

Projeto de Churn

Redes Neurais

Arthur Fernandes Scanoni
João Marcos Alcântara Vanderley
Kennedy Edmilson Cunha Melo
Mateus Elias de Andrade Pereira
Rafael dos Reis de Labio



Centro de
Informática
UFPE




UNIVERSIDADE
FEDERAL
DE PERNAMBUCO

Dataset

Customers churned in telecom services


Dataset

 ALEXANDER KAPUROV · UPDATED 5 MONTHS AGO

26

Code

Download



Customers churned in telecom services

Customers information related to telecom services and whether they have churned

Data Card

Code (9)

Discussion (0)

Suggestions (0)

About Dataset

If you continuously find you have a high customer churn rate, you'll quickly find that your business isn't sustainable. Getting new customers to sign up to your service is one thing, but it's not enough to keep your business afloat for long. To survive, your business needs loyal customers, and that means continuously looking at ways you can improve your service to keep your customers happy. If you don't, your business will become unviable.


What causes churn in telecoms?

- Poor service experience
- Poor customer service or experience
- Easy to switch providers

How to reduce churn rate in telecoms?

- Improve customer service
- Create a memorable customer experience
- Invest in new technologies
- Make Better use of data

View more



Usability

10.00

License

CC0: Public Domain

Expected update frequency

Annually

Tags

Beginner

Intermediate

E-Commerce Services

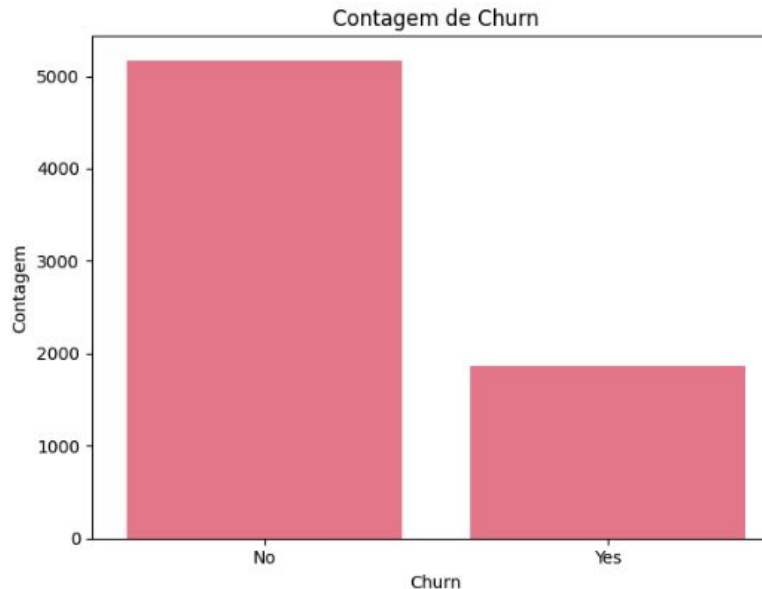
Mobile and Wireless

Binary Classification

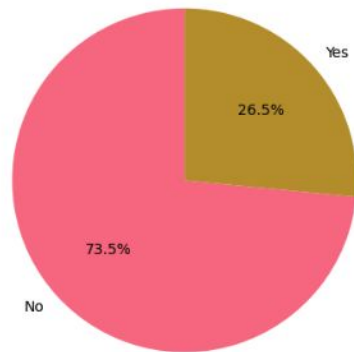
Pré-processamento

Pré-processamento

- **Carregamento e Exploração Inicial**
 - Dataset de churn telecom com 7.043 clientes e 21 features
 - Distribuição desbalanceada: 73.4% No Churn vs 26.6% Yes Churn
 - Identificação de 11 valores ausentes na coluna 'TotalCharges'



Distribuição de Churn



Pré-processamento

- Tratamento de Dados Ausentes
 - Conversão de 'TotalCharges' para formato numérico
 - Preenchimento com mediana (R\$ 1,397.47) dos valores ausentes
 - Zero valores ausentes após tratamento

🔍 Análise de Dados Ausentes:

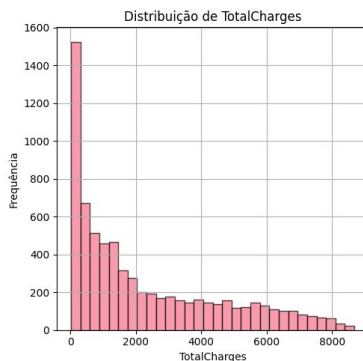
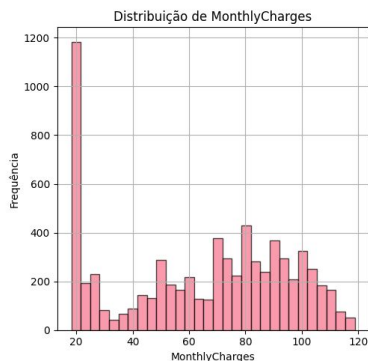
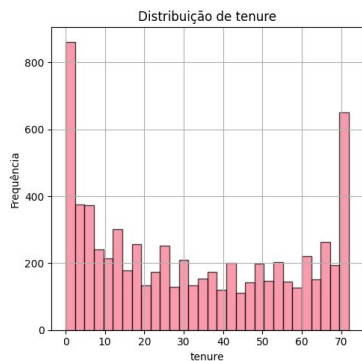
	Missing Count	Percentage
TotalCharges	11	0.156183

```
🔍 Tratando valores ausentes na coluna 'TotalCharges'...  
- Mediana calculada: 1397.47  
- Valores ausentes preenchidos com sucesso!
```

```
🔍 Verificação pós-tratamento (não deve haver mais NaNs):  
gender                0  
SeniorCitizen         0  
Partner               0  
Dependents            0  
tenure                0  
PhoneService          0  
MultipleLines         0  
InternetService       0  
OnlineSecurity        0  
OnlineBackup          0  
DeviceProtection      0  
TechSupport           0  
StreamingTV           0  
StreamingMovies       0  
Contract              0  
PaperlessBilling       0  
PaymentMethod         0  
MonthlyCharges        0  
TotalCharges          0  
Churn                 0  
dtype: int64
```

Pré-processamento

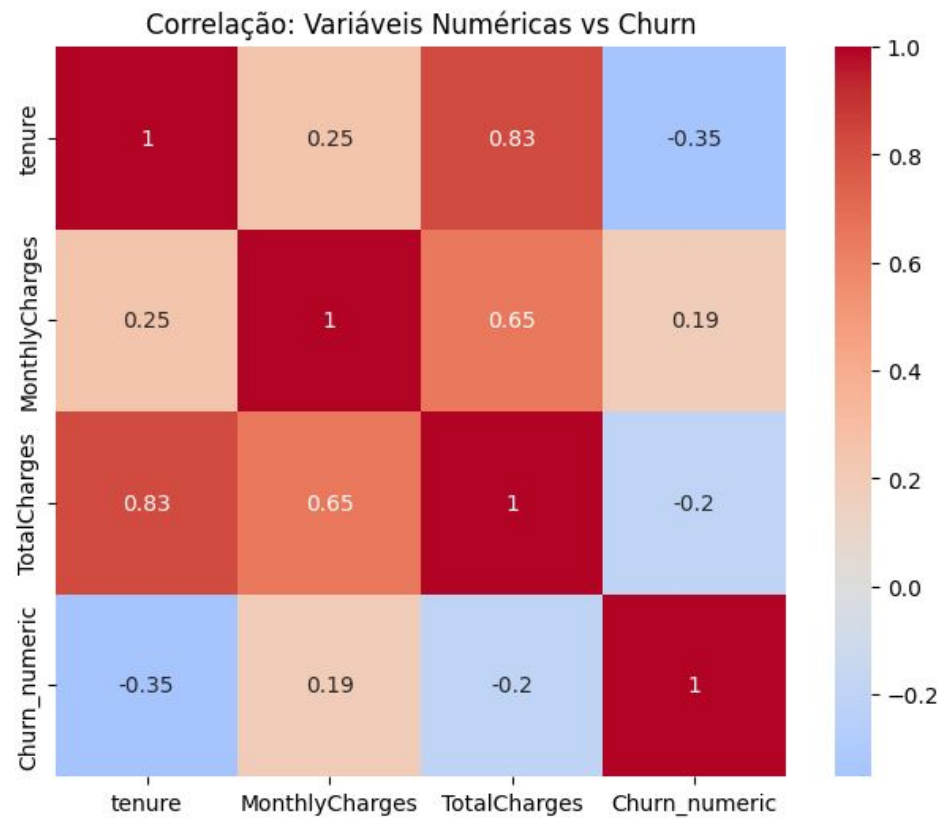
- Codificação de Variáveis
 - LabelEncoder aplicado em 16 variáveis categóricas
 - Conversão da variável alvo 'Churn': No=0, Yes=1
 - Matriz final: 20 features (16 categóricas + 4 numéricas)



```
Codificando variáveis categóricas...  
✓ gender  
✓ Partner  
✓ Dependents  
✓ PhoneService  
✓ MultipleLines  
✓ InternetService  
✓ OnlineSecurity  
✓ OnlineBackup  
✓ DeviceProtection  
✓ TechSupport  
✓ StreamingTV  
✓ StreamingMovies  
✓ Contract  
✓ PaperlessBilling  
✓ PaymentMethod  
✓ Churn (0=No, 1=Yes)  
  
Shape final: X=(7043, 19), y=(7043,)
```

Pré-processamento

- Correlação
 - Tenure (Ocupação)
 - MonthlyCharges
 - TotalCharges
 - Churn_numeric



Pré-processamento

- Divisão Estratificada dos Dados
 - Treinamento: 4.224 amostras (60%)
 - Validação: 1.408 amostras (20%)
 - Teste: 1.411 amostras (20%)
 - Distribuição de classes preservada em todos os conjuntos

```
Dividindo os dados...
Treinamento: 4,225 amostras (60.0%)
Validação: 1,409 amostras (20.0%)
Teste: 1,409 amostras (20.0%)

Distribuição de classes:
Treinamento - Classe 0: 3,104 (73.5%) | Classe 1: 1,121 (26.5%)
Validação - Classe 0: 1,035 (73.5%) | Classe 1: 374 (26.5%)
Teste - Classe 0: 1,035 (73.5%) | Classe 1: 374 (26.5%)
```

Pré-processamento

- Normalização e Balanceamento
 - StandardScaler aplicado apenas nos dados numéricos
 - Oversampling básico para balancear classes de treino
 - Dados balanceados: 5.161 Classe 0 e 5.161 Classe 1

```
# Normalização usando StandardScaler
print("⚙️ Normalizando dados...")

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)

print("✅ Normalização concluída")
print("⚠️ Scaler ajustado apenas no conjunto de treinamento")

# Verificando normalização
print(f"📊 Estatísticas pós-normalização (treinamento):")
print(f" Média (primeiras 5 features): {np.mean(X_train_scaled, axis=0)[:5]}")
print(f" Desvio padrão (primeiras 5): {np.std(X_train_scaled, axis=0)[:5]}")
```

⚙️ Normalizando dados...

✅ Normalização concluída

⚠️ Scaler ajustado apenas no conjunto de treinamento

📊 Estatísticas pós-normalização (treinamento):

Média (primeiras 5 features): [-5.71797705e-17 9.92237193e-17 -6.72703182e-18 1.01746356e-16 -1.34540636e-17]

Desvio padrão (primeiras 5): [1. 1. 1. 1. 1.]

```
📊 Distribuição original (treinamento): Classe 0 = 3,104, Classe 1 = 1,121
📊 Distribuição balanceada: Classe 0 = 3,104, Classe 1 = 3,104
✅ Oversampling aplicado com sucesso!
```

Experimentos

Random Forest

Parâmetros Iniciais

```
rf_model = RandomForestClassifier(  
    n_estimators=100,      # 100 árvores no ensemble  
    max_depth=10,         # Profundidade máxima = 10  
    max_features='sqrt',  #  $\sqrt{\text{features}}$  por split  
    min_samples_leaf=5,   # Mínimo 5 amostras por folha  
    random_state=42,      # Reprodutibilidade  
    n_jobs=-1             # Usar todos os CPUs  
)
```

Random Forest

Treinamento

```
[ ] # Fazendo predições com Random Forest
print("📊 Fazendo predições...")

y_train_pred_rf = rf_model.predict(X_train_scaled)
y_val_pred_rf = rf_model.predict(X_val_scaled)
y_test_pred_rf = rf_model.predict(X_test_scaled)

# Probabilidades para métricas KS e ROC-AUC
y_train_proba_rf = rf_model.predict_proba(X_train_scaled)[: , 1]
y_val_proba_rf = rf_model.predict_proba(X_val_scaled)[: , 1]
y_test_proba_rf = rf_model.predict_proba(X_test_scaled)[: , 1]

print("✅ Predições concluídas!")
```



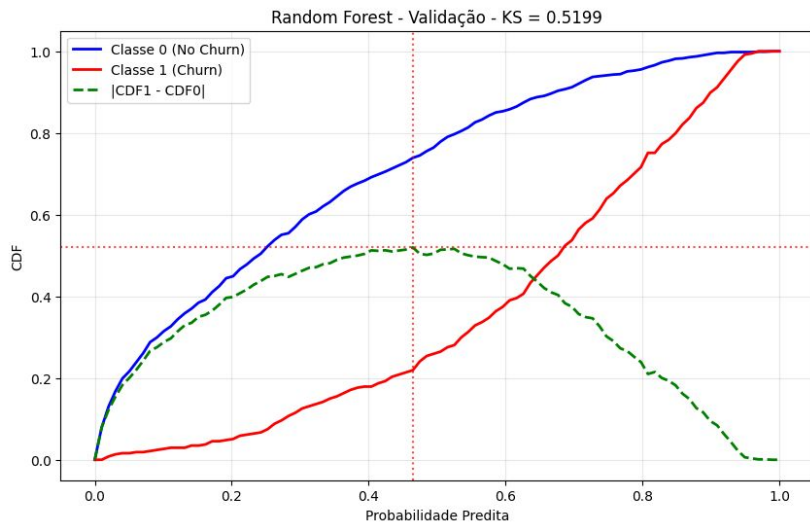
Fazendo predições...



Predições concluídas!

Random Forest

Avaliação



AVALIAÇÃO DO RANDOM FOREST

TREINAMENTO

KS (Principal): 0.7477
ROC-AUC: 0.9438
F1-Score: 0.7669
MSE: 0.1070
Cross-Entropy: 0.3436

Matriz de Confusão:
[[TN: 2572, FP: 532]
[FN: 93, TP: 1028]]

VALIDAÇÃO

KS (Principal): 0.5226
ROC-AUC: 0.8329
F1-Score: 0.6232
MSE: 0.1579
Cross-Entropy: 0.4777

Matriz de Confusão:
[[TN: 797, FP: 238]
[FN: 97, TP: 277]]

TESTE

KS (Principal): 0.5368
ROC-AUC: 0.8373
F1-Score: 0.6290
MSE: 0.1569
Cross-Entropy: 0.4719

Matriz de Confusão:
[[TN: 790, FP: 245]
[FN: 90, TP: 284]]

Random Forest

Otimização

Primeiramente, vamos definir a função que o Optuna vai otimizar usando cross-validation com a métrica KS.

```
def objective(trial):  
    """Função objetivo para Optuna - otimiza parâmetros sugestivos"""  
  
    # Parâmetros sugestivos  
    params = {  
        'n_estimators': trial.suggest_int('n_estimators', 50, 500, step=25),  
        'max_depth': trial.suggest_int('max_depth', 3, 25),  
        'max_features': trial.suggest_categorical('max_features', ['sqrt', 'log2', 0.3, 0.5]),  
        'min_samples_leaf': trial.suggest_int('min_samples_leaf', 1, 20),  
        'random_state': 42,  
        'n_jobs': -1,  
        'class_weight': 'balanced'  
    }
```

🏆 MELHORES PARÂMETROS:
n_estimators: 425
max_depth: 19
max_features: sqrt
min_samples_leaf: 20

Melhor KS Score: 0.5641

Random Forest






Comparações Finais


COMPARAÇÃO BASELINE vs OTIMIZADO:

Métrica	Baseline	Otimizado	Melhoria_Pct
KS	0.5226	0.5326	1.9200
ROC-AUC	0.8329	0.8359	0.3700
F1-Score	0.6232	0.6266	0.5500
MSE	0.1579	0.1627	2.9900
Cross-Entropy	0.4777	0.4896	2.4800






AVALIAÇÃO MODELO OTIMIZADO


TREINAMENTO - OTIMIZADO

 KS (Principal): 0.6319
 ROC-AUC: 0.8992
 F1-Score: 0.6882
 MSE: 0.1363
 Cross-Entropy: 0.4179






 Matriz de Confusão:
 [[TN: 2395, FP: 709]
 [FN: 161, TP: 960]]


VALIDAÇÃO - OTIMIZADO

 KS (Principal): 0.5326
 ROC-AUC: 0.8359
 F1-Score: 0.6266
 MSE: 0.1627
 Cross-Entropy: 0.4896

 Matriz de Confusão:
 [[TN: 769, FP: 266]
 [FN: 82, TP: 292]]

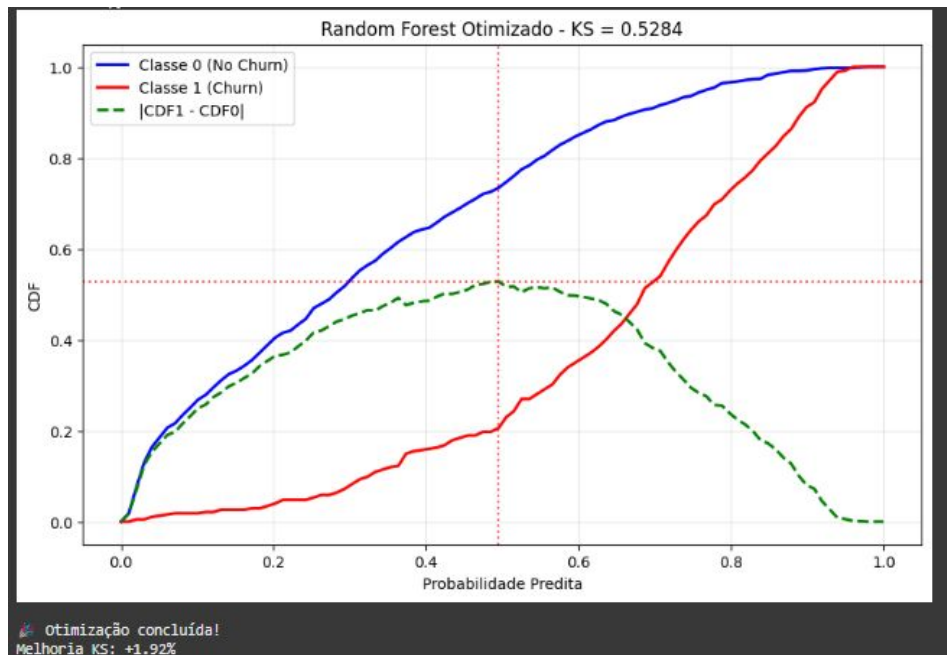
TESTE - OTIMIZADO

 KS (Principal): 0.5362
 ROC-AUC: 0.8429
 F1-Score: 0.6267
 MSE: 0.1600
 Cross-Entropy: 0.4790

 Matriz de Confusão:
 [[TN: 767, FP: 268]
 [FN: 81, TP: 293]]

Random Forest

Comparações Finais



MLP

- Modelo: MLPClassifier
- Tuning: Análise sistemática de hiperparâmetros + Optuna (para otimização avançada)
- Estrutura da rede: [10], [50], [100], [50,25], [100,50], [100,100], [100,50,25], [200,100,50]
- Funções de ativação: Logistic (Sigmoid), ReLU, Tanh
- Taxa de aprendizado: 0.0001, 0.001, 0.01, 0.1 + Estratégias adaptativas (Reduz pela pquando loss para de melhorar)
- Otimizadores: ADAM, SGD, L-BFGS + configurações específicas
- Regularização: 0.00001, 0.0001, 0.001 (L2 penalty)
- Loss: Cross-entropy (log-loss)
- Métricas: KS Score (principal), ROC-AUC, F1-Score, MSE

MLP

Implementação dos Experimentos

```
print("🐛 TESTE 3: Taxa de Aprendizagem")
print("=" * 50)

# Diferentes taxas de aprendizagem
learning_rates = [
    (0.001, "Taxa Baixa (0.001)"),
    (0.01, "Taxa Média (0.01)"),
    (0.1, "Taxa Alta (0.1)"),
    (0.0001, "Taxa Muito Baixa (0.0001)")
]

learning_rate_results = {}

for lr, name in learning_rates:
    results, model = test_mlp_configuration(
        hidden_layers=best_architecture,
        learning_rate=lr,
        config_name=f"LR {name}"
    )
```

```
print("🐛 TESTE 2: Funções de Ativação")
print("=" * 50)

# Funções de ativação para testar (conforme especificado no projeto)
activation_functions = [
    ('logistic', 'Logística (Sigmoid)'),
    ('tanh', 'Tangente Hiperbólica'),
    ('relu', 'ReLU (Rectified Linear Unit)')
]

# Usar a melhor arquitetura encontrada anteriormente (ou uma padrão)
best_architecture = (100, 50) # 2 camadas como padrão

activation_results = {}

for activation, name in activation_functions:
    results, model = test_mlp_configuration(
        hidden_layers=best_architecture,
        activation=activation,
        config_name=f"Ativação {name}"
    )
```

MLP

Implementação dos Experimentos

TESTE 1: Número de Camadas e Neurônios	TESTE 2: Funções de Ativação	TESTE 3: Taxa de Aprendizagem
<p>Testando 1 camada – 10 neurônios:</p> <p>Camadas: (10,)</p> <p>Ativação: relu</p> <p>Taxa aprendizado: 0.001</p> <p>Regularização (alpha): 0.0001</p> <p>Otimizador: adam</p> <p>Dados: Balanceado</p>	<p>Testando Ativação Logística (Sigmoide):</p> <p>Camadas: (100, 50)</p> <p>Ativação: logistic</p> <p>Taxa aprendizado: 0.001</p> <p>Regularização (alpha): 0.0001</p> <p>Otimizador: adam</p> <p>Dados: Balanceado</p>	<p>Testando LR Taxa Baixa (0.001):</p> <p>Camadas: (100, 50)</p> <p>Ativação: relu</p> <p>Taxa aprendizado: 0.001</p> <p>Regularização (alpha): 0.0001</p> <p>Otimizador: adam</p> <p>Dados: Balanceado</p>
<p>KS (Principal): 0.5072</p> <p>ROC-AUC: 0.8266</p> <p>F1-Score: 0.6070</p> <p>MSE: 0.1714</p> <p>Cross-Entropy: 0.5103</p> <p>Matriz de Confusão:</p> <p>[[TN: 748, FP: 287]</p> <p>[FN: 86, TP: 288]]</p> <p>Convergiu em 89 iterações</p> <p>KS: 0.5072 ROC-AUC: 0.8266 F1: 0.6070</p> <p>...</p> <p>Convergiu em 84 iterações</p> <p>KS: 0.4217 ROC-AUC: 0.7685 F1: 0.5388</p>	<p>KS (Principal): 0.5156</p> <p>ROC-AUC: 0.8311</p> <p>F1-Score: 0.6181</p> <p>MSE: 0.1742</p> <p>Cross-Entropy: 0.5158</p> <p>Matriz de Confusão:</p> <p>[[TN: 745, FP: 290]</p> <p>[FN: 77, TP: 297]]</p> <p>Convergiu em 45 iterações</p> <p>KS: 0.5156 ROC-AUC: 0.8311 F1: 0.6181</p> <p>...</p> <p>Convergiu em 156 iterações</p> <p>KS: 0.4208 ROC-AUC: 0.7769 F1: 0.5545</p>	<p>KS (Principal): 0.4208</p> <p>ROC-AUC: 0.7769</p> <p>F1-Score: 0.5545</p> <p>MSE: 0.2142</p> <p>Cross-Entropy: 0.8143</p> <p>Matriz de Confusão:</p> <p>[[TN: 786, FP: 249]</p> <p>[FN: 135, TP: 239]]</p> <p>Convergiu em 156 iterações</p> <p>KS: 0.4208 ROC-AUC: 0.7769 F1: 0.5545</p> <p>...</p> <p>Convergiu em 87 iterações</p> <p>KS: 0.5241 ROC-AUC: 0.8314 F1: 0.6184</p>

MLP

Otimizações

Espaço de busca

```
n_layers = trial.suggest_int("n_layers", 1, 2)
units1 = trial.suggest_categorical("units1", [10, 32, 64, 128])
units2 = trial.suggest_categorical("units2", [16, 32, 64, 128]) if n_layers == 2 else 0
activation = trial.suggest_categorical("activation", ["relu", "tanh", "sigmoid"])
opt_name = trial.suggest_categorical("optimizer", ["adam", "rmsprop", "adadelata", "sgd"])
lr = trial.suggest_float("learning_rate", 1e-4, 1e-1, log=True)
dropout = trial.suggest_float("dropout", 0.0, 0.5)
l2_alpha = trial.suggest_float("l2_alpha", 1e-6, 1e-2, log=True)
batch_size = trial.suggest_categorical("batch_size", [32, 64, 128])
patience = trial.suggest_categorical("patience", [10, 20])
```






Espaço de Busca


Melhores parametros

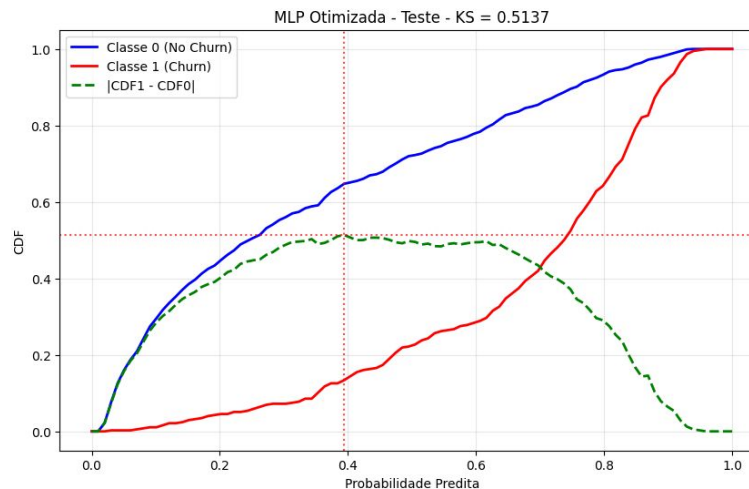
n_layers	1
units1	10
activation	"tanh"
optimizer	"adadelata"
learning_rate	0.014737337784272326
dropout	0.023095576748070612
l2_alpha	0.0001714155043478194
batch_size	32
patience	20

MLP Otimizações






TESTE - MLP OTIMIZADA


 KS (Principal): 0.5146
 ROC-AUC: 0.8283
 F1-Score: 0.6084
 MSE: 0.1732
 Cross-Entropy: 0.5131

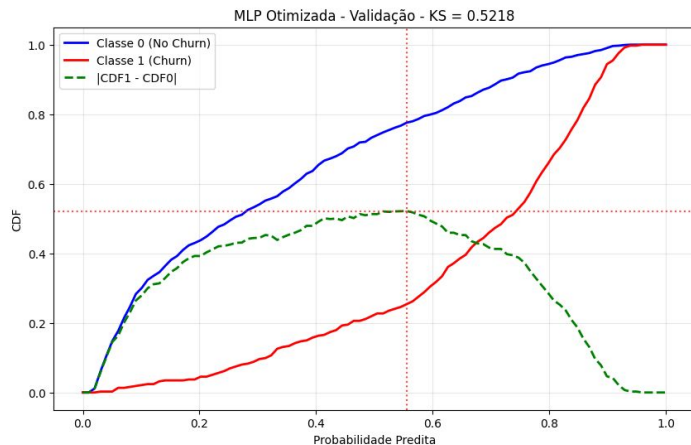
 Matriz de Confusão:
 [[TN: 748, FP: 287]
 [FN: 85, TP: 289]]



VALIDAÇÃO - MLP OTIMIZADA

 KS (Principal): 0.5238
 ROC-AUC: 0.8280
 F1-Score: 0.6206
 MSE: 0.1698
 Cross-Entropy: 0.5052

 Matriz de Confusão:
 [[TN: 760, FP: 275]
 [FN: 82, TP: 292]]



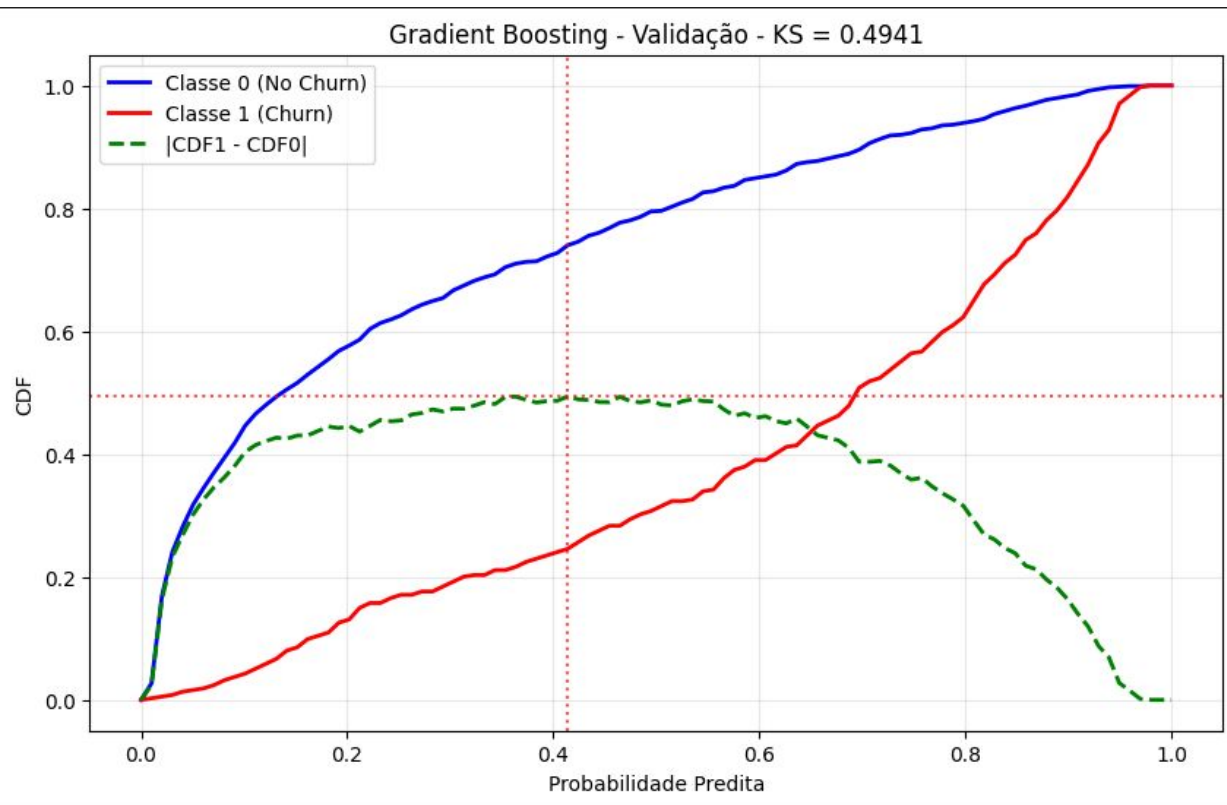
Gradient Boost

Experimentação inicial

```
gb_model = GradientBoostingClassifier(  
    n_estimators=100,          # Número de árvores  
    learning_rate=0.1,        # Taxa de aprendizado  
    max_depth=6,              # Profundidade máxima das árvores  
    min_samples_split=20,     # Mínimo de amostras para dividir um nó  
    min_samples_leaf=5,       # Mínimo de amostras em uma folha  
    subsample=0.8,            # Fração de amostras para cada árvore  
    max_features='sqrt',      # Número de features consideradas  
    random_state=42,  
    verbose=0  
)  
  
print("Treinando Gradient Boosting...")  
gb_model.fit(X_train_balanced, y_train_balanced)
```

```
===== AVALIAÇÃO DO GRADIENT BOOSTING =====  
  
TREINAMENTO  
-----  
KS (Principal): 0.7953  
ROC-AUC: 0.9573  
F1-Score: 0.8060  
MSE: 0.0889  
Cross-Entropy: 0.2939  
  
Matriz de Confusão:  
[[TN: 2680, FP: 424]  
 [FN: 78, TP: 1043]]  
  
VALIDAÇÃO  
-----  
KS (Principal): 0.4955  
ROC-AUC: 0.8280  
F1-Score: 0.6105  
MSE: 0.1582  
Cross-Entropy: 0.4751  
  
Matriz de Confusão:  
[[TN: 824, FP: 211]  
 ...  
  
Matriz de Confusão:  
[[TN: 816, FP: 219]  
 [FN: 112, TP: 262]]
```


Gradient Boost



Gradient Boosting

Variação dos parâmetros

- Loss: deviance
- Learning rate
- Número de estimadores
- Subsample
- Criterion: friedman_mse
- Min_samples_leaf
- Max depth

```
learning_rate = trial.suggest_float('learning_rate', 0.01, 0.3)
n_estimators = trial.suggest_int('n_estimators', 50, 300)
subsample = trial.suggest_float('subsample', 0.5, 1.0)
min_samples_leaf = trial.suggest_int('min_samples_leaf', 1, 20)
max_depth = trial.suggest_int('max_depth', 2, 10)
criterion = trial.suggest_categorical('criterion', ['friedman_mse'])
loss = trial.suggest_categorical('loss', ['log_loss'])

# Modelo
model = GradientBoostingClassifier(
    learning_rate=learning_rate,
    n_estimators=n_estimators,
    subsample=subsample,
    min_samples_leaf=min_samples_leaf,
    max_depth=max_depth,
    criterion=criterion,
    loss=loss,
    random_state=42
)
```

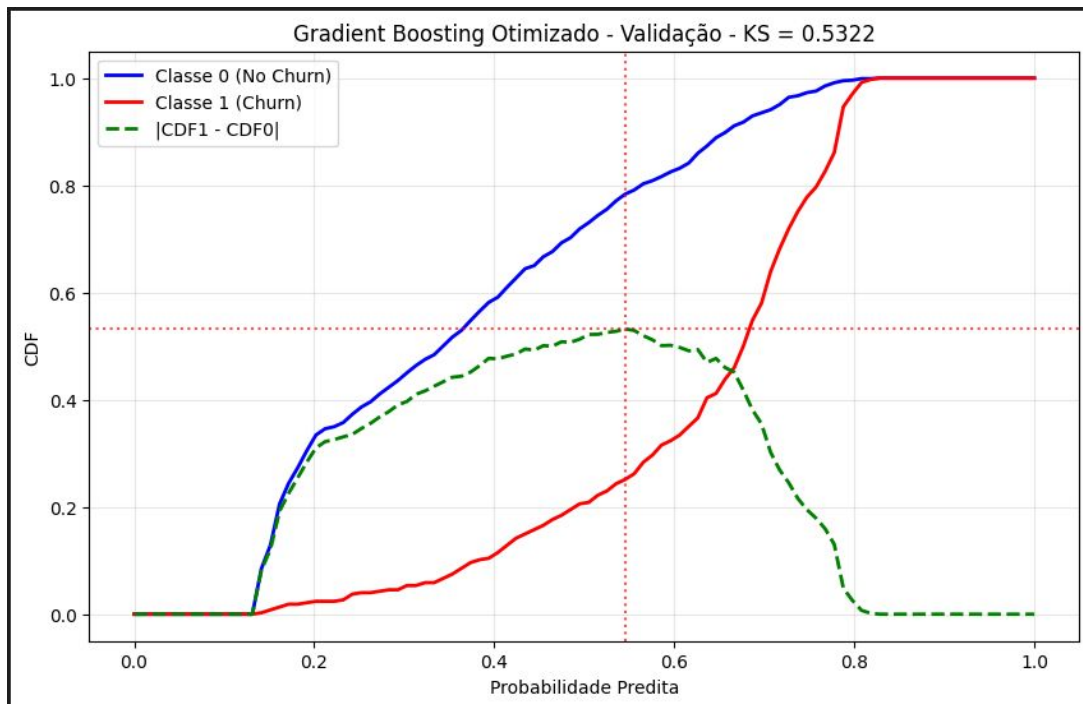
Melhores hiperparâmetros encontrados:

{'learning_rate': 0.013078175960039679, 'n_estimators': 111, 'subsample': 0.5013015627577913, 'min_samples_leaf': 13, 'max_depth': 4, 'criterion': 'friedman_mse', 'loss': 'log_loss'}

Melhor KS (validação): 0.5393

Gradient Boosting

Variação dos parâmetros - resultados



VALIDAÇÃO

KS (Principal): 0.5393
ROC-AUC: 0.8385
F1-Score: 0.6220
MSE: 0.1709
Cross-Entropy: 0.5164

Matriz de Confusão:
[[TN: 751, FP: 284]
[FN: 77, TP: 297]]

TREINAMENTO

KS (Principal): 0.5723
ROC-AUC: 0.8706
F1-Score: 0.6486
MSE: 0.1599
Cross-Entropy: 0.4910

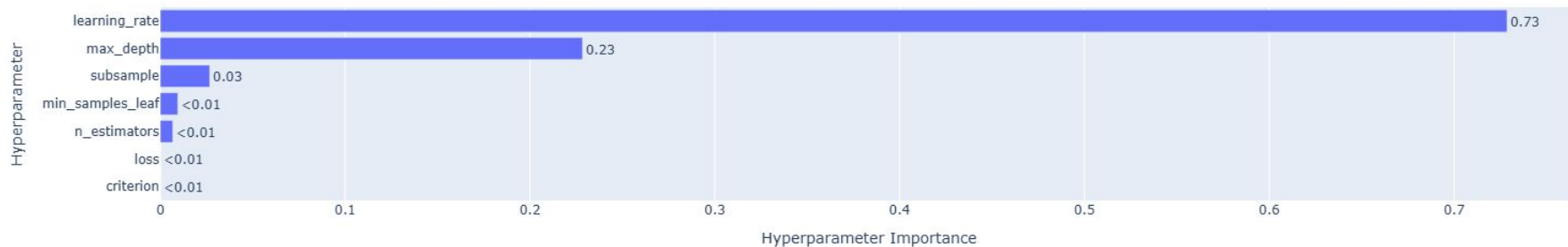
...

[[TN: 755, FP: 280]
[FN: 77, TP: 297]]

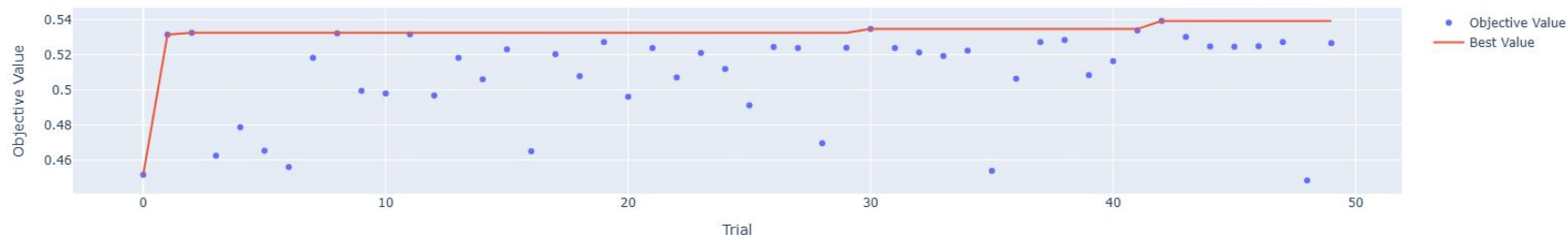
Gradient Boost

Variação dos parâmetros - resultados

Hyperparameter Importances



Optimization History Plot



XGBoost

Variação dos parâmetros

```
# Hiperparâmetros a serem otimizados
learning_rate = trial.suggest_float('learning_rate', 0.01, 0.3)
n_estimators = trial.suggest_int('n_estimators', 50, 300)
subsample = trial.suggest_float('subsample', 0.5, 1.0)
min_child_weight = trial.suggest_int('min_child_weight', 1, 20)
max_depth = trial.suggest_int('max_depth', 2, 10)
colsample_bytree = trial.suggest_float('colsample_bytree', 0.5, 1.0)
gamma = trial.suggest_float('gamma', 0, 5)
reg_alpha = trial.suggest_float('reg_alpha', 0, 2)
reg_lambda = trial.suggest_float('reg_lambda', 0, 2)

# Modelo
model = XGBClassifier(
    learning_rate=learning_rate,
    n_estimators=n_estimators,
    subsample=subsample,
    min_child_weight=min_child_weight,
    max_depth=max_depth,
    colsample_bytree=colsample_bytree,
    gamma=gamma,
    reg_alpha=reg_alpha,
    reg_lambda=reg_lambda,
    use_label_encoder=False,
    eval_metric='logloss',
    random_state=42
)
```


XGBoost

Variação dos parâmetros - resultados


Melhores hiperparâmetros encontrados para XGBoost:






```
{'learning_rate': 0.058527884594206596, 'n_estimators': 81, 'subsample': 0.635738428162489, 'min_child_weight': 16, 'max_depth': 8, 'colsample_bytree': 0.6913496408383459, 'gamma': 1.2203162237179979, 'reg_alpha': 1.8072366886606457, 'reg_lambda': 1.634653986989167}
```


Melhor KS (validação): 0.5444







 AVALIAÇÃO DO XGBOOST OTIMIZADO

=====

 TREINAMENTO

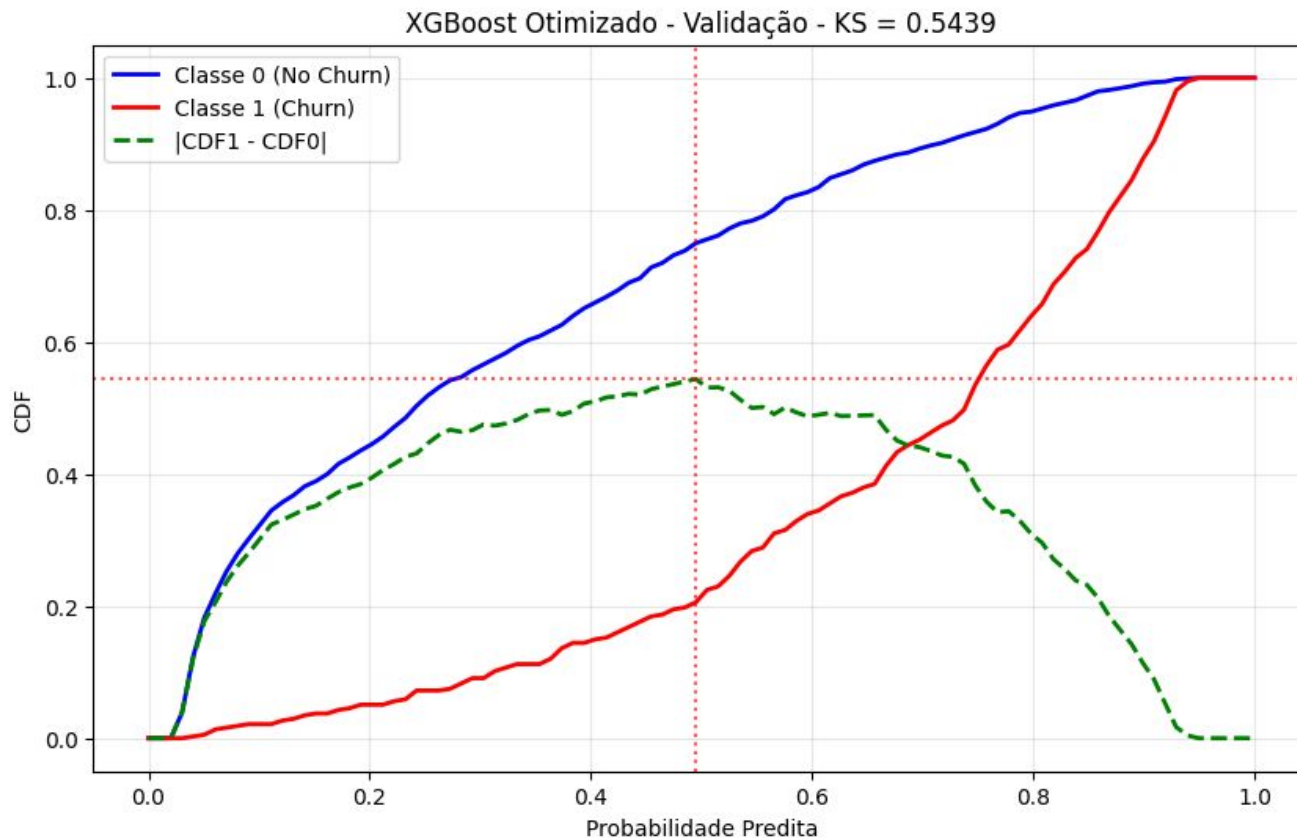
-  KS (Principal): 0.6245
-  ROC-AUC: 0.8905
-  F1-Score: 0.6857
-  MSE: 0.1388
-  Cross-Entropy: 0.4253
- ...
- [[TN: 773, FP: 262]
[FN: 82, TP: 292]]

 VALIDAÇÃO

-  KS (Principal): 0.5444
-  ROC-AUC: 0.8412
-  F1-Score: 0.6371
-  MSE: 0.1615
-  Cross-Entropy: 0.4842
-  Matriz de Confusão:
[[TN: 778, FP: 257]
[FN: 79, TP: 295]]

XGBoost

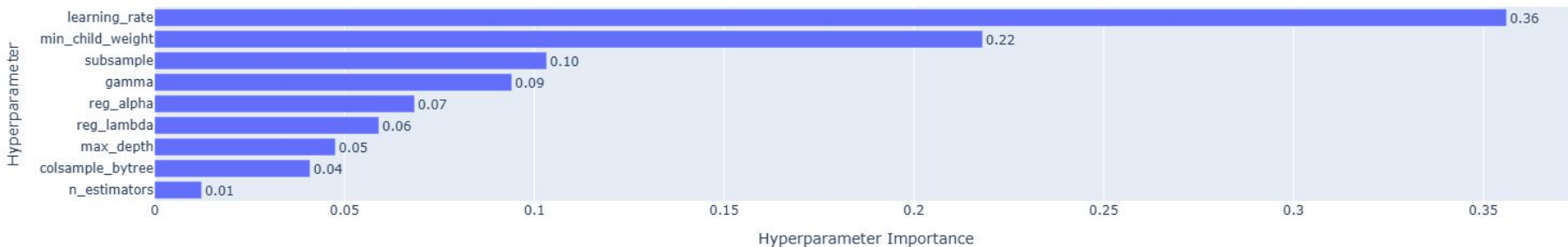
Variação dos parâmetros - resultados



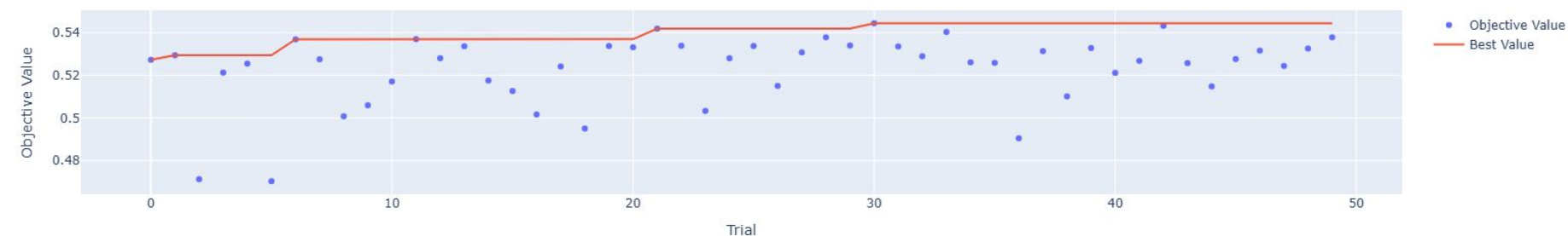
XGBoost

Variação dos parâmetros - resultados

Hyperparameter Importances



Optimization History Plot



TabPFN

- Configuração do Modelo:
 - Modelo: TabPFN (Tabular Prior-Fitted Network)
 - Tuning: Não aplicável (modelo pré-treinado)
 - Estrutura: Transformer pré-treinado para dados tabulares
 - Device: CUDA (GPU acelerada)
 - N_ensemble_configurations: Padrão (32)
 - Preprocessing: Dados originais

TabPFN

- Características Técnicas
 - Prior Knowledge: Aprende padrões de múltiplos datasets durante pré-treinamento
 - Few-shot Learning: Excelente performance com poucos dados
 - Auto-balancing: Lida internamente com desbalanceamento de classes
 - No hyperparameter tuning: Modelo zero-shot para dados tabulares
 - Inference: Rápida predição sem necessidade de treino adicional

TabPFN

- Implementação dos Experimento:

```
print("🤖 Implementando TabPFN...")

tabpfn_model = TabPFNClassifier(device='cuda')

tabpfn_model.fit(X_train, y_train)

print("✅ TabPFN treinado com sucesso!")
```

```
🤖 Implementando TabPFN...
Treinando TabPFN...
✅ TabPFN treinado com sucesso!
```

```
print("📊 Fazendo predições com TabPFN...")

y_train_pred_tabpfn = tabpfn_model.predict(X_train)
y_val_pred_tabpfn = tabpfn_model.predict(X_val)
y_test_pred_tabpfn = tabpfn_model.predict(X_test)

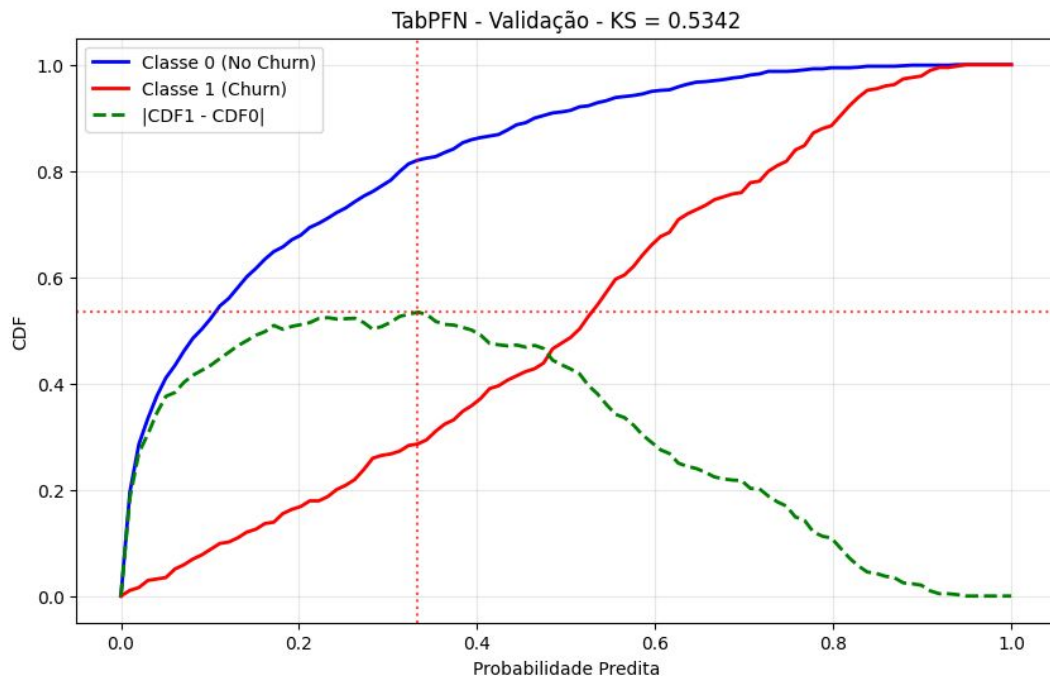
y_train_proba_tabpfn = tabpfn_model.predict_proba(X_train)[: , 1]
y_val_proba_tabpfn = tabpfn_model.predict_proba(X_val)[: , 1]
y_test_proba_tabpfn = tabpfn_model.predict_proba(X_test)[: , 1]

print("✅ Predições concluídas!")
```

```
📊 Fazendo predições com TabPFN...
✅ Predições concluídas!
```

TabFPN

Avaliação de Performance



TREINAMENTO

KS (Principal): 0.5774
ROC-AUC: 0.8667
F1-Score: 0.6161
MSE: 0.1261
Cross-Entropy: 0.3897

Matriz de Confusão:
[[TN: 2830, FP: 274]
[FN: 500, TP: 621]]

VALIDAÇÃO

KS (Principal): 0.5361
ROC-AUC: 0.8404
F1-Score: 0.5918
MSE: 0.1362
Cross-Entropy: 0.4253

Matriz de Confusão:
[[TN: 945, FP: 90]
[FN: 179, TP: 195]]

TESTE

KS (Principal): 0.5365
ROC-AUC: 0.8514
F1-Score: 0.5967
MSE: 0.1336

STab

- Modelo: STab (*Self-supervised Transformer for Tabular Data*)
- Tuning: Optuna (otimização de hiperparâmetros com *MedianPruner* para early stopping)
- Estrutura: Transformer customizado para dados tabulares (múltiplas camadas de atenção + feed-forward)
- Device: CUDA (GPU acelerada)
- Preprocessing: Conversão para tensores float32, normalização e envio para GPU
- Avaliação: Métrica principal = KS; secundárias = ROC-AUC, F1-Score, MSE, Cross-Entropy

STab

Variação dos parâmetros

n_layers– Número de camadas Transformer

n_heads– Número de cabeças de atenção

d_model– Dimensão dos embeddings internos

dropout– Taxa de *dropout* para regularização

learning_rate– Taxa de aprendizagem

batch_size– Tamanho do lote no treino

activation – Função de ativação nas camadas
feedforward

weight_decay– Regularização L2

max_epochs– Número máximo de épocas (com
possibilidade de *early stopping*)

```
VALIDAÇÃO
-----
KS (Principal): 0.5312
ROC-AUC: 0.8368
F1-Score: 0.6262
MSE: 0.1403
Cross-Entropy: 0.4322

Matriz de Confusão:
[[TN: 889, FP: 146]
 [FN: 137, TP: 237]]

...
Train on 4225 samples, validate on 1409 samples:
▶ Época 1/6...
Epoch 1/6
```

Comparando KS dos melhores modelos

MLP: 0.5238

Random Forest: 0.5362

Stab: 0.5312

TabFN: 0.5342

Gradient Boosting: 0.5322

XgBoost: 0.5439