Mapeamento de Churn através de Machine Learning

Alunos: Eleonora Avello, Julio Cesar Fagundes e Mateus Balotin
eleonora.avello@ufpr.br, julio.fagundes@ufpr.br e mateusbalotin@ufpr.br

Professores: Eduardo Vargas e Lucas Pedroso

2º Semestre de 2022

1 Introdução

O Churn nada mais é do que uma métrica que representa a perda de clientes por uma empresa, ou seja, a porcentagem de clientes que deixam de usar os produtos ou serviços de uma empresa durante um determinado período de tempo. Para uma empresa, ter conhecimento sobre Churn é de suma importância, pois a perda de clientes tem um impacto significativo nos seus lucros e crescimento. Além disso, segundo [1, p. 619] "o custo de atrair novos consumidores é cinco vezes maior que o custo de manutenção do atual consumidor", portanto a redução do Churn pode ter um impacto positivo significativo nas finanças da empresa.

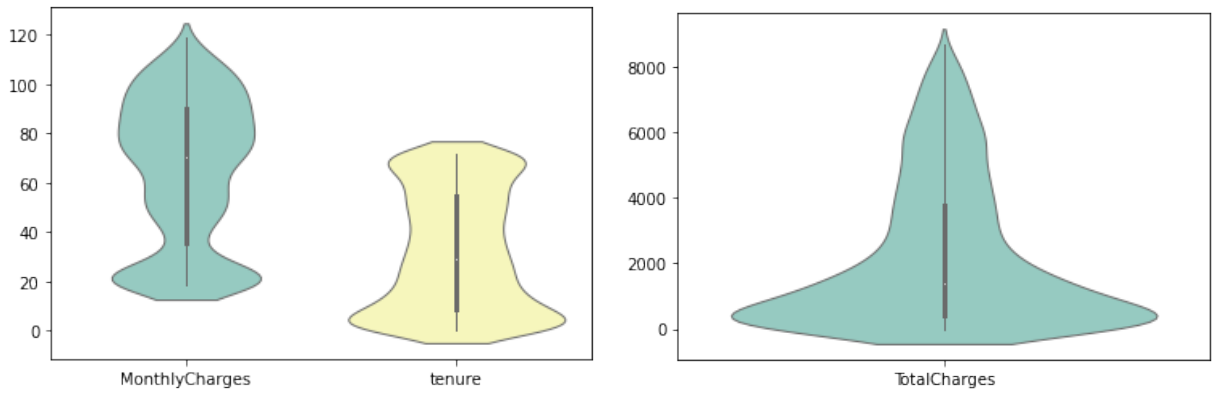
Neste projeto, vamos explorar como o Machine Learning pode ser usado para prever quais clientes irão dar ou não Churn. Vamos discutir diferentes algoritmos que podem ser implementados, bem como suas ferramentas de pré-processamento e tratamento de dados.

2 A Base de Dados

O dataset escolhido foi o Telco Customer Churn, disponível no site do kaggle. Essa base de dados tem 4 categorias gerais.

- Churn: Clientes que deixaram de usar o serviço no último mês.
- Serviços que os clientes usam: celular, linhas múltiplas, internet, segurança online, backup online, proteção dos dispositivos, suporte técnico, streaming de TV e filmes.
- Informações dos clientes: há quanto tempo os clientes utilizam o serviço, contrato, método de pagamento, faturamento, cobrança mensal, cobrança total.
- Informações demográficas sobre os consumidores: Gênero, idade e se tem parceiro ou dependentes.

A seguir, na Figura 1, podemos ver a distribuição dos dados do dataset:



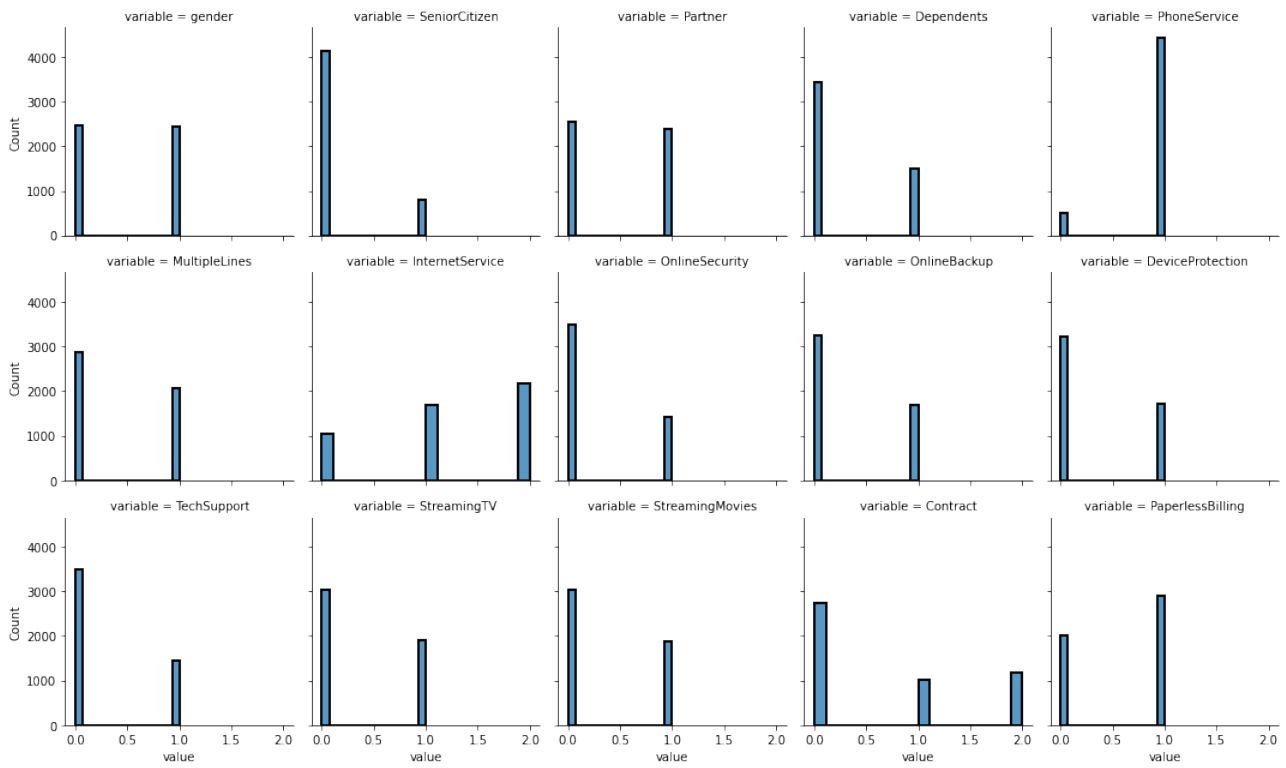


Figura 1: Os graficos de densidade são relativos às variáveis Monthly Charges, Tenure e TotalCharges (float) e o gráfico em barras é relativo às variáveis de classificação. Fonte: Acervo próprio.

2.1 Divisão da Base

Para fazer o treinamento do modelo resolvemos testar dois métodos para divisão de dados: o hold-out e k-fold. Para o hold-out, dividimos a base em 80% para dados de treinamento e 20% para validação. Para o k-fold, decidimos fazer cinco splits que serão utilizados alternadamente de maneira que quatro farão parte do treinamento e um da validação. Após a separação dos dados entre treino e validação, foram criados, tanto para o treino quanto para a validação, os conjuntos X e Y, sendo X correspondente ao conjunto de dados do cliente e Y a variável resposta.

2.2 Dados de Treino

O pré-processamento de dados é uma parte essencial no pipeline de qualquer modelo de Machine Learning. É nesse passo que conseguimos melhorar a acurácia, confiabilidade, consistência e legibilidade do modelo.

A primeira análise que fizemos foi entender quais os tipos das nossas variáves (float, object, int) e observamos que a maioria delas está como object. Em especial a variável Total Charges, que corresponde ao valor total pago pelo cliente até o momento.

Por ser uma variável de valor monetário, não é interessante que a tipagem seja objeto, pois dificulta plots de gráficos e operações matemáticas. Portanto, o primeiro

passo, neste caso, é realizar esta conversão.

Além disso, neste tipo de problema é interessante analisar a necessidade da coluna de Id dos clientes. Optamos por retirar esta coluna para evitar problemas com vazamento de respostas.

Posteriormente, tratamos os dados de classificação, transformando-os em dados numéricos, possibilitando a aplicação de modelos de Machine Learning que necessitam de entradas numéricas.

Por último, utilizamos a regularização lasso para realizar a filtragem das variáveis importantes para a nossa resposta. Para tunar o parâmetro alpha, fizemos um grid search e testamos os resultados para dois valores.

2.2.1 Balanceamento de dados

Um dataset desbalanceado pode levar o modelo a ser enviesado à classe majoritária, resultando em uma piora na performance na hora de aplicar os modelos. Ao balancear o dataset, o modelo tem uma representação igual de ambas as classes, resultando em um performance melhor.

Como no nosso caso temos um desbalanceamento de clientes que não deram churn para os que deram churn, utilizamos a técnica de oversampling, que consiste em duplicar elementos da classe menor até que a mesma fique do mesmo tamanho da classe majoritária, para balancear o dataset.

2.2.2 Conversão de Objetos em Dados Numéricos

Ao realizar a conversão da coluna TotalCharges de objeto para númerico, nos deparamos com uma situação bastante interessante. Ao realizar uma análise da feature antes da conversão, a função *info()* do python nos apontou que esta feature não tinha dados faltantes.

Mas, após converter os dados, apareceram seis clientes com dados faltantes nesta feature. O motivo de não constar como dado faltante na tipagem anterior, foi de que ao tipar a coluna como objeto, podemos atribuir strings nos campos e isso inclui strings vazias ou com espaços somente. Então ao converter para dados numéricos, foi possível detectar este problema.

2.2.3 Dados Faltantes

Dados faltantes em um conjunto de dados são valores ausentes devido a erros de entrada, falhas de coleta ou outras razões. Esses dados faltantes podem ter um grande impacto na precisão de um modelo de Machine Learning, portanto, é importante tratar esse tipo de dado antes de usar um modelo. Existem várias maneiras de tratar os dados faltantes, como substituir os valores ausentes por valores médios, medidas de tendência central, valores mais próximos ou até utilizar um modelo para previsão dos mesmos.

Para tratar os dados faltantes da base, usamos uma simples regra de negócios, onde o Total gasto pelo cliente até o momento pode ser descrito pela multiplicação do tempo em que o cliente consome o produto pelo valor do produto em questão. Ou seja, (TotalCharges) = (MonthlyCharges) x (Tenure). Analisando os dados faltantes, notamos que em todos os campos onde Total Charges era faltante, a tenure do cliente também era zero, ou seja, ele não tinha completado um mês de serviço ainda. Assim, entendemos que a primeira fatura ainda não havia sido paga e, portanto, o TotalCharges nesse caso é igual a zero.

2.2.4 Transformação de Dados Categóricos

A transformação de dados categóricos em numéricos é um processo no qual os dados categóricos são convertidos em números para que possam ser usados em análises estatísticas e alguns métodos de Machine Learning que não trabalham muito bem com categorias. Por exemplo, se uma variável categórica tiver três categorias (A, B e C), pode-se atribuir um número a cada - por exemplo, A = 1, B = 2, C = 3.

2.2.5 Normalização dos Dados

A normalização de dados é um processo de transformação que tem como objetivo ajustar os dados para que eles possam ser comparados e analisados de forma mais eficaz. Ela é usada para ajustar os dados a fim de que eles fiquem uniformes, isso ajuda a melhorar a precisão dos resultados, pois não serão influenciados por valores extremos. Para normalizar, podemos utilizar a seguinte fórmula:

$$X_{modificado} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

2.2.6 Matriz de Correlação

A matriz de correlação consiste de uma matriz que mostra a relação entre duas ou mais variáveis. Tal matriz é usada para identificar padrões de relação entre variáveis, bem como para ravaliar a força da relação entre elas e prever o comportamento de uma variável com base no comportamento de outra. Utilizando este método nos dados dos clientes, obtivemos a matriz disposta na Figura 2

Podemos ver aqui que a matriz de correlação vai ao encontro da regra de negócios que determinamos em sessões anteriores. A correlação entre as variáveis Total Charges com Tenure e Total Charges com Monthly Charges, são positivas e relativamente altas, validando a proporcionalidade da regra proposta.

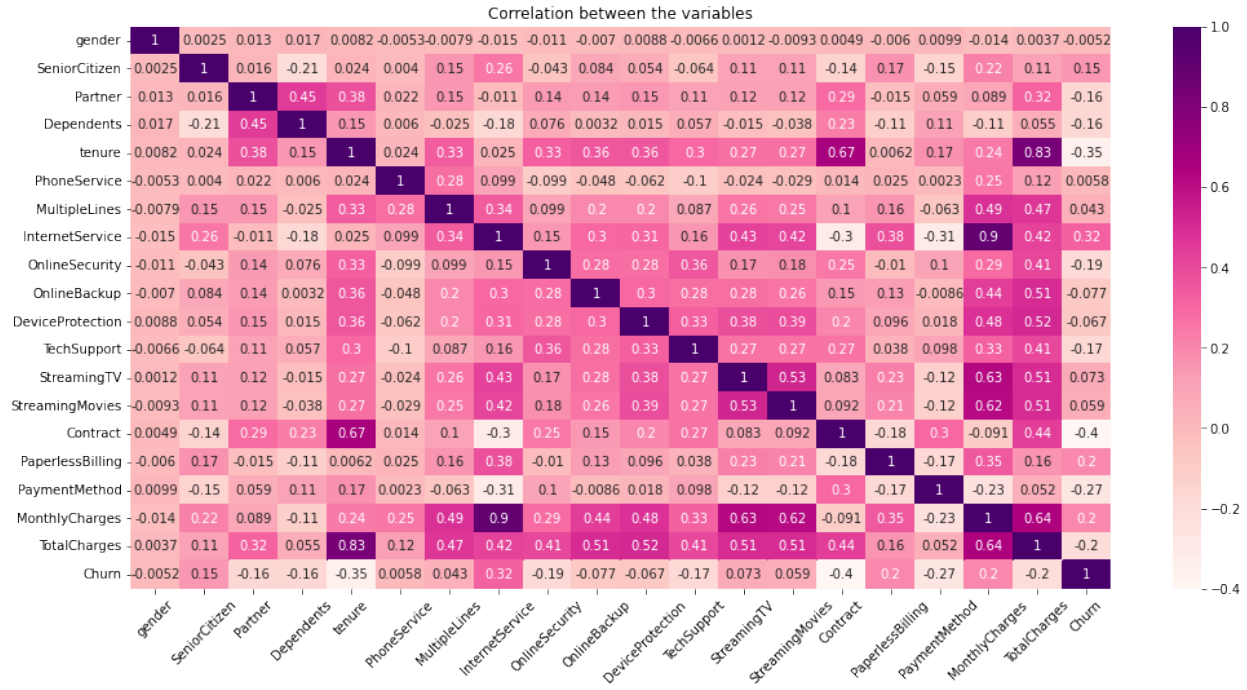


Figura 2: Matriz de Correlação entre as variáveis. Fonte: Acervo próprio.

2.2.7 Filtragem de Variáveis

Entendemos que nem sempre todas as variáveis disponíveis são relevantes para o nosso resultado, podendo aumentar a variância das nossas predições e causando uma 'poluição' no algoritmo. Portanto, para fazer a filtragem das variáveis que não são importantes para a nossa predição, utilizamos a regressão lasso. Na regressão lasso, nós queremos minimizar a seguinte função objetivo:

$$\sum_{i=1}^{n} \left(y_i - \sum_j x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} \mid \beta_j \mid$$

Onde a segunda parte da expressão é o que chamamos de penalização. Quanto maior o valor de retorno, maior a penalidade. Ou seja, quanto maiores os betas encontrados ou quanto maior o parâmetro alpha escolhido, maior a penalização na função.

Sendo assim, quanto maior o parâmetro alpha escolhido, menores os valores de beta retornados pelos algoritmos, consequentemente, teremos menos features em nosso treinamento. Portanto alphas muito baixos não filtram bem as variáveis importantes e alphas muito altos jogam fora variáveis importantes para a nossa resposta. Com isso em vista, realizamos um grid-search para escolher os valores de alpha, como representa a Figura 3.

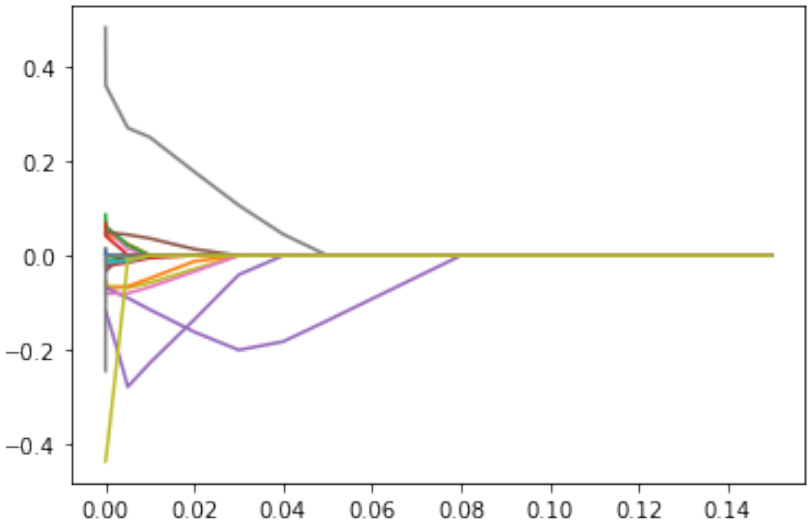


Figura 3: Gridsearch Regressão Lasso. Fonte: Acervo próprio.

3 Treinamento de modelos

Em machine learning existem vários modelos para treinamento de dados, alguns deles muito populares e amplamente utilizados, como Regressão Linear, Árvore de Decisão, Random Forest, Naive Bayes, etc.

A escolha do modelo ideal, contudo, depende da natureza do problema a ser resolvido e dos dados disponíveis. No nosso caso, optamos por treinar os seguintes modelos: KNN (K-Nearest Neighbors), Random Forest e XGBoost.

3.1 Métricas

As métricas de performance são usadas para avaliar o desempenho de um modelo de Machine Learning em relação a um problema específico. Em um problema de aprendizado supervisionado as mais comuns são Acurácia, Precisão, Recall e F1 Score.

Para avaliar os modelos no problema escolhido, Churn, usaremos a métrica Recall. Ela consiste na proporção de previsões corretas positivas em relação ao total de positivos reais e é calculada da seguinte forma

$$Recall = \frac{VP}{VP + FN}.$$

Utilizamos o Recall pois queremos a menor quantidade possível de falsos negativos, ou seja, quem deu **Churn** porém foi previsto como **Não Churn**. Esses clientes seriam simplesmente perdidos pelo erro de predição pois nenhuma estratégia de retenção seria aplicada sobre eles.

3.2 KNN

KNN (K-Nearest Neighbors) procura classificar o valor de uma nova observação de acordo com seus K vizinhos mais próximos. A previsão é baseada na maioria dos valores dos K vizinhos selecionados, sendo K um hiperparâmetro .

Ele é um método muito simples de entender e implementar, porém muito sensível, por isso é sempre importante considerar o escalamento dos dados.

Para escolher o valor de K adequado fizemos um Gridsearch, utilizando $K = 2$ a 30, e escolhendo o K com maior Recall para retreinamento.

A Figura 4 contém o algoritmo para o gridsearch do KNN, onde obtivemos $K = 3$ como resultado.

```
from sklearn.neighbors import KNeighborsClassifier
# definindo a quantidade de vizinhos
neigh = KNeighborsClassifier(n_neighbors=3)

param_grid = {'n_neighbors': np.arange(2,30,1)}
grid_search = GridSearchCV(neigh, param_grid, scoring='recall')
grid_result = grid_search.fit(X_train_norm, Y_treino)

print(f'Best result: {grid_result.best_score_} for {grid_result.best_params_}')
```

```
Best result: 0.8881037139775003 for {'n_neighbors': 3}
```

Figura 4: Gridsearch do KNN. Fonte: Acervo próprio.

A Figura 5 contém as matrizes de confusão do KNN para 2 e 6 variáveis.

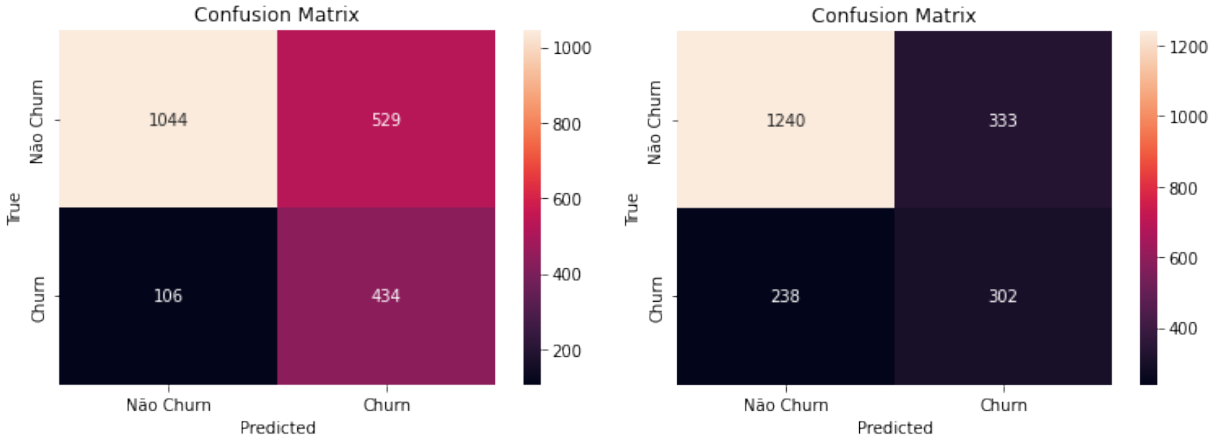


Figura 5: Matriz de Confusão do KNN para 2 variáveis (esquerda) e para 6 variáveis (direita). Fonte: Acervo próprio.

A Figura 6 mostra o recall obtido no KNN para 2 e 6 variáveis.

```
[ ] recall_score(Y_val, predict)

0.8037037037037037
```

```
[ ] recall_score(Y_val, predict)

0.5592592592592592
```

Figura 6: Recall obtido no KNN para 2 variáveis (esquerda) e para 6 variáveis (direita). Fonte: Acervo próprio.

3.3 Random Forest

Random Forest é um método que constrói vários modelos de árvore de decisão e combina seus resultados para melhorar o score e a estabilidade do modelo. Para cada partição, tem-se uma seleção aleatória de m preditores, de um total de p (tipicamente $m \approx \sqrt{p}$). Fazendo isso há uma descorrelação entre as árvores, pois as variáveis utilizadas na construção de cada árvore não são mais as mesmas.

Random Forest é uma técnica poderosa que pode lidar com problemas de classificação e regressão, e também é capaz de lidar com dados com múltiplas classes, características categóricas e dados faltantes.

Utilizamos o Gridsearch para determinar qual é o número ideal de árvores a serem treinadas, variando os estimadores de 500 a 2000, e escolhendo o com maior Recall possível para retreinamento.

A Figura 7 contém o algoritmo para o gridsearch do Random Forest, onde obtivemos $estimadores = 900$ como resultado.

```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier()
param_grid = {'n_estimators': np.arange(500,2000,100)}
grid_search = GridSearchCV(clf, param_grid, scoring='recall')
grid_result = grid_search.fit(X_train_norm, Y_treino)

print(f'Best result: {grid_result.best_score_} for {grid_result.best_params_}')
```

```
Best result: 0.9689120049314225 for {'n_estimators': 900}
```

Figura 7: Gridsearch do Random Forest. Fonte: Acervo próprio.

A Figura 8 contém as matrizes de confusão do Random Forest para 2 e 6 variáveis.

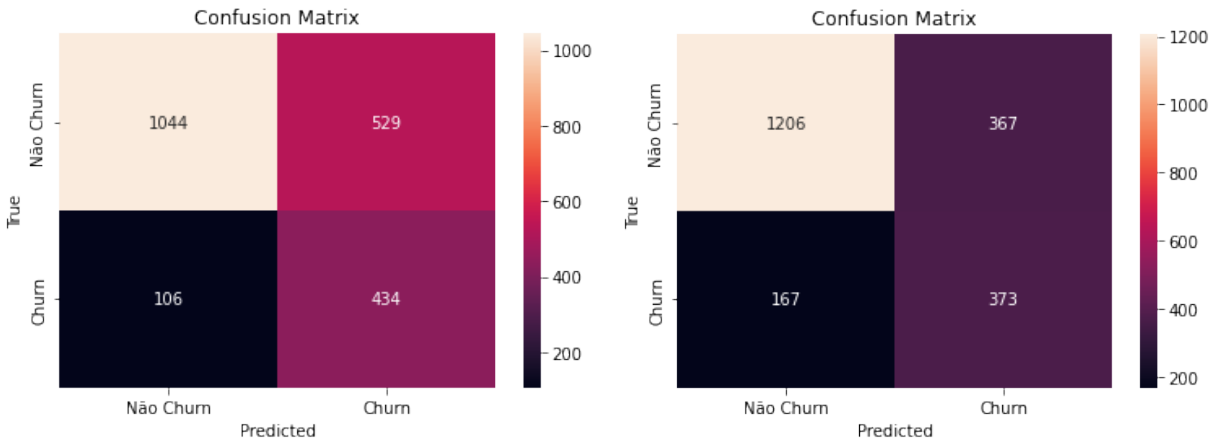


Figura 8: Matriz de Confusão do Random Forest para 2 variáveis (esquerda) e para 6 variáveis (direita). Fonte: Acervo próprio.

A Figura 9 mostra o recall obtido no Random Forest para 2 e 6 variáveis.

```
[ ] recall_score(Y_val, predict)

0.8037037037037037
```

```
[ ] recall_score(Y_val, predict)

0.6907407407407408
```

Figura 9: Recall obtido no Random Forest para 2 variáveis (esquerda) e para 6 variáveis (direita). Fonte: Acervo próprio.

3.4 XGBoost

XGBoost (eXtreme Gradient Boosting) é um método que usa Gradient Boosting para construir modelos de árvore de decisão. É o algoritmo mais popular de Machine Learning atualmente, desde a sua criação tem sido o mais vitorioso no Kaggle.

Uma das características únicas do XGBoost é sua capacidade de lidar com dados faltantes e trabalhar com matrizes de dados esparsas. Além disso, é aderente a uma ampla variedade de aplicações, é compatível com os principais sistemas operacionais e linguagens de programação e oferece suporte para integração com recursos na nuvem e processamento de *big data*.

A Figura 10 contém o gridsearch para os hiperparâmetros learning_rate e gama e a Figura 11 contém o gridsearch para os hiperparâmetros max_depth e min_child_weight.

```
import xgboost as xgb
from sklearn.model_selection import GridSearchCV

# Modelo do XGBoost com as variáveis padrões
model = xgb.XGBClassifier()

# parameter to be searched
param_grid = {'gama': np.arange(0.0,20.0,2),
              'learning_rate': [0.0001, 0.01, 0.1, 1]}

# find the best parameter
grid_search = GridSearchCV(model, param_grid, scoring='recall')
grid_result = grid_search.fit(X_train_norm, Y_treino)

print(f'Best result: {grid_result.best_score_} for {grid_result.best_params_}')
```

```
Best result: 0.8911534905224225 for {'gama': 0.0, 'learning_rate': 1}
```

Figura 10: Gridsearch do XGBoost para os hiperparâmetros learning_rate e gamma. Fonte: Acervo próprio.

```
# Hiperparâmetro max_depth e min_child_weight
param_grid = {'max_depth': range(1,8,1),
              'min_child_weight': np.arange(0.0001, 0.5, 0.01)}

# Achando melhores parâmetros
grid_search = GridSearchCV(model, param_grid, scoring='recall')
grid_result = grid_search.fit(X_train_norm, Y_treino)

print(f'Best result: {grid_result.best_score_} for {grid_result.best_params_}')
```

```
Best result: 0.893081368469718 for {'max_depth': 1, 'min_child_weight': 0.0001}
```

Figura 11: Gridsearch do XGBoost para os hiperparâmetros max_depth e min_child_weight. Fonte: Acervo próprio.

Os hiperparâmetros ótimos obtidos foram:

- learning_rate: 1
- gama: 0
- max_depth: 1
- min_child_weight: 0.0001

A Figura 12 contém as matrizes de confusão do XGBoost para 2 e 6 variáveis.

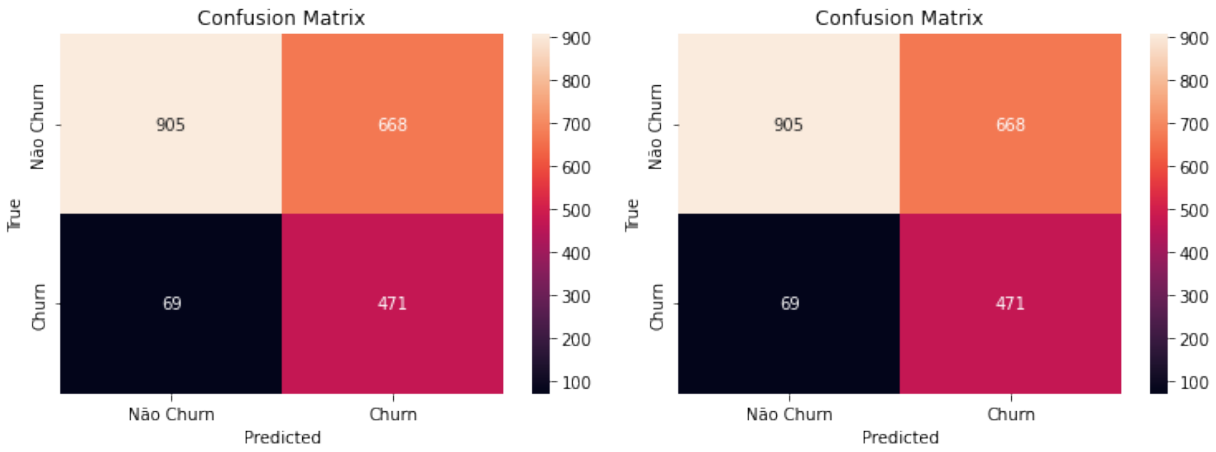


Figura 12: Matriz de Confusão do XGBoost para 2 variáveis (esquerda) e para 6 variáveis (direita). Fonte: Acervo próprio.

A Figura 13 mostra o recall obtido no XGBoost para 2 e 6 variáveis.

```
[ ]  recall_score(Y_val, predict)

0.8722222222222222
```

```
[ ]  recall_score(Y_val, predict)

0.8722222222222222
```

Figura 13: Recall obtido no XGBoost para 2 variáveis (esquerda) e para 6 variáveis (direita). Fonte: Acervo próprio.

4 Conclusão

O problema de churn de clientes é uma preocupação constante das empresas, tendo em vista que o custo para se manter um cliente tende a ser menor que para adquirir clientes novos com estratégias de marketing e advertising.

A engenharia de dados desempenhou um papel fundamental na otimização do desempenho do modelo, conforme observado nos treinamentos realizados. Em particular, mesmo o modelo KNN apresentou bons resultados quando treinado apenas com as duas variáveis, selecionadas pelos nossos métodos.

Além disso, é importante destacar que optamos por utilizar a métrica Recall para avaliar o modelo, uma vez que, neste caso, o que interessa é prever corretamente quais clientes irão dar churn, para que a gerência possa tomar medidas para mantê-los e fidelizá-los.

Portanto, embora algumas métricas tenham performado melhor em outros treinamentos, nossa escolha foi baseada no modelo que retornou o melhor Recall, sendo ele 87.22% para o modelo XGBOOST.

Referências

[1] Kotler P., *Administração de Marketing.* São Paulo: Editora Atlas, 1998.