# **EXPEDIÇÃO** | Análise de Dados: Previsão do tempo de separação utilizando o algoritimo XGBoost.

Área de atuação: **Logísitca FC** By: Estteves Santos

#**XGBoost** XGBoost é um dos algoritmos mais utilizados em problemas de previsão envolvendo dados estruturados/tabulares.

O nome XGBoost vem de eX*treme* Gradient Boosting, e representa uma categoria de algoritmo baseada em Decision Trees (árvores de decisão) com Gradient Boosting (aumento de gradiente).

As árvores de decisão ajudam a identificar padrões nos dados, fornecer interpretabilidade, lidar com diferentes tipos de variáveis, lidar com ruído e outliers, e generalizar as relações aprendidas para fazer previsões em novos dados.

O XGBoost implementa o algoritmo de gradient boosting, uma técnica de aprendizado de máquina que combina várias árvores de decisão fracas para criar um modelo preditivo mais forte. Ele usa um processo iterativo em que cada nova árvore é treinada para corrigir os erros do modelo anterior, ajustando gradualmente as previsões.

## # Utilização do XGBoost para prever o tempo de separação (TEMP_SEP) com base nas variáveis QTDE_UNIDADES, PESO, QTDE_ITENS, DEPOSITO, TIPO_ENTREGA e GRUPO_PESO na filial PAL

O objetivo é utilizar o modelo para estimar o tempo de separação dos separadores de depósito.

1. Carregar os dados e preparar as variáveis

Query: SELECT * FROM SFC_ENTREGA_IMEDIATA WHERE COD_EMPRESA ='3' AND INICIO_SEPARACAO >='01/01/2023' AND TIPO_ENTREGA IN ('I','P') AND GRUPO_PESO <> 'E'

```
#Importar as bibliotecas
import pandas as pd
import xgboost as xgb
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import RobustScaler
```

```
from sklearn.model_selection import train_test_split

# Carregar os dados em um DataFrame do pandas
PAL = pd.read_csv('/content/PAL_v3.csv', sep=';', encoding='latin1')

# Converter as colunas para os tipos adequados
PAL['TEMP_SEP'] = PAL['TEMP_SEP'].str.replace(',', '.').astype(float)
#PAL['PESO'] = PAL['PESO'].str.replace(',', '.').astype(float)
PAL['PESO'] = round(PAL['PESO'])
PAL['DEPOSITO'] = PAL['DEPOSITO'].astype(int)

# Adicionar as novas variáveis de entrada
X = PAL[['QTDE_UNIDADES', 'PESO', 'QTDE_ITENS', 'DEPOSITO',
'TIPO_ENTREGA', 'GRUPO_PESO']]

# Converter variáveis categóricas em variáveis dummy
X = pd.get_dummies(X, columns=['TIPO_ENTREGA', 'GRUPO_PESO'])

# Definir a variável alvo (TEMP_SEP)
y = PAL['TEMP_SEP']

# Remover outliers usando Z-Score
z_scores = (y - y.mean()) / y.std()
outlier_threshold = 2.5  # Defina o limiar de Z-Score para identificar
outliers
X_clean = X[(z_scores > -outlier_threshold) & (z_scores <
outlier_threshold)]
y_clean = y[(z_scores > -outlier_threshold) & (z_scores <
outlier_threshold)]

# Normalizar os dados de entrada usando RobustScaler
scaler = RobustScaler()
X_clean_norm = scaler.fit_transform(X_clean)

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

# Ajustar os hiperparâmetros
parametros = {
    'learning_rate': 0.1,
    'max_depth': 5,
    'min_child_weight': 2,
    'subsample': 0.8,
    'colsample_bytree': 0.8,
    'gamma': 0.1,
    'reg_alpha': 0,
    'reg_lambda': 1.0,
    'objective': 'reg:squarederror',
    'seed': 42
```

```
}

# Treinar o modelo XGBoost
dtrain = xgb.DMatrix(X_train, label=y_train)
model = xgb.train(params=parametros, dtrain=dtrain,
num_boost_round=500)

# Salvar o modelo em um arquivo binário
model.save_model('modelo.bin')

# Fazer previsões no conjunto de treinamento
train_predictions = model.predict(xgb.DMatrix(X_train))

# Calcular o RMSE no conjunto de treinamento
rmse_train = mean_squared_error(y_train, train_predictions,
squared=False)

print("RMSE no conjunto de treinamento:", rmse_train)

# Fazer previsões no conjunto de teste
test_predictions = model.predict(xgb.DMatrix(X_test))

# Calcular o RMSE no conjunto de teste
rmse_test = mean_squared_error(y_test, test_predictions,
squared=False)

print("RMSE no conjunto de teste:", rmse_test)

# Adicionar as previsões como uma nova coluna no DataFrame
PAL['PREVISAO'] = model.predict(xgb.DMatrix(scaler.transform(X)))
PAL['DIFERENCA'] = PAL['TEMP_SEP'] - PAL['PREVISAO']

# Exibir o DataFrame com as previsões e a diferença
print(PAL)


RMSE no conjunto de treinamento: 2.0176457950407944
RMSE no conjunto de teste: 2.1977049617427773
       ï»¿COD_EMPRESA  QTDE_UNIDADES  PESO  QTDE_ITENS  DEPOSITO
TIPO_ENTREGA  \
0                   3             14   195           3         1
I
1                   3              1    12           1         1
I
2                   3              2    45           2         1
I
3                   3              3    60           1         1
I
4                   3             30   751           1         1
P
...               ...            ...   ...         ...       ...
```

```
...
69967                3               4     9           2          1
I
69968                3              33   331           3          1
P
69969                3               2    74           1          1
I
69970                3               1     5           1          1
I
69971                3               2    31           1          1
I

      GRUPO_PESO  TEMP_SEP  PREVISAO  DIFERENCA
0              A      3.27  6.298564  -3.028564
1              A      2.82  3.239501  -0.419501
2              A      3.13  3.498134  -0.368134
3              A      1.22  2.513503  -1.293503
4              C      4.95  6.406050  -1.456050
...          ...       ...       ...        ...
69967          A     11.17  4.706844   6.463156
69968          B     12.07  7.188062   4.881938
69969          A     10.37  3.457589   6.912411
69970          A      3.82  2.180587   1.639413
69971          A      3.53  3.267838   0.262162

[69972 rows x 10 columns]
```

```python
# Previsão com valores inseridos pelo usuário
qtde_unidades = 4
peso = 54
qtde_itens = 2
deposito = 1
tipo_entrega = 'P'
grupo_peso = 'A'

# Criar um DataFrame com os valores inseridos manualmente
novos_dados = pd.DataFrame({
    'QTDE_UNIDADES': [qtde_unidades],
    'PESO': [peso],
    'QTDE_ITENS': [qtde_itens],
    'DEPOSITO': [deposito],
    'TIPO_ENTREGA_P': [1] if tipo_entrega == 'P' else [0],
    'TIPO_ENTREGA_I': [1] if tipo_entrega == 'I' else [0],
    'GRUPO_PESO_A': [1] if grupo_peso == 'A' else [0],
    'GRUPO_PESO_B': [1] if grupo_peso == 'B' else [0],
    'GRUPO_PESO_C': [1] if grupo_peso == 'C' else [0],
    'GRUPO_PESO_D': [1] if grupo_peso == 'D' else [0]
})

# Reordenar as colunas para que estejam na mesma ordem do treinamento
```

```
novos_dados = novos_dados[X_clean.columns]

# Normalizar os novos dados usando o mesmo scaler do conjunto de
treinamento
novos_dados_norm = scaler.transform(novos_dados)

# Fazer a previsão com o modelo treinado
previsao = model.predict(xgb.DMatrix(novos_dados_norm))

# Exibir a previsão
print("Previsão:", previsao[0])
```

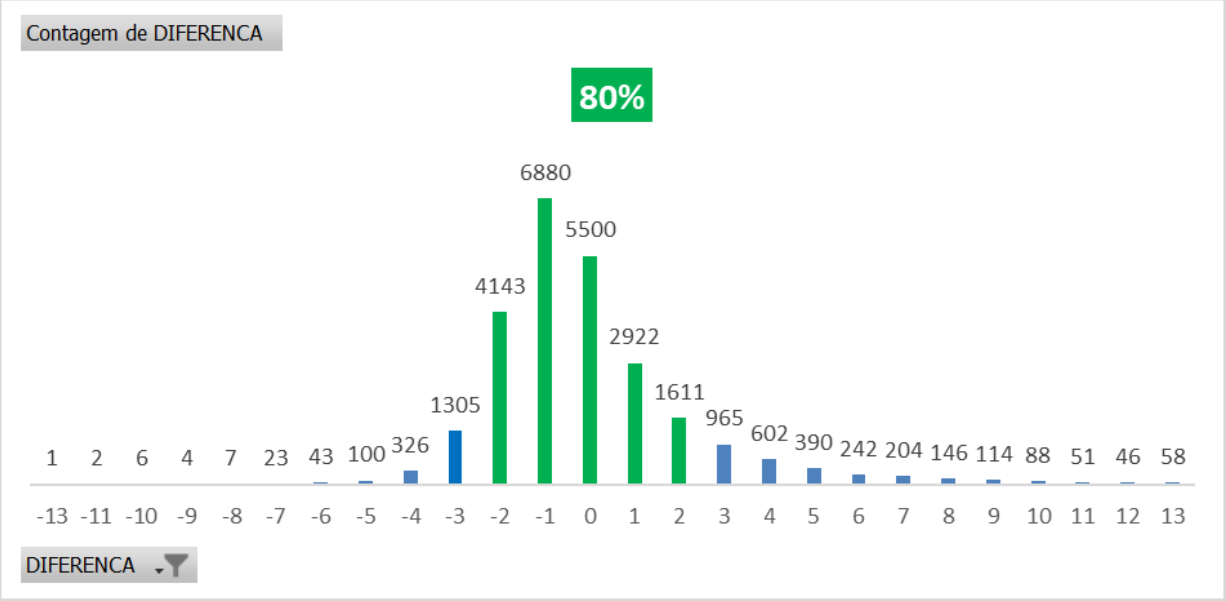
```
Previsão: 2.8956895
```

```
# Salvar o modelo em um arquivo binário
model.save_model('modelo.bin')

# Previsão com valores inseridos por arquivos CSV (automática)
import pandas as pd
import xgboost as xgb
from sklearn.preprocessing import RobustScaler

# Carregar os dados em um DataFrame do pandas
df = pd.read_csv('/content/IMB_v3.csv', sep=';', encoding='latin1')

# Converter as colunas para os tipos adequados
df['TEMP_SEP'] = df['TEMP_SEP'].str.replace(',', '.').astype(float)
df['PESO'] = df['PESO'].str.replace(',', '.').astype(float)
df['PESO'] = round(df['PESO'])
df['DEPOSITO'] = df['DEPOSITO'].astype(int)

# Aplicar as mesmas transformações nos dados de entrada
X = df[['QTDE_UNIDADES', 'PESO', 'QTDE_ITENS', 'DEPOSITO',
'TIPO_ENTREGA', 'GRUPO_PESO']]
X = pd.get_dummies(X, columns=['TIPO_ENTREGA', 'GRUPO_PESO'])

# Normalizar os dados de entrada usando o mesmo scaler utilizado no
treinamento
scaler = RobustScaler()
X_norm = scaler.fit_transform(X)

# Carregar o modelo treinado
model = xgb.Booster()
model.load_model('modelo.bin')

# Fazer previsões nos novos dados
predictions = model.predict(xgb.DMatrix(X_norm))

# Adicionar as previsões como uma nova coluna no DataFrame
df['PREVISAO'] = predictions
```

```
# Salvar o DataFrame com as previsões em um novo arquivo
df.to_csv('resultado.csv', index=False, sep=';')
```

# Conclusão:

Com quase 70mil dados, foi possível prever um modelo que 80% dos tempos previstos estão entre (-2,2)minutos do tempo real.




Além disso, é possivel observar que nos dados em que o modelo realizou uma previsão muito diferente do tempo de separação, possívelmente foi uma falha humana no processo de separação. Pois, conforme o exemplo abaixo, um pedido com mais QTDE de itens, maior QTDE de unidades e maior peso o separador realizou em 6,4min enquanto um pedido menor e mais leve, ele realizou em 10,12min. Se o processo de separação estivesse ocorrido de acordo 'normalmente', a diferença entre o tempo previsto e o tempo de real (TEMP_SEP), teria sido menor. E, conseguentemente o RMSE do modelo seria menor.

| COD_EMPRESA | QTDE_UNIDADES | PESO | QTDE_ITENS | DEPOSITO | TIPO_ENTREGA | GRUPO_PESO | TEMP_SEP | PREVISAO | DIFERENCA | DIFERENCA |
|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 1 | 8 | 1 | 1 | I | A | 2,37 | 3,99 | -2 | -1,62 |
| 2 | 9 | 99 | 4 | 1 | I | A | 6,4 | 5,72 | 1 | 0,68 |
| 2 | 1 | 20 | 1 | 1 | P | A | 1,48 | 2,88 | -1 | -1,40 |
| 2 | 1 | 8 | 1 | 5 | I | A | 1,5 | 2,28 | -1 | -0,78 |
| 2 | 1 | 31 | 1 | 1 | I | A | 10,12 | 3,72 | 6 | 6,40 |