devem ser tratados antes do treinamento.

Tratamento de Dados

Dados Incorretos

```
[ ]  df["type_of_steel"].unique()

     array(['TypeOfSteel_A300', 'TypeOfSteel_A400', 'TypeOfStel_A300',
            'TypeOfSteel_????'], dtype=object)
```

Só há dois valores possíveis para essa coluna

Verificando quantos valores com '???' haviam (9 linhas)

```
df.loc[df["type_of_steel"] == 'TypeOfSteel_????']
```

max_y_defect	area_pixels	slab_width	slab_length	sum_pixel_luminosity	min_pixel_luminosity	max_pixel_luminosity	conveyer_width	slab_thickness	type_of_steel	defe
310597.0	335.0	39.0	31.0	35066.0	NaN	127.0	1690.0	70.0	TypeOfSteel_????	
335000.0	101.0	12.0	12.0	11471.0	NaN	140.0	1690.0	70.0	TypeOfSteel_????	
643193.0	124.0	17.0	13.0	13135.0	NaN	127.0	1686.0	70.0	TypeOfSteel_????	
1928773.0	192.0	24.0	32.0	19002.0	NaN	125.0	1688.0	70.0	TypeOfSteel_????	
3277821.0	91.0	15.0	17.0	10957.0	105.0	141.0	1658.0	100.0	TypeOfSteel_????	
46725.0	56.0	11.0	11.0	5729.0	NaN	117.0	1362.0	70.0	TypeOfSteel_????	
224035.0	83.0	18.0	31.0	7446.0	71.0	108.0	1360.0	70.0	TypeOfSteel_????	
226495.0	187.0	20.0	21.0	13351.0	49.0	101.0	1362.0	200.0	TypeOfSteel_????	
2278020.0	46.0	10.0	8.0	5003.0	95.0	127.0	1354.0	80.0	TypeOfSteel_????	

Tratamento de Dados

Dados Incorretos

```
df.loc[df["type_of_steel"] == 'TypeOfSteel_????', "type_of_steel"] = None
df.loc[df["type_of_steel"].isna()]
```

```
df = df.dropna(subset=['type_of_steel'])
```

Excluindo as 9 linhas incorretas

Sobrescrevendo valores escritos incorretamente e verificando os valores atuais do dataframe

```
[ ]  df.loc[df["type_of_steel"] == 'TypeOfStel_A300', ["type_of_steel"]] ='TypeOfSteel_A300'

[ ]  df["type_of_steel"].unique()

     array(['TypeOfSteel_A300', 'TypeOfSteel_A400'], dtype=object)
```

Tratamento de Dados

Dados Nulos

```
[ ]  print(df.isna().sum(axis=0))

min_x_defect              0
max_x_defect              0
min_y_defect              0
max_y_defect              0
area_pixels               0
slab_width                0
slab_length               0
sum_pixel_luminosity      0
min_pixel_luminosity    233
max_pixel_luminosity      0
conveyer_width            0
slab_thickness            0
type_of_steel             0
defect_type               0
dtype: int64
```

Verificando qual coluna possui dados nulos

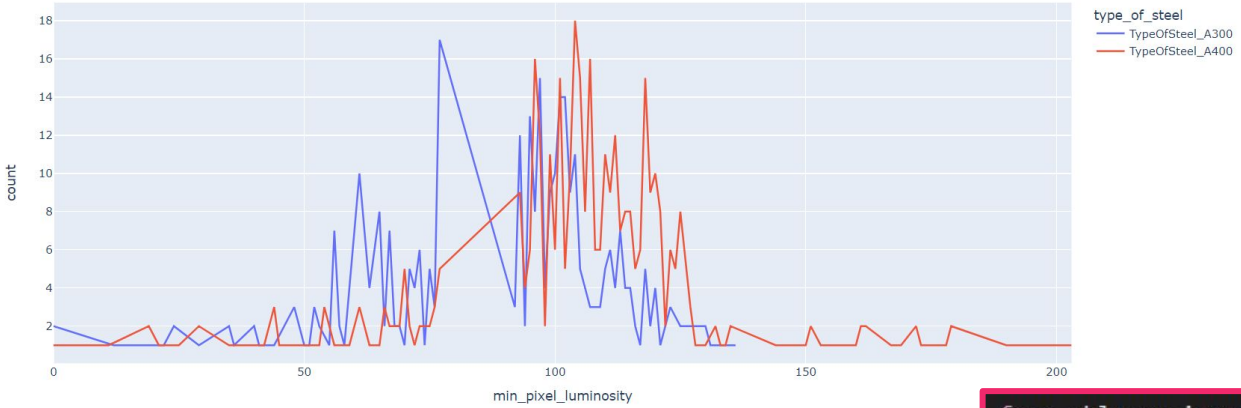
233 linhas de dados nulos

```
[ ]  df.loc[df.isna().sum(axis=1) > 0, :]
```

Formas de Resolver:

- Substituir o valor pela média dos valores da coluna
- Substituir o valor pela mediana
- Substituir o valor pelos valores mais frequentes

Tratamento de Dados
Dados Nulos



Valores após serem preenchidos

	min_pixel_luminosity	type_of_steel
1	77.0	TypeOfSteel_A300
6	104.0	TypeOfSteel_A400
7	104.0	TypeOfSteel_A400
8	104.0	TypeOfSteel_A400
11	104.0	TypeOfSteel_A400
...	...	...
901	77.0	TypeOfSteel_A300
903	77.0	TypeOfSteel_A300
905	77.0	TypeOfSteel_A300
907	77.0	TypeOfSteel_A300
926	77.0	TypeOfSteel_A300

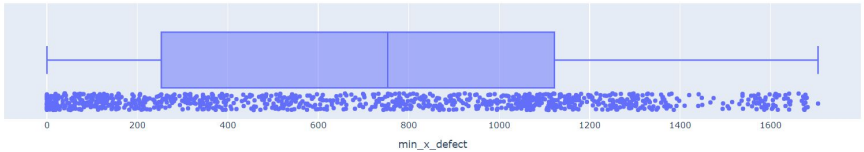
233 rows × 2 columns

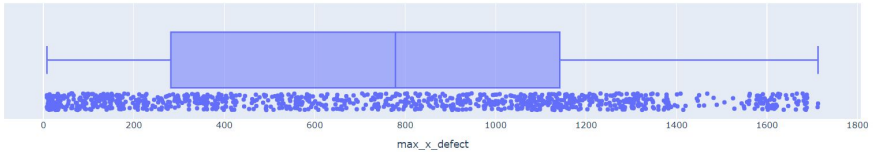
Altera valores nulos por valores frequentes de sua categoria

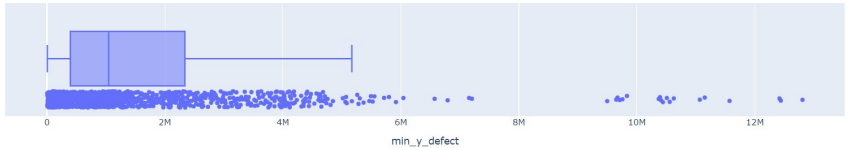
```
from sklearn.impute import SimpleImputer

classes = df['type_of_steel'].unique()

for classe in classes:
    subset = df[df['type_of_steel'] == classe]
    simp = SimpleImputer(strategy='most_frequent')
    subset = simp.fit_transform(subset)
    df.loc[df['type_of_steel'] == classe] = subset
```

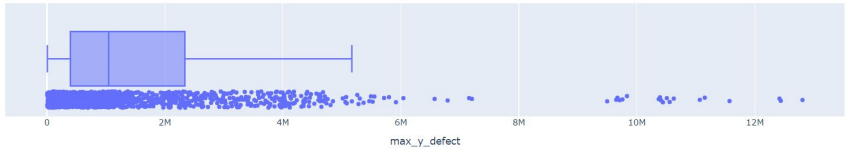
Tratamento de Dados

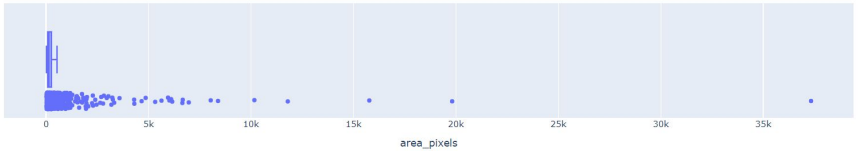
Outliers

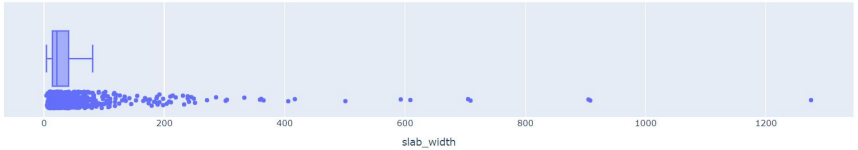


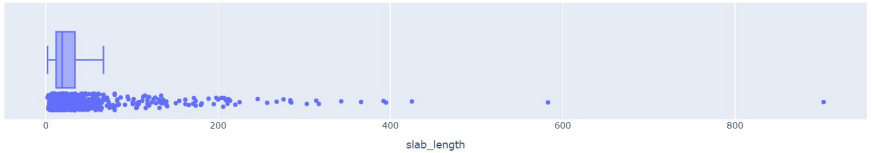












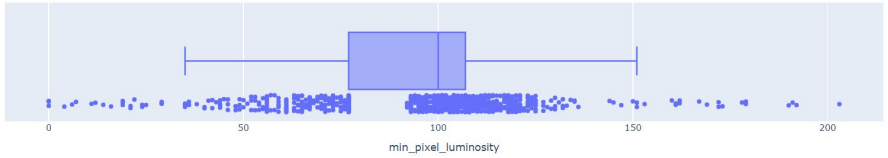
Tratamento de Dados
Outliers

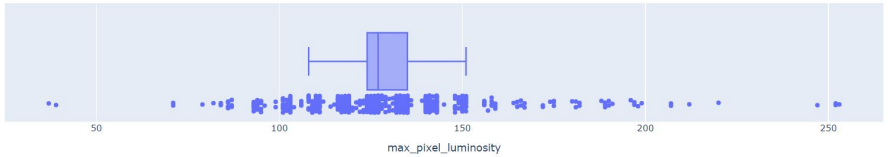
	min_x_defect	max_x_defect	min_y_defect	max_y_defect	area_pixels	slab_width	slab_length
345	889.0	921.0	12438460.0	12438491.0	699.0	38.0	32.0
346	1094.0	1124.0	12806495.0	12806520.0	571.0	37.0	25.0
548	1135.0	1162.0	12416454.0	12416473.0	305.0	33.0	27.0

Linha 345:
max_y_defect em micrometros ->12.438491
Min_x_defect em micrometros -> 12.43846

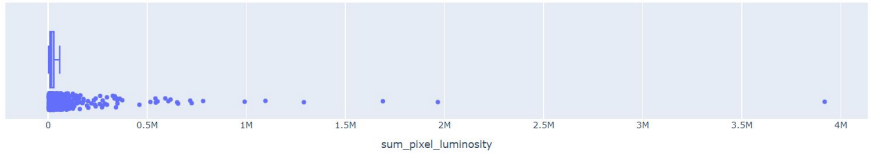
A variação nos valores é muito pequena, em micrômetro esse valor fica mais notório

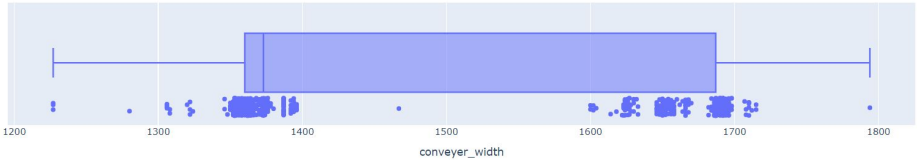
Tratamento de Dados
Outliers

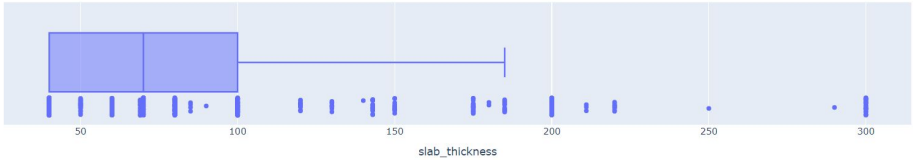




Varia de 0 a 255





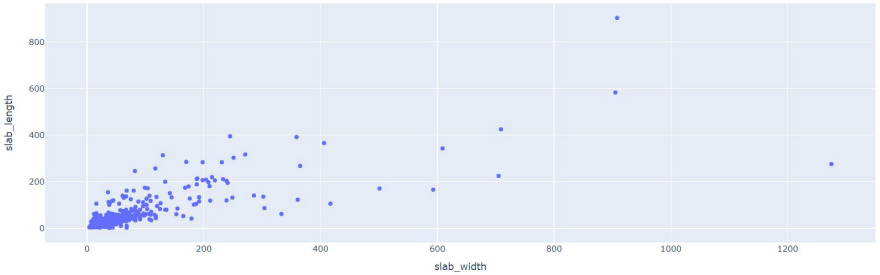


Exemplo: 37.334 (maior área de pixels) x 252 (maior valor de luminosidade) = 9.408.168 Esse seria um valor máximo hipotético. E o outlier apresentado na coluna abaixo é de aprox. 3M. Podendo ser um valor possível.

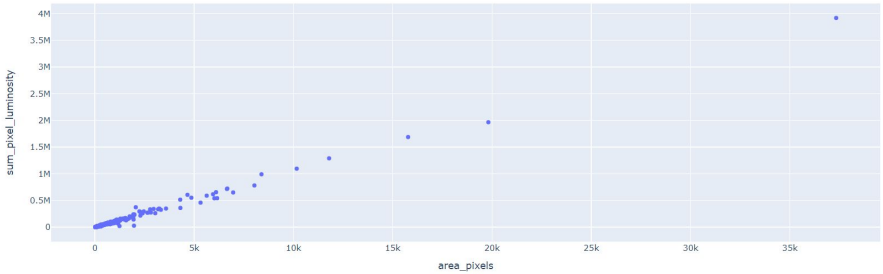
3M < 9M

Tratamento de Dados
Outliers

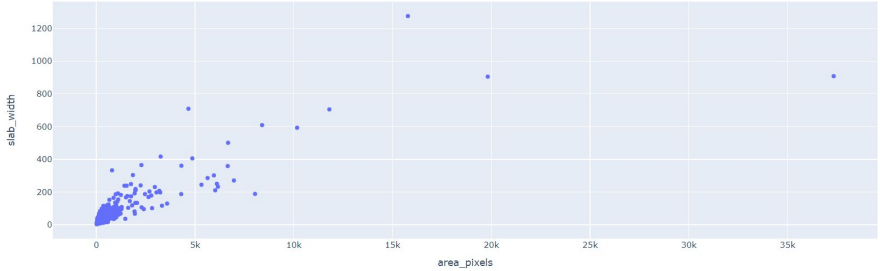
Correlação do comprimento e largura



Correlação da soma da luminosidade e área de pixels



Correlação da largura da placa com a área de pixels



Encoding
Ordinal Encoder

```
[ ]  from sklearn.preprocessing import OrdinalEncoder

     ord_enc = OrdinalEncoder(dtype=int)
     ord_enc.fit(df[['type_of_steel']])

     df_enc = df.copy()
     df_enc[['type_of_steel']] = ord_enc.transform(df[['type_of_steel']])
     df_enc
```

type_of_steel
0
0
1
1
1
...
1
1
1
1
0

Separação de Lotes e Treinamento do modelo

Lotes de Treino e Teste (Folds)

```python
from sklearn.model_selection import StratifiedKFold

def split_data(df_enc, N_folds = 5):
  # Escolha do número de folds
  N_folds = 5

  # Criação do Splitter
  splitter = StratifiedKFold(
      n_splits=N_folds,
      random_state=38,
      shuffle=True)

  # Separar X do y
  df_sem_y = df_enc.copy().drop(columns='defect_type')
  df_y = df_enc.copy()['defect_type'].to_frame()

  return splitter, df_sem_y, df_y
```

Cria o Splitter e separa o target do resto do df

```python
N_folds = 5
splitter, df_sem_y, df_y = split_data(df_enc, N_folds)

# Listas para armazenar os grupos da validação cruzada

X_train_fold = []
y_train_fold = []
X_test_fold = []
y_test_fold = []

for index_train, index_test in splitter.split(df_sem_y, df_y):
  X_train = df_sem_y.iloc[index_train]
  y_train = df_y.iloc[index_train]
  X_test = df_sem_y.iloc[index_test]
  y_test =  df_y.iloc[index_test]

  # adicionar na lista
  X_train_fold.append(X_train)
  y_train_fold.append(y_train)
  X_test_fold.append(X_test)
  y_test_fold.append(y_test)
```

Separa os dados em lotes e em conjuntos de treino e de teste

Separação de Lotes e Treinamento do modelo

Treino do Modelo com Validação Cruzada

```python
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
from sklearn.linear_model import LogisticRegression

# dicionário que vai conter as métricas do treino
train_metrics = {
    'accuracy': [],
    'recall': [],
    'precision': [],
    'f1': []
}

# dicionário que vai conter as métricas do teste
test_metrics = {
    'accuracy': [],
    'recall': [],
    'precision': [],
    'f1': []
}

# Listas para guardar as previsões feitas pro grupo de treino e teste
y_hat_train_fold = []
y_hat_test_fold = []
y_hat_test_proba_fold = []

# Execução para cada lote
for fold in range(N_folds):

  # Recuperando os dados desse fold específico
  X_train = X_train_fold[fold]
  X_test = X_test_fold[fold]
  y_train = y_train_fold[fold]
  y_test = y_test_fold[fold]
```

```python
# Criando o scaler aqui dentro para só escalar nos dados de treino
x_scaler = StandardScaler()
x_scaler.fit(X_train)

# Aplicar a normalização
X_train_norm = x_scaler.transform(X_train)
X_test_norm = x_scaler.transform(X_test)

# Treino do modelo
model = LogisticRegression(
    fit_intercept=False,
    random_state=38
)
model.fit(X_train_norm, y_train.iloc[:,0])
```

Separação de Lotes e Treinamento do modelo

Treino do Modelo com Validação Cruzada

```
y_hat_train = model.predict(X_train_norm)
y_hat_test = model.predict(X_test_norm)
y_hat_proba_test = model.predict_proba(X_test_norm)[:,1]

# Salvar previsões para o conjunto de treino
y_hat_train_fold.append(y_hat_train)
y_hat_test_fold.append(y_hat_test)
y_hat_test_proba_fold.append(y_hat_proba_test)

# Calcular métricas do treino
acc = accuracy_score(y_train, y_hat_train)
train_metrics['accuracy'].append(acc)

rec = recall_score(y_train, y_hat_train)
train_metrics['recall'].append(rec)

precision = precision_score(y_train, y_hat_train)
train_metrics['precision'].append(precision)

f1 = f1_score(y_train, y_hat_train)
train_metrics['f1'].append(f1)

# Calcular métricas do teste
acc = accuracy_score(y_test, y_hat_test)
test_metrics['accuracy'].append(acc)

rec = recall_score(y_test, y_hat_test)
test_metrics['recall'].append(rec)

precision = precision_score(y_test, y_hat_test)
test_metrics['precision'].append(precision)

f1 = f1_score(y_test, y_hat_test)
test_metrics['f1'].append(f1)
```

Salvando predições e resultado das métricas (Rec, Precision, Acc, F1)

Avaliação do Modelo

Métricas de treino e teste

Média dos valores

	Treino	Teste
accuracy	0.670	0.656
recall	0.766	0.754
precision	0.540	0.527
f1	0.634	0.620

- Métricas do Treino

```
[ ] df_train_metrics = pd.DataFrame(train_metrics)
    df_train_metrics
```

	accuracy	recall	precision	f1
0	0.673629	0.761404	0.543860	0.634503
1	0.677546	0.800000	0.545455	0.648649
2	0.665796	0.744755	0.537879	0.624633
3	0.662321	0.762238	0.533007	0.627338
4	0.671447	0.762238	0.542289	0.633721

```
[ ] df_train_metrics = df_train_metrics.mean(axis=0)
    df_train_metrics
```

```
accuracy     0.670148
recall       0.766127
precision    0.540498
f1           0.633769
dtype: float64
```

- Métricas do Teste

```
df_test_metrics = pd.DataFrame(test_metrics)
df_test_metrics
```

	accuracy	recall	precision	f1
0	0.625000	0.736111	0.500000	0.595506
1	0.671875	0.694444	0.549451	0.613497
2	0.656250	0.788732	0.523364	0.629213
3	0.659686	0.788732	0.528302	0.632768
4	0.664921	0.760563	0.534653	0.627907

```
df_test_metrics = df_test_metrics.mean(axis=0)
df_test_metrics
```

```
accuracy     0.655546
recall       0.753717
precision    0.527154
f1           0.619778
dtype: float64
```

Acurácia: Cerca de 65% das vezes ele previu certo em cima de todas as suas previsões.
Recall: Cerca de 75% por cento do que ele deveria ter acertado, ele acertou.
Precisão: Mais de 50 % das vezes que ele tentou acertar ele acertou
F1: Avalia o equilíbrio no Recall e na precisão, possui um valor de desempenho acima de 60%.

Avaliação do Modelo
Matriz de Confusão

```
y_test_list = []
for y_fold in y_test_fold:
  y_test_list.extend(y_fold.iloc[:,0])

y_hat_test_list = []
for lista_fold in y_hat_test_fold:
  y_hat_test_list.extend(lista_fold)

y_test_hat_proba_list = []
for lista_fold in y_hat_test_proba_fold:
  y_test_hat_proba_list.extend(lista_fold)
```

```
confusion_matrix = pd.DataFrame(
    sklearn.metrics.confusion_matrix(y_test_list, y_hat_test_list),
    index=['Defeito 0', 'Defeito 1'],
    columns=['Defeito 0', ' Defeito 1'],
)

display(confusion_matrix.style.background_gradient(axis=None))
```

	Defeito 0	Defeito 1
Defeito 0	359	242
Defeito 1	88	269

Verdadeiro

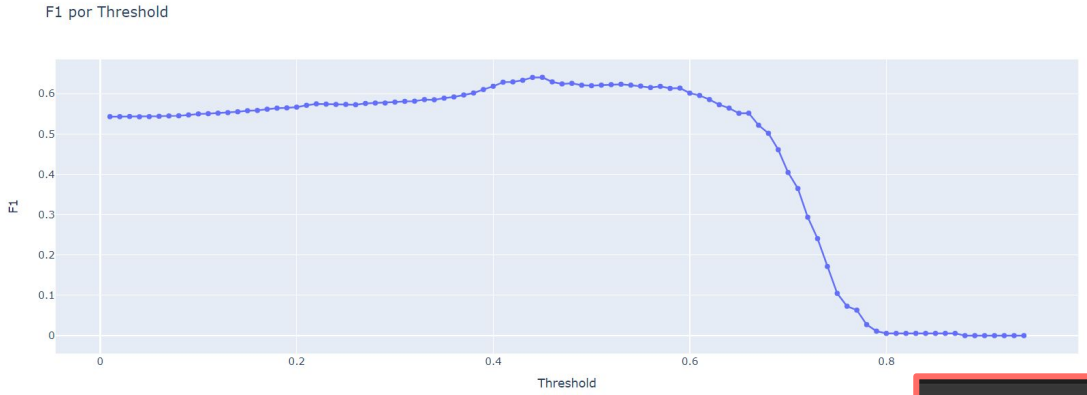
Predito

Ajuste do threshold

```
import numpy as np

# Lista de thresholds
thresholds = np.arange(0.01, 0.95, 0.01)

f1_scores = []
for threshold in thresholds:
  y_pred_th = (y_test_hat_proba_list > threshold).astype(int)
  f1 = f1_score(y_test_list, y_pred_th)
  f1_scores.append(f1)

optimal_threshold = thresholds[np.argmax(f1_scores)]
print("Limiar de Decisão Ótimo:", optimal_threshold)
```

```
Limiar de Decisão Ótimo: 0.45
```

Avaliação do Modelo
Matriz de Confusão



F1 de 0.62 para 0.646

Predito

Verdadeiro

F1: 0.646

	Defeito 0	Defeito 1
Defeito 0	317	284
Defeito 1	55	302

Grid-Search

'C': Parâmetro que controla a força da regularização, sendo valores menores mais fortemente regularizados e valores maiores menos regularizados.

'penalty': Especifica o tipo de regularização, com 'l2' indicando a penalização dos coeficientes quadrados para evitar overfitting.

'class_weight': Atribui pesos às classes, sendo 'balanced' útil para lidar automaticamente com desbalanceamento proporcional às frequências das classes

```
from sklearn.model_selection import train_test_split

# Considere df como o seu DataFrame
X = df_enc.drop('defect_type', axis=1)
y = df_enc['defect_type']

# Dividir os dados em conjuntos de treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

y_train = y_train.ravel()
y_test = y_test.ravel()
```

Transformando dados em arrays de 1 dimensão

Grid-Search

```python
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
from sklearn.linear_model import LogisticRegression

# Criando o scaler aqui dentro para só escalar nos dados de treino
x_scaler = StandardScaler()
x_scaler.fit(X_train)

# Aplicar a normalização
X_train_norm = x_scaler.transform(X_train)


# Treinar o modelo
model = LogisticRegression(
    random_state=38,
    max_iter=1000
)

param_grid = {'C': [ 0.001, 0.01, 0.1, 1, 10, 100, 500, 800, 1000],
              'penalty': ['l2'],
              'class_weight': [None, 'balanced']
              }
grid_search = GridSearchCV(model, param_grid, cv=5, scoring='f1')
grid_search.fit(X_train_norm, y_train)
best_params = grid_search.best_params_
best_params
```

```
{'C': 500, 'class_weight': 'balanced', 'penalty': 'l2'}
```

```python
# Treinar o modelo
model = LogisticRegression(
    fit_intercept=False,
    random_state=38,
    C=500,
    class_weight='balanced',
    penalty='l2'

)

model.fit(X_train_norm, y_train.iloc[:,0])
```

	Treino	Teste
accuracy	0.684	0.668
recall	0.810	0.785
precision	0.552	0.538
f1	0.657	0.638

Valores depois do GS

	Treino	Teste
accuracy	0.670	0.656
recall	0.766	0.754
precision	0.540	0.527
f1	0.634	0.620

Valores antes do GS

Questão 1

Questão 1:

O modelo construído será utilizado para reduzir o erro na hora da classificação do defeito. Sabe-se que:

- Cada acerto do modelo significa um custo de 500 reais para a recuperação da placa;
- Identificar o defeito 0 incorretamente como defeito 1 gera um custo de 500 reais para recuperação da placa mais um custo de 3500 reais por conta do custo logístico de ter enviado a placa para o tratamento incorreto;
- Identificar o defeito 1 incorretamente como defeito 0 gera um custo de 500 reais para a recuperação da placa. Além disso, há também um custo de 6213 reais por conta do erro logístico de ter enviado a placa para o tratamento incorreto, sabendo que o tratamento do defeito 1 ocorre numa etapa do processo anterior ao do defeito 0.

Em posse dessas informações, qual seria o custo total do processo com o uso em produção do modelo desenvolvido? Deixe bem claro todo o passo a passo utilizado para a obtenção do resultado.

Verdadeiro

.	D0	D1
D0	500	500 + 3500
D1	500 + 6213	500

Predito

Questão 1

```python
import numpy as np
from sklearn.metrics import confusion_matrix

# Lista de threshold
thresholds = np.arange(0.01, 0.95, 0.01)

scores = []

# Para cada threshold gerado
for threshold in thresholds:

  y_test_th = (np.array(y_test_hat_proba_list) > threshold).astype(int)

  # Criação da matriz de confusão
  cm = pd.DataFrame(
      confusion_matrix(y_test_list, y_test_th),
      index=['Defeito 0', 'Defeito 1'],
      columns=['Defeito 0', ' Defeito 1'],
  )

  # Verifica a quantidade de acertos e erros, e calcula o valor de custo total
  total = (cm.iloc[0,0]*500)+(cm.iloc[0,1]*4000)+(cm.iloc[1,0]*6713)+(cm.iloc[1,1]*500)
  scores.append(total)

# Impressão do melhor threshold e o menor valor de custo
optimal_threshold = thresholds[np.argmin(scores)]
print("Limiar de Decisão Ótimo:", optimal_threshold)
print("Menor Valor de custo:", np.min(scores), "reais")
```

```
Limiar de Decisão Ótimo: 0.45
Menor Valor de custo: 1814715 reais
```

Predito

Verdadeiro

	Defeito 0	Defeito 1
Defeito 0	317	284
Defeito 1	55	302

Questão 2

Questão 2: Sem uma ferramenta mais tecnológica ao seu dispor, atualmente a distinção dos dois defeitos é feita de forma manual por um especialista. Para esse conjunto de dados específico, ele obteve os seguintes resultados:

- 350 placas com defeito tipo 0 foram identificadas corretamente;
- 256 placas com defeito tipo 0 foram identificadas como defeito tipo 1;
- 161 placas com defeito tipo 1 foram identificadas corretamente;
- 200 placas com defeito tipo 1 foram identificadas como defeito tipo 0.

Conhecendo o custo de cada tipo de erro (conforme questão 1), qual seria a economia que a utilização do seu modelo traria para o processo? Deixe bem claro todo o passo a passo utilizado para a obtenção do resultado.

```
[ ]  # Calculando o custo SEM O MODELO x COM O MODELO

     totalSemModelo = (350*500)+(256*4000)+(200*6713)+(161*500)
     totalComModelo = (cm.iloc[0,0]*500)+(cm.iloc[0,1]*4000)+(cm.iloc[1,0]*6713)+(cm.iloc[1,1]*500)

     print('Total SEM O MODELO:', totalSemModelo)
     print('\nTotal COM O MODELO:', totalComModelo)
```

```
Total SEM O MODELO: 2622100

Total COM O MODELO: 1814715
```

Sendo assim teria uma economia financeira de:

```
[ ]  print(f'{(((totalSemModelo-totalComModelo)*100)/totalSemModelo):.2f}%')
     print(f'{totalSemModelo-totalComModelo} reais')
```

```
30.79%
807385 reais
```

Obrigada pela Atenção!